Digital Thermometer

Posted on February 27, 2008, by Ibrahim KAMAL, in <u>Sensor & Measurement</u>, tagged

This article shows you how to build a digital thermometer from the beginning to the end, using a thermistor and a 8051 micro controller. Being based on our tutorial about Analog to Digital conversion, it is very easy to understand the functioning of the device, and you can build it with any micro controller even if it doesn't have a built in ADC.

Note: This article relies on a famous analog To digital conversion scheme called counting type ADC. In case you are not familiar with this concept, refer to this dedicated <u>tutorial</u> for more information.

The following flow chart shows the general principle of operation of a digital thermometer, and each one of those four sections shall be studied in this article.
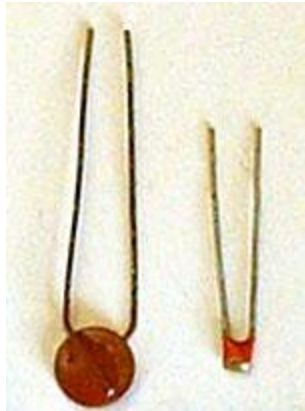

The temperature sensor

figure 1.A

The temperature sensor used in this project is thermistor which is very suitable for measuring ambient atmospheric temperatures, but you could replace it with any other type of temperature sensor, that would function in a range that is more adequate to your application. The response time of thermistors is relatively big, but again, the performance of a thermistor is accepted for our application.

A thermistor (***figure 1.A***), is a resistor whose resistance varies with temperature. They come in two types: NTC and PTC. NTC stands for negative temperature coefficient, meaning that the resistance of an NTC thermistor will decrease if temperature increases, while PTC stands for Positive temperature. coefficient,

and PTC thermistors behave in the opposite way: An increase in temperature causes an increase in resistance.

The thermistor used in this project is an NTC type (the small one in *figure 1.A*). In order to be able to read the resistance variation (corresponding to temperature variation), we need to convert this resistance variation into a voltage variation, which will be proportional to the temperature. To do this we simply insert the thermistor into a voltage divider configuration shown in *figure 1.B* (The 10K resistor is not part of the voltage divider).
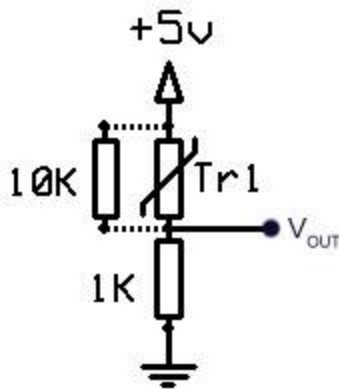

figure 1.B

The following equation describes the relation between the voltage of the point Vout:

$$V_{OUT} = \frac{R_{TH}}{1 + R_{TH}} \times 5$$

where resistances are expressed in K ohms
and R_{TH} is the resistance of the thermistor

It is clear That Vout increases when R decreases, then Vout increases proportionally with temperature.

Sometimes the response of thermistors are not quite linear, so adding a 10K parallel resistor increase the linearity of the response of the thermistor over a specific range which corresponds to ambient temperatures.

The mathematical proof for this application of increasing the linearity of a signal would require a dedicated article, which is not the target of this one.

The Analog to digital converter

As you noticed in the last paragraph, the Temperature was translated to a directly proportional voltage, and the relation was considered to be linear over the concerned operating range. Now we need to convert those signals to digital signals, so that the microcontroller can read it, store it and display it. Many of the

recent microcontrollers incorporate integrated digital to analog converters, but for educational purposes we are going to assemble our own counting type ADC which is shown in the blue shading in *figure 2*. The operation of such a converter is explained in detail in this tutorial about counting type ADCs. One important advantage of manually building your own DAC is that you can easily increase the resolution of your converter while this is not always possible when you are working with integrated ADCs. In this application, we used a 10 bit ADC.

The output of the the last stage (the analog voltage) is fed to the ADC via the non-inverting pin of the LM358 Op-Amp (U$_2$). The rest of the components are the 7-segments display system (**in the red shading**), the voltage regulator to provide clean 5V for the microcontroller, the Jack J$_1$ for the ISP (In System Programming) and finally the reset circuitry coposed of R$_{12}$ and C$_3$ and the 24 Mhz crystal.

If the voltage being converted changes from 0 to 5 volts, then the resolution of our converter would be:

$$\frac{5}{2^{10}} = 0.0048828125v$$

Which is sufficient for our application.
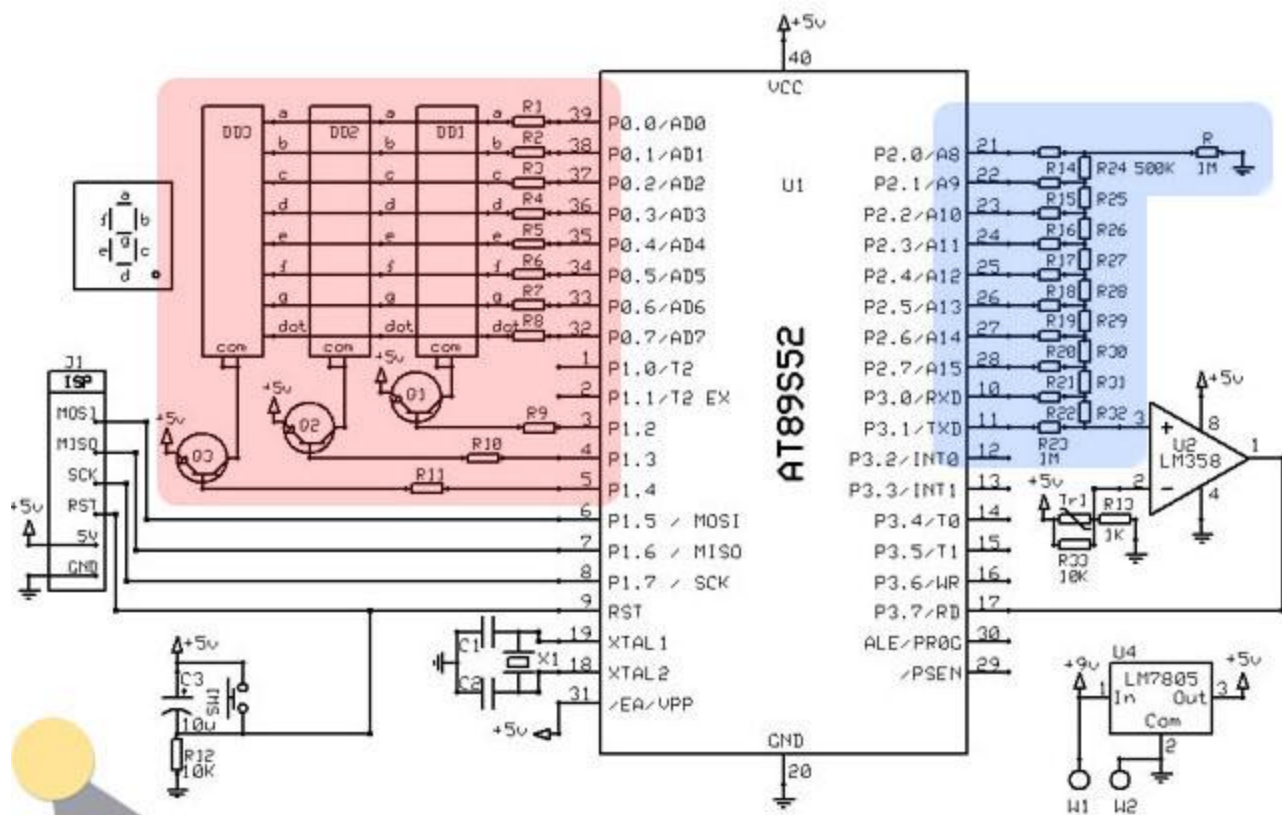
Q$_1$, Q$_2$,Q$_3$: 2n2907 R$_1$ to R$_{11}$: 220 Ohm, R$_{14}$ to R$_{23}$: 1 MOhm
R$_{24}$ to R$_{32}$: 500 KOhm, J$_1$: connection for <u>ISP programmer</u>

So, with the help of the ADC, the micro controller is able to determine a 10 bits digital value (from 0 to 1024) that corresponds to a certain temperature. Those values still don't provide any direct temperature indication, but they are considered to be directly proportional to the temperature.

Calibration and signal processing

After the analog signals are converted to 10 bit digital values, some operations have to be done to convert that number into a temperature in Celsius. This is done using the following formula:
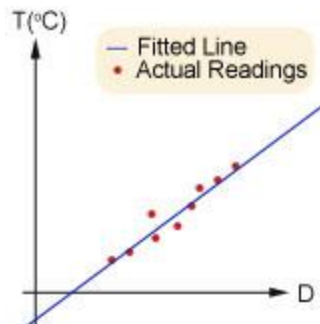
$$T = (F_c \times D) + C$$

figure 3.A

Actually the formula above is the canonical form of the equation of a line, where T is the temperature in °C and D the raw reading obtained from the ADC. C is a constant number. The calibration process is all about determining the value of the conversion factor F$_c$ and the constant C. IF you have the datasheet of the thermistor, you could calculate those variables, but still it would be a very complicated operation. The method I am proposing is called line fitting, which consist of drawing the line that passe as near as possible from a maximum of readings so that it can be used later on to calculate new points. It is the reverse process of readings points from a graph, now you have the points, but you want to build the graph that is at the origin of those points. This graph can never be precisely found because it never existed, but a very similar one can be 'fitted' and used later to simulate a linear relation ship over a wide range of readings.

Lot of free software on the internet will calculate for you the two variables F$_c$ and C, Personally I used my scientific calculator.

In other words, in order to precisely calibrate the digital thermometer, we need to run a multitude of test measurements, noting at the end of each test the actual temperature measured from a reference thermometer (equivalent to T in the graph) and the reading of our thermometer (equivalent to D). At the end of those test you obtain a series of temperatures and corresponding ADC readings. Feeding those values into a scientific calculator or a line fitting software will provide you with the values of F$_c$ and C

It is clear that the Calibration process can only be started when the device is properly functioning and already incorporates a display system to read the results.

Then all we have to do is to display the temperature that was just calculated from the formula above, but before doing this,we can apply some digital filtering to the signals, to reduce the effect of any eventual noise. Note that temperature is a measurand that is relatively stable so any fast fluctuation in the reading would be unlogical, would cause a lot of undesired flickering in the display and would probably be caused by the noise anyway.

That digital filter can be implemented simply by displaying the average of the last five readings rather that directly displaying the values calculated by the microcontroller. This can be implemented in C code as in the following example:

```
temperature         =         (adc_data         *         conversion_factor)         +         constant;
reading_5                                =                                reading_4;
reading_4                                =                                reading_3;
reading_3                                =                                reading_2;
reading_2                                =                                reading_1;
reading_1                                =                                temperature;
actual_temperature = (reading_1+reading_2+reading_3+reading_4+reading_5)/5;
```

Where the variable 'actual_reading' is the reading to be displayed.

The display system

figure 4.A

As you can see in **figure 4.A**, three 7-Segments cells are used to display the temperature. The main idea for this display system, is to connect all the 7-segment cells together in parallel (all the cells show the same digit), but only power the first cell, then switch the first cell off, and power the second one, then do the same thing with the last one and repeat this cycle at a very fast rate. If you provide the appropriate DATA to the cells at the appropriate time, the number will be displayed without any noticeable flickering. See the display system of this Frequency Meter for more information.

The decimal point is fixed (from the software) at the middle cell, so there is always one number after the point. All the cells are common anode type, the cathodes of each cell are connected with the cathodes of the other cells, then directly connected to the Port 0 of the micro controller. We could have saved four pins of port 0 by using a BCD to 7-Segment decoder, but since we don't need those pins, we choose to connect them directly and decrease the number of components.
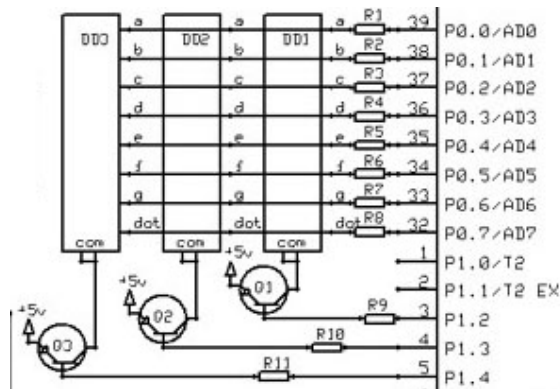

figure 4.B

Figure 4.B is a small slice of the full schematic, showing the electrical connection for the display system. DD_1 to DD_3 are the 7-segment displays. Port 0 of the 89S52 outputs the the bits configuration corresponding to the numbers to be displayed, while pin 2,3, and 4 of Port 1 switch on or off the cells via a 2N2907 PNP Transistor. (NPN would have been used for common cathode displays). Note that PNP transistors are 'turned ON' when their base is connected to ground. The software routine to operate this display can seem complicated, but it only executes the sequence described above, which consists of turning ON one cell of the display, displaying the corresponding number and keep cycling between the three cells abundantly.

The following function is the one used by the thermometer to display the temperature, it is periodically called by the timer 0 interrupt:

```
display_sequence() interrupt 1{

    display_period++;

    if (display_period > 15){

        display_period = 0;

        segment_counter++;

        if (segment_counter > 2) segment_counter = 0;


        switch(segment_counter){

        case 0:
```

```
        P1 = 255;           //Write 1's on the pins of Port 1

        P1_2 = 0;           //Then switch on the transistor Q1

        P0 = bcd[dig[segment_counter]];//Display the corresponding digit

        break;

    case 1:

        P1 = 255;

        P1_3 = 0;

        P0 = bcd[dig[segment_counter]]-128;

        break;

    case 2:

        P1 = 255;

        P1_4 = 0;

        P0 = bcd[dig[segment_counter]];

        break;

    }


    }

}
```

You can notice that the whole function's execution rate can be adjusted using the variable 'display_rate' when the maximum timing of the timer 0 is still too fast. In order to understand the code, it's important to note that the temperature is stored in the array 'dig', where:

dig[0]   =   first   digit   to   be   displayed
dig[1]   =   second   digit   to   be   displayed
dig[2] = third digit to be displayed (the decimal)

So each time the function is executed, the variable 'segment_counter' is incremented in its cycle from 0 to 1, to 2, and then back to 0, and each time the corresponding digit is written on port 0 using the following statement:

P0 = bcd[dig[segment_counter];

Where bcd[] is an array where are stored the bits configurations to show the digits 0 to 9 on the seven segment display, according to the connection diagram in *figure 4.B*:

bcd[0] = **192;**
bcd[1] = **249;**
bcd[2] = **164;**
bcd[3] = **176;**
bcd[4] = **153;**
bcd[5] = **146;**
bcd[6] = **130;**
bcd[7] = **248;**
bcd[8] = **128;**
bcd[9] = 144;

You will also notice that in case of the middle cell, the data written to P0 is different:

P0 = bcd[dig[segment_counter]]-128;

The fact of subtracting 128 to the number being displayed makes the last bit (P0_7) always Low, this way the led of the decimal dot is always ON.

One last note about the display system, is that there is another function that have to be called before displaying the digits of the temperature:

int_to_digits(number);

Which will take as an argument the the temperature in a single variable, then store the seperate digits into the array dig[0,1,2]. The detailed function, which was modified from a code provided by KEIL™, is as the following:

```
void int_to_digits(unsigned long number){
float itd_a,itd_b;
number = number * 10;
itd_a = number / 10.0;
dig[0] = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
dig[1] = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
dig[2] = floor((modf(itd_a,&itd_b)* 10)+0.5);
itd_a = itd_b / 10.0;
dig[3] = floor((modf(itd_a,&itd_b)* 10)+0.5);
}
```

As you can see in the function above, the number is multiplied by 10, this way we don't have to worry about the decimal point which is fixed on the middle cell. This way, a number like '24.3′ will become '243′, each cell will display one of those three digits, and the decimal point being fixed at the middle cell, the number displayed will be '24.3′. An integer like '28′ for example will be displayed as '28.0′.

I hope this article covered all the main aspects of the construction of a simple digital thermometer.

Source: http://www.ikalogic.com/digital-thermometer/