ELECTRONIC LOCKING FOR HIGH VOLTAGE GROUNDING SYSTEM

Eduard Boja

Josep Font Teixidó

10 June 2014

Contents

The present document includes all the documentation and information regarding this final degree project. This project is basically designed with the idea to improve the security and make it easier for the people who have to deal with high-voltage lines to do their work. Such information includes all the possible topics going through the very first ideas and reaching the conclusions in the end. Also it explains all the hardware and software that it includes and how they are related one another. Also it explains the desiging procedure that has been followed.

It is of crucial importance to mention that this project has been designed and created under the request of the company Sofamel, SL to fulfill it's needs and ambitions. They firstly presented a need of improving a system that had always been mechanical, and our objective was to present a solution. Our idea was introducing to this basic system a few electronic components that combined achieve the objective of replacing the mechanical system. Afterwards, we presented our solution to them and they valorated it and idicated what could be improved in the case of the need to continue with this project later on.

1 Introduction

The main objective of this project is designing a security system for those people who work at high voltage lines. Basically, we found out that the system that was used to connect the three-phase line to the ground was way too rudimentary. Basically, a key was used to enable the worker to connect every line to the ground. This lead to situations like loosing the key and having to just throw away the whole equipment.

In the technological world like the one we live in, we decided that probably we could design some kind of electronical system to perform that activity and improve the actual system. The basic operation we must perform is authentication, as it is the main issue when we are talking about security. In the previous system this was done by using a simple key but as we said this was quite of a problem. Our idea, is to create an universal system to be able to access to any system we have.

The basic idea is that we have system with two way clamp that will be connected respectively to the ground and to the high voltage line. Previously, any of the two clamp could only be operated when the key was put inside the rabbet. The fact is that the normal way to operate is the one that follows:

First, we put the key into the rabbet of the clamp that is going to be connected to the ground. Once we did this, we are able to manually open the clamp and connect it to the ground.

After this, we can remove the key from the first clamp and put it into the rabbet of the clamp that we will connect to the high voltage. Once this is done we can manually open the second clamp and connect it to the voltage.

Look out that when we do this, the first clamp can't be operated at all, so when we have both clamps connected respectively, the key can't be removed from the second clamp. The only way to remove the key is dissconecting the clamp from the high voltage line. This is the security system that we want to improve electronically.

What we will develop is an external system that will identify every pair of clamps as something unique. This way, we will be able to use this key in any pair of clamps that we have and will always work, taking into account that we connect the system properly. Later on we will explain how we will do this. To communicate our external system with the pair of clamps, we will use a bidirectional communication bus. The one we have choosen is the I2C system which enables us to do exactly what we want.

This system basically consists in one master controlling a certain number of devices connected to the bus. In our case the AVR will be the master, controlling all the slaves connected to the bus. The idea is basically to connect the AVR to the pair of clamps, detect wheter the identification has been a success of a failure, and afterwards releasing an interlocking that will enable us to manually open the clamp.

The idea is trying to reduce to the minimum the energy that will be needed to this operation, so as to reduce the number of necessary batteries.

This project has four basic phases:

- Investigation: Information resarch regarding the project realization. This includes material, software, etc.
- Development: Hardware integration and software development to include all the parts in a single one
- Presentation to the company: As it is a project designed for a company, we will show them our design to exchange points of view

- Improvements: After having exchanged points of view with the people in the company, we will do the necessary changes in the application to fit their expectations

I must be said that the Development phase has two different parts, the first one is to design a prototype to fulfill the project expectations. Secondly there is the need of developing a group of boards to achieve the final objective of implanting the project in the final device.

1.1 Investigation

The first step in every well designed project is the investigation. The basic idea is gathering all the needed specifications and start looking for suitable solutions. Basically, the idea is to strip the project into small parts and start trying to find solution to each of this parts and as soon as we solve them start to gather them into bigger ideas and, at the same time, problems.

We knew that we wanted to interact with many possible slaves at the same time, beeing commanded by a master so we needed a system that could be used this way.

What came into our mind was using Arduino and the i2c bus protocol to communicate with different slaves. In fact this is what we chosed, and to do this we used the i2c.c library to get some extra time.

So to begin we first thought in what we wanted to do, and in consequence, what we needed to do it. The first thing that we were asked was to perform an authentication system that replaced the key that was used since the begining and that also did not need any power supply.

The first idea we had was the EEPROM memory, able to store data for a long time without needing any power supply. So the first thing that we did was search for a pair of suitable EEPROM memories for our project. Soon after this be begun developing a library for the interaction with the memories and for making it easier to deal with them. The way to use the memory is easy, we will only use a small part of the memory (1 byte) and we will compare if the value stored at both memories is the same. In the case of getting a good authentication, we can proceed with the operation of the clamp, else we won't.

When we finished with the memories and the authentication, we thought it was a good idea to use a display to show the user in which state was the device. Concerning this matter we used a very simple display that we already used in some projects during our degree. The usage of this display will be very simple, it will only show the user what is the system doing at that exact moment. For example, when the user connects the device to the clamp and presses the button, the device wakes up and starts with the indentification of the clamp. While doing it, it prints on the screen of the display, whether the authentication has been a failure or not.

When the authentication is over we bump into the serious part of the project, that is the physical operation of the clamp. We needed something that, in the case of a failure in the authentication, would not allow the person working with the clamp to open it. The idea we had is to move an interlocking 90 degrees so as to enable the clamp to be opened manually. In the case of a failure during the authentication the interlocking would remain closed and the clamp won't be opened.

After having a clear idea of how our design would work, we started thinking what could we be using to move the interlocking. The best idea of which we could think was a bipolar stepper motor with which we would be able to excacly move a certain interlocking 90 degrees. It must be said that this will be done two times like with the EEPROM's because we have a pair of clamps so each one needs it's own interlocking.

The next thing we thought of is the battery needed to make the arduino work. The arduino needs 5V to be working and will only need the power to work himself as the only thing that will be doing is eventually moving the stepper motor 90 degrees. We will later on do some calculations to know how long would a battery last and what kind of it do we need.

The last thing we thought we needed was a pair of buttons to operate the whole device, and that will have a function that we will explain later on.

1.2 Project Explanation

1.2.1 Introduction

During this chapter we will be explaining the basis of both our project and the original one, that is the one we will be using as example. From the original one we will be keeping the physical design as we just want to use it to install an electronic system.

Furthermore we will be using the same concept of key to open the system, but instead of using a physical one we will be using an electronic one. Also the design of our project is structured in two basic phases. The first one is designing a prototype for our system, and as soon as it works, we will start designing all the physical electronics and board layout for our final system.

1.2.2 Existing system

At this point, you may have a more or less clear idea of how our project works but how did the existing systems work? And also what are their basic parts and what needs to be improved from them?

The previous systems were much more basic and mechanic but the problem is that they did not have any external security at all. The system was completely mechanic and that's what we wanted to change.

The basic idea of the existing system is the one that follows. We have a pair of clamps, one of which can be opened manually to ground it, meaning connecting it into the ground. While you don't have the first clamp closed you are unable to open the second one. Once you grounded the first one, you can open the second one and connect it to the high voltage line.

When you have both of them connected and you want to unplug the system, you have to do it backwards. By this we mean that you can't unplug the ground clamp before desconnecting the second one that is connected to the high voltage. First you have to unplug the second clamp and secondly the ground clamp.

In the following picture we can apreciate one of the pairs of the clamp that will be connected to the ground.

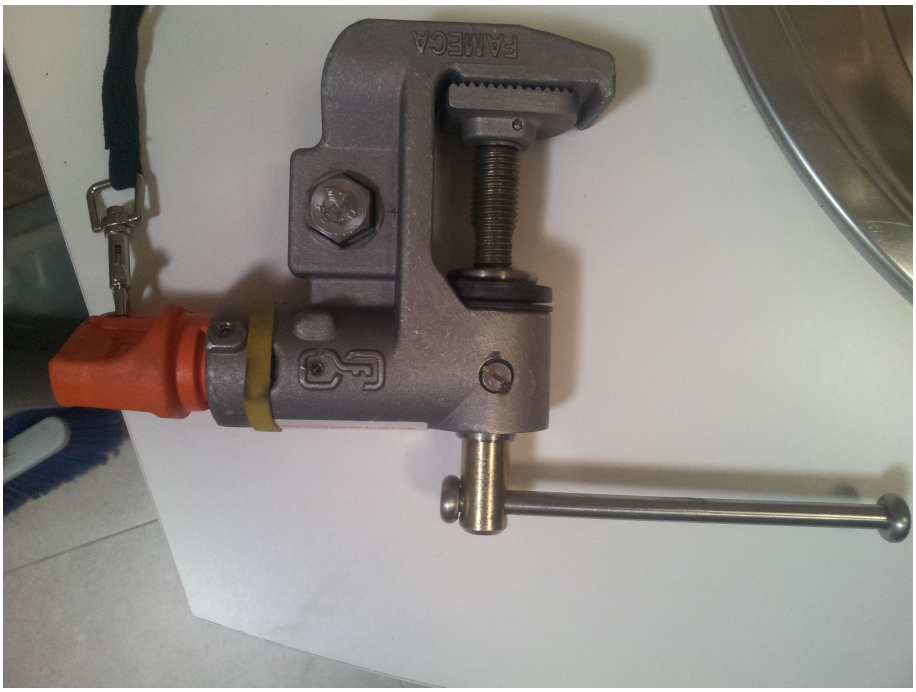

Figure 1.1: One pair of the clamp

This is something made completely mechanically and we want to perform this same operation but using our electronic system. Basically the physical part and apearence of the clamp needs no change at all, what we want to do is changing the security system that is operated with a key, by an electronic authentication.

In the following image we will see how the security system works in the Existing system.

Figure 1.2: Key of the clamp

1.2.3 Our system

As it has already been explained, the basic idea of our system is to transform a purely mechanical system into an electronic one. This will be made using an external device that will control the pair of clamps using an Atmega328p processor. The idea is to use an external device as if it was a key, and it fact it will be one, but not a physical one but an electronic one.

Our key will be an electronic external device with a display, a microprossesor and a connector that once it is plugged to the clamp it operates as a key. The idea is that the external device checks the value of a certain register in each of the pair of clamps and in the case of beeing the same value, proceed with the authentication. After this the operator will have the oportunity to open the first locking and connect the clamp to the ground, after this the second clamp will be available to be opened and connected to the ground also.

As we already explained how our project worked, we will explain it now using a diagram so that you can get a clearear glance of how the system interacts with the user and with the clamp.

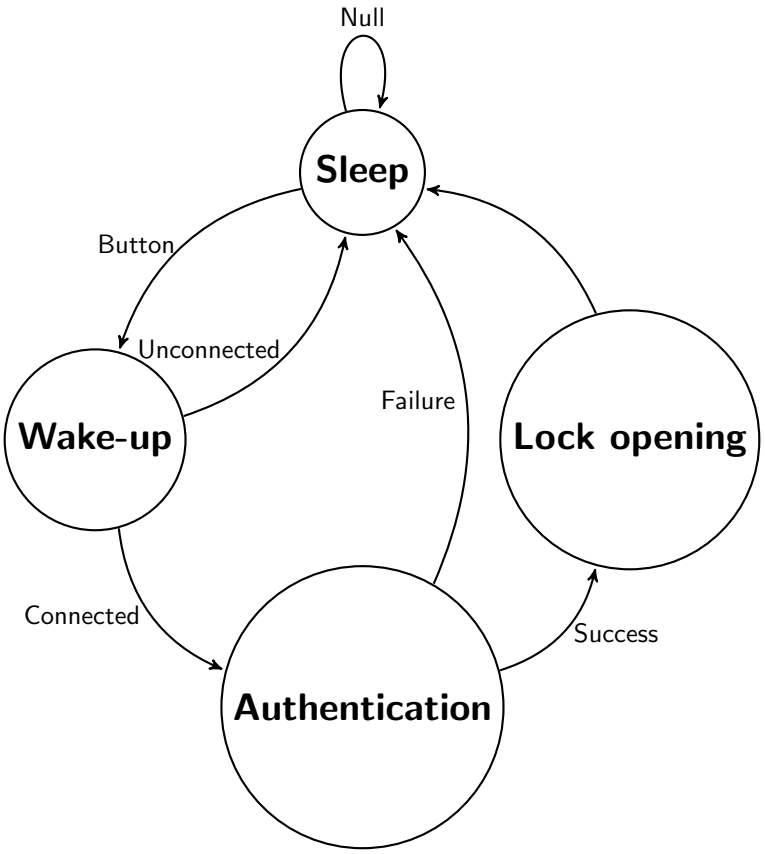

Figure 1.3: State diagram for our system

1.3 Meetings with the tutor and the Company

While develoving the project, we performed several meetings with both the tutor responsible of the project, Josep Font, and the Company. This has been done to know at which point we are when referring to achieveing the project expectations and specifications.

During this meetings I showed physically how the device was working and evolving since the last meeting and also I explained what was the next step we will be developing.

After my explanations it was the turn for the tutor and the company to tell me what improvements could be made and also what modifications they wanted me to do in the system.

It must be said that usually the meetings with the company were not made directly, meaning that the tutor had a meeting with the company and afterwards it was only me and the tutor doing a project evolution meeting.

2 Hardware description

2.1 Introduction

Certainly, one of the most important parts in any project is the hardware. As we already explained, our system consists of a very few hardware parts as it is a very basic electronic system. The basic idea consists of an external device that will be connected to any clamp and that will make it work. We will later on explain what this external device consists on.

The way we will connect the external device to the clamps will be using a pair of Db9 connectors, one for each of the pairs of the clamp. The external device will be the one containing the processor and all the user interface parts. This includes the display and a pair of buttons. This external device contains the most of the hardware parts our project will have.

The hardware inside the clamp won't be acessible to any person and it will be only for identificating any pair of clamps as something unique. Furthermore using the Db9 connector it will be the clamp board the one moving the stepper motor inside the clamp to enable the worker to externally open the clamp.

It is very important to understand that first we will design a prototype for testing our hardware and software and afterwards we will design the final board that will be inside the device using Eagle.

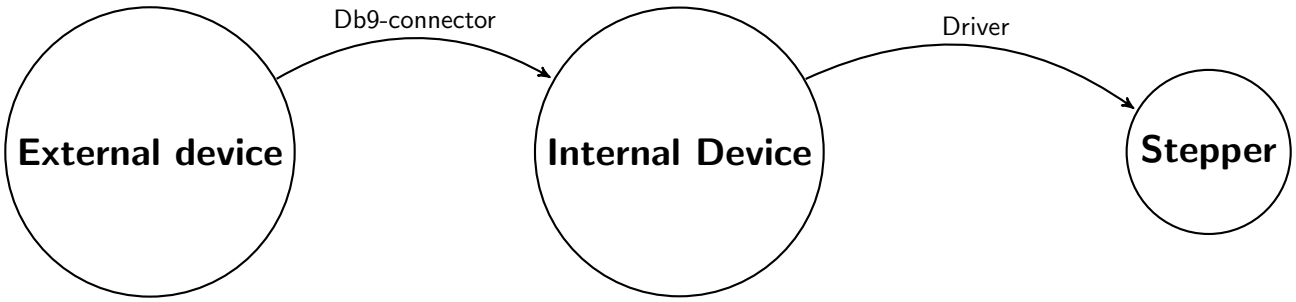

Figure 2.1: State diagram for our system

2.2 External device

This external device consists on a system that has as objective to control the whole clamp. Meaning this we understand interacting with the user, and sending the orders to the clamp. Firstly we designed a prototype to make all the tests and configure all the hardware properly. Afterwards we designed the board layout with all the components together so as to be integrated in a single device.

The board contains the following hardware, whoose operation will be explained lately:

- One Atmega 328p processor
- Two Capacitors of 22pF and one of 100pF
- One 100KOhms Resistor, one 2kOhms Resistor and two 4.8KOhms resistor
- Two Push-up buttons
- One 16Mhz Crystal oscillator
- One MCCOG21605B6W Display
- Two Db9 Output connectors
- One Double Input pin
- One ISP Input connector

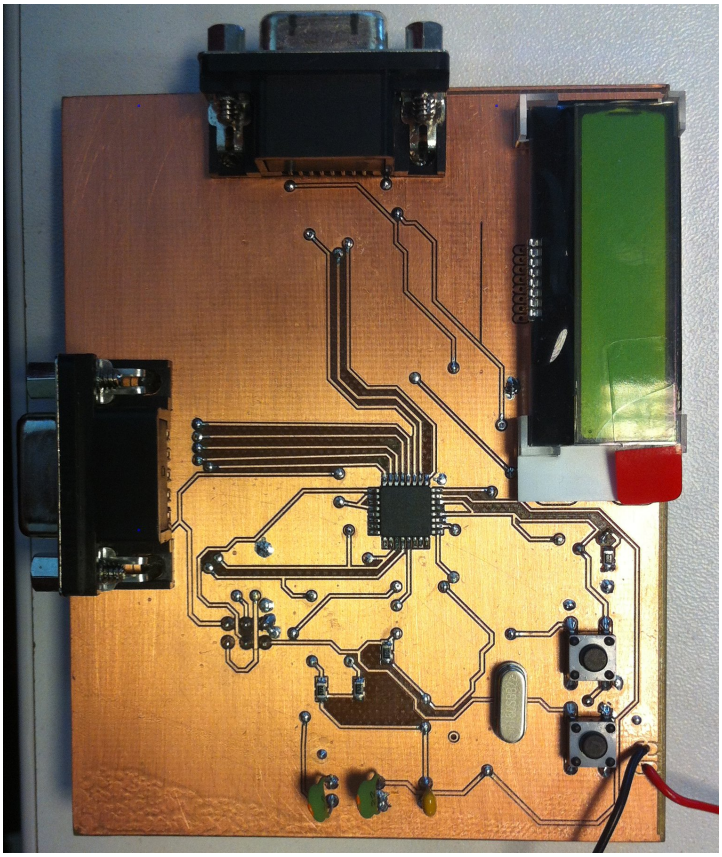

Figure 2.2: External device board

In the figure above, we can see the main board we designed for our project. This board is responsible for all the system configuration and operation as the rest of the system is pasive and only waits for orders from the main board.

2.2.1 Microcontroller

The microcontroller we will be using is the Atmega 328p as it is the one that best fits ours project needs. To use our microcontroller we added to it when designing the board schematic the necessary components to adapt it to something similar to the Arduino board which is the one we used for our prototype. So as to do this, we added a pair of push-up buttons, some resistances as well as capacitors and a 16Mhz Oscillator Crystal.

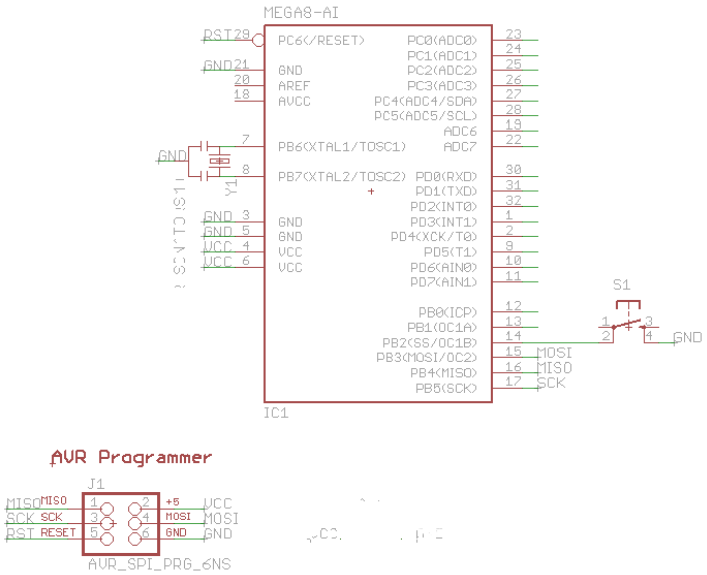

Figure 2.3: Atmega Original Schematic

In the picture above we can see a picture of the the Atmega328p schematic, in the picture below you can see the physical aperance of the microcontroller welded to the board.

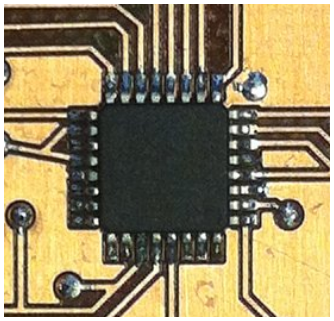

Figure 2.4: Atmega 328p

The utilities that we will be using from our processor are the one's that follow:

- Eight digital outputs: PD4-PD7 and PB0-PB3
- SDA and SCL bus outputs
- Sleep Mode
- Voltage and Ground outputs
- PC6/Reset as Reset Button
- PD3/INT1 as a Digital input

The eight digital outputs will be used to control the two steppers, one in each of the pair of clamps. What we do is just setting the voltage high (5V) or low (0V). The SDA and SCL i2c bus outputs are used for the communication with the E2proms and the display, also it is our crucial importance that we connect the Voltage output and the Ground. The way to operate is the following, we connect the power supply and the ground to the processor and then it is the processor the one that powers up the rest of the system.

The reset is a very important matter for a system like this, so we connected a push-up button to the reset pin in the processor so as to reset the system as soon as the push-up button triggers up. Also we have connected to the PD3 pin another push-up button. This pin and his twin PD2 are the only ones that can tell the Atmega to completely wake up when it is asleep so it is a very important part of our project also.

In the following table we can see a brief explanation of the ports that we are using from the processor and what are we using them for. For a bigger detail you can check the schematic attached at the annex.

Microcontroller Pin	**Name**	**Description**
PC6	Reset	Atmega reset pin, used for general reset.
AREF/AVCC	Voltage input	General processor power.
GND	Ground	General connection to the ground.
GND	Ground	General connection to the ground.
PB6/PB7	XTAL1-TOSC1/XTAL2-TOSC2	External crystal oscillator connection
PC4/PC5	SDA/SCL	i2c bus connectors
PD4/PD5/PD6/PD7	T0/T1/AIN0/AIN1	First four outputs used as stepper motor controllers
PB0/PB1/PB2/PB3	ICP/0C1A/0C1B/MOSI	Second four outputs used as stepper motor controllers
PB0/PB1/PB2/PB3	ICP/0C1A/0C1B/MOSI	Second four outputs used as stepper motor controllers
PD3	INT1	Input pin used for waking up the processor.
PB4/PB5	MISO/SCK	Input pins used for programming the processor.

Figure 2.5: Atmega 328p connections

2.2.2 Power

To power up our system we will be using four 1,2V AA NiMH Batteries PRT00335 able to supply 2500mAh. They will be connected right into the external device, and consequently to it's board. Our system is almost all the time asleep and it only wakes up for eventually authenticating the system and moving the steppers and right after this it falls asleep again.

The boards on the clamps have no power supply connected to them so once they have been programmed they remain unpowered until the external device is plugged in. This is the reason why we used e2prom memories. This way the memories can remain unpowered for a long time and still be able to keep stored it's authentication values.

In the following image we can see an example of one of our batteries.

Figure 2.6: System Battery

The Atmega current consumption is of the order of mA when it is awake and 0.1μ A when it is asleep. Knowing this we can make an apoximation of how long would our battery last. If we get in worst possible case, the Atmega can reach peaks of 15mA consumption when working at full speed with the oscillator. In this case we would get a life of 160 hours which is quite a lot, because our Processor works only for 30 seconds every time it is awakened, then it falls asleep again.

If we get into the real case, the Atmega is awake for no longer than 30 seconds, and then it goes to sleep again. Knowing this and getting an average that for every time the arduino stays awake it has been 20 min asleep, we can make the following calculations.

Asuming the proportionality we already stated between the time it is asleep and the time it is awake, we can calculate the average current consumption of our system.

$$Current = \frac{(0.1 * 10^{-3}) * 20 + (15) * 0.5}{2} = 3.751mA \tag{2.1}$$

Figure 2.7: Average current calculation in mA

With this calculations we now can calculate how long will our batteries last if we know that their capacity is 2400mAh.

$$Time = \frac{2400mAh}{3,6mA} = 660hours \tag{2.2}$$

Figure 2.8: Battery duration calculation

Now we have calculated how long would our batteries last. Of course this is an aproximation and the duration of the batteries will depend on how the person working uses the system.

2.2.3 Display

The display that we will be using is MCCOG21605B6W from Midas. It is a very simple display and we used it because the information we need to share with the user is very small. In fact, the only thing that we will be telling the user via display is whether the authentication has been a success or not and if the locking has been opened or not.

Here you can see a picture of our simple display attached to the board:

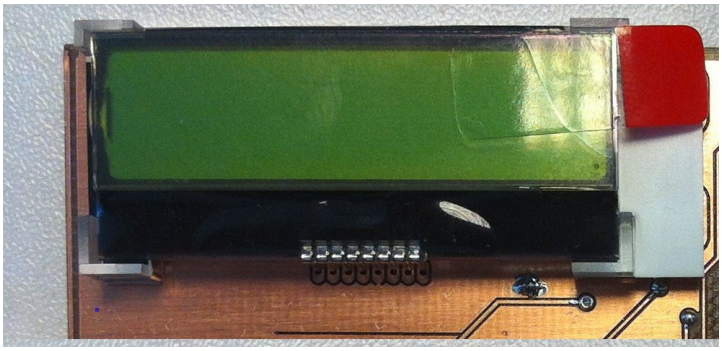

Figure 2.9: Display connections

This display was choosen because it communicates easily using the i2c bus technology which is the one we use in our system. As you will see in the display module, we programmed a specific library to write in the display easily. Also you will see now the connection diagram for the display as well as the dimensions. The display pins increase from left to right. By this we mean that the first on the left is number one and the last one on the left is eight.

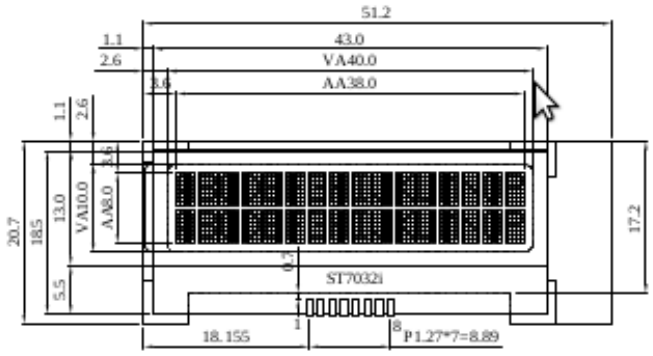

Figure 2.10: Display connections and dimensions

In the following table you will see a basic description of the connections. Bascially we connect to voltage inputs and a ground. Also we connect the SCL and the SDA inputs to the bus system. The reset is connected to the ground also, in the case you want to reset the display you have to trigger up the system to 5V. The CAP1N and the CAP1P pins remain unused.

Display pin	Description.
1	Vout pin, voltage input connection.
2	Cap1N pin.
3	Cap1P pin.
4	Vdd connection, voltage input connection.
5	Vss, ground connection.
6	SDA connection.
7	SCL connection.
8	RST, reset connection.

Figure 2.11: Display connection table

2.3 Clamp hardware

Inside each one of the pair of clamps, we have a small board that contains the hardware necessary to control the stepper and to identify the clamp pair as a single one. This is basically done using the following hardware, whoose operation will be explained lately:

- One BR24L08-W E2PROM Memory
- One Db9 connector
- One six output Pin
- One Stepper Driver
- One Stepper Motor

In the following image we can see how the board inside the clamp is:

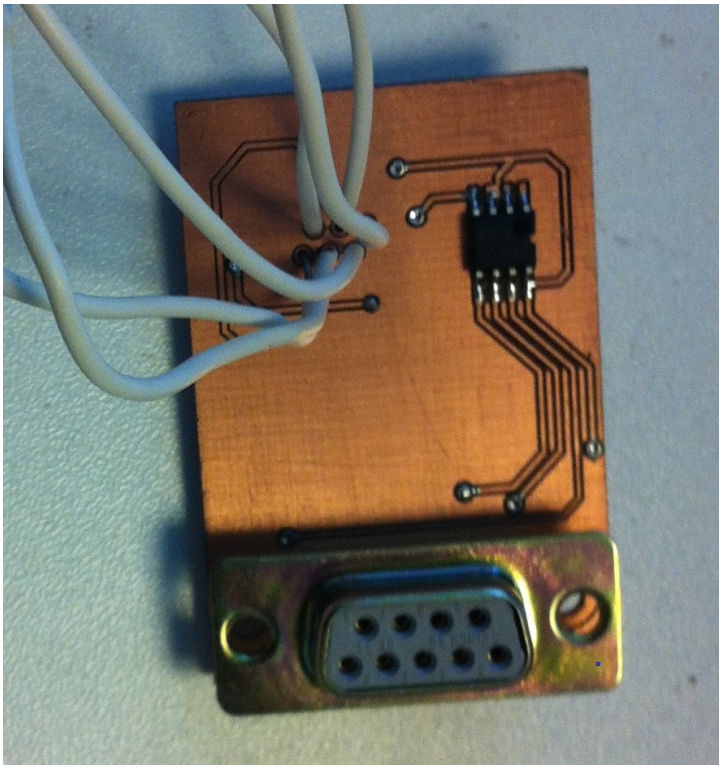

Figure 2.12: Clamp board

In the board we can clearly see the three important elements. The first one is the E2PROM memory that is used to authenticate the system, secondly the DB-9 input connector that powers up the board and also controls the stepper, and third but not least important, the 6 outputs that control the stepper.

In the following image and it's self-explaning table, we will see the pinout connection of the DB-9 connector. Regarding the 6 pin-out output for the driver stepper, we will explain them in the driver and stepper section.

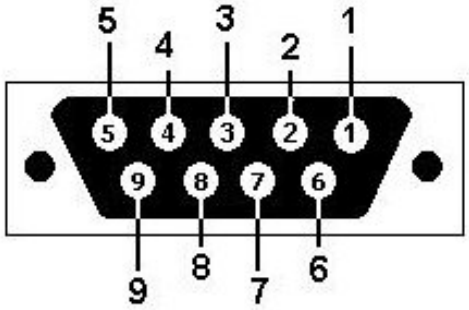

Figure 2.13: Db-9 connection

In this table we can see the connections of the Db-9. In some cases there are two connections, the boards are symetrical, meaning that both do the same, but the ports that control the stepper are different in each board thats why there are two connections in ports 5 to 2.

Display pin	Name	Description.
1	None	Not connected.
2	PB4 and PD7	Fourth and last port of the stepper controller.
3	PB2 and PD6	Third port of the stepper controller.
4	PB1 and PD5	Second port of the stepper controller.
5	PB0 and PB4	First port of the stepper controller.
6	SCL	i2c serial clock connection.
7	SDA	i2c serial data connection.
8	VCC	5 Volt input.
9	GND	Ground connection.

Figure 2.14: Db-9 connection table

2.3.1 E2prom

The memory we will be using in our clamp is BR24L08-W from Rohm. It has 8Kbit of memory that can be used for storage, but this is very far from what we need for our authentication. In our case we will only be using 1 register for storing a byte, that will be the same in the two E2proms of the clamp and that will later on be used for authenticating the system.

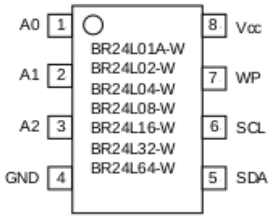

Figure 2.15: BR24L08-W e2prom memory

It is important to mention that we are using this E2prom not only because it's specifications fit perfectly with our project needs but also because it is specifically designed for working with i2c system, which is the core of our project design.

In the table below you will find a brief explanation of the connections in the memory. It is important to mention that this memories have already set all the bits for adressing them but one, which is A2. So to distinguish both memories we connected this pin to GND and Vcc respectively in each memory, so that we can address them properly. A0 and A1 remain unconnected in this memories. Also the write protect terminal is used for enabling or disabling the write in the memory, when is set to 0 and 1 respectively. In our design we set it to 0 as we believe that is inteligent to leave the write available because if we set it to 1, once the board is welded and in the case of having any problem which required rewritting the memory, we would have to buy a whole board.

Display pin	Name	Description.
1	A0	First bit for i2c memory addressing
2	A1	Second bit for i2c memory addressing
3	A2	Third bit for i2c memory addressing
4	GND	Ground connection.
5	SDA	i2c serial data connection.
6	SCL	i2c serial clock connection.
7	WP	Write protect terminal.
8	VCC	5 Volt input.

Figure 2.16: E2prom connection table

2.3.2 Stepper and driver

Connected right into our special designed clamp board, we have the stepper and it's driver whoose main puropuse is to open and close the locking for the clamp. The stepper motor that we will be using is the 28BYJ-48 and we having it working with the ULN2003 driver. Later on we will further explain it's technical characteristics. In the following picture you can see the stepper.

Figure 2.17: 28BYJ-48 stepper motor

The stepper is commanded by our microprocessor in the main board and it is basically done by sending him small pulses of 5V (steps) to it's four inputs. The problem we had, is that altough the stepper is quite small, our microprocessor hadn't enough power to move it properly. Because of this, and after beeing unable to move any stepper just with the system we had, we decided to add a driver.

The basic operation of the driver is to increase the current, ideally this driver can transform a current of 1,5mA to 150mA reaching a maximum output current of 500mA. In our case it increases the current that the microprocessor sends that has a value of 2mA to reach the necessary input current for our stepper, aproximately 100mA.

Figure 2.18: ULN2003 stepper driver

Apart from increasing the current the driver does another very important job. In the last part of the driver there is a diode included so as to not allowing the current to drive the other direction. This is very important because if it wasn't for this the current could be driven in inverse which is something that we don't want!

If you want to read further technical information about this two components you can find them in the annex section. Furthermore, the code that is used to program the stepper will be explained in the software chapter.

2.4 Board design

At the very beginning we used a protoboard to design our project and also to check the behavior of all the components, and as soon as we advanced with the project, we kept integrating the components into a single system. Once the system worked we decided it was about time to create a board that could fullfil the necessities of our project.

More certainly, we needed three boards two of which would be practically identical. As we already explained, we would have a central device that will control the whole system and that will have the big board containing the Microprocessor and the user interface. Also, we need one small board inside each pair of the clamp, to which we will connect the big board and that we will be using for authenticating the system and moving the stepper. By moving the stepper we understand tranismitting the microprocessor orders to the stepper, as the small board has no active component able to transmitt anything.

In the following picture we can see the interface of the eagle program that we are using for our system designing.

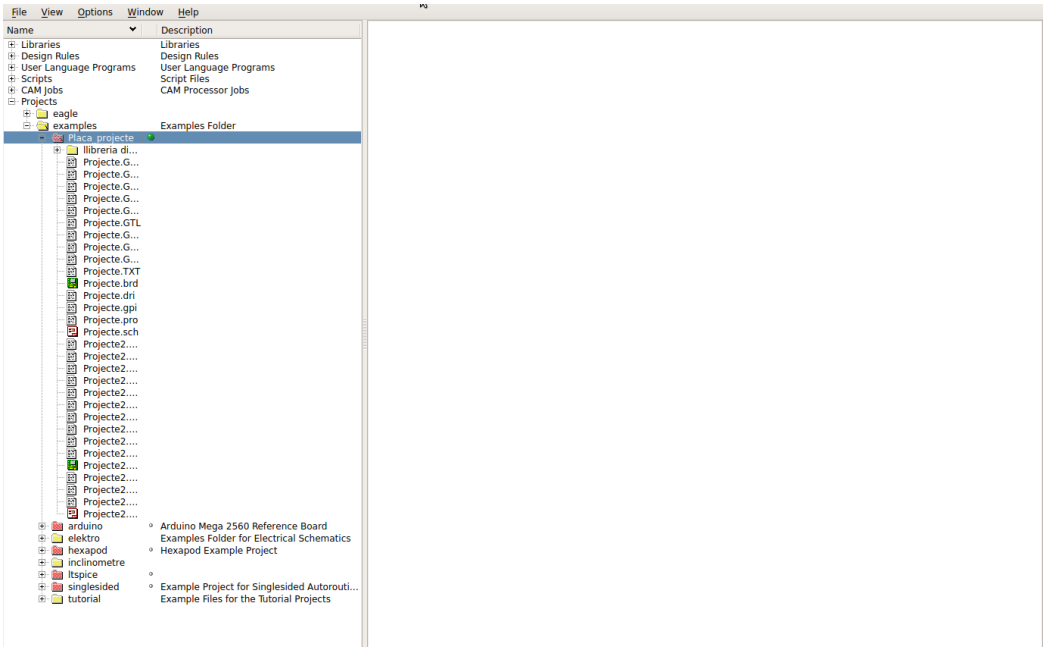

Figure 2.19: Eagle interface picture

The design of the board has been performed using Eagle, a very specific program for board designing. The user interface of the program is very easy-going and we got used to it very easily. The designing of the board has two basic parts:

- Schematic design
- Board layout design

To design the schematic the first step is choosing all the components that we need in our board and connecting them to one another using the eagle software. It is of vital importance not forgetting any component nor any interconnection as later on we will link this schematic file to the board layout and if there isn't a single component we will have lots of troubles.

When finished, our schematics from the main board and clamp board looked like this:

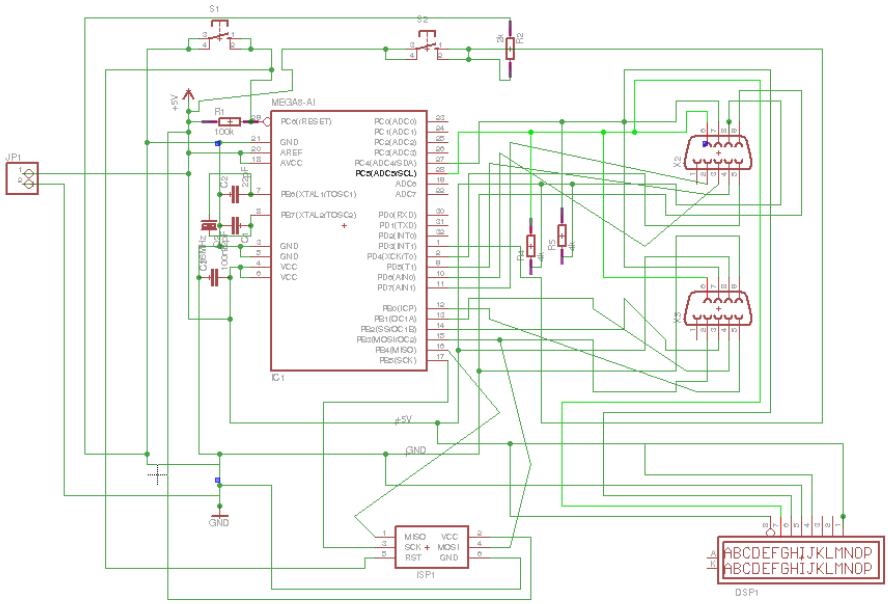

Figure 2.20: Main board schematic design

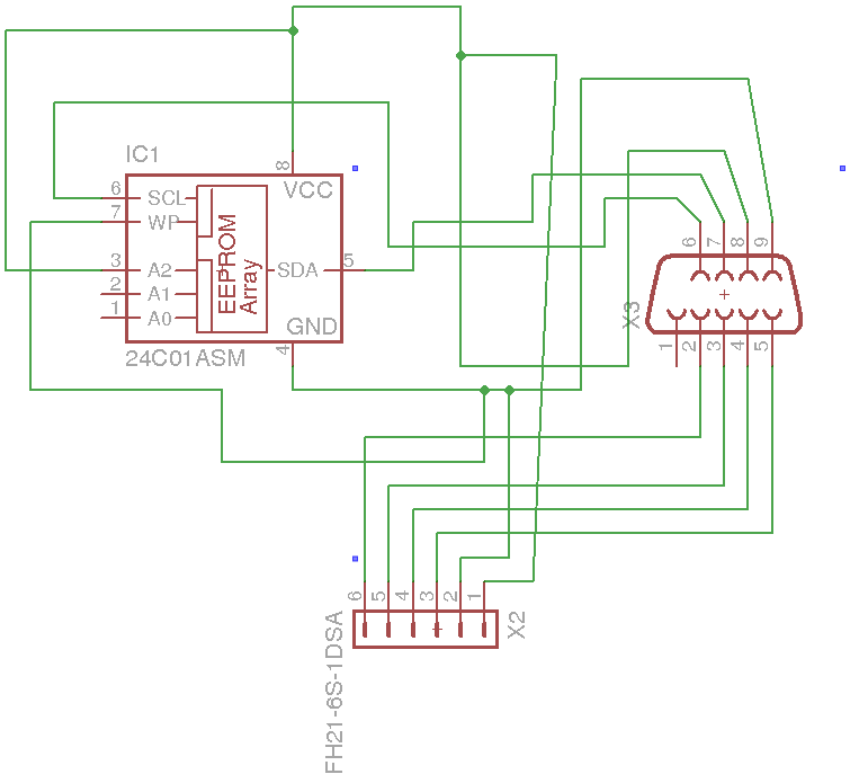

Figure 2.21: Clamp board schematic design

Once we finish our schematic it is time for the layout designing. Unlike the schematic this the closer you can be to hell if you have never designed a board layout. In our case both boards are quite simple, yet we had lots of problems until we finished them successfully.

The main problem is that not a single track of the board can ran over any other, as we would get a short-circuit. This may seem quite of a basic clue but, in fact, it isn't. The reality is that there are quite a few dozens of tracks going from one side of the board to the other and is quite a hard job achieving to optimize the space as well as getting a good looking board.

To make it easier for such a newbie like I am with board design, we will use double side designing so we can drive tracks through both sides of the board. Also we can connect one another using drills, that will shortcircuit both sides of the board, we could say it is more or less like making a tunnel.

In the following pictures you can see how our board layout final design looks like.

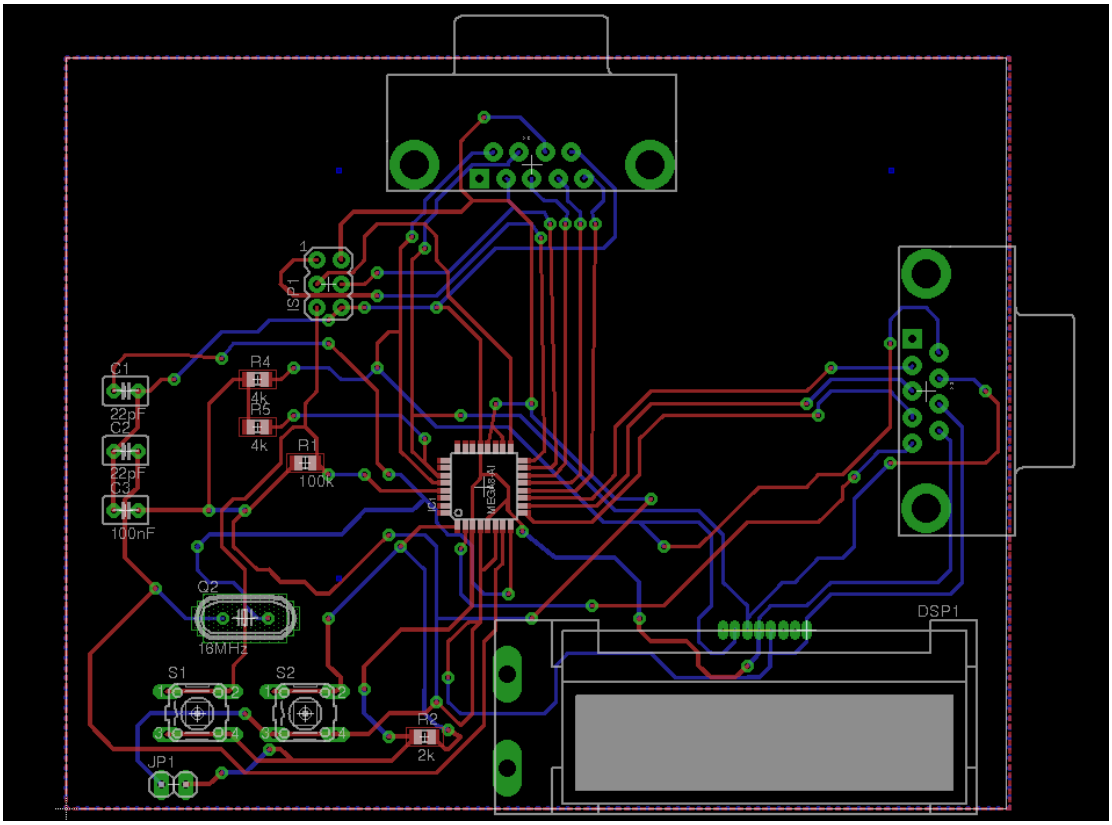

Figure 2.22: Main board layout design

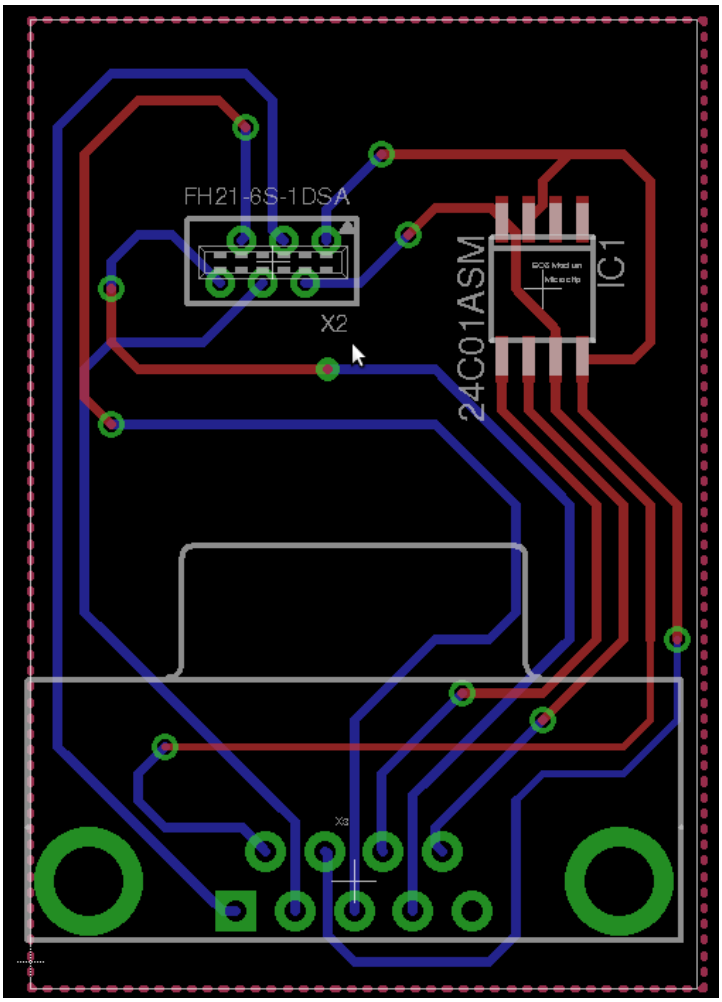

Figure 2.23: Clamp board layout design

In the layout design you can see connections from two colors, blue and red. Red colors are the one's that are in the TOP of our board while the blue one's are in the bottom. As we already explained, we use this to avoid making shortcircuits between the different connections. The same happens with the devices attached to the board, blue ones will be attached at the bottom while red ones at the TOP. The rest of the board will be connected to the ground.

The green button look like things you see are the drills that we already mentioned. This drills are what we use to communicate both sides of the board without making any disgrace. This drills are very useful to jump over a line.

3 Software description

3.1 Introduction

Having explained the hardware that is used in our project, it is now about time to explain the most important matter, the software that makes this hardware work properly. The software is certainly the most important part in our project as it is by far to what we dedicated more time developing.

Our project has been programmed using C and it is jerarquically structured by modules and based on the i2c system as a communication tool. This is the basis of the system, later on we will explain how this structure works in the Module section.

In general, we haven't written a big amount of code but a very specific one. The idea was to be very concrete and specific rather than writting huge modules full of unusefull lines. During this introduction we will explain the basis of our project software.

By this we mean explaning both the code and the communication tool we will be using. First of all we will explain the specifications and usage of the i2c bus protocol and how we will use it in our system. Later on we will make an introduction of how ous software system works and how the modules interact with one another. In the end to finish the chapter we will be explaining briefly what each module is responsible of, as later on we will include the code in the annex and it is not necessary to do a huge explanation.

3.2 I2C Bus

We will explain now the basis of our project intercomunication, the i2c bus. The i2c is basically a multimaster serial single-ended bus invented by the Philips semiconductor division, today NXP Semiconductors, and used for attaching low-speed peripherals to a motherboard, embedded system, cellphone, or other digital electronic devices. Several competitors, such as Siemens AG (later Infineon Technologies AG, now Intel mobile communications), NEC, Texas Instruments, STMicroelectronics (formerly SGS-Thomson), Motorola (later Freescale), and Intersil, have introduced compatible I2C products to the market since the mid-1990s.

The i2c is a very usefull bus to communicate a master with several slaves as it's protocol it's simple to implement and to understand. In our case we use the i2c to communicate with the Display and the E2prom memories. We use the 300Khz mode so we could still be using a faster mode in the case of needing it.

The basis of this system consists in two lines, one is the serial clock (SCL) and the other one is the serial data (SDA). Every slave or master has to be connected to both lines to have communication with the system. The beginning of our system will be the Arduino to which we connect the SDA and SCL to the analogic pins A4 and A5 respectively.

In the following picture you can see where physically are the SDA and SCL connections for the Atmega328p.

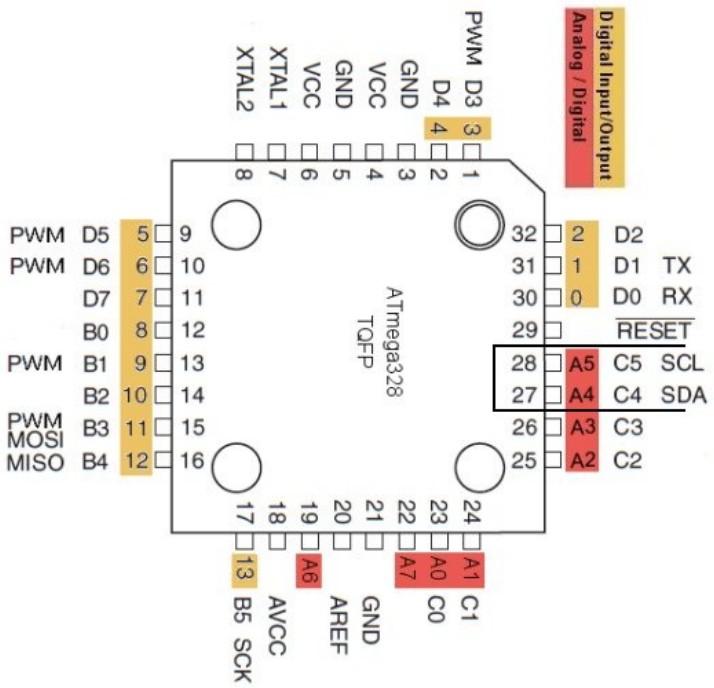

Figure 3.1: i2c pins location

It is of crucial importance to connect both lines using a pull-up to 5V. This is very important when using i2c as the system wouldn't work without the pull-up's, the value of the resistors is 4'8KOhms each.

In the following picture you will see a diagram of how the pullup should be connected, as well as some aleatory devices.

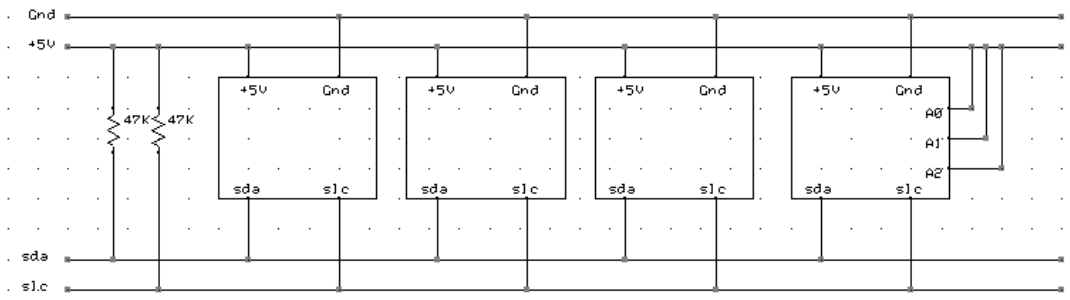

Figure 3.2: i2c connection diagram with pull-up

After all this previous matter regarding the setup, we program our Processor to initialize the system and communicate to the slaves that we want. When this is done it is time to connect the slaves to the line. It must be said that it is crucial when defining the code that we compile in our Processor, that the slave adresses are defined properly, both physically and in our software (They must be the same!).

Usually most of the systems have 6-7 bit adress that are fixed and there are 1-2 bits that can be choosen by hardware externally. This allows us to define the address of our slaves in the case of connecting two slaves of the same kind.

3.3 Modules

As we already explained, our system is structured in modules because it is the best way the define a clear structure. To make an approach to this structure we will begin listing all the modules that are included in our system and we will later on explain the communication and dependencies between them using a jerarchy and dependency diagram.

In later chapters we will include all the code in our project so that we can take even a closer look to our system. Also we will explain what each module is responsible of. This is the list of the modules included in our system:

- twi.c
- display.c
- E2PROM.c
- motor.c
- menu.c
- main.c
- sleep.c
- polsadors.c

Also we have a Makefile which is the responsible of structuring the dependencies between modules and compiling it into a single file. Our system is very simple and easy to understand, as we have very few lines of code. In the next page we will illustrate you with a module diagram so that you can better understand which modules is used for who and what are the dependencies between them.

In the following diagram, we want to show how our modules interact with one another. The arrow indicates how the information flows in our system. By this we mean that the main.c is the module using information from the four below, and the menu is using the other two and so on.

Also it must be said that all the modules include some packages that are not self explained in this diagram but that are included in the annex so that you can check them.

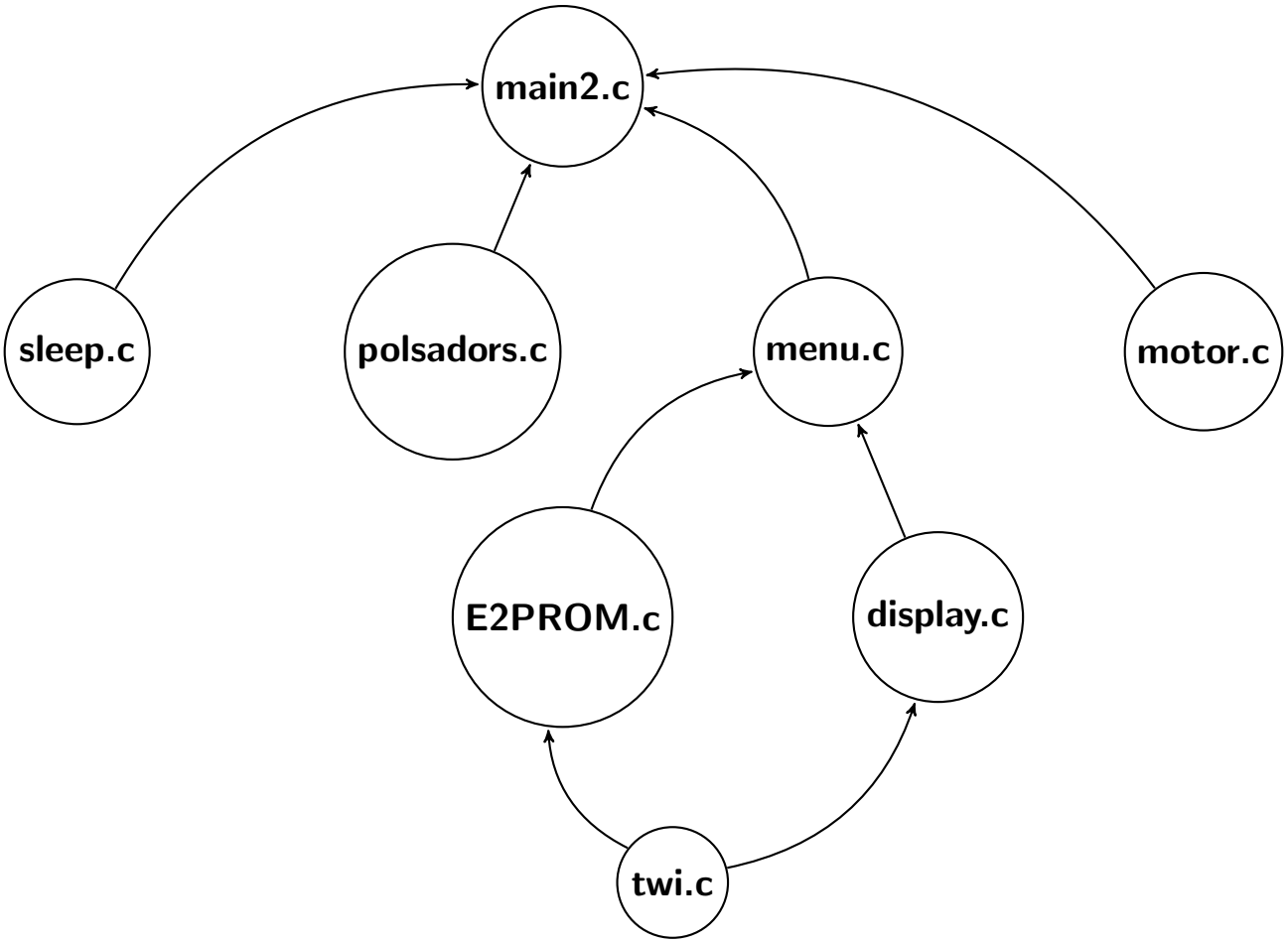

Figure 3.3: Software modules digaram

3.3.1 twi.c

The twi module is the one responsible for the most basic communication with the i2c bus. This module is responsible for first of all configurating the speed, the address, the master and the salves. Also it physically sends the instructions for reading writting the registers in a certain slave within the i2c connection.

The i2c module doesn't need to use any other module to work but it is used for the display. and the E2PROM.c modules to help them work.

3.3.2 display.c

The display.c module is the one responsible for interacting with the display. It uses the twi module to send orders to the display slave so that we can write anything. The module is programmed so that we can send what we want to write and the module itself uses twi.c to write it.

3.3.3 E2PROM.c

The E2PROM is a ver specific module that is used to adress two different kinds of E2PROM memories. This module is used to access the different registers in the memory and writting or reading them. With this module we can write to any memory of the same family of the one designed for our project.

This module uses the twi.c module to physically communicate with the memory and send to it orders such as what he wants to read etcetera. The way we programmed the module we can access any register we want and reading it or writting to it.

3.3.4 motor.c

This module is the one that interacts with the stepper motor and does not need any other module to work, it just interacts with the main.c to receive orders of when to move the stepper. It main task is to move the lock 45 degrees in one direction or another depending if you want to open or to close the lock.

This module interacts directily with the output pins of the processor to set them high or low, so interacts with the most physical layer of the system. It basic operation is fullfilling a complete loop to make the stepper move one step, and do a certain number of steps to move 45 degrees.

3.3.5 polsadors.c

The polsadors.c module is the module in charge of controlling the push-up buttons in our system. Basically it controls the button that wakes up the processor when it is asleep. This is done by using an atention interrupt routine, that wakes up the atmega once the button is pushed.

This module works by its own self and it is only used by the main.

3.3.6 sleep.c

The sleep.c module is a very simple module. This module the only thing that does is awaking and sending the arduino to sleep. This is done using the ¡avr/sleep.h¿ library that is specifically designed for this. This module works by its own and does not need any other module to work, furthermore it is only used by the main.

3.3.7 menu.c

The menu.c module is the second most important module in our sytem. This module uses some of the most basic modules to send information to the main.c. His objective is reducing the amount of code inside the main and making easier to understand.

This module uses the E2PROM.c, and the display.c module, and inderectly uses also the twi.c module. The idea of this module is to aglutinate in further implementations also the sleep.c, motor.c and the polsadors.c modules. This way the main will be only using the menu.c module and will be this one mentioned the one that is used by the main.c.

3.3.8 main.c

This is our princpial module, as it name indicates. It's objective is to use, directly or indirectly, all the modules in our system. Our main is based in a routine that attends the interruption when the button to wake up the system is pushed.

After this, the system does the authentication of the two E2PROM's in the clamp and if the authentication is alright the system opens the locking of the sytem, allowing the user to open the clamp manually. After this it send the processor back to sleep. If the authentication is a failure the system goes back to sleep again.

It must be said that the system while processing all this actions, it mantains the user informed of the actuation using the display, sending the system status at all time.

3.3.9 Makefile.c

The makefile is not a module at all, but we thought we had to include it as it is a very important piece of our design. The makefile is used for compiling the main and sending it to the processor successfully. If you want to further see what is it based on you can take a look at it on the annex chapter.

4 Conclusions

The conclusion of our project is that we successfully designed a device to improve system that had worked mechanically for so long. This was our objective from the very begining as all the mechanical systems have huge limitations. Our design, apart from beeing very compact, is suitable to lots of changes in the case of beeing necessary.

In this first design we used a single byte for the system authentication, this is something we can easily modify reserving more bits for the authentication. Also the code we programmed to our processor can easily be used for much more bigger stepers. The only need in the case of needing bigger stepers to move a bigger lock, would be using a driver or a power bridge able to provide the stepper with the necessary current.

In general de project has been a success as we achieved the objectives stated in the beginning. During the project design we achieved the first objective of making the prototype work and once we knew that our project was factible we begun the design of the board with the final device.

This last steps of designing the final device were much more complicated than the prototype design, as we weren't used to the usage of Eagle and to designing board layouts. Anyway, once we knew that our project was factible and worked properly we can say that our project has been a success.

4.1 Improvements

As we said from the very beginning, this project has been designed for a company, Sofamel, SL an so on the objectives were stated by them and the tutor of the project. The first objective was to think of the suitable design that could substitute the existing mechanical system for a more modern one.

All together, we thought that we could use some kind of digital key instead of the physical key that was used until then. The idea we had was to design an external system that when plugged into the clamp would work as a key. The difference between the mechanical system and the one that was about to be designed was that the digital one would work for any clamp, while the key worked only for a certain one.

This idea needed the design of an external device able to be connected to the clamp, and afterwards interact with some kind of internal device, that would authenticate the clamp as an unique pair. After this we would open the clamp locking using the external device.

Having fulfilled the company Sofamel, SL objectives, it is clear that we designed a device that could be used in real live. The fact is that this device is suitable to huge changes, as we knew it wasn't the final device but just the beginning. The idea is that this project can be further developed in the case of the Company beeing realy interested in developing it for real life usage.

5 Bibliography

E2prom = Brl24L, Pdf = "http://rohmfs.rohm.com/en/products/databook/datasheet/ic/memory/eeprom/br24l-e.pdf"

Stepper = 28BYJ-48, Pdf = "http://robocraft.ru/files/datasheet/28BYJ-48.pdf",

Driver = Uln2003, Pdf = "http://www.ti.com/lit/ds/symlink/uln2003a.pdf",

Processor = Atmega328p, Pdf = "http://www.atmel.com/Images/doc8161.pdf",

Display = MCCOG21605B6W, Pdf = "http://www.farnell.com/datasheets/1663636.pdf",

i2c = i2c user manual, Pdf = "http://www.nxp.com/documents/user_manual/UM10204.pdf",

Eagle = Eagle manual, Pdf = "http://www.cadsoft.de/wp-content/uploads/2011/05/V6_manual_en.pdf",

6 Annexes

6.1 Code

6.1.1 twi.c

```
1
2 #include <avr/io.h>
3 #include <util/twi.h>
4 #include <avr/interrupt.h>
5 #include <stdbool.h>
6 #include <inttypes.h>
7 #include "twi.h"
8 #include "display.h"
9
10
11 /* Envia */
12 static void start(void){
13   TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
14 }
15
16 static void data(uint8_t d){
17   TWDR = d;
18   TWCR = (1<<TWINT)|(1<<TWEN);
19 }
20
21 static void stop(void){
22   TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);
23 }
24
25 /* Espera */
26 static void wait_twint(void){
27   while (!(TWCR&(1<<TWINT)));
28 }
29
30 static void wait_stop(void){
31   while((TWCR&(1<<TWSTO)));
32 }
33
34 /* Verifica */
35 static bool check_start(void){
36   return ((TWSR&0xF8) != TW_START);
37 }
38
39 static bool check_rep_start(void){
40   return ((TWSR&0xF8) != TW_REP_START);
41 }
42
43 static bool check_mt_address(void){
44   return ((TWSR&0xF8) != TW_MT_SLA_ACK);
45 }
46
47 static bool check_mt_data(void){
48   return ((TWSR&0xF8) != TW_MT_DATA_ACK);
49 }
50
51 static bool check_mr_address(void){
```

```
52    return ((TWSR&0xF8) != TW_MR_SLA_ACK);
53  }
54
55  /*-------------------------------------------------*/
56
57
58  bool catch_channel(void){
59    start();
60    wait_twint();
61    return check_start();
62  }
63
64  bool rep_catch_channel(void){
65    start();
66    wait_twint();
67    return check_rep_start();
68  }
69
70  bool send_address_mt(uint8_t a){
71    data(a<<1);
72    wait_twint();
73    return check_mt_address();
74  }
75
76  bool send_address_mr(uint8_t a){
77    data((a<<1)|(0b00000001));
78    wait_twint();
79    return check_mr_address();
80  }
81
82  bool send_data(uint8_t d){
83    data(d);
84    wait_twint();
85    return check_mt_data();
86  }
87
88  int8_t receive_data(void){
89    //uint8_t d = TWDR;
90    TWCR = (1<<TWINT)|(1<<TWEN)|(1<<TWEA);
91    wait_twint();
92    int8_t d = TWDR;
93    return d;
94  }
95
96  int8_t receive_data_last(void){
97
98    //uint8_t d = TWDR;
99    TWCR = (1<<TWINT)|(1<<TWEN);
100   wait_twint();
101   int8_t d = TWDR;
102
103   return d;
104
105 }
106
107
108 void release_channel(void){
109   stop();
110   wait_stop();
111
112 }
113
114 /*-------------------------------------------------*/
115
116 /* Innit */
117 void setup_tw(void){
```

```
118   DDRC = 0xFF;          //Ports de sortida
119   TWSR &= 0b11111100; //Prescaler 1
120   TWBR = 72;            //Freq. 100kHz
121    sei();
122 }
123
124
125
126 uint8_t regRead(const uint8_t slave_addr, const uint8_t reg){
127  //register read
128    uint8_t data;
129    catch_channel();
130    send_address_mt(slave_addr);
131    send_data(reg);
132    display_write("OK1");
133    rep_catch_channel();
134    display_write("OK2");
135    send_address_mr(slave_addr);
136    display_write("OK3");
137    data = receive_data_last();
138    display_write("OK4");
139    release_channel();
140    return data;
141 }
142
143
144 void regWrite(const uint8_t slave_addr, uint8_t reg, uint8_t data){
145 //register write
146    catch_channel();
147    send_address_mt(slave_addr);
148    send_data(reg);
149    send_data(data);
150    release_channel();
151 }
152 void regWriteS(const uint8_t slave_addr, uint8_t data){
153    catch_channel();
154    send_address_mt(slave_addr);
155    send_data(data);
156    release_channel();
157 }
```

6.1.2 polsadors.c

```
1  #include <inttypes.h>
2  #include <util/delay.h>
3  #include <avr/io.h>
4  #include <stdbool.h>
5  #include "polsadors.h"
6
7  void init_pols(void){
8    /* External interrupts enable, registers to modificate:
9       - PORTD - Per activar pull-ups interns
10      - EICRA [ ISC11 | ISC10 ] - Per seleccionar el mode
11      - EIMSK [ INT1 | INT0 ] - Per activar / desactivar les interrupcions
12      - EIFR [ INTF1 | INTF0 ] - Per comprovar si hi ha interrupcions
13
14   */
15
16   DDRD &= ~(1 << DDD3);
17   PORTD |= (1 << PORTD2);        // Activem pull-up
18   EICRA |= (1 << ISC10) | (1 << ISC11);    // S'activa interrupcio per qualsevol canvi
19   EIMSK |= (1 << INT1);
20   //EIMSK |= (1 << INT0);       // Interrupcions habilitades
21 }
22
23
24 void desactiva_int(void){
25   /* Interrupt disable
26    */
27   EIMSK |= (1 << INT1);
28   //EIMSK |= (1 << INT0);
29 }
30
31 bool valor_alt(void){
32   /*Rise up interruption checking
33    */
34   return ((PIND & (1 << PD3)) != 0);
35 }
36
37 void desactiva_flag(void){
38   /* flag disable
39    */
40   EIFR |= (1 << INTF1);
41 }
```

6.1.3 motor.c

```
1
2 #include "motor.h"
3 //#include <stdlib.h>
4 #include <avr/interrupt.h>
5 #include <avr/sfr_defs.h>
6 #include "queue.h"
7 #include <util/delay.h>
8 #include <avr/io.h>          //This is our usual include
9 #define F_CPU 16000000UL     //F=16Mhz
10 #include <util/delay.h>         //The delay functions/routines
11
12 void set_up2(void){
13
14
15 DDRD |= (1<<PD7);
16 DDRD |= (1<<PD6);
17 DDRD |= (1<<PD5);
18 DDRD |= (1<<PD4);
19 DDRB |= (1<<PB3);
20 DDRB |= (1<<PB2);
21 DDRB |= (1<<PB1);
22 DDRB |= (1<<PB0);
23
24 }
25
26
27 void degree(void){
28   //45 degrees rotation
29   int t=0;
30   while(t<130){
31     gir1();
32     gir2();
33     t+=1;
34 }
35
36 }
37
38
39 void gir1(void){
40   //First stepper move
41   int x=1;
42   //1
43
44 PORTD = 128;
45 _delay_ms(x);
46 PORTD = 192;
47 _delay_ms(x);
48 PORTD= 64;
49 _delay_ms(x);
50 PORTD= 96;
51 _delay_ms(x);
52 PORTD=32;
53 _delay_ms(x);
54 PORTD=48;
55 _delay_ms(x);
56 PORTD=16;
57 _delay_ms(x);
58 PORTD=144;
59 _delay_ms(x);
60
61 }
62
63 void gir2(void){
64
```

```
65    //Second stepper move
66    int x=1;
67    //1
68  PORTB = 8;
69   _delay_ms(x);
70  PORTB = 12;
71   _delay_ms(x);
72  PORTB= 4;
73   _delay_ms(x);
74  PORTB= 6;
75   _delay_ms(x);
76  PORTB=2;
77   _delay_ms(x);
78  PORTB=3;
79   _delay_ms(x);
80  PORTB=1;
81   _delay_ms(x);
82  PORTB=9;
83   _delay_ms(x);
84
85  }
```

6.1.4 menu.c

```
#include <avr/io.h>
#include <util/delay.h>
#include <util/twi.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <inttypes.h>
#include "serial.h"
#include "display.h"
#include "E2PROM.h"
//#include <avr/sleep.h>

char x;
char y;




/*
void adorm_ard(void) {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  sleep_cpu();
}
*/

void init_princ(void){
    //init
    serial_init();
    setup_tw();
    e2prom_init();
    display_init(2,0);
    display_on();
    display_cursor_on();
    display_clear();

}

void write_eprom(void){

      e2prom_write(0xEA, 0x02);//E2prom write
      e2prom_write2(0xEA, 0x02);

}

void comparacio(void){
    //Authentication comparsion
    x = e2prom_read(0xEA);
    y = e2prom_read2(0xEA);
    //y= 0x02;

if (x == y) {
     display_clear();
     display_write("Authentication");
     display_setcursor(0,1);
     display_write("OK");
     _delay_ms(2000);
}
 else {
    display_clear();
    display_write("Authentication");
    display_setcursor(0,1);
    display_write("failure");
    _delay_ms(2000);
}
```

```
65 }
66
67
68
69 void benvinguda(void){
70
71     //Welcome
72     init_princ();
73     display_write("Welcome");
74     _delay_ms(2000);
75     display_clear();
76     display_write("Initializing the");
77     display_setcursor(0,1);
78     display_write("system");
79     _delay_ms(2000);
80 }
81
82
83
84 void authentication(void){
85       //System authentication
86       display_clear();
87       display_write("System");
88       display_setcursor(0,1);
89       display_write("authentication");
90       _delay_ms(2000);
91       comparacio();
92
93 }
94
95
96 void adeu (void){
97 //Goodbye
98 display_clear();
99 display_write("Autentificacio");
100 display_setcursor(0,1);
101 display_write("acabada");
102 _delay_ms(2000);
103 display_clear();
104 display_setcursor(0,0);
105 display_write("Desbloquejant");
106 _delay_ms(2000);
107 display_clear();
108 display_write("Enclavament");
109 display_setcursor(0,1);
110 display_write("obert");
111 _delay_ms(2000);
112 display_clear();
113 display_write("Fins aviat");
114 _delay_ms(2000);
115 display_clear();
116 }
```

6.1.5 E2PROM.c

```
1
2 #include <stdint.h>
3 #include "E2PROM.h"
4 #include "twi.h"
5 #include "display.h"
6
7 void e2prom_init(void){
8   // Init
9   setup_tw();
10 }
11
12 static void e2prom_addressing(uint16_t address){
13   /* First eeprom direction */
14
15   catch_channel();
16   send_data(E2PROM_SLA_W);
17   send_data((int)(address>>8));     //MSB
18   send_data((int)(address&0xFF));    //LSB
19 }
20
21 static void e2prom_addressing2(uint16_t address){
22   /* Second Eeprom direction*/
23   catch_channel();
24   send_data(E2PROM_SLA_W2);
25   send_data((int)(address&0xFF));    //LSB
26 }
27
28
29 void e2prom_write(uint16_t address, char data){
30   /*First eeprom write*/
31   e2prom_addressing(address);
32   send_data(data);
33   release_channel();
34 }
35
36 void e2prom_write2(uint16_t address, char data){
37   /*Second eeprom write*/
38   e2prom_addressing2(address);
39   send_data(data);
40   release_channel();
41 }
42
43
44 char e2prom_read(uint16_t address){
45   /*First eeprom read*/
46   e2prom_addressing(address);
47   release_channel();
48   rep_catch_channel();
49   send_data(E2PROM_SLA_R);
50   char data=receive_data();
51   release_channel();
52   return data;
53 }
54
55 char e2prom_read2(uint16_t address){
56    /*Second eeprom read*/
57   e2prom_addressing2(address);
58   release_channel();
59   rep_catch_channel();
60   send_data(E2PROM_SLA_R2);
61   char data=receive_data();
62   release_channel();
63   return data;
64 }
```

6.1.6 display.c

```
1  #include <stdbool.h>
2  #include <inttypes.h>
3  #include <util/delay.h>
4  #include "display.h"
5  #include "twi.h"
6
7  uint8_t addr,cols;
8  uint8_t display_basic;
9  uint8_t display_extended;
10 uint8_t displaycontrol;
11 uint8_t displaymode;
12
13 static bool send_address_display_t(void);
14 //static bool send_address_display_r(void);
15 static void instruction(uint8_t value);
16 static void write(uint8_t value);
17
18
19
20 static bool send_address_display_t(void){
21   //Funcio de base
22   /* Catch channel and send display address
23      in write mode.*/
24
25   catch_channel();
26   return send_address_mt(62);
27 }
28
29
30 static void instruction(uint8_t value){
31
32   send_address_display_t();
33   _delay_us(5);
34   send_data(0x00);  //RS clear
35   _delay_us(5);
36   send_data(value);
37   _delay_us(5);
38   release_channel();
39   _delay_us(30);
40 }
41
42 static void write(uint8_t value){
43   /* Escrivim el valor a la pantalla*/
44   send_address_display_t();
45   _delay_us(5);
46   send_data(0x40); //Rs Set
47   _delay_us(5);
48   send_data(value);
49   _delay_us(5);
50   release_channel();
51   _delay_us(30);
52 }
53 //*****
54 // Usage functions
55 //*****
56 void display_init(uint8_t rows, uint8_t height){
57
58   addr = 62;
59   cols = 8;
60
61   display_basic = LCD_INSTRUCTION_SET_BASIC | LCD_8BITMODE | LCD_1LINE | LCD_5x8DOTS ;
62   display_extended = LCD_INSTRUCTION_SET_EXTENDED | LCD_8BITMODE | LCD_1LINE |
         LCD_5x8DOTS;
63   displaycontrol = LCD_DISPLAYON;
```

```
64    uint8_t contrast = 0x0A;
65
66    if (rows > 1){
67      display_basic |= LCD_2LINE;
68      display_extended |= LCD_2LINE;
69    }
70    if ((height == 1) & (rows == 2)){
71      display_basic |= LCD_5x16DOTS;
72      display_extended |= LCD_5x16DOTS;
73    }
74
75    _delay_ms(40);
76
77    instruction(LCD_FUNCTIONSET | display_basic);
78    _delay_us(30);
79    instruction(LCD_FUNCTIONSET | display_extended);
80    _delay_us(30);
81
82    instruction(LCD_BIAS_OSC_CONTROL | LCD_BIAS1_5 | LCD_OSC_192);
83    _delay_us(30);
84
85
86  // contrast low nible
87    instruction(LCD_CONTRAST_LOW_BYTE | (contrast & LCD_CONTRAST_LOW_BYTE_MASK));
88    _delay_us(30);
89
90  // contrast high nible / icon / power
91    instruction(LCD_ICON_CONTRAST_HIGH_BYTE | LCD_ICON_ON | LCD_BOOSTER_ON | (contrast >>
          4 & LCD_CONTRAST_HIGH_BYTE_MASK));
92    _delay_us(30);
93
94  // follower control
95    instruction(LCD_FOLLOWER_CONTROL | LCD_FOLLOWER_ON | LCD_Rab_2_00);
96    _delay_ms(200);
97
98  // function set basic
99    instruction(LCD_FUNCTIONSET | display_basic);
100   _delay_us(30);
101
102 // display on , unicament posem en marxa el display , el cursos i el blink estan en off
103   instruction(LCD_DISPLAYCONTROL |  LCD_DISPLAYON |  LCD_CURSOROFF | LCD_BLINKOFF );
104   _delay_us(30);
105
106 // entry mode set, increment d'esquerra a dreta i sense autoscroll
107   displaymode=LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
108   instruction(LCD_ENTRYMODESET | displaymode);
109   _delay_us(30);
110
111 // clear display nateja el display i apunta a la posicio 0
112   instruction(LCD_CLEARDISPLAY);
113   _delay_ms(2);
114
115 // return home apunta a la posicio 0
116   instruction(LCD_RETURNHOME);
117   _delay_ms(2);
118
119
120 }
121
122
123 void display_write(char *data){
124   /*Printem cadenes llargues de caracters*/
125   for (int i =0; data[i]!='\0'; i++){
126     write((uint8_t)data[i]);
127   }
128 }
```

```
129
130
131
132 void display_setcursor(uint8_t col, uint8_t row){
133
134   if (row > 1){
135     row = 1;}
136
137   if (row == 1)
138     instruction(LCD_SETDDRAMADDR | (col + 0x40));
139   else
140     instruction(LCD_SETDDRAMADDR | (col));
141
142   _delay_us(30);
143 }
144
145 void display_clear(void){
146   /*It clears the screen*/
147   instruction(LCD_CLEARDISPLAY);
148   _delay_ms(2);
149 }
150
151 void display_home(void){
152   instruction(LCD_RETURNHOME);
153   _delay_ms(2);
154 }
155
156 void display_on(){
157   /* encen el display*/
158   displaycontrol |= LCD_DISPLAYON;
159   instruction(LCD_DISPLAYCONTROL | displaycontrol);
160   _delay_us(30);
161 }
162
163 void display_off(){
164   /* atura el display*/
165   displaycontrol &= ~LCD_DISPLAYON;
166   instruction(LCD_DISPLAYCONTROL | displaycontrol);
167   _delay_us(30);
168 }
169
170 void display_cursor_on(void){
171   displaycontrol |= LCD_CURSORON;
172   instruction(LCD_DISPLAYCONTROL | displaycontrol);
173   _delay_us(30);
174 }
175
176 void display_cursor_off(void){
177   displaycontrol &= ~LCD_CURSORON;
178   instruction(LCD_DISPLAYCONTROL | displaycontrol);
179   _delay_us(30);
180 }
181
182 void display_blink_on(void){
183   /* Activem el blink del cursor*/
184   displaycontrol |= LCD_BLINKON;
185   instruction(LCD_DISPLAYCONTROL | displaycontrol);
186   _delay_us(30);
187 }
188
189 void display_blink_off(void){
190   displaycontrol &= ~LCD_BLINKON;
191   instruction(LCD_DISPLAYCONTROL | displaycontrol);
192   _delay_us(30);
193 }
194
```

```
195 void display_left(void){
196   /* shifta el display cap a l'esquerra*/
197   instruction(LCD_FUNCTIONSET | display_basic);
198   _delay_us(30);
199
200   instruction(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
201   _delay_us(30);
202 }
203 void display_right(void){
204   /* shifta el display cap a la dreta*/
205   instruction(LCD_FUNCTIONSET | display_basic);
206   _delay_us(30);
207
208   instruction(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
209   _delay_us(30);
210 }
211
212 void display_left_to_right(void){
213   /* canvia la direccio d'increment del cursor (escrivim d'esquerra a dreta)*/
214   displaymode |= LCD_ENTRYLEFT;
215   instruction(LCD_ENTRYMODESET | displaymode);
216   _delay_us(30);
217 }
218
219 void display_right_to_left(void){
220   /* canvia la direccio d'increment del cursor (escrivim de dreta a esquerra)*/
221   displaymode &= ~LCD_ENTRYLEFT;
222   instruction(LCD_ENTRYMODESET | displaymode);
223   _delay_us(30);
224 }
225
226 void display_autoscroll(void) {
227   /* Quan escrius desplasa tots els registres
228      en direccio contraria a la d'escriptura
229   */
230   displaymode |= LCD_ENTRYSHIFTINCREMENT;
231   instruction(LCD_ENTRYMODESET | displaymode);
232   _delay_us(30);
233 }
234
235
236 void display_autoscroll_off(void) {
237   displaymode &= ~LCD_ENTRYSHIFTINCREMENT;
238   instruction(LCD_ENTRYMODESET | displaymode);
239   _delay_us(30);
240 }
241
242 void display_setcontrast(uint8_t contrast){
243   instruction(LCD_FUNCTIONSET | display_extended);
244   _delay_us(30);
245
246   instruction(LCD_ICON_CONTRAST_HIGH_BYTE | LCD_ICON_ON | LCD_BOOSTER_ON | (contrast >>
        4 & LCD_CONTRAST_HIGH_BYTE_MASK));
247   _delay_us(30);
248
249   instruction(LCD_CONTRAST_LOW_BYTE | (contrast & LCD_CONTRAST_LOW_BYTE_MASK));
250   _delay_us(30);
251
252 }
```

6.1.7 main.c

```
1
2 #include <util/twi.h>
3 #include <avr/interrupt.h>
4 #include <stdbool.h>
5 #include <inttypes.h>
6 #include "serial.h"
7 #include "display.h"
8 #include "E2PROM.h"
9 #include "menu.h"
10 #include "motor.h"
11 #include <avr/sleep.h>
12 #include <util/delay.h>
13 #include <avr/io.h>
14 #include <stdbool.h>
15 #include "polsadors.h"
16 #include "sleep.h"
17
18
19 void loop(void){
20     //Main loop of the system
21     desperta();
22     benvinguda();
23     //write_eprom();
24     //comparacio();
25     authentication();
26     adeu();
27     set_up2();
28     degree();
29     display_clear();
30     adorm();
31 }
32
33
34
35 int main(void){
36   //system init
37   //benvinguda();
38   init_pols();
39   sei();
40   //degree();
41   //loop();
42   int a;
43   while(true){
44   loop();
45   }
46   return 0;
47 }
48
49 // Interrupt atention routine
50 ISR(INT1_vect){
51   _delay_ms(20);
52   loop();
53   //if (valor_alt()){
54      //display_clear();
55      //loop();
56     // If it gets here is because there is an interrupt
57  //}
58   desactiva_flag();
59 }
```

6.1.8 Makefile

```
CC=avr-gcc
CPPFLAGS=-DF_CPU=16000000UL
CFLAGS=-Wall -std=c99 -Os -mmcu=atmega328p -fshort-enums -Ilibpbn
LDFLAGS=-mmcu=atmega328p

#CC=avr-gcc
#CPPFLAGS=-DF_CPU=16000000UL
#CFLAGS=-Wall -std=c99 -Os -mmcu=atmega328p -fshort-enums
#LDFLAGS=-mmcu=atmega328p



queue.o: queue.h
serial.o: serial.h queue.h
twi.o: twi.h serial.h display.h queue.h
display.o: display.h twi.h serial.h
E2PROM.o: E2PROM.h i2c.h display.h twi.o
menu.o: E2PROM.h menu.h i2c.h display.h serial.h twi.h
i2c.o: i2c.h
motor.o: serial.h motor.h queue.h
sleep.o: polsadors.h serial.h queue.h
polsadors.o: serial.h polsadors.h

#prova_display.o: twi.h display.h serial.h
#prova_display: twi.o display.o serial.o queue.o

main2.o: serial.h display.h i2c.h E2PROM.h twi.h menu.h motor.h polsadors.h sleep.h
main2: serial.o queue.o display.o i2c.o E2PROM.o twi.o menu.o motor.o polsadors.o sleep.
    o



#main3.o: twi.h display.h serial.h queue.h
#main3: twi.o display.o serial.o queue.o



#driver_pressure.o: twi.h

%.hex : %
  avr-objcopy -Oihex $< $@

%_c: %.hex
  avrdude -c arduino -p atmega328p -P /dev/ttyACM0 -U $<
  rm *.o

%_c1: %.hex
  avrdude -c arduino -p atmega328p -P /dev/ttyACM1 -U $<
  rm *.o

%_isp: %.hex
  avrdude -c avrisp -b19200 -p atmega328p -P /dev/ttyACM0 -U $<
  rm *.o


.PHONY: clean veryclean
clean:
  \rm -f *~ *.o *.s *.hex
  $(MAKE) -C libpbn clean
veryclean: clean
  \rm -f master
  $(MAKE) -C libpbn veryclean
```