Temperature Sensor

Benjamin Palay: 1815593

School of Electrical and Information Engineering, University of the Witwatersrand

ELEN2021/2006 – Microprocessors

8 June 2020

Abstract

This report discusses and analyses the design and execution of the reading and measuring of the temperature by an MCP9700 temperature sensor and subsequent display of the temperature in degrees Celsius on two seven segment displays. The circuit for this is constructed on a single breadboard and is linked to an Arduino Uno with an ATMEGA328P microcontroller at its centre. The code for this programme is written in AVR assembly language and the platform used is Atmel 7. Every second, the environmental temperature is read as an analogue signal and is converted to a 10-digit binary number. This 10-digit binary number corresponds to a two-digit temperature which is then outputted to the two seven segment displays, whereby each segment is connected to a different port on the Arduino. The system can measure between 0°C and 40°C and the design successfully outputs the temperature accurate to the nearest unit with an error margin of 4°C.

1. Introduction

The objective of this project is to construct a circuit that measures the temperature in degrees Celsius and outputs it to two seven segment displays. The sensor used is the MCP9700 which varies its voltage based on the environmental temperature [1]. Every degree increases the voltage linearly by 10 mV, beginning at 500 mV corresponding to 0 degrees centigrade [1]. Once the temperature is outputted as a voltage, the voltage, which is analogue, is converted to a binary number using the ADC (analogue to digital conversion) technique in AVR assembly. Based on this binary number, a temperature is outputted to two BS-C506RD seven segment displays. These displays function by each LED segment being connected to a voltage source, namely a port on the Arduino Uno, and glowing red when a high is outputted to it [2]. Although the sensor used has a range of -40°C to 150°C [1], this project only accounts for a range of 0°C to 40°C.

The success criteria for this project include reading an accurate temperature and displaying it on two seven segment displays, as well as gaining skills in engineering, logical thinking, and time management. Some of the constraints include coding exclusively in AVR assembly language, the specific components used and their limitations as will be discussed, as well as a time frame in which to complete the project.

In this report, the design of the circuit and code is presented in section 2, followed by a brief section (section 3) showing the results. Section 4, called 'analysis and discussion', discusses the environmental and social impact of this project, as well as future recommendations for how the project can be improved. There will also be a critical reflection within this section. The conclusion is outlined in section 5, followed by references and appendices with photographs and figures.

2. Design

The circuit set up is shown in figure 1 below. The connection from each segment and from the sensor are connected to labels representing ports on the ATMEGA 328P and its Arduino base [3], instead of connecting each wire directly to the Arduino. This was done for clarity. Furthermore, only the relevant ports were indicated. The method used to connect the sensor segment of the circuit with capacitors and the Arduino was learnt in part from an online tutorial [4].

Photographs of the circuit are displayed in Appendix A on page 4 as figures 2 and 3.

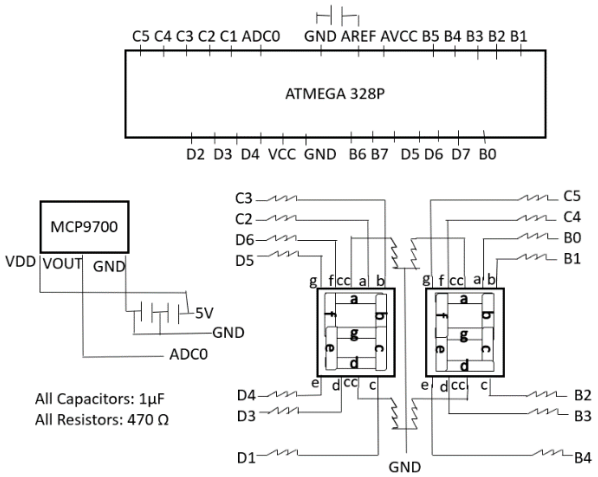

Figure 1: Circuit design sketch.

Each LED segment is connected in series with a current limiting resistor of 470 Ohms to reduce the current being drawn from the Arduino. Each segment, without resistance, draws a maximum of 30 mA of current [2]. The Arduino outputs a high of 5V (but actually outputs less than this – the typical voltage is actually around 1.7V for the LED segments and has a maximum of 2.5V for each segment [2] -) and each resistor is rated at 470 Ohms.

$$I = V/R \quad (1)$$

Ohm's Law as shown by equation (1) above indicates that the current being drawn from each segment is approximately 3.6 mA (1.7V/470 Ohms). An extra resistor is also placed at each common anode and cathode of the displays, further reducing the current. The maximum current that can be drawn from the I/O ports of the Arduino Uno is 200 mA [5]. Here, 14 ports are being used to power the displays, drawing an absolute maximum of 74.2 mA (2.5V/470 Ohms = 5.3 mA multiplied by 14 ports = 74.2 mA). This will usually be far less as not all the ports are set high at once. Multiplexing the displays together using transistors and fewer ports was considered in order to reduce the current drawn [6]. However, due to the large amounts of code and cycles that would be needed led to the decision to use 14 ports instead, seeing as this could be done while drawing a current far below the maximum ratings of the Arduino.

Two capacitors, each rated at 1 microfarad, are connected in series from the temperature sensor's Ground [1] to the 5 Volts port of the Arduino to limit noise [7]. The middle pin of the sensor is the Voltage output pin and is connected to the ADC0 port, in order to convert the voltage to a digital binary number.

The programme is written in AVR assembly for an ATMEGA328P microcontroller. A flow chart of this code and logic is shown in Appendix B on page 4 as figure 4. Within the initialization of the code, timer 1 is set up so that a timer interrupt occurs every second - when the count reaches the value in the output compare register A [5]. This was done because realistically, the temperature will not change drastically in a time frame shorter than 1 second. Therefore, there is a mostly continuous reading of the temperature while saving power in between readings. Also, in the initialization, all the ports barring the port for ADC0 are set to be able to write to, so that an LED segment lights up when a high is written from the port to which it is connected. Port ADC0 is used to read in a voltage from the sensor. Furthermore, analogue to digital conversion is initialized. Originally, it was set up in free running mode [5], in which it would constantly be reading in and converting new voltages. However, considering that realistically the temperature would not change that drastically and that quickly, power conservation was favoured. The ADC is therefore set up in single conversion mode, so that a conversion is done only when the ADSC bit is set in the ADCSRA register [5]. Additionally, the ADC0D bit in the DIDR0 register is set in order to disable digital input on port C0, thereby saving power [5]. Within the timer Interrupt service routine, a single conversion is initialized. This means that every second, a new conversion is set up and displayed a maximum of 30 cycles later. The main portion of the code simply has a sleep command in a loop. The sleep command is set up in ADC noise reduction mode in order to limit the noise as well as to reduce the power consumption [5]. Between timer interrupts, therefore, unnecessary cycles and power consumption are reduced. Moreover, there are two possibilities for reading the conversion to a 10-digit binary number [5]. It can be left adjusted to read from the ADCH register or right adjusted to read from the ADCL register. In order to save time and power, a realistic range of 0°C to 40°C is accounted for. This range only uses 8 bits as opposed to 10 bits using equation (2) [8]

$$(\text{Voltage}*5/1024 - 0.5)*100 \quad (2)$$

to obtain a binary number that, within this range, never exceeds 0b11111111, which corresponds to 75 degrees. Therefore, only the ADCL register is used when reading the conversion to binary as it is large enough for the given range.

During the initialization of the interrupt vector table at the start of the programme, a 'jumptable' is set up. This 'jumptable' begins at 0x0368**.** The location 0x03 is chosen arbitrarily within the programme memory, however the location 0x68 is chosen to correspond to 0°C using equation (2). Each subsequent location moving upwards corresponds to a different jump, whereby each jump goes to a piece of code that outputs a different number to the displays. While the data memory could have been used, the programme memory was chosen because it is non-volatile[5], and therefore the contents of the table are not lost each time the Arduino is powered off.

After the ADC conversion completes, taking a maximum of 25 cycles, or 0.02 microseconds (the ADC was set up to run at 125 KHz), an interrupt is triggered. Within this interrupt service routine, the Zh register (R31) is loaded with the high part of the 'jumptable' (0x03). The converted binary number, which is stored in the ADCL register, is moved to the Zl register (R30) so that the command 'icall' jumps to the location 0x30yy, where yy is the hexadecimal number corresponding to the binary number held in ADCL [9]. For example, if the number 0x77 is moved to Zl, the location 0x0377 is called, which then jumps to code that outputs the number 8 to the seven segment displays. While the original set up of the

'jumptable' takes considerable time, this use of indirect addressing is used as it is efficient as well as easily read and understood code.

3. Results

The temperature is successfully displayed on the two seven segment displays with an error margin of ±4°C, in line with the error margin illustrated in the datasheet [5]. The design was tested extensively in a range of different temperatures, using the air conditioning and heating system *SIRAIR* model C5-51V3A-W87AE2J. To test the sensor and display, the sensor is held in front of the heater for up to 20 seconds. The accuracy obtained is within 4 °C.

4. Analysis and Discussion

There are many fields of impact that this project can affect. These include environmental and social impacts. Environmentally, current limiting resistors are used to prevent ignition. All wires are insulated with plastic for safety when using the device. Socially, this type of sensor is used in battery packs, home appliances and entertainment systems [1]. Furthermore, temperature sensors are being used extensively in the current COVID-19 climate. This application could potentially be modified to be used in this field.

The design and implementation of this sensor can be improved in several ways. To measure more accurately, several readings could be taken and then averaged. Due to noise, one reading alone can be easily affected. Moreover, an improvement could be reducing power consumption of the Arduino by multiplexing the two displays together and alternating the high outputs to each segment so that it still looks to the human eye like one continuous high output, but it is really only outputting a high to a few ports at one time[6]. Furthermore, the range could be increased to output numbers from -40 degrees to 150 degrees, which would include adding a third display using the multiplexing method above. This third display could further be used to display a more accurate number.

The results obtained reach the success criteria as outlined in section 1. An accurate temperature is measured and displayed on two seven segment displays. Skills in engineering, logical thinking and time management were gained. In order to arrive at the final product, many steps had to be thought out and tested. Each step had to logically follow the previous step, and often some steps did not work. For example, a pushbutton was originally planned to be used to trigger an interrupt and set up the ADC, however there were many problems faced in this setup, including a debounce and inaccurate readings. There were many other smaller bugs and wrong assumptions. These made the project take up a far larger time frame than was originally planned. Therefore, time management skills were gained in that extra time must be allocated for these types of tasks. I originally coded the project in the language C and in the Arduino IDE, which was not so difficult for me. The difficulty was learning how to translate this logically into assembly language, and to navigate through online sources and datasheets. The time put into this has allowed invaluable engineering and thinking skills to be gained. Other parts that went well include setting up timers and interrupts, using indirect addressing effectively, and gaining a much better understanding of microcontrollers and CPUs.

5. Conclusion

The design of a temperature sensor to measure the temperature in degrees Celsius and then to display this on two seven segment displays was successful. Temperatures between 0°C and 40°C were accurately displayed every second to the nearest unit, with an error margin of ±4°C.

6. References

[1] MCP9700/9700A/MCP9701/9701A – Low Power Linear Active Thermistor ICs – Datasheet. https://www.mouser.com/datasheet/2/268/20001942F-461622.pdf, Last accessed 1 June 2020.

[2] Single Digit LED Displays 7-Segment, ± 1 Overflow – Datasheet. https://www.digchip.com/datasheets/parts/datasheet/036/BS-C506RD.php, Last accessed 1 June 2020.

[3] Components101.com. 2018. Arduino Uno Pin Diagram, Specifications, Pin Configuration & Programming. https://components101.com/microcontrollers/arduino-uno, Last accessed 1 June 2020.

[4] Arduino Serial Thermometer Circuit And Sketch. https://startingelectronics.org/beginners/start-electronics-now/tut15-arduino-serial-thermometer, Last accessed 1 June 2020.

[5] ATmega48A/PA/88A/PA/168A/PA/328/P – Data Sheet. https://witse.wits.ac.za/access/content/group/ELEN2021-2020SM1/AssemblyandArchitecture/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061A, Last accessed 1 June 2020.

[6] Margolis, M., Jepsin, B. and Weldin, N., 2020. *Arduino Cookbook: Recipes To Begin, Expand, And Enhance Your Projects*. O'Reilly Media, Incorporated.

[7] Filter Out Unwanted Electrical Noise With Capacitors. https://www.arrow.com/en/research-and-events/articles/using-capacitors-to-filter-electrical-noise, Last accessed 1 June 2020.

[8] How To Measure Temperature With Arduino And MCP9700 – Starter Kit. https://starter-kit.nettigo.eu/2010/how-to-measure-temperature-with-arduino-and-mcp9700/, Last accessed 1 June 2020.

[9] AVR Instruction Set Manual. https://wits-e.wits.ac.za/access/content/group/ELEN2021-2020-SM1/AssemblyandArchitecture/atmel-0856-avr-instruction-set-manual.pdf, Last accessed 1 June 2020.

Appendices

Appendix A: Photographs of the circuit.

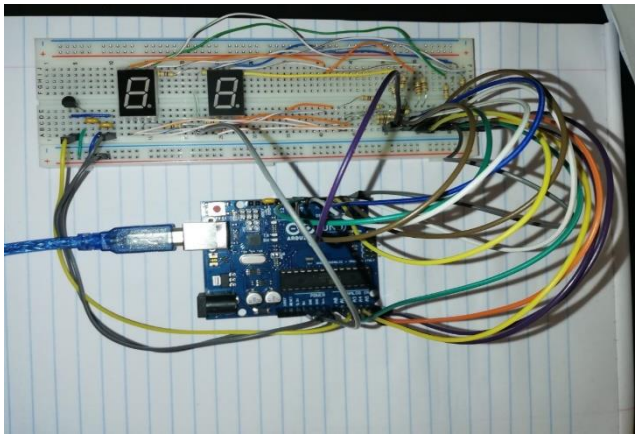

Figure 2: Photograph of the circuit from above.

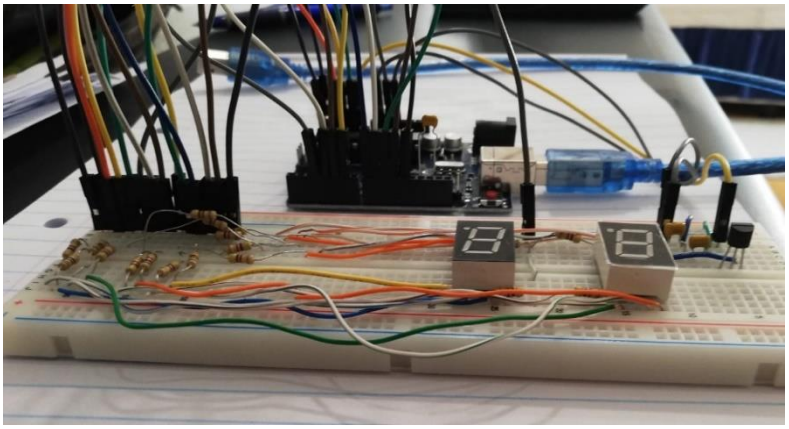

Figure 3: Photograph of the circuit from the side.

Appendix B: Flow diagram and declaration.

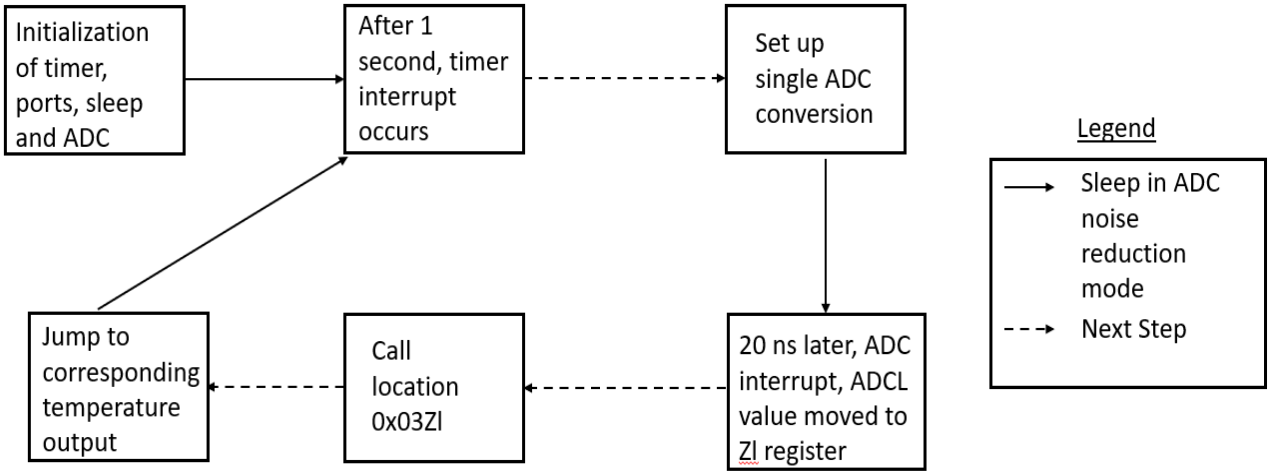

Figure 4: Flow diagram of the programme in assembly

I, Benjamin Palay, declare that this project is my own work