



**CHALMERS**  
UNIVERSITY OF TECHNOLOGY

---



# Industrial circuit board design and microprocessor programming

Bachelor of Science thesis in Mechatronics

FREDRIK HÖGBERG

---

Department of Signals and Systems  
CHALMERS UNIVERSITY OF TECHNOLOGY  
Gothenburg, Sweden 2017



REPORT NO. XXXX/XXXX

# Industrial circuit board design and microprocessor programming

FREDRIK HÖGBERG

Department of Signal and Systems  
CHALMERS UNIVERSITY OF TECHNOLOGY  
Gothenburg, Sweden 2017





Industrial circuit board design and microprocessor programming  
FREDRIK HÖGBERG

© FREDRIK HÖGBERG, 2017

Technical report no xxxx:xx  
Department of Signals and Systems  
Chalmers University of Technology  
SE-412 96 Göteborg  
Sweden  
Telephone + 46 (0)31-772 1000

Cover:

Hot Screen 2000 is an industrial heat press machine that can be used to attach screen prints on clothing. This thesis will go in to details on how the circuit board and program was made for this machine.

Chalmers Reproservice  
Göteborg, Sweden 2017

# Abstract

The main object of this project is to design and make circuitry to control a industrial heat press machine. The circuitry consist of a 2 layer board containing a micro controller, display module, AC/DC module, temperature controller and electronics to control electromechanical devices.

The frame of reference will go through the basics steps in designing and manufacturing electrical circuitry with digital and analog technology and explain the basics of the most common electrical components used in circuitry today. The frame of reference will also show how to create schematic and circuit design using KiCAD and how to program a Microchip microprocessor in the C programing language.

This knowledge will then be implemented in the circuit design of the industrial heat press machine. The implementation will provide useful information about common practises but also pitfalls to avoid.

# Sammanfattning

Huvudsyftet med projekt är att designa och producera kretskort för att styra en industriell värmepressmaskin. Kretskortet består av en 2-lagerskiva som innehåller en mikrokontroller, displaymodul, AC / DC-modul, temperaturregulator och elektronik för att styra elektromekaniska anordningar.

Referensramen kommer att gå igenom grunderna i design och tillverkning av elektriska kretsar med digital och analog teknik. Detta avsnitt kommer att förklara grunderna för de vanligaste elektriska komponenterna som används i kretsar idag. I det här avsnittet avhandlas hur man skapar schemaritning och kretskortslayout i KiCAD. Vidare beskrivs hur man kan programmera en Microchip mikroprocessor i programmeringsspråket C. Sist beskrivs även processen för etsnings av kretskort samt placering och lödning av komponenter med hjälp av ytmonteringsteknik.

Kunskapen kommer implementeras i kretskortsdesignen för den industriella värmepressmaskinen. I Genomförandet används vanligt förekommande tekniker och metoder men ska också visa på fallgropar som bör undvikas.

# Table of Contents

|   |           |
|---|-----------|
| <b>1. INTRODUCTION.....</b>                                 | <b>12</b> |
| 1.1. Background.....  | 12        |
| 1.2. Purpose and goal.....                                  | 12        |
| 1.3. Delimitation.....                                      | 13        |
| <b>2. METHODOLOGY.....</b>                                  | <b>14</b> |
| <b>3. FRAME OF REFERENCE.....</b>                           | <b>15</b> |
| 3.1. Electrical component theory.....                       | 15        |
| 3.1.1. Resistors and the basics laws of electricity.....    | 15        |
| 3.1.2. Capacitor and the complex number representation..... | 16        |
| 3.1.3. Inductor.....  | 17        |
| 3.1.4. Diode.....   | 17        |
| 3.1.5. Bipolar Junction Transistor (BJT).....               | 18        |
| 3.1.6. Field Effect Transistor (FET).....                   | 19        |
| 3.1.7. Triode for alternating current (TRIAC).....          | 20        |
| 3.1.8. Opto-isolator.....                                   | 20        |
| 3.1.9. Microcontroller Microchip PIC16F877A.....            | 21        |
| 3.1.10. Displayinterface HD44780.....                       | 22        |
| 3.1.11. Pneumatic system and control.....                   | 23        |
| 3.1.12. Part suppliers and manufacturers.....               | 23        |
| 3.2. Programming the PIC16F877A in C.....                   | 24        |
| 3.2.1. GNU Emacs Editor.....                                | 24        |
| 3.2.2. Compiler tools.....                                  | 25        |
| 3.2.3. Microcontroller setup.....                           | 26        |
| 3.3. Introduction to KiCAD.....                             | 27        |
| 3.4. Schematics design in KiCAD.....                        | 29        |
| 3.4.1. Adding a component.....                              | 29        |
| 3.4.2. Creating a component.....                            | 29        |
| 3.4.3. Connect the components.....                          | 30        |
| 3.4.4. Annotate the components.....                         | 31        |
| 3.4.5. Design rules and validation.....                     | 31        |
| 3.5. Generate netlist in KiCAD.....                         | 32        |
| 3.6. Generate BOM in KiCAD.....                             | 32        |

|  |           |
|--|-----------|
| 3.7. Circuit Board Layout in KiCAD.....  | 32        |
| 3.7.1. Design rules and validation.....  | 35        |
| 3.7.2. 3D viewer.....  | 35        |
| 3.7.3. Making footprints.....  | 35        |
| 3.8. Generating production files.....  | 35        |
| <b>4. IMPLEMENTATION.....</b>  | <b>37</b> |
| 4.1. Creating new software written in C that will run on already produces machines.....                      | 37        |
| 4.2. Find out why 15% of current production run do not pass test and verification and correct the issue..... | 37        |
| 4.3. Create the schematics for the new circuit design.....   | 37        |
| 4.4. Design the PCB layout of the new circuit design.....  | 38        |
| 4.5. Port the C program so it will run on the new circuitry.....   | 38        |
| 4.6. Make a prototype of the new design.....   | 39        |
| 4.7. Test and verify the function of the prototype.....  | 41        |
| <b>5. RESULT.....</b>  | <b>42</b> |
| 5.1. The program.....  | 42        |
| 5.1.1. Setup.....  | 42        |
| 5.1.2. LCD driver.....   | 44        |
| 5.1.3. Interrupt function.....   | 44        |
| 5.1.4. Destroying eeprom memory.....   | 46        |
| 5.2. Previous production hardware update.....  | 46        |
| 5.2.1. Stable boot in previous design.....   | 46        |
| 5.2.2. Noise reduction of AC/DC transformator in previous design.....  | 46        |
| 5.2.3. Modification of the ground plane in previous design.....  | 46        |
| 5.3. The new design.....   | 46        |
| 5.3.1. Schematic design.....   | 48        |
| 5.3.2. PCB layout.....   | 49        |
| 5.4. The prototype.....  | 50        |
| 5.5. The 3D model of an mid and entry level machine.....   | 51        |
| <b>6. ANALYSIS AND DISCUSSION.....</b>   | <b>54</b> |
| 6.1. Choosing a (FOSS) development environment.....  | 54        |
| 6.1.1. Choosing editor.....  | 54        |
| 6.1.2. Microchips biggest mistake.....   | 54        |
| 6.1.3. KiCAD.....  | 55        |

|  |           |
|--|-----------|
| 6.2. The conservative approach.....                          | 55        |
| 6.2.1. Keeping the old processor.....                        | 55        |
| 6.3. The clean sheet approach.....                           | 55        |
| 6.3.1. Going Atmel.....                                      | 55        |
| 6.3.2. The pitfalls of modern displays and technologies..... | 55        |
| 6.4. Thought about a entry level machine.....                | 56        |
| <b>7. CONCLUSION.....</b>                                    | <b>57</b> |
| 7.1. Choosing a suitable project for your first PCB.....     | 57        |
| 7.1.1. The old school project.....                           | 57        |
| 7.1.2. FOSS will kickstart your microcontroller.....         | 57        |
| 7.1.3. Going professional.....                               | 57        |
| 7.2. Recommendations on how to get started.....              | 57        |
| 7.2.1. Tutorials.....  | 57        |
| 7.2.2. KiCAD video tutorial.....                             | 57        |
| 7.2.3. Using a reference design.....                         | 58        |
| 7.2.4. How to learn the C program language.....              | 58        |
| 7.2.5. Datasheets and manuals.....                           | 58        |
| 7.2.6. Getting help.....                                     | 59        |
| <b>REFERENCES.....</b>                                       | <b>60</b> |



# Preface

This thesis was written at SVENSK Elektronikproduktion WearOne AB in Kungsbacka, Sweden during the spring of 2017.

A special thanks to Jörgen Andresson CEO at SVENSK Elektronikproduktion WearOne AB who insisted to start this project before Hotscreen AB was on board, making the project possible to finish before exams. I would like to thank Peter Skörvald for giving valuable advice on PCB design and layout. I also would like to thank the staff at the production floor for the advice on how to improve the manufacturability of the design.

Fredrik Högberg

Göteborg, June 2017





# 1. Introduction

Printed Circuit Boards PCB together with integrated Circuits (IC) are key technologies that made the computing, automation and internet era possible. Today almost all companies and professions do use products containing these technologies in their daily work to increase productivity, profit and remove dangerous and unhealthy work tasks. This evolution has developed into new fields in science and greatly improved and helped other fields, this makes knowledge in these technologies still very desirable.

## 1.1. Background

Hot Screen AB is a Swedish company that makes and sells screen prints for textile clothing and fabrics. To apply these prints on textile it's common to use a heat press machine, which is why Hotscreen do also sell heat press machines. There is two different machines available, the Hot 2000 and the Hot 4000 which are both produced at SVENSK Elektronikproduktion WearOne AB in Kungsbacka, Sweden.

The existing design have been around for a while and components is getting hard to find. Over the years some minor flaws have been found and new ideas of improvement have been suggested. The original designer will not cooperate anymore and the design files and source code are not fully available. Hotscreen AB wish to address these problems by developing a new circuit design and new software. SVENSK Elektronikproduktion WearOne AB will make this new circuit and software design.

## 1.2. Purpose and goal

The main purpose of this project is to develop a new circuitry and software design that is compatible with the current machines Hot 2000 and Hot 4000. The design will be produced and verified as a prototype. The project also aims to correct flaws in already made machines by creating and upgrading to new software. There is also problems in the current production run that must be addressed and will be part of this project.

The project consist of the following goals:

- Create new software written in C which will be run on previously manufactured machines.
- Find out why 15% of current production run do not pass test and verification and correct the issue.
- Create the schematics for the new circuit design.
- Design the PCB layout of the new circuit design.
- Port the C program so it will run on the new circuitry.
- Make a prototype of the new design.
- Test and verify the function of the prototype.
- Make suggestion of a new entry level machine by makeing 3D CAD model of it mechanical design.

### **1.3. Delimitation**

The project will only deliver at prototype of the new design. A complete production run of the new design will not be part of this project. The thesis will not explain and expose all of the details in this project as the design is the property of a company.

## 2. Methodology

As part of the first objective a C program, targeting a Microchip controller was made for the current design. The new program improved stability and timing of functions such as writing to the LCD and communications with the watchdog timer circuit.

Next up was to solve some problems in the current hardware design including failure to boot, unwanted shutdown, AC/DC circuit making some bad noise and some minor problems. To completely understand the current design the schematics, circuit board layout and bill of material (BOM) was studied carefully. The problems was then solved by using a oscilloscope comparing the readings to the datasheet specifications of the components in the design.

The new hardware design including schematics, layout and BOM was created using free and open source software (FOSS) called KiCAD. By reading the official documentation of KiCAD and by studying other peoples work in KiCAD enough knowledge was gained to start creating the new design. SVENSK Elektronikproduktion WearOne AB do follow ISO9001:2008 quality policy and did provide company documents containing guidelines on how to make clean and safe circuit designs. Then the C program was ported so that is could be used in the new design.

The last step was to build a prototype and verify the function of the design. The BOM was converted using SVENSK Elektronikproduktion WearOne AB document standard to a format that contains part manufacturer, part suppliers, cost of material and cost of production. Necessary part and material was collected and purchased to make a prototype which was later assembled by hand. The prototype was then tested by probing with a oscilloscope which resulting in some minor modifications. The working circuit was finally put to long term test in a production machine.

## 3. Frame of Reference

### 3.1. Electrical component theory

The circuitry of mechatronic systems performs the task of reading physical measurement, process them and depending on the result activate different kind of tools. When transforming physical measurement into electrical signals, sensors of different kind are used. The outputs from these sensors may be modified by electrical components such as resistors, capacitors, transistors, opto-coupler or IC-filter. This modifications may be necessary to make the signal readable for the microcontroller or to activate output signals. The microcontroller analyzes the data and send the appropriate output signals that may be amplified and modified by other components to drive motors, lights, speakers, control valves or do other stuff.

#### 3.1.1. Resistors and the basics laws of electricity

The resistor is used to limit the flow of current, the higher the resistance is the less current will flow through it. The resistor is made by a material that's partly conductive and the resistance is adjusted by changing the shape of the conducting material. Ohms law

$$u=i * R$$

defines the relationship between the current and voltage in a resistor, where U is voltage across the resistor, I is the current flow through the resistor and R is the resistance. Because of Ohms law it's possible to use the resistor to control the voltage if the current is constant and vise versa.

A common usage of the resistor is the voltage divider, see illustration 1. To understand the voltage divider one must first study Kirchhoff's voltage law that says that the total voltage drop in a closed loop in a circuit is equal to zero , see illustration 1.

$$v_1+v_2+v_3=0$$

Rearrange the formula we get

$$v_3=v_1+v_2$$

where  $v_3$  is the total voltage over the resistors. By using ohms law we get

$$v_3=i * R_1+i * R_2$$

and rearranging gets

$$i=\frac{v_3}{R_1+R_2}$$

Ohms law says that

$$i=\frac{u_2}{R_2}$$

and by putting this expression in previous expression we get the formula of the voltage divider

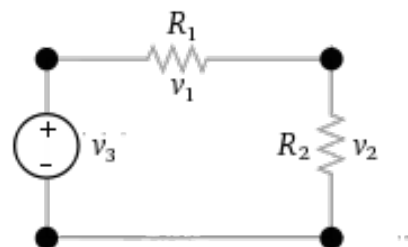


Illustration 1: Kirchhoff's voltage law  $v_1 + v_2 + v_3 = 0$

$$u_2 = \frac{R_2}{R_1 + R_2} * u_3 \quad .$$

There is also the Kirchhoff's current law that describes the relation of current flow through a junction  $i_1 + i_2 + i_3 + i_4 = 0$  , see illustration 2.

The Kirchhoff's laws is commonly used to calculate the total resistance in a resistor network. The total resistance of arbitrary resistors in series can be derived from the Kirchhoff's voltage law and get

$$R_{tot} = R_1 + R_2 + R_3 + \dots \quad .$$

There is also a expression derived from the Kirchhoff current law which describes the total resistance of resistors in parallel which is as follows

$$R_{tot} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots} \quad .$$

There two expressions can be used together to calculate the total resistance of any resistor network.

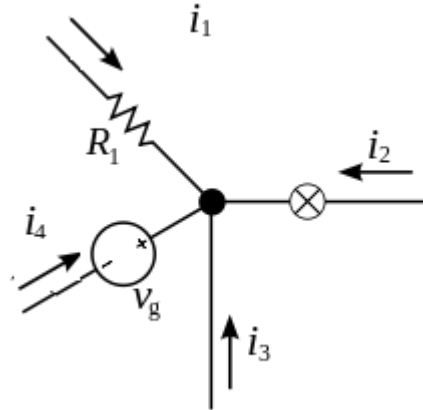


Illustration 2: Kirchhoff's current law  $i_1 + i_2 + i_3 + i_4 = 0$

### 3.1.2. Capacitor and the complex number representation

The capacitor is mainly used as a frequency dependent conductor in AC circuitry where high frequency and capacity equals better conductivity. The capacitor consists of two multilayer metal plates which are separated by a isolating material. It's ability to charge and discharge is useful in power circuitry. The electrical properties of the capacitor can be described by the formula

$$i(t) = C \frac{dV(t)}{dT} \quad ,$$

Where C is the value of the capacitor.

The analysis of resistance-capacity (RC) networks in AC circuitry is commonly done by a complex number representation of the impedance in the circuitry. The Impedance Z is the sum of the resistance R and reactance X in the circuit

$$Z = R + jX \quad ,$$

where the reactance is a frequent dependent resistance. This representation is useful because we can now use both Kirchhoff's laws and Ohms law by replacing R with Z.

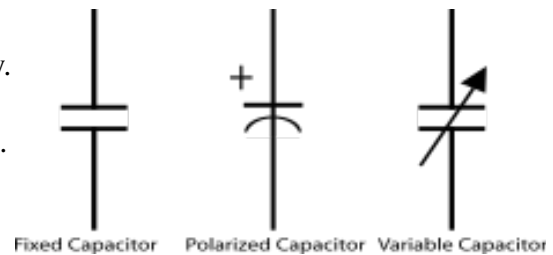


Illustration 3: Capacitor schematic symbols



Illustration 4: Left side displays different sizes of surface mount capacitors. Top right side is a tantalum through-hole capacitor, and at the bottom right side is a electrolytic through-hole capacitor

To put everything together we need to find an expression of the impedance of a capacitor. The voltage amplitude in a AC circuit can be described by a sinusoidal function

$$u(t) = V_{max} \sin(\omega t) ,$$

where  $\omega$  is the angular frequency. With a little help from ohms law and the physics law of the capacitor we get:

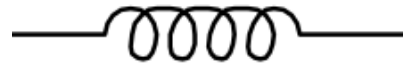
$$Z = \frac{U}{I} = \frac{u(t)}{i(t)} = \frac{V_{max} \sin(\omega t)}{C \frac{dV(t)}{dT}} = \frac{V_{max} \sin(\omega t)}{\omega C V_{max} \cos(\omega t)} = \frac{\sin(\omega t)}{\omega C \sin(\omega t + \pi/2)} = \frac{1}{\omega C} e^{-j\frac{\pi}{2}} = \frac{1}{j\omega C}$$

### 3.1.3. Inductor

The inductor is basically a wire wound in circles. The Inductor acts as a frequency dependent resistor in AC circuitry where high frequency and inductance equals lower conductivity. It's ability to charge and discharge is useful in power circuitry. The charge is stored as a magnetic field around the inductor and is usually improved by placing a ferrite core inside the winding. The electrical properties of the inductor can be described by the formula

$$v(t) = L \frac{di(t)}{dT} .$$

The current amplitude in a AC circuit can be described by a sinusoidal function



$$i(t) = I_{max} \sin(\omega t)$$

*Illustration 5: Inductor*

From ohms law and the physics of the capacitor we get:

$$Z = \frac{U}{I} = \frac{u(t)}{i(t)} = \frac{L \frac{dI(t)}{dT}}{I_{max} \sin(\omega t)} = \frac{\omega L I_{max} \cos(\omega t)}{I_{max} \sin(\omega t)} = \frac{\omega L \sin(\omega t + \pi/2)}{\sin(\omega t)} = \omega L e^{j\frac{\pi}{2}} = j\omega L$$

### 3.1.4. Diode

A diode is a component that conducts current in only one direction as long as the voltage is kept in its operating domain. Diodes are usually made from silicon where every atom uses 4 electrons to bond to 4 neighbour atoms. The silicon diode is made of two sections of semiconductor material where one area is positively doped and the other is negatively doped. The negatively doped area is made by adding material that has five free electrons which of four will form the bond with the neighbour atoms and the fifth will act as a negative carrier. The positively doped area is made by adding material with only three free electrons that forms the bond with the neighbour atoms, this creates a hole that acts as a positive carrier.

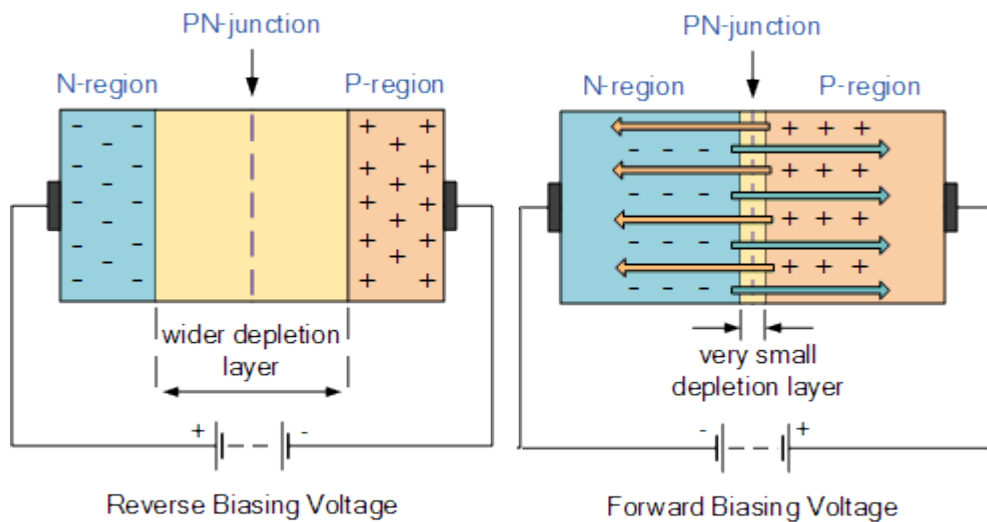


Illustration 6: The PN-junction of a diode

In the junction between the doped areas the free electrons from the negatively doped side will pair up with the free holes from the positively doped side. This creates a barrier in the junction which has no free carriers available. If the diode is connected so that the positive side of the voltage source is connected to the negatively doped area and the negative side of the voltage source is connected to the positively doped area, the connection is called backwards bias. When connected as backwards bias the holes on the positive side of the diode is drawn away from the junction toward the negative battery connection, the free electrons is also moving away from the junction in the same fashion. The junction therefore enlarged when the diode is backwards bias and will prohibit the flow of current. The opposite is happening if the polarity of the voltage source is switched, the diode is now operating as forward bias and the current may flow if the voltage difference is more than 0.7v.

### 3.1.5. Bipolar Junction Transistor (BJT)

The BJT is used to amplify current signals. BJT:s have moderate input impedance but can be produced with smaller amplification tolerance than the Field Effect Transistor (FET).

#### Using the BJT

The BJT amplifies the collector (C) current as a function of the base (B) current. The amplification is described by the formula:

$$I_C = \beta I_B$$

Where  $\beta$  is the amplification of the transistor. The emitter current is derived from Kirchhoff's current law:

$$I_E = I_B + I_C$$

There is two kinds of BTJ, the NPN and the PNP, where P means positive doped area and N negative doped area, see illustration 7. Both have 3 terminals, the collector (C) the base (B) and the emitter (E). In the NPN transistor the current running

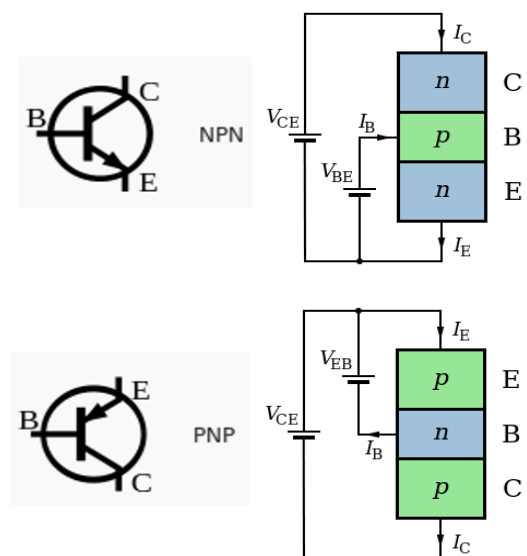


Illustration 7: Left side displays schematic symbols, right side displays the current flow



from the base to the emitter amplifies the current running from the collector to the emitter. In the PNP transistor the current running from the emitter to the base amplifies the current running from the emitter to the collector.

### Looking under the hood

To understand how a transistor works one must first understand the principle of the diode. The transistor consists of two junctions that behaves as the junction in a diode. When studying the NPN transistor where the the junction between the collector and the base is reversed bias and the other junction between base and collector is forward bias. The reversed bias limits the current flow between the collector-emitter when there is no base-emitter voltage. When voltage across the base-emitter reaches more than 0.7v the free electrons in the highly doped emitter starts to flow to the base. Only a few free electrons finds a hole in the base area and flow out of the base terminal, the rest of the free electrons is pushed though the base-collector junction because of the electrostatic force from the base-emitter voltage.

#### 3.1.6. Field Effect Transistor (FET)

The FET is used for amplifying voltage signals. FET:s generally have very high input impedance but suffers from big spread in amplifications comparing different production runs. They produce less noise compared to a BJT and do only draw current at the gate when the signal is changed. The Complementary Metal Oxide Semiconductor (CMOS) is today used in digital circuitry such as processors, memory and motherboards because of its advantage that is only consuming power at the gate upon changing state. FET:s is sensitive to voltage overload and needs to be handled in an antistatic environment.

### Using the FET

FET:s comes with three terminals which has similar functions as the terminal of the BJT. The drain (D) is similar to the collector, the gate (G) is similar to the base and the source (S) is similar to the emitter.

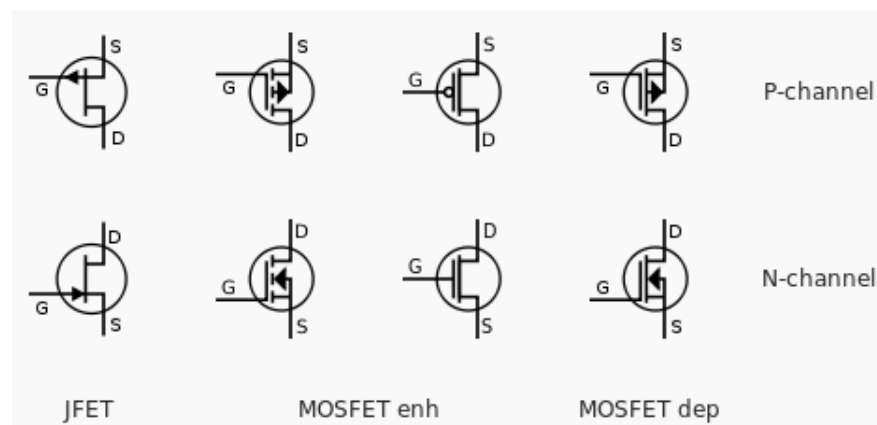


Illustration 8: JFET and MOSFET symbols

There are mainly two types of FET:s that are used in today's electronics, the Junction Field Effect transistor (JFET) and the Metal Oxide Semiconductor Field Effect Transistor (MOSFET). Both the JFET and the MOSFET can be made with a positive doped channel (p-channel) or a negative doped channel (n-channel) between drain and source. If the device is in depletion mode the channel is normally open but can be shut off by adjusting the gate-source voltage. If the device is in enhancement mode the channel is normally shut off but can be opened by adjusting the gate-source voltage. MOSFET:s can be made in enhanced mode or depletion mode but the JFET exists only in depletion mode, *see illustration 8*.

The relation between channel conductance and gate-source voltage on different FET:s is described below:

- JFET N-channel (depletion):  $G > S$  closes the channel, (open by default when  $G = S$ ).
- JFET P-channel (depletion):  $G < S$  closes the channel, (open by default when  $G = S$ ).
- MOSFET N-channel enhanced:  $G > S$  opens the channel, (closed by default when  $G = S$ ).
- MOSFET P-channel enhanced:  $G < S$  opens the channel, (closed by default when  $G = S$ ).
- MOSFET N-channel depletion:  $G < S$  closes the channel, (open by default when  $G = S$ ).
- MOSFET P-channel depletion:  $G > S$  closes the channel, (open by default when  $G = S$ ).

### 3.1.7. Triode for alternating current (TRIAC)

The TRIAC is made by a NPN and a PNP transistor connected together, see illustration 9. The collector from the PNP transistor is connected to the base of the NPN transistor. The base of the PNP transistor is connected to the collector of the NPN transistor. The AC current flows between the emitter of the PNP transistor to the emitter of the NPN transistor and is controlled by opening the gate of the NPN transistor.

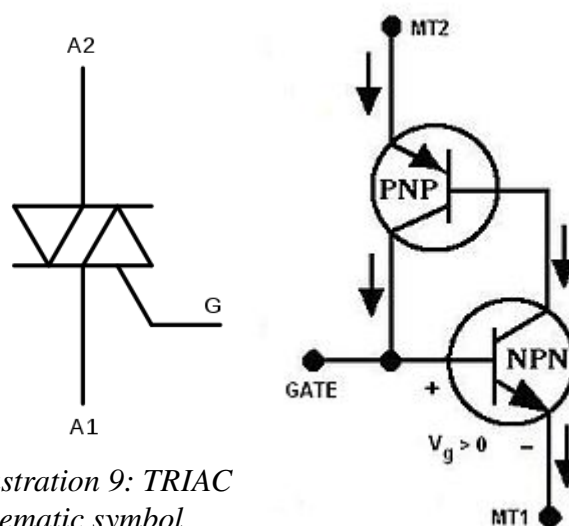


Illustration 9: TRIAC schematic symbol

### 3.1.8. Opto-isolator

The opto-isolator is built up by a LED and a phototransistor. This configuration is used for isolate the source signal from the output terminal. This is useful because it prohibits voltage spikes occurring on the input side to transfer to the output side.

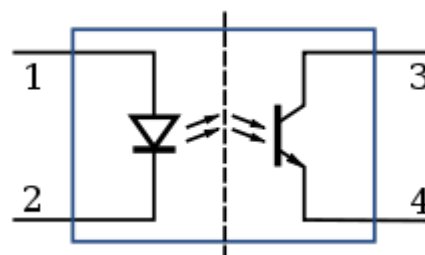


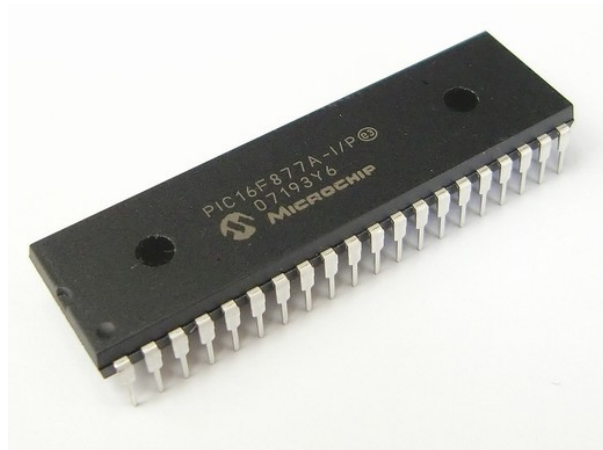
Illustration 10: Schematic diagram of an opto-isolator

### 3.1.9. Microcontroller Microchip PIC16F877A

A microcontroller is a small computer with processor, memory and input/output (I/O) peripherals on a single IC. The I/O peripherals makes it possible read/write analog signals, digital signals, serial communications and do pulse width modulation (PWM).



*Illustration 12: TQFP packaging*



*Illustration 11: DIP packaging*

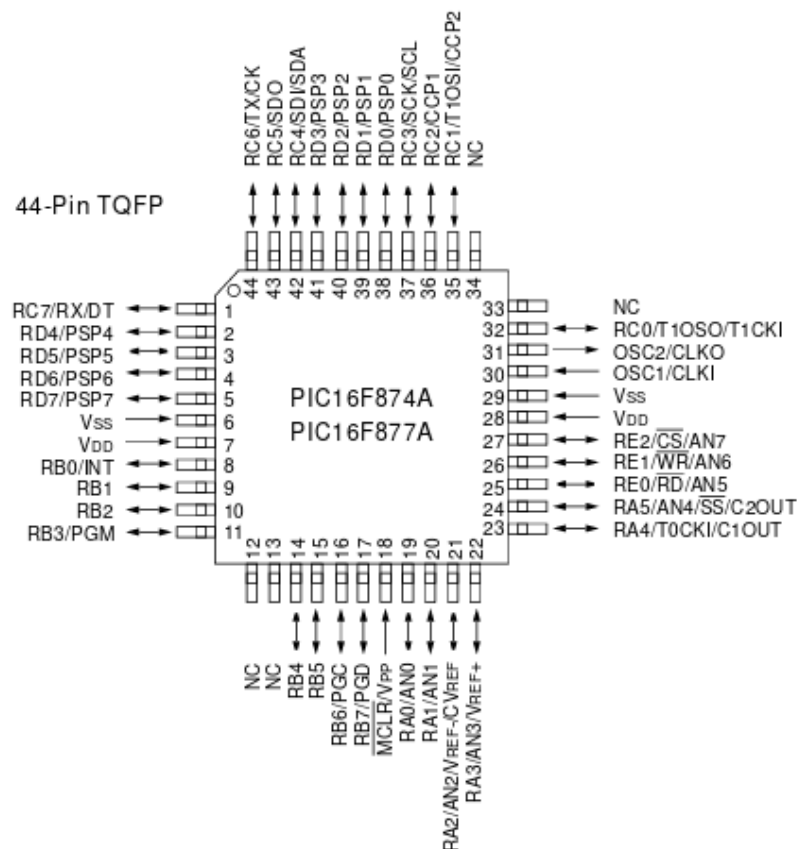
The PIC16F877A is a fairly old 8-bit microcontroller from Microchip, see illustration 11 and 12. In new design it's recommended to use PIC16F18877 instead which is newer and has a similar specification. This project will use the older PIC16F877A that is used in the previous design so that only one program will be necessary to make. The pinout of these microcontrollers is the same so the PCB layout is not necessary to change if swapping microcontroller. A brief hardware specification of the PIC16F877A can be viewed I table 1.

| Parameter Name                    | Value                               |
|-----------------------------------|-------------------------------------|
| Program Memory Type               | Flash                               |
| Program Memory (KB)               | 14                                  |
| CPU Speed (MIPS)                  | 5                                   |
| RAM Bytes                         | 368                                 |
| Data EEPROM (bytes)               | 256                                 |
| Digital Communication Peripherals | 1-UART, 1-SPI, 1-I2C1-MSSP(SPI/I2C) |
| Capture/Compare/PWM Peripherals   | 2 Input Capture, 2 CCP,             |
| Timers                            | 2 x 8-bit, 1 x 16-bit               |
| ADC                               | 8 ch, 10-bit                        |
| Comparators                       | 2                                   |
| Temperature Range (C)             | -40 to 125                          |

|                             |          |
|-----------------------------|----------|
| Operating Voltage Range (V) | 2 to 5.5 |
| Pin Count                   | 40       |

*Table 1: Microchip PICF877A specification*

All the specification and information can be found in the datasheet of the PIC16F877A (13). When programming a microcontroller the datasheet is a necessary reference on how to set up registers and peripherals. There are many tutorials of these microprocessors available online which contains runnable programs controlling different parts of the peripherals in the microcontroller [22].



*Illustration 13: Microchip PIC16F877A pinout*

## Serial programming

To transfer a program to the microprocessor it's necessary to use a programming device. Computers with parallel ports can be connected directly to the microprocessor for programming. Most computers don't have a parallel port but instead uses a serial programmer that can be hooked up to a USB port. Microchip sells a programmer called PICkit 3 which can handle the PIC family microcontrollers. PICkit can be used with the Microchip IPE software.

### 3.1.10. Displayinterface HD44780

The HD44780 is a display originally made by Hitachi with support of writing ASCII characters. The hardware interface on the Hitachi HD44780 has since grown in popularity and is now used by several LCD manufacturers in their displays. These displays comes in sizes of

8x1, 16x2, 20x2, 20x4 and 40x4 characters which do all use the same interface of 16 pins. The interface can be setup to use 4-bit or 8-bit data bus. The complete instruction of the display set can be found in the Hitachi HD44780 datasheet and a display driver written in C for the PIC16F877A can be found in attachment B.

Pinout of the HD44780:

1. Ground
2. VCC (+3.3 to +5V)
3. Contrast adjustment (VO)
4. Register Select (RS). RS=0: Command, RS=1: Data
5. Read/Write (R/W). R/W=0: Write, R/W=1: Read (This pin is optional due to the fact that most of the time you will only want to write to it and not read. Therefore, in general use, this pin will be permanently connected directly to ground.)
6. Clock (Enable). Falling edge triggered
7. Bit 0 (Not used in 4-bit operation)
8. Bit 1 (Not used in 4-bit operation)
9. Bit 2 (Not used in 4-bit operation)
10. Bit 3 (Not used in 4-bit operation)
11. Bit 4
12. Bit 5
13. Bit 6
14. Bit 7
15. Backlight Anode (+) (If applicable)
16. Backlight Cathode (-) (If applicable)

### **3.1.11. Pneumatic system and control**

Air is normally compressed by a piston based compressor. The pressure can mechanically be regulated by a valve containing a spring where the preload on the spring controls the air pressure. The pressure can mechanically be displayed by a manometer which uses a spring connected to an indicator to show the value.

A common method to transform pressure into an electrical signal is by the use of quarts that build up electric charge when pressurised, also called the piezoelectric effect. The sensor signal can be used for creating a closed looped system that controls pressure by actuating a servo motor valve. An on/off valve can be created by the use of electromechanical solenoid which basically is an electrically controlled magnet that shuts or opens the valve.

### **3.1.12. Part suppliers and manufacturers**

For low volume production or prototyping there will be a substantial wage cost per unit of just finding the right parts. In this case it's essential to find a supplier that sells a good range of products and brands. In other cases the lead time may be the crucial factor when choosing the supplier. In both these cases the big distributors in electronic components will be the preferred

source. A way to search for parts and to compare prices of these big distributors is the use of a dedicated search engine for electrical components such as Octoparts (18) .

For high volume production the price per unit is more crucial than the initial cost of finding the components. In this case you may want to find a manufacturer that sells directly or find a first hand distributor. It's usually necessary to negotiate with several suppliers to get good quality at a decent price. The busiest hub when trading electronic components and circuitry is the Chinese province Shenzhen. It's common to build your purchase network around this area, maybe even hire people on site to visit fabricators if you demand specially made components at high quality. Some may look for new suppliers on e-markets that specializes in business to business trading, one of the most popular one is Alibaba (19). When sourcing from e-markets sites the buyers may want to check the reputation of the seller by reading reviews of the seller and their products. It's also common to use a protective layer when transferring money such as PayPal which also makes it easier to get the money back when returning a product.

### **PCB-supplier**

When sourcing PCB:s the most common way is to have a direct contact with the fabricator. There's a few fabricators in Sweden, they're provide short lead time and customer service. If you on the other hand require affordable PCB:s there are many cheap sources in Shenzhen, China.

## **3.2. Programming the PIC16F877A in C**

Before getting started it's necessary to set up a development environment on your desktop computer and get a serial programmer as mentioned in §3.1.9. Many programmers prefers to use a GNU+Linux based operating system to run their development environment. it comes in many flavors and can be customizable to infinity. Some people call it the Linux operating system, Linux is merely a kernel which is a small part of the operating system. To give the developers of the GNU operating system credits for their work it should be called the GNU+Linux system.

Microchip supplies their own Integrated Development Environment (IDE) which together with their IPE do contain all necessary tools to develop software for the PIC family processors. IDE:s are preferred by beginners because of the short setup time required. Microchip IDE and IPE called Mplab X is freeware but proprietary closed source software.

Pinguino is a open source IDE targeting the PIC18F and PIC32MX microcontrollers. Its made to suit beginners by being easy to learn and easy use (20). There is also a development board with a PIC18F47J53 by the name Pinguino.

The experienced programmer may have special requirements on their development environment which a closed source IDE may not fulfill. The option is then to pick the tools necessary by own preference to make a complete development environment. A complete development environment should at least have a editor to edit the source code and a compiler toolchain that can convert the source code into a runnable program.

### **3.2.1. GNU Emacs Editor**

The GNU Emacs editor is a extremely powerful editor, it's in fact so powerful that is could be run as a operating system. The GNU Emacs editor was created by Richard Stallman in 1985, founder of the Free Software Foundation, GNU operating system and the GPL software

license. Emacs support syntax highlighting for a broad range of program language by default. There are also vast options of plugin available to the Emacs editor.



*Illustration 14: Richard Stallman, the father of free (libre) software*

Emacs is design to run in a terminal but can also be launched as a graphical program. There is a tutorial suitable for first time user which can be evoked by pressing Ctrl-h and then t while in the editor. To get help press Ctrl-h then h or press Ctrl-h then m to access the complete manual.

### **3.2.2. Compiler tools**

A compiler toolchain is usually made up of a preprocessor, compiler, assembler, standard library, linker and a debugger. The preprocessor parse and subsitute source code where specified, preprocessor directives is denoted by # in the source code. The compiler translate from one language to another, usually from source code (C in this case) to assembly code which is processor specific instructions. The assembler then translate it into object code. The linker combines the object files and libraries into a runnable program. The debugger is used for testing the program. When dealing with a microcontroller you'll also need a tool to upload the program to the microcontroller.

The compiler toolchain that is supplied with Mplab X exists both as a free basic version and an improved pro version that requires a license bought from Microchip. If requiring a open source compiler the the Small Dev C Compiler (SDCC) is available for the PIC 16 and 18 family (22). The SDCC compiler comes under different licenses, where the GPL+LE supplies

the microchip headers files and linker script which is necessary when compiling for any PIC device. The PIC support on SDCC is still a Work in progress and may contain bugs and miss some features. Usbpicprog is an open source hardware/software programmer for the PIC family processor that comes with a bootloader and uploader software.

### 3.2.3. Microcontroller setup

The configuration bits determine what mode the micro controller enters at startup. If these are set incorrectly the device can be damaged. For example if code protection is set you'll never be able to read the device memory again. Below is an example of how the configuration bits can be set up with a short comment on what it does.

```
#pragma config FOSC = XT      /* External clock. */
#pragma config WDTE = OFF     /* Watchdog timer off. */
#pragma config PWRTE = OFF    /* Disable power up timer. */
#pragma config CP = OFF       /* program code protection off. */
#pragma config CPD = OFF      /* EEPROM Memory Code Protection off. */
#pragma config BOREN = ON     /* Brown-out reset enable. */
#pragma config LVP = OFF      /* Low-voltage programming disable. */
#pragma config DEBUG = OFF    /* Debugging off. */
#pragma config WRT = OFF      /* Watchdog reset timer off. */
```

The configuration of the peripherals in the microcontroller is done by writing data into specific memory addresses.



### 3.3. Introduction to KiCAD

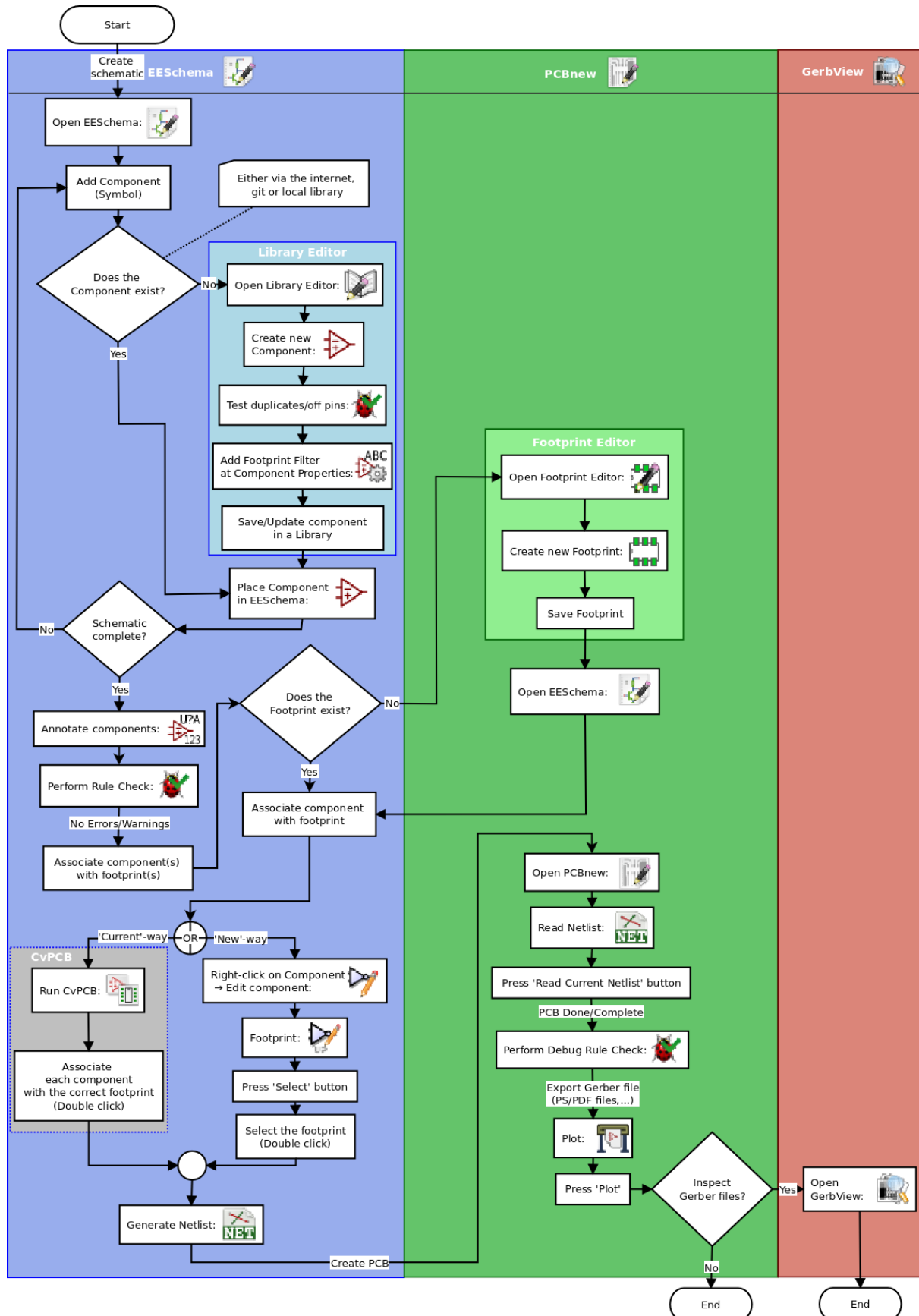
KiCAD is a open source Electronic Development Environment (EDA) that is suitable for schematic design and PCB layout. It assembles a packages of stand-alone software tools that handles different part of the PCB development, see *table 2*.

| Program name     | Description  |
|------------------|--|
| KiCad            | Project manager  |
| Eeschema         | Schematic editor (both schematic and component)                                      |
| CvPcb            | Footprint selector   |
| Pcbnew           | Circuit board board editor   |
| GerbView         | Gerber viewer  |
| Bitmap2Component | Convert bitmap images to components or footprints                                    |
| PCB Calculator   | Calculator for components, track width, electrical spacing, color codes, and more... |
| Pl Editor        | Page layout editor   |

*Table 2: KiCAD program suit*

Section 3.4 – 3.8 in this report is a summary of the getting started section in the KiCAD documentation. The complete documentation is available at the KiCAD webpage (17).

The workflow of creating a PCB design and layout in KiCAD can be viewed in *illustration 15*.





*Illustration 15: KiCAD work flowchart*

### 3.4. Schematics design in KiCAD

1. Start the KiCAD project manager. Choose **File** → **New Project** → **New Project**.

#### 3.4.1. Adding a component

2. Create a schematic in the schematic editor *Eeschema*, . Save the whole schematic project: **File** → **Save Schematic Project**
3. Click on the *Place component* icon  in the right toolbar or use the shortcut (*a*). Then click in the middle of the sheet. The *Components* window will pop up, filtering for R and pick the resistor by double click it. Place the resistor anywhere you want it on the sheet by click the mouse.

Pro tip: To show a list of shortcuts you can press the (?) key.


Pro tip: Scroll the mouse wheel up/down to zoom in/out on the sheet. Hold down the mouse wheel to move around.

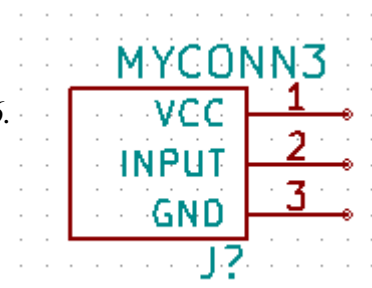
4. Change the value of the components by right click it and select **Edit Component** → **Value** or press the (*v*) key. Set the value to *1 k*. Please check out other component specific editing by right click it, notice that the keyboard shortcut is listed to the right of every command.

Pro tip: While hovering the mouse over the components, Duplicate a component by pressing the (*c*) key. Drag a component by pressing the (*g*) key. Move a component or text by pressing the (*m*) key. Rotate the component/text by pressing the (*r*) key or mirror it by pressing the (*x*) or (*y*) key.

5. Change the grid size to 50.0 mils by **Right-Click** → **Grid select**.
6. To add a new component that isn't configured in the default project, go to the main menu and choose **Preferences** → **Component Libraries** and click the **Add** button for **Component library files**. Add the *microchip\_pic12mcu* library which should be under the directory */usr/share/kicad/library/*. Repeat the add-components step but choose the *PIC12C508A-I/SN* instead.

#### 3.4.2. Creating a component

7. Read the section titled [Make Schematic Components in KiCad](#) to learn how to create a component from scratch and make the MYCONN3 component as in *illustration 16*.
8. Add a VCC power pin and a GND pin by clicking the *Place a power port* button  on the right toolbar
9. Now add and organize your schematics so it looks like *illustration 17*



*Illustration 16: Creating a component in KiCAD*

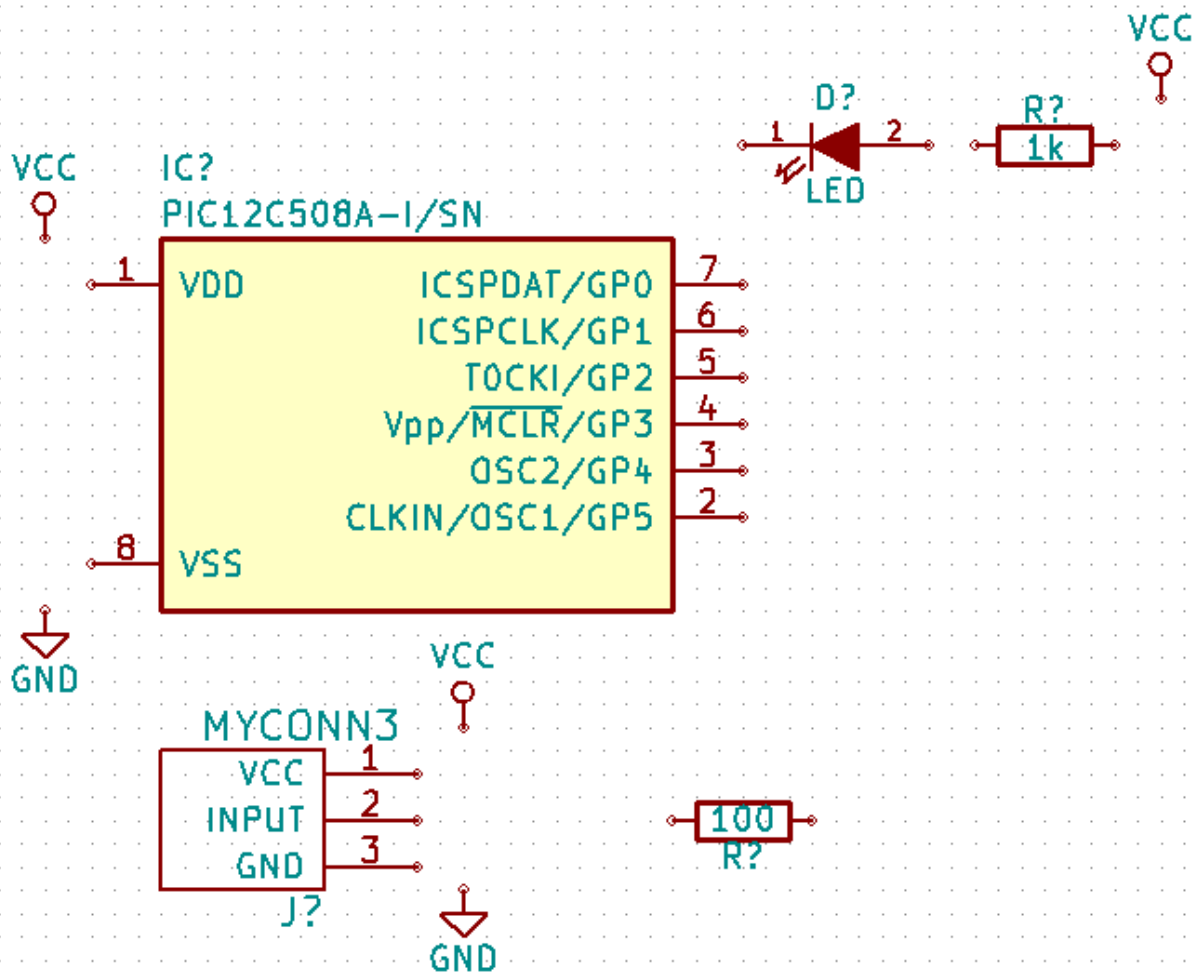



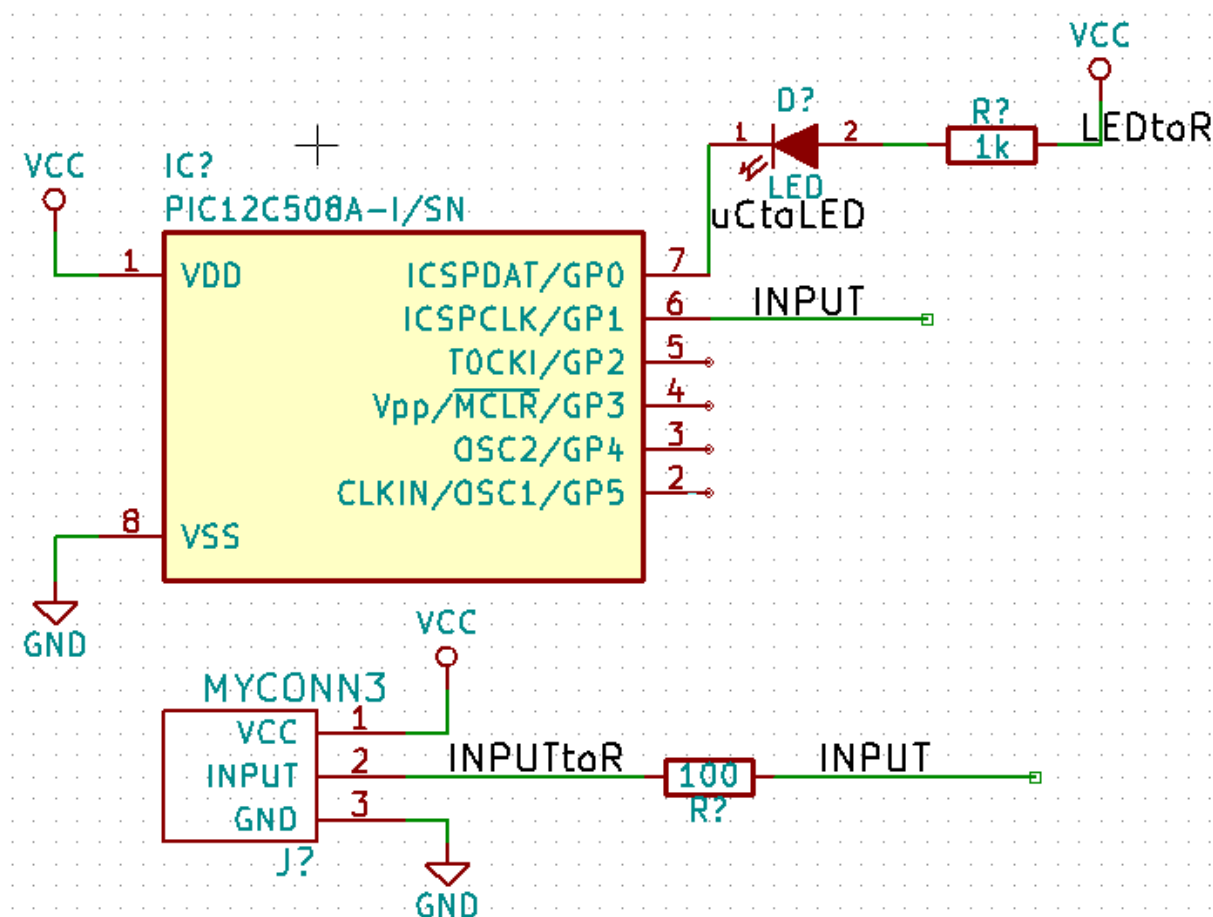


Illustration 17: Placing components

### 3.4.3. Connect the components


10. Click on the *Place wire icon*  on the right toolbar to attach the components together. Double-click to terminate a wire. Another way to deal with wiring is by the use of labels which is done by clicking the *Place net name icon*  on the right toolbar. Two labels acts as a invisible wire which will be useful when drawing a bigger design.
11. To avoid error on pins which should remain unconnected a *not connect flag*  must be placed upon these connections.
12. Power flags are necessary to attach so that KiCAD can validate the design. These are placed as a normal component.

13. Now connect your schematics so it looks like *illustration 18*.




*Illustration 18: Making connections*








#### 3.4.4. Annotate the components

14. Automatically annotate the components by clicking the *Annotate schematic* icon  on the top toolbar. Select *Use the entire schematic* and click on the *Annotation* and press OK.


#### 3.4.5. Design rules and validation

15. Scan the design for errors by *Perform electrical rules check* icon  on the top toolbar. This should return 0 errors and 0 warnings. If there is any error/warnings a green marker will appear on the sheet to highlight the error.
16. The schematics of this simple circuit is now completed and verified.



### 3.5. Generate netlist in KiCAD


17. A netlist contains information on how the components is connected together. To generate a netlist click on the *Generate netlist* icon  on the top toolbar and save it under the default file name.
18. To to associate each component in the netlist with a footprint, that is the physical shape of the component, click on the *Run Cypcb* icon  on the top toolbar.
19. Select *D1* and you'll see a list of footprints to the right. Scroll down and select *LEDs:LED-5MM* by double-click it.
20. The icons ,  and  can enable/disable filter by keywords, pin count, and type of component
21. For *IC1* select the *Housings\_DIP:DIP-8\_W7.62mm* footprint. For *J1* select the *Connect:Banana\_Jack\_3Pin* footprint. For *R1* and *R2* select the *Discret:R1* footprint.
22. Click the *View selected footprint* icon  to preview a component. use the icon  to save the netlist.

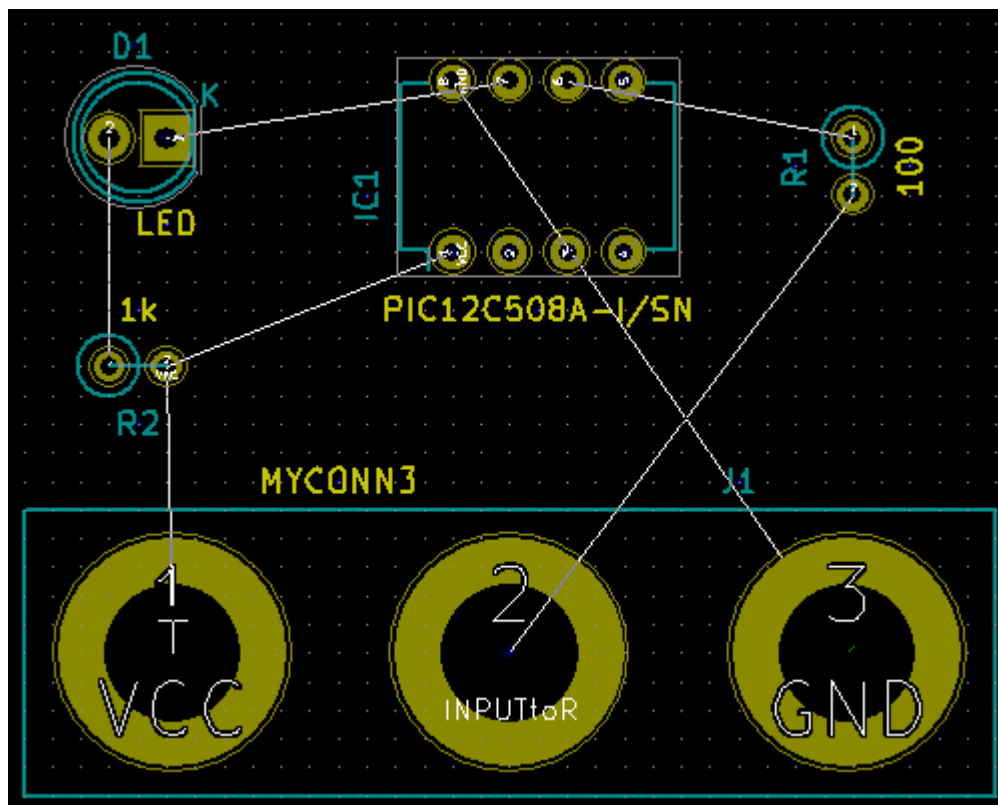
### 3.6. Generate BOM in KiCAD

23. A Bill Of Materials (BOM) may be generated to assist the purchase department and the assembly line. To generate BOM go to the *Eeschema* schematic editor and press the *Bill of materials* icon  on the top toolbar. Click on **Add Plugin** button. Select *bom2csv.xsl* in the directory */usr/lib/kicad/plugins/*. You may want to replace "%O" with "%O.csv".
24. Press *Generate*, you can now open and edit the csv file that is located in your project directory with LibreOffice Calc.



### 3.7. Circuit Board Layout in KiCAD

1. In KiCad project manager open the PCB layout editor by clicking the *Pcbnew* icon .
2. Click **Design Rules** → **Design Rules** and specifies minimum track width and clearance to those of your PCB manufacturer. A general recommendation would be to set track width and clearance to at least 0.25mm in both in *Net Classes Editor* and *Global Design Rules*.
3. Click on the *Read Netlist* icon  on the top toolbar to import the netlist, choose *Read Current Netlist*. You shall now see the components in the upper left side of the screen, if not please move around until you see them.

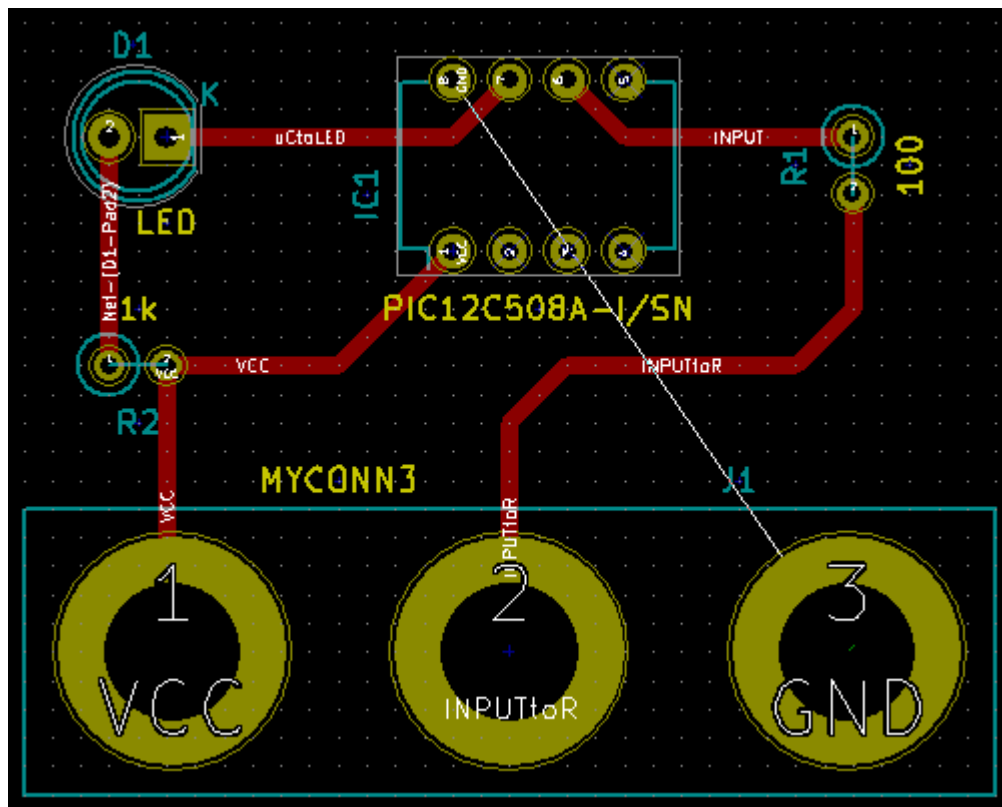
4. To find out how the components is connected to each other please make sure the *Hide board ratsnest* button  is pressed.
5. Move the components around by pressing (g) while hovering the mouse over it and the click to place it. Move the component in such a way that there is minimal crossover when wiring, see *illustration 19*.



*Illustration 19: Placing the components*


6. To define the board size Select *Edge.Cuts* from the drop-down menu in the top toolbar. Then click *Add graphic line or polygon* icon  on the right toolbar. Draw a square that is big enough to house all the components.
7. Select *F.Cu (PgUp)* in the drop-down menu on the top toolbar which is the top copper layer of the circuit board.
8. Be sure to select the grid size before wiring the components, **Right click** → **Grid Select**.
9. Connect all the components except for the ground connections by using the *Add Tracks and vias*  on the right toolbar. If you want another trace with please add/select it in the *Design Rules* → *Design Rules* → *Net Classes Editor*.


10. Wire all the components until it looks like *illustration 20*.



*Illustration 20: Wiring the board*

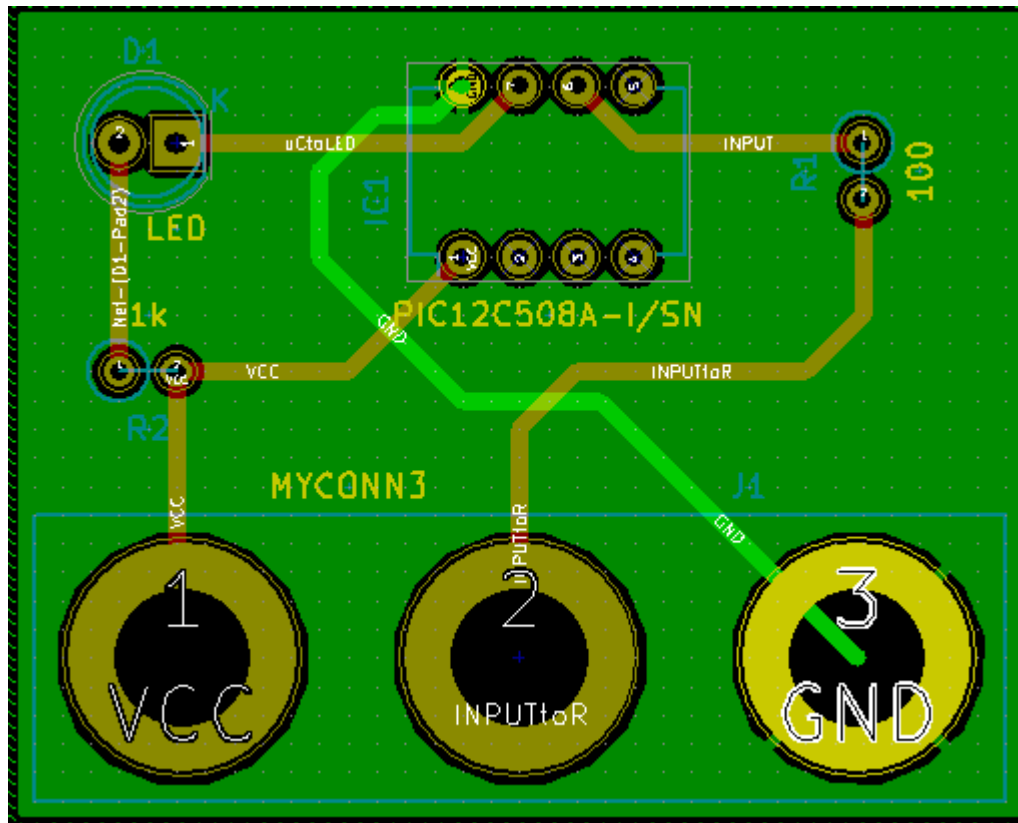
11. You can draw a connection on the bottom side by changing the layer to B.Cu in the drop-down menu on the top toolbar.
12. A connection between layers can be made by placing a via. While drawing a wire press the right mouse button and click *place via* or use the shortcut (v).

Pro tip: To inspect the connection to a specific pad press the *Net highlight* icon  on the right toolbar and the click on the pad.

13. It's common to use a ground plane to connect all the GND pins to. Click on the *Add Zones* icon  on the right toolbar. Select the area of the ground plane by clicking where you want your first corner. A the dialog that appear, set *Pad in Zone* to *Thermal relief* and *Zone edges orient* to *H,V* and click OK. Select the entire area of the board to be ground plane. Right click inside the area and select *Fill or Refill All Zones*.




14. Make sure the the circuit looks like in the illustration 21.



*Illustration 21: Wiring complete with ground plane*

### 3.7.1. Design rules and validation

15. Check the design for errors by clicking *Perform Design Rules Check* icon  on the top toolbar. Run the *DRC*, There should be no errors or unconnected track.

16. Your layout is now complete, save the layout **File** → **Save**.

### 3.7.2. 3D viewer

17. You can now see what your circuit looks like in 3D, click **View → 3D Viewer**.

### 3.7.3. Making footprints

Please read the KiCAD documentation on how to make footprints [22].

### 3.8. Generating production files

To have your board produced at a PCB fabricator you'll need to send them Gerber files of your design.

1. Click on **File** → **Plot**. Select *Gerber* as the format. The layer for at typical 2-layer board can be seen in table 3. Select these layer and click *plot*.

|                      | KiCad Layer Name | Old KiCad Layer Name | Default Gerber Extension | "Use Protel filename extensions" is enabled |
|----------------------|------------------|----------------------|--------------------------|---|
| Bottom Layer         | B.Cu             | Copper               | .GBR                     | .GBL  |
| Top Layer            | F.Cu             | Component            | .GBR                     | .GTL  |
| Top Overlay          | F.SilkS          | SilkS_Cmp            | .GBR                     | .GTO  |
| Bottom Solder Resist | B.Mask           | Mask_Cop             | .GBR                     | .GBS  |
| Top Solder Resist    | F.Mask           | Mask_Cmp             | .GBR                     | .GTS  |
| Edges                | Edge.Cuts        | Edges_Pcb            | .GBR                     | .GM1  |

*Table 3: Gerber files of a 2-layer board*

## **4. Implementation**

### **4.1. Creating new software written in C that will run on already produces machines**

The target microcontroller in this case was a Microchip PIC16F877A hence the datasheet of this controller was the main source of information for this task [13]. Personal project that uses a similar microcontroller was also used as a source of information. The current program which was written in Pikbasic suffered from problems such as writing to the lcd-display and keeping the watchdog circuit happy. To get better control over the hardware it was necessary to remove the extra layer that Pikbasic do create. A completely new program was necessary to create in the C language to regain control.

### **4.2. Find out why 15% of current production run do not pass test and verification and correct the issue.**

The problem that the microcontroller wouldn't boot from time to time on a number of circuits is not the usual way digital technologies behave. Digital technologies will produce the same result every time if the procedure stays the same. The assumptions where therefore that the supply voltage may be unstable during startup or that the master clear signal did not pull the reset long enough to for the power to stabilize. Probing with an oscilloscope did not show any problem with the power supply but instead the master clear signal did not behaved as expected. The problem was a capacitor that wasn't specified at the right value and tolerance. Capacitor are usually made with quite a big tolerance regarding its capacitance and this is the explanation of why only a part of the boards did have this problem. The value and tolerance of this capacitor was adjusted to correct the issue.

### **4.3. Create the schematics for the new circuit design.**

Octopart was used to look for components that was needed for the circuit. The datasheet of these components was then used to get information on how to properly setup and connect them. The KiCAD manual was used to get information about how to use the program [17].

The schematic design was created using the Eeschema which is part of the KiCAD EDA. Components was used from the KiCAD standard library and user library when available or otherwise created with the KiCAD Schematic Library Editor. The design was made around the microcontroller. A prefabricated module was used to transform 230v AC to 24V DC which is used to feed the magnetic valves and other outputs. The 24V DC was also transformed into 5V DC by the use of a buck switching regulator. Most of the components on the PCB uses 5V DC to power them.

An external oscillating crystal was connected to the microcontroller to ensure precise timing. The external master clear signal ensures that the microcontroller wont start until the feeding voltage is stable and the clock is running. A connector was added to the microcontroller to enable serial programming and debugging. The external watchdog was created using a counter which regularly must be reseted by the microcontroller to not overflow and generate a error signal.

The circuit has several outputs which controls magnetic valves, heater iron, speakers, display and LED indicators. Because the outputs of the microcontroller is only good for 10mA at 5v the signals was needed to be amplified in all cases but the display. The signals were amplified with the help of transistors that could provide enough current to drive the outputs, where some was driven by 24v to provide a higher output voltage. The heater element required 230v AC and this output was isolated by an optoisolator and driven by a TRIAC directly connected to the 230v AC main. The output signal was connected to coupling capacitors to reduce high frequency noise.

Most of the input signals did only required a pull down resistor and a coupling capacitor. But the response from the pt1000 temperature sensor is so weak it needs very accurate feeding and reading circuitry. The precision instrumentation amplifier INA125 from Texas instrument was chosen to do this job. Two operation amplifier is used to create a comparator which detects if the temperature sensor is disconnected or faulty.

The Electrical Rule Checker (ERC) was set to default values and then executed and errors such as unconnected wires and connection conflicts was sorted out. Parts of the circuit was connected and tested on a prototype board to verify its function.

#### **4.4. Design the PCB layout of the new circuit design.**

The netlist was generated from the schematic drawing. The components in the netlist was then associated with footprints from the KiCAD standard library, user library or created with the KiCAD footprint editor.

The PCB layout was created with the KiCAD PCB editor. The KiCAD manual was used to get information about how to use the program [17]. The board is a 2-layer design with most of its components on the front side (components side) and the ground plane on the back side (solder side). The 230v AC is isolated from the low voltage DC and do not have any ground plane. Standard track width is set to 0.6mm with a clearance of 0.2mm to minimize its resistance while not pick up too much noise. The standard via diameter is set to 0.6mm and the drill via is set to 0.4mm which is usually supported by the fabricators. Bigger traces are used where needed, in the 24v network and in the 230v AC network. At the front plane most of the traces is drawn in the horizontal direction and at the back plane most of the traces is drawn in the vertical direction. Bigger components and especially connectors uses through-hole mounting to get a higher structural strength in their attachment. Smaller components uses surface mounting as they are more cheap to manufacture and assembly.

#### **4.5. Port the C program so it will run on the new circuitry.**

The changes in the program was minimal as the same microcontroller was used in the new design. The display driver was altered in such a way that the initialization is only done once as the new display would stop working if not properly powered off between initialization. The version number printed on the screen was also changed to a major new version to show that the circuit is a newer generation.

## 4.6. Make a prototype of the new design.

The prototype circuit was assembled by hand as it was only made as a single copy. The inventory of SVENSK Elektronikproduktion WearOne AB was searched to find components that's used in the design. The PCB was produced by a Swedish company named Cogra. The remaining electrical components were sourced from Digikey and then assembled and soldered at SVENSK Elektronikproduktion WearOne AB.

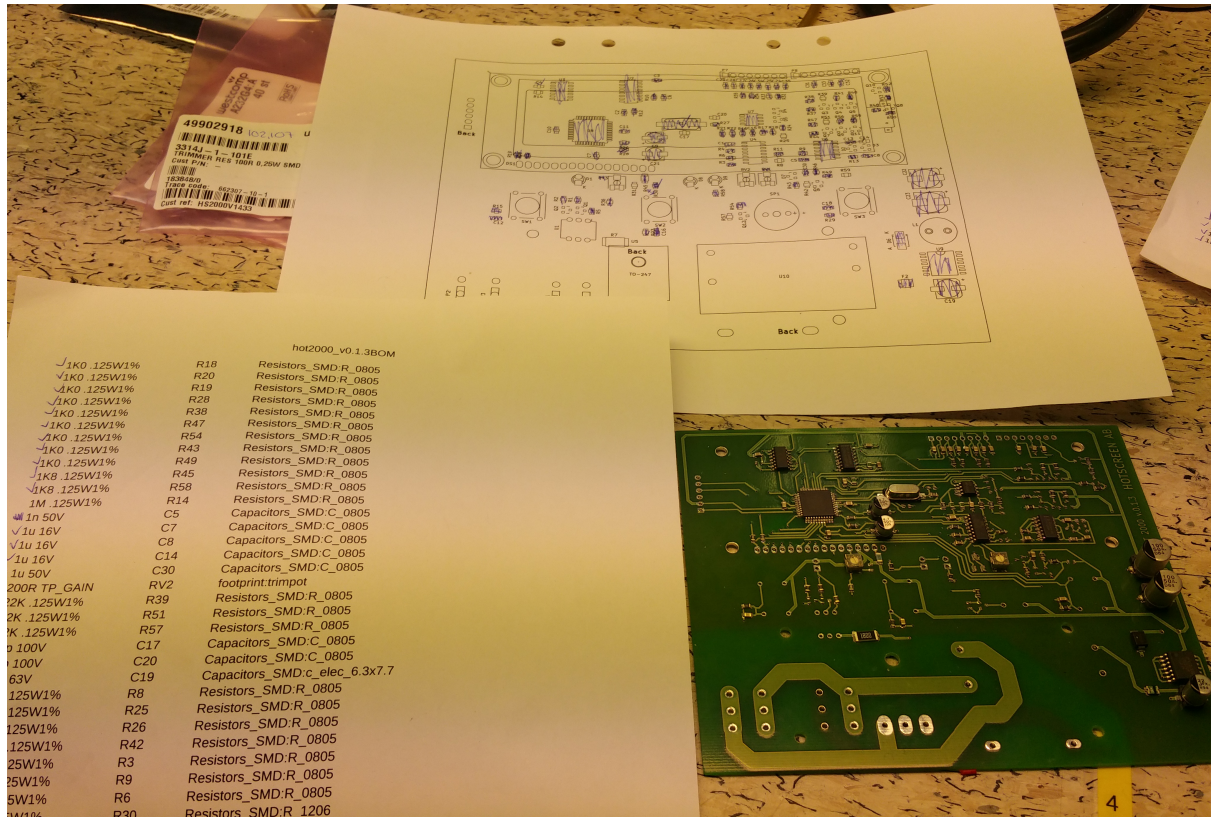
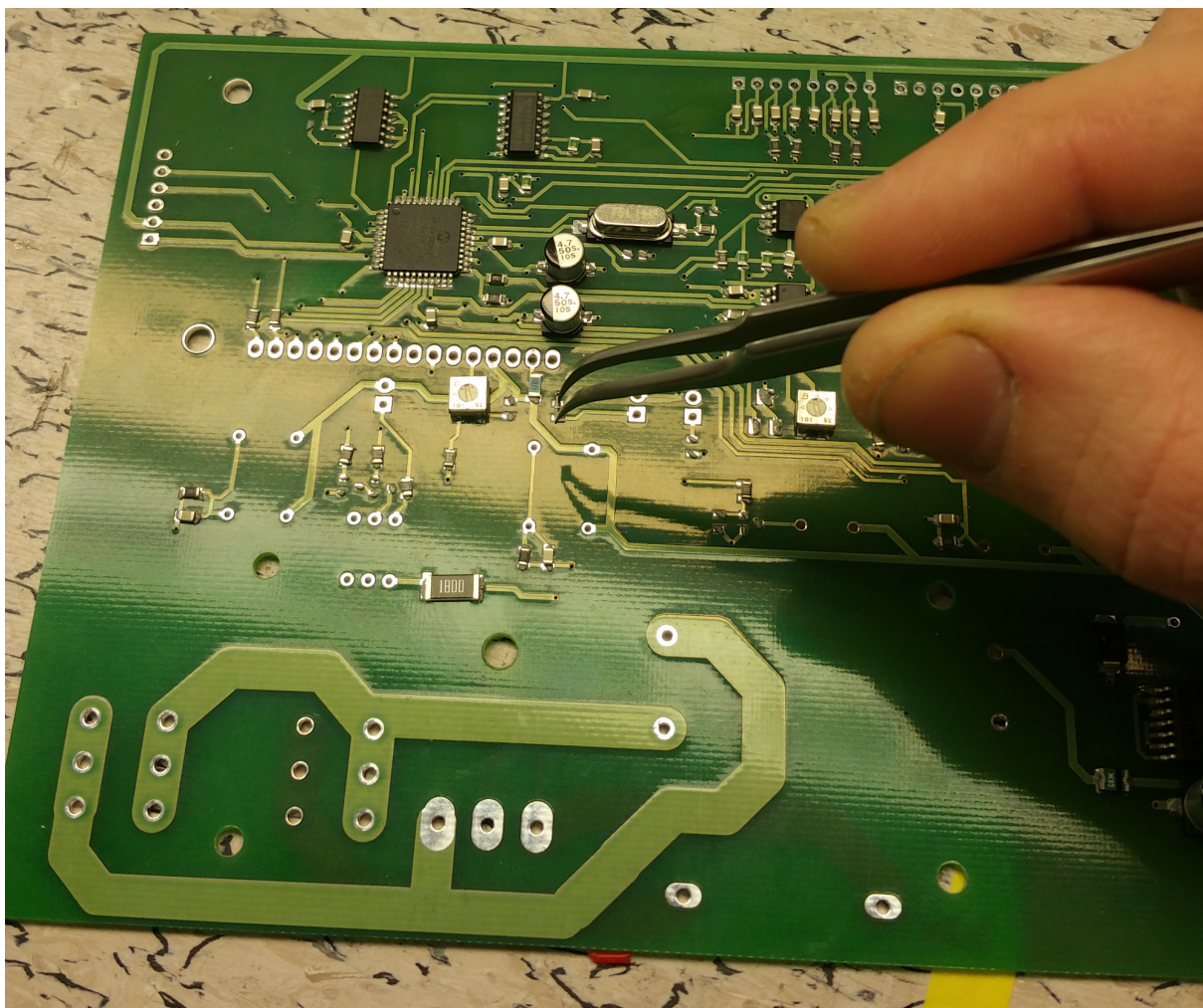


Illustration 22: The BOM (left) and placement drawing (top) is of great use when manually assembling the board

The assembly starts with applying solder paste on the pads of the surface mounted components. This is done using a manual dispenser as a dispensing stencil was not ordered for the prototype board. The surface mounted components was placed by hand using a tweezer as the setup and programming of the pick and place machine would have taken more time. The PCB was then put in a reflow oven to solder the surface mounted components. Because the number of through hole components is quite low these were soldered by hand.

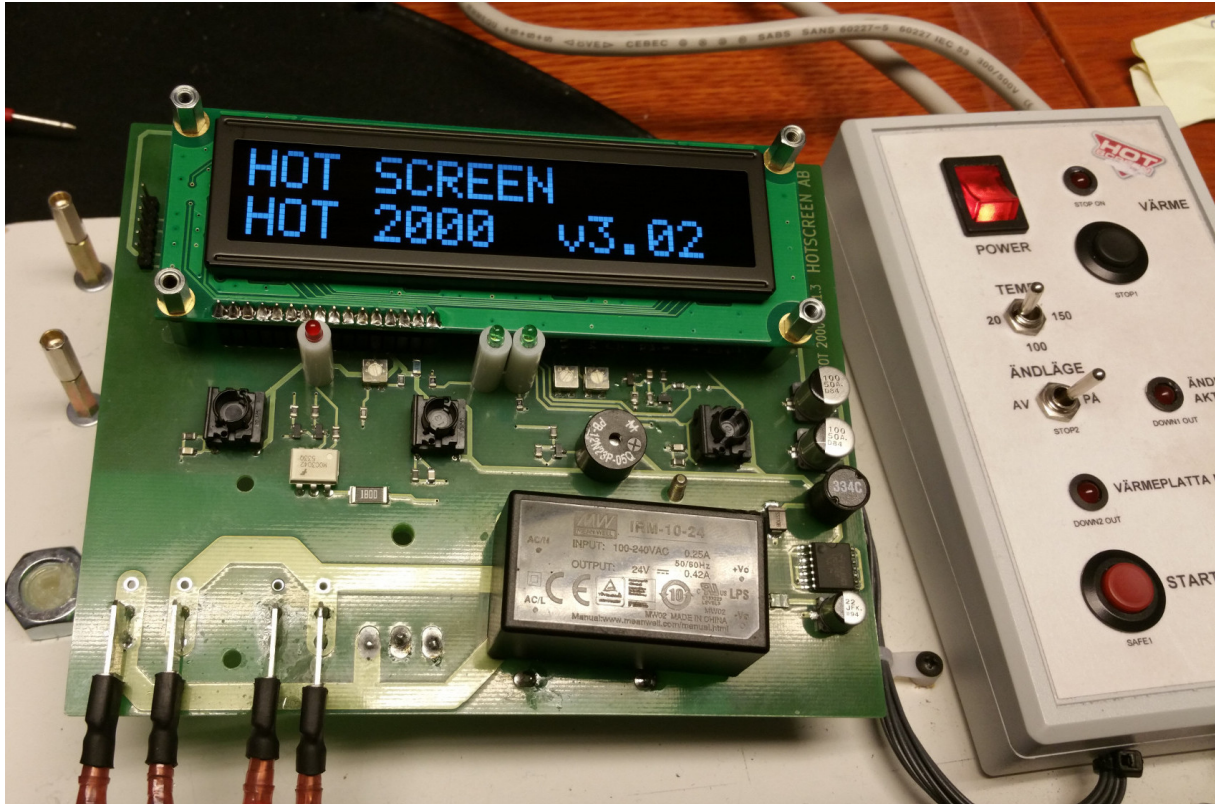




*Illustration 23: Manually placing the SMD components*

## 4.7. Test and verify the function of the prototype.

The first test of the prototype was done with the use of the testing equipment for the previous design. At first start the circuit was not working properly but with some probing using the oscilloscope the problems were found. A problem were that a few components was placed in the wrong location, this was caused by the silk screen text did not line up with the component which got the component mounted in the wrong location. A minor fault in the software caused the screen to fail which is mentioned in section 4.5.



*Illustration 24: Testing the circuit in the test equipment*

The prototype did now pass the first test in the testing equipment. The next step was to test the circuit in an actual machine. A machine that was left from a previous production run at SVENSK Elektronikproduktion WearOne AB was borrowed to do some real testing. The circuit did well except from the fact that it shut down from time to time. The cause of this error was the use of a too small polyfuse. When replaced by a bigger one the machine did run flawlessly. A long term test has also been done with satisfying results.

## 5. Result

This chapter will have a closer look of the result. Ideally the complete result would be shown here but as this design is the property of a company, not everything may be shown.

### 5.1. The program

The whole program wont be available but certain fragmets of it is presented below. The code is written to comply with the GNU coding standards (24).

#### 5.1.1. Setup

The setup code handle the configure of the hardware in the microcontroller. Some code is followed by a comment and where it's missing you'll may want to check the datacheet for more information (13). The setup code:

```
#pragma config FOSC = XT      /* External clock. */
#pragma config WDTE = OFF     /* Watchdog timer off. */
#pragma config PWRTE = OFF    /* Disable power up timer. */
#pragma config CP = OFF       /* program code protection off. */
#pragma config CPD = OFF      /* EEPROM Memory Code Protection off. */
#pragma config BOREN = ON     /* Brown-out reset enable. */
#pragma config LVP = OFF      /* Low-voltage programming disable. */
#pragma config DEBUG = OFF    /* Debugging off. */
#pragma config WRT = OFF      /* Watchdog reset timer off. */

/* Definitions. The comment is the name from schematics. */
#define _XTAL_FREQ 4000000
#define START1 RC2 /* Safe1. */
#define START2 RC3 /* Safe2. */
#define BUTTON_DOWN RD0 /* Dec. */
#define BUTTON_UP RD1 /* Inc. */
#define BUTTON_MODE RD2 /* Mode. */
#define STOP1 RD3 /* Not in use. */
#define END_POSITION_IN RD4
#define HEAT_CONTROL RC4 /* Heat ctrl. */
#define DOWN RD6 /* Down2. */
#define END_POSITION_OUT RD5 /* Down1. */
#define BEEP RE2 /* Beep. */
#define STOP_OUT RC1 /* rc1. */
#define TEMPERATURE RA0 /* Temp. */
#define TEMPERATURE_ERROR RC5 /* NMI. */
#define WATCHDOG RA2 /* WD. */
#define REFERENCE_24V RA4 /* UV. */

init (void)
{
    int i;
    int j;
```



```

/* Register. */
OPTION_REG = 0b11111111;
CMCON = 0b00000111; /* No comparators. */
ADCON1 = 0b10001110; /* AD0 analog, right adjusted. */
ADCON0 = 1; /* AD on. */

/* Port. */
TRISA = 0b00010001;
PORTA = 0b00000000;
TRISB = 0b00000000;
PORTB = 0b00000000;
TRISC = 0b00101100;
PORTC = 0b00000000;
TRISD = 0b00011111;
PORTD = 0b00000000;
TRISE = 0b00000000;
PORTE = 0b00000000;

/* Timer. */
TMR1H = 0b11111010; /* Set high byte of timer1. */
TMR1L = 0b11111101; /* Set low byte of timer1. */
T1CON = 0b00110000; /* Timer1 control register. Prescaler = 1:8. */
PIE1 = 0b00000001; /* TMR1 interrupt enable. */
TMR1ON = 1; /* Start timer. */
INTCON = 0b11000000; /* Global/Peripheral interrupt enable. */

delay_100ms (1);
lcd_init ();
lcd_clear ();
lcd_goto (0);
lcd_writesc ("HOT SCREEN");
lcd_goto (0x40);
lcd_writesc ("HOT 2000 v4.00");
/* Factory reset. */
if (BUTTON_UP && !BUTTON_DOWN && !BUTTON_MODE)
{
    while (BUTTON_UP && !BUTTON_DOWN)
    {
        if (BUTTON_MODE)
        {
            while (BUTTON_UP)
                __delay_ms (50);

            if (BUTTON_DOWN)
            {
                temperature_select = 20;
                temperature_offset = 0;
                time_select = 5;
                press_counter = 0;
            }
        }
    }
}

```

```

    }
}

delay_100ms (20);
if (time_select > 60)
    time_select = 0;

if (temperature_select > 250)
    temperature_select = 0;

if (temperature_offset >= 10)
    temperature_offset = 9;

if (temperature_offset <= -10)
    temperature_offset = -9;

/* Sound when initializing is done. */
for (i = 0; i < 600; i++)
{
    BEEP = !BEEP;
    for (j = 0; j < 10; j++);
}

/* Temperature offset adjustment. */
if (BUTTON_MODE && !BUTTON_UP && !BUTTON_DOWN)
{
    char text[] = " TEMP OFFS ";
    char unit[] = " C";
    unit[0] = 0xDF;
    temperature_offset = change_value ((int) temperature_offset,
                                        -9, 9, text, unit);
}
}

```

### 5.1.2. LCD driver

The display driver written in C for the PIC16F877A can be found in attachment B. It's compatible with any display that is using a HD44780 interface.

### 5.1.3. Interrupt function

```

/* Timer. */
TMR1H = 0b11111010; /* Set high byte of timer1. */
TMR1L = 0b11111101; /* Set low byte of timer1. */
T1CON = 0b00110000; /* Timer1 control register. Prescaler = 1:8. */
PIE1 = 0b00000001; /* TMR1 interrupt enable. */
TMR1ON = 1; /* Start timer. */
INTCON = 0b11000000; /* Global/Peripheral interrupt enable. */

void interrupt
ISR(void)

```

```

{
if (PIR1bits.TMR1IF && PIE1bits.TMR1IE)
{
/* Tell external watchdog that the processor is alive. */
WATCHDOG = !WATCHDOG;

heat_timer_10ms++;
if (heat_timer_10ms%50 == 0)
    passed_500ms = 1;

if (START1 || START2)
    disable_press_timer++;
else
    disable_press_timer = 0;

/* Low voltage will freeze the program. */
if (REFERENCE_24V == 0)
{
/* Interrupt may happen before lcd is initialized. */
lcd_init ();
lcd_clear ();
lcd_goto (0);
lcd_writesc (" LOW VOLTAGE  ");
lcd_goto (0x40);
lcd_writesc ("          ");
HEAT_CONTROL = 0;
DOWN = 0;
END_POSITION_OUT = 0;
WATCHDOG = 0;
STOP_OUT = 0;
while (1);
}

/* Temperature error will freeze the program. */
if (TEMPERATURE_ERROR == 1)
{
/* Interrupt may happen before lcd is initialized. */
lcd_init ();
lcd_clear ();
lcd_goto (0);
lcd_writesc (" TEMP ERROR  ");
lcd_goto (0x40);
lcd_writesc ("          ");
HEAT_CONTROL = 0;
DOWN = 0;
END_POSITION_OUT = 0;
WATCHDOG = 0;
STOP_OUT = 0;
while (1);
}
}

```

```

TMR1ON = 0;
TMR1H = 0b11111010;
TMR1L = 0b11111101;
TMR1ON = 1;
/* Reset interruptflag. */
PIR1bits.TMR1IF = 0;
}
}

```

#### **5.1.4. Destroying eeprom memory**

After a month running the program it started to behave differently. The cause was an exhausted eeprom data position. This was a mistake and it was adjusted in a newer version of the program. The manual say that the eeprom memory can be written to about 100 000 times to a single position, we calculated that the actual average write cycle is well above 1 million times.

## **5.2. Previous production hardware update**

### **5.2.1. Stable boot in previous design**

The faulty boot was caused by a bad timing on the master clear which in this circuit is also used as a power on reset. By changing the capacitor value of the RC network that delays the master clear signal this problem was adjusted.

### **5.2.2. Noise reduction of AC/DC transformator in previous design**

The noise from the AD/DC module is noticeable when the circuit is idling but is less loudly when the module is supplying more than idle current. A workaround to this problem was then to put a load resistor on the module which reduced the noise. The module was later replaced in the new design.

### **5.2.3. Modification of the ground plane in previous design**

To make the device more stable the safety ground was removed from the ground plane.

## **5.3. The new design**

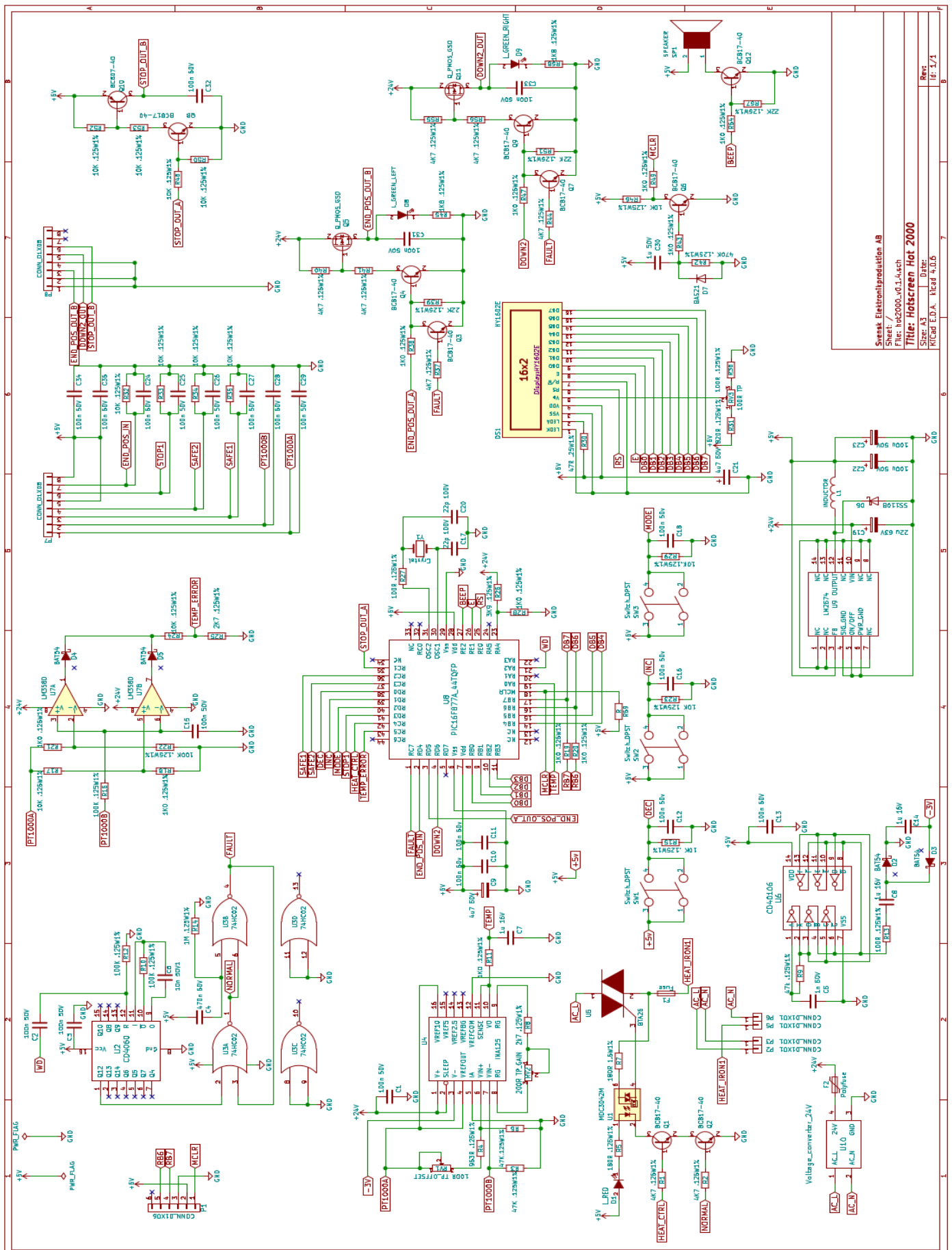
The main specifications of the circuit:

- 2-layer PCB layout.
- Controlled by a Microchip PIC16F877A microcontroller.
- Mean Well IRM-10-24 is used to transform 230v AC to 24v DC.
- Texas Instrument LM2574 buck converter transform 24v DC to 5v DC.
- Display is an Winstar OLED 2x16 character in white/black.
- 3 menu buttons, 2 operation buttons.
- External oscillating crytal.
- External master clear reset. Internal master clear available by software modification and the change of some components.

- In circuit serial programming and debugging available.
- External watchdog.
- Texas Instrument INA125 instrumentation amplifier as a temperature reading circuit for the pt1000 sensor.
- Faulty temperature detection by the use of a Texas Instrument LM358D dual operational amplifier.
- Position detection.
- Pneumatic valve control.
- 230v AC heating control by the use of a TRIAC.

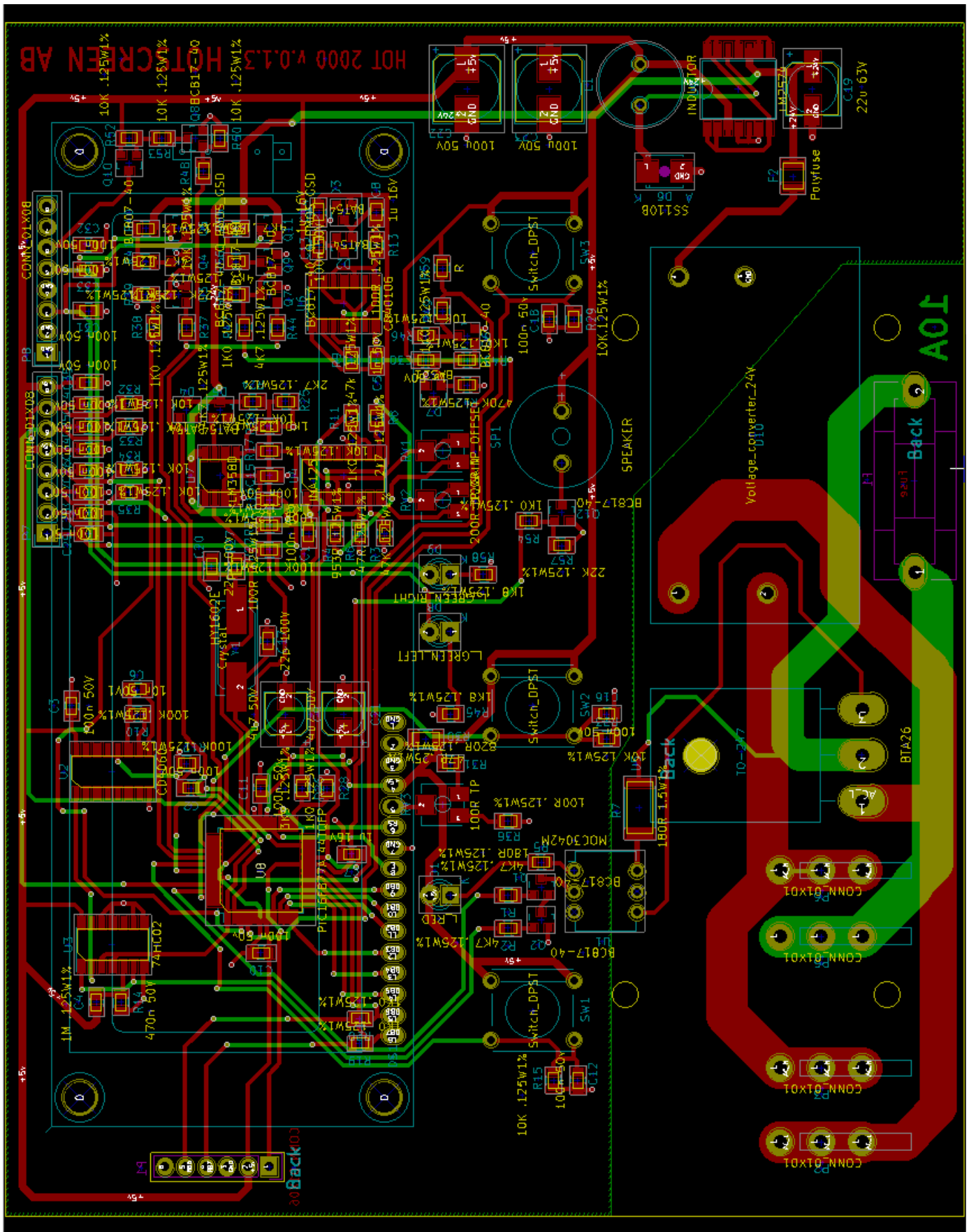
### 5.3.1. Schematic design

More information about the schematic design can be found in section 4.3 .



### 5.3.2. PCB layout

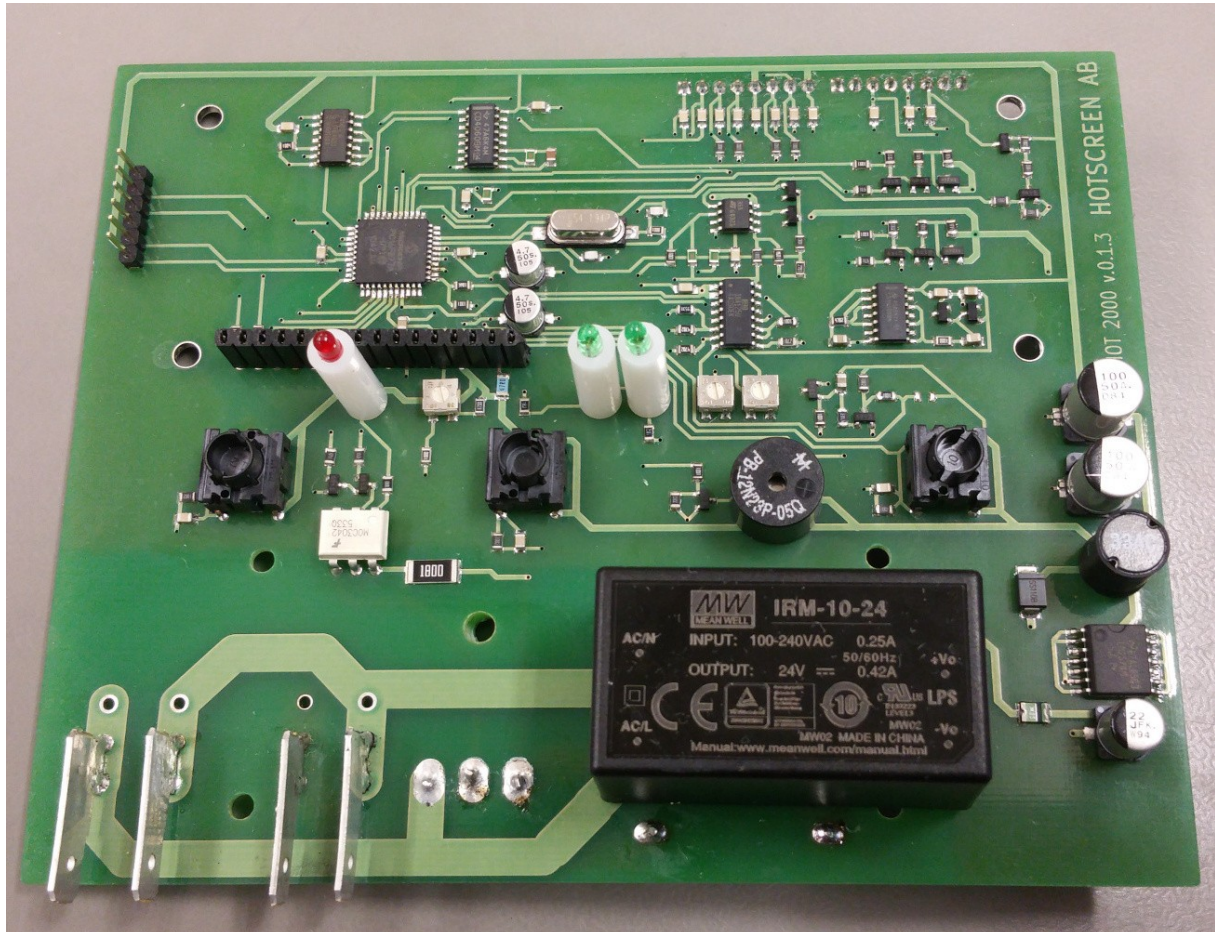
More information about the PCB design can be found in section 4.4 .



## 5.4. The prototype

The prototype PCB do not come with the silkscreen layer containing the text and markings. The only text you can see is the one in the copper layer.

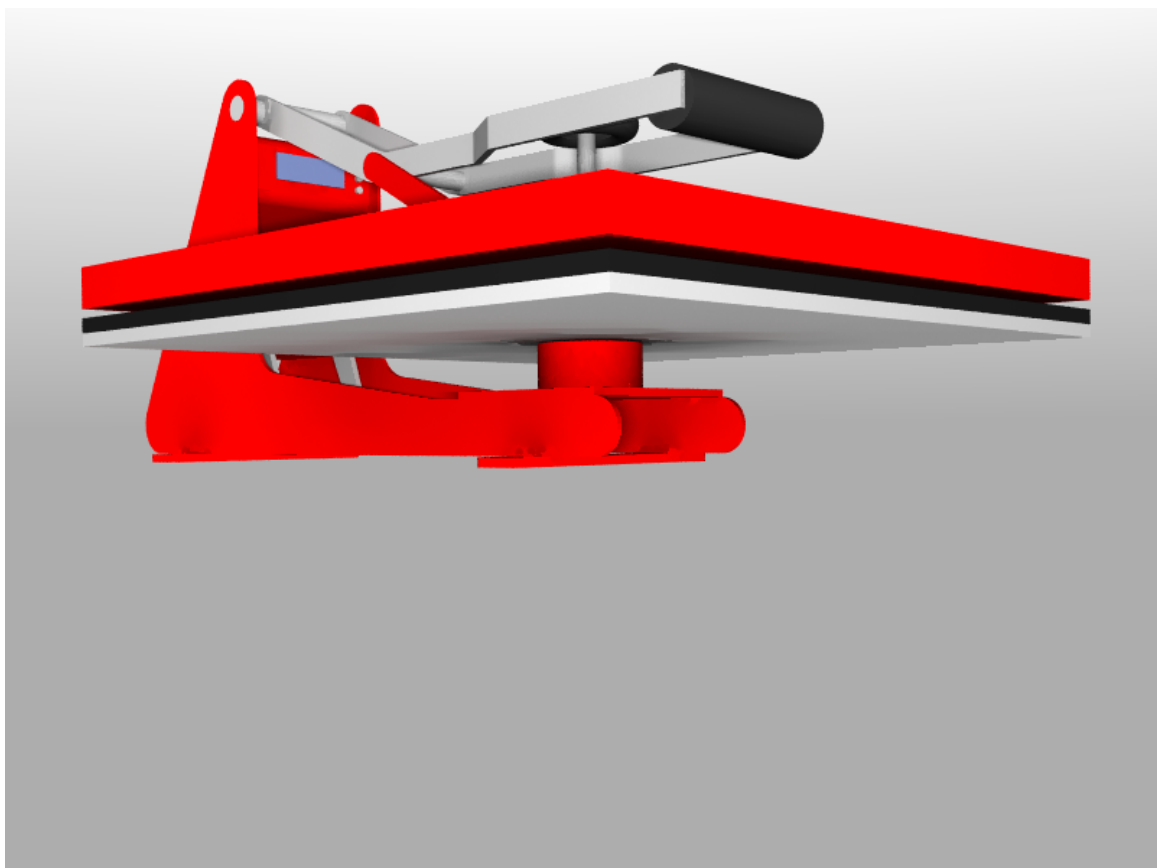
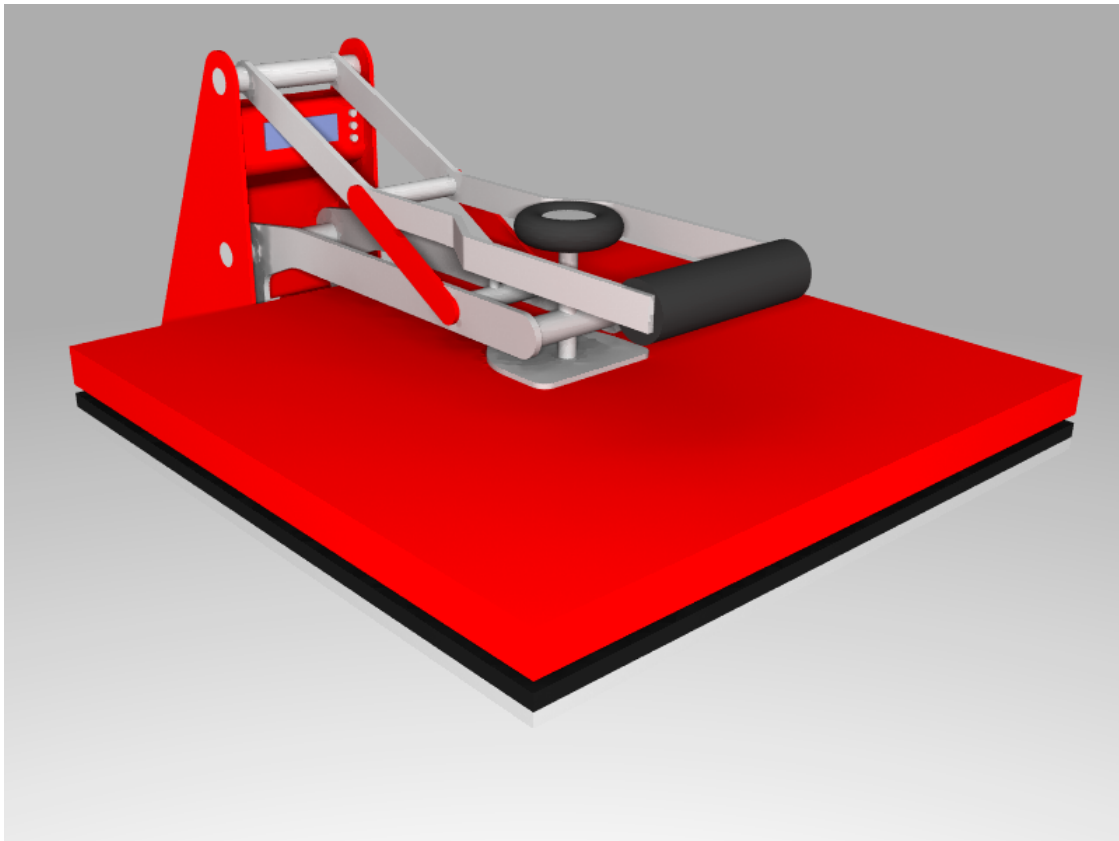
At first there were some issues because components was placed in the wrong place. Then there was a problem because of a poly-fuse that had a to low value. The AC/DC module got its footprint mirrored and needed to be attached by wires. The software needed minor modification to drive the new OLED display. When those issues got corrected the circuit worked as intended.



*Illustration 25: The complete circuit without the display attached*



## 5.5. The 3D model of an mid and entry level machine







## 6. Analysis and Discussion

### 6.1. Choosing a (FOSS) development environment

It's a great privilege to get to choose development environment by personal preferences. This is not always the case as customer may have certain demand, which is also a reason why it's good to master different environments. It may also be necessary to work with different environments to know which one to pick.

A vital aspect of any type of software tool would be the software licensing. This is especially true if you plan to add any tool to your product, which will need to have a license that allow it's usage in your product. A permissive license would do the trick, the taste of freedom is sweet. But we are also concerned with the freedom of others, hence we need a license that respects peoples freedom. This is why we prefer any version of GPL software license.

#### 6.1.1. Choosing editor

Choosing and mastering the right editor will enhance your productivity. We choose GNU Emacs for this purpose. Emacs comes with a steep learning curve for beginners but it will pay back, trust me! The vast range of features combined with the great number of script written for Emacs will provide many useful functions for different needs. If you currently use VI/VIM as your editor you'll may want to check out Evilmode in Emacs.

#### 6.1.2. Microchips biggest mistake

One thing I really like about Microchip is their nicely composed datasheets so thats not what this discussion is about. The problem is the lack of a good compiler toolchain. Microchip did developed their own toolchain back in the days, which weren't very good. Because of this a company called HI-TECH Software developed a compiler for the Microchip PIC family. This compiler was later bought by Microchip and is now sold as a PRO version compiler. They have also made a cripple freeware version of this compiler which packs the program with unnecessary instructions that makes the program run slow and take up a lot of memory space. They even advertise their pro version as being "up to 60% smaller and 400% faster code".

Microchip makes their money by selling microcontrolles and their revenue from pro compilers is nothing in comparison, so why are they selling a PRO version of their compiler? We can only speculate, but it may be because the finance department at Microchip wants a return on investment. The pro version will surely return their investment, but this has and will affect their business of selling microcontrollers which is vastly more important.

So why isn't there a mature FOSS compiler for the PIC microcontrollers (SDCC support for PIC is still under development)? Well the architecture of the PIC controllers is not an ideal when it comes to making a compiler for it. Combining this with the lack of help from Microchip has slow things down. This is sad as everybody would benefit if there was a collaboration between Microchip and the FOSS movement.

Because of the limited development of open source tools for the Microchip PIC family one may want choose another microprocessor. The Atmel AVR microcontrolles uses the GNU AVR toolchain which is bases on the GNU C Compiler (GCC). This is a mature compiler toolchain with a great reputation. AVRDUDE is a widely used uploader for the AVR

microcontrollers that support a wide range of programmers. Lady Ada supplies a tutorial on how to get started programming your Atmel AVR [21].

### **6.1.3. KiCAD**

KiCAD has proven to be a great EDA with some really nice features and a solid base. They have recently added some vital tools for high speed electronics which makes it's useful for all kinds of designs. It's possible to check for design errors in both for the schematic and the PCB layout by defining your rules and run a check. The user library that is available for download has proven to be very useful by saving a lot of time when creating components. A very nice feature is the 3D modeller which makes it possible to do a 3D model of your PCB. Customer who do their mechanical CAD in house do often appreciate the 3D PCB model when designing the casing.

I also find the community of KiCAD to be helpful and the development crew seems to be working hard on improving the software. This great community is also the reason why there is so many open hardware designs made in KiCAD.

## **6.2. The conservative approach**

Some important factors in this project was time, money, robustness, and a design that is future proofing regarding its manufacturability. They also requested that the design should be possible to change and upgrade if needed. It shall also be backwards compatible with their already produced machines. With these requirements we decided to go with the conservative approach when designing the circuit. Keeping som aspects of the original design made for at faster and cheaper development with less problems and bugs because it's already been proven by time.

### **6.2.1. Keeping the old processor**

The reason behind keeping the old processor was that we already developed a program for this processor and application to the current machine. The PCB layout is done in such a way that it doesn't require any change if the processor were replaced by its successor. It's likely that the program will be ported to the newer processor and used in the design in the future .

## **6.3. The clean sheet approach**

### **6.3.1. Going Atmel**

Hopefully the news that Microchip bought Atmel won't change the reasons why we prefer to choose an Atmel microcontroller. The mature FOSS tools available for the Atmel AVR is the main reason behind this choice. The architecture of Atmel processors and their instruction sets is somewhat simpler which may also be the reason why there is better software tool available to them. Atmels proprietary IDE is only available for the Windows operating system so it's useless and wont be needed. Another drawback is that the Atmel microcontrollers is slightly more expensive.

### **6.3.2. The pitfalls of modern displays and technologies**

It's tempting to choose a high pixel density graphical LCD with touch feature as these are getting ridiculously cheap nowadays. Well off course they are not as cheap as those old text interface LCD and you will need a more expensive micro controller to control them and the

development cost will increase. But in many cases this improvement will make the product more attractive so that it outweigh the extra cost. So what's the hangup then? Well the development of these technologies has been so intense that a 3-4 year old display will most likely have its production discontinued. This is not a problem for big companies with big production series as they do afford and do want to upgrade and redesign their products after a couple of years. It's even the usual case that when let's say a big phone manufacturer discontinues one of their phones, the producer of that display will also stop making the display. For smaller companies with smaller production series this will hit them hard as they may calculate with a design life of up to 10 years to repay the development cost. This is why an engineer needs to be careful when selecting hardware, especially if the technology is under heavy development.

#### **6.4. Thought about a entry level machine**

The machine sold today is intended for high volume production in industrial applications. Hot Screen have a wide range of customer buying their screen prints and not all of them are willing to pay the price for a high level machine. The small shops may look for other alternatives and buy a manual machine from other sources. The reason why we proposed two designs of an entry level machine is to fill this need. Hot Screens customers may rather buy an entry level machine from Hot Screen than from any other sources as they want a machine that is tailor made for the screen prints of Hot Screen. As manual machines do not suit high volume production we don't think that this machine will lower the sales of the current machine.

## 7. Conclusion

### 7.1. Choosing a suitable project for your first PCB

A good recommendation for any project is to choose something that's not too far away from your own knowledge. In this topic I'll give you some suggestion on what that may be.

#### 7.1.1. The old school project

The old school project will be a good option for anyone that do not have any former programming knowledge and can't wait to have their own PCB made. By old school I'm referring to any design that excludes a microprocessor. As mentioned in 7.2.2 you'll might want to use a video tutorial to understand the basics in your EDA of choice. A Nixie Tube or a Vacuum fluorescent watch would be a nice project to start of with.

#### 7.1.2. FOSS will kickstart your microcontroller

A super fast way to get amazing things done would be through the use of a Complete and FOSS development environment for microcontrollers suitable for beginners. Be sure to check out the open source Arduino project which is both a complete IDE, hardware platform and community, There is many good tutorials and a lot of information about Arduino, both official and unofficial. There is also a ton of open source projects available for the Arduino that is worth to check out.

#### 7.1.3. Going professional

At some point you might feel kind of restricted by using an IDE and library such as Arduino because it acts a layer between you and your microcontroller. You'll need to get closer to that hardware in the microcontroller to get more control. From now on your primary source of information for your microcontroller configuration and setup should be its datasheet. Microcontrollers from the same fabricator do often have similar datasheets and if you understand one of their microcontrollers you'll easily understand the rest of them.

### 7.2. Recommendations on how to get started

#### 7.2.1. Tutorials

Tutorials can be a great way to learn new things. They are kind of like a cooking recipe with step by step instructions on how to do something. If you follow the recipe you'll most likely succeed. A good tutorial also explains every step in details so that you'll be able to alter the recipe to your liking and use part of it for your own applications.

#### 7.2.2. KiCAD video tutorial

If you want to learn any program with a Graphical User Interface (GUI) a efficient way would be by watching somebody use and explain the program. This is also true for other skills that contains physical real world activities. Research has shown that the more senses you activate the more efficient you'll learn and remember. Well you may not always have a friend that's available to teach you what you want to learn, luckily there is a ton of video material on different topics available on the internet. I did find the "Getting to blinky" tutorial to be of great help (25). The drawback of video instructions and lessons is the difficulties to repeat and rehearse the information in a speedy manner. A god complement when watching a video

tutorial would be to take notes, maybe even as simple as noting the time in the video of different subtopics.

### **7.2.3. Using a reference design**

Regardless of field, an engineer should study similar project/products to his own. This can give useful information on how to design and improve such a product. One may also find that a completely different idea may share some similarities and can be used to pick parts from. This is obviously very common in software design as libraries are almost always used and many programs is released together with their source code under different FOSS licenses which may tolerate reuse of its code.

It's unfortunately not that common with free/open hardware designs. But open hardware seems to be gaining momentum and you might find some interesting stuff in the open hardware repository [29], at opencores [30] or at any hacking community. Many manuals and datasheets of IC:s will contain a reference design of its intended use. Then there is the old fashion way of purchase and disassemble a product to find out whats under the hood.

Luckily there is a ton of IC:s available for purchase and together with programmable devices such as the field-programmable gate array (FPGA) you may never need to design an application-specific integrated circuit (ASIC).

### **7.2.4. How to learn the C program language**

As I mentioned in §7.2.1 a tutorial is a great way to learn new stuff and you might choose this route to begin learning C, check out [26]. This is especially true for the part of setting up a develop environment and compiling a program for the first time. The C programming language is such a great tool that I recommend you to learn it thoroughly. A tutorial may not get you all the way so you may also want to use other sources to expand your knowledge. A good book will serve both as reference material and a learning source. I highly recommend "The C programming language" ANSI version which is written by the fathers of the C language, Brian W. Kernighan and Dennis M. Ritchie. Regardless of which book or tutorial you might choose be sure to check out variable types, operators, statemets, functions and pointers before doing your first project. Every C programmer should also know how to get around with the Standard C Library and its documentation [27].

When you understand the basics a good way to improve your skills is to start a project that you'll find exciting. You may start a project from scratch or hack an existing project into your liking. A smart choice for your first project is to do something quite small and not to complicated. But if you're really fired up on something that is a little bigger it's not a bad idea ether. Just be sure to find a good source of information on the topic, maybe a tutorial or a reference project.

### **7.2.5. Datasheets and manuals**

Datasheets and manuals directly from the designer/producer of the hardware/software should always be the primary source of information when using that product in the design. Rewritten tutorials and other information may be more efficient and easier to understand, especially for beginners. But it's also important to have a look at the datasheet. As the datasheet from the producer is always the first hand source of information, other sources do often use this first hand source to make the information more compact and comprehensible. This may lead to loss of information and the rewriting may not always be correct. To better understand this you may read the chapter 3.3 to 3.8 and compare it to the official KiCAD documentation [17] .



### 7.2.6. Getting help

Most datasheets and manual requires that the reader have prior knowledge to its field of technology. There is a ton of information available on most electronic subject. The amount of information available is so large the hardest part is in many cases to pick the right one. The reference sources in chapter 8 could be of use. They might not be the best but they have helped me get this far. A fun way to learn new stuff is by studying other project and maybe even find ways to improve the product, in this case you may look out for open source projects.

Once you understand the basics a really good and fun way to learn it by doing a project as discussed in section 7.1 . You may do a project with someone that have a certain knowledge of some part of the project and you understand the other part. This is a great way to exchange information and knowledge which is also fun. You may search for a local hackerspace to meet with other people that shares the same passion. A hackerspace is a great way to find project partners, learn from others and ask for help and borrow equipment.

You may also search for an online community that into the same field of technologies as you are. Don't forget to check out Stack Exchange which is a great site for asking tech related questions, you'll may already have the answer in their great database.

## References

1. Barnett R., O'cull L. and Cox S., 2007: Embedded C Programming and the Atmel AVR, Cengage Learning: 2 edition, Delmar.
2. Bergström L. and Nordlund L., 2012: Ellära: krets och fältteori, Liber AB: 3 edition, Stockholm.
3. Brown S. and Vranesic Z., 2005: Fundamentals of Digital Logic with VHDL Design, McGraw-Hill: 2 edition, New York.
4. Gough B., 2004: An Introduction to GCC, Network Theory Limited, Bristol.
5. Horowitz P. and Winfield H., 2015: The Art of Electronics, Cambridge University Press; 3 edition, New York.
6. Jacobson S., Widén L. and Osbeck M, 2016: Don: för Mekatronikingenjörsprogrammet, Chalmers campus Lindholmen; institutionen för Signaler och System, Göteborg.
7. Jacobson S., Widén L. and Osbeck M, 2016: Givare, Chalmers tekniska högskola; institutionen för Rymd och geovetenskap, Göteborg
8. Kernighan B. and Ritchie D., 1988; C Programming Language, Prentice Hall; 2 edition, New York.
9. Nordlund L. and Wiklund I, 2012: Grundläggande Elektronik, Liber AB: 2 edition, Stockholm.
10. Osbeck M. and Hult G, 2015: Styrteknik EI, MeI, Chalmers campus Lindholmen; institutionen för Signaler och System, Göteborg.
11. Raymond E., 2000: The Cathedral and the Bazaar, O'Reilly Media, Sebastopol.
12. Stallman R., 2010: Free Software, Free Society: Selected Essays of Richard M. Stallman, Free Software Foundation; 2 edition, Boston.
13. PIC16F87XA: 28/40/44-Pin Enhanced Flash Microcontrollers, Microchip Technology Inc, 2013 Phoenix, <http://ww1.microchip.com/downloads/en/DeviceDoc/39582C.pdf> (Acc 2017-06-10)
14. PIC16(L)F18857/77: Full-Featured 28/40/44-Pin Microcontrollers, Microchip Technology Inc, 2016 Phoenix, <http://ww1.microchip.com/downloads/en/DeviceDoc/40001825A.pdf> (Acc 2017-06-10)
15. MPLAB® XC8 C Compiler User's Guide, Microchip Technology Inc, 2012 Phoenix, [http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB\\_XC8\\_C\\_Compiler\\_User\\_Guide.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_XC8_C_Compiler_User_Guide.pdf) (Acc 2017-06-10)
16. Microchip Developer help, Microchip Technology Inc, 2011 Phoenix, <http://microchipdeveloper.com/> (Acc 2017-06-10)
17. KiCAD EDA Documentation, KiCAD, <http://kicad-pcb.org/help/documentation/> (Acc 2017-06-10)
18. Octopart component search engine, Octopart, Inc., New York, <https://octopart.com/> (Acc 2017-06-10)

19. Alibaba e-commerce, Alibaba, inc., Dongguan, <https://www.alibaba.com/> (Acc 2017-06-10)
20. Pinguino, <http://www.pinguino.cc/> (Acc 2017-06-10)
21. Lady Adas Atmel AVR tutorial, Lady Ada, <http://www.ladyada.net/learn/avr/index.html> (Acc 2017-06-10)
22. Small Dev C Compiler, <http://sdcc.sourceforge.net/> (Acc 2017-06-10)
23. KiCAD EDA Documentation on how to make a footprint, KiCAD, [http://docs.kicad-pcb.org/stable/en/getting\\_started\\_in\\_kicad.html#make-component-footprints](http://docs.kicad-pcb.org/stable/en/getting_started_in_kicad.html#make-component-footprints) (Acc 2017-06-10)
24. GNU coding standards, Free Software Foundation, 2013 Boston  
<https://www.gnu.org/prep/standards/standards.html> (Acc 2017-06-10)
25. KiCAD video tutorial “getting to blinky”, Contextual Electronics, 2016 Chicago  
[https://youtu.be/JN\\_Y93RTdSo?list=PLy2022BX6Eso532xqrUxDt1u2p4VVsg-q](https://youtu.be/JN_Y93RTdSo?list=PLy2022BX6Eso532xqrUxDt1u2p4VVsg-q) (Acc 2017-06-10)
26. C programming tutorial, Tutorialspoint, 2017 Hyderabad,  
<https://www.tutorialspoint.com/cprogramming/index.htm> (Acc 2017-06-10)
27. C language library, <http://www.cplusplus.com/reference/clibrary/> (Acc 2017-06-10)
28. Stack Exchange sites, Stack Exchange, New York 2017, <https://stackexchange.com/sites> (Acc 2017-06-10)
29. Open Hardware Repository, <https://www.ohwr.org/> (Acc 2017-06-10)
30. Opencores, <https://opencores.org/> (Acc 2017-06-10)