



Dominic C. L. McKean

Temperature Transducer to ADC Converter

Investigation, Design, Simulation, Build and Test

Module: 6EJ517

Electronic Devices and Systems

Loughborough College

University of Derby

April 2017

Contents

[Section 1: Design and Simulation of an Analogue to Digital Converter 3](#_Toc481184230)

[Bidirectional Counter 5](#_Toc481184231)

[Successive Approximation Register 6](#_Toc481184232)

[Digital to Analogue Converter 7](#_Toc481184233)

[Simulation Testing 7](#_Toc481184234)

[Section 2: Harware Design of Thermostate With Digital Display Using Arduino Microcontoller 10](#_Toc481184235)

[Arduino Circuit Design 10](#_Toc481184236)

[Hardware Setup Using Simulink® Interface 12](#_Toc481184237)

[Interfacing via Simulink® ‘S’ function 14](#_Toc481184238)

[Atmel ATMEGA328P-PU Internal Temperature Sensor 15](#_Toc481184239)

[MAX6675 Thermocouple Amplifier 16](#_Toc481184240)

[Cold-Junction Compensation 17](#_Toc481184241)

[New K-type Thermocouple 17](#_Toc481184242)

[The Arduino Powered Thermostat 18](#_Toc481184243)

[Costing 19](#_Toc481184244)

[Ideas for Further Work 19](#_Toc481184245)

[References 20](#_Toc481184246)

[Appendix 23](#_Toc481184247)

[Appendix i LM35 & LCD Code Developed in Circuits.IO 23](#_Toc481184248)

[Appendix ii ATMEGA328P Internal Temperature Sensor Sketch 25](#_Toc481184249)

[Appendix iii MAX6675 Thermocouple Code 25](#_Toc481184250)

[Appendix iii 27](#_Toc481184251)

[Appendix iv Final Code for Physical Arduino + MAX6675 + LCD Display + Degree Symbol + Push Buttons + LEDs 27](#_Toc481184252)

[Appendix v Initial Circuits.io Code + MAX6675 Setup Flowchart 30](#_Toc481184253)

[Appendix vi Initial Circuits.io Code Loop Flowchart 30](#_Toc481184254)

# Section 1: Design and Simulation of an Analogue to Digital Converter

“Analog-to-digital converters are the link between the real world and our digital approximation of it” (Yates, 2014)

This report will focus on the Arduino uno microcontroller and how it interprets real world analogue signals to produce a digital approximation. The Arduino Uno uses a microcontroller chip namely the ATmega328p. On board this chip along with just about every other piece of electronic equipment there is an analogue to digital converter (ADC). An ADC takes real world analogue signals such as a varying temperature or pressure and converts it into a digital approximation which is just good enough for a CPU of some electronic equipment such as a digital thermometer or pressure sensor to understand and manipulate. The analogue input signal is checked or ‘sampled’ at a set time or ‘frequency’ to see how the signal has changed during that time. The ADC only has a finite amount of room or ‘bits’ in which to store the infinitely varying input signal data and so a sacrifice has to be made. A change to the sampled signal can only take place if the analogue signal moves past a set voltage level or step. The number of steps that are available is where the number of bits, a 10bit word will represent 1 of 1024 possible steps or voltage values. In the case of the ATmega328p the 1024 steps are divided by the reference voltage of 5 volts giving approximately 4.88mV per division. Due to the fact that this process is only an approximation of the analogue input signal, the quantisation by the ADC conversion produces an error, called "quantisation error". (AVR, nd) This error can be minimised by increasing the number of discrete steps used to sample the analogue signal but also by increasing the sampling rate.

The sampling rate is how fast the ADC is able to look at the value of the analogue signal. The ATmega by default is set at 9615 samples per second, which is about 104μs per sample, however it is possible to adjust the sampling rate by adjusting the ADC ‘Prescaler’. The prescaler is part of the ADC circuit that reduces the ATmega’s clock speed from 16MHz down to between 50 – 200 kHz for optimal resolution of the ADC’s 10bits. By default the prescaler uses a reduction factor of 128; 16MHz/128 = 125 kHz. In order to get to our sample rate 9.615ksps however we need to understand how this particular ADC of the MTmega works. The MTmega uses a ‘Successive Approximation Register’ (SAR) ADC, sometimes just SA\_ADC. Successive approximation can be thought of as a similar approach to looking for a word in a dictionary. If you are looking for the word ‘mouse’ for example you wouldn’t start at page 1 and check every page until you eventually found mouse. You would most likely open the book somewhere in the middle and see where you are relative to the letter ‘M’. You would then concentrate on the one half of the book in which you know contains the Ms and split that half in half again, continuing this approach until you zero in on the correct word. This is basically how successive approximation works. Using a 10 bit binary counter which counts from 0 – 1023, a comparator checks successively to see if the incoming analogue signal is higher or lower than the most significant bit (MSB) which is set to 1 which given a 5Volt reference = 2.5Volts, in binary this is 1000000000. If the analogue signal is higher than 2.5V; the MSB value is kept in the SAR, else on the next clock pulse the MSB is reset to 0-low and the SAR moves on to the next significant bit setting it to 1 giving; 0100000000 and the input signal is compared to. This continues until all bits have been checked. It has thus taken 10 clock cycles to check every ‘bit’, however; there are overheads to be considered supposedly for switching the direction of the up/down counter, outputting the result of the register, resetting the register etc. In total there are 13 clock cycles per sample and so given an ADC clock frequency of 125 kHz/13 = 9615 samples/ second. However, as mentioned previously, this can be improved by the Pre-scale factor at the expense of ‘bits’, it is a matter of what type of signal you are trying to capture.

This is not the whole story. In order for the vale from the up/down counter to be fed back into the comparator, it needs to be converted from a digital signal back into an analogue signal. This is done by an internal digital to analogue converter (DAC) usually of the capacitive type but that level of detail is not necessary for this report. The output of the comparator tells the up/down counter when to switch from counting up to counting down. This change in direction is also initiated if the counter reaches the max or min binary value.

(Hedayati, 2011) (EETech Media, LLC, ND) (Robot Platform, 2017) (Vandenberg, 2014) (AVR, nd) (Yates, 2014) (Russell & Thornton, nd) (Atmel, 2015)

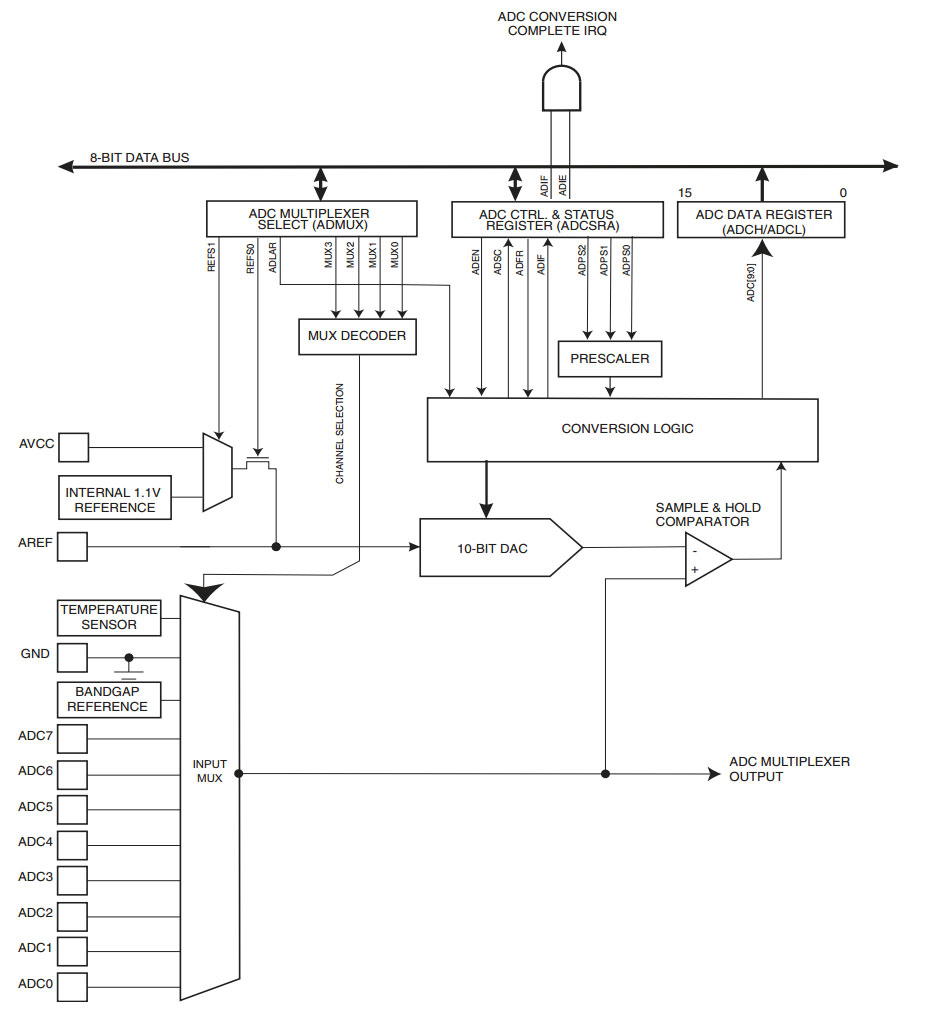


Figure , Block diagram of the ATmega SA\_ADC, (Atmel, 2015)

The block diagram above shows the ATmega SA\_ADC used by the Ardunio. This will serve as the base for our simplified Matlab®/Simulink® model. The main parts we are interested in are the comparator, the internal DAC and the successive approximation conversion logic. It’s worth mentioning that the conversion logic could have been realised using a state machine or state diagram within the Simulink model, however it was thought that a more physical depiction of the system would be more intuitive.

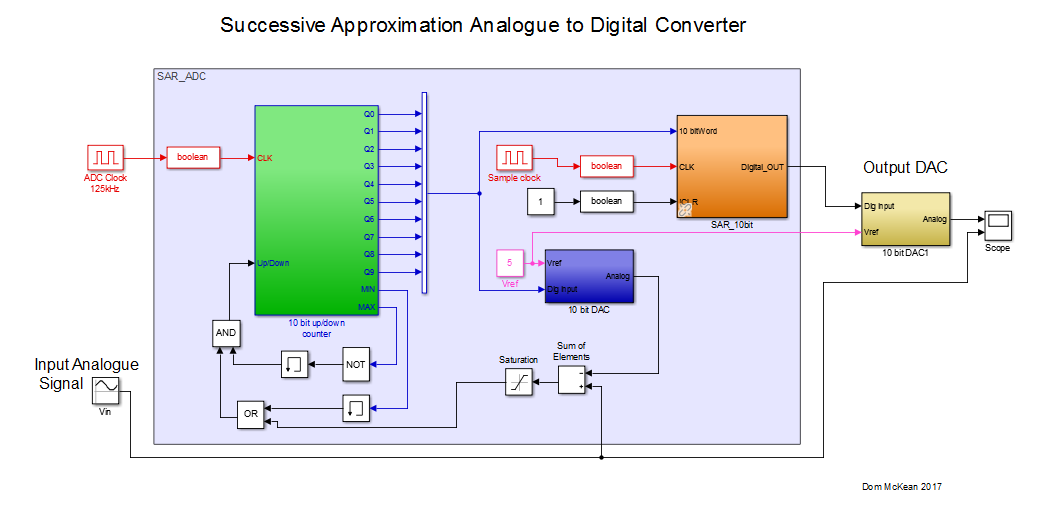


Figure , Successive Approximation Analogue to Digital converter.

The above figure shows a simplified Simulink model of the ATmega’s SA\_ADC. What can be seen is the sine wave input in black at the bottom left going into a makeshift comparator made from a summing and saturation block, which is comparing the input against the output from the internal DAC (Blue block). The output from the comparator is fed into the up/down counter (Green block) which triggers a change in counting direction and also tells the SAR (Orange block) to store the current value from the counter and reset the register. The output form the counter is also fed into the internal DAC along with Vref (Pink) to complete the cycle. Again, there is also logic between the comparator and the up/down counter for the counter to reset itself if it reaches its max or min value.

## Bidirectional Counter

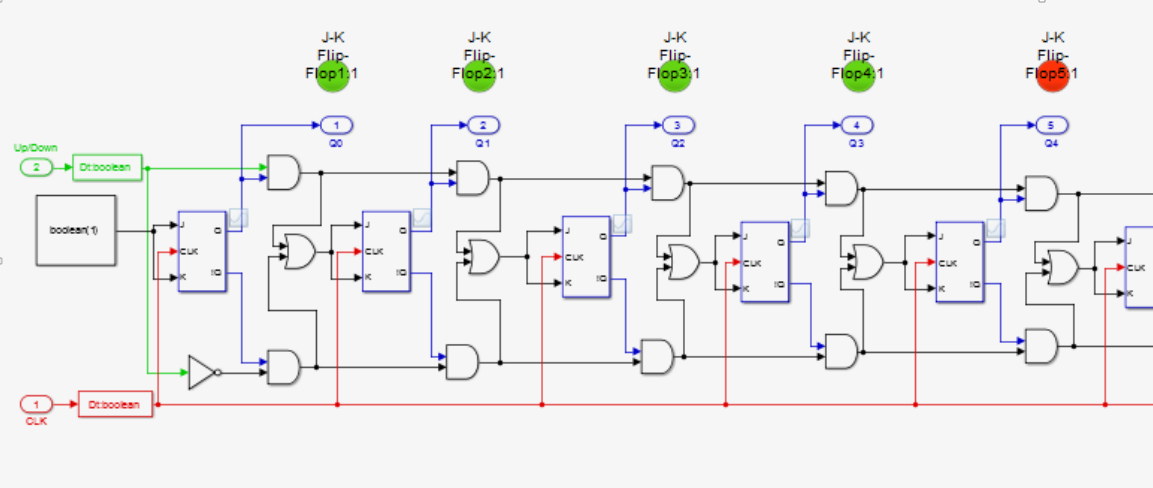


Figure , Up/down counter. Control and click to follow link.

In the figure above, JK flip-flops and logic gates have been used to build a bidirectional counter. 10 flip-flops have been used to mimic the ATmega’s ADC counter on the Arduino. LEDs have also been added to each flip-flop to verify the binary output during the design process. In order to maintain clarity, only five of the ten flip-flops are shown. The clock input can be seen in red at the bottom which is connected to all flip-flops and the signal from the comparator is in green at the top. For counting upwards, all the flip-flops are able to toggle if the preceding flip-flops are high AND the up/down signal is high OR if the preceding flip-flops are all low AND the up/down signal is also low in the case of counting backwards. The first flip-flip toggles on every clock pulse, the second can only toggle if the first flip-flop is high, the third can only toggle is the previous two are high and the fourth only if the previous three are high. This is successive binary counting. The up/down counter will continue to count up or down until there is a change in the up/down command signal from the comparator or due to the counter reaching the end of its limits. These limits are fed into the same command signal as the comparator. The outputs of the last two AND gates provide the counting limits.

(AspenCore, Inc, 2017) (EETech Media, LLC, nd)

## Successive Approximation Register

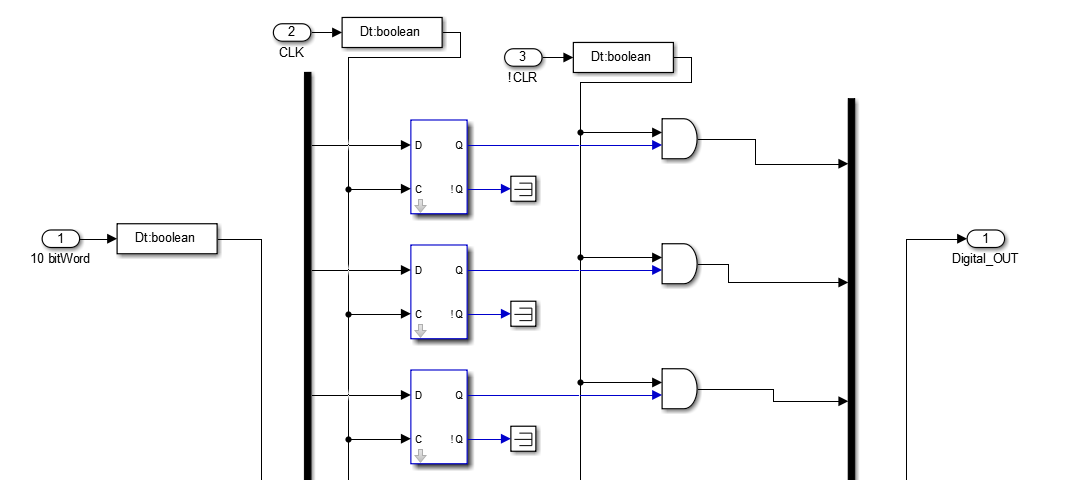


Figure , Successive approximation register latch

The above figure shows the SAR latch assembly. Only three of the ten D flip-flops are shown for clarity. The digital output from the up/down counter is fed via multiplexers into the SA register. D type flip-flops have been used to build this SAR model as they will hold the counter values at D until the next clock pulse when all the data is passed through to the digital output.

(Stewart, 2004) (EETech Media, LLC, nd)

## Digital to Analogue Converter

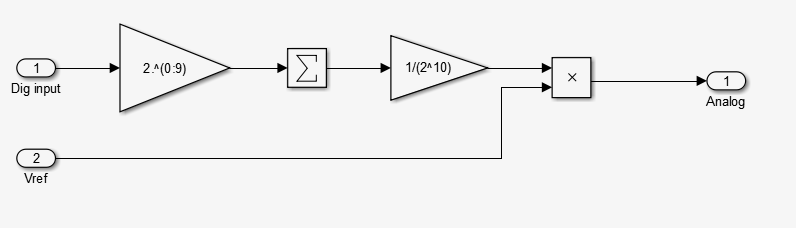


Figure , Internal DAC algorithm

The figure above shows the internal DAC algorithm for converting the binary output of the up/down counter into an analogue signal for comparison with the analogue input signal at the comparator. The Dig input from the SAR is a one dimensional vector with 10 elements, . These elements are summed and multiplied by the reciprocal of total number of steps, . This is then multiplied by the reference voltage (5V) and outputted to the comparator. The same DAC is used on the output of the SA\_ADC in order for us to use a scope and visualise the results of the system.

## Simulation Testing

What follows is a collection of output graphs at various input and sampling frequencies.

In keeping with the Arduino, the ADC counter clock frequency will be 125kHz. This has been pre-scaled down from the 16MHz clock, and since it takes 13 ADC clock cycles to perform one conversion 125000/13 = 9600 samples per second, which is also the serial communications speed between the Arduino and the PC if you are using one. However, at this frequency the two waveforms, the original sine wave and the digitised sine wave are coincident, and so in order to visualise the circuit and understand how different sampling frequencies affect the outcome a drastically reduced sampling frequency will be used.

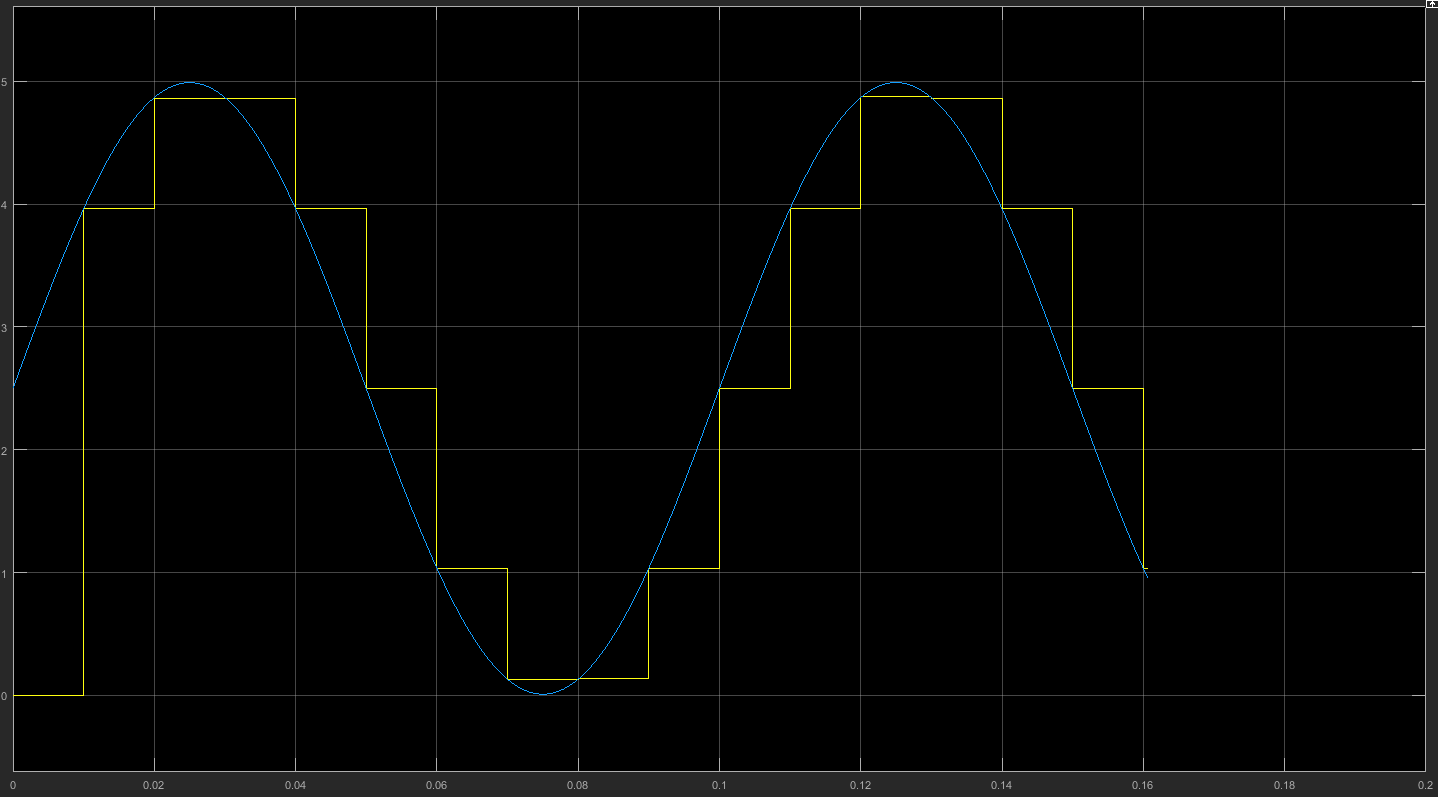


Figure 100Hz sampling for a 10Hz input signal.

The chart above shows a 10Hz analogue input signal being sampled at 100Hz by the simulated SA\_ADC. This is 10X higher than the input signal which is considered to be the minimum sampling frequency to capture the input signal. To confirm the sampling frequency, the first pulse is at 0.01s, feq = 1/period. 1/0.01= 100Hz. This sampling rate does not give a great representation of the input signal and so next we will increase the sampling rate to 500Hz.

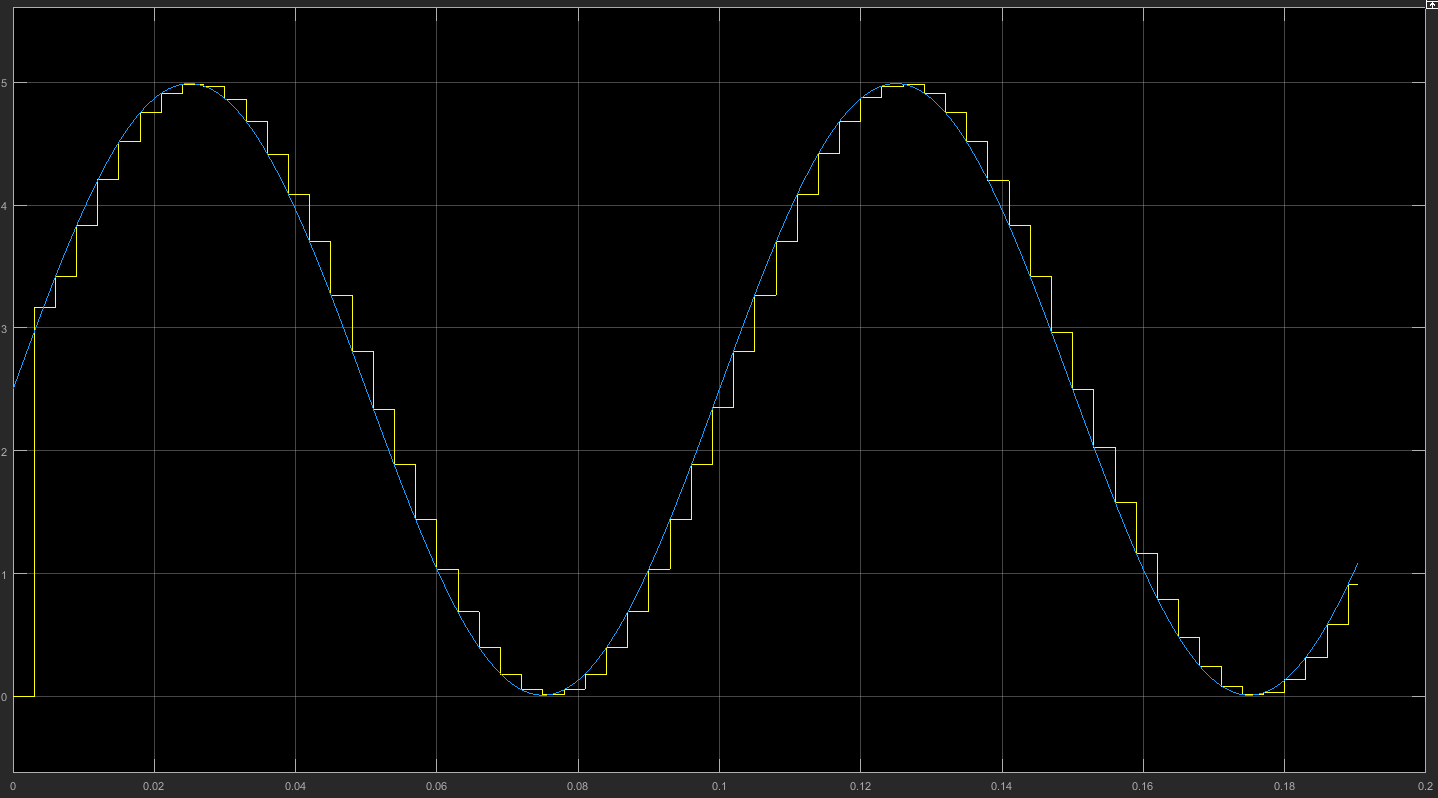


Figure 500Hz sample of 10Hz input signal

The above scope image shows the same 10Hz signal being sampled at 500Hz. The number of discrete steps has greatly improved giving a much better representation of the input signal. There are now 16 steps for a slope as opposed to the previous 4.



Figure 10kHz sampling for a 10Hz input

The sampling frequency is now 1000x faster and now looks good enough for use as a digital display on an oscilloscope for example. The two signals are now coincidental.

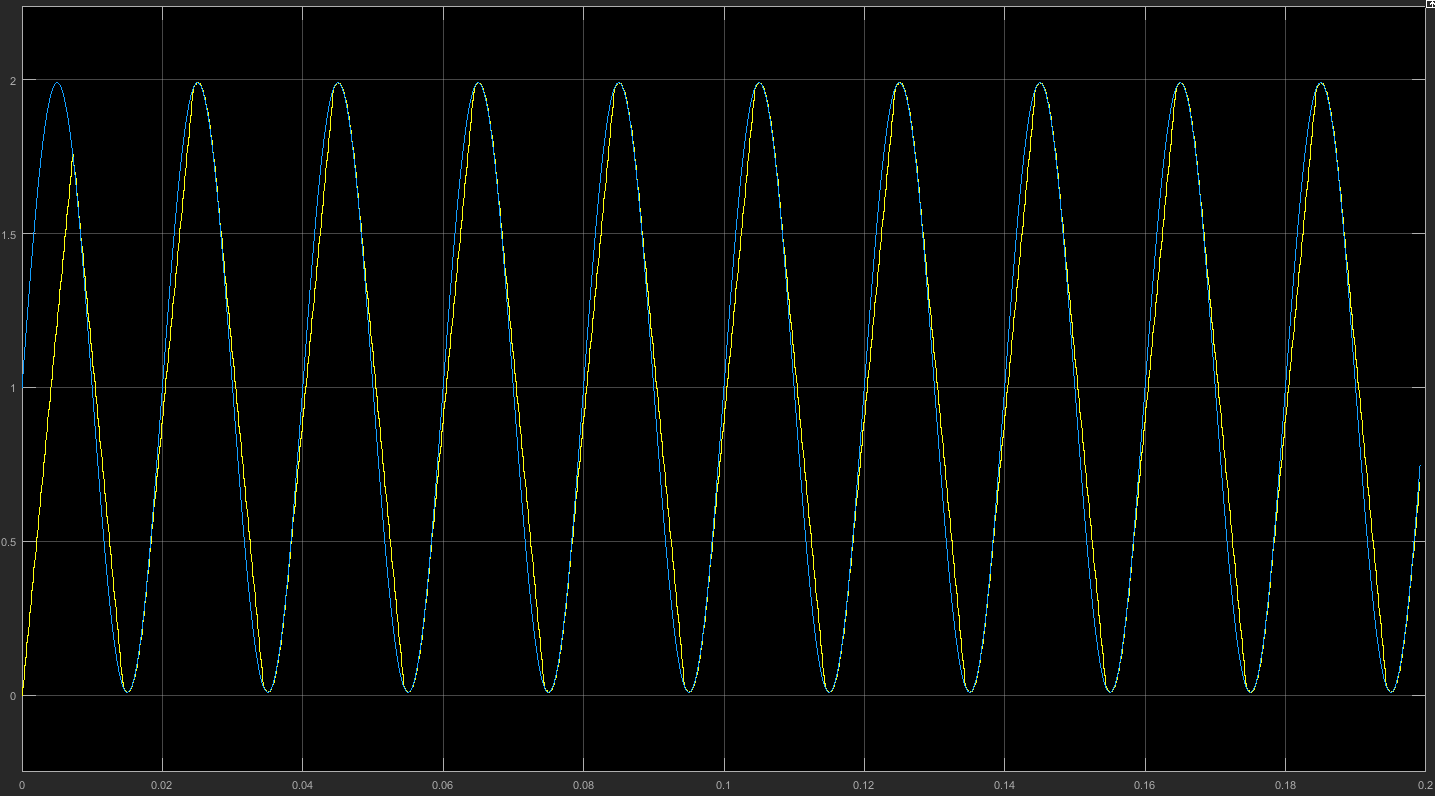


Figure , 9600Hz sampling for a 50Hz input signal

The figure above shows a 50Hz input signal being sampled at the proper 9600Hz. It can be seen that there is a small amount of error occurring on the rise and fall of the digital signal. This should be easily sampleable by the Arduino and is thought to be an computation error to the host PC running at max capability.

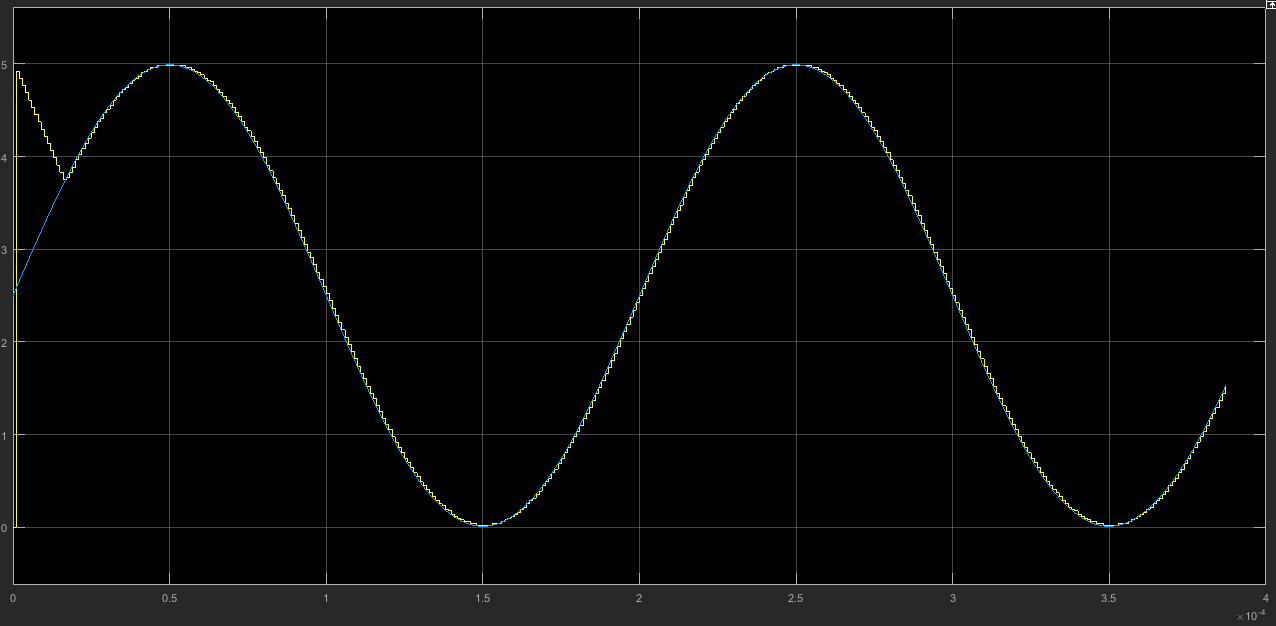


Figure , 5kHz input sampled at 1MHz with ADC counter CLK at 16MHz.

The figure above shows how 5KHz input signal was successfully sampled with an over clocked converter running at 1MHz. This is getting a little closer to the expected behaviour described in literature, (Yates, 2014) (Atmel, 2015) albeit with different parameters. Also this was only possible with the ADC counter clock running at 16MHz. It is suspected that maybe the up/down counter actually runs off the main 16MHz clock and not the pre-scaled ADC clock, although there is no literature to back this up. It is more likely that there is either a major error in the Simulink design or it is just not capable of running at ‘real time’ on a Windows OS. Using an ad-hoc real time clock did not make any difference. It would be interesting to run the simulation again on a real time OS such as XPC.

# Section 2: Harware Design of Thermostate With Digital Display Using Arduino Microcontoller

In this section a physical working thermostat/temperature sensor with digital display will be designed and built. The hardware for this project will be an Arduino Uno which has been described in the previous section. Two types of temperature sensors will be examined, firstly the LM35 solid state sensor, and then a generic 0.5mm K-type thermocouple plus the accompanying MAX6675 thermocouple amplifier. The temperature will be displayed on a 16:2 LCD display plus a red and green LED to indicate if the ambient temperature is above or below the setpoint. The setpoint will be adjusted via two push button switches. The LEDs could just as easily be MOSFETs switching a cooling and/or heating system.

The main components used are listed below but will be described in more detail in their own section.

LM35 Temperature Sensor

The LM35 is a very common solid state temperature sensor similar in package to a transistor and is used by many electronics hobbyists due to its low price and simple application. LM35 output= 10mV/°C.

K-type Thermocouple

The K-type thermocouple is a very widely used industry standard temperature sensor and come in a variety of sizes for various applications. Within the Powertrains Research Facility at Loughborough University they are widely used for engine monitoring for air, oil, coolant, fuel etc. The thermocouple used in this project will be a 0.5mm diameter variant. This has been deliberately chosen for its small thermal mass and so should transmit small changes in temperature rather quickly.

MAX6675 Thermocouple Amplifier

The thermocouple can only provide very small changes in voltage (41μV/°C) and so this signal needs amplifying, but the main purpose of the amplifier is the cold junction compensation which the amplifier uses to determine the temperature at the hot end of the thermocouple.

Generic 16:2 LCD Display

This display was chosen ahead of a simpler 7segment display because it can display more information at one time, can output words and even pictures as opposed to simple numbers which gives the ability to display messages and/or warning.

## Arduino Circuit Design

The proposed design will first be drawn up and tested on AutoDesk’s ‘Circuits.io’ (McKean, 2017) (AutoDesk, 2017) online software which is very similar to the popular ‘Fritzing’ (Fritzing, 2017) software used for modelling and coding Arduino projects. Below is the proposed design using the LM35. The diodes are preceded by 1KΩ resisters and the push button switches are using 10kΩ resisters. The primary circuit paths for each of the push button switches was later removed when it was found that the circuits were continuously sinking current and flattening the 9V battery used to power the end product. This primary circuit is not really needed, and the 10kΩ resister was moved to be between the switch and the digital input of the arduino as so not to overload the digital input pins. The backlight of the LCD display is protected by a 220Ω resister and the contrast of the display is controlled via the potentiometer. The code for the analogread, LCD display, push buttons, and LED logic was also developed in the Circuits.io software app and can be found in the appendix.

Here is a link to my circuit for you to see it for yourself: <https://circuits.io/circuits/4681962-d-mckean_thermostat_project>

The software also allows you to develop your own printed circuit board (PCB) which you can either build yourself or order through the website.

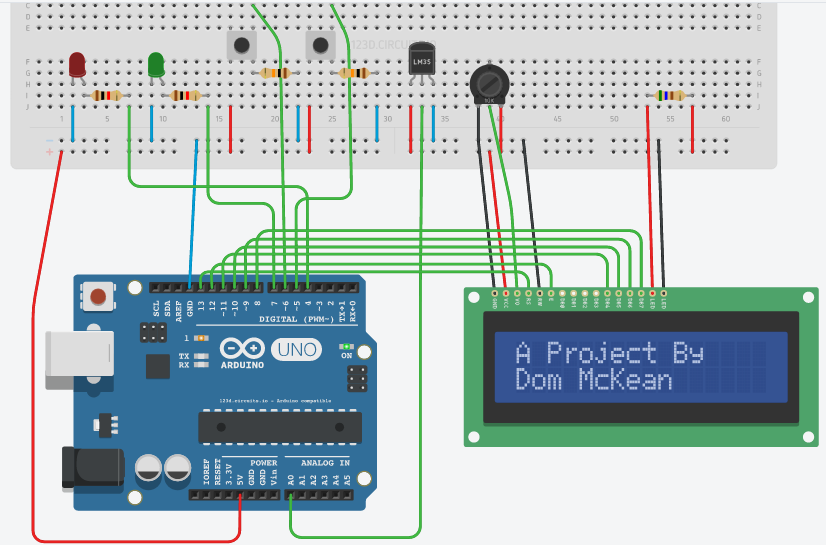


Figure Arduino with LM35 Temperature Circuit. Software by Autodesk

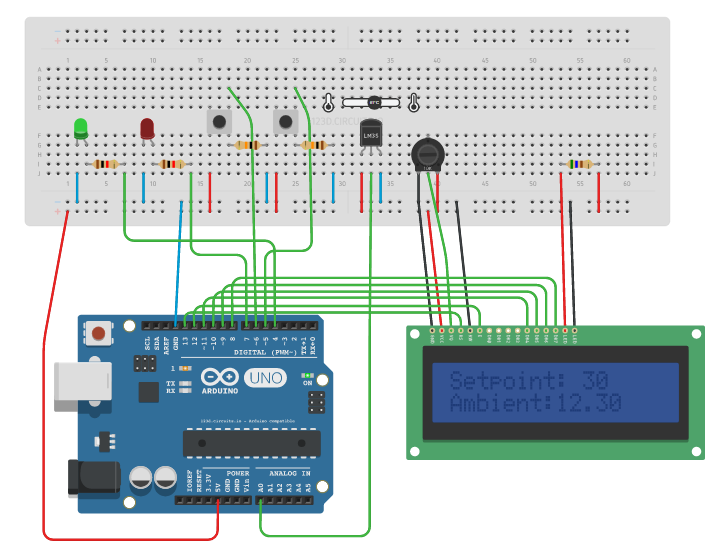


Figure , Arduino Thermostat

In the figure above the same circuit is shown, but the now the temperature is displayed however it should look a little harder to read because the contrast has been adjust by the potentiometer in the simulation in real time. Also note the ambient temp can be adjust via the slide bar above the LM35 and the setpoint can also be adjust by pressing the push buttons, again all in real time. As the ambient is below the setpoint the green LED is illuminated.

## Hardware Setup Using Simulink® Interface

The first build was to confirm that sensible readings were coming back from the LM35 and so the design simply consisted of the Arduino and the LM35 plugged into a breadboard powered by the Arduino and with the output pin connected to analogue input 1. Simulink® and the Arduino Toolbox was used to program the device and provide visual output in the form of a digital display and a scope showing the waveform.

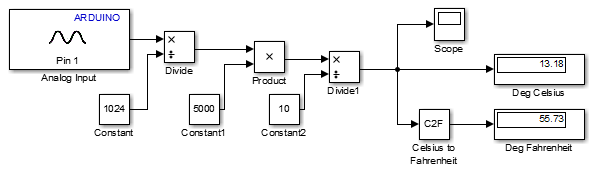


Figure Simulink model of analogue input voltage from an LM35 to an Arduino Uno and converted into a temperature reading

The block diagram above shows how the analogue input voltage from the LM35 is being captured on pin A1 of the Arduino, the analogue voltage is then divided by the resolution of the Arduino’s ADC which is -1 = 1023 steps, then multiplied by the full scale voltage of 5000mV and finally divided by 10 to give then temperature in °C. (Arduino.cc, 2017) (Atmel, 2015)

This is an extra and correction to what is mentioned above. It was decided to keep the above description of how the analogue value is converted into a temperature reading because it shows and supports the thinking behind what is written here: While going over the report it was found that the above formula for converting the analogue value and the formula in the datasheet didn’t add up. In fact the above formula didn’t make much sense at the time of writing but it worked and so it was just left alone. In the end this didn’t seem right and searching the internet found that there are many others whom feel the same, and so investigating the matter further finally resolved the issue. From the data sheet 1mV = 1°C. 250mv should therefore equal 25°C. Using this simple formula everything is fine but there is a lot of literature out there (Arduino.cc, 2017) (ZMO, et al., 2014) (David, 2013) which give a different formula; “ temp = (5.0 \* analogRead(tempPin) \* 100.0) / 1024;” AnalogRead is even referred to as ‘rawvoltage’ in some Arduino code sketches and this is where the confusion lies. We know from the data sheet that 250mV = 25°C but (5\*0.25\*100)/1024 ≠ 25°C. It equals 0.122. It was then suspected that the ‘analogRead’ was not the raw analogue voltage coming from the sensor but was in fact the digital approximation of the analogue value produced by the Arduino. Finding the digital value for 250mV: 5V/0.25V = 20V, 1024steps/20V = 51.2steps. This is the magic number we should be starting with. (5\*51\*100)/1024 ≈ 25°C. In hind sight, this seems obvious as it is only the digital value the Arduino can work with.

During this small investigation it was also realised that the ADC Vref should be the internal 1.1V and not 5V which is commonly used because the LM35 can only output a maximum value of 1.1V at 110°C and so by using a 5Vref we are using only a fifth of the resolution and only a tenth up to 25°C.

This first build using the LM35 did not work too well. The circuit was left in a calm isothermal condition, however the sensor output was very chaotic. It can be seen on the chart below that there is an unacceptable amount of noise on the signal from the LM35 and so a simple RC filter was added to the circuit to help reduce the noise.

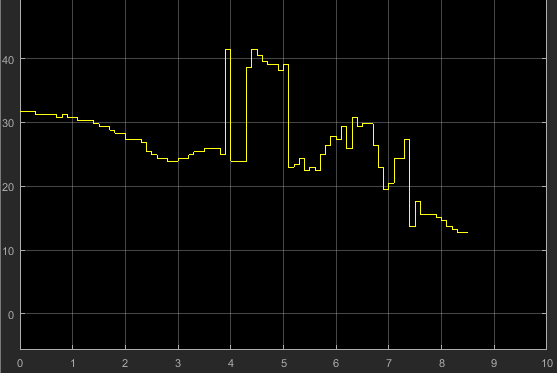


Figure Simulink scope chart showing large amount of noise on the LM35 signal input to the Arduino Uno.

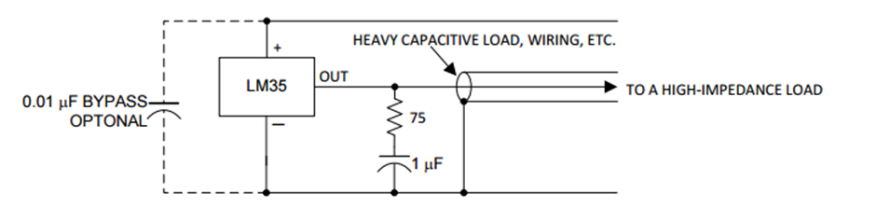
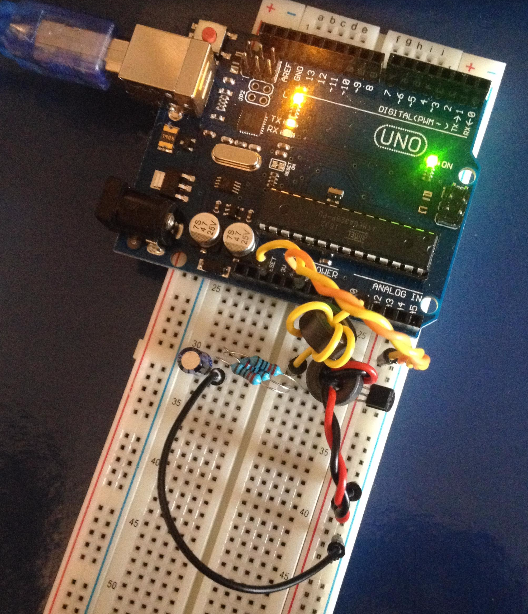


Figure LM35 with RC damper. (Texas Instruments, 2016)

The above filter design is from the LM35 datasheet (Texas Instruments, 2016) and shows 1 capacitor in series with a 75Ω resister. Also shown is a 0.01μF decoupling capacitor across the power leads and a screened cable to the high impedance load being grounded.

In the figure below the Arduino is with the LM35 next to it on the breadboard. To try and reduce noise as much as possible the filter recommended by TI has been implemented, plus the power leads have been twisted together to try and cancel any power related noise and both the power leads and the signal lead have been wrapped around a small ferrite bead each.



There was still an unacceptable level of oscillation from the LM35 despite the addition of the RC filter and decoupling capacitor. Another method attempted to solve the issue was to mount the LM35 directly in to the analogue pins of the Arduino. The signal pin from the LM35 was mounted straight into A1 (analogue input 1) and the power pins were mounted in the inputs either side, A0 and A2. As A0 and A2 were not enabled it was ok to then insert the power pins from the Arduino’s +5V and Ground into them to power the sensor. This method proved much more stable than before managing to sustain a constant temperature output, however when applying any heat to the sensor it again became erratic and oscillatory as can be seen in the chart below. The conclusion was that this particular sensor was faulty.

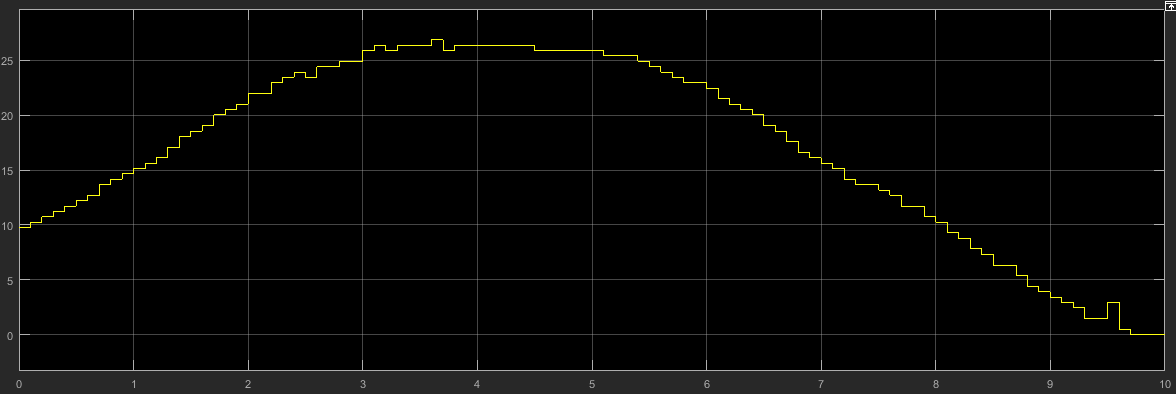


Figure Simulink scope chart showing unacceptable oscillations in the voltage reading from the LM35

## Interfacing via Simulink® ‘S’ function

It was originally hoped that the end product would all be programmed using Simulink® which would allow the use of all the signal processing and controls tools to be used to find the absolute best design in terms of stability, noise immunity and power consumption. This would have been what is known as ‘Hardware In Loop’ (HIL) testing and is a common technique used for verification and validation of new equipment. The reason for the S function was due to there being no library for the LCD in Simulink/Arduino toolbox and so one had to be created from scratch. Code from an Arduino sketch, or some other source, can be broken down into libraries, setup and loops then entered into the function block and used as an ordinary Simulink block witch interacts with most other Simulink blocks to build systems. The code used to define the S function can be found in the appendix and the block diagram of the system can be seen below. The reason why the S function wasn’t implemented was because there was a reoccurring error with the .cpp complier used by MATLAB® and SIMULINK®.

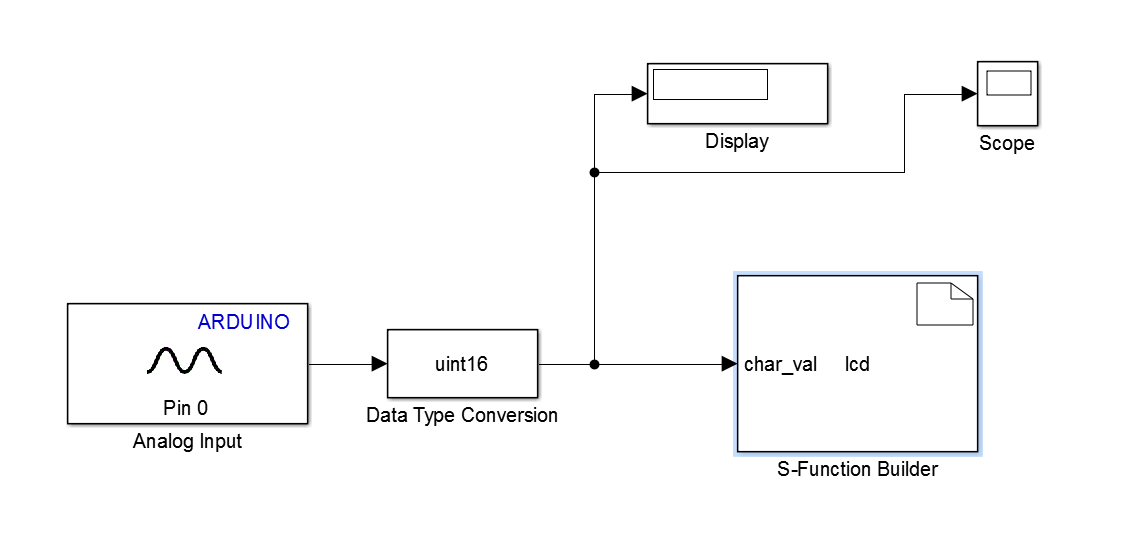


Figure , Arduino coding in Simulink® using 'S' function.

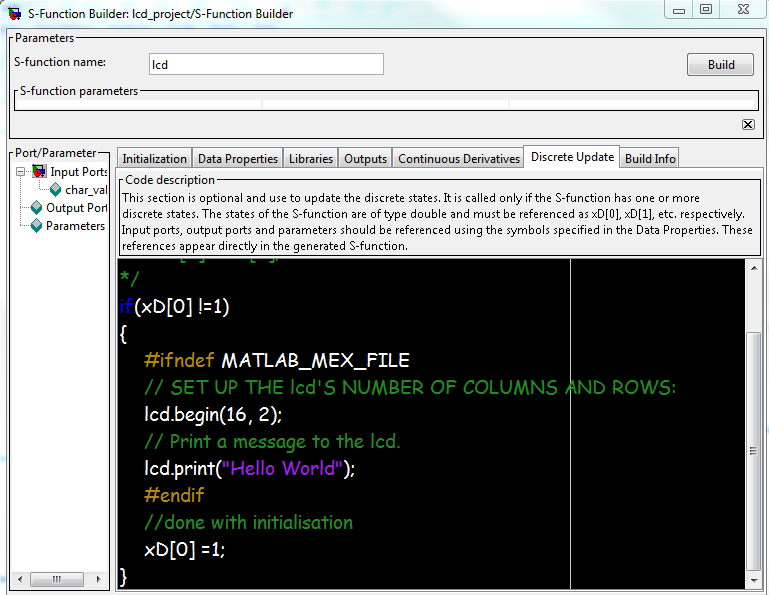


Figure , Example of the 'S' function Builder block, Simulink.

## Atmel ATMEGA328P-PU Internal Temperature Sensor

From the Atmel’s datasheet it became apparent that the ATMEGA328P had its own internal temperature sensor, similar to the LM35 it is a solid state device which is part of the chip’s wafer. The sensor is not very accurate, giving 1 /mV @ ±10 although this accuracy can be improved by calibration. (Atmel, 2016) It is supposed that this is why it is not often used, but worth investigating and could be used to turn off the Arduino or turn on a fan if it gets too hot, similar to a computer’s CPU temperature sensor. The ATMEGA382P has a max working temperature range of up to 85 (Atmel, 2015) but will work better at lower temperatures and so a cooling fan would help increase chip performance.

The Arduino.cc IDE was used to call the internal temp sensor and the output was displayed in the serial monitor below.

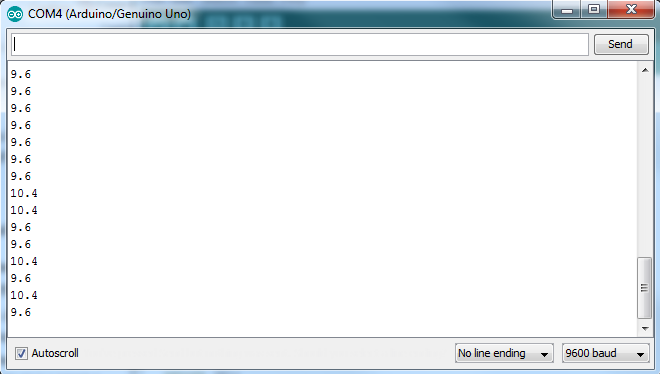


Figure Readout from Arduino Serial Monitor

The readout from the Arduino Serial Monitor confirms that the temperature is indeed inaccurate, in fact it is out by approximately 15 when compared with the reading from a non-contact IR (infra-red) thermometer.



Figure Section of sketch code for temperature offset (Arduino, 2017)

Finding the section of code (see appendix i) responsible for the offset, the value 324.31 was changed to 307.31. This altered the output temperature valve from about 10 to the more realistic 25 which agreed with the IR thermometer value.

## MAX6675 Thermocouple Amplifier

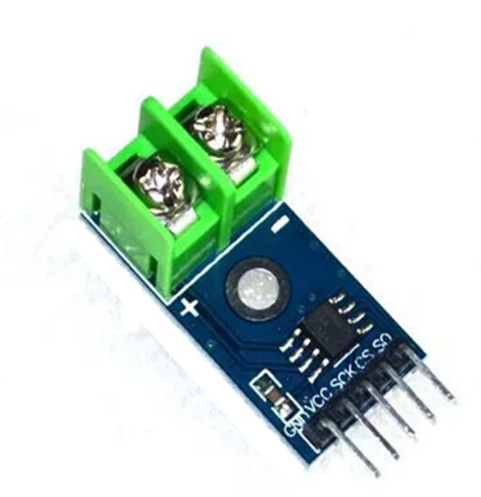


Figure , MAX6675

Due to the apparent instability and unreliability of both the LM35 and the ATMega’s own built in temperature sensor, it was decided to try and use K-type thermocouple and MAX6675 amplifier. The benefit of using the thermocouple was that it gave a very stable output, it has a very simple robust design and can be used in very harsh, high temperature and pressure environments. From experience, one thing to remember when using thermocouples is to prevent them from vibrating as this can cause them to fracture internally which will then give spurious readings. In order to maximise the usability of the MAX6675 IC with the Arduino IDE, an external library was added which allows the IDE to understand the MAX6675’s pin names and functions. (Henry, nd) (Maxim Intergrated Products, 2002)

The MAX6675 itself uses a 4.3MHz single channel 12 bit variant of the successive approximation register ADC and although not explicitly written in the datasheet, it can be inferred because it take 16 clock cycles to perform 1 data conversion, 12bits plus 4 overheads. This ADC being a 12bit type gives 4095 voltage steps which given a 5V reference would give a least significant bit (LSB) of 1.22mV per division, however the a K-type thermocouple’s output voltage changes by 41μV/°C and the MAX6675 datasheet says the maximum output is 1024°C which means a max output voltage of 42mV, so using a 5V reference would not make any sense. Although not mentioned on the datasheet, the MAX6675 must use its own internal Vref otherwise it would be losing a very large amount of it resolution. The MEX6675 also conditions the signal before being passing it to the ADC to reduce noise errors from the thermocouple wires.

“This converter resolves temperatures to 0.25°C, allows readings as high as +1024°C, and exhibits thermocouple accuracy of 8LSBs for temperatures ranging from 0°C to +700°C. VOUT = (41µV / °C) ✕ (TR - TAMB) Where: VOUT is the thermocouple output voltage (µV). TR is the temperature of the remote thermocouple junction (°C). TAMB is the ambient temperature (°C)” (Maxim Integrated Products, 2002)

## Cold-Junction Compensation

The MAX6675 uses on-board cold-junction compensation to determine the temperature at the hot end of the thermocouple. The ambient temperature at the cold junction of the MAX6675 is measured by a temperature sensing diode. The chip then outputs a temperature using the difference in voltage between these two temp sensors plus a compensating factor to account for changes in ambient temperature. There is also a cold junction compensation error of ±3°C.

The figure below is the functional block diagram of the MAX6675 showing input signal amplifier, voltage buffer, cold junction compensation, ADC and reference voltage.

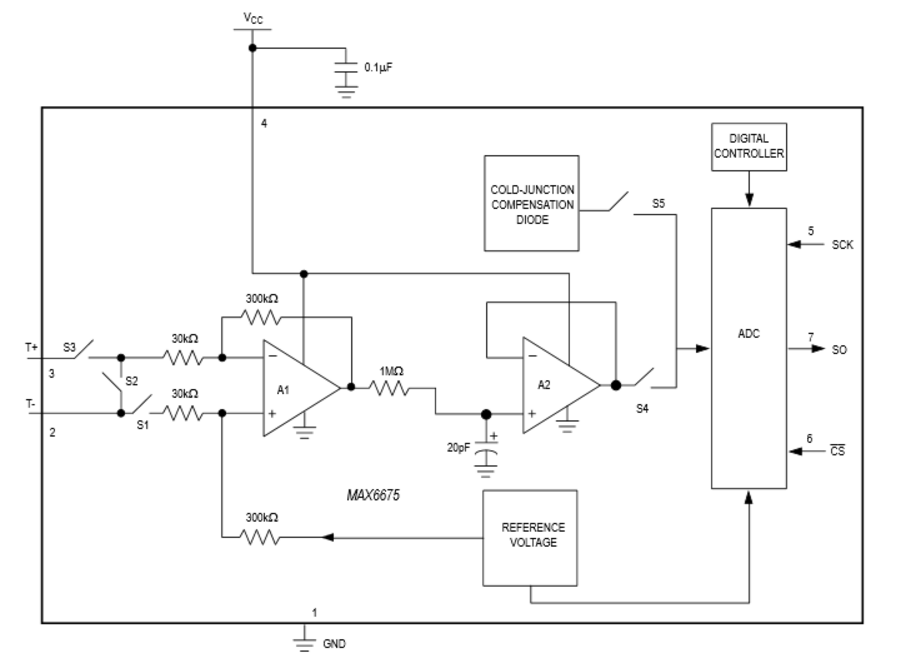


Figure , MAX6675 functional block diagram

## New K-type Thermocouple

A new thermocouple purchased from TC Direct was a 0.5mm diameter mineral insulated K Type. Being only 0.5mm means it has a much smaller thermal mass when compared to a previous thermocouples tried and consequently is able to detect smaller changes in temperature much quicker. (TC Direct, 2017)

## The Arduino Powered Thermostat

The new code used to control the thermostat was based on the previous code developed in the Circuits.io program with the LM35. The new library calls were added for the MAX6675 and the pin calls changed which is when it was found that there were too few digital pins to have the LCD display, the MAX6675, the pushbuttons and the LEDs all on one Arduino. It was thought that maybe a new larger Arduino Mega would have to be purchased. Fortunately it was realised that the Arduino’s analogue input pins can be configured like any other digital pin using ‘pinMode’ solving the problem. The new code can be found in appendix iiii.

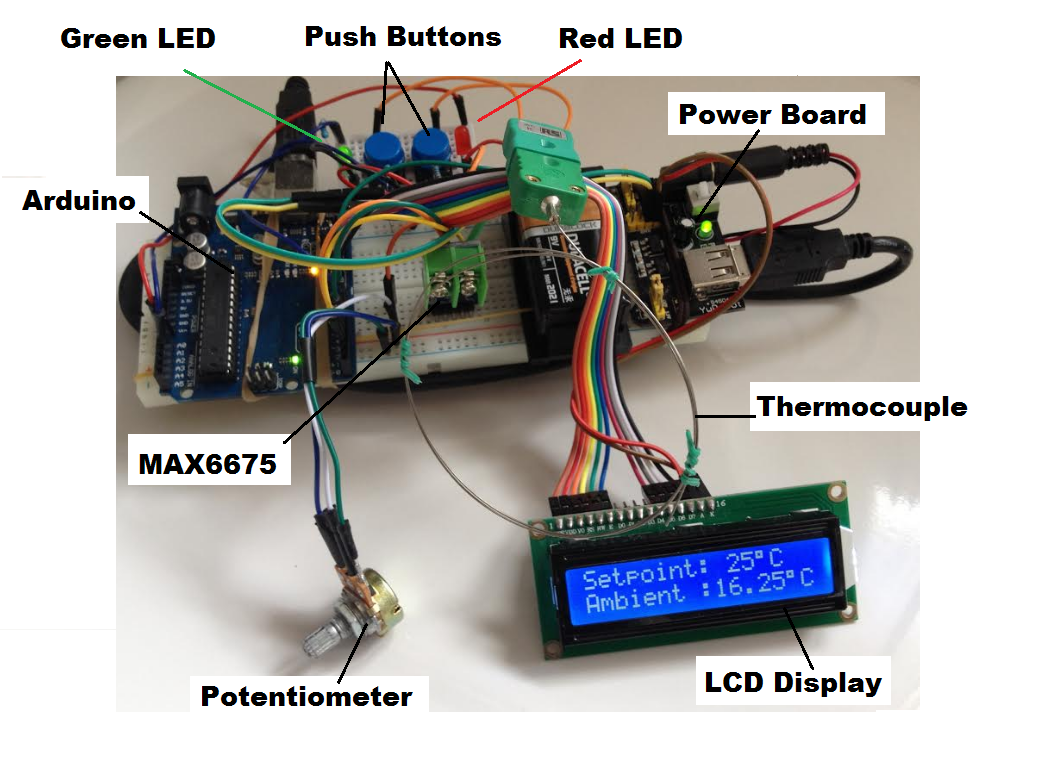


Figure , Arduino thermostat with LCD display.

The above figure shows the completed thermostat. A power board was added to allow easy switching of the thermostat power. The setpoint of 25°C can be seen on the LCD display and can be adjusted using the blue push buttons shown in the top of the picture. When the ambient temp is below the setpoint the green LED is on and when the ambient is above the setpoint the red LED is on. The thermocouple itself can be seen coiled up above the LCD display. Notice how thin the wire sheath is. This allows very fast sampling of the ambient air temperature. The potentiometer in the bottom left is used to control the LDC display contrast.

All that was left was to add some finishing touches, such as a calibration factor of -10 °C and the little degree symbols to go with the C for Celsius symbols.

A very exciting video of the thermostat working can be found at the link below:

<https://www.youtube.com/user/Kingstanding23/videos>

## Costing

All the parts needed for this project were not expensive, which is one main reason why microcontroller projects like this are so popular with electronics hobbyists. The only exception is the thermocouple from TC Direct which was £25, but this is a little more specialised and there are a lot of other cheaper versions on the market such as the one reserved with the MAX6675 amplifier module. For demonstration purposes through it was deemed too slow to react to changes in temperature. A breakdown of the parts used and their cost is shown in the table below.

|  |  |  |
| --- | --- | --- |
| Project Costing For Arduino Controlled temperature Sensor | | |
| Item | Cost | Remarks |
| 0.5mm K-Type (TC Direct) | £25.90 |  |
| Arduino Kit | £30.00 | Inc. Arduino, bread board, power module, potentiometer, wiring, LCD display and much more not used in this project. |
| MAX6675 | £6.00 | Inc. MAX6675 preassembled board and K-Type thermocouple. |
|  | £0.00 |  |
| Total | £61.90 |  |

## Ideas for Further Work

This design could be further improved by using an infrared (IR) sensor and remote control to set the temperature set point. The system could also be controlled via the internet using a wifi module and its own IP address or by running the system on the ‘SparkFun’ streaming network which allows other people to join in the fun of monitoring the temperature of my bedroom. <https://data.sparkfun.com/streams/YGzwpxb6Nbhp6v1jXK58>

As mentioned earlier in the section on Simulink® S functions, It would be good to further this work by using a function generator to provide an input to the thermostat. Then, having a known input value we can check for errors such as Total Harmonic Distortion (THD), Signal-to-Noise Ratio (SNR), Signal-to-Noise and Distortion (SINAD), Spurious Free Dynamic Range (SFDR) and Gain Error which is the ADC output Vs the analogue input. There was a small amount of gain error in the SA\_ADC simulation in section one. It would have been good to spend a lot more time investigating the various types of error that affect ADCs as these may provide better understanding of electronics design. Other errors found during this report were usually just offset using a constant, it is not yet know whether this was the correct approach and if the correction factor would have held up against large temperature fluctuation. (ti.com, 2013)

# References

Arduino.cc, 2017. *LM35HigherResolution.* [Online]   
Available at: http://playground.arduino.cc/Main/LM35HigherResolution  
[Accessed 19 04 2017].

Arduino, 2017. *Internal Temperature Sensor.* [Online]   
Available at: http://playground.arduino.cc/Main/InternalTemperatureSensor  
[Accessed 02 04 2017].

AspenCore, Inc, 2017. *Bidirectional Counters.* [Online]   
Available at: http://www.electronics-tutorials.ws/counter/count\_4.html  
[Accessed 01 03 2017].

Atmel, 2015. *ATMEL 8-BIT MICROCONTROLLER.* [Online]   
Available at: http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P\_datasheet\_Complete.pdf  
[Accessed 02 04 2017].

Atmel, 2016. *AVR122: Calibration of the AVR's Internal Temperature Reference.* [Online]   
Available at: http://www.atmel.com/Images/Atmel-8108-Calibration-of-the-AVR's-Internal-Temperature-Reference\_ApplicationNote\_AVR122.pdf  
[Accessed 02 04 2017].

AutoDesk, 2017. *AutoDesk Circuits.* [Online]   
Available at: https://circuits.io/  
[Accessed 10 02 2017].

AVR, nd. *The Analog to Digital Converter (ADC).* [Online]   
Available at: http://www.avrbeginners.net/architecture/adc/adc.html  
[Accessed 15 04 2017].

David, 2013. *How to Build a LM35 Temperature Sensor Circuit.* [Online]   
Available at: http://www.learningaboutelectronics.com/Articles/LM35-temperature-sensor-circuit.php  
[Accessed 19 04 2017].

EETech Media, LLC, nd. *Parallel-in, Parallel-out, Universal Shift Register.* [Online]   
Available at: https://www.allaboutcircuits.com/textbook/digital/chpt-12/parallel-in-parallel-out-universal-shift-register/  
[Accessed 10 03 2017].

EETech Media, LLC, ND. *Successive Approximation ADC.* [Online]   
Available at: https://www.allaboutcircuits.com/textbook/digital/chpt-13/successive-approximation-adc/  
[Accessed 01 03 2017].

EETech Media, LLC, nd. *Synchronous Counters.* [Online]   
Available at: https://www.allaboutcircuits.com/textbook/digital/chpt-11/synchronous-counters/  
[Accessed 01 03 2017].

Fritzing, 2017. *Fritzing.* [Online]   
Available at: http://fritzing.org/home/  
[Accessed 01 03 2017].

Hedayati, R., 2011. *A Study of Successive Approximation Registers and Implementation of an Ultra Low Power 10-bit SAR ADC in 65nm CMOS Technology.* [Online]   
Available at: http://www.diva-portal.org/smash/get/diva2:462318/FULLTEXT01.pdf&sa=U&ei=d0NOU-XlB4KMyATorILYDQ&ved=0CEoQFjAJ&usg=AFQjCNGl6wCLfPa\_9FmelE79HCujTETokQ  
[Accessed 01 03 2017].

Henry, nd. *MAX6675 Temp Module Arduino Manual and Tutorial.* [Online]   
Available at: http://henrysbench.capnfatz.com/henrys-bench/arduino-temperature-measurements/max6675-temp-module-arduino-manual-and-tutorial/  
[Accessed 07 04 2017].

Maxim Integrated Products, 2002. *Cold-Junction-Compensated K-Thermocoupleto-Digital Converter (0°C to +1024°C).* [Online]   
Available at: https://cdn-shop.adafruit.com/datasheets/MAX6675.pdf  
[Accessed 19 04 2017].

Maxim Intergrated Products, 2002. *Cold-Junction-Compensated K-Thermocouple to-Digital Converter (0°C to +1024°C).* [Online]   
Available at: http://henrysbench.capnfatz.com/wp-content/uploads/2015/05/MAX6675-Datasheet.pdf  
[Accessed 07 04 2017].

McKean, D., 2017. *D.McKean\_Thermostat\_Project.* [Online]   
Available at: https://circuits.io/circuits/4681962-d-mckean\_thermostat\_project  
[Accessed 01 04 2017].

Robot Platform, 2017. *ADC in Atmega8 – Summary.* [Online]   
Available at: http://www.robotplatform.com/knowledge/ADC/adc\_tutorial\_4.html  
[Accessed 01 03 2017].

Russell, D. & Thornton, M., nd. *Analog­to­Digital Conversion.* [Online]   
Available at: http://web.csulb.edu/~hill/ee470/Lab%202c%20-%20Analog-to-Digital-Conversion.pdf  
[Accessed 04 03 2017].

Stewart, D., 2004. *Flip-Flops and Registers.* [Online]   
Available at: http://cpuville.com/register.htm  
[Accessed 10 03 2017].

TC Direct, 2017. *Miniature 0.25, 0.5 and 0.75mm dia Mineral Insulated Thermocouple with Miniature Flat Pin Plug.* [Online]   
Available at: https://www.tcdirect.co.uk/Default.aspx?level=2&department\_id=200/2  
[Accessed 08 04 2017].

Texas Instruments, 2016. *LM35 Precision Centigrade Temperature Sensors.* [Online]   
Available at: http://www.ti.com/lit/ds/symlink/lm35.pdf  
[Accessed 21 03 2017].

ti.com, 2013. *ADC Performance Parameters.* [Online]   
Available at: http://www.ti.com/lit/an/slaa587/slaa587.pdf  
[Accessed 20 04 2017].

Vandenberg, G., 2014. *dvanced Arduino ADC – Faster analogRead().* [Online]   
Available at: http://www.microsmart.co.za/technical/2014/03/01/advanced-arduino-adc/  
[Accessed 01 03 2017].

Yates, D., 2014. *Arduino’s analog-to-digital converter: how it works.* [Online]   
Available at: http://apcmag.com/arduino-analog-to-digital-converter-how-it-works.htm/  
[Accessed 02 04 2017].

Yates, D., 2014. *Arduino’s analog-to-digital converter: how it works.* [Online]   
Available at: http://apcmag.com/arduino-analog-to-digital-converter-how-it-works.htm/  
[Accessed 01 03 2017].

ZMO, shajib0o & Trustchildi, 2014. *what is conversion system of LM35 (temperature sensor) in Celsius?.* [Online]   
Available at: http://stackoverflow.com/questions/24066852/what-is-conversion-system-of-lm35-temperature-sensor-in-celsius  
[Accessed 19 04 2017].

# Appendix

## Appendix i LM35 & LCD Code Developed in Circuits.IO

#include <LiquidCrystal.h>

#include <stdio.h>

#include <EEPROM.h>

// Pin 13 has an LED connected on most Arduino boards.

// give it a name:

int red = 4;

int green=7;

int sw=5;

int sw2=6;

int temp=0;

int refTemp=25;

int addr = 0;

LiquidCrystal lcd(13, 12, 11, 10, 9, 8);

// the setup routine runs once when you press reset:

void setup() {

lcd.begin(16, 2); // Initializing 16X2 LCD.

lcd.home(); // Home location : 0,0

lcd.print("Temperature"); // Print on LCD.

lcd.setCursor(0, 1);

lcd.print("Sensor Display");

delay(3000);

lcd.clear();

lcd.home(); // Home location : 0,0

lcd.print("Using Arduino"); // Print on LCD.

lcd.setCursor(0, 1);

lcd.print("LM35 & LCD ");

delay(3000);

lcd.clear();

lcd.home(); // Home location : 0,0

lcd.print("A Project By"); // Print on LCD.

lcd.setCursor(0, 1);

lcd.print("Dom McKean");

delay(3000);

lcd.clear();

// initialize the digital pin as an output.

pinMode(red, OUTPUT);

pinMode(green, OUTPUT);

pinMode(sw, INPUT);

pinMode(sw2, INPUT);

pinMode(temp, INPUT);

// Serial monitor debug code

Serial.begin(9600);

int epromVal=EEPROM.read(addr);

if (epromVal!=0){

refTemp=epromVal-40;

Serial.println("SetPointRead"+refTemp);

}

lcd.begin(16, 2);

// Print a message to the LCD.

lcd.print("Setpoint:"+String(refTemp)+"°C");

}

void toggle(int led) {

if (digitalRead(led)==HIGH){

digitalWrite(led, LOW);

} else {

digitalWrite(led, HIGH);

}

}

void checkSwChangeTemp(int sw, int change){

int swStatus=digitalRead(sw);

if (swStatus==HIGH && refTemp>-39 && refTemp<124){

refTemp+=change;

EEPROM.write(addr, refTemp+40);

lcd.clear();

lcd.print("Setpoint: "+String(refTemp)+"°C");

}

}

// the loop routine runs over and over again forever:

void loop() {

checkSwChangeTemp(sw,1);

checkSwChangeTemp(sw2, -1);

delay(200);

float tempC=(analogRead(temp)\*5.00/1024.00-0.4971)\*100.00;

lcd.setCursor(0,1);

lcd.print("Ambient:"+String(tempC)+"°C ");

//Serial.println(tempC);

if (tempC<refTemp){

digitalWrite(red, HIGH);

digitalWrite(green, LOW);

} else {

digitalWrite(green, HIGH);

digitalWrite(red, LOW);

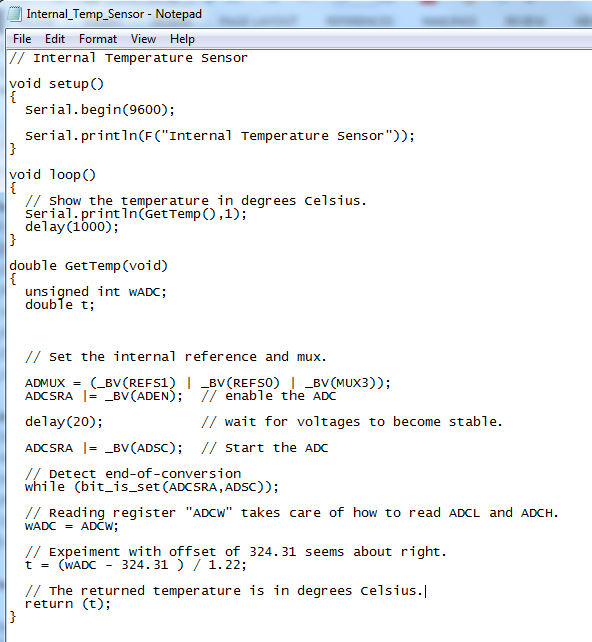
}

}

//end of code

------------------------------------------------------------------------------------------------------------------------------------

## Appendix ii ATMEGA328P Internal Temperature Sensor Sketch



(Arduino, 2017)

## Appendix iii MAX6675 Thermocouple Code

// Sample Arduino MAX6675 Arduino Sketch

#include "max6675.h"

int ktcSO = 8;

int ktcCS = 9;

int ktcCLK = 10;

MAX6675 ktc(ktcCLK, ktcCS, ktcSO);

void setup() {

Serial.begin(9600);

// give the MAX a little time to settle

delay(500);

}

void loop() {

// basic readout test

Serial.print("Deg C = ");

Serial.print(ktc.readCelsius());

Serial.print("\t Deg F = ");

Serial.println(ktc.readFahrenheit());

delay(500);

}

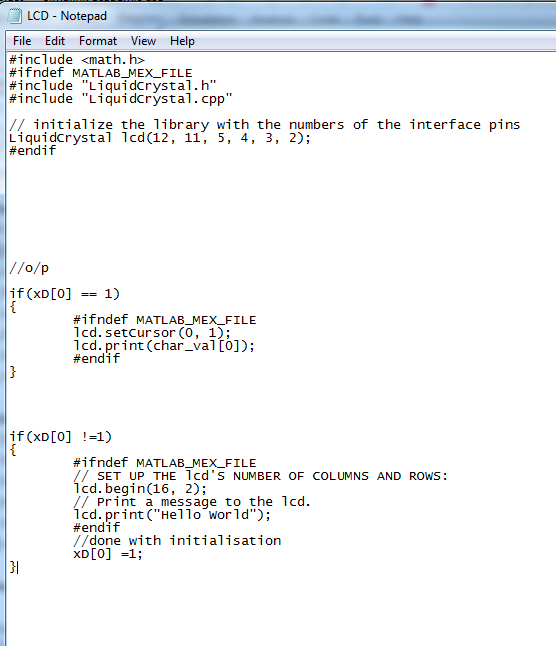
//end of code

(Henry, nd)

----------------------------------------------------------------------------------------------------------------------------------

## Appendix iii

Code used for Simulink S function for integrating the LCD display into the Simulink interface code for controlling the Arduino thermostat.



## Appendix iv Final Code for Physical Arduino + MAX6675 + LCD Display + Degree Symbol + Push Buttons + LEDs

#include <LiquidCrystal.h>

#include "max6675.h" //new library for MAX6675

#include <EEPROM.h>

// MAX6675 pins to arduino digital inputs

int ktcSO = 2;

int ktcCS = 3;

int ktcCLK = 4;

//initialise MAX6675

MAX6675 ktc(ktcCLK, ktcCS, ktcSO);

// declare LEDs and switches

int red = A0;

int green=A1;

int sw=5;

int sw2=6;

//int temp=

int Setpoint=25;

int addr = 0;

LiquidCrystal lcd(13, 12, 11, 10, 9, 8);

// the setup routine runs once when you press reset:

void setup() {

lcd.begin(16, 2); // Initializing 16X2 LCD.

lcd.home(); // Home location : 0,0

lcd.print("Temperature"); // Print on LCD.

lcd.setCursor(0, 1); // prints on second row

lcd.print("Sensor Display");

delay(2000); // 2 second delay

lcd.clear(); // clear the display

lcd.home(); // Home location : 0,0

lcd.print("A Project By"); // Print on LCD.

lcd.setCursor(0, 1);

lcd.print("Dom McKean");

delay(2000);

lcd.clear();

// Declare digital pin as either inputs/outputs.

pinMode(red, OUTPUT);

pinMode(green, OUTPUT);

pinMode(sw, INPUT);

pinMode(sw2, INPUT);

// Send debugging information via the Serial monitor

Serial.begin(9600);

/\* int epromVal=EEPROM.read(addr);

if (epromVal!=0){

Setpoint=epromVal-40;

//Serial.println("SetpointRead"+Setpoint);

}\*/

lcd.begin(16, 2);

// Print a message to the LCD.

lcd.print("Setpoint: "+String(Setpoint));

lcd.print((char)223);

lcd.print ("C");

}

void toggle(int led) {

if (digitalRead(led)==HIGH){

digitalWrite(led, LOW);

} else {

digitalWrite(led, HIGH);

}

}

void checkSwChangeTemp(int sw, int change){

int swStatus=digitalRead(sw);

if (swStatus==HIGH && Setpoint>-39 && Setpoint<50){

Setpoint+=change;

EEPROM.write(addr, Setpoint+40);

lcd.clear();

lcd.print("Setpoint: "+String(Setpoint));

lcd.print((char)223);

lcd.print ("C");

}

}

// the loop routine runs over and over again forever:

void loop() {

checkSwChangeTemp(sw,1);

checkSwChangeTemp(sw2, -1);

delay(200);

double c = ktc.readCelsius();

lcd.setCursor(0, 1);

if (isnan(c))

{

lcd.print("T/C Problem");

}

else

{

lcd.print("Ambient :");

lcd.print(c-10); // prints temp value + correction factor

lcd.print((char)223);

lcd.print("C"); // prints the c charictor for Celsius

}

/\*lcd.setCursor(0,1);

lcd.print(ktc.readCelsius ());

//Serial.println(ktc);

\*/ if (c-10<Setpoint){

digitalWrite(red, LOW);

digitalWrite(green, HIGH);

}

if (c-10>Setpoint){

digitalWrite(green, LOW);

digitalWrite(red, HIGH);

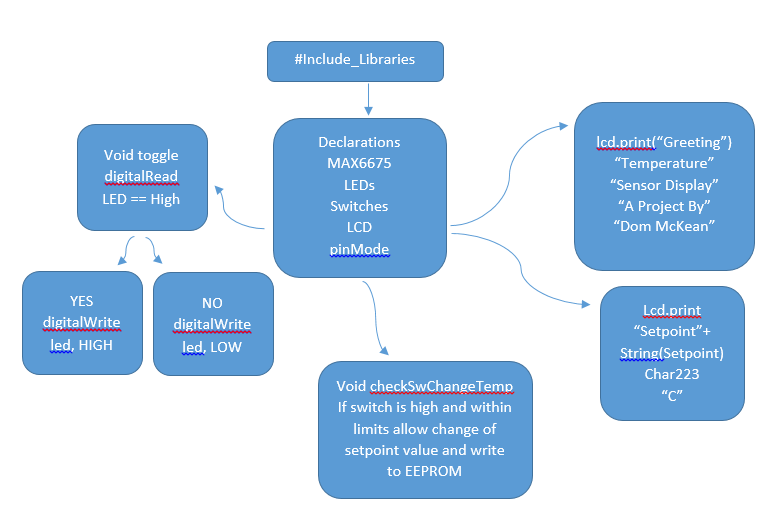
}

delay(500);

}

//end of code

## Appendix v Initial Circuits.io Code + MAX6675 Setup Flowchart



## Appendix vi Initial Circuits.io Code Loop Flowchart

