Basic Analog to Digital Thermometer using Atmega328p

Douglas Smith, Noah Harvey

November 12, 2014

Abstract

A thermometer is a device used to measure and display the temperature of an external environment. A typical thermometer can be created using an Atmega328p microcontroller, thermistor and seven segment display. In the design presented temperature is converted to an analog voltage signal by the thermistor. The voltage signal is then converted to a binary number by an analog to digital converter and is stored in the Atmega328p. Linear interpolation is then performed on this number to obtain a temperature value which is then displayed via a two digit seven segment display. The temperature conversion and display tasks are controlled asynchronously via two interrupt service routines. Testing of this method showed that this design worked for temperatures ranging from $0 - 40°$C.

This report is divided into two sections; the first half is for program logic and the second is for hardware. The software section will cover configuration of the ATmega328p's timer for updating the seven-segment display and digit multiplexing, the ADC for converting the thermistors voltage signal into digital data and extracting the thermistor temperature from the data. The hardware extion will cover setting up the Dragon programmer, powering the Atmega and the ADC, seven-segment pin layout and setup, and connecting the thermistor.

Chapter 1

Program Logic

1.1 Overview

The Atmega328p application program uses two processes to asynchronously control the external display and temperature conversion tasks (as shown in Figure 1.1). Temperature calculations are handled by the ADC interrupt service routine (or ADC ISR). The ADC ISR obtains the result from the ADC and gives the value to the temperature conversion interface to handle temperature calculations. The setNum() function is then called to set the temperature to display which is handled by the two digit display interface. An eight-bit timer on the Atmega328p executes the timer ISR which uses the data in the two digit display interface to control the seven segment display.

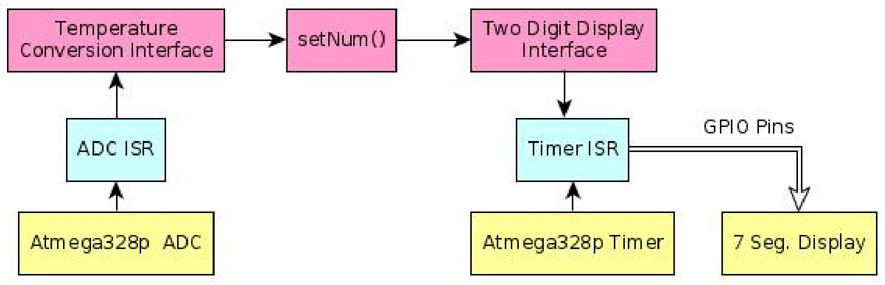

Figure 1.1: Thermometer Program Logic

1.2 Configuration

The above program execution begins after a basic configuration process.

Figure 1.2 shows the boot process for the Atmega328p.

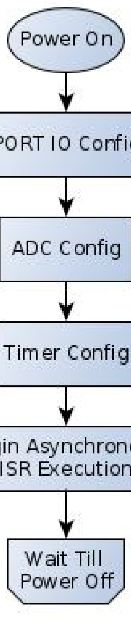

Figure 1.2: Atmega328p Configuration Process

1.2.1 Timer Configuration

The eight-bit timer is configured to run at approximately fifty-two kilohertz in Clear Timer on Compare mode. CTC mode is an operation mode of the timer in which the internal counter is reset when the counter matches the value stored in the timer's output compare register. Once the timer is reset a timer interrupt service routine (or timer ISR) is executed.

1.2.2 ADC Configuration

The ADC is configured to run in free running mode. This is an operation mode in which the ADC is constantly performing conversions. Once a conversion is performed an ADC flag is set and an interrupt is called and the ADC immediately begins performing another conversion. The ADC uses the ADC5 port as the voltage signal input.

1.2.3 IO Register Configuration

Table 1.1 shows specific ports and their respective pin configurations. These configurations are set before the execution of the two ISRs (timer and ADC).

Port	Pins	Configuration
PORTD	0-7	Output pins to control segment LEDs on seven-segment display.
PORTC	0-1	Output pins to control digit selection on seven segment display.
PORTC	5	Input pin for ADC

Table 1.1: Atmega328p Pin Configurations

1.3 Two Digit Display Interface

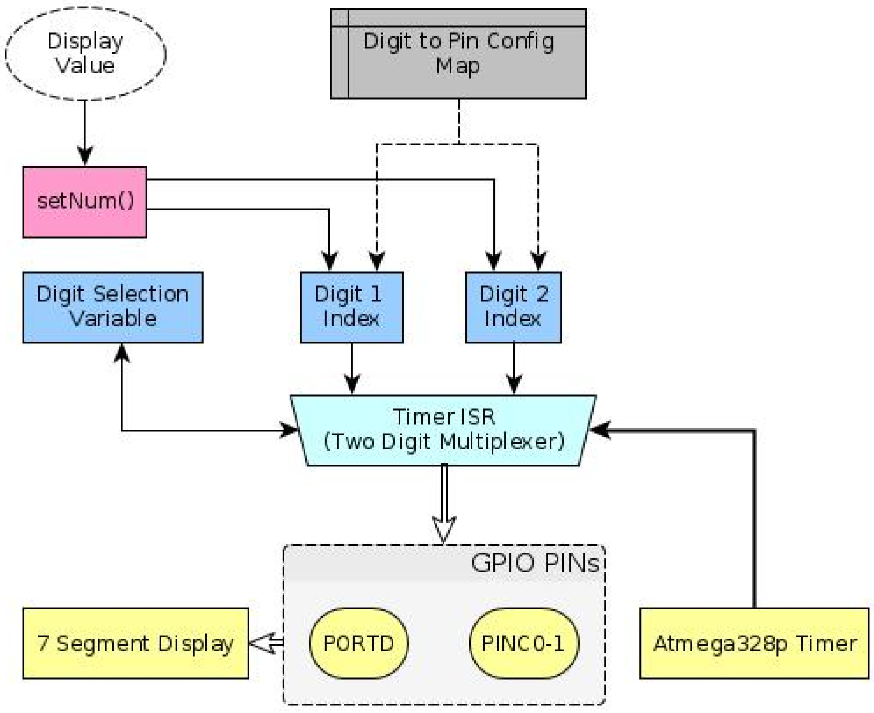

Figure 1.3: Two Digit Display Interface

1.3.1 Single Digit Control

The two digit seven segment display is controlled by the Atmega328p via two I/O ports. PORTD controls the segments on the display and PORTC controls the digit selection. A mapping array is used to map digits 0 through 9 to their respective output register values. For example, the value at index 3 in the mapping array stores the respective value for the PORTD register. This register value is then used to show the number 3 on the seven segment display. Two variables store indexes to this mapping array, each for a single digit on the display. These variables are set by the setNum() function and are used by the timer ISR to set the PORTD register. A variable is used to determine which digit to display when the ISR is called. Figure 1.3 shows an overview of the display system.

1.3.2 Two Digit Multiplexing

To display a two digit number a multiplexing method is used. This is done by alternatively toggling each digit on the display so that only one digit is on for a given time period. When the digits are alternatively toggled at a frequency larger than 10kHz they appear as if they are displaying simultaneously.

The timer ISR handles the multiplexing of the two digits. Below is the process used in the timer ISR to perform two digit multiplexing:

1. Turn currently displayed digit off.
2. Select other digit to display using the digit selection variable.
3. Turn selected digit on using the corresponding index variable.

1.4 Temperature Conversion Interface

The Atmega328p has a ten bit successive approximation analog to digital converter (noted as ADC). Successive approximation is an analog to digital conversion method in which the input signal is constantly compared to the output of a guessed analog value. (see Figure 1.4 for a simplified flow diagram of a successive approximation ADC.). After all n bits of the converter are set the ADC raises a conversion the ADC interrupt service routine (or ADC ISR) is called to handle temperature conversion (see the ADC to Temperature Conversion section below).

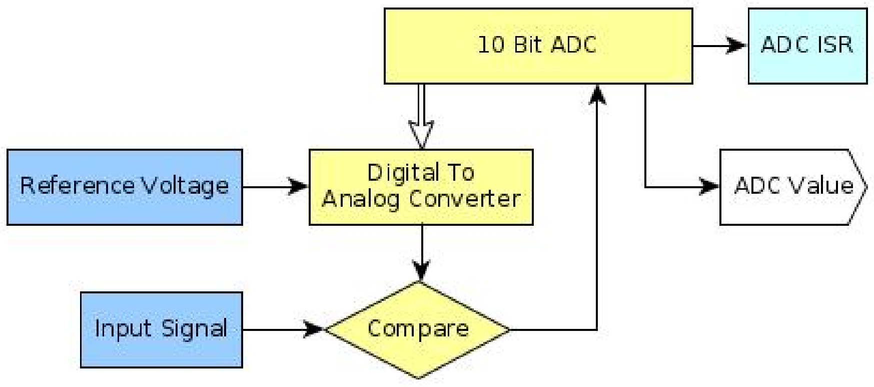

Figure 1.4: ADC Successive Approximation

1.4.1 Obtaining the ADC Conversion Result

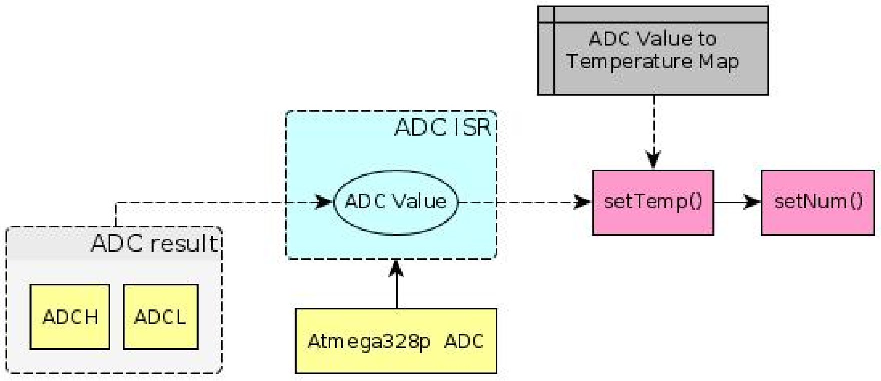

Figure 1.5: Temperature Conversion Interface

The ADC uses two eight bit registers (called ADCL and ADCH) to store the result from an analog to digital conversion. The conversion result is obtained by fetching the ADCL register data first (which is required according to the Atmega328p datasheet) and then the ADCH register data. The final result is stored in a sixteen bit integer (noted as the ADC value) which is then passed to a temperature conversion function setTemp(). Figure 1.5 shows the process in which the temperature conversion interface was designed.

1.4.2 Calculating the Temperature

A function called setTemp() handles the conversion of an ADC value to a temperature. Below is the process for which setTemp() uses to convert and ADC value.

1. Fetch nearest data points from Temperature conversion table.
2. Perform linear interpolation using integer math to compute a temperature.
3. call setNum() to display the result to the seven segment display.

Linear interpolation was the method used to calculate temperatures based on given ADC values. See the below section for details on linear interpolation.

1.4.3 Linear Interpolation

Linear interpolation is a method of estimating function values from a table of known input and function values. The idea is that two points are chosen from the table (one greater than, one less than the desired value) and a linear equation is formed from those two points. Then the desired function value is found by computing the formed function with a desired given input value into this function.

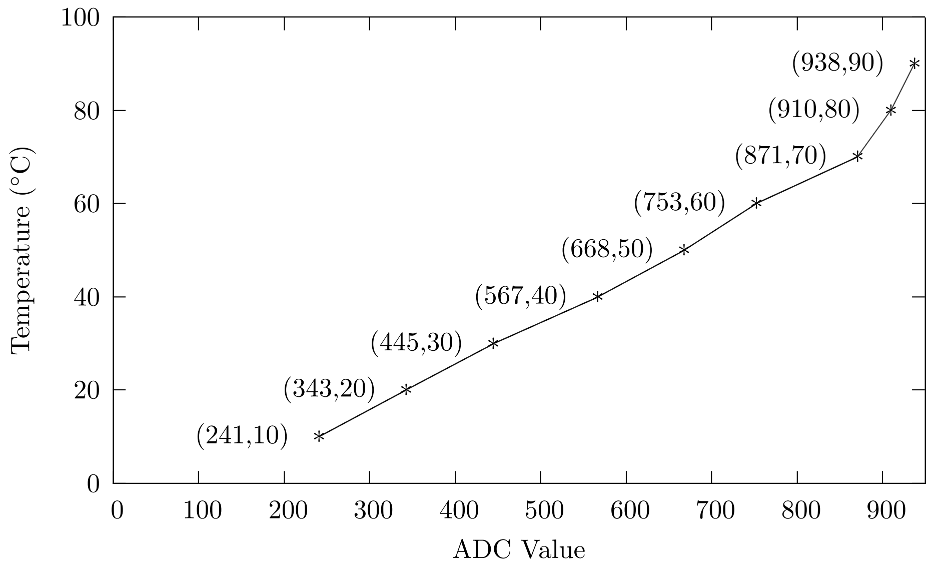

Figure 1.6: Sample ADC value to Temperature Linear Interpolation

Figure 1.6 shows an example of a linear interpolation using ADC values and their temperatures. The lines between each data point represent the functions to be used for linear interpolation. Here is an example of computing the temperature for the ADC value 400.

To perform a linear interpolation for the ADC value 400 one would find the two nearest ADC values to 400 in the table ($(343, 20)$ and $(445, 30)$) and create the linear function between the two. Substituting 400 for x gives the corresponding temperature value (25.59°C).

$$\begin{aligned} f(x) &= \frac{(30-20)}{(445-343)}(x-343)+20 \\ \therefore & \quad f(400) \approx 25.59^\circ\text{C} \end{aligned}$$

A generic equation would be:

$$f(x) = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1$$

where $(x_{1,2}, y_{1,2})$ are two nearest points in the table surrounding a given ADC value x_n. Each line in Figure 1.6 shows the linear interpolation for the corresponding table points[1].

[1]Note: there are no interpolation lines past the end points of the table ($(241, 10)$ and $(938, 90)$). Interpolation beyond these points can be made by continuing their previous lines, however this could result in under/over approximation which can be dangerous in some applications.

Chapter 2

Hardware Setup

2.1 Overview

Constructing a thermistor circuit is a simple process requiring very few parts. The parts used in this report are:

- 7 150 ohm resistors
- 2 100nF capacitors
- 1 10uH inductor
- The Dragon programmer
- An ATmega328p microcontroller
- A two digit seven-segment display
- A Thermistor and a resistor in parallel

2.2 Programmer

To program a microcontroller, a programmer will be needed. The programmer used in this report will be the Dragon. Setting up the Dragon requires a USB connection to the computer where the program will be uploaded from and wiring the programmer pins to microcontroller pins as shown in Figure 2.1.

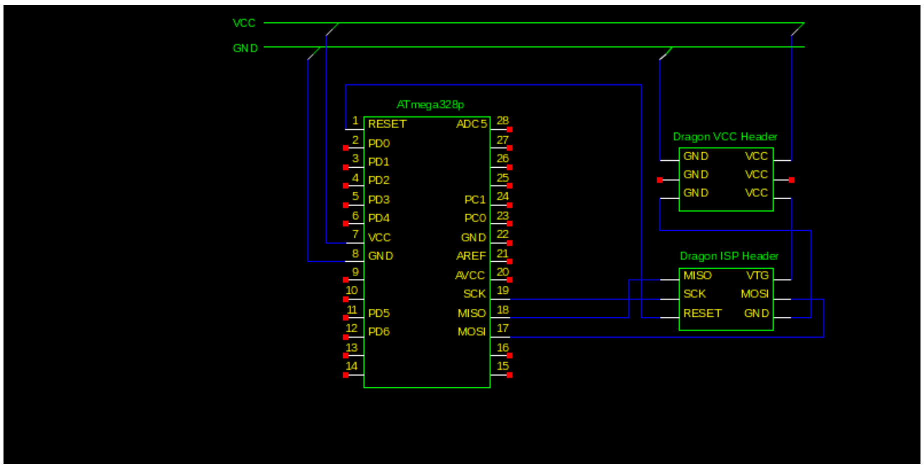

Figure 2.1: Dragon Header Wiring

2.3 Seven-Segment Display

2.3.1 General I/O

When a pin on the ATmega328p is set high, it will supply 5 volts on that pin. To power an LED as common anode, the forward voltage drop and the operating current must be known. The value of the resistor in series with the LED then becomes:

$$(5 - V_f)/I_f$$

The seven-segment display consistes of seven of these circuits per digit. That will be two sets of seven in common anode. In common anode, all seven segments share a common 5V and setting the pin that matches the segment low will turn the LED on, sinking current into the ATmega.

2.3.2 Mapping

Having the display show a number requires powering the segments that correspond to that digit. The pin layout for the seven-segment display used in this report is shown in Figure 2.2. So to display the number 4 on the left digit, pin 4 must be high for common anode and pins 14, 13, 16, and 1 must be low while the rest are high.

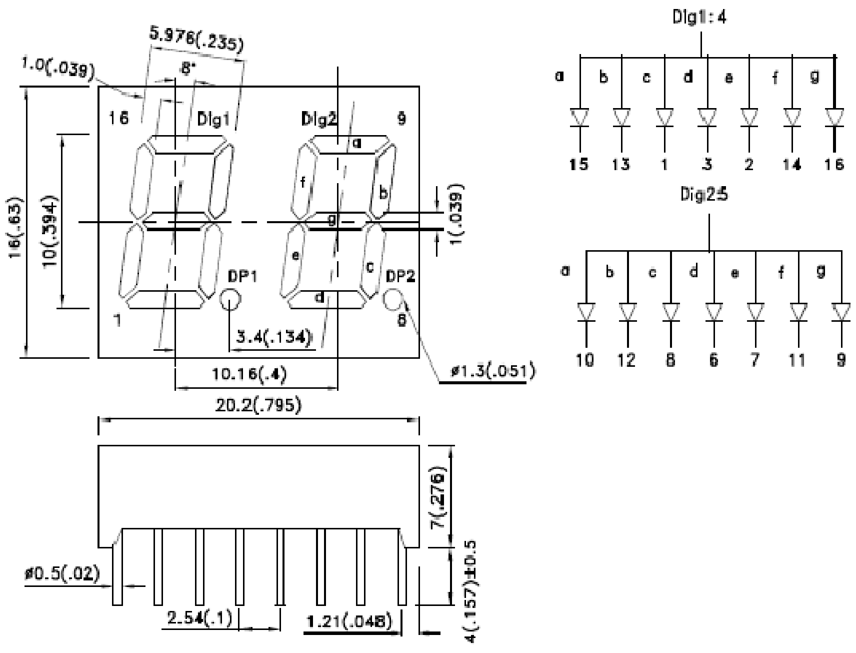

Figure 2.2: Seven-segment Display Pin Layout

2.3.3 Digit Pulsing

Controlling 14 LEDs with the ATmega would require 14 pins if every pin would control a single segment for both digits, but by alternating which digit is powered after every few clock cycles, the number of pins can be reduced to 9 (7 for the segments, and 2 for turning the digits on/off via the common anode pins). Because of the speed at which the ATmega alternates the digits, both digits will appear to be lit simultaneously. The new circuit is shown in Figure 2.3.

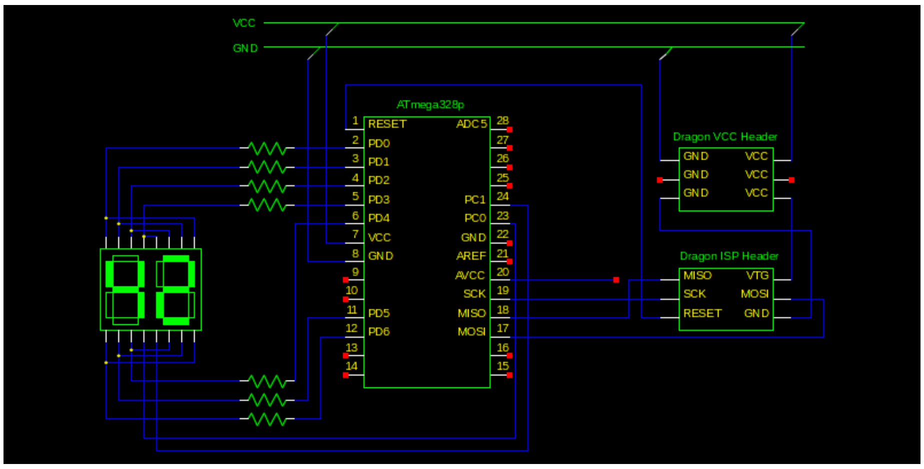

Figure 2.3: Both Digits Wired in Parallel

2.4 ADC

The ADC module of the ATmega is powered by three seperate pins (20, 21, 22). The circuit for these pins is shown in Figure 2.4. This setup connects the reference Voltage externally to VCC while also reducing noise that may interfere with the measurements.

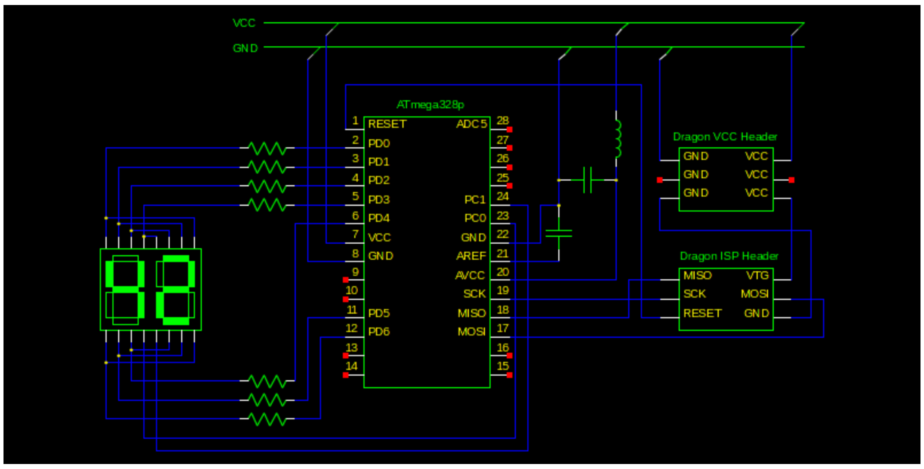

Figure 2.4: ADC Power Added

2.5 Thermistor

The ADC module reads from a pin determined in the program. In this report ADC5 (pin 28) is used with the thermistor setup shown in Figure 2.5. The ADC measures the varying voltage drop across the thermistor and the program will then match that to its corresponding temperature.

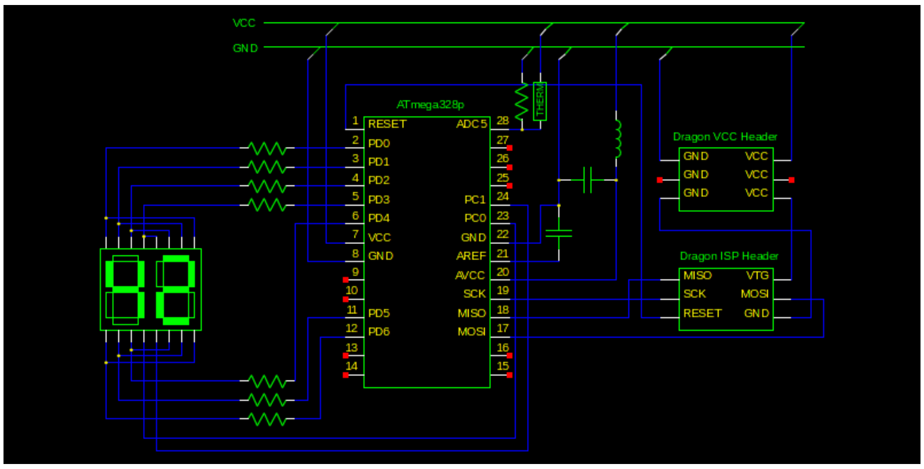

Figure 2.5: Completed Thermistor Circuit