UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

## Escuela Técnica Superior de Ingeniería de Telecomunicación

# Desarrollo de herramientas hardware y software para la mejora de el proceso de control de calidad en instrumentación y accesorios para sistemas de sensores QCM

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Forcada García, Manel

Tutor/a: Arnau Vives, Antonio

Cotutor/a externo: GARCIA NARBON, JOSE VICENTE

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

# DEVELOPMENT OF HARDWARE AND SOFTWARE TOOLS FOR THE IMPROVEMENT OF QUALITY CONTROL PROCESSES IN INSTRUMENTATION AND ACCESSORIES FOR QUARTZ CRYSTAL MICROBALANCE SENSOR SYSTEMS

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
**www.etsit.upv.es**

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

## Abstract

AWSensors is a company that designs Quartz Crystal Microbalance with Dissipation (QCMD)-based sensing systems,

There are many steps from design to final product, including prototyping, verification, calibration and quality control, the latter being the one of interest for this end-of-degree dissertation

AWSensors manufactures a variety of cables for internal and external connections within their products. One of these cables is made up of 12 lines and a coaxial cable; in the event that one of them fails, the system will not work, and the cable will need to be tested manually to find the fault and fixed

A way to improve the manufacturing process would be to build a cable testing device for cables such as this one.

A project of this kind requires expertise about digital and analog electronics, microcontroller programming, and PCB design.

A good option for the microcontroller would be the ATmega family, which is a family of affordable and capable 8-bit microcontrollers.

This, as well as other components can be easily obtained through retailers like Digikey or Mouser. The PCBs can be manufactured by third parties such as Eurocircuits, and the assembly and soldering process can be done by hand.

The final product will aid in the manufacturing process at AWSensors.

## Resumen

AWSensors es una empresa que diseña sistemas de detección basados en QCMD,

Desde el diseño hasta el producto final hay muchos pasos, como la creación de prototipos, la verificación, la calibración y el control de calidad, siendo este último el de interés para este TFG.

AWSensors fabrica una variedad de cables para las conexiones internas y externas de sus equipos, uno de estos cables está compuesto por 11 líneas y un cable coaxial, en el caso de que uno de ellos falle el sistema no funcionará y se debe probar el cable manualmente para encontrar el fallo y arreglarlo

Una forma de mejorar el proceso de fabricación sería diseñar un dispositivo de prueba para cables como éste,

Un proyecto de este tipo requeriría conocimientos de electrónica digital y analógica, programación de microcontroladores y diseño de placas de circuito impreso.

Una buena opción para el microcontrolador sería la familia atmega, que es una familia de microcontroladores de 8 bits potentes y asequibles.

Este y otros componentes se pueden obtener fácilmente a través de distribuidores como digikey o mouser, las pcbs pueden ser fabricadas por terceros como eurocircuits y el proceso de montaje y soldadura se puede hacer a mano.

El producto final ayudará en el proceso de fabricación en AWSensors.

## Resum

AWSensors és una empresa que dissenya sistemes de detecció basats en QCMD,

Des del disseny fins al producte final hi ha molts passos, com la creació de prototips, la verificació, el calibratge i el control de qualitat, i aquest és el subjecte d'interès per a aquest TFG.

AWSensors fabrica una varietat de cables per a les connexions internes i externes dels seus equips, un d'aquests cables està compost per 11 línies i un cable coaxial, en cas que un falli el sistema no funcionarà i s'ha de provar el cable manualment per trobar el error i arreglar-ho

Una manera de millorar el procés de fabricació seria dissenyar un dispositiu de prova per a cables com aquest,

Aquest projecte requeriria coneixements d'electrònica digital i analògica, programació de microcontroladors i disseny de plaques de circuit imprès.

Una bona opció per al microcontrolador seria la família atmega, que és una família de microcontroladors de 8 bits potents i assequibles.

Aquest i altres components es poden obtenir fàcilment a través de distribuïdors com digikey o mouser, les pcbs poden ser fabricades per tercers com eurocircuits i el procés de muntatge i soldadura es pot fer a mà.

El producte final ajudarà en el procés de fabricació a AWSensors.

# Index

# Chapter 1. Objectives

The main objective for this project is to provide a tool that serves a purpose within the company AWSensors.

This means that the final result should be a physical device, and this work will not be purely theoretical.

To develop this tool there will be a theoretical study, a design and a prototype, the design will be mainly in the subject of electronics and PCBs (Printer Circuit Boards), and software written in C++.

An area of the company has been chosen, in this case the workshop, and a specific task has been identified for this tool, this task is the quality control of an complex interconnection cable used for equipment built by the company.

The task that this device will perform is to inject signals in the cable at one end and read them at the other, process the information and display information to the user.

The information at hand and the specifics of the device and the cable will be explained in the following chapters, as well as the design and manufacturing process.

Finally, when the prototype is built, the software for its functioning will be written and debugged.

The final and most important objective is for the device to be useful for the people working in the workshop at AWSensros, so ease of use and durability will be priorities when developing this project.

# Chapter 2. Design proposal and theoretical study

## 2.1 Problem to solve

AWSensors uses proprietary cables for the external connections between the different modules in their sensing systems

One such cable is, as their internal denomination calls it, AWS WIR 000080 A

This cable consists of a main cable, shielded by a metal mesh, within which there are 12 individual cables as a bus, and an external coaxial cable running parallel to it.

The 12 individual cables are used for digital signals such as USB protocols or control of different modules, as well as powering lines

The coaxial cable is used to transmit a clock signal used in the sensing devices.

The connectors for both ends of both cables are manually soldered at the company; this is the most critical part, as any mistake in the soldering process could cause a short circuit or open circuit within the cable, which would result in the malfunctioning of the sensing system.

A detail of the cables with the connectors and pinout used for manufacturing can be seen in figure 1 below.

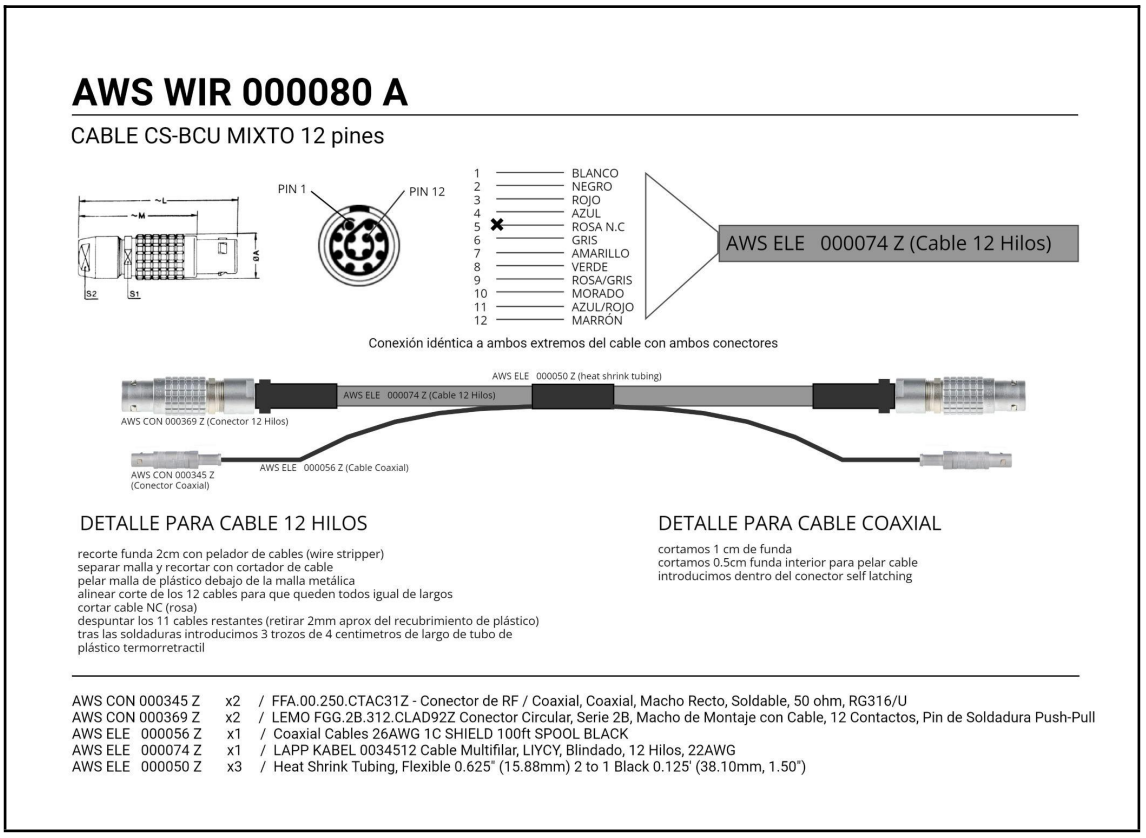


**Figure 1.** Detail for AWS WIR 000080 A

## 2.2 Design proposal

A device to test all the connections in such cable would need, first of all, two pairs of female connectors for both the main and coaxial cables.

Such device will need a way to test each connection by separately checking its impedance and its possible short circuit to any other connection; this suggests a sequential action.

Finally, it would need a way to display such information to the user.

A known resistance configuration to read the drop in voltage in the cables is a possible solution, this would mean driving a current through each individual cable, in series with a known resistance, and reading the voltage drop across the cable.

The sequential action needed creates a complex problem,which can be solved using digital electronics for control; the impedance reading creates a problem of an analog nature, altogether suggesting a hybrid analog/digital design for this solution.

For the digital side, an ATmega family microcontroller would be useful, since it has many digital input and output ports and also has analog input pins, as well as sufficient memory and power to run the program that we will need.

For the analog side, we will need a way to detect a voltage drop in the cables, suggesting an analog reading of 14 individual cables, including the 12 cables in the main bus and the active and shield connections in the coaxial cable.

Therefore, given that the ATmega microcontrollers don't have this many analog inputs we will need an analog multiplexer.

To generate this drop in voltage, we will need a way to drive a current through the cables; a suitable way to do this is with the use of Darlington pairs.[1] There are commercial chips that integrate multiple Darlington pairs in a single integrated circuit (IC).

The suggested design for the device is shown in figure 2 and will be explained in more detail in the next chapter.

[1] https://en.wikipedia.org/wiki/Darlington_transistor

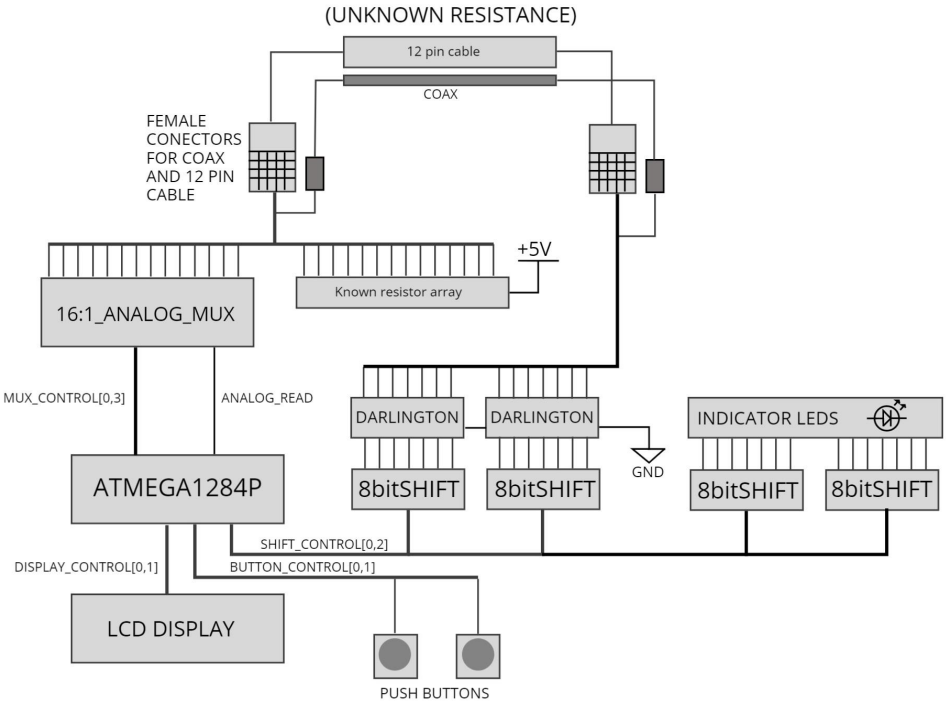


**Figure 2.** Design proposal block diagram

## 2.3 Mode of operation

### *2.3.1 Connection health*

The following text will reference bus connections in the previous diagram written in bold letters and followed the first and number of the bytes it uses, for example SHIFT_CONTROL[0,2] is a bus with 3 connections, 0, 1, and 2.

If the name in bold letters is not followed by any number this means this is a single wire and not a bus.

The mode of operation is as follows.

The main control of the device is the ATMEGA1284P microcontroller; it will drive the shift registers through the SHIFT_CONTROL[0,2] bus, selecting one of 14 outputs for such registers.

The selected output will in turn, drive a Darlington pair connected to one of 14 connections in the cable, this will allow a current to flow from +5 V, through the known resistor of the selected connection, as well as through the cable and then to ground, closing the known resistor circuit.

With this circuit closed, the microcontroller will then drive the MUX_CONTROL[0,3] bus; this bus selects which of the 16 inputs to the multiplexor is forwarded to its output; in this way the microcontroller will select the input corresponding to the connection that is being tested.

This output is connected to an analog input pin in the microcontroller through the ANALOG_READ wire, all together allowing the microcontroller to do an analog read on any of the fourteen connections of the cable.

If there is a good connection, the 5 V should all fall in the known resistor, and the voltage read at the tested connection should be very close to 0.

In the case of a short circuit there will be no current flow, so there will be no voltage drop in the resistor and the connection reading will show 5 V.

As for defective connections, these will have a much higher impedance than normal, so there will be some voltage drop in the connection, and the voltage read will be above 0 V.

### *2.3.2 Short Circuits*

Up to this point we have checked if the individual connection is good, but we haven't checked for short circuits between the cables; to do this an additional step for each connection is needed.

For each connection, an analog read will be done, and the shift registers will be cycled, enabling all of the Darlington pairs sequentially; if there is no short circuit, the value of the read will always be 5 V except in the case when the connection that is being fed current is the one that is being read.

If when feeding a connection that is not the one being read we detect a 0 V reading, this means that such connection is short-circuited to the one being read.

This whole process is repeated 14 times, assessing all of the connections and their relation to others if any.

### *2.3.3 Data output*

As for the data output, there are 14 indicator light-emitting diodes (LEDs), each one corresponding to one connection in the cable.

There will be 2 modes of operation, *connections* and *short-circuits*; the user will be able to toggle between them with the use of a push button.

The connections mode will display a turned-on LED for a good connection, a turned-off LED for an open circuit. and a blinking LED led for a high impedance connection.

The short-circuits mode will display, if there are any, the pairs of cables that are short-circuited by lighting them up; if there are multiple, it will cycle through them.

Two pins from the microcontroller have also been forwarded to a connector so that a future revision could allow the use of a liquid crystal display (LCD) screen.

## 2.4 Individual component study

### *2.4.1 Microcontroller*

**Figure 3.** ATMEGA1384 DIP packaging

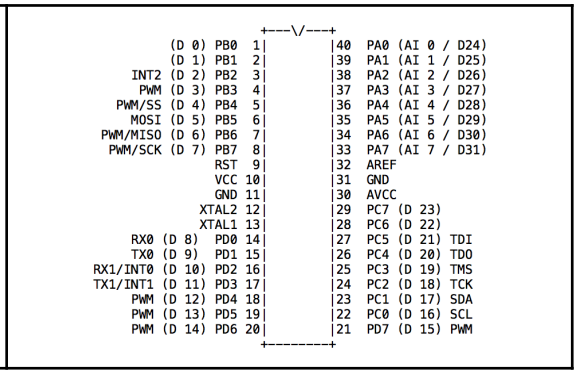


**Figure 4.** ATMEGA1384 pinout

The microcontroller chosen is the ATMEGA1284P, seen in figure 3, It has an 8 bit reduced instruction set computer (RISC) central processing unit (CPU) capable of working at up to 20 MHz.

This microcontroller is capable enough to run any program that we write for the task at hand, given its simplicity, but there are a few other reasons why I have chosen this microcontroller for this task:

- It has 8 analog input pins capable of taking readings with 10 bits of precision; this function will be crucial in the voltage readings required for this design;
- It has 24 digital I/O pins that can be used to control all of the peripheral electronics needed for the design;
- It is well supported, meaning that it has good software and hardware for the programming, and it can be programmed in C;
- It comes in DIP and SMD packages, which makes it easy to solder;
- and finally, and most importantly, it is very cheap.

### *2.4.2 Analog multiplexer*

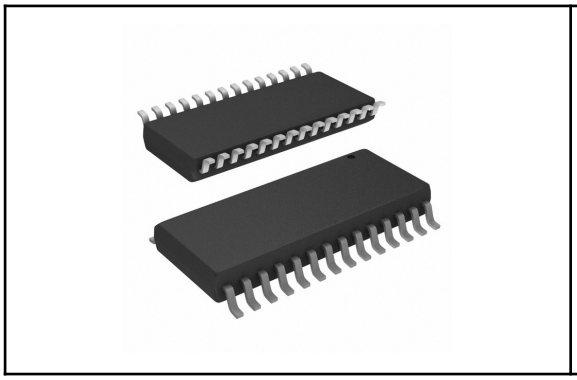

**Figure 5.** MAX307 SMD packaging

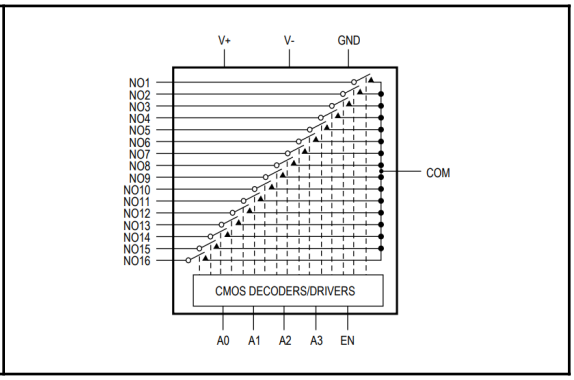


**Figure 6.** MAX307 function detail

Given the fact that there are 14 points where analog readings need to be taken and only 8 analog inputs, a way to multiplex these analog signals is needed.

A good option was the MAX307CWI, shown in figure 5; it is a 16:1 analog multiplexer, meaning that we can direct any of the 14 readings to a single analog input in the microcontroller.

It is controlled through 4 control pins; by changing the digital values of these 4 pins we can route any of its 16 inputs to the common (COM) output.

This device can operate with voltages up to 30 V, which is more than enough for the 5 V readings that we need.

### *2.4.3 Shift Registers*

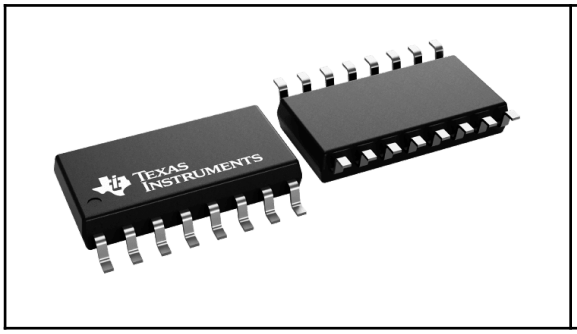


**Figure 7.** SN74HC595 shift register

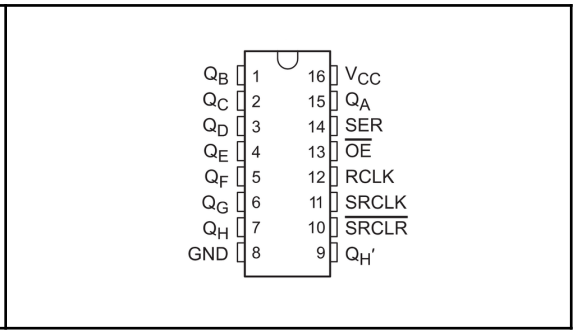


**Figure 8.** SN74HC595 pinout

The Texas Instruments SN74HC595 is a simple and inexpensive integrated circuit implementing a single 8 bit shift register; its use in our design is to increase the number of digital outputs.

It will be used in 2 stages in our design: 2 shift registers will drive the inputs of the Darlington high current drivers and 2 other registers will drive the 14 indicator LEDs.

These shift register ICs have a clock signal, a serial input for the data and a latch signal; when the 8 bits of data have been serially sent to the IC, the latch signal is activated, outputting the data to the 8 output pins.

These ICs can be connected serially, allowing any number of them to be driven only with 3 connections, serial data, clock and latch.

### *2.4.4 High current drivers*

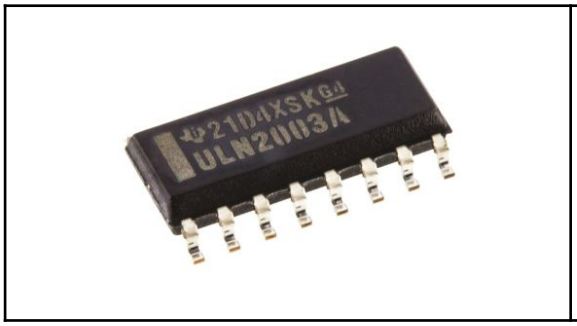


**Figure 9.** ULN2003A Current driver

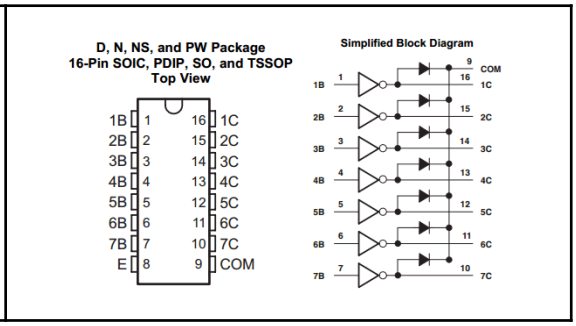


**Figure 10.** ULN2003A pinout and detail.

The ULN2003A, shown in figure 9, is an array of 7 Darlington pairs; this IC allows driving 7 individual circuits with currents up to 500 mA.

Two of them will be used to drive the 14 connections in the design.

Initially the option to drive the connections directly with the outputs from the shift registers was considered, but these ICs are limited to 5 mA per output.

This fact combined with the low resolution of the analog inputs in the microcontroller would mean that the resolution in ohms would be very bad when measuring impecandce. This will be later explained in more detail.

The B ports are the inputs: when a 0 V signal is present, the corresponding C port is set to high impedance.

When a 5V signal is forwarded to one of the B inputs, the circuit is closed between its corresponding C port and the common E pin, which is connected to ground, thus driving current through that circuit.

### *2.4.5 Peripheral electronics*

Things such as connectors, LEDs, capacitors and resistors will also be needed as support electronics for the project.

Many of the above-mentioned ICs require decoupling capacitors at their power pins for correct operation.

Resistors are also needed in voltage dividers or as pull-up resistors.

Resistors and capacitors will be all in surface-mounted device (SMD) packages, while the LEDs and connectors will be through hole devices.

## 2.5 Theoretical study

The complexity of our design stems from the fact that the readings have to be done sequentially on a number of connections and the different protocols that need to be followed to identify problems in the cable. The analog reading, on the other hand, is quite simple.

The following figure is a diagram of the analog reading taken at each connection of the cable.

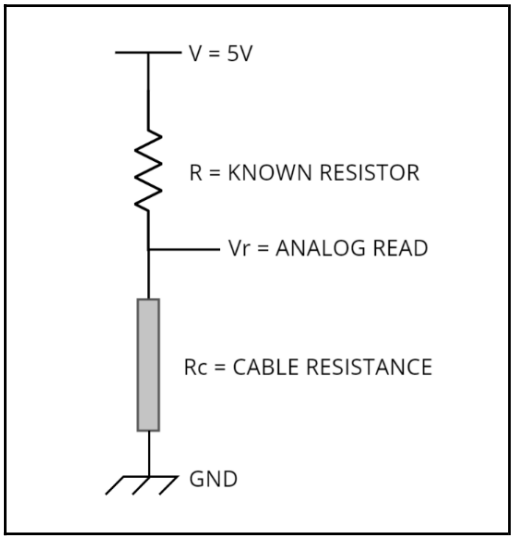


**Figure 11.** Analog read diagram

### *2.5.1 Equations for analog reading*

As the circuit shows, a simple voltage divider will be used for the analog reading of impedances.

V is the 5V supply voltage, R is the known resistor corresponding to each line, Rc is the impedance of the line, the value we want to measure, and finally Vr is the voltage value that will be red by the analog input of the microcontroller.

The voltage for the analog reading is given by the following formula:

$$V_r = V \frac{R + R_c}{R_c}$$

Given a certain bit depth in the analog to digital converter for the analog inputs in the microcontroller, we can calculate the minimum voltage difference that can be read at that point:

$$V_{resolution} = \frac{V}{2^{BitDepth}}$$

If we isolate $R_c$ from the first formula, we get the following equation:

$$R_c = \frac{V_r * R}{V - V_r}.$$

Using derivatives, we can calculate the step in resistance given by a step in voltage:

$$\frac{dR_c}{dV_r} = R \frac{d}{dV_r}\left(\frac{V_r}{V - V_r}\right) = \frac{R * V}{(V - V_r)^2}.$$

Thus giving:

$$\Delta R_c = \Delta V_r \times \frac{R * V}{(V - V_r)^2}$$

Given the fact that the measurements that we are interested in are very small values in the resistance of a line we can assume that $V_r$ will be a value a lot smaller than $V$.

In other words, for $R_c \ll R$ we have $V_r \ll V$

Assuming this, we can simplify the equation for the change in $R_c$.

$$\Delta R_c = \frac{R * V}{(V - V_r)^2} \Delta V_r$$

Assuming $V_r \ll V$ we get:

$$\Delta R_c \simeq \frac{R}{V} \Delta V_r,$$

### *2.5.2 Resolution values*

If we choose a small value for $R$, the values for $R_c$ that we will be able to detect will also be smaller; therefore, so $R$ has to be minimized.

The main factor that limits this is the maximum current per circuit of the high current drivers, we know that this current is 500 mA.

Given this value, we can calculate a minimum value for $R$ with the following equation

$$R_{min} = \frac{V}{Imax}$$

Given a value for $V$ of 5 V and an $I_{max}$ value of 500 mA we get a minimum resistance of 10 Ω.

This would mean a maximum current of 500 mA per line, but pushing the circuit to its limits is not a good idea.

500 mA is a considerable current, and not all power supplies will be able to give us this current, a resistance of 20 Ω, giving a maximum current of 250 mA is more manageable.

Now that we have a comprehensive understanding of the resolution of our readings and a chosen value for *R* we can estimate the minimum change in resistance detectable.

The analog reading of the analog-to-digital converters in the microcontroller has a resolution of 8 bits; this means that the minimum change in voltage detectable will be;

$$V_{r\,min} = \frac{V}{2^{BitDepth}} = \frac{5\,V}{2^{8}} = 19.55\,mV$$

Now using our simplified formula, we can calculate the first value over 0 ohm that will be detected by the design

$$\Delta R_c \simeq \frac{R}{V}\,\Delta V_r = \frac{20\,ohm}{5\,V}\,19.55mV = 0.0782\,ohm$$

We can assume an approximate precision of 100 mΩ.

# Chapter 3. Design and manufacturing

## 3.1 Design environment

### *3.1.1 Altium Designer*

Altium Designer[2] is a software suite specifically intended for the design of printed circuit boards.

This was the preferred software at the electronics department at AWSensors and therefore it is the one that I chose to learn.

It is a very powerful tool, as it combines schematic capture with PCB layout, routing, component management, simulation and manufacturing outputs for the PCB's

### *3.1.2 Workflow*

The work flow, in my case, was as follows.

After the design proposal, the components that were needed for the design were checked for availability in electronics suppliers such as Mouser or Digikey, in the case that a component was not available a substitute component had to be determined.

When all the components needed were checked for availability, the footprints and schematic symbols for these components were downloaded from the website SnapEDA;[3] in the case that a symbol or footprint was not available, it had to be done manually.

In our Altium project we create schematics and PCB libraries in which we place these files, and we can pair them letting the program know which symbol corresponds to which footprint.

A schematic symbol is an abstraction of a component, with its connections; it is used when designing a schematic diagram, which is itself an abstraction of a real circuit.

The footprint ties the schematic symbol to real dimensions and pin placements of the actual integrated circuit placed on the PCB.

This allows for the program to check if the schematic diagram and the PCB you design match electrically, assuring that the PCB being designed behaves exactly as the circuit is intended to behave.

The following is an example of a schematic symbol and footprint pair for the DIP version of the SN74HC595 shift register:

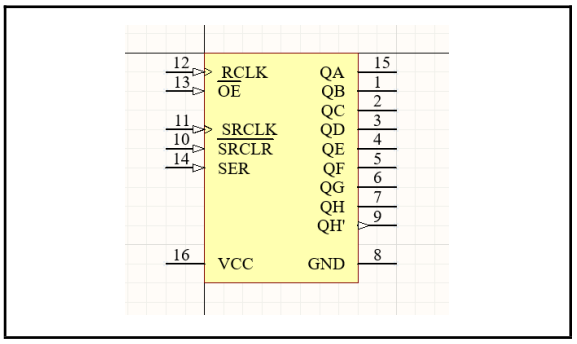


**Figure 12.** Schematic symbol for SN74HC59

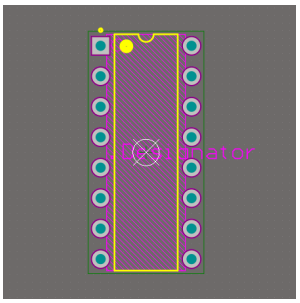

**Figure 13.** footprint for SN74HC595

[2] https://www.altium.com/altium-designer
[3] https://www.snapeda.com/

The next step is the schematic capture; we create a schematic document, and when we open it we access the schematic editor, in which we can place our previously loaded components and connect them in the way that our design specifies.

Once our schematic is completed we can begin to work on our PCB, we create a PCB document and we import the data from our schematic. If all our components are correctly indexed and all of our footprints are in order, we will get no errors.

We can also configure design rules, which stipulate parameters such as the minimum separation between tracks and the minimum track width. These rules are configured in order to follow the guidelines for a specific PCB fabrication standard, and will allow the program to warn us during the design of the PCB in the case that we violate one of the rules.

In the PCB editor will see our board with all the components placed in a straight line, not connected to anything, there will be gray lines, called rats nests, representing the connections that need to be made in order for the PCB to be electrically equal to the schematic.

The next step is the most time consuming in the design of the PCB: component placement and routing.

The components have to be placed in a way that the routing is the simplest possible, using the rats nests we can try to place the components in a way that these cross the least amount of times, although in practice the rats nests will most certainly overlap.

Once the components are placed, the connections have to be made; this is done by the use of tracks, which represent copper lines within the PCB. Depending on the type of PCB that we are designing we will be able to use 2, 4 or more layers to draw said tracks.

Tracks in different layers can be connected by the use of *vias*, which are drill holes in the pcb coated with copper to connect the different layers.

Once all of this is done we can perform a check to see if all the connections have been made and all the rules have been followed, if this check turns out positive, we will know that our PCB is correctly designed and routed.

We can also add labels for the components and text to indicate the function of different components. Labels and text will be printed on top of the board during fabrication.

In Figures 14 and 15 is an example of a schematic with its corresponding PCB document respectively.

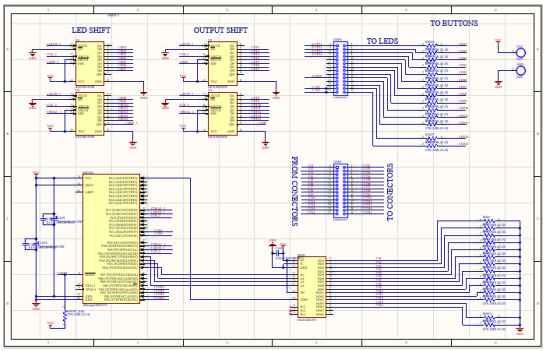


**Figure 14.** Schematic example

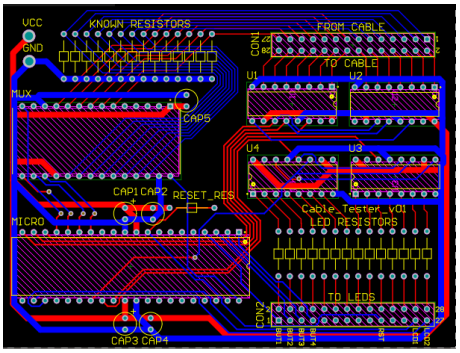


**Figure 15.** PCB example

The tracks can be seen in blue and red, corresponding to the 2 layers in this particular design.

The symbolic representation of the components can be seen in the schematic, as well as their corresponding footprints on the PCB document.

As can be observed, due to its abstract nature, connections are simpler and more comprehensive on the schematic; this is why we start with a schematic and then design the board.

The final step in the process is generating the fabrication files. There are various standards for these files, but the one that I used is Gerber X2. This is because it is the preferred format used by Eurocircuits,[4] the PCB manufacturer used by AWSensors. These files contain the information for all the layers of the PCB, as well as the position of each component on the board.

## 3.2 First design and motives for redesign

The first design was early in the PCB design learning curve; there are 3 decisions that made this design flawed.

Firstly, the components selected were dual in-line packaged (DIP) format, this format has two parallel rows of pins at each side of the chip for its connections, it is a through-hole design and, due to the large spacing between the pins, is quite large.

This made the board very large, in this case it was 100 × 80 mm.

Another flaw was the lack of high current drivers, which limited the current per circuit to 5 mA severely reducing the precision of the impedance reading.

Finally, the board had only 2 layers for connections, this meant that all the connections between the components including signals and power lines shared the same two layers.

A better alternative is to have 4 layers, using 2 of them for connections and 2 of them exclusively for power; this reduces impedance in the power connections and is better suited for high frequency applications.

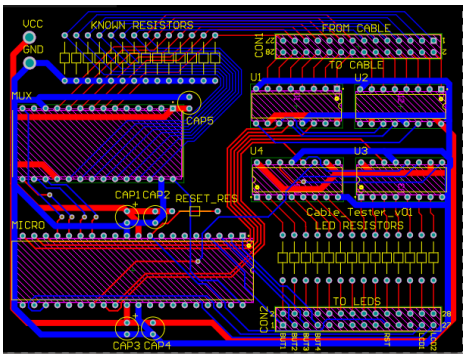


**Figure 16.** PCB document for first design

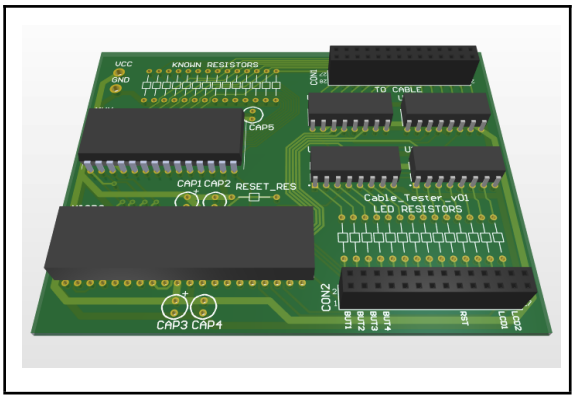


**Figure 17.** *3D representation of first design*

[4] https://www.eurocircuits.com/blog/why-you-should-supply-us-with-gerber-x2-files/

### 3.3 Second design

The second design was expanded with the high current drivers, a 4 layer board, and all components in SMD format.

SMD, or *surface mount device* components are much smaller than DIP components, the distance between their pins is a lot smaller, and these pins are not intended to go through the board, instead they are soldered to pads on the surface of the board

These pads are areas of exposed copper covered with solder; they are determined by the footprints, and are connected to the signal layers within the board

Figure 18 shows a view of the connections within the second design, and an explanation of the different footprints visible.

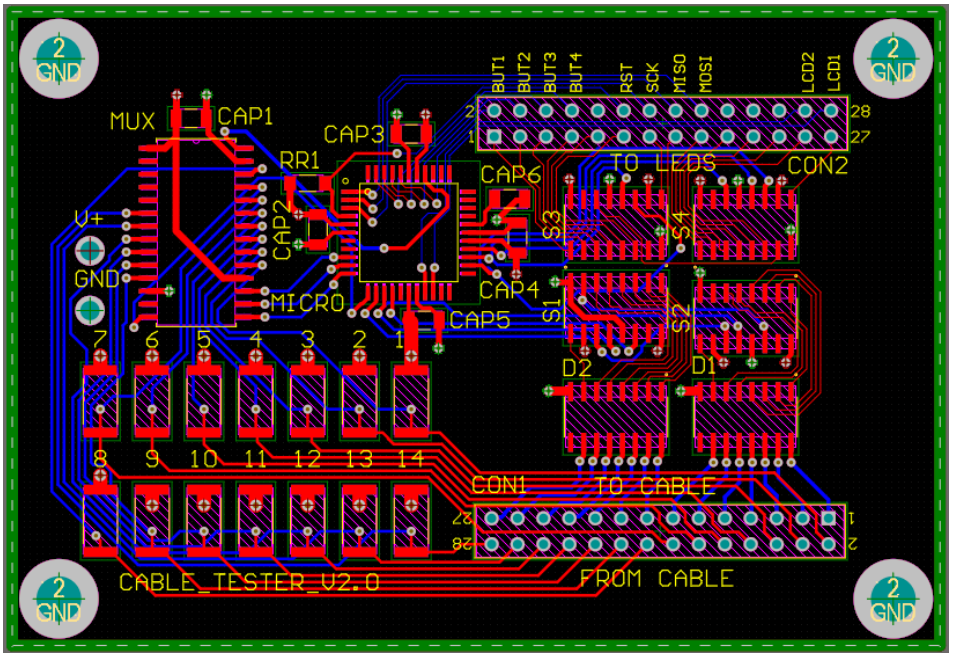


**Figure 18.** PCB document for the second design

The layers that are visible in this representation are the top and bottom layers, top being shown in red and bottom in blue, there are two more layers that are used for power, the ground and 5 V planes, which are not shown in the figure.

All components that need power are connected to these planes through the use of vias.

At the top right and bottom right we can see two connectors, labeled CON2 and CON1, these are used to connect the LEDs, the 14-pin cable to be analyzed, as well as programming the microcontroller and some peripheral buttons and LCD protocols.

At the top left corner, we can see a component labeled MUX; this is the analog multiplexer responsible for commuting the signals from the different lines to the analog input to the microcontroller.

The component to the right of the multiplexer, labeled MICRO, is the microcontroller, in our case the ATMEGA1284P, which orchestrates the functioning of all of its neighboring components, feeding signals to the shift registers, collecting data from the multiplexer and executing our code.

Directly to the right of the microcontroller we see 6 footprints which look identical to each other; the top 4 ones, labeled S1 to S4, are the 4 shift registers in our design

S3 and S4 feed the output to the LEDs as can be seen on the label below the connector on top, which reads: “TO LEDS”.

S1 and S2 feed the input signals to the high current drivers, which are the remaining 2 IC’s below labeled D1 and D2

The high current output of the Darlington high-current drivers goes into the connector CON1, specifically the side labeled “TO CABLE”; this current will flow through the cable and come back through the side labeled “FROM CABLE”

This side of the connector is routed to the 14 known resistors seen at the bottom left corner, arranged in two rows, these resistors are then connected to ground.

Between the resistors and the cable lines from CON1 there are 14 lines connected to the analog multiplexer seen in blue; these close the circuit for the measurement.

The four large drill holes seen on the corners of the board are placed to secure the board with screws in an enclosure. They are also connected to ground; in the case that the enclosure is metallic this will serve as a larger ground mass.

FIgure 19 shows  a 3D representation of what the board would look like with all the components soldered

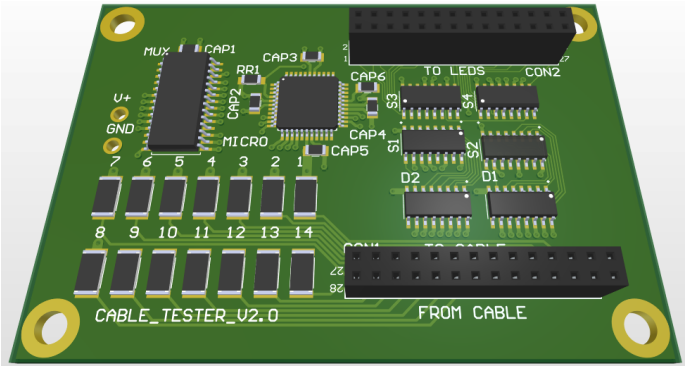


**Figure 19.** ***3D representation of the final design***

## 3.4 Board manufacture and soldering

### *3.4.1 Eurocircuits*

Eurocircuits is a European manufacturer specialized in printed circuit boards.

It has gained a reputation for manufacturing and assembling high quality PCBs.

In our case, we will not require their assembly services as we will do it ourselves. As for manufacturing, they have a very capable and user-friendly online platform where you can upload and check PCB designs for errors.

Using the online platform you can also get an instant quotation of your design, and there are design rule templates available for their different fabrication standards that can be used to configure the design rules for the design environment.

They can supply low volume orders, in our case we only ordered 2 PCBs, and they do it fairly quickly: in our case it took 10 days from uploading the design and placing the order to receiving the manufactured PCBs

Figure 20 shows the naked PCB as received from the manufacturer

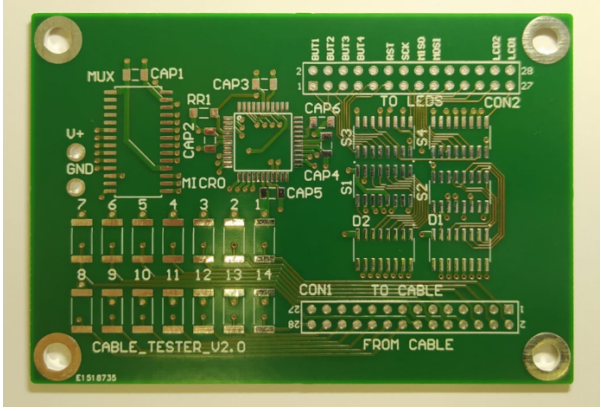


**Figure 20. PCB from Eurocircuits**

### *3.4.2 Soldering*

The two large connectors are through-hole components, all the rest are surface mount components.

The board shown previously is quite small, 6 × 9 cm; this puts into context the tiny distances between the pins of the SMD components.

The green top-most layer is the solder mask, which protects the tracks from corrosion and also protects the pads from short circuits, as it repels solder.

The most critical component on the board is the microcontroller, the spacing between the pads of this IC is the smallest, a mere 0.254 mm.

But with a thin point soldering iron and solder of a thin diameter the ICs were soldered to the board. Coworkers at AWSensors informed me of the correct procedure to solder SMD components, as I had never done it.

At first it seemed difficult, but with a bit of practice it turned into a trivial affair.

Figures 21 and 22 show images of the board at different stages in the soldering process; first the ICs were soldered (Figure 21), and then the discrete SMD components (Figure 22).

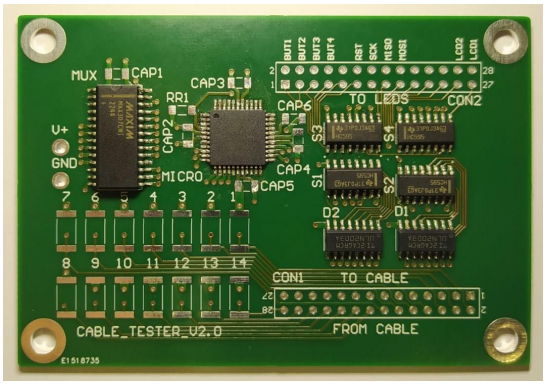


**Figure 21.** Soldering the SMD ICs

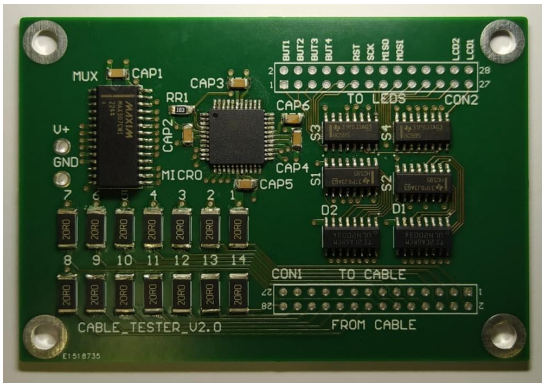


**Figure 22.** Soldering the discrete SMD components

Finally, the through-hole connectors were soldered. We can now compare the manufactured and assembled board (Figure 23) to the 3D render (Figure 24) that we made to predict the board's appearance.

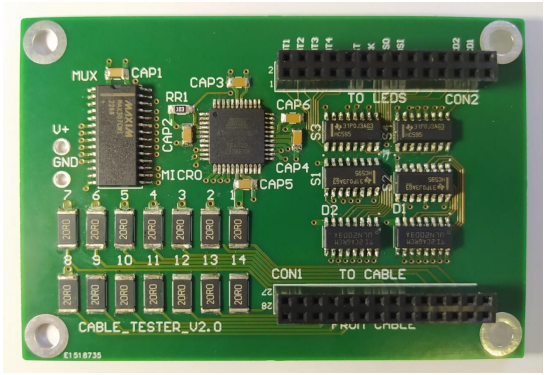


**Figure 23.** Real assembled PCB

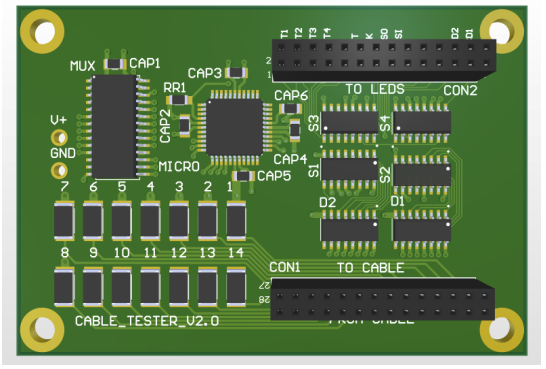


**Figure 24.** Simulated assembled PCB

# Chapter 4. Software and troubleshooting

## 4.1 Programming environment and hardware

### *4.1.1 Hardware*

Now that the board is assembled we have the final physical product, but the board is a mere arrangement of components; without software it will do absolutely nothing, we must program it.

The main control of our device is the microcontroller; for the device to do the desired task, we must program the microcontroller to drive all of its neighboring circuits to perform the tasks that we previously described.

This is done with the use of a special tool designed to program ATMEL microcontrollers such as the atmega1284P-PU that we are using. This tool is called USBtinyisp: it is a USB in-system programmer (ISP) that connects to 4 pins on the microcontroller.

The 4 programming pins of the microcontroller were routed to 4 connections at the top connector of the board for easy programming.

Figure 26 shows the programmer connected to the board and Figure 25 shows the programming pins in the connector.

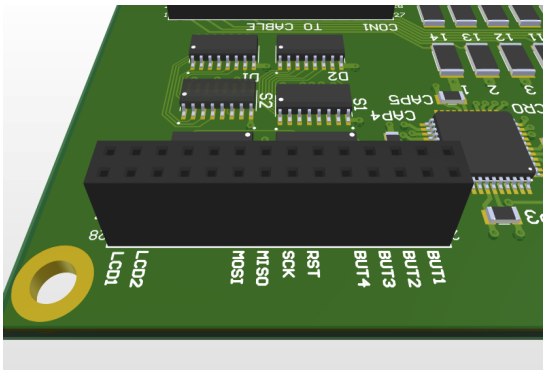


**Figure 25.** Programming pins detail

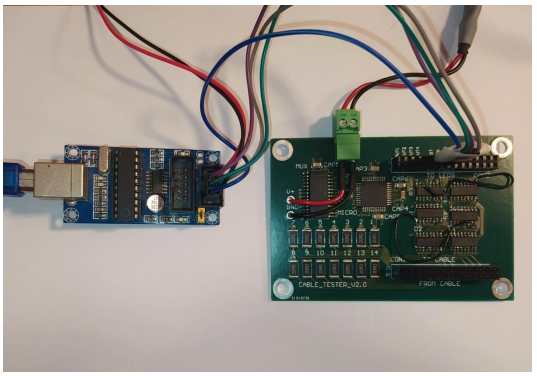

**Figure 26.** USBtinyisp connected to board

The connections labeled MISO, MOSI, SCK and RST are the programming pins, which correspond to the blue, green, purple and grey cables from the USBtinyisp programmer.

The black and red cables are the 5V and GND connections.

### *4.1.2 Software*

The software used for the programming and compiling is a programming suite called Atmel Studio,[5] it specializes in Atmel microcontrollers, so that the programs can be specifically compiled for each microcontroller.

Paired with a tool to write to the microcontroller with the ISP called avrdude[6] we could code, compile and write to memory.

The tool avrdude is a simple command line program that can be used to write programs into the chip's flash memory; within the programming environment of Atmel Studio we can add a

[5] https://microchipdeveloper.com/atstudio:studio7intro
[6] https://www.nongnu.org/avrdude/

function that uses avrdude, so that we can compile a program and instantly send it to the microcontroller.

## 4.2 First tests and troubleshooting

### *4.2.1 JTAG and fuses change*

A good first test on the board would be to write a simple program that turns all the digital pins on and off every second.

This test will give us a lot of information; firstly, it will tell us if the microcontroller is being programmed properly.

Secondly, it will tell us if the solder joints on the IC pins are good, as we can check with a multimeter if the signals are propagating properly.

The first thing to look out for is if the programmer detects the microcontroller and writes properly to it, after every write the programmer reads the microcontroller's memory to check if the write was successful, and relays the information to the user.

When the test was performed, the microcontroller was detected and the program was written properly, and the microcontroller started outputting the signals.

There was a problem though; 4 pins from the C port of the microcontroller (PC2 through PC5) were not outputting any signal.

PC5 is used for the clock signal for the shift registers that drive the LEDs in our design, so this pin not working would mean that the LEDs would not work.

After a bit of reading it was determined that the problem was in the microcontroller's fuses; these are reserved bytes of memory that change the functioning mode of the microcontroller.

One of the functions that was activated was the JTAG function, JTAG is a standard for debugging and testing,[7] but in our case it will not serve any purpose, and it coincidentally uses the 4 pins that were not functioning.

To change the fuses we need to access avrdude, which is the tool used to write to the microcontroller, and input a special command intended to change such fuses.

With the help of an online tool called AVR[8] Fuse Calculator[9] we can input the current values for our fuses, check which functions are enabled, change the functions, and then get the new values for the fuses to be changed.

Figure 27 shows a command window; the first line "`avrdude:safemode: Fuses OK (E:FF, H:99, L:62)` " shows the values for the fuses currently in the microcontroller

The following is a command that changes the H or high fuse from 99 to D9, which is the value previously calculated to change the JTAG function whilst leaving the others unchanged.

After this change, all of the pins worked properly.

[7] JTAG (named after the Joint Test Action Group which codified it) is an industry standard for verifying designs and testing printed circuit boards after manufacture (Wikipedia article for JTAG).
[8] AVR designates the family of microcontrollers developed by Atmel.
[9] https://www.engbedded.com/fusecalc/

```
avrdude: safemode: Fuses OK (E:FF, H:99, L:62)

avrdude done.  Thank you.


C:\Users\manel>avrdude -c usbtiny -p m1284p -b 19200 -U hfuse:w:0xD9:m

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.02s

avrdude: Device signature = 0x1e9705 (probably m1284p)
avrdude: reading input file "0xD9"
avrdude: writing hfuse (1 bytes):

Writing | ################################################## | 100% 0.02s

avrdude: 1 bytes of hfuse written
avrdude: verifying hfuse memory against 0xD9:
avrdude: load data hfuse data from input file 0xD9:
avrdude: input file 0xD9 contains 1 bytes
avrdude: reading on-chip hfuse data:

Reading | ################################################## | 100% 0.01s

avrdude: verifying ...
avrdude: 1 bytes of hfuse verified

avrdude: safemode: Fuses OK (E:FF, H:D9, L:62)

avrdude done.  Thank you.
```

**Figure 27.** Fuse change with avrdude

### *4.2.2 Shift register trouble*

After this initial test the shift registers were tested, once again by writing a simple program that would write a 01010101010101 pattern to their output.

14 LEDs were connected to connector two's "TO LEDS" side, and then each one of them connected each to a 1 kΩ resistor, and then to ground, all with the use of a protoboard.

Instead of getting the desired on off on off pattern on all of the leds, the last seven would be either turned all on or all off, which meant that the second shift register was not working properly.

After reviewing the code many times and not finding the error a hardware design mistake was considered.

Reviewing the circuit and the connections to and between the shift registers a mistake was found.

Figure 28 shows a detail of the connections for the shift registers.

As can be seen, the shift registers have 8 outputs, labeled QA through QH, but QH is duplicated.

This is designed so that you can use the extra QH output to connect multiple registers in series, in our case we have connected 2, the procedure to load the data to the registers is as follows.

While the CLK signal cycles, every cycle a bit of data is written to the SERIAL input of the register, the first bit that enters goes to the QA position, but as bits come after they move all of the other bits a position ahead.

When a bit reaches position QH it goes through the LOOP wire to position QA of the second register, which shares the CLK signal with the first.

Once the 16 bits are written, the LATCH signal is turned on, outputting the shift registers data to the outputs.

For this to work, one of the QH outputs is affected by the latch, but the other one has no latch; that pin's data changes as the data is entered, so that it can be looped to the next register.

When designing the circuit the QH pin that was used for the LOOP signal was the one with the latch, in other words, data could not loop to the second register, this is why the pattern only appeared on the first 7 bits.

This posed a significant problem, the design was already manufactured, the mistake was now permanently written in copper and resin.

A solution was suggested by my tutor: the silk covering the PCB tracks can be scraped off, and the copper from the track cut and lifted from the board, in effect opening the tracks circuit.

And, after this, the correct connection could be made by soldering a thin wire between the correct pins in the shift registers.

With the aid of a sharp cutter, a bench magnifying lens and a bit of patience this was the procedure followed.

Figures 29 and 30 show both the wires connecting the pins and the areas where the silk was scraped and the copper cut.

This fixed the problem and the shift registers were now properly working.

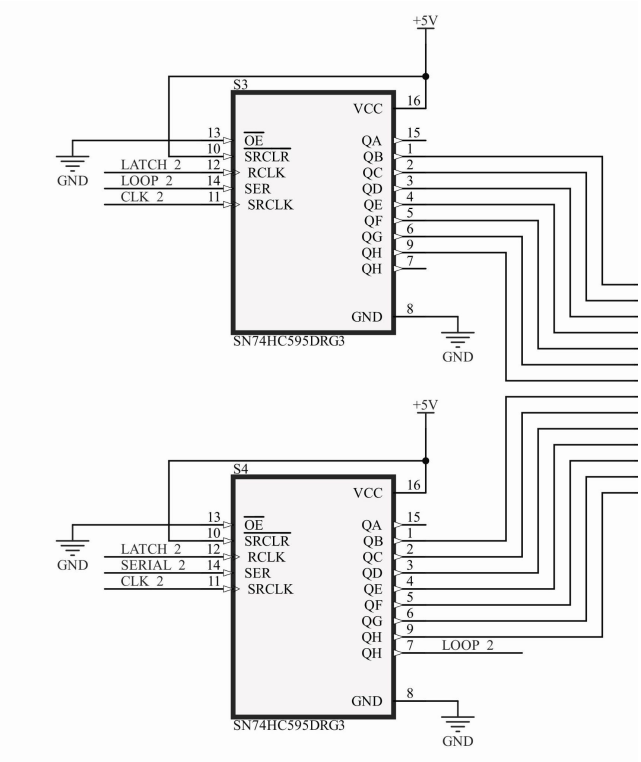


**Figure 28.**

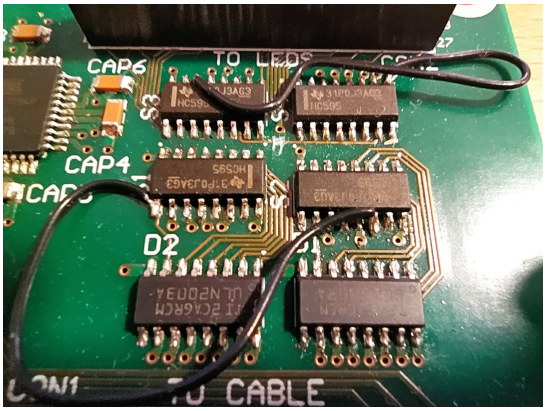


**Figure 29. cables connecting the correct pins**

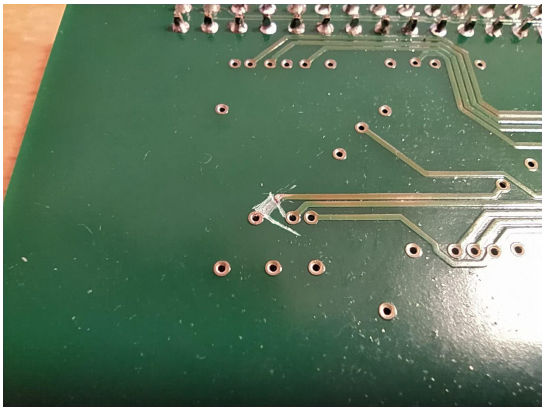

**Figure 30. incorrect line connection broken by cutting it**

### *4.2.3 High-current drivers*

With the shift registers and the microcontroller working, we went ahead to test the high current drivers.

This test is very simple; the drivers sole function is to close the circuit formed by the known resistors and the connections, forwarding the current to ground.

To test for the correct functioning of this, we simply connect a resistor at connector 1, from the *to* cable side to the *from* cable side, mimicking a cable with a high impedance.

When the input signal for the Darlington pair is +5V this circuit should open, letting current flow through the resistor, and so a voltage drop will be measured at said resistor, but not so if the circuit is open.

The function is checked at all of the 14 connections to make sure that all of the signals are routed properly in the circuit.

After this test and the previous test, we can assume that our circuit is working as intended in the design.

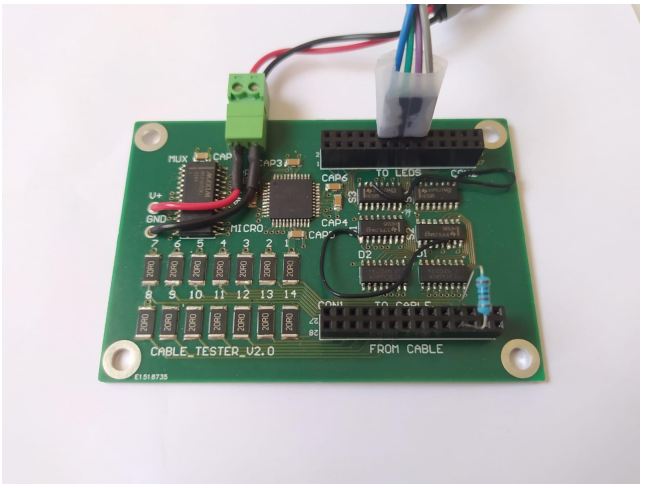


**Figure 31.** Testing of the Darlington high current drivers

# Chapter 5. The program

Now that all the individual tests are done comes the program to implement the mode of operation.

To make things easier a simple setup for the troubleshooting was arranged, 14 leds were set up with 1kΩ resistors in series, and using jumper cables they were connected to their corresponding position at the "TO LEDS" side of connector 2.

The setup can be seen in figure 32.

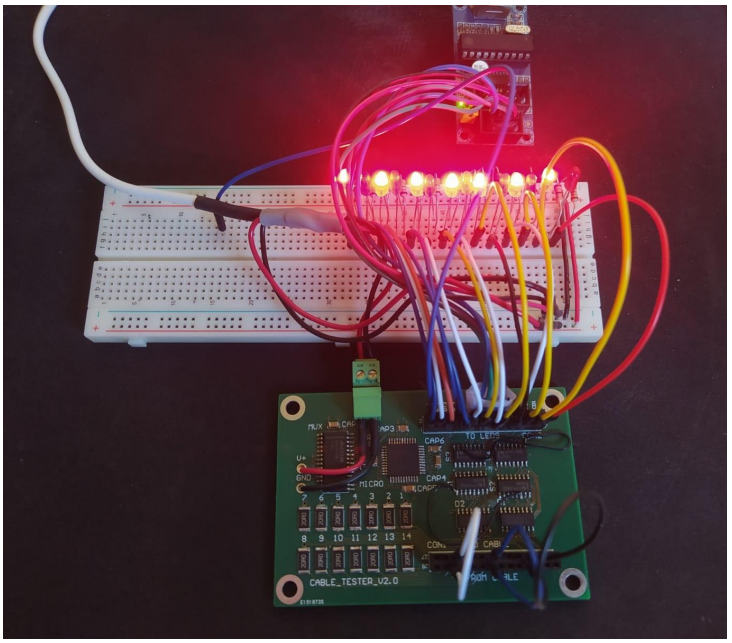


**Figure 32.** Troubleshooting setup for the programming

The topmost board is the ISP programmer, connected to the programming pins in connector 2.

In between is the breadboard used for the LEDs, which are themselves connected using jumper wires to their corresponding positions at connector 2.

At the bottom of the image connector 1 can be seen, with 3 jumper wires connecting the "TO CABLE" to the "FROM CABLE" sides, simulating connections with no impedance.

## 5.1 The functions

To make the task of implementing the mode of operation easier and more readable many functions are used.

### *5.1.1 SHIFT1Write();*

The first function implemented was the function to send data to the shift registers, this would be used to control the LEDs and the high current drivers.

Given that there are two pairs of shift registers, one for the LEDs and one for the high current drivers, two functions were written, SHIFT1Write and SHIFT2Write, both of them being identical except for the pins they used to output the data.

This function takes a byte of information and sends it serially to the shift register, it controls 3 connections in the shift register, assigned to 3 pins of the microcontroller.

it sends each bit from the given byte serially to the SER (serial) input of the shift register, but does this while pulsing the SRCLK (serial clock) signal, and when all 8 bits are sent, activates the RCLK (retain clock), which acts as a latch signal outputting the sent data to the output pins of the registers.

This function has a couple of auxiliary functions defined for internal use, being SHIFT1Latch and SHIFT1Pulse, the first one activates the previously mentioned RCLK signal of the register and then deactivates it, latching the information to the output.

The second one pulses the SRCLK line, activating it and deactivating it, it is used for every bit sent, acting as a clock signal.

### *5.1.2 SETMUXPIN();*

This is a very simple function designed to control the analog multiplexer, it functions in a way that you simply feed it a line number, such as line 8, and it sends the corresponding bits to the selection line of the multiplexer.

These selection lines control which of the multiplexers 16 inputs is redirected to its COM or common pin, so 4 selection lines have to be set.

The function assigns a 4 bit combination to every line number, and depending on the line selected sends the corresponding combination to the selection pins of the multiplexer, configuring said line to be read.

### *5.1.3 ACTIVATELINE();*

This function is similar to the previous, in the sense that its input is the number of a given line.

This time the function generates two bytes of data, these bytes have all of the bits set to 0 except for 1, the position of this single 1 bit depends on the line that has been selected.

The two bytes are then sent using the SHIFT1Write(); function to the shift registers that control the high current drivers, the 1 bit at the inputs of the high current drivers activates a single line for current to flow through it, the one that has been selected.

### *5.1.4 ANALOGREAD();*

the code for this function is responsible for the analog reading taken by the microcontroller to read the voltage at the voltage divider set by our circuit.

It configures the pin to be read from,in our case PA0, the way in which the data is going to be stored, activates de ADC (analog to digital converter) and then takes the reading, storing it in a special register, designed with this purpose, called ADCH.

The value can then be read from this register, assigning it to any variable, or operating directly with it from the register.

### *5.1.5 Brief explanation*

In order to understand the last 2 functions a bit of context is needed.

In the main part of the program, there are two loops that collect data from the cable, this part of the program will be explained in the next section, but the data that these loops return needs to be understood to understand the next functions

These two loops return a 14 digit array called states[] and a 14 x 14 digit matrix called shorts[][].

states[] contains continuity and impedance information for each of the 14 lines, in the form of 14 digits, which are either 0, 1 or 2, these 3 states represent a good connection, an open circuit and a high impedance connection respectively

shorts[][] is a 14 x 14 matrix that contains information for the shorts, if any, in the cable, it is initially set to all 0.

the rows and columns represent connections to the lines from both sides of the cable, if when sending a current through line 1 a connection is detected in line 3 this means that lines 1 and 3 are short circuited, so a 1 would be written at line 1 column 3 and line 3 column 1.

The diagonal of this matrix is all 1s , corresponding to the good connections between lines that are supposed to be connected, such as row 1 column 1, corresponding to the first connection of the cable.

Now that the data generated by the main code is understood the two last functions can be explained.

### *5.1.6 PRINTSTATES();*

Printstates is a function designed to represent the data in the array states[] using the LEDs

Each state drives the LEDs in a different manner, a good connection lights up its corresponding LED, a bad one turns it off, and a high impedance connection makes it blink.

In order to implement this the data from the integer array states[] has to be converted to bytes that can be sent to the shift registers for the LEDs

This is done by calculating two pairs of bytes, given that we have to blink LEDs in the case that the corresponding line is in the high impedance state, we will need two pairs of bytes, that we can alternate between to create the blinking action

So in the first pair of bytes the good connections will be turned on, the open circuits will be turned off and the high impedance connections will be turned off also, while in the secon pair of bytes everything will be the same except the high impedance byte, which will be on.

after these two pairs of bytes have been calculated, a simple loop sends one pair of bytes, waits 500ms, sends the second pair and waits again, this is repeated until a signal is detected on pin PB0 of the microcontroller, which is connected to a button.

### *5.1.7 PRINTSHORTS();*

PRINTSHORTS(); is similar function to PRINTSTATES(); in the way that it translates data to bytes for the LEDs

The matrix shorts[][] contains a diagonal of 1s representing each good connection, if any row of this matrix has more than one bit set to 1 this means that there is a short between the lines corresponding to the ones shown in said row.

This makes the data very convenient, if any row has more then one bit set to 1 this row can be sent to the LEDs for the user to see, and he can easily identify which lines are short circuited.

If this is the case for more than one row, they can be shown sequentially to the user.

This is exactly what the function does, it checks each row sequentially, if any has more than 1 bit set to 1 then it calculates its two bytes for the shift registers and sends them, displaying the information at the LEDs.

This continues indefinitely until a signal is detected on pin PB1 of the microcontroller, which is connected to another button.

## 5.2 The main program

The previous functions are put to use in the main program, the program consists of two sections, one that performs the readings and calculates the data corresponding to the state of the cable, and another that displays this data to the user.

The first loop, which is labeled “LOOP 1” in its adjacent comment, calculates the states[] array, it activates all of the lines sequentially using the ACTIVATELINE(); function, at each line it also reads the voltage using the ANALOGREAD(); function.

Depending on the reading, it writes to the states[] array a 0, a 1 or a 2, corresponding to a good connection, an open circuit and a high impedance connection respectively.

The second loop labeled “LOOP 2” activates all of the lines sequentially , but this time for every line it reads all of the others, to check if the line is short circuited to any of the others. and stores the information in the shorts[][] matrix.

The last loop labeled “LOOP 3” is simply the two functions PRINTSTATES(); and PRINTSHORTS(); in an infinite loop.

When the device is turned on, after the calculations are done, it enters the PRINTSTATES(); mode, if the user presses button 1, the program exits this state and enters the PRINTSHORTS(); mode, the user can then press button 2 to return to the previous state.

This loop also checks the microcontroller's pin PB2, which is connected to button 3, if the button is pressed, the program jumps to the first section, performing the readings and calculating the data again.

# Chapter 6. Current state of development

While troubleshooting the program all of the functions were individually tested to check if they worked properly.

Placeholder values were given to the shorts[][] matrix and states[] array, so that the functions PRINTSTATES(); and PRINTSHORTS(); ,meant to represent the data, could be checked.

The SETMUXPIN(); function was checked by giving it different line values, then reading the voltage at the multiplexers selection pins, to check if the correct sequence of bits was being applied to the selection pins

The ACTIVATELINE(); function was checked by giving it different line values and connecting a LED between its corresponding connector position and 5V, checking that it was letting current through.

The SHIFTWrite(); functions were checked by giving them different bit patterns and checking if they would show up at the LEDs and at the outputs of the shift registers next to the high current drivers.

All of the above functions worked properly, but the function meant to take the analog reading, ANALOGREAD(); was not working.

When feeding a signal to the analog pin at the microcontroller the reading worked, but when feeding the same signal to the input pins at the multiplexer the reading was always 255, meaning it was detecting the maximum voltage value of 5V.

After inspecting the design and the multiplexers datasheet to make sure that no mistakes were made no conclusive answer has been reached, This leaves two options.

The first option is that the IC is defective, in which case it would need to be replaced by a new one.

The second option is that there is some mistake in the design of the circuit, in which case, if possible, the circuit would need to be fixed, and if not possible, redesigned and manufactured.

Sadly this means that the circuit won’t be able to perform its intended task in this state.

# Chapter 7. Further work and concluding remarks

## 7.1 Further work

In order for this to be a usable product a number of things have to be done, firstly, fixing the issue with the analog multiplexer so that the circuit functions properly.

The next step is to find a fitting enclosure for the board, an enclosure in which the LEDs can be installed, as well as the two connectors needed, a connector for the power supply and the buttons to operate the device.

The circuit was also designed so that further programming could implement an LCD display to show data in a more detailed way, such as exact values for impedance.

A possible enclosure for the project is shown below in figure 33.

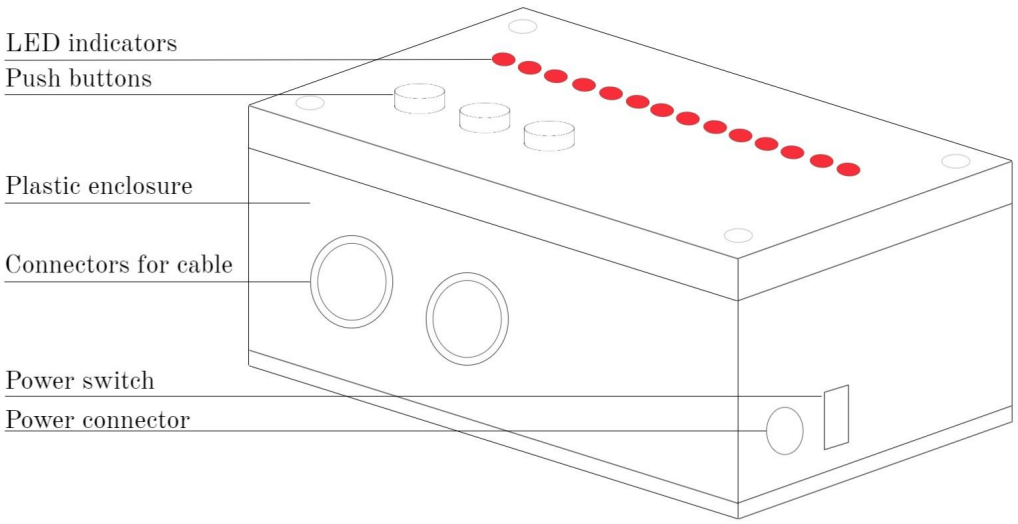


**Figure 33.** Enclosure for the board

## 7.2 Concluding remarks

Even though the design has not been fully realized, the steps necessary to do it should be, with the exception of the trouble with the analog multiplexer, simple.

When fully built, this device will aid in the manufacturing process at AWSensors, with a simple interface and an output that is very easy to interpret, its operation will be swift.

With the proposed enclosure it will also be durable, and given the quality of the connectors used by AWSensors it will have a long service life.

And in the case of a replacement, the cost is small. The most expensive parts are the connectors and the manufacturing of the printed circuit board and the cost is still below 200 euros.

## Appendix: code listing

```
#define F_CPU 1000000UL  //CPU freq set to 1 MHz
#include <avr/io.h>      //include for input output protocols
#include <util/delay.h>  //include for delay functions


#define SHIFT1_PORT   PORTD       //port D to SHIFT1_PORT
#define SHIFT1_DDR    DDRD        //data direction register D to SHIFT1_DDR

#define SERIAL1 PD6                 //Data pin (DS) pin location
(serial in)
#define CLK1 PD4                    //Shift Clock (SH_CP) pin location (clk)
#define LATCH1 PD5                  //Store Clock (ST_CP) pin location
(latch)

#define SHIFT2_PORT   PORTC       //Same for SHIFT2
#define SHIFT2_DDR    DDRC

#define SERIAL2 PC7
#define CLK2 PC5
#define LATCH2 PC6

#define MUXA0 PD0                //define pins for the address lines for
the multiplexer
#define MUXA1 PD1
#define MUXA2 PD2
#define MUXA3 PD3

#define BUTT1 PB3                //define pins for button inputs
#define BUTT2 PB2
#define BUTT3 PB1
#define BUTT4 PB0

#define LCD1 PC1                  //define pins for LCD control
#define LCD2 PC0


#define CONNECTED 0
#define OPEN 1
#define HI_IMPEDANCE 2

//Initialize HC595 System

void SHIFT1init()
{
      //Make the Data(DS), Shift clock (SH_CP), Store Clock (ST_CP) lines
output
      SHIFT1_DDR|=((1<<CLK1)|(1<<LATCH1)|(1<<SERIAL1));
}
void SHIFT2init()
{
      //Make the Data(DS), Shift clock (SH_CP), Store Clock (ST_CP) lines
output
```

```
	SHIFT2_DDR|=((1<<CLK2)|(1<<LATCH2)|(1<<SERIAL2));
}


//Low level macros to change data (DS)lines
#define SHIFT1DataHigh() (SHIFT1_PORT|=(1<<SERIAL1))
#define SHIFT1DataLow() (SHIFT1_PORT&=(~(1<<SERIAL1)))
#define SHIFT2DataHigh() (SHIFT2_PORT|=(1<<SERIAL2))
#define SHIFT2DataLow() (SHIFT2_PORT&=(~(1<<SERIAL2)))

//Sends a clock pulse on SH_CP line
void SHIFT1Pulse()
{
	//Pulse the Shift Clock

	SHIFT1_PORT|=(1<<CLK1);//HIGH

	SHIFT1_PORT&=(~(1<<CLK1));//LOW

}

void SHIFT2Pulse()
{
	//Pulse the Shift Clock

	SHIFT2_PORT|=(1<<CLK2);//HIGH

	SHIFT2_PORT&=(~(1<<CLK2));//LOW

}

//Sends a clock pulse on ST_CP line
void SHIFT1Latch()
{
	//Pulse the Store Clock

	SHIFT1_PORT|=(1<<LATCH1);//HIGH
	_delay_loop_1(1);

	SHIFT1_PORT&=(~(1<<LATCH1));//LOW
	_delay_loop_1(1);
}

void SHIFT2Latch()
{
	//Pulse the Store Clock

	SHIFT2_PORT|=(1<<LATCH2);//HIGH
	_delay_loop_1(1);

	SHIFT2_PORT&=(~(1<<LATCH2));//LOW
	_delay_loop_1(1);
}

void SHIFT1Write(uint8_t data)
{
```

```
	//Send each 8 bits serially

	//Order is MSB first (10010000)
	for(uint8_t i=0;i<8;i++)
	{
		//Output the data on DS line according to the
		//Value of MSB
		if(data & 0b10000000)
		{
			//MSB is 1 so output high

			SHIFT1DataHigh();
		}
		else
		{
			//MSB is 0 so output high
			SHIFT1DataLow();
		}

		SHIFT1Pulse();  //Pulse the Clock line
		data=data<<1;  //Now bring next bit at MSB position

	}

	//Now all 8 bits have been transferred to shift register
	//Move them to output latch at one
	SHIFT1Latch();
}

void SHIFT2Write(uint8_t data)
{
	//Send each 8 bits serially

	//Order is MSB first (10010000)
	for(uint8_t i=0;i<8;i++)
	{
		//Output the data on DS line according to the
		//Value of MSB
		if(data & 0b10000000)
		{
			//MSB is 1 so output high

			SHIFT2DataHigh();
		}
		else
		{
			//MSB is 0 so output high
			SHIFT2DataLow();
		}

		SHIFT2Pulse();  //Pulse the Clock line
		data=data<<1;  //Now bring next bit at MSB position

	}

	//Now all 8 bits have been transferred to shift register
```

```
	//Move them to output latch at one
	SHIFT2Latch();
}



void SETMUXPIN(int pin){                                //sets the mux
control line according to the pin that wants to be read

	PORTD = PORTD & ~(0b00001111);

	if(pin == 1){PORTD=PORTD|(0b00000000);}
	if(pin == 2){PORTD=PORTD|(0b00000001);}
	if(pin == 3){PORTD=PORTD|(0b00000010);}
	if(pin == 4){PORTD=PORTD|(0b00000011);}
	if(pin == 5){PORTD=PORTD|(0b00000100);}
	if(pin == 6){PORTD=PORTD|(0b00000101);}
	if(pin == 7){PORTD=PORTD|(0b00000110);}
	if(pin == 8){PORTD=PORTD|(0b00000111);}
	if(pin == 9){PORTD=PORTD|(0b00001000);}
	if(pin == 10){PORTD=PORTD|(0b00001001);}
	if(pin == 11){PORTD=PORTD|(0b00001010);}
	if(pin == 12){PORTD=PORTD|(0b00001011);}
	if(pin == 13){PORTD=PORTD|(0b00001100);}
	if(pin == 14){PORTD=PORTD|(0b00001101);}
	if(pin == 15){PORTD=PORTD|(0b00001110);}
	if(pin == 16){PORTD=PORTD|(0b00001111);}

}

void ACTIVATELINE(int line){

	uint8_t shiftline[2];
	if(line == 1){ shiftline[0] = 0b00000010; shiftline[1] =
0b00000000;}
	if(line == 2){ shiftline[0] = 0b00000100; shiftline[1] =
0b00000000;}
	if(line == 3){ shiftline[0] = 0b00001000; shiftline[1] =
0b00000000;}
	if(line == 4){ shiftline[0] = 0b00010000; shiftline[1] =
0b00000000;}
	if(line == 5){ shiftline[0] = 0b00100000; shiftline[1] =
0b00000000;}
	if(line == 6){ shiftline[0] = 0b01000000; shiftline[1] =
0b00000000;}
	if(line == 7){ shiftline[0] = 0b10000000; shiftline[1] =
0b00000000;}
	if(line == 8){ shiftline[0] = 0b00000000; shiftline[1] =
0b00000010;}
	if(line == 9){ shiftline[0] = 0b00000000; shiftline[1] =
0b00000100;}
	if(line == 10){ shiftline[0] = 0b00000000; shiftline[1] =
0b00001000;}
	if(line == 11){ shiftline[0] = 0b00000000; shiftline[1] =
0b00010000;}
	if(line == 12){ shiftline[0] = 0b00000000; shiftline[1] =
```

```
0b00100000;}
      if(line == 13){ shiftline[0] = 0b00000000; shiftline[1] =
0b01000000;}
      if(line == 14){ shiftline[0] = 0b00000000; shiftline[1] =
0b10000000;}


      SHIFT1init();
      SHIFT1Write(shiftline[0]);
      SHIFT1Write(shiftline[1]);
      _delay_ms(500);

}

void ANALOGREAD(int line){

      PORTD = PORTD & ~(0b00001111);  //set selection bits to 0
      //PORTD = PORTD | (0b00001111);  //set selection bits of the analog
MUX to NO16
      SETMUXPIN(line);

      DDRA = 0b00000000; //set port A as input
      ADMUX = 0b01100000; //set ADC to be left justified AVCC as reference
and pin ADC0
      ADCSRA = 0b10000111; //enable ADC and set PRESCALER to max value of
128

      ADCSRA = ADCSRA | (1<<ADSC); //start conversion
      while(ADCSRA & (1<<ADSC)); // wait until ADSC is clear

    //VALUE IS NOW STORED IN ADCH // //led_pattern[0] = ADCH; (example to
extract value)
}

void PRINTSTATES(int * states){                                  //THIS
FUNCTION CONVERTS THE states[14] ARRAY TO ITS CORRESPONDING BYTES FOR THE
LEDS, AND THEN SENDS THEM TO THE LEDS

      uint8_t Mask =0b00000010;  /* LSB not used */
      uint8_t Blink1A = 0b00000000;
      uint8_t Blink2A = 0b00000000;
      for (int i = 0; i < 7; i++) {
            if (states[i]==CONNECTED) {Blink1A|=Mask; Blink2A|=Mask; }
            /* if (states[i]==OPEN)  do nothing */
            if (states[i]==HI_IMPEDANCE) {Blink1A|=Mask; }
            Mask = Mask << 1;
      }
      Mask = 0b00000010; /* LSB not used */
      uint8_t Blink1B = 0b00000000;
      uint8_t Blink2B = 0b00000000;
      for (int i = 7; i < 14; i++) {
            if (states[i]==CONNECTED) {Blink1B|=Mask; Blink2B|=Mask; }
            /* if (states[i]==OPEN)  do nothing */
            if (states[i]==HI_IMPEDANCE) {Blink1B|=Mask; }
            Mask = Mask << 1;
      }
```

```
	SHIFT2init();
	while(1){
		SHIFT2Write(Blink1A);
		SHIFT2Write(Blink1B);

		_delay_ms(500);


		SHIFT2Write(Blink2A);
		SHIFT2Write(Blink2B);

		_delay_ms(500);
		if ( (PINB & (1 << BUTT1)) == (1 << BUTT1) ){continue;}
	}
}

void PRINTSHORTS(int shorts[14][14]){
	int x = 0;
	while (1){

		int sum = 0;
		for(int y = 0; y <= 13; y++){
			sum = sum + shorts[x][y];
	    }
		if(sum == 1){continue;}

		uint8_t Mask =0b00000010;  /* LSB not used */
		uint8_t ShortsA = 0b00000000;
		uint8_t ShortsB = 0b00000000;
		for (int i = 0; i < 7; i++) {
			if (shorts[x][i]==1) {ShortsA|=Mask;}
			/* if (shorts[i]==0)  do nothing */
			Mask = Mask << 1;
		}
		for (int i = 7; i < 14; i++) {
			if (shorts[x][i]==1) {ShortsB|=Mask;}
			/* if (shorts[i]==0)  do nothing */
			Mask = Mask << 1;
		}


		SHIFT2init();
		SHIFT2Write(ShortsA);
		SHIFT2Write(ShortsB);

		_delay_ms(2000);
		x++;
		if(x>13){x=0;}
		if ( (PINB & (1 << BUTT2)) == (1 << BUTT2) ){continue;}


	}
}
void main()
{
```

```
	start:

	DDRD = 0b11111111;		//set MUX and SHIFT1 pins as
outputs
	DDRC = 0b11111111;


	int i = 1;
	int states[14];
	while(i <= 14){					//(LOOP
1)THIS LOOP TESTS FOR THE STATE OF EACH LINE, BE IT HIGH IMPEDANCE, NO
IMPEDANCE OR OPEN CIRCUIT
		ACTIVATELINE(i);				//AND
STORES THE INFORMATION IN THE ARRAY CALLED states[14];
		ANALOGREAD(i);
		// voltage value is now stored in ADHC
		uint8_t VOLTAGE = ADCH;

		if(VOLTAGE < 128){states[i-1] = 0;} // no impedance aka good
		if(VOLTAGE > 128){states[i-1] = 1;} // open circuit
		else {states[i-1] = 2;} // high impedance

		i++;
	}

	int shorts[14][14] = {0};
	i = 1;
	while(i <= 14){					//THIS LOOP
CHECKS FOR SHORTS BETWEEN THE LINES AND STORES THE INFORMATION IN THE
MATRIX CALLED shorts[14][14]
		int j = 1;
		while(j <= 14){
			//if(i==j){continue;}
			ACTIVATELINE(i);
			ANALOGREAD(j);
			uint8_t VOLTAGE = ADCH;
			if(VOLTAGE != 255){shorts[i-1][j-1] = 1;
shorts[j-1][i-1] = 1;}
			j++;
		}
		i++;
	}

	while(1){
		PRINTSTATES(states);
		PINTSHORTS(shorts);
		if ( (PINB & (1 << BUTT3)) == (1 << BUTT3) ){goto start;}
	}
}
```