In this lab, you will use an MSP430 and several peripherals to implement a system that measures temperature and displays the time to the LCD. You will measure temperature using the ADC12’s internal temperature sensor once per second, using a timer. The main purpose of this assignment is to gain some experience with the MSP430’s analog-to-digital converter. In addition, you will make use of some elements from previous labs and homework assignments like timers, decimal-to-ASCII conversion, and textual display of results.

Lab Assignment

Digital time and temperature displays are a common sight. In this lab, you will make a mini-time and temperature display using the MSP430. First, you will implement a UTC-style clock with one second resolution. The temperature sensor you will use is the MSP430’s internal temperature sensor, which is connected to analog input channel 10 (ADC12INCH_10). You will also use the “scroll wheel” potentiometer on the lab board to provide additional functionality.

Pre-lab Assignment

Remember that BOTH partners are required to do the prelab—this is an individual assignment.

Before coming to lab, you should do the following:

1. **READ THE ENTIRE LAB ASSIGNMENT!** Write down any questions you may have to ask in lab.

2. What are the ADCII codes for the decimal digits 0-9? How would you convert the integer 7 to the ASCII character ‘7’? (Hint: the answer is not “put quotes around it.”)

3. Given an integer number (say, 5678), how can you (in code) find the number in the “ones” place? What about the “tens” place? The “hundreds” place?

4. A key part of this lab involves displaying data on the display. While we have used the LCD in our previous labs, we have almost always been displaying pre-defined strings. To display arbitrary values on the display, we need to convert a decimal number into an array of ASCII characters. For example, to display the number 12345, we need to convert it into an array of characters such that:
   

   Start writing a C function that performs this decimal-to-ASCII conversion: given a number and an array, write a function to fill the array with characters corresponding to the given number. An example prototype and usage of this function is shown in the example below.

   When writing your function, remember that 1) an array of characters is displayed from left to right (starting at index 0), 2) strings should be null-terminated, and 3) you should only use integer math.

   ```c
   char arr[10]; // Array with space for 10 characters
   // Call our conversion function, which will fill “arr” with a string
   // representation of 1234
   itoa(arr, 1234); // Pointer to arr is passed as first argument to function

   void itoa(char *s, int val)
   {
     // Inside this function, s[0] accesses the first character
     // of arr, s[1] is the second character, etc.
     // . . .
   }
   ```
Lab Requirements

To implement your time and temperature display, you are required to complete each of the following tasks. As in the previous labs, you do not need to complete the tasks in the order listed. Be sure to answer all of the highlighted questions fully in your lab report.

As always, if you have conceptual questions on how to approach any of the requirements or about specific C programming or software design concepts, please feel free to ask us—we are happy to help!

System Requirements

1. No welcome screen this time—just get right to business: as soon as the program is started, your system should display the date (month and day), the time, and the current temperature.

2. Your system will take temperature measurements once per second. To do this, you should configure TimerA2 to measure 1 second intervals. To keep track of the time, implement a UTC-style clock count capable of holding 1 year of time (in seconds). What data type did you use to store your time count?

3. Since it would be difficult to pre-program your MSP430 with the current time, initialize your UTC count to the last 7 digits of your student ID number (or some similar 7 digit number). To display the time, you will need to write code to convert the UTC count into a number of months, days, hours, etc. Manually convert that number to a date and time to confirm your time conversion code is correct. Include an example of this conversion in your report.

4. Your system will measure temperature using the MSP430’s internal temperature sensor. You should select your reference voltage so provide the best resolution for this sensor. Justify your choice of reference voltage in your report. What is the resolution of your temperature readings in volts/bit and °C per bit?

You can find a complete example (adc12_tempSensor.c) for how to configure the ADC12 to read from the temperature sensor on the labs page of the course website and in the lecture notes, which takes readings using single-channel, single-conversion mode. As a test, you can run this example as a separate CCS project by using the same procedure for loading blink.c from Lab 0. You are welcome to use parts of this example in your code—however, keep in mind that you may want to configure the ADC differently to support the other parts of the lab (for example, you may wish to do multi-channel conversions or use interrupts).

5. Your LCD should continually display the following items, formatted as shown below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>MMM DD</td>
<td>Feb 22</td>
</tr>
<tr>
<td>Time</td>
<td>HH:MM:SS</td>
<td>10:20:49</td>
</tr>
<tr>
<td>Temp (°C)</td>
<td>DDD.F C</td>
<td>24.7 C</td>
</tr>
<tr>
<td>Temp (°F)</td>
<td>DDD.F C</td>
<td>72.3 F</td>
</tr>
</tbody>
</table>

You have a choice for how you can display the items: either print them all on the screen at once and refresh the whole display once per second, or display them one at a time and cycle between them at a rate of once per second. To display these values, you must write your own functions to build strings from your converted values—you may NOT use sprintf() or other built-in functions to do this for you. For more information on this, see the “Important Considerations” section.
6. Configure the ADC to read values from the scroll wheel (using your work from HW 4 as a guide). You should select your reference voltage to provide the best resolution reading the values from the potentiometer. Which reference voltage should you select? If you are unsure, ask us for help! Remember that you are also using the ADC12 to take readings from the temperature sensor. How will you use the ADC to take readings from two inputs? There are many valid way to do this. Explain how you configured the ADC for both sensors and how you set the reference voltages for both inputs.

7. We didn’t hear enough of the buzzer in lab 2, so we will use it again in this lab to give us something to do with the scroll wheel 😊. When the button S1 on the lab board is pressed, the buzzer should sound—the pitch of the buzzer should be controlled by the scroll wheel. That is, when the wheel is turned in one direction, the pitch of the buzzer should go up. When the wheel it turned in the other direction, the pitch should go down. Pressing button S2 on the lab board should turn the buzzer off. It is up to you to decide how you will use the output value from the scroll wheel to control the buzzer’s pitch. Explain how you do this in your report.

8. BONUS (5 pts): Use ADC12 interrupts instead of busy-bit polling to signal completion of your ADC conversions.

9. BONUS (15 pts): Making the scroll wheel control the buzzer pitch is nice, but really we should use the scroll wheel to do something related to the time and temperature display. After demonstrating using the scroll wheel to control the buzzer, implement an “Edit mode” where you can use the scroll wheel to adjust the fields for the time.

   Pressing S1 on the lab board should enter Edit Mode: the months should change by turning the scroll wheel. If S1 is pressed again, the next field (days) is selected, and turning the scroll wheel will adjust its value. Successive presses of S1 moves the user to the next day and time fields (hours, minutes, and seconds), and should cycle back to months after subsequent presses. Pressing S2 should accept the changes (ie, by computing a new UTC time value based on the values for all of the fields) and move back to display mode. For extra fun, you can underline or invert the colors of the display to show the field currently being selected—ask the course staff for information on how to do this.

   How will you map the output of the scroll wheel to months, days, etc? To do this, you do NOT need to consider the current value for the field—just use the scroll wheel to set a new value in the appropriate range\(^1\). Explain how you output of the scroll wheel into months, days, etc. in your report.

10. Write a high quality lab report using the instructions below. Be sure to answer all of the questions in this section!

---

\(^1\) In other words, you don’t need to perform relative edits based on the current field value. Instead, you can use the absolute position of the scroll wheel.
Important Considerations and Coding Standards

All of the previous software design guidelines from the previous labs still apply to this one. In particular, you should remember the following requirements:

- Try to use sane control flow
- Always keep your ISRs short
- Avoid using floating point math when possible

As always, if you have questions on these guidelines, or need help structuring your code, feel free to ask us for help! In addition to the typical coding standards, this lab has two unique software design considerations involving number conversions and string building.

Number Conversions

When converting the time and date, you can sometimes encounter some fun issues relating to how the MSP430 compiler handles constant data types. If you type a line like this:

```c
long x = 24 * 3600;
```

You will see a warning reading “Integer operation is out of range.” This is because the compiler will treat the constants 24 and 3600 as ints before multiplying them, meaning the multiplication will overflow! This is silly, but it makes sense: operations on longs are emulated in software, so the compiler wants to avoid using them as much as possible. To make the compiler actually treat the constant values as longs, you can either use integer suffixes to tell the compiler that the constants are of type long, like this:

```c
long x = 24L * 3600L;
```

... or you can cast any of them:

```c
long x = ((long)24) * 3600;
```

Using either of these options will have the same effect. When you perform any conversions, it is a good idea to check the results of the conversion in the debugger at each step to make sure your math is correct. This can make it a lot easier to debug issues caused by intermediate computations.

The moral of this story is to always pay attention to your compiler warnings, which will point out if your code is affected by this issue. The compiler is your friend—it’s trying to help you out, so listen to it!

String Building

A decent portion of this lab involves formatting data to display it for the user. To do this, you need to turn integer values into strings to draw on the LCD. The standard C library provides a range of functions to do this, including printf() and sprintf(). However, we discourage their use in this class and thus require you to build your own functions to build strings in this lab. Why? To put it simply, sprintf() is big and slow—it uses a lot of code memory, and is extremely slow because it has so many features for formatting its output. In addition, it tends to consume a lot of stack memory when run with complex format strings, which can easily overflow the stack. During our lab section, we will discuss the prelab and how to build strings on your own without using sprintf(). To be more prepared, you may wish to review how strings are stored in C. In addition, feel free to ask for help with string building (or its associated programming concepts) at any time!
Writing your Report

Since this lab was mainly a tutorial, the report does not need to be substantial—however, we are asking you to write one as practice for future reports. Your lab should be written in a professional style. It should be an electronically-prepared technical document like what you would submit to a fellow engineer or your boss. Only one report is required per lab team. The report should include:

- **Introduction** (1-2 paragraphs max): Succinctly state the objectives of the lab and give an overview of what you accomplished.
- **Discussion and results**: Discuss what you did in each part of the lab and how you solved any problems. Describe what you did and be sure to thoroughly answer and explain the questions asked in the lab assignment. In general, this section should be as long as necessary to say what you need—no padding or fluff!
- **Summary and Conclusion** (1-2 paragraphs max): Summarize what you accomplished in the lab and what you learned. This should be a “bookend” to the introduction.
- **Appendices**: You should not need any in this lab. **DO NOT PASTE YOUR CODE INTO THE END OF THE LAB REPORT!** Instead, your code will be submitted as an archive file alongside your report, which is a lot cleaner!

Lab reports are important. In industry, the FIRST view of YOUR work by anybody other than your immediate supervisor will see will probably be in WRITING!

Learning to be an effective communicator of technical information is probably THE MOST IMPORTANT job skill you can have.

Thus, we care about lab reports. We read them. Really.

Submitting your Work

When you are done with your report, you will submit it and your code on Canvas for grading. In order to receive a grade, you must submit both your code and your report online—even though you did not write much code for this lab, we will start the submission process now. Only one member of your team needs to submit files for your lab.

In addition, you must turn in your signoff sheet to the course staff—usually, you will do this when receiving your last signoff. If not, you can turn it in by placing it in the box in the ECE office, or handing it to a member of the course staff.

To submit your code for grading, you will need to create a zip file of your CCS project so that the course staff can build it. You can also use this method to create a complete backup copy of your project (perhaps to send to your partner or save for later). Unfortunately, the only reliable method for doing this is from inside CCS using the instructions below—do NOT attempt to just create a zip file of your code. To export your code:

1. Inside CCS, right click on your project and select "Rename..."
2. If you are submitting your project, enter a name in the following format: `ece2049e16_lab3_username1_username2`, where username1 and username2 are the user names of you and your partner. (NOTE: Failure to follow this step will result in points deducted from your lab grade! If you don’t do it, it makes a lot of extra work for the graders!)
3. Click **OK** and wait for CCS to rename your project.
4. Right click on your project again and select **"Export..."** then select “General” and "Archive file" from the list and click **Next**.
5. In the next window, you should see the project you want to export selected in the left pane and all of the files in your project selected in the right pane. Select all. You should not need to change which files are selected.
6. Click the **"Browse"** button, find a location to save the archive (like your R drive) and type in a file name using the EXACT SAME NAME used in Step (2).
7. Click "Finish". CCS should now create a zip file in the directory you specified.
8. Go to the Assignments page on the class Canvas website. Click on the assignment for Lab 0 and attach the archive file of your project that you just created and your report. When you are ready, hit the Submit button. Only one code and report submission is required per team.