Technical Report

EE-3501

Embedded Systems

Time and Temperature Display

Author:
Levi Joseph Vande Kerkhoff

Submitted to:
Dr. Jian Zhang

April 6, 2024

Spring Semester, 2024

***Objective*—The goal of this project was to design and implement a fully functional time and temperature display using an F401RE microcontroller board and a 16x2 LCD display. This project affords students an opportunity to learn about general purpose inputs and outputs (GPIO), analog to digital signal conversion, keypad scanning algorithms, and LCD driver implementation.**

I. Program Description

The program functions as a state machine that progresses through states depending upon the user inputs entered via a keypad. The default state displays on the LCD display both the current time and the current temperature with options for Celsius and Fahrenheit. The format is as follows:

HH:MM:SS XX YY T

HH – is the 2-digit representation of the current hour

MM – is the 2-digit representation of the current minute

SS – is the 2-digit representation of the current second

XX – is either AM or PM depending on the current time

YY – is the 2-digit representation of current temperature rounded to the nearest integer

T – is either C or F. These letters are used to represent the unit of temperature in degrees Celsius or Fahrenheit, respectively.

The clock displayed on the LCD display has two modes: normal mode and set mode. Normal mode consists of the previously described time display that updates once per second. Set mode allows the user to enter the time using a 4x4 keypad. The keypad used in the project can be referenced in table 1. Set mode consists of menus that display the unit of time to be entered and the current entry value. The entry menus are in order as follows: hour, minute, and AM/PM. Upon entering a correct time, the seconds are reset to zero. If any of the user entries are incorrect, an error message is displayed and the current entries are reset and appear as blanks. If the user has entered any previous entries, they are retained.

II. Engineering Design

Appendix A, shows the structure of the program. At a high level, the program consists of two sections:

1) An initial set up section wherein GPIO objects, variables, and the RTC are initialized.
2) A “while true” loop that handles keypad scanning, input handling, mode switching, and LCD display updating.

Upon startup, the GPIO pins are initialized and the input modes are selected. This includes inputs for the temperature sensor, and the column scanning. The pins that correspond to the rows of the keypad are also set high. Within this startup procedure, a Real Time Clock (RTC) object is also initialized which sets the time on the RTC hardware to 12:00:00 AM initially.

After startup procedures, the “while true” loop is entered and the program begins in normal mode. As aforementioned, normal mode updates the LCD display once per second; normal mode also checks for key pad entries continuously via the keypadScan() function. A full list of functions called in the program can be seen in Figure 1. Also within the while true loop are multiple sections that check conditional statements based upon user inputs from the keypad. These conditions control whether or not the LCD display should be updated, the current mode of entry, and if the entries are correct upon user submission.

Table 1
Equipment List

Name	Qty.	Manufacturer	Part #
Nucleo-64 Board	1	STMicroelectronics	STM32F401RE
Jumper Wires	21	Elegoo	N/A
LCD Module	1	Tinsharp	TC1602A-09T
10KΩ Pot.	1	Suntan Tech. LTM.	TSR-3386
Centigrade Temp. Sensor	1	Texas Instruments	LM35
USB Cable	1	Molex	N/A
Breadboard Power Supply	1	Elegoo	MB-V2
Digital Multimeter	1	Plusivo	DM101

Three action keys, which are ‘#’, ‘*’, and ‘D’, are present on the keypad and do not act as possible entry values but rather as the name suggests, action inputs. If the ‘*’ key is pressed and the error signal is not being displayed, the program enters the time set mode. From here, all entries besides the action keys update the current entry field. If the user is inside one of the entry menus, the ‘*’ key resets the current entries and takes the user back to the “HOUR” screen, if they are not there already. If the user presses the ‘D’ key, the current entries are discarded and the RTC object remains unaltered. Finally, if the ‘#’ key is pressed, the current entries are evaluated by conditionals. If the entries for the present mode are correct, the entry mode advances to the next. This process repeats until all modes of entry have been completed correctly. Invalid entries send the user to the “error mode” upon the ‘#’ key press. While in error mode, the screen displays the following message: “---- ERROR! ----". This remains on the screen for two seconds; then, the program clears the current invalid entries and the user is returned to the same entry mode. Once all modes of entry are completed correctly, the program updates the RTC object and the program returns to normal mode.

At any time during the program, pressing the button wired to GPIO pin PC_13 on the STM32F401RE will trigger an interrupt handler that will change the current temperature display from C to F, or vice versa. This handler toggles a boolean value found in the getTemp function and in the character array holding the ‘C’ and ‘F’ units.

III. Test Plan

The goal of a test plan is to find bugs in software and should cover all elements of a project. In this project, key hardware components and software were tested together and separately to ensure that all parts work as intended.

Each component used in the project was tested independently from the finalized circuit. In order to confirm proper operation of the temperature sensor within the stated tolerances found within the documentation, a digital multimeter was used to measure the output voltage given a known input voltage. The breadboard power

supply was also tested for proper voltage outputs for each high and low rail on the breadboard before further use. A simple test program was written also to ensure that the LCD screen initialized properly and proper communication between the LCD controller and the STM32 Microcontroller was established. The keypad scan algorithm was developed separately from the rest of the program and each key was tested to ensure that the input was correctly recorded.

Each component of software was tested by the following methodologies. The series of entry modes were each tested by entering invalid parameters to ensure that the error screen was displayed and the entry erased. The error screen was also tested to make sure that the duration that it was shown was indeed 2 seconds. The appropriate time change was tested by setting the clock to 11:59 PM and watching the time shift to 12:00 AM. Likewise, the conversion from 11:59 AM to 12:00 PM was tested. The interrupt for toggling the temperature display was tested while in set mode to ensure that the boolean value is retained.

IV. Conclusion

This program accomplishes all the minimum set requirements set forth by the project description but there are a few key disadvantages to the way in which it accomplishes the goals. One disadvantage would be in the delay mechanism within the key scan algorithm. The use of ThisThread::sleep_for pauses the program and blocks the functionality of other processes. A key improvement would be using non-blocking methods of delaying this key scan. Another improvement would be creating more modularity within the program; certain sections like the LCD screen updating section could be modularized as a function. This would make the code more readable and easier to understand.

Some positive highlights of the program would include the use of the Real-Time Clock to keep track of time accurately. This ensures that the clock maintains the correct time even after power cycles or resets, which is essential for time-sensitive applications. The use of interrupts (InterruptIn for temperature unit toggle) for handling asynchronous events is also an advantage. Interrupts improve responsiveness by allowing the program to react immediately to external stimuli without polling, enhancing real-time behavior.

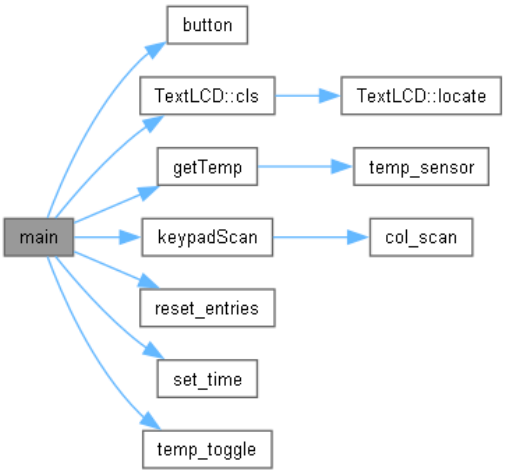
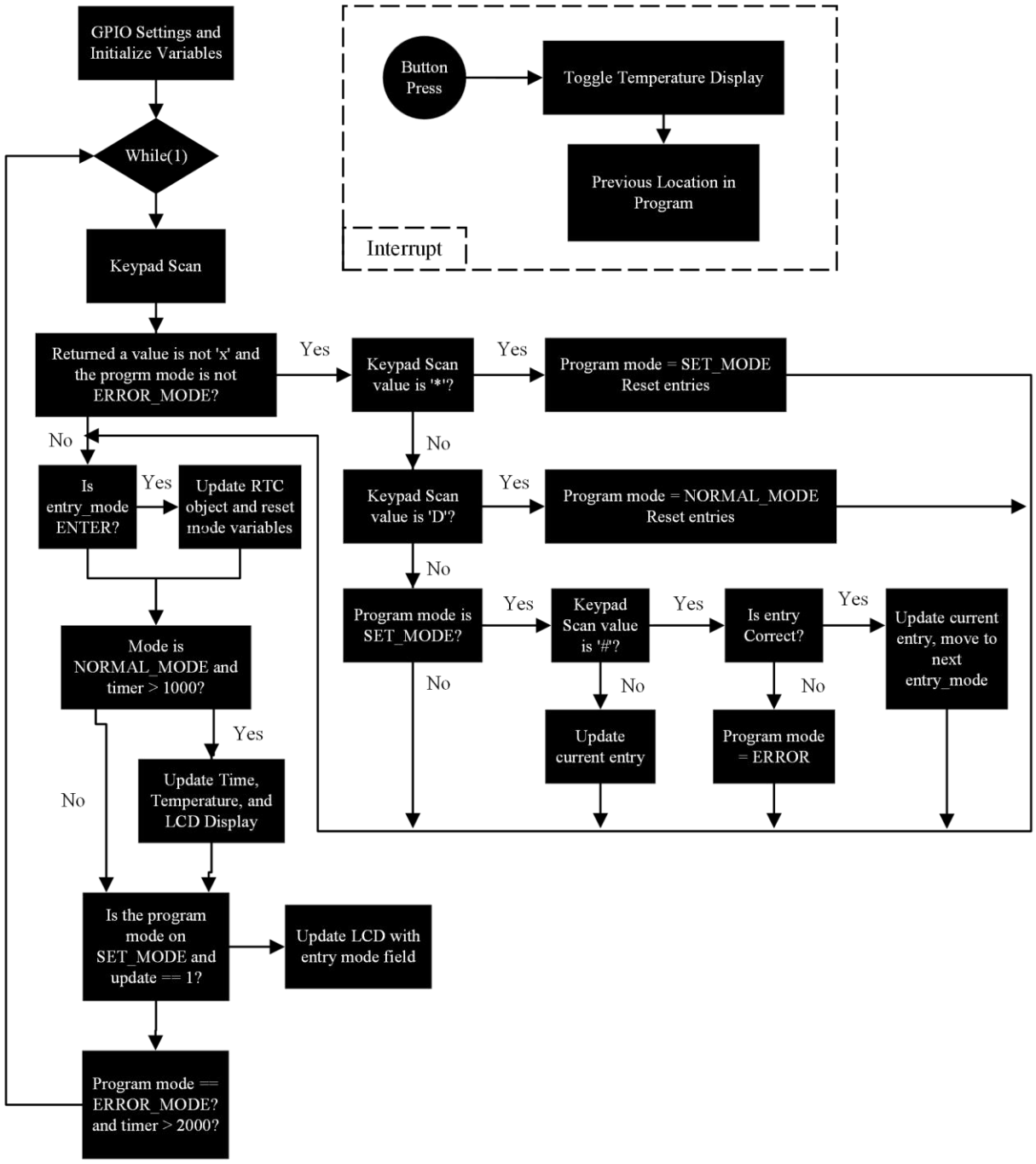


Figure 1: Function Call Graph

V. Appendix A: Program Flowchart



VI. Appendix B: LCD Display C++ Code

```
/**
 * @file lcd_clock.cpp
 *
 * @brief This cpp file was made within Keil
 * Studio Cloud and commented inside of CLion.
 * This program utilizes the mbed OS and the
 * TextLCD header file for 4-bit interface
 * with the HD44780 LCD display controller
 *
 * @author Levi Vande Kerkhoff
 *
 */

#include "mbed.h"
#include "TextLCD.h"
#include <string>

/**
 * @brief Initializes the lcd object using
 * the TextLCD library.
 *
 * The pins on the LCD screen that correspond
 * to the output pins on the Nucleoboard are
 * as follows:
 *
 * TextLCD lcd(RS, E, D4, D5, D6, D7);
 *
 */
TextLCD lcd(PA_0, PA_1, PA_4, PB_0, PC_1, PC_0);

/**
 * @brief This instantiates the temperature
 * sensor analog input pin.
 *
 */
AnalogIn temp_sensor(PC_2);

/**
 * @brief DigitalOut instantiates the GPIO pin objects that
 * read the rows of the 4x4 keypad
 *
 * They are set as output pins.
 */
DigitalOut rows[] = {(PA_6), (PA_7), (PB_6), (PC_7)}; // array of GPIO pin objects -- set to out

/**
 * @brief DigitalIn instantiates the GPIO pin objects that
 * read the columns of the keypad.
 *
```

```
 * They are input pins.
 */
DigitalIn cols[] = {(PA_9), (PA_8), (PB_10), (PB_4)}; // array of GPIO pin objects -- set to in

/**
 * @brief This instantiates the interrupt used to toggle the
 * temperature unit shown on the LCD.
 */
InterruptIn button(PC_13, PullUp);

/** Array of keypad values for user input */
char key_map [4][4] = {
        {'1', '2', '3', 'A'}, //1st row
        {'4', '5', '6', 'P'}, //2nd row
        {'7', '8', '9', 'M'}, //3rd row
        {'*', '0', '#', 'D'}, //4th row
};


/**
 * @brief Scans the columns of the keypad.
 *
 * This checks to see if the input pins are low (if
 * a button is pressed).
 *
 * @return Index of a button press or a "null character".
 */
int col_scan(void){
    for(int i=0; i<4; i++){
        if(cols[i].read() == 0)
            return i;
    }
    return -1;
}

/**
 * @brief Function scans the rows of the keypad and calls
 * the col_scan function.
 *
 * This function also acts as a button debounce when a button
 * is pressed.
 *
 * @return a null character ('x') if no keys are pressed,
 * or return the character pressed.
 */
char keypadScan(void){
    static int row_col[] = {-1, -1};
    static int last_key[] = {-1, -1};
    int delay = 4;
    bool noKeyPressed = true;
    static Timer debounceTimer;
    const int debounce_time = 500;
```

```
    for(int i=0; i<4; i++){
        rows[i].write(0);
        ThisThread::sleep_for(chrono::milliseconds(delay));
        row_col[1] = col_scan();
        ThisThread::sleep_for(chrono::milliseconds(delay));
        rows[i].write(1);

        if(row_col[1] > -1){
            row_col[0] = i;
            noKeyPressed = false;
            debounceTimer.start();
            break;
        }
    }

    /** No key pressed */
    if (noKeyPressed) {
        last_key[0] = -1;
        last_key[1] = -1;
        return 'x';
    }

    /** If the same key is pressed again or the debounce time is not elapsed, return no key */
    if ((last_key[0] == row_col[0] && last_key[1] == row_col[1]) || debounceTimer.read_ms() < debounce_time) {
        return 'x';
    }

    /** Update last key and return the character corresponding to the key press */
    last_key[0] = row_col[0];
    last_key[1] = row_col[1];
    debounceTimer.reset();
    return key_map[row_col[0]][row_col[1]];
}

/** These constants act as mode macros **/
const int HOUR = 0,
          MIN = 1,
          AM_PM = 2,
          ENTER = 3;

/** These constants act as mode macros **/
const int NORMAL_MODE = 0,
          SET_MODE = 1,
          ERROR_MODE = 2;

bool toggle = 0;

/**
 * @brief This function is used in conjunction
 * with the boolean variable, toggle, for
 * controlling the temperature output unit.
```

```
 */
void temp_toggle(void){
    toggle = toggle == 0 ? 1 : 0;
}

/**
 * @brief This function reads the GPIO pin value, converts the voltage
 * value to Celsius and optionally converts further to Fahrenheit.
 *
 * @param toggle
 * @return Temperature value converted from voltage in C or F
 */
int getTemp(int toggle){
    if(!toggle)
        return int((temp_sensor.read()*3300.0)/10.0);
    return int(((temp_sensor.read()*3300.0)/10.0)*(9.0/5.0))+32;
}

int index = 0;
char current_entry[2];
/**
 * @brief This function resets the current time entries.
 */
void reset_entries(void){
    current_entry[0] = '_';
    current_entry[1] = '_';
}



/**
 * @brief Program entry point.
 *
 * The program is organized as follows:
 *      1 - SET UP
 *          - This initializes all variables and
 *            sets the GPIO settings for keypad
 *      2 - OPERATION
 *          - Keypad scan
 *          - Keypad Press Interpretation
 *          - Time Object Update
 *          - LCD UPDATE
 *
 * @return 0, This will never be returned.
 */
int main()
{
    /** SET UP SECTION */

    for(int i=0; i<4; i++){
        rows[i].write(1);
        cols[i].mode(PullUp);
```

```
} /** settings for the GPIO pins */

char key_map_val;

/** This is called in start up to initialize the blank characters */
reset_entries();

/** Get initial temperature and initialize temp. unit characters */
int temp = getTemp(toggle);
char C_F[2] = {'C', 'F'};

/** Time set up */
time_t seconds = time(NULL);
struct tm *timeinfo = localtime(&seconds);
button.fall(temp_toggle);
int hr = 0, min = 0;


static Timer timer;
timer.start();

int mode = NORMAL_MODE, entry_mode = HOUR; /** initializing start mode */
bool update_LCD = 0; /** this tells the program whether or not to update the screen */

/** OPERATION SECTION */
while (1) {

    /**
     * The program constantly scans for key presses
     */
    key_map_val = keypadScan();

    /**
     * This is the conditional entry point for key press entries.
     *
     * Null characters, 'x' represent no key presses.
     *
     * Key presses are not used to update entries when ERROR_MODE
     * is active. This prevents bugs/errors in operation.
     *
     * SET_MODE is activated by the '*' key. Pressing the '*' key
     * will also reset the current entries and take the user back
     * to the HOUR entry menu.
     *
     * Pressing the 'D' key at any time returns the user to
     * NORMAL operation without updating the time.
     *
     * Each entry can be checked/entered by pressing the '#' key.
     * If the entry is incorrect/out of bounds, then ERROR_MODE
     * will be entered for 2 seconds.
     *
     */
```

```
        if(key_map_val != 'x' && mode != ERROR_MODE){
            /**
             * If the '*' key is entered, the program enters SET_MODE and the screen is updated.
             * If the '*' key is pressed while in SET_MODE the entries are reset and the user is
             * returned to HOUR entry.
             */
            if(key_map_val == '*'){
                mode = SET_MODE;
                entry_mode = HOUR;
                update_LCD = 1;
                index = 0; /** Returns user entry to first entry field. */
                reset_entries();
            }
            else if(key_map_val == 'D'){
                mode = NORMAL_MODE;
                entry_mode = HOUR;
                reset_entries();
            }
            else if(mode == SET_MODE){
                if(key_map_val != '#'){
                    current_entry[index] = key_map_val;
                    index = index == 0 ? 1 : 0; /** Switches entry index location. */
                    update_LCD = 1;
                }
                else{
                    if(entry_mode == HOUR){
                        hr = (current_entry[0] - '0')*10 + (current_entry[1] - '0');
                        if(hr < 1 || hr > 12){
                            timer.reset();
                            mode = ERROR_MODE;
                        }
                        else
                            entry_mode++;
                    }
                    else if(entry_mode == MIN){
                        min = (current_entry[0] - '0')*10 + (current_entry[1] - '0');
                        if(min > 59){
                            timer.reset();
                            mode = ERROR_MODE;
                        }
                        else
                            entry_mode++;
                    }
                    else if(entry_mode == AM_PM){
                        if(current_entry[0] != 'A' && current_entry[0] != 'P' || current_entry[1] != 'M'){
                            timer.reset();
                            mode = ERROR_MODE;
                        }
                        else{
                            if(current_entry[0] == 'P' && hr != 12)
                                hr = hr + 12;
                            entry_mode++;
```

```
                    }
                }
                /** This ensures the screen is always updated after a key press. */
                update_LCD = 1;
                reset_entries();
                index = 0;
            }
        }
    }

    /**
     * TIME UPDATE SECTION
     *
     * This section updates the RTC time and resets the mode of operation.
     *
     * */
    if(entry_mode == ENTER){
        timeinfo->tm_hour = hr; // Hour (24-hour format)
        timeinfo->tm_min = min; // Minutes
        timeinfo->tm_sec = 0; // seconds
        time_t new_time_t = mktime(timeinfo);
        set_time(new_time_t);
        mode = NORMAL_MODE; // normal
        entry_mode = HOUR;
        reset_entries();
    }


    /**
     * SCREEN UPDATE SECTION
     *
     * This section updates the LCD Screen.
     *
     * It updates once every second for the time display
     * while in NORMAL MODE.
     *
     * It updates for each keyp press while in SET_MODE.
     *
     * It updates after 2 seconds in ERROR_MODE.
     *
     */

    if((mode == NORMAL_MODE && timer.read_ms() >= 1000)){
        temp = getTemp(toggle);
        seconds = time(NULL);
        tm *timeinfo = localtime(&seconds);
        lcd.cls();
        lcd.printf("%02d:%02d:%02d %s %02d %c",
                   (timeinfo->tm_hour % 12 == 0) ? 12 : timeinfo->tm_hour % 12,
                   timeinfo->tm_min,
                   timeinfo->tm_sec,
                   (timeinfo->tm_hour > 12) ? "PM" : "AM",
```

```
                        temp,
                        C_F[toggle]);
            timer.reset();
        }
        else if(mode >= SET_MODE && update_LCD == 1){
            lcd.cls();
            if(mode == ERROR_MODE)
                lcd.printf("---- ERROR! ----");
            else if(entry_mode == HOUR)
                lcd.printf("HOUR:  %c%c", current_entry[0], current_entry[1]);
            else if(entry_mode == MIN)
                lcd.printf("MIN:  %c%c", current_entry[0], current_entry[1]);
            else if(entry_mode == AM_PM)
                lcd.printf("AM or PM:  %c%c", current_entry[0], current_entry[1]);
            update_LCD = 0;
        }
        else if(mode == ERROR_MODE && timer.read_ms() >= 2000){
            timer.reset();
            mode--;
            index = 0;
            update_LCD = 1;
        }
    }


    return 0;
}
```