EK307 Lab: Microcontrollers

Laboratory Goal: Program a microcontroller to perform a variety of digital tasks.
Learning Objectives: Learn how to program and use the Atmega 323 microcontroller
Suggested Tools: Arduino IDE, Arduino UNO, LEDs, breadboard.

Pre Lab: Please have the answers ready at the beginning of your lab session.

a) **Read the background**

b) **Draw the circuit in Figure 2 in your notebook. Set the value of R1 to 10kΩ. This value will provide you with a good operating point.**

c) **Calculate the approximate voltage across and current through R1 at 0 C, 30 C, and 100 C.**

You may also want to go to the Arduino website [http://www.arduino.cc/](http://www.arduino.cc/) and explore the function reference [http://arduino.cc/en/Reference/HomePage](http://arduino.cc/en/Reference/HomePage). The Arduino development environment IDE is free to download from their website. The IDE is on the lab computers but you may want to put it on your computer so you can use your board outside of the lab. Some Arduino clones may require a third party driver to access the USB COM port.

For more accurate voltage vs. temperature characteristics look up the datasheet for the thermistor. It is a Vishay NTCLE100E3 10KΩ thermistor.

**BACKGROUND**

The combination of low cost sensors and networked computing has lowered the barrier to ‘intelligent sensing’. For less than 50 dollars one can purchase a microcontroller and sensor and put them to work with a little bit of coding.

Many sensors operate in the analog or continuous domain. In this week’s lab we will use a thermistor which is a resistor that changes resistance with changing temperature. An example of this behavior can be seen in **Figure 1**: the resistance of the thermistor decreases as the temperature increases.
A common way to use a thermistor as a sensor is to use it in place of one of the two resistors in a voltage divider as in Figure 2. Since the resistance of the thermistor (device labeled TR in Figure 2) will change, the proportion of voltage across R1 vs TR will also change. This is because the sum of the voltage across both devices must remain constant due to the 5 volt supply in the KVL loop. Knowing this, it is possible to measure the voltage across R1. Then the voltage across the thermistor can be determined from the loop equation and we find its corresponding temperature using the plot in Figure 1.

In the lab you will build a temperature sensor using this circuit and an Arduino.

Ver. 2/25/2015
Microcontrollers are self-contained computing devices. They are typically low power, single chip solutions. This makes them useful in portable devices because they consume little power and are physically small. Microcontrollers have a CPU, program memory, RAM, and IO (input/output) all on one chip. Many also have additional features such as analog to digital converters (ADC), low power modes, and hardware serial ports.

Having all these features on one chip limits performance in the speed, storage, and memory dimensions. A microcontroller cannot compete with a personal computer in computations per second or memory or storage. Microcontrollers excel at low speed repetitive tasks. They are also good at waiting for events. One example of an implementation is in a wristwatch. A basic watch does not require much computing power. It simply needs to count and display the time of day and provide a user interface (UI) to allow for setting of the time. A watch is such a mundane task for a microcontroller that it turns off between updating the seconds display to save power. Only a small subset of the chip remains awake to maintain the crystal oscillator which keeps time.

Microcontrollers (uC) are connected to the outside world by pins on their packages. The pins are used to provide power, input, and output to the uC. The most basic form of input or output (IO) is a digital IO. Digital IO can have two states, a high and a low. They can be referred to by different names. Table 1 lists some of the names. To send a digital ‘high’ to the uC a voltage equal to the Vcc (power supply voltage) of the chip is applied to a digital pin. To send a digital low to the chip, the pin is connected to ground. Pins can also provide digital outputs. Setting a digital output low means setting it to output zero volts. Setting it high means setting it to output Vcc volts. Our microcontroller has a Vcc of 5 Volts. You can think of a high pin as a 5V source, and a low pin as a short circuit to ground.

<table>
<thead>
<tr>
<th>Logic</th>
<th>Voltage</th>
<th>Binary</th>
<th>Device</th>
<th>Pin</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>State 1</td>
<td>True</td>
<td>High</td>
<td>1</td>
<td>Vcc</td>
<td>On</td>
</tr>
<tr>
<td>State 2</td>
<td>False</td>
<td>Low</td>
<td>0</td>
<td>Ground</td>
<td>Off</td>
</tr>
</tbody>
</table>

Table 1: Some common digital logic representations.

Analog input is used by a microcontroller to read sensor data. Unlike digital inputs, analog inputs read a continuum of voltage that typically has a range from 0 to Vcc. Your temperature sensor is analog thus will be read by an analog input. Because a microcontroller is a digital circuit it can only approximate analog input with a certain precision. The built in Arduino ADC is 10 bits. The range of integers in ten bits is from 0 to 1023. In other words the ADC must put its reading of the analog pin into an integer bin ranging from 0 to 1023. The ADC maps voltages from 0-5V (0-Vcc) to the range 0-1023. This means that it quantizes the range of 0-5 volts into 1024 bins. If we divide 5 by 1024 we get 4.88 mV which is the bin size. Therefore the minimum resolution we can detect is 4.88mV. For example, if our analog pin was at 2mV when it was read it would return the same integer as if it was at 4mV. This integer would be 0. If the pin was at 5mV when it was read the integer would be a 1. If it was at 5 volts the integer would be 1023.

THE LAB: LEVEL 1

Ver. 2/25/2015
Using a thermistor and an Arduino, make a temperature sensor that indicates when the temperature is above 30 C by turning on an LED. For this lab there are two main parts: constructing the hardware and writing the software.

**Figure 3.** Schematic of Arduino connected to thermistor circuit and LED. The values of C1 and C2 are not critical. They serve to smooth out the signal. They can be in the 0.01 to 0.1μF range. R1 is 10kΩ. R2 should be selected to limit the current in the LED to approximately 10mA.

**Figure 4.** Sample layout of the complete circuit from figure 3. Note that the jumper wires plug into the header sockets on the ‘duino. The ‘duino pins are labelled on the printed circuit board (PCB)

**Construct the Hardware**

The basic thermistor circuit in Figure 2 will be modified as shown in Figure 3 and be constructed on your breadboard as show in Figure 4. The thermistor is available at the window

Ver. 2/25/2015
in the lab. The 5-volt supply for the circuit should be taken from the Arduino board. The two capacitors are there to smooth out the voltage. We will learn more about this type of filtering later in the semester. Their values are not too important and can be in the range of 0.01µF to 0.1µF. The voltage divider node from the thermistor and R1 is connected to analog input 0 on the Arduino. Digital Pin 7 is used as an output to drive the LED. Remember from Lab 1 that LEDs must be current limited. Thus, the purpose of R2 is to limit current flowing into the LED. This current limiting protects both the LED and the Arduino from damage. Choose an appropriate value for R2 such that approximately 10mA (+-2mA) flows through the LED when the logic pin is high (at 5 volts).

It is important to note that microcontrollers such as the one on the Arduino do not tolerate voltages on their pins that are greater than Vcc or less than ground. Be careful when using them with external power supplies.

The Arduino can be powered from a USB port. It is also programmed through the USB port. Connect the Arduino to your computer.

**Write the Software**

There are two main components to the software or code you will write for the temperature sensor. The first part is initialization or setup code. This is run only once when the Arduino is powered up. It sets all the appropriate ports and peripherals to perform the functions you want. One example is programming the pins. Since one pin may have multiple functions you need to specify what is should do. In the case of the pin connected to the LED you want to set it as a digital output so it can output voltage to the LED. The connection from the thermistor sensor divider should be an analog input since we want to read analog input. You may also want to set up serial communication with the console on the computer so you can send information to your computer for debugging purposes.

The main loop runs after the setup. This is where the real time action takes place. The loop runs forever or is endless. This is good for this type of application because we want the temperature sensor to be reading whenever it is turned on. In this loop you want to read the analog port, do any computations to determine the temperature, decide whether the LED should be turned on, possibly turn it on or off, then wait until the next iteration through the loop.

Humans do not usually equate temperature to bits. Because of this we may need to convert the bit values read from the ADC to a value that humans can interpret such as Celsius or Fahrenheit. Starting from the thermistor, in Figure 1 we saw that the resistance of the device is dependent on temperature. This dependency affects the voltage across R1 in Figure 2. If R1 is being read by the microcontroller ADC its value will be in bits. In the prelab you calculated the voltage across R1 for a few temperature values. If you divide the voltage across R1 by 4.882mV you will get the approximate digital bin that the read voltage will fall into. Once you know this bin value you can write control statements that turn the LED on or off depending on the value. One example of a control statement is an IF ELSE. This can be used to control the LED by saying “if the value of the ADC output is greater than x then do something…”

The Arduino function reference [http://arduino.cc/en/Reference/HomePage](http://arduino.cc/en/Reference/HomePage) is a good place to get information on the functions. It also has many examples. To assist in what may be your first
embedded computing program we put skeleton code up on GitHub for you to use. The code compiles but is not complete. You can use it to get started. It can be found here: https://github.com/EkZosuls/EK307Lab3/tree/master/arduinoLabSkeletonSpring2015

Testing Your Circuit

The room temperature in the lab is below 30C and your skin surface temperature is probably above 30C unless you were holding something cold or just came in from the snow. Since we may not have an accurate thermometer in the lab, demonstrate that the LED lights when you hold the thermistor and turns off when you let go of the thermistor. When using the thermistor to measure temperature try not to touch the metal leads since your fingers are conductive. There will be a time delay on the thermistor reading because the thermistor has thermal resistance and heat capacity. Another way to heat the thermistor is to connect an appropriate resistor to your power supply and put it in close proximity to the thermistor. You can use the IDE serial monitor to see what your microcontroller is outputting to the console.

Troubleshooting Your Circuit

Things may not work the first time around. That is ok. You can use the multimeter to test that the voltages in your circuit are what you expect. Similarly you can use print statements in your code to test states of your variables. Print statements can be thought of as a multimeter for software. For example, print the value of your ADC read to the console on your computer. If it is reading all zeros then maybe there is no power to the thermistor. If it is wildly fluctuating maybe there is a loose connection or no ground. We do expect a few bits of fluctuation in the ADC values because there is noise in the system.

Another tactic to coding is to build the project up in stages. For example, write code that simply turns on the LED, waits a few seconds, and then turns it off. Once this is working and proven by actually seeing the LED turn on and off then you can add the next part which would be measuring temperature by reading the ADC.

LEVEL 2

Modify the Level 1 circuit and software to make a visual hi/low temperature alarm. The alarm should flash an LED once per second if the temperature is below 20C. If the temperature is above 30C the alarm should flash the LED 3 times per second. From 21 to 29 C the LED should remain off. Once the alarm works disconnect your Duino from the USB port and power it from the bench supply. Supply it with 8 volts between the VIN and GND pins on the Duino. Be sure you are connected properly and the voltage is less than 12 volts or your board may go up in smoke. Now measure the current that is being delivered to the board. How does it change when the LEDs are on?