Nate Ambrad and Pranav Jain

ECE2049-C24

21 February 2024

Lab3 – Time and Temperature Display

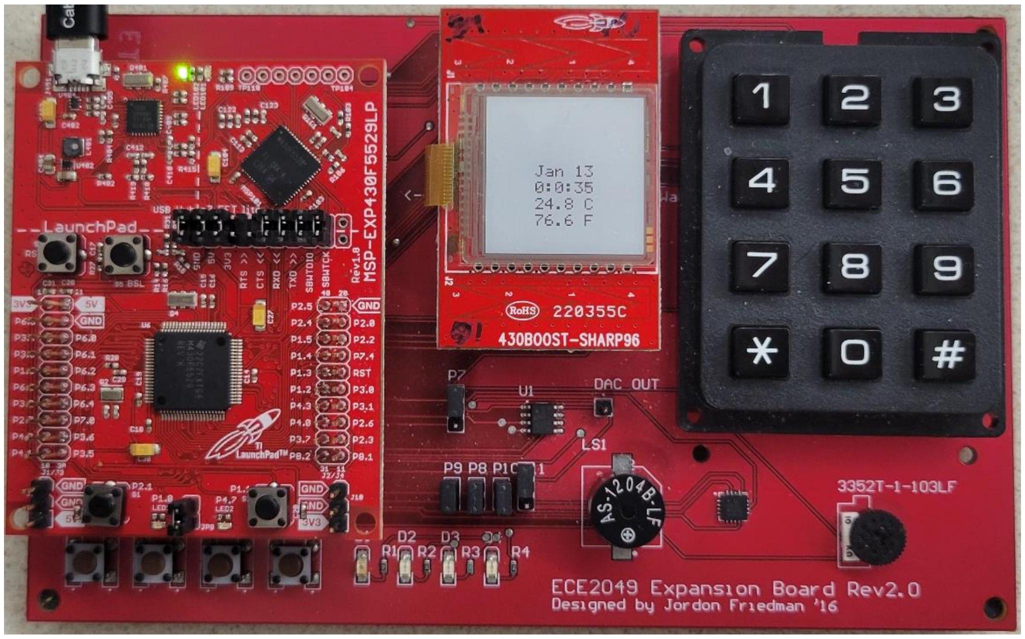

Jan 13
0:0:35
24.8 C
76.6 F
MSP-EXP430F5529LP
430BOOST-SHARP96
ECE2049 Expansion Board Rev2.0
Designed by Jordon Friedman '16

Figure 1 - MSP430F5529 running the time and temperature display program. The date is displayed with month abbreviation, day, hour, minute, and second. The time runs on a 24-hour system. The reading of the internal temperature sensor is also displayed.

Introduction

The purpose of this lab was to create a time and temperature display on the LCD screen by harnessing the capabilities of the MSP430F5529, Internal Temperature Sensor, ADC12, TimerA2, and Scroll Wheel. The display is designed to show the reading of the internal temperature sensor in both Celsius and Fahrenheit, updating every second. The time is shown in months, days, hours, minutes, and seconds with a range of 1 non-leap year calendar year. The time can be stopped for edit mode which allows any time measurement to be changed. The left launchpad user button is used to cycle through each starting with months and circularly going in order of magnitude. Once the time has been edited, counting is resumed by pressing the right launchpad user button.

This lab presented many new design challenges. Unlike previous labs, this program does not have a state machine structure and it is heavily dependent on the ability to properly configure and utilize the ADC12 and TimerA2. The ADC12 was used for converting the temperature sensor reading as well as for using the potentiometer as a scroll wheel to alter time in edit mode. TimerA2 was used to count and track time in increments of 1 second as well as determine the regularity of updating the temperature display.

Discussion and Results

Flow Diagram of Program

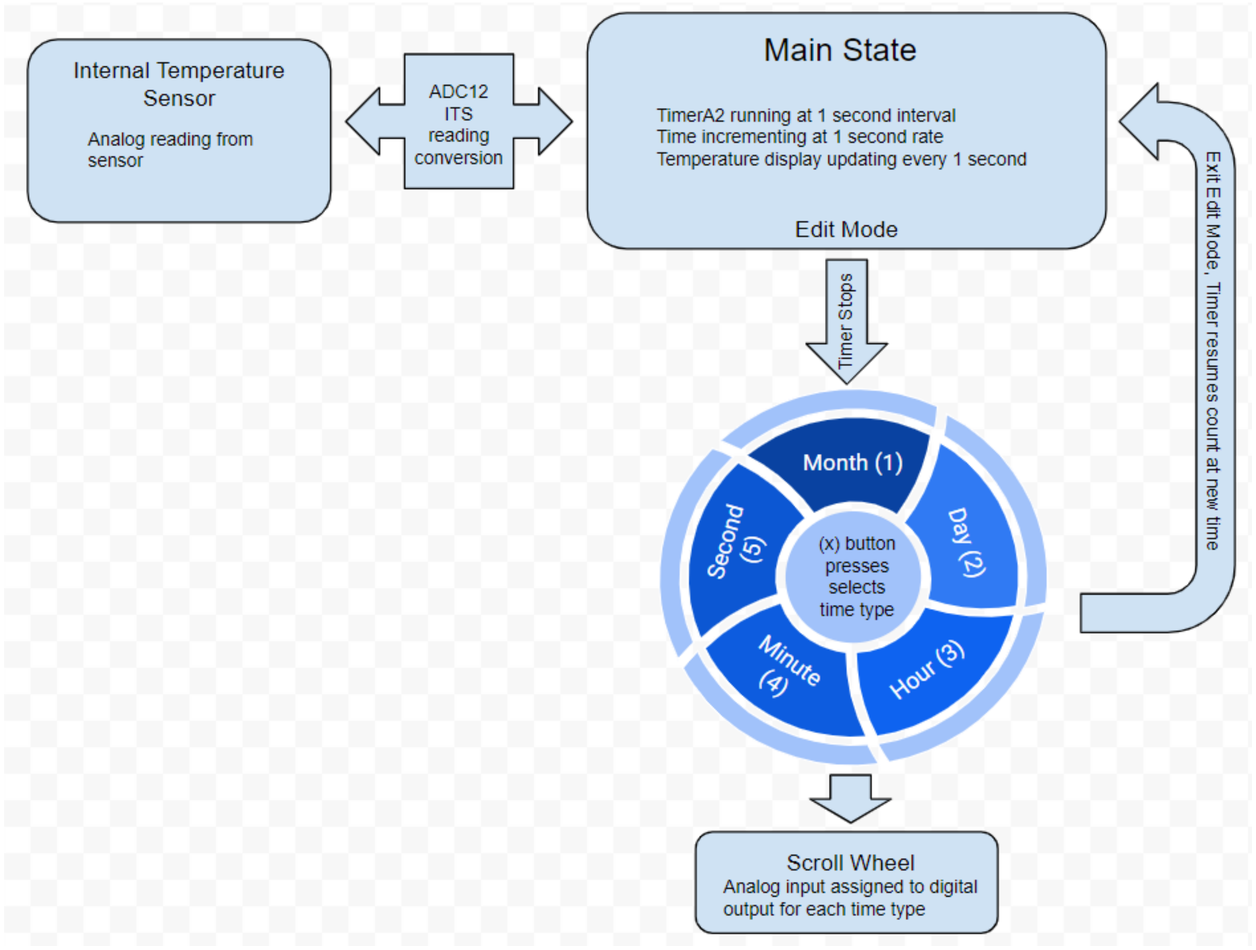

Internal Temperature Sensor
Analog reading from sensor
ADC12 ITS reading conversion
Main State
TimerA2 running at 1 second interval
Time incrementing at 1 second rate
Temperature display updating every 1 second
Edit Mode
Timer Stops
Exit Edit Mode, Timer resumes count at new time
Month (1)
Day (2)
Hour (3)
Minute (4)
Second (5)
(x) button presses selects time type
Scroll Wheel
Analog input assigned to digital output for each time type

Figure 2 - Flow diagram of program. Main state is only interrupted by edit mode which alters the time. Temperature sensor reading is handled by ADC12.

LCD Display

The LCD display is the output of the program. It displays the date and time as well as the internal temperature sensor reading. Two functions, timeDisplay and tempDisplay, are responsible for taking in input data from the ADC12 and using arithmetic to convert that data into months, days, hours, minutes, seconds, degrees Celsius, and degrees Fahrenheit. This data is then put into a character array and cast by using the sprintf function and the graphics library functions to display on the LCD and update at a steady interval.

Use of Timer

TimerA2 was used in this lab. It was set for ACLK with up mode and /1 divider. Max_cnt was set to 16,383 which provided a 1 second interval and no error. The timer was ran during the main state of the program to increment time at a rate of 1 second as well as update the display for the temperature reading. The clock was stopped during edit mode in which the timer count was altered by use of the scroll wheel. When edit mode is ended, the timer begins counting again from the time it was set to during edit mode.

Highlighted Questions

Explain why it is important to pass a copy of the time into the function rather than just using the global variable.

It is better for modularity of the code and there is a delay when converting the time to months, days, etc. from seconds. This delay may be greater than 1 second, meaning that the timer may increase before the completion of the calculation, resulting in an erroneous result. Passing a copy of the time captures the increment if it occurs, so the delay will not cause an error.

What data type did you use to store your time count?

Long unsigned integer. This data type has a maximum value of 4,294,967,295. The largest value required to store is 31,536,000, the length of 1 year in seconds, which is far less than what is available. It is an overkill data type, but it guarantees that there will be no bit overflow.

Justify why your reference voltage choice gives the best resolution. What will the resolution of your readings be in volts and in C^o?

The reference voltage is 2.5 V. 2.5 V was chosen as the reference voltage rather than 1.5 V or 2.0 V because it has the largest range when compared to the supply voltage of 3.3 V. This means that a more significant portion of the input range is available to the ADC. The resolution of the voltage is equal to FSR/2^k with k = 12 for the ADC12. Therefore, res(voltage) = 0.61mV/bit. For C^o, resolution is equal to the difference of the max and min temperatures of 85 C^o and 30 C^o over the calibration code values for each of those temperatures which in this case is 4095 for 85 C^o and 0 for 30 C^o. res(C^o) = (85 - 30) / (4095 - 0) = 0.0134 C^o/bit.

Explain why using "circular indexing" (i.e. something like index = time count modulo 30) to index your array rather than shifting the arrays around is much more efficient.

Shifting the elements would take more CPU processing time and power. Circular indexing helps to reduce this by changing the values in each address of the array one at a time. The modulus 30 helps to circulate back to the start of the array once the index exceeds the array size.

Explain how you mapped the output of the scroll wheel to months, to days, etc.

The output of the scroll wheel was mapped to months, days, etc. by dividing the ADC value by 4095 and multiplying it by 12; 31 or 30 or 28; 24; 60; 60 respectively. These values gave the number months, days, etc. These values were then converted to seconds and output.

Summary and Conclusion

To summarize, this lab accomplished the goal of creating a display for both time and temperature that could be edited to alter time as well as display both Celsius and Fahrenheit readings. The use of the TimerA2 as well as proper configuration of the ADC12 for both the internal temperature sensor and scroll wheel were critical components of the program. The use of software to do the arithmetic on the input values from the ADC12 was important to make the display functional and readable.

In conclusion, this lab served as a good application of analog to digital conversion as well as another case of timer usage. The use of the C language to control the hardware of the custom lab boards results in a reliable and dynamic final product.

Appendices

Pre Labs

Nate Ambrad

```
#include <stdio.h>

// Function prototypes
void displayTime(long unsigned int inTime);
void displayTemp(float inAvgTempC);

// Globals
long unsigned int time_cnt;
const unsigned int monthLength[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
31};
const char* monthAbbr[] = {"JAN", "FEB", "MAR", "APR", "MAY", "JUN", "JUL",
"AUG", "SEP", "OCT", "NOV", "DEC"};

void displayTime(long unsigned int inTime) {
    // Convert inTime(seconds) to Months, Days, Hours, Minutes, Seconds
    long unsigned int totalDays = inTime / (24 * 60 * 60);
    long unsigned int months = 0;
    long unsigned int days = 0;
    while (totalDays >= monthLength[months]) {
        totalDays -= monthLength[months];
        months++;
    }
    days = totalDays + 1; // Adding 1 since days are 1-indexed

    long unsigned int remainder = inTime % (24 * 60 * 60);
    long unsigned int hours = remainder / (60 * 60);
    remainder %= (60 * 60);
    long unsigned int minutes = remainder / 60;
    long unsigned int seconds = remainder % 60;

char timeBuffer[8];

    // Display date and time in specified format
    sprintf(buffer, "%s %d", monthAbbr[months], day);
    Graphics_drawStringCentered(&g_sContext, timeBuffer, AUTO_STRING_LENGTH, 48,
35, OPAQUE_TEXT);
    sprintf(buffer, "%d:%d:%d", (int)hours, (int)mins, (int)secs);
    Graphics_drawStringCentered(&g_sContext, timeBuffer, AUTO_STRING_LENGTH, 48,
45, OPAQUE_TEXT);
    Graphics_flushBuffer(&g_sContext);

}

void displayTemp(float inAvgTempC) {
    // Convert inAvgTempC to AvgTempF
    float AvgTempF = (inAvgTempC * 9 / 5) + 32;

    // Display temperature in C and F to LCD in specified format
    char tempBuffer[8];
    sprintf(buffer, "%d C", (int)inAvgTempC);
    Graphics_drawStringCentered(&g_sContext, tempBuffer, AUTO_STRING_LENGTH, 48,
55, OPAQUE_TEXT);
    sprintf(buffer, "%d F", (int)AvgTempF);
    Graphics_drawStringCentered(&g_sContext, tempBuffer, AUTO_STRING_LENGTH, 48,
65, OPAQUE_TEXT);
    Graphics_flushBuffer(&g_sContext);
}
```

Pranav Jain

```
// Declare globals here
char months[12][3] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug",
"Sep", "Oct", "Nov", "Dec"};
int days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
long long unsigned int prevSec = 0;
long long unsigned int currSec = 0;
char month[3] = "Jan";

int interval = 100;

void displayTime(long unsigned int inTime){
    int i = 0;
    while(1){
        i %= 12;
        long long unsigned int test = days[i];
```

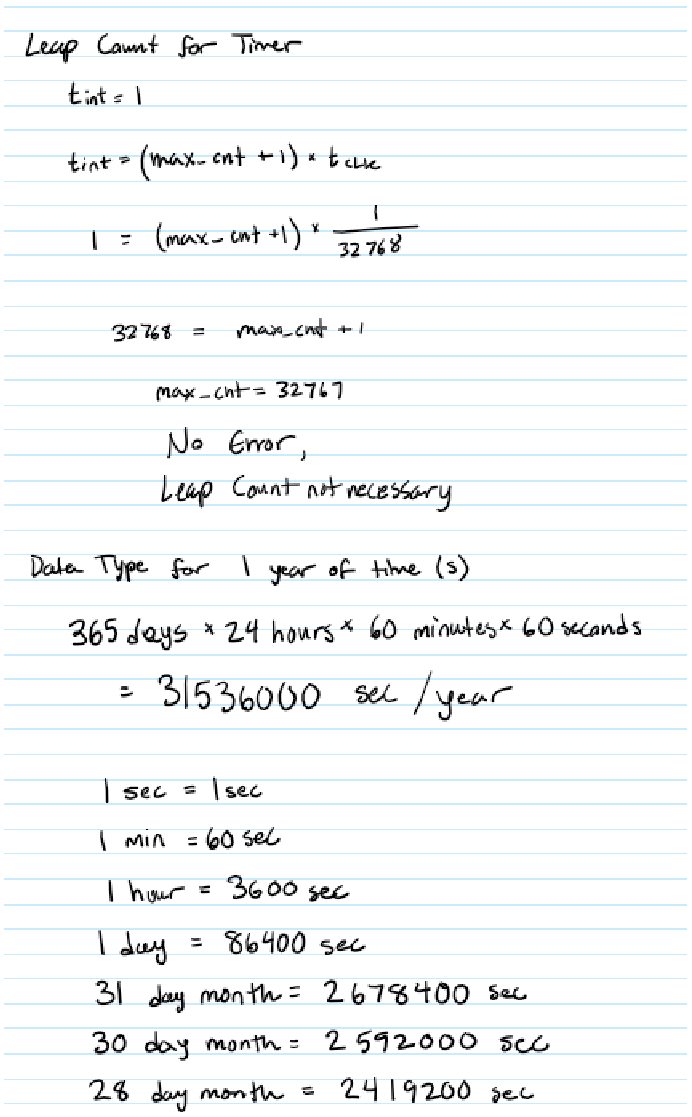
```
            // test = 60*60*24*test;
            test = 86400 * test;
            currSec = prevSec + (test);
            //currSec = prevSec + (days[i]*60*60);
            if(currSec > inTime){
                int j;
                for(j = 0; j < 3; j++){
                    month[j] = months[i][j];
                }
                break;
            }
            else{
                prevSec = currSec;
            }
            i++;
        }
        long long unsigned int left = inTime - prevSec;
        long long unsigned int day = left / (86400); // Days
        left = left % (86400);
        long long unsigned int hour = left / (3600); // Hours
        left = left % (3600);
        long long unsigned int min = left / 60; // Minutes
        left = left % 60;
        long long unsigned int sec = left; // Seconds

        char buffer[8];
        //Graphics_drawStringCentered(&g_sContext, month, AUTO_STRING_LENGTH, 40, 45,
    OPAQUE_TEXT);
        sprintf(buffer, "%c%c%c %d", month[0], month[1], month[2], day);
        Graphics_drawStringCentered(&g_sContext, buffer, AUTO_STRING_LENGTH, 48, 45,
    OPAQUE_TEXT);
        sprintf(buffer, "%d:%d:%d", (int)hour, (int)min, (int)sec);
        Graphics_drawStringCentered(&g_sContext, buffer, AUTO_STRING_LENGTH, 48, 55,
    OPAQUE_TEXT);
        Graphics_flushBuffer(&g_sContext);
    }

    int getDecimalPlaces(float floatTemp, int decimalPlaces){
        int i;
        float floatTemp1 = floatTemp - (int)floatTemp;
        for(i = 0; i < decimalPlaces; i++){
            floatTemp1 *= 10;
        }
        long int intTemp = (long int) floatTemp1;
        return intTemp;
    }

    void displayTemp(float inAvgTempC){
        char buffer[8];
        sprintf(buffer, "%d.%d C", (int)inAvgTempC, getDecimalPlaces(inAvgTempC, 1));
        Graphics_drawStringCentered(&g_sContext, buffer, AUTO_STRING_LENGTH, 48, 65,
    OPAQUE_TEXT);
        float inAvgTempF = (inAvgTempC * 1.8) + 32;
        sprintf(buffer, "%d.%d F", (int)inAvgTempF, getDecimalPlaces(inAvgTempF, 1));
        Graphics_drawStringCentered(&g_sContext, buffer, AUTO_STRING_LENGTH, 48, 75,
    OPAQUE_TEXT);
        Graphics_flushBuffer(&g_sContext);
    }
```

Other Items



Leap Count for Timer

$t_{int} = 1$

$t_{int} = (\text{max\_cnt} + 1) * t_{clk}$

$1 = (\text{max\_cnt} + 1) * \frac{1}{32768}$

$32768 = \text{max\_cnt} + 1$

$\text{max\_cnt} = 32767$

No Error,
Leap Count not necessary

Data Type for 1 year of time (s)

365 days * 24 hours * 60 minutes * 60 seconds

= 31536000 sec / year

1 sec = 1 sec

1 min = 60 sec

1 hour = 3600 sec

1 day = 86400 sec

31 day month = 2678400 sec

30 day month = 2592000 sec

28 day month = 2419200 sec

Figure 3 - Calculations to confirm no error for TimerA2. Calculations done to determine time types in seconds. Used to decide which data types to use for storing time variables.