



# PROGRAMMING OF A LED MATRIX WITH A DIGITAL VU METER APPLICATION

WS 2015 BACHELOR ARBEIT

Academic Supervisor: Prof. Dr. Roland Mandl

Technical Supervisor: Gerald Schickuber

Student: Norberto Albujer

Matriculation Number: 2738245



# INDEX

---

<b>INTRODUCTION .....</b>	<b>9</b>
<b>CHAPTER 1 .....</b>	<b>11</b>
<b>ARDUINO BASIC FEATURES.....</b>	<b>11</b>
1.1    WHAT IS ARDUINO?.....	12
1.2    HARDWARE .....	12
1.2.1 ARDUINO DUE.....	12
1.3 SOFTWARE.....	23
1.3.1 ARDUINO SOFTWARE .....	23
1.3.2 ADAFRUIT NEO PIXEL LIBRARY .....	26
<b>CHAPTER 2 .....</b>	<b>27</b>
<b>DEVICES CONNECTED TO ARDUINO .....</b>	<b>27</b>
2.1 LED MATRIX .....	28
2.2 WS2812b LED .....	30
2.3 ELECTRIC CIRCUIT.....	37
2.3.1THEORICAL ELECTRIC CIRCUIT .....	37
2.3.2 PRACTICAL CIRCUIT ON PROTOBOARD .....	50
2.3.3 PCB DESIGN .....	52
<b>CHAPTER 3 .....</b>	<b>58</b>
<b>3D CAD DESIGN WITH SOLIDWORKS.....</b>	<b>58</b>
3.1 WHAT IS SOLIDWORKS?.....	59
3.2 3D CAD LED MATRIX BOX DESIGNS.....	59
<b>CHAPTER 4 .....</b>	<b>63</b>
<b>3D PRINTER .....</b>	<b>63</b>
4.1 PRINTER SOFTWARE .....	64
4.2 PRINTER SLICER .....	65
4.3 RESULTS .....	66
<b>CHAPTER 5 .....</b>	<b>69</b>
<b>APPLICATION .....</b>	<b>69</b>
5.1 APPLICATION MODES.....	70
5.2 PROGRAMMING USER FUNCTIONS .....	71

5.2 FLOW CHART .....	77
<b>CHAPTER 6 .....</b>	<b>79</b>
<b>CONCLUSIONS AND VALUATIONS .....</b>	<b>79</b>
6.1 CONCLUSION AND PROBLEMS .....	80
6.2 VALUATIONS .....	81
<b>BIBLIOGRAPHY .....</b>	<b>82</b>
<b>ANNEX 1. ADAFRUIT NEOPIXEL LIBRARY .....</b>	<b>83</b>
<b>ANNEX 2. APPLICATION CODE.....</b>	<b>87</b>
<b>ANNEX 3. VU METER PCB.....</b>	<b>116</b>
<b>ANNEX 4. 3D CAD PLANS .....</b>	<b>120</b>

## LIST OF FIGURES

- **Figure 1.** Front and back side of Arduino Due
- **Figure 2.** Arduino Due PinOut diagram
- **Figure 3.** Arduino's Due USB ports
- **Figure 4.** Composition of arduino's software
- **Figure 5.** Ws2812b LED matrix
- **Figure 6.** Ws2812b LED mechanical dimensions
- **Figure 7.** Ws2812b LED pin configuration
- **Figure 8.** Ws2812b LED sequence chart
- **Figure 9.** Ws2812b LED cascade method
- **Figure 10.** Ws2812b LED data transmission code
- **Figure 11.** Ws2812b LED typical application circuit
- **Figure 12.** Schematic of an operational amplifier
- **Figure 13.** Schematic of TLV2772A operational amplifier
- **Figure 14.** Dual voltage power supply for an operational amplifier
- **Figure 15.** Artificial ground for an operational amplifier
- **Figure 16.** Operational amplifier with the inputs connected to the same potential.
- **Figure 17.** AC capacitive coupling in an operational amplifier
- **Figure 18.** Schematic of a differential amplifier
- **Figure 19.** Schematic of an inverter operational amplifier
- **Figure 20.** Schematic of a buffer
- **Figure 21.** Schematic circuit for bode diagram
- **Figure 22.** Bode diagram
- **Figure 23.** Microphone und amplifier signals
- **Figure 24.** Schematic of the final amplifier electric circuit
- **Figure 25.** Control panel of EAGLE software
- **Figure 26.** Schematic of the electronic board
- **Figure 27.** Layout of the electronic board
- **Figure 28.** PCB with elements solded
- **Figure 29.** Safety connections for the PCB
- **Figure 30.** SolidWorks software
- **Figure 31.** First 3D LED matrix box design
- **Figure 32.** Second 3D LED matrix box design
- **Figure 33.** Final 3D LED matrix box design
- **Figure 34.** X400 3D printer
- **Figure 35.** Simplify 3D software
- **Figure 36.** 3D printer parameters
- **Figure 37.** Failed printout
- **Figure 38.** Third trial results
- **Figure 39.** Application flow chart
- **Figure 40.** Application modes

## LIST OF TABLES

- Table 1. Summary of Arduino's Due features
- Table 2. Specifications LED matrix
- Table 3. WS2812b LED pin functions
- Table 4. WS2812b LED absolute maximum ratings
- Table 5. WS2812b LED electrical characteristics
- Table 6. WS2812b LED switching characteristics
- Table 7. WS2812b LED characteristics parameter
- Table 8. WS2812b LED data transfer time
- Table 9. Features of the TLV2772A Operational Amplifier.

## ABBREVIATIONS AND DEFINITIONS

- **ADC:** Analog to digital converter
- **API:** In computer programming, an application programming interface (API) is a set of routines, protocols, and tools for building software applications. An API expresses a software component in terms of its operations, inputs, outputs, and underlying types. An API defines functionalities that are independent of their respective implementations, which allows definitions and implementations to vary without compromising each other. A good API makes it easier to develop a program by providing all the building blocks. A programmer then puts the blocks together.
- **ARM:** ARM is a family of instruction set architectures for computer processors based on a reduced instruction set computing (RISC) architecture developed by British company ARM Holdings.
- **AVR microcontroller:** The AVR is a modified Harvard architecture 8-bit RISC single-chip microcontroller, which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to one-time programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time. MegaAVR chips became popular after they were designed into the 8-bit Arduino platform.
- **CPU:** Central Processing Unit
- **DAC:** Digital to analog converter (DAC, D/A, D2A or D-to-A).
- **I<sup>2</sup>C:** *I<sup>2</sup>C (Inter-Integrated Circuit)*, pronounced I-squared-C, is a multi-master, multi-slave, single-ended, serial computer bus invented by Philips Semiconductor, known today as NXP Semiconductors, used for attaching low-speed peripherals to computer motherboards and embedded systems. Alternatively I<sup>2</sup>C is spelled I2C (pronounced I-two-C) or IIC (pronounced I-I-C).
- **ICSP:** It is an AVRtiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, GND. It is often referred to as an SPI (Serial Peripheral Interface) which could be considered an "expansion" of the output, but really, you are slaving the output device to the master of the SPI bus.

- **JTAG:** Joint Test Action Group (JTAG) is the common name for the IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture. It was initially devised by electronic engineers for testing printed circuit boards using boundary scan and is still widely used for this application.

Today, JTAG is also widely used for IC debug ports. In the embedded processor market, essentially all modern processors implement JTAG when they have enough pins. Embedded development relies on debuggers communicating with chips with JTAG to perform operations like single stepping and break pointing.

- **MCU:** Microcontroller Unit
- **O.A.:** Operational Amplifier
- **PCB:** Printed Circuit Board
- **PWM:** Pulse With Modulation
- **RAM:** Random Access Memory
- **USB:** Universal Serial Bus
- **USB OTG:** *USB On-The-Go*, often abbreviated to USB OTG or just OTG, is a specification first used in late 2001, that allows USB devices such as digital audio players or mobile phones to act as a host, allowing other USB devices like a USB flash drive, digital camera, mouse or keyboard to be attached to them.
- **UART:** Universal Asynchronous Receiver-Transmitter
- **SPI:** The Serial Peripheral Interface (SPI) bus is a synchronous serial communication interface specification used for short distance communication, primarily in embedded systems.
- **SDA:** Synchronous Data Adapter
- **SCL:** Synchronous Clock
- **TTL:** TTL serial (*transistor-transistor logic*). Serial communication at a TTL level will always remain between the limits of 0V and Vcc, which is often 5V or 3.3V. A logic high ('1') is represented by Vcc, while a logic low ('0') is 0V.



# INTRODUCTION

---

It's amazing how the light emitting diode (LED), which began as a faint glimmer in the sixties, has already surpassed incandescent and fluorescent lamps in terms of efficiency. Every time we are closer to the center of the era of LED, we use its light in many applications and many more are coming. In this project LED is one of the most important things, because of that, I am going to discuss a little about the evolution of LED lighting.

We could say that the era of LED begins in 1962 with Nick Holonyak Jr., who developed the first visible LED. These LEDs emitted a faint red light. Shortly after was introduced to the market, but the light output was so small that only was used as an indicator light.

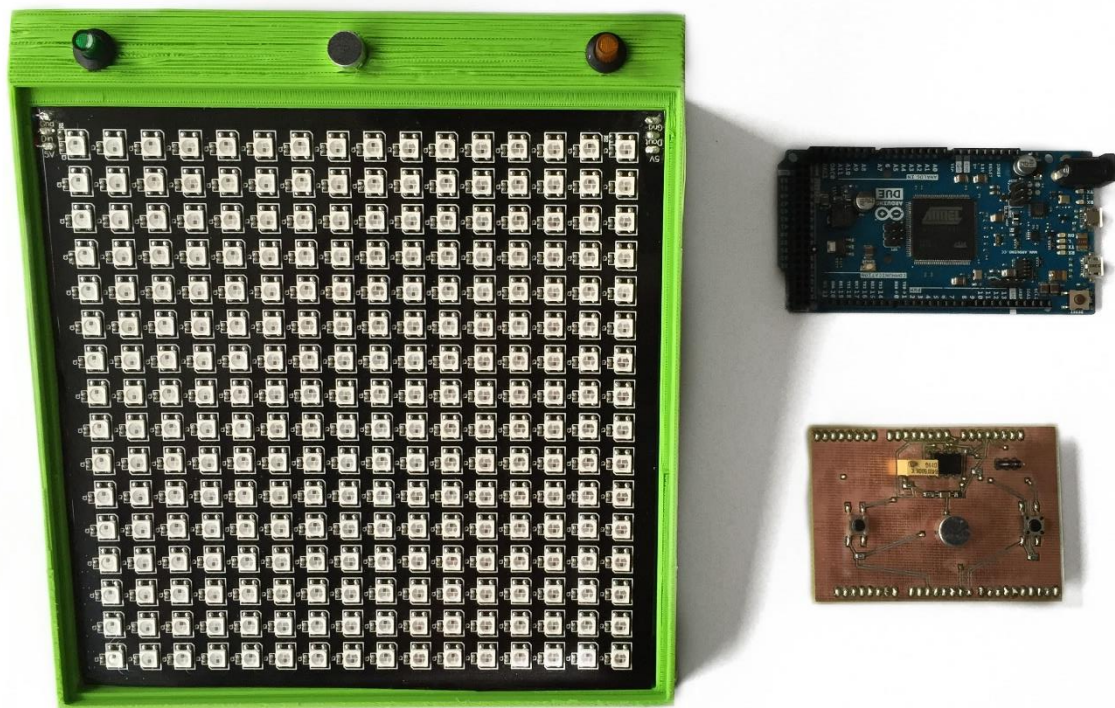
For about 20 years the LED remained as a source of low brightness, until in the eighties the first high-brightness red LEDs were created. This transformed the faint glimmer of Holonyak, opening new applications for LEDs, mainly in traffic lights. But they remained indicator lights. The decisive event occurred in 1993 when Shuji Nakamura developed the first superluminescent blue LED. For the first time could be generated a white LED light. The blue LED superluminescent produced enough light to excite a phosphor coating and thereby generate white light. Thus was born an opened way to new LED lighting technology, also known as solid-state lighting. Shortly thereafter, in 1995, the Japanese company Nichia brought to market the first white LEDs. These LEDs had little luminous efficiency ( $\sim 5 \text{ lm} / \text{W}$ ), and a very low colour rendering index ( $\text{CRI} \sim 60$ ), among other limitations.

Despite all disadvantages, many people began to glimpse the great potential of LEDs, and in 1999 Philips launched the first high-power LEDs 1W. LEDs quickly exceeded the limit of efficiency of  $17 \text{ lm} / \text{W}$  imposed by the incandescent bulb. In 2002, Philips introduced LEDs with luminous efficacies of up to  $22 \text{ lm} / \text{W}$ . Last year, the LED exceed the maximum luminous efficiency of fluorescent lamps. This, along with its other advantages, LED is standing in front of all lighting technologies. LED has more challenges remain, particularly the acquisition cost. According trends and predictions, LED prices will drop enough to enter the market in 2015 and it is expected that by 2020, dominate all markets.

In this project I want to develop an application for a 16\*16 LED Matrix. It's composed of 256 RGB 5050 programmable LEDs, in concrete the ws2812b LED. The idea is to use an Arduino board (microcontroller) to control the LED Matrix with a programming code and create, as a principal application, a digital equalizer and use two additional buttons for other modes.

To create the application, we are going design a PCB with a microphone to receive the signal of the music or voice. Also, we'll add the two additional buttons in this PCB.

The last step of this project is to improve the physical aspect of the LED Matrix. If we have time we would create a 3D CAD design that later would be printed in a 3D printer.



# CHAPTER 1

## ARDUINO BASIC FEATURES

---

In this section we are going to define what Arduino is, describe the principal elements of an Arduino board and the development of the Arduino programming code, that's mean hardware and software that works in Arduino.

## 1.1 WHAT IS ARDUINO?

Arduino is an “open-source” electronics platform based on “easy-to-use” hardware and software. It's intended for anyone making interactive projects.

## 1.2 HARDWARE

Arduino senses the environment by receiving inputs from many sensors, and affects its surroundings by controlling lights, motors, and other actuators.

Being free Arduino hardware platform, their design and their distribution can be freely used for the development of any project without acquiring a license. So there are different types of boards, ones created by the Arduino community (official) or others created by third parties but with similar features. In our project we used Arduino Due board. The features of it are described in the next point.

### 1.2.1 ARDUINO DUE

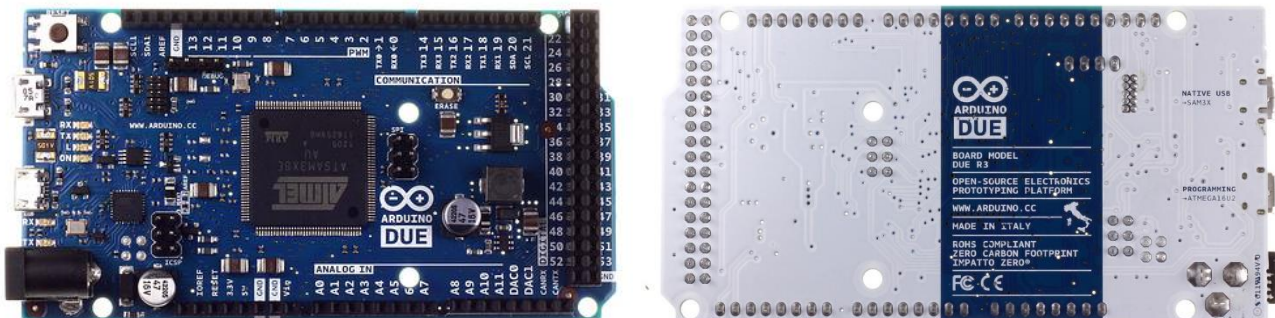


Figure 1: Front and back side of Arduino Due

## INTRODUCTION

The Arduino Due is a microcontroller board based on the *Atmel SAM3X8E ARM Cortex-M3 CPU*. It's the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as *PWM* outputs), 12 analog inputs, 4 *UARTs* (hardware serial ports), a 84 MHz clock, an *USB OTG* capable connection, 2 *DAC* (digital to analog), 2 *TWI*, a power jack, an *SPI* header, a *JTAG* header, a reset button and an erase button.

**Warning:** *Unlike other Arduino boards, the Arduino Due board runs at 3.3V. The maximum voltage that the I/O pins can tolerate is 3.3V. Providing higher voltages, like 5V to an I/O pin could damage the board.*

The board contains everything needed to support the microcontroller; simply connect it to a computer with a micro-USB cable or power it with an ADC adapter or battery to get started. The Due is compatible with all Arduino shields that work at 3.3V and are compliant with the 1.0 Arduino pinout.

The Due follows the 1.0 pinout:

- *TWI*: SDA and SCL pins that are near to the AREF pin.
- The IOREF pin which allows an attached shield with the proper configuration to adapt to the voltage provided by the board. This enables shield compatibility with a 3.3V board like the Due and AVR-based boards which operate at 5V.
- An unconnected pin, reserved for future use.

## ARM CORE BENEFITS

The Due has a 32-bit ARM core that can outperform typical 8-bit microcontroller boards. The most significant differences are:

- A 32-bit core, that allows operations on 4 bytes wide data within a single CPU clock.

- CPU Clock at 84 MHz
- 96 Kbytes of SRAM
- 512 Kbytes of Flash memory for code
- A DMA controller that can relieve the CPU from doing memory intensive tasks

## FEATURES SUMMARY

Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Outputs Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

Table 1. Summary of Arduino's Due Features

## POWER

The Arduino Due can be powered via the USB connector or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are:

- VIN. The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or if supplying voltage via the power jack, access it through this pin.
- 5V. this pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the Vin pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board.
- 3.3V. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 800 mA. This regulator also provides the power supply to the *SAM3X microcontroller*.
- GND. Ground pins.
- IOREF. This pin on the Arduino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs for working with the 5V or 3.3V.



## MEMORY

The SAM3X has 512 KB (2 blocks of 256 KB) of flash memory for storing code. The boot loader is preburned in factory from Atmel and is stored in a dedicated ROM memory. The available SRAM is 96 KB in two contiguous bank of 64 KB and 32 KB. All the available memory (Flash, RAM and ROM) can be accessed directly as a flat addressing space.

It is possible to erase the Flash memory of the SAM3X with the onboard erase button. This will remove the currently loaded sketch from the MCU. To erase, press and hold the Erase button for a few seconds while the board is powered.

## INPUT AND OUTPUT

- **Digital I/O:** pins from 0 to 53

Each of the 54 digital pins on the Due can be used as an input or output, using `pinMode ()`, `digitalWrite ()`, and `digitalRead ()` functions. They operate at 3.3 volts. Each pin can provide a current of 3 mA or 15 mA, depending on the pin, or receive a current of 6 mA or 9 mA, depending on the pin.

They also have an internal pull-up resistor (disconnected by default) of 100 KOhm. In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX)
- Serial 1: 19 (RX) and 18 (TX)
- Serial 2: 17 (RX) and 16 (TX)
- Serial 3: 15 (RX) and 14 (TX)

Used to receive (RX) and transmit (TX) TTL serial data (with 3.3 V level). Pins 0 and 1 are connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip.



- **PWM:** Pins 2 to 13.

Provide 8-bit PWM output with the analogWrite () function. The resolution of the *PWM* can be changed with the analogWriteResolution () function.

- **SPI:** *SPI* header (*ICSP* header on other Arduino boards).

These pins support SPI communication using the SPI library. The SPI pins are broken out on the central 6-pin header, which is physically compatible with the Uno, Leonardo and Mega2560. The SPI header can be used only to communicate with other SPI devices, not for programming the SAM3X with the In-Circuit-Serial-Programming technique. The SPI of the Due has also advanced features that can be used with the Extended SPI methods for Due.

- **CAN:** CANRX and CANTX

These pins support the *CAN* communication protocol but are not yet supported by Arduino *APIs*.

- **"L" LED:** 13

There is a built-in LED connected to digital pin 13. When the pin is HIGH, the LED is on, when the pin is LOW, it's off. It is also possible to dim the LED because the digital pin 13 is also a *PWM* output.

- **TWI 1:** 20 (*SDA*) and 21 (*SCL*)

- **TWI 2:** SDA1 and SCL1.

- **Support TWI communication using the Wire library.**

SDA1 and SCL1 can be controlled using the Wire1 class provided by the Wire library. While *SDA* and *SCL* have internal pull-up resistors, SDA1 and SCL1 have not. Adding two pull-up resistor on SDA1 and SCL1 lines is required for using Wire1.

- **Analog Inputs:** pins from A0 to A11

The Due has 12 analog inputs, each of which can provide 12 bits of resolution (i.e. 4096 different values). By default, the resolution of the readings is set at 10 bits, for compatibility with other Arduino boards. It is possible to change the resolution of the ADC with analogReadResolution ().

The Due's analog inputs pins measure from ground to a maximum value of 3.3V. Applying more than 3.3V on the Due's pins will damage the SAM3X chip. The **analogReference ()** function is ignored on the Due.

The *AREF* pin is connected to the SAM3X analog reference pin through a resistor bridge. To use the *AREF* pin, resistor *BR1* must be desoldered from the *PCB*.

- **DAC1 and DAC2**

These pins provides true analog outputs with 12-bits resolution (4096 levels) with the **analogWrite ()** function. These pins can be used to create an audio output using the Audio library.

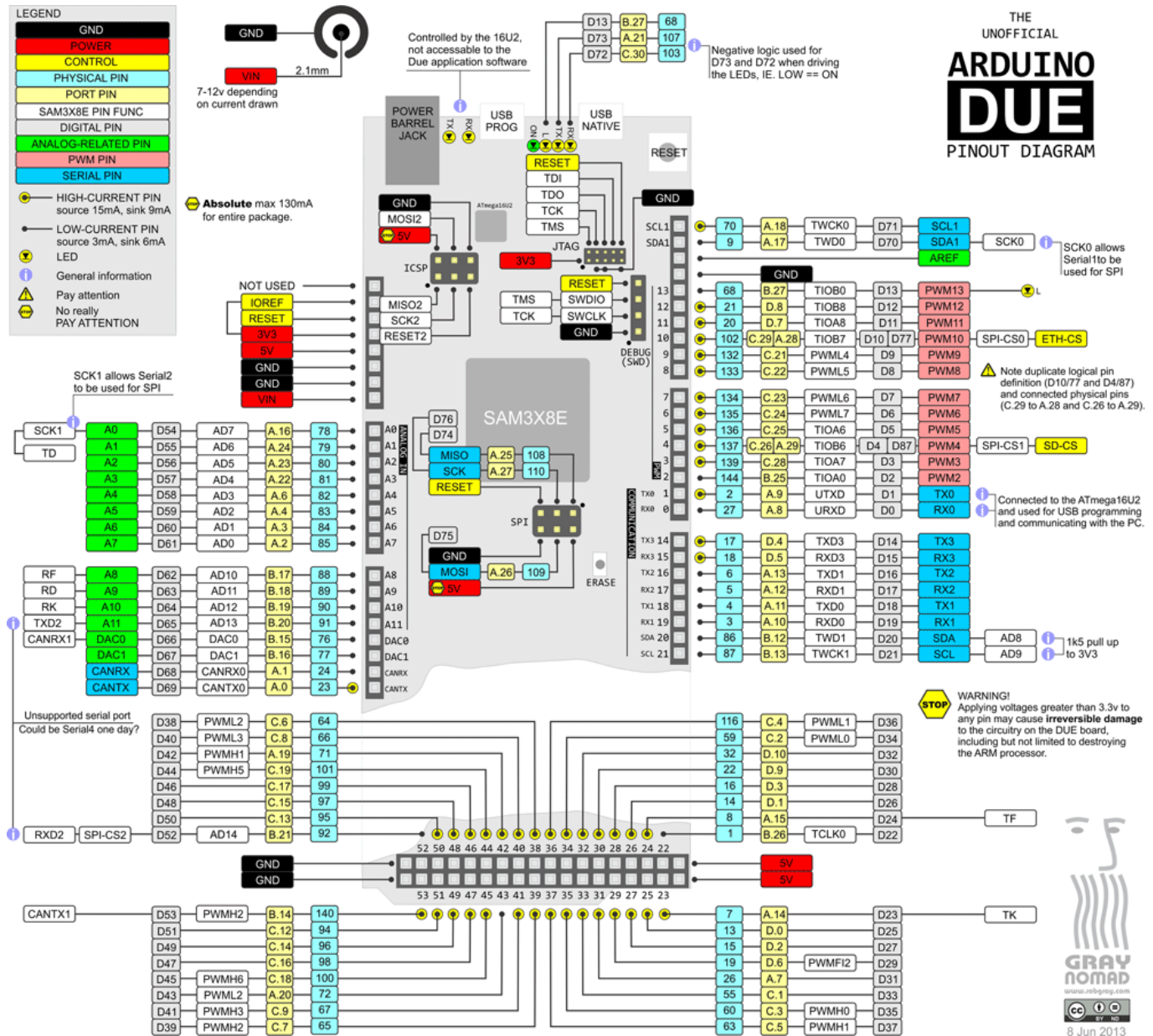
#### OTHER PINS ON THE BOARD:

- **AREF**

Reference voltage for the analog inputs. Used with **analogReference ()**.

- **Reset**

Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.



## COMMUNICATION:

The Arduino Due has a number of facilities for communicating with a computer, another Arduino or other microcontrollers, and different devices like phones, tablets, cameras and so on. The SAM3X provides one hardware *UART* and three hardware *USARTs* for *TTL* (3.3V) serial communication.

The Programming port is connected to an *ATmega16U2*, which provides a virtual COM port to software on a connected computer (To recognize the device, Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically.). The 16U2 is also connected to the SAM3X hardware *UART*. Serial on pins RX0 and TX0 provides Serial-to-USB communication for programming the board through the ATmega16U2 microcontroller. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board.

The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

The Native USB port is connected to the SAM3X. It allows for serial (*CDC*) communication over USB. This provides a serial connection to the Serial Monitor or other applications on your computer. It also enables the Due to emulate a USB mouse or keyboard to an attached computer. To use these features, see the Mouse and Keyboard library reference pages.

The Native USB port can also act as a USB host for connected peripherals such as mice, keyboards, and smartphones. To use these features, see the *USBHost* reference pages.

The SAM3X also supports *I2C* and *SPI* communication. The Arduino software includes a Wire library to simplify use of the *I2C* bus. For *SPI* communication, use the SPI library.

## PROGRAMMING

Uploading sketches to the SAM3X is different than the AVR microcontrollers found in other Arduino boards because the flash memory needs to be erased before being re-programmed. Upload to the chip is managed by ROM on the SAM3X, which is run only when the chip's flash memory is empty.

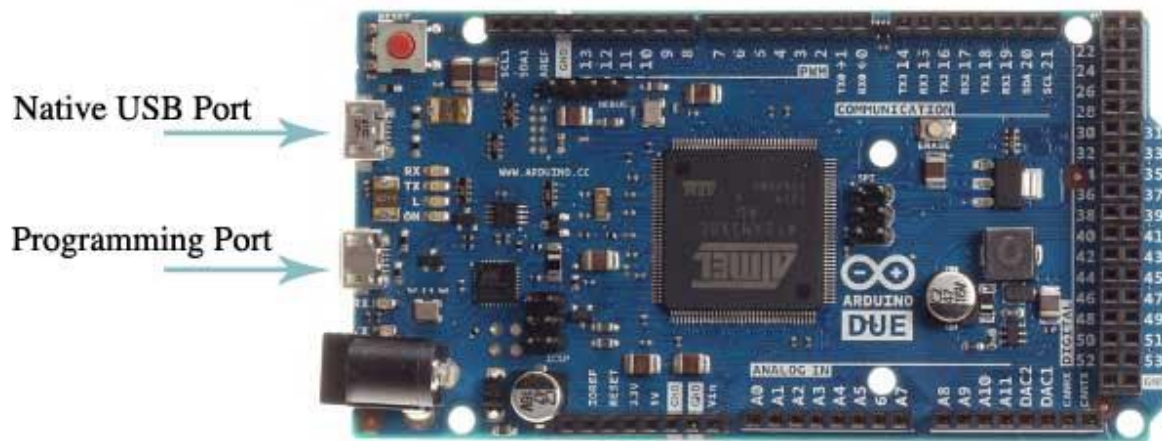


Figure 3. Arduino Due USB Ports

Either of the USB ports can be used for programming the board, though it is recommended to use the Programming port due to the way the erasing of the chip is handled:

- Programming port:** To use this port, select "Arduino Due (Programming Port)" as your board in the Arduino IDE. Connect the Due's programming port to your computer. The programming port uses the 16U2 as a USB-to-serial chip connected to the first UART of the SAM3X (RX0 and TX0). The 16U2 has two pins connected to the Reset and Erase pins of the SAM3X. Opening and closing the Programming port connected at 1200bps triggers a "hard erase" procedure of the SAM3Xchip, activating the Erase and Reset pins on the SAM3X before communicating with the UART. This is the recommended port for programming the Due. It is more reliable than the "soft erase" that occurs on the Native port, and it should work even if the main MCU has crashed.

- **Native port:** To use this port, select "Arduino Due (Native USB Port)" as your board in the Arduino IDE. The Native USB port is connected directly to the SAM3X. Connect the Due's Native USB port (the one closest to the reset button) to your computer. Opening and closing the Native port at 1200bps triggers a 'soft erase' procedure: the flash memory is erased and the board is restarted with the bootloader. If the MCU crashed for some reason it is likely that the soft erase procedure won't work as this procedure happens entirely in software on the SAM3X. Opening and closing the native port at a different baudrate will not reset the SAM3X.

The ATmega16U2 firmware source code is available in the Arduino repository. You can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

\*The Arduino Due can be programmed with the Arduino software.

## USB OVERCURRENT PROTECTION

The Arduino Due has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

## PHYSICAL CHARACTERISTICS AND SHIELD COMPATIBILITY

The maximum length and width of the Arduino Due *PCB* are 4 and 2.1 inches respectively, with the USB connectors and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Arduino Due is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent *AREF* and *GND* pins), analog inputs 0 to 5, the power header, and "*ICSP*" (*SPI*) header are all in equivalent locations. Further the main *UART* (serial port) is located on the same pins (0 and 1). *Please note that I<sup>2</sup>C is not located on the same pins on the Due (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).*

## 1.3 SOFTWARE

The Arduino platform has its own language that is based on C / C++ and therefore supports standard C functions and some C++. However, it is possible to use other programming languages and popular applications in Arduino like Java, Processing, Python, Mathematica, Matlab, Perl, Visual Basic, etc. This is possible because Arduino communicates by transmitting serial data which is a format that most of the above languages can support. For those who do not support natively series format, you can use a software to translate the messages sent by both sides to allow a fluid communication.

It is quite interesting to be able to interact with Arduino by this variety of systems and languages. Depending on which are the needs of the problem that we are going to solve we can take advantage of the great media compatibility offered.

The Arduino development environment is simple and intuitive. Also is available for free download from their official website and for different operating systems. It has been implemented with Processing that is similar to Java. Its latest version is 1.6.0 but in the project has been used 1.5.8.

### 1.3.1 ARDUINO SOFTWARE

Now we are going to discuss the usefulness of each program area focusing only on what's important.

#### MENU

The most important part is in Tools. From here we can configure the program so that it can communicate with the Arduino. Doing click in card will list the types of Arduino boards that the program understands. Here, we select Arduino Due because is the one that we are going to use. In the Serial Port field select the one that matches our board connect via USB. If you use Windows the port will have name more or less like this COMx but in Linux will be / dev / ttyUSBx where x is a number. In case there are multiple serial ports and do not know which is for our board, we disconnect our board, we note the ports that appear, we reconnect the board and turn to check again the list of ports. The new port that appear will be the port of our board.



## COMMON BUTTONS

These buttons are quick access to certain actions which are also available by the menu. The buttons are:

- **Verify:** Verify and compile the code.
- **Upload:** in addition to compile the code is injected into the board.
- **New:** Creates a new sketch.
- **Open:** Opens a previously saved sketch.
- **Save:** stored on disk changes in the sketch.
- **Serial Monitor:** opens a new window where you can communicate bidirectionally via serial with the board, we can read the information that Arduino send us or we can provide it.

## TEXT EDITOR

In this area we will write the implementation (named for the sketch program) to load it into the Arduino board. The program has 3 parts. The first is the inclusion of libraries and the declaration of constants or global variables that can be used in any program function. The second is the **setup()** method, which is responsible for initializing the devices connected to the board and will be executed only after the system reboot. The third part is the **loop()** method, which you can run your code continuously. This is where the logic of the Arduino board will be written. As the language is very similar to C is possible to create other methods to separate functional blocks and leave ordered the program.

## MESSAGE AREA

Displays the status of the program using one of the common buttons.

## TEXT CONSOLE

Here appear in detail the events of message area.





Figure 4. Composition of Arduino's Software

### 1.3.2 ADAFRUIT NEO PIXEL LIBRARY

Controlling NeoPixels “from scratch” is quite a challenge, so we use Adafruit Neo Pixel Library in our project. This library is focus on the fun and interesting bits and works with most mainstream Arduino boards and derivatives with an Atmel AVR 8-bit processor from 8 to 16 MHz. Also works with the Arduino Due.

Installation of the library is as follows:

1. Visit the Adafruit\_NeoPixel library page at Github.com.
2. Select the “Download ZIP” button.
3. Uncompress the ZIP file after it’s finished downloading.
4. The resulting folder should contain the files “Adafruit\_NeoPixel.cpp”, “Adafruit\_NeoPixel.h” and an “examples” sub-folder. Sometimes in Windows you’ll get an intermediate-level folder and need to move things around.
5. Rename the folder (containing the .cpp and .h files) to “Adafruit\_NeoPixel” (with the underscore and everything), and place it alongside your other Arduino libraries, typically in your (home folder)/Documents/Arduino/Libraries folder. Libraries should not be installed alongside the Arduino application itself.
6. Re-start the Arduino IDE if it’s currently running.

Before you start with the code is important to know how are the LED of the matrix are connected. In our matrix are connected in serial, it means that is equal as a LED strip doing Zig-Zag. The most important functions of this library are explained in chapter five.

# CHAPTER 2

## DEVICES CONNECTED TO ARDUINO

---

## 2.1 LED MATRIX

The chosen LED Matrix display product is specially designed for the field of LED-Clothing. 16 Pixels are placed in each line, and there are 16 lines on each panel. The space between each pixel is 1cm. This product is totally able to meet the basic requirement of displaying. When you used it with a controller, it can also display numbers, video and so on. It has small size, light weight, an arbitrary curved, is easy to carry, Low-voltage drive, green energy, high brightness, low power and long life.

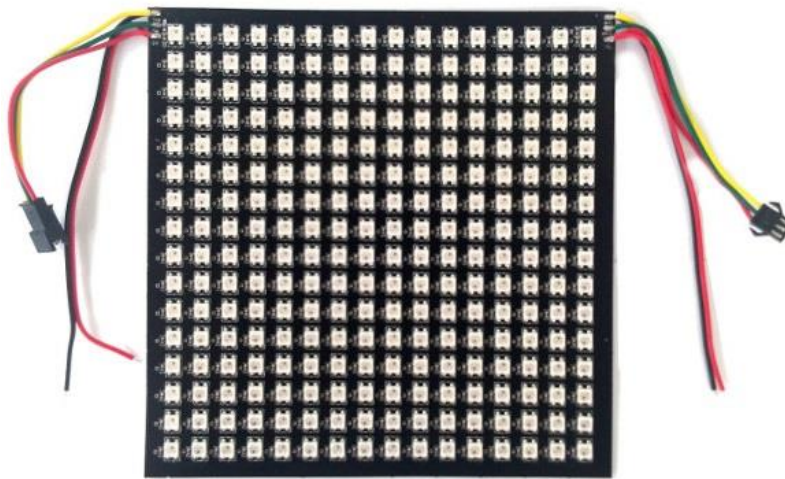


Figure 5. LED Matrix

## APPLICATIONS

- Widely used for home, hotels, clubs and shopping malls decoration.
- Architectural decorative lighting and boutique atmosphere lighting
- Extensively applied in Backlighting, concealed lighting and channel letter lighting
- Emergency & security lighting, advertisement sign lighting
- Decorative lights for holiday, event, and show exhibition
- Applicable for automobile and bicycle decoration, border or contour lighting

## FEATURES

- Super bright SMD top LED & viewing angle, high brightness output, no spot and shadow.
- Colours can be chosen as Full colour
- Low power consumption & operating voltage, safe and energy-saving.
- Solid-state, high shock or vibration resistant, long lifetime, more than 50000 hours.
- Easy installation with mounting holes and 3M adhesive tape on the back.
- ICRGB module with ws2812b
- Individually Control
- DMX 512 Controllable
- Matrix compatible

## SPECIFICATIONS

<b>Light source:</b>	LED	<b>Item type:</b>	Light Strips
<b>Type:</b>	Flex LED Strips	<b>Input voltage(v):</b>	5
<b>Lamp luminous flux(lm):</b>	4500	<b>Cri (ra&gt;):</b>	80
<b>Working temperature(°C):</b>	-20 - 60	<b>Working lifetime(hour):</b>	50000
<b>Emitting color:</b>	RGB	<b>Place of origin:</b>	China (Mainland)
<b>Model number:</b>	WS2812B1616	<b>Color:</b>	RGB
<b>Waterproof:</b>	IP68	<b>Angle:</b>	180
<b>Chip brand:</b>	Epistar	<b>Brand name:</b>	LC
<b>Led light source:</b>	Epistar	<b>Voltage:</b>	DC5V
<b>Lamp power:</b>	76.8W	<b>Ic:</b>	ws2801 ws2811 ipd8806
<b>Color temperature(cct):</b>	2700-7000	<b>Chip type:</b>	5050
<b>Size:</b>	170x170mm	<b>FPCB Color:</b>	Black

Table 2. Specifications LED Matrix

## 2.2 WS2812b LED

In this section we are going to see the features of the LEDs that are in the matrix. The model of the LED is IC RGB WS2812b.

### FEATURES AND BENEFITS:

- Intelligent reverse connect protection that does not damage the IC.
- The control circuit and the LED share the only power source.
- Control circuit and RGB chip are integrated in a package of 5050 components.
- Built in signal reshaping circuit.
- Built-in electric reset circuit and power lost reset circuit.
- Each pixel of the three primary color can achieve 256 brightness display, completed 16777216 color full color display, and scan frequency not less than 400Hz/s.
- Cascading port transmission signal by single line.
- Any two point the distance more than 5m transmission signal without any increase circuit.
- When the refresh rate is 30fps, cascade number are not less than 1024 points.
- Send data at speeds of 800Kbps.
- The color of the light were highly consistent.

### APPLICATIONS

- Full color module. Full color soft lights a lamp strip.
- LED decorative lighting. Indoor/outdoor LED video irregular screen.

## GENERAL DESCRIPTION

WS2812B is an intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components. It internal include intelligent digital port data latch and signal reshaping amplification drive circuit. Also include a precision internal oscillator and a 12V voltage programmable constant current control part, effectively ensuring the pixel point light color height consistent. The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto reshaping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission. LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angle is large, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume and convenient installation.

## LED MECHANICAL DIMENSIONS

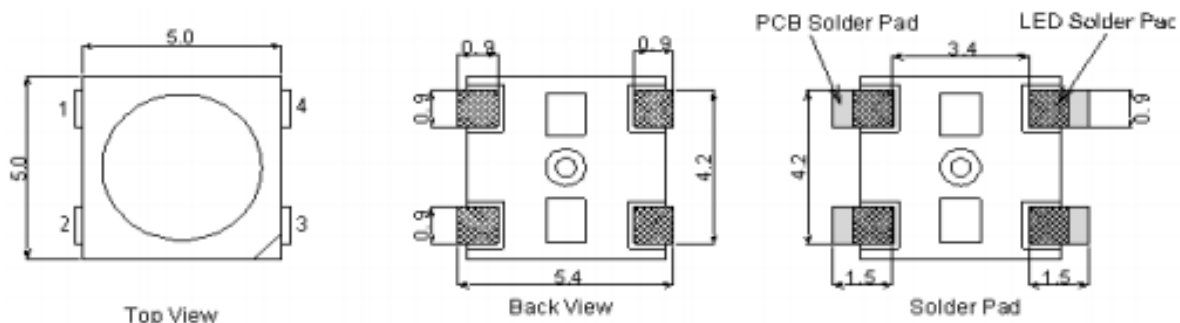


Figure 6. LED Mechanical Dimensions

## PIN CONFIGURATION

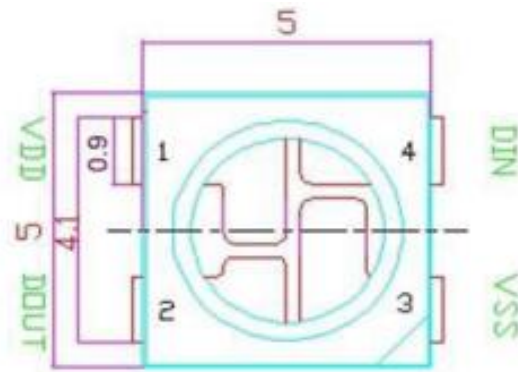


Figure 7. LED Pin Configuration

## PIN FUNCTION

NO.	Symbol	Function description
1	VDD	Power supply LED
2	DOUT	Control data signal output
3	VSS	Ground
4	DIN	Control data signal input

Table 3. LED Pin Functions

## ABSOLUTE MAXIMUM RATINGS

Prameter	Symbol	Ratings	Unit
Power supply voltage	$V_{DD}$	+3.5~+5.3	V
Input voltage	$V_I$	-0.5~ $V_{DD}+0.5$	V
Operation junction temperature	$T_{opt}$	-25~+80	°C
Storage temperature range	$T_{stg}$	-40~+105	°C

Table 4. Absolute Maximum LED Ratings



## ELECTRICAL CHARACTERISTICS

Parameter	Symbol	conditions	Min	Typ	Max	Unit
Input current	$I_I$	$V_I = V_{DD}/V_{SS}$	—	—	$\pm 1$	$\mu A$
Input voltage level	$V_{IH}$	$D_{IN}, SET$	$0.7V_{DD}$	—	—	V
	$V_{IL}$	$D_{IN}, SET$	—	—	$0.3 V_{DD}$	V
Hysteresis voltage	$V_H$	$D_{IN}, SET$	—	0.35	—	V

( $T_A = -20 \sim +70^\circ C$ ,  $V_{DD} = 4.5 \sim 5.5V$ ,  $V_{SS} = 0V$ , unless otherwise specified)

Table 5. LED Electrical Characteristics

## SWITCHING CHARACTERISTICS

Parameter	Symbol	Condition	Min	Typ	Max	Unit
Transmission delay time	$t_{PLZ}$	$CL = 15pF, D_{IN} \rightarrow D_{OUT}, RL = 10K\Omega$	—	—	300	ns
Fall time	$t_{THZ}$	$CL = 300pF, OUTR/OUTG/OUTB$	—	—	120	$\mu s$
Input capacity	$C_I$	—	—	—	15	pF

( $T_A = -20 \sim +70^\circ C$ ,  $V_{DD} = 4.5 \sim 5.5V$ ,  $V_{SS} = 0V$ , unless otherwise specified)

Table 6. LED Switching Characteristics

## LED CHARACTERISTICS PARAMETER

Emitting color	Model	Wavelength(nm)	Luminous intensity(mcd)	Voltage(V)
Red	13CBAUP	620-625	390-420	2.0-2.2
Green	13CGAUP	522-525	660-720	3.0-3.4
Blue	10R1MUX	465-467	180-200	3.0-3.4

Table 7. LED Characteristics Parameter

## DATA TRANSFER TIME

( $T_H + T_L = 1.25\mu s \pm 600ns$ )

T0H	0 code ,high voltage time	0.4us	$\pm 150ns$
T1H	1 code ,high voltage time	0.8us	$\pm 150ns$
T0L	0 code , low voltage time	0.85us	$\pm 150ns$
T1L	1 code ,low voltage time	0.45us	$\pm 150ns$
RES	low voltage time	Above 50 $\mu s$	

Table 8. LED Data Transfer Time

## SEQUENCE CHART

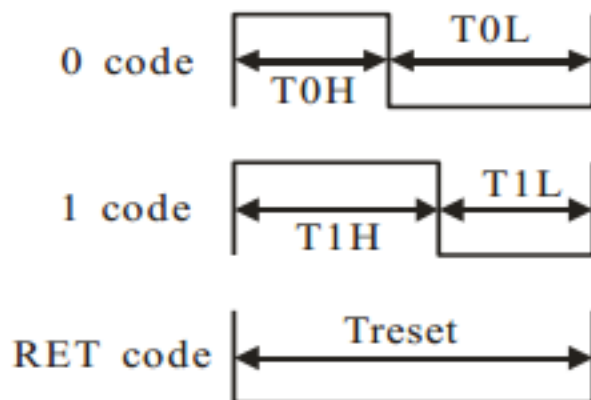


Figure 8. LED Sequence Chart

## CASCADE METHOD

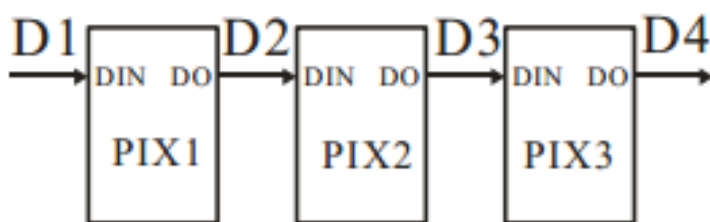
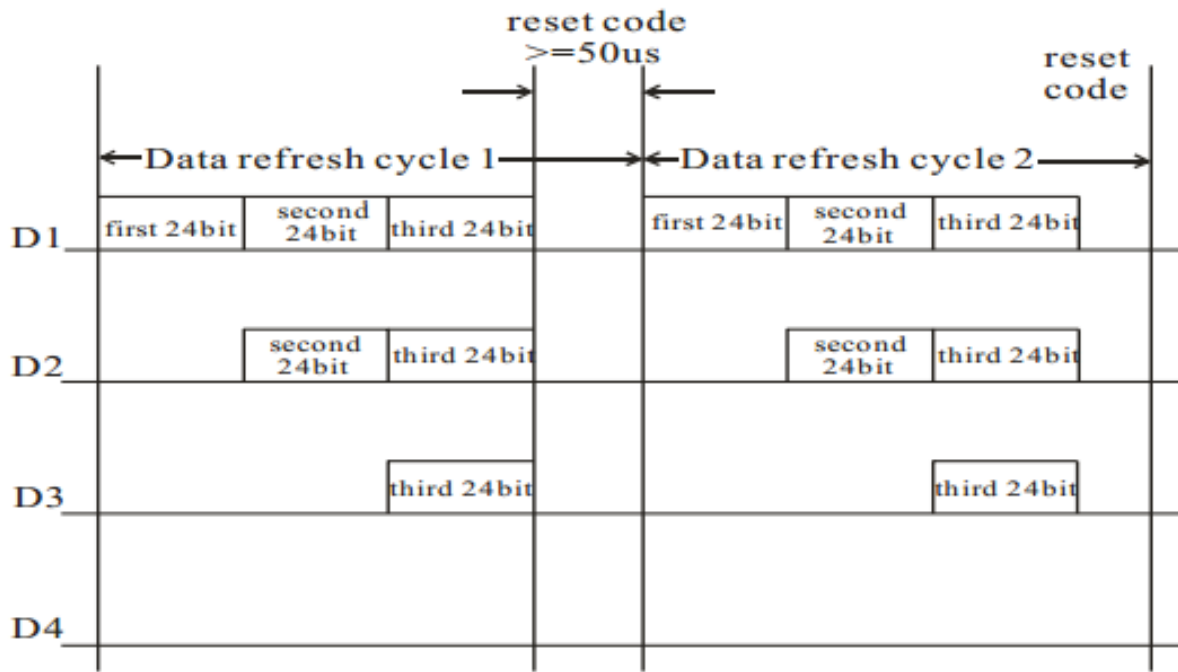


Figure 9. Cascade LED Method

## DATA TRANSMISSION METHOD



Note: The data of D1 is send by MCU, and D2, D3, D4 through pixel internal reshaping amplification to transmit.

Figure 10. LED Data Transmission Code

## COMPOSITION OF 24 BIT DATA

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Note: Follow the order of GRB to sent data and the high bit sent at first.

## TYPICAL APPLICATION CIRCUIT

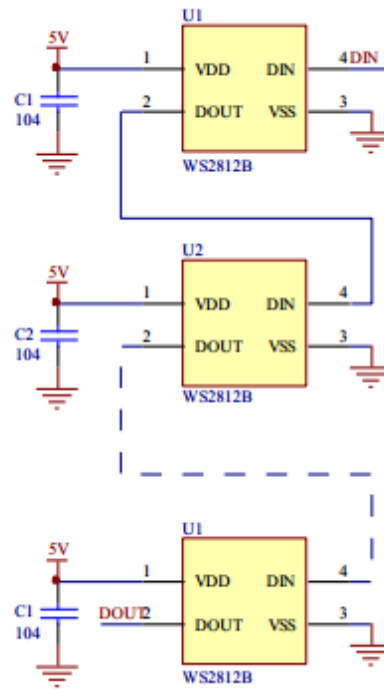


Figure 11. LED Typical Application Circuit

## 2.3 ELECTRIC CIRCUIT

As we are working on a project that responds to sounds, our circuit will require some type of microphone to transduce the sound into a modulated voltage and, most likely, some type of amplification of this modulated voltage.

We are going to use an electret microphones to transduce sound because they require relatively simple circuits.

### 2.3.1 THEORETICAL ELECTRIC CIRCUIT

Whenever we need to pick up a signal, we have to look all the way from its origin until we checked it. Particularly for a beep, I mean things like:

- **A good reconditioning**, observing the ideal conditions to generate the signal with minimal noise. This includes for example an anechoic chamber, a quiet room, isolate the system from mechanical vibrations, etc. When is possible, using a directional microphone we can prevent that ambient noise signals are captured. If we capture noise at such an early stage, will be almost impossible then to remove it, so we will find ways to grasp the signal as clean as possible.
- **A right microphone**: It doesn't means the most expensive. Sometimes it is better a carbon microphone and sometimes is preferable one condenser or electret microphone.

To choose the right microphone we will have to answer the next questions about the microphones:

- Directional or omnidirectional?
- What is the output impedance?
- What maximum SPL?
- Pre-amplified or not?

If the microphone is not well shielded also could capture electrical noise, that once amplified, can mask the useful signal.

- **Shielded conductors:** We must pay special attention not to pick up noise through the cables connecting the microphone with the preamplifier, and the different stages between them.
- **A low noise preamplifier:** Depending on the origin, the signal can be very weak. In this case, we can use several cascaded amplifier stages. It's important to minimize the noise, especially in at the beginning, not to amplify it with the signal.
- **Further treatment:** Whether we record the signal on a PC or we send it to a recorder, amplifier, etc. Maybe we will have to use different filters. These depend on the characteristics of the signal that interest us.

## WHAT IS AN OPERATIONAL AMPLIFIER?

An operational amplifier is a DC coupled high gain electronic voltage amplifier with a differential input and, usually, a single-ended output. In this configuration, an op-amp produces an output potential (relative to circuit ground) that is typically hundreds of thousands of times larger than the potential difference

## IDEAL OP-AMPS

An ideal op-amp is usually considered to have the following properties:

- Infinite open-loop gain  $G = v_{out} / v_{in}$
- Infinite input impedance  $R_{in}$ , and so zero input current
- Zero input offset voltage
- Infinite voltage range available at the output
- Infinite bandwidth with zero phase shift and infinite slew rate
- Zero output impedance  $R_{out}$
- Zero noise
- Infinite Common-mode rejection ratio (CMRR)
- Infinite Power supply rejection ratio.

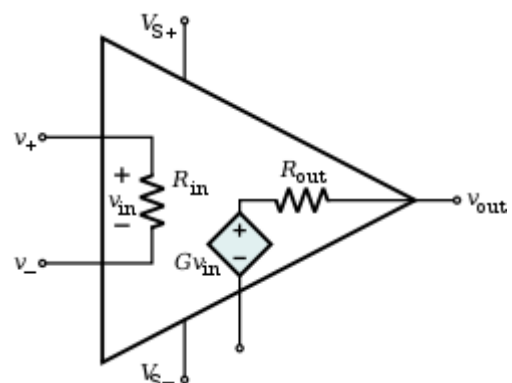


Figure 12. Schematic of an O.A.

These ideals can be summarized by the two "golden rules":

1. The output attempts to do whatever is necessary to make the voltage difference between the inputs zero.
2. The inputs draw no current.

The first rule only applies in the usual case where the op-amp is used in a closed-loop design (negative feedback, where there is a signal path of some sort feeding back from the output to the inverting input). These rules are commonly used as a good first approximation for analysing or designing op-amp circuits.

None of these ideals can be perfectly realized. A real op-amp may be modelled with non-infinite or non-zero parameters using equivalent resistors and capacitors in the op-amp model. The designer can then include these effects into the overall performance of the final circuit. Some parameters may turn out to have negligible effect on the final design while others represent actual limitations of the final performance that must be evaluated.

### TLV2772x OP AMP FAMILY

The TLV277x CMOS operational amplifier family combines high slew rate and bandwidth, rail-to-rail output swing, high output drive, and excellent dc precision. The device provides 10.5 V/ $\mu$ s of slew rate and 5.1 MHz of bandwidth while only consuming 1 mA of supply current per channel. This ac performance is much higher than current competitive CMOS amplifiers. The rail-to-rail output swing and high output drive make these devices a good choice for driving the analog input or reference of analog-to-digital converters. These devices also have low distortion while driving a 600- $\Omega$  load for use in telecom systems.



These amplifiers have a 360- $\mu$ V input offset voltage, a 17 nV/ $\sqrt{\text{Hz}}$  input noise voltage, and a 2-pA input bias current for measurement, medical, and industrial applications. The TLV277x family is also specified across an extended temperature range (-40°C to 125°C), making it useful for automotive systems, and the military temperature range (-55°C to 125°C), for military systems.

These devices operate from a 2.5-V to 5.5-V single supply voltage and are characterized at 2.7 V and 5 V. The single-supply operation and low power consumption make these devices a good solution for portable applications. The following table lists the packages available.

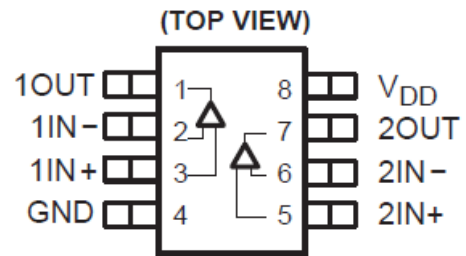


Figure 13. Schematic of TLV2772A O.A.

The meaning of rail to rail is that the input of the op amp is very similar to the output voltage. With this kind of op amp we can use more efficiently the low voltage rate with which they work.

	TLV2772A
Number of Channels (#)	2
Total Supply Voltage (Min) (+5V=5, +/-5V=10)	2.5
Total Supply Voltage (Max) (+5V=5, +/-5V=10)	5.5
Iq per channel (Max) (mA)	2
Slew Rate (Typ) (V/us)	10.5
Vos (Offset Voltage @ 25C) (Max) (mV)	1.6
Offset Drift (Typ) (uV/C)	2
CMRR (Min) (dB)	70
GBW (Typ) (MHz)	5.1
IIB (Max) (pA)	60
Vn at 1kHz (Typ) (nV/rtHz)	17
Rail-Rail	OUT
Rating	Catalog
Operating Temperature Range (C)	-40 to 125
Pin/Package	8PDIP 8SOIC 8TSSOP

Table 9. Features of the TLV2772A O.A.



## DUAL AND SINGLE POWER SUPPLY:

Operational amplifiers generally operate with dual voltage. That is, to supply 0, + V and -V. It would be something like:

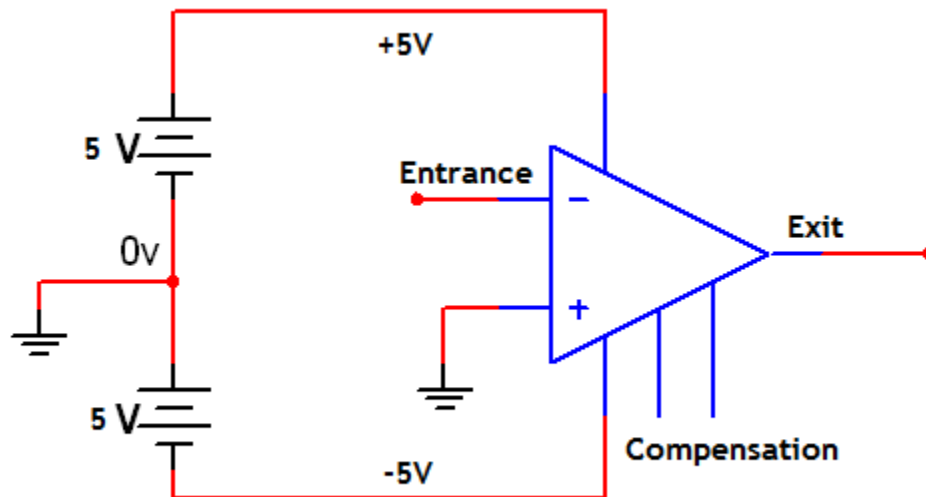


Figure 14. Operational Amplifier of Dual Voltage Power Supply

But often we only have a single source, 0 power, and + V, as a battery. When measuring voltages we always look a benchmark and as what we measure are potential differences, the 0 is an arbitrary point that depending on where we put the black lead of the multimeter we measure different voltages.

The trick to use operational amplifiers in circuits that do not have dual voltage is to create an artificial ground. Using a resistive divider with two resistors of equal value the intermediate node is just half the supply voltage.

Suppose we have a 5V battery. We put our reference, negative tip of the tester, on the negative side. We measure 0V in the negative thread, normal, there is no potential difference between our reference and herself. At the midpoint we measure 2.5V and 5V on the positive terminal of the battery. Now we change our reference and we put the negative lead at the halfway point. Will be measured -2.5V in the negative battery terminal, 0V at the junction and in the upper +2.5V.

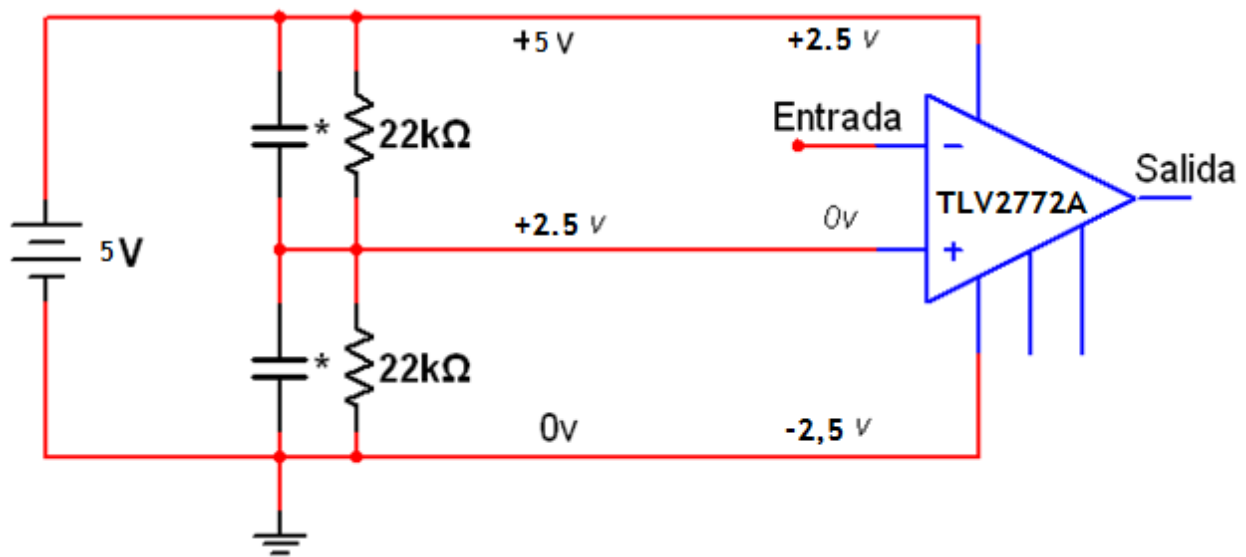


Figure 15. Artificial Ground for an Operational Amplifier

That will be the reference voltage for the operational amplifier. When using 5V it will be believed that we are supplying with a dual voltage of  $\pm 2.5\text{V}$ . As inputs require little current, the resistor value is not critical, it's enough if we have a stable voltage. Sometimes two small-capacity capacitors are added in parallel with the resistors, their function is to absorb any transients; normally can be removed without problem and are only used when the power supply is particularly noisy, such as in a car. It's often used a value of nF.

Another option for our artificial ground is to use another operational amplifier and connect together its inputs. In the image of the amplifier you can see that the inputs are connected at the same potential. Then its output should be 0V (with a minimum offset). But the operational amplifier thinks that is being supplied with dual voltage, its output will be 0 addressed to the dual voltage. Really the voltage in the output will be provided so that there the same potential between this and the positive supply voltage, between this and the negative voltage. In practice, this is just half the supply voltage, which is what we wanted.

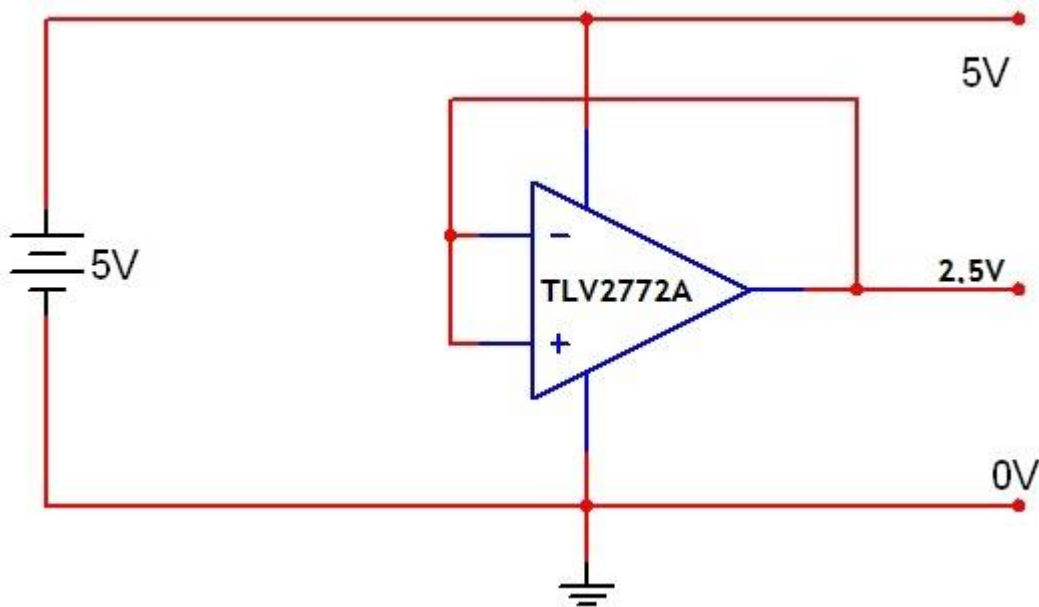


Figure 16: Operational Amplifier with the inputs connected to the same potential

The problem using the artificial ground for the non-inverting input is that it is not at the same potential as the real ground. In the last circuit the artificial ground (midpoint of the divisor) was 2.5V above the real ground (negative pole of the battery). It's necessary to remove the DC component at the input and the output and leave only the AC signal. This is achieved interposing a capacitor and is called AC capacitive coupling.

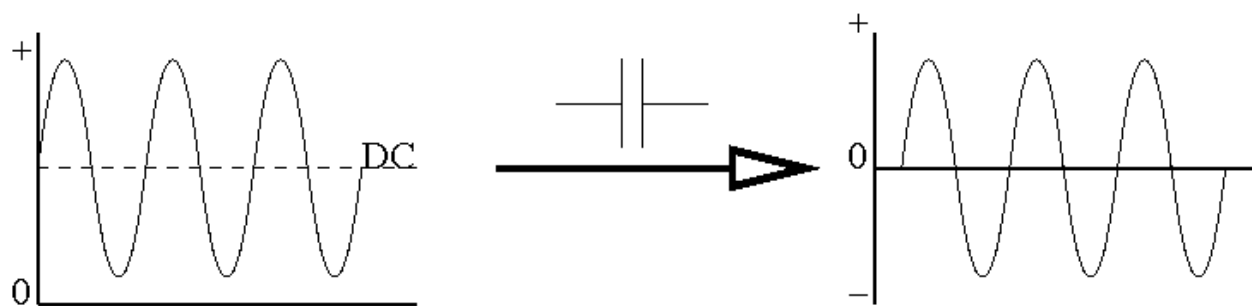


Figure 17. AC Capacitive Coupling in an O.A.

The value of these capacitors determine the minimum frequency that can amplify our circuit, thus acting as a high pass filter. If they have very little capacity, low frequencies are strongly attenuated. If they are too large you can have significant losses and we do not want that. It is often used a value between 100nF and 10μF.

### WHAT IS AN INVERTER AMPLIFIER?

Before explaining what an inverter O.A. is. I am going to show you some equations about the differential amplifier that will help you to understand how an inverter amplifier works.

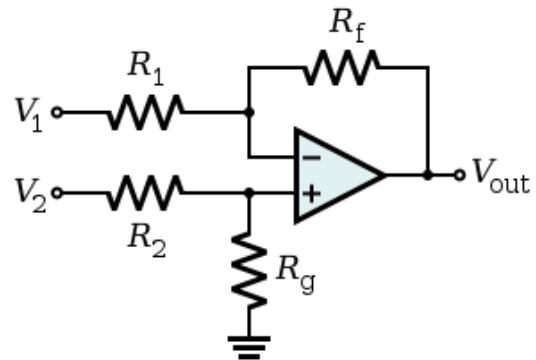


Figure 18. Schematic of a Differential Amplifier

The circuit shown before computes the difference of two voltages, multiplied by some gain factor. The output voltage:

$$V_{out} = \frac{(R_f + R_1) R_g}{(R_g + R_2) R_1} V_2 - \frac{R_f}{R_1} V_1 = \left( \frac{R_1 + R_f}{R_1} \right) \cdot \left( \frac{R_g}{R_g + R_2} \right) V_2 - \frac{R_f}{R_1} V_1.$$

Or, expressed as a function of the common mode input  $V_{com}$  and difference input  $V_{dif}$

$$V_{com} = (V_1 + V_2)/2; V_{dif} = V_2 - V_1,$$

The output voltage is:

$$V_{out} \frac{R_1}{R_f} = V_{com} \frac{R_1/R_f - R_2/R_g}{1 + R_2/R_g} + V_{dif} \frac{1 + (R_2/R_g + R_1/R_f)/2}{1 + R_2/R_g}.$$

In order for this circuit to produce a signal proportional to the voltage difference of the input terminals, the coefficient of the  $V_{com}$  term (the common-mode gain) must be zero, or

$$R_1/R_f = R_2/R_g$$

With this constraint in place, the common-mode rejection ratio of this circuit is infinitely large and the output is:

$$V_{out} = \frac{R_f}{R_1} V_{dif} = \frac{R_f}{R_1} (V_2 - V_1).$$

\*The simple expression  $R_f / R_1$  represents the closed-loop gain of the differential amplifier.

The special case when the closed-loop gain is unity is a differential follower, with:

$$V_{out} = V_2 - V_1.$$

On the other hand the inverter amplifier is the most used to connect a micro electret and it is very easy to build. It is so named because the output signal is inverse to the input, in polarity, but could be higher, equal or lower depending on the gain we give the amplifier in closed loop. The signal, as shown in the figure, is applied to the inverter or negative terminal of the amplifier and the positive or non-inverting is connected to masa. The resistance  $R_f$ , which runs from the outlet to the negative input terminal is called feedback.

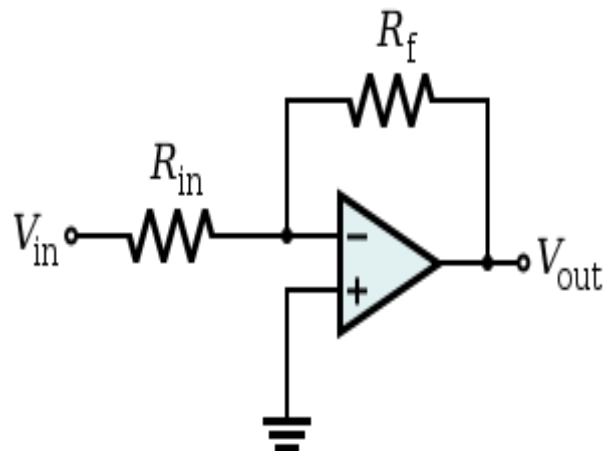


Figure 19. Schematic of an Inverter Operational Amplifier

An inverting amplifier is a special case of the differential amplifier in which that circuit's non-inverting input  $V_2$  is grounded, and the signal is applied in the inverting input  $V_1$ , identified in this case, with  $V_{in}$  in the last picture. The closed-loop gain is  $R_f / R_{in}$ , hence  $V_{out}$  is:

$$V_{out} = -\frac{R_f}{R_{in}} V_{in}$$

The simplified circuit above is like the differential amplifier in the limit of  $R_2$  and  $R_g$  very small. In this case, though, the circuit will be susceptible to input bias current drift because of the mismatch between  $R_f$  and  $R_{in}$ .

To intuitively see the gain equation above, calculate the current in  $R_{in}$ :

$$i_{in} = \frac{V_{in}}{R_{in}}$$

\*recall that this same current must be passing through  $R_f$ , therefore (because  $V_- = V_+ = 0$ ):

$$V_{out} = -i_{in} R_f = -V_{in} \frac{R_f}{R_{in}}$$

A mechanical analogy is a seesaw, with the  $V_-$  node (between  $R_{in}$  and  $R_f$ ) as the fulcrum, at ground potential.  $V_{in}$  is at a length  $R_{in}$  from the fulcrum;  $V_{out}$  is at a length  $R_f$ . When  $V_{in}$  descends "below ground", the output  $V_{out}$  rises proportionately to balance the seesaw, and *vice versa*.

If we want a very high gain we have two options:

1. **Turn up the gain of the stage.** It means decreasing  $R_{in}$  and increasing  $R_f$  as much as we need. It's very simple to do but the disadvantages are numerous: reduced input impedance, increased electronic noise (grows with the gain), decreased bandwidth and may appear autoswaying.
2. **Add another stage.** We can get a high gain using two or more cascaded stages. We got rid of the above disadvantages but instead we have, an increased consumption, a more complicated schematic and the difficulty to adjust each stage to not saturate to the next.

It's recommend using a single operational amplifier for minor gains  $\times 20$  and two or more stages over  $\times 20$  gain.

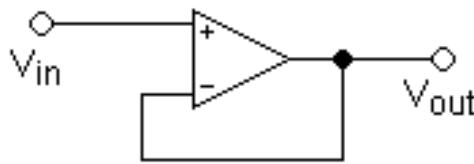


Figure 20. Buffer's Schematic

In some cases a buffer that is an  $\times 1$  gain amplifier that means that nothing is amplified. Its mission is to adapt the impedance, owing to it presents a high impedance input, useful for taking the microphone's signal; and a low output impedance, which is applicable to the following stages.

## FREQUENCY RESPONSE

In the frequency band in where we can use the amplifier it is important to keep three factors in mind:

- **Resistor R1.** Which determines the input impedance.
- **Capacitor C1.** That along with the input impedance forms a high-pass filter, cutting the DC component, but also the frequencies below the cut-off frequency.
- **The slew-rate of the integrated.** The operational amplifiers have an internal compensation to prevent oscillate spontaneously when they are working with high gain. This limitation restricts the speed with can vary the output voltage, and thus imposes a maximum frequency. This will depend on the gain and amplitude of the input signal.

So we have a high-pass filter (first order) at the entrance and one low pass to the output. Let's take the following circuit and simulate to obtain a bode diagram.

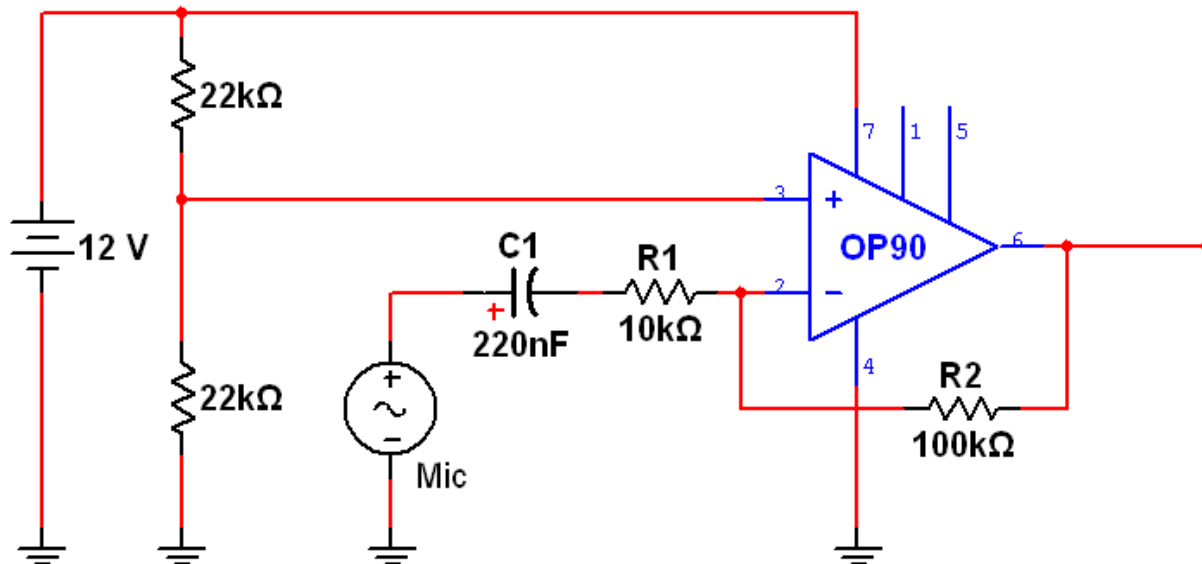


Figure 21. Schematic Circuit for Bode Diagram

### WHAT IS A BODE DIAGRAM?

A Bode diagram is a graphical representation used to characterize the frequency response of a system. Normally consists of two separate graphs, one corresponding to the magnitude of said function and other to the matching phase. Named after the American scientist who developed it, Hendrik Wade Bode.

It is very used in the analysis of electronic circuits, being fundamental to the design and analysis for filters and amplifiers.

The Bode magnitude plot module draws the transfer function (gain) in decibels as a function of frequency (or angular frequency) in logarithmic scale. Is often used in signal processing to show the frequency response of a linear, time invariant system.

Being the next values:

$R1 = 10K$      $R2 = 100K$      $C1 = 220nF$      $IC = OP90$



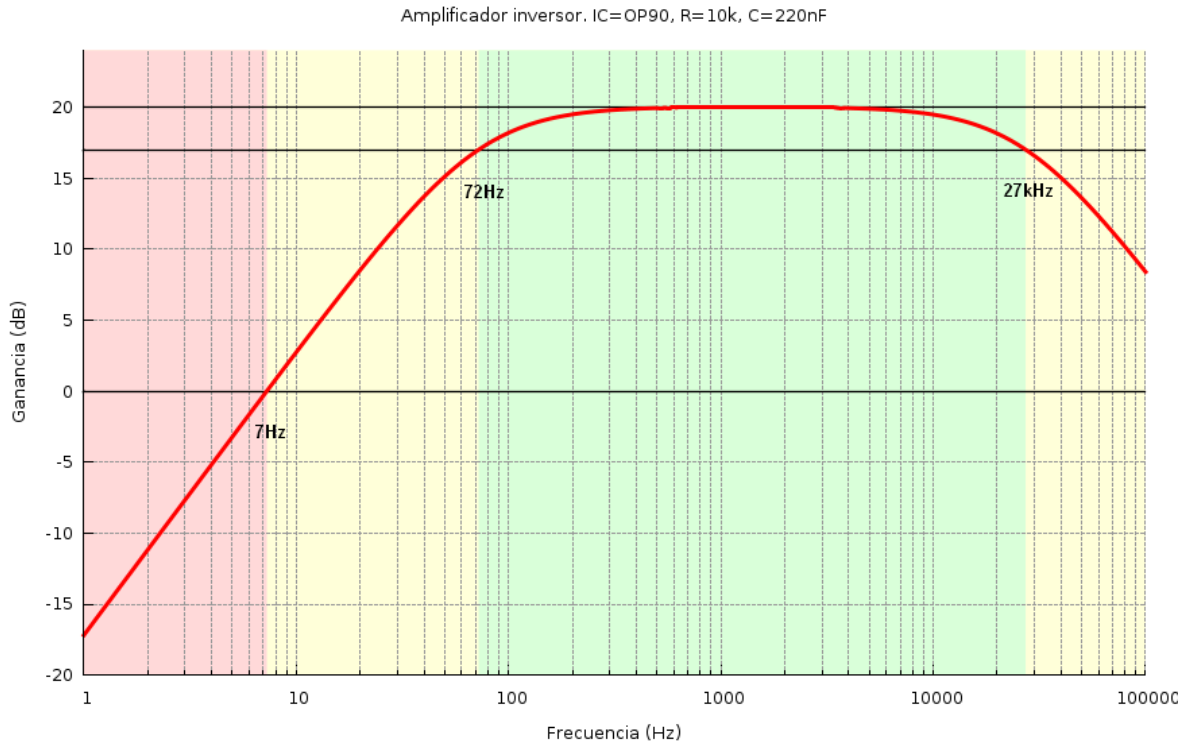


Figure 22. Bode Diagram

The graph is divided into three colors. The green area is the gain  $\times 10$  (or 20 dB),  $R_2 / R_1$ . To the left is the cutoff frequency of the filter  $C_1 / R_1$ . This begins when the gain is already 3dB lower than it was expected, in this graph is 72Hz, yellow zone. From there begins a downward slope of -20dB per decade till reach the red zone that begins at 7.2Hz. Here not only there is no amplification, in addition the circuit attenuates the lower frequencies. On the right side the high frequencies begin to decay to 27 kHz, higher switching frequency, yellow zone. It is enough if you have in mind that we do not hear tones above 20 kHz.

### 2.3.2 PRACTICAL CIRCUIT ON PROTOBOARD

Before building our *PCB* with *EAGLE*, we did several electric circuits in a protoboard to be sure of the properly functioning of our LED Matrix. We measured several times the data exit with an oscilloscope to see the difference between the normal microphone's signal and the amplification of it.

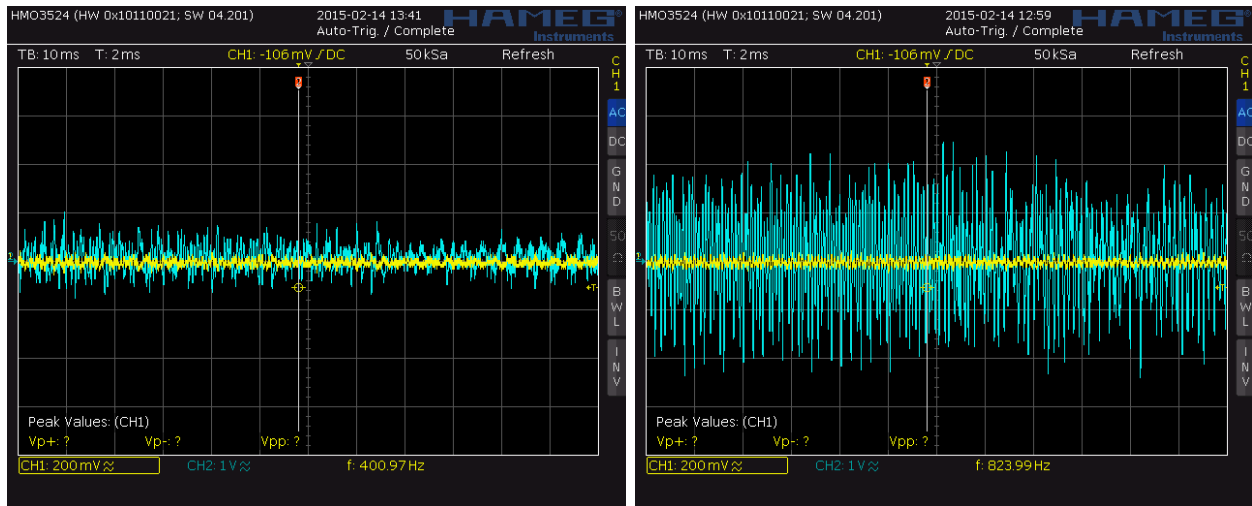


Figure 23: Microphone und amplifier signals

The blue line that we can see in the last pictures is the signal of the amplification and the yellow one, the signal of the microphone. In the picture on the left, the potentiometer was in the lowest point of amplification and in the picture on the right, the potentiometer was the highest point and we get the maximum amplification. In the blue line of the right picture we can see that the range peak to peak is 5V. As the voltage supply is 5V and we have, more or less, the same value in range peak to peak, we can say that we have a good amplification.

We tried with different resistors, capacitors and operational amplifiers and we obtained the results of the last pictures with the next circuit. R3 and R4 are our artificial ground, which is half of the supply voltage. R5 is the polarization resistance of the electret microphone while R1 and C1 form a filter to remove the DC component from the microphone. R2 and R6 are the feedback resistor that determines the gain by R1.

I have colored the positive voltage in red, negative voltage in black, blue it would be the artificial ground and green the signal path.

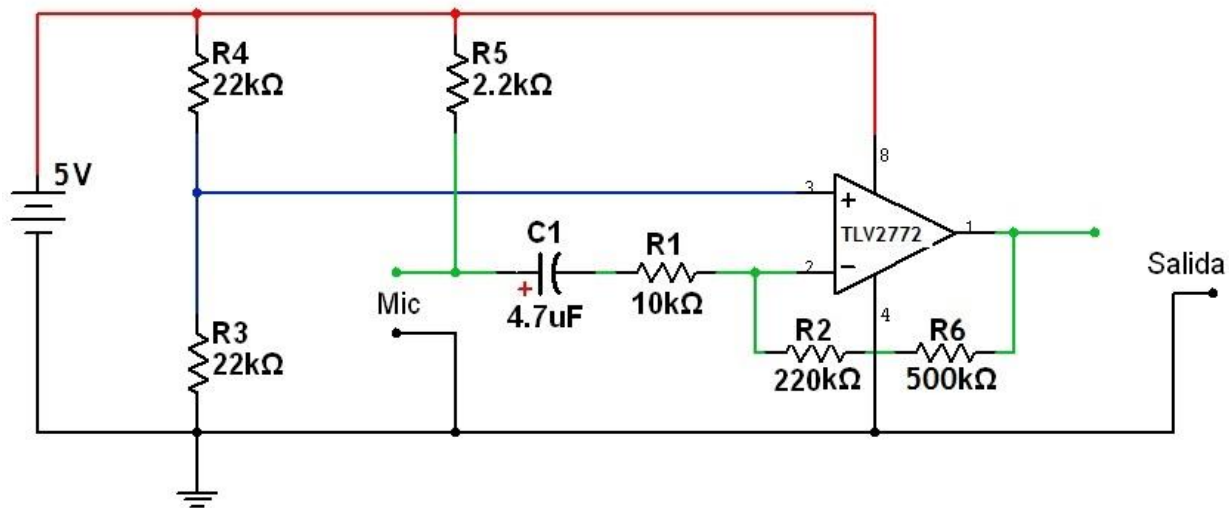


Figure 24. Schematic of our electric circuit

If you need more amplification the best option is chaining another stage just below. Another option is to change the value of R2 to be worth 50 or 100 times R1.

In our case what we did to have more amplification was to add a 500K potentiometer (R6) after R2. Like this we can regulate the gain turning only the potentiometer to the right or to the left. Now we calculate the gain of our circuit:

Gain:

$$A = \frac{R2 + R3}{R1}$$

$$A_{min.} = \frac{R2 + 0}{R1} = \frac{220K}{10K} = 22K$$

$$A_{max.} = \frac{R2 + R3}{R1} = \frac{220K + 500K}{10K} = 72K$$

In conclusion we have a gain  $\times 22$  when the potentiometer works with the minimum value. When the potentiometer works with the maximum value (500K) the gain is  $\times 72$ . Like this we can play with a gain until we find the correct one.

Whenever you use an operational amplifier you must pay attention in:

- **The supply voltage.** With dual sources no problem, but when using simple sources for operational amplifier remember that it is as if the voltage is divided in half and can not reach the minimum voltage recommended by the manufacturer. In addition, with lower power we will obtain a lower output and the signal can be distort.
- **The noise factor.** Important if we want to capture faint sounds.
- **The passband.** There amplifiers that are slower than others but instead have other desirable properties, such as low voltage or noise. We need to reach a compromise between what we need on the one hand and on the other. The most common datasheets are seamlessly Internet

### 2.3.3 PCB DESIGN

As in the protoboard is possible to have loses because of the poor connections, we decided to design a PCB making our project nicer and more efficiently. A PCB is a printed circuit board mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from copper sheets laminated onto a non-conductive substrate. PCBs can be single sided (one copper layer), double sided (two copper layers) or multi-layer.

Conductors on different layers are connected with plated through holes called vias. Advanced PCBs may contain components capacitors, resistors or active devices embedded in the substrate. The Software used for the PCB design was EAGLE.

## WHAT IS EAGLE?

EAGLE (Easily Applicable Graphical Layout Editor) by CadSoft Computer is a flexible, expandable and scriptable EDA application with schematic capture editor, PCB layout editor, autorouter and CAM and BOM tools developed by CadSoft Computer GmbH, Germany, since 1988. Famous worldwide for the design of electronic projects DiY, because many versions of this program have a Freeware license and lots of component libraries around the net. EAGLE contains an electronic diagrams editor where components can be placed in the diagram with a single click and easily routable with other components based on "cables" or labels. Also EAGLE contains a PCB editor with a rather efficient autorouter. The editor is able to produce GERBER files and others, which are used at the time of production. Eagle brings component libraries included, easy to make and available from companies such as SparkFun or fans that spread them around the net for free.



## STARTING WITH EAGLE

In the next picture we can see EAGLE's panel control. To start with a new project we have to click on "file" → "new" → "project". After that we will have to write a name for our new project.

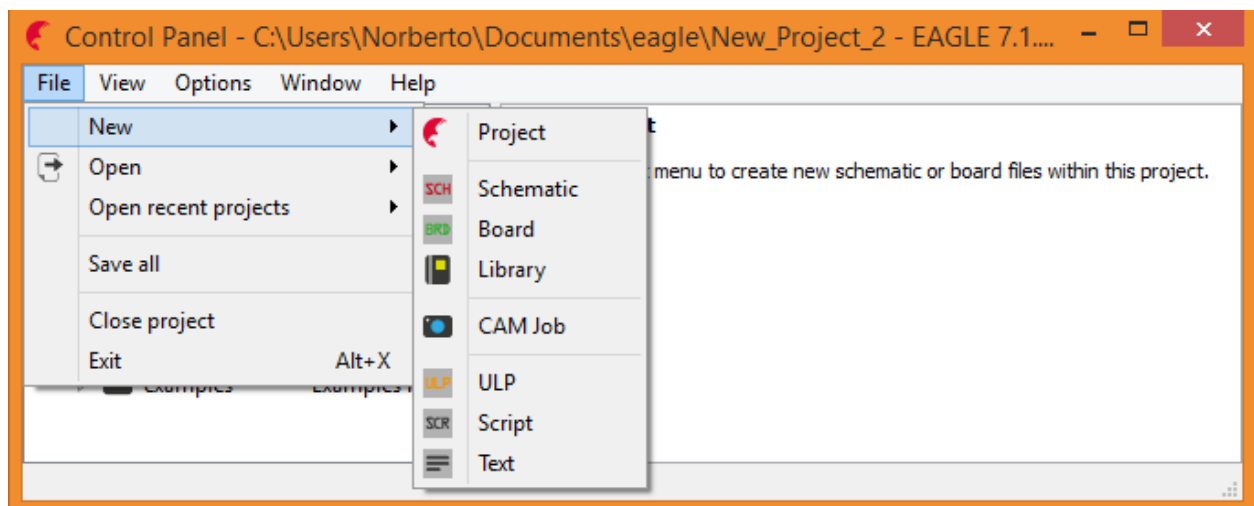


Figure 25. Control Panel of EAGLE Software

Once we have created the file for our new Project we must do the schematic design of the electronic circuit that we did in the protoboard. For it we have to click on the file of our project with the right button and click on “New” → “Schematic”. Then it is time to create the schematic of the circuit using EAGLE’s toolbar. In the next picture you will see the Schematic of my PCB.

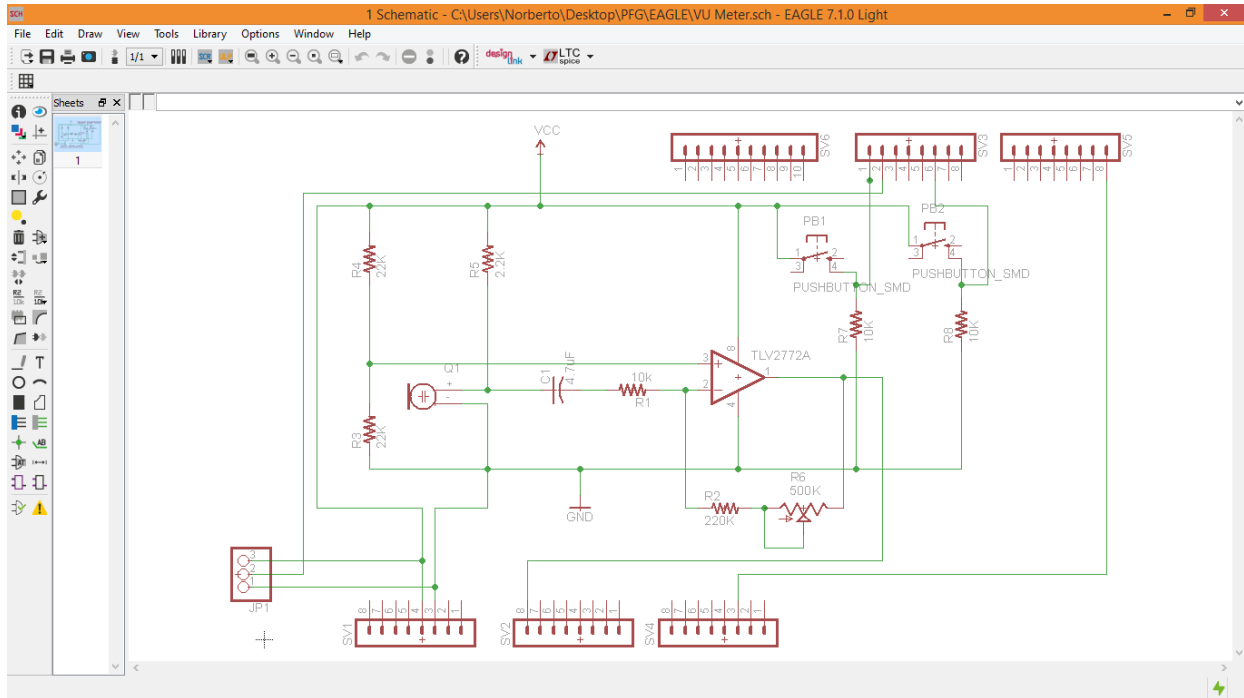



Figure 26. Schematic of the electronic board

The next step is to design the physical appearance of our electronic board. So, when we finish and save our Schematic, in the Schematic window we have to click on  and a different window will appear to design the board. As you will see in the next picture, you will have to order and connect all the elements of the electronic circuit on the real delimited board space. Sometimes this operation can be difficult.

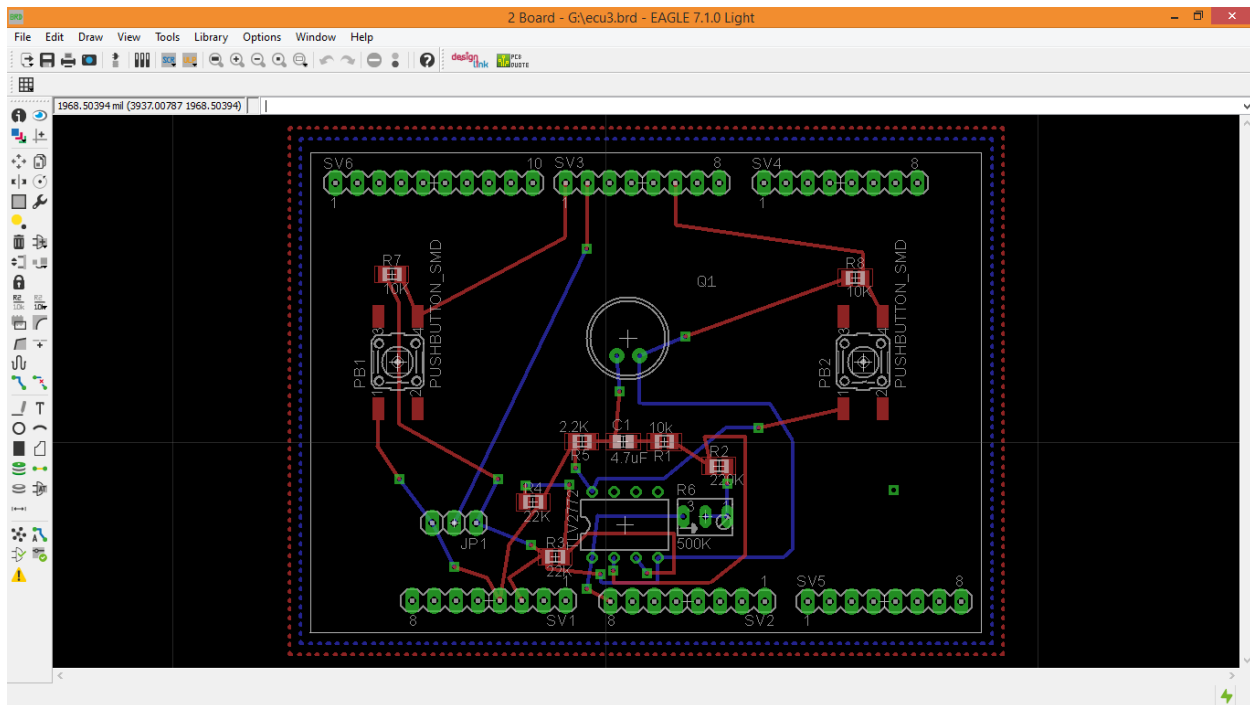


Figure 27. Layout of the electronic board

## PCB PRINCIPAL COMPONENTS

With the board physically printed over our table, we started soldering our smallest components with a microscope where we used compressed stain liquid. The biggest components we sold them with a normal electronic solder and a small bobbin of stain.

The components that we sold to the board were:



7X SMD Resistor



1X Operational Amplifier



1X SMD Capacitor 1X



1X Potentiometer



2X SMD Button



1X Electret microphone

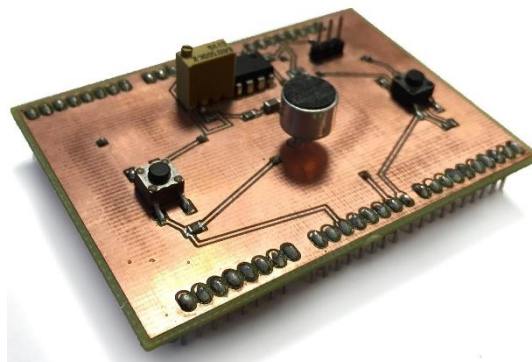


Figure 28: PCB with elements soldered



## PROBLEMS

Designing our board we noticed that we had to use the two faces on it. That means that we had to use some “vias” to connect the electric lines of the different sides.

Other thing that was a bit complicate was to put the pins of this board in the correct position not to have problems after with the connection pins of Arduino. To solve this problem we printed in paper the design of the board, we put the paper over the Arduino’s board and we probed that the holes that were drew in the paper for our future electronic board ran into the Arduino’s board. Also, as we have to sold very small components we made thicker the width of the paths.

The first time we connect together both boards something strange happened. Button 1 and 2 worked properly but meanwhile were not pushed, the equalizer did not worked properly. We measured the current and the voltage in different parts of the circuit and we checked the signal in the oscilloscope until we found the problem. One side of the capacitor was not well solded and because of that, the AC capacitive coupling didn’t work.

## IMPROVEMENTS

As our board are going to be together inside the matrix’s box. We had to do some modifications for the buttons and the microphone of the board. We design a witty connection to keep some liberty between the board, buttons and microphone. We evolved the solded points of our witty connections with pieces of thermoretractable plastic tube not to make a short circuit and for the safety of everyone.



Figure 29: Safety Connections for the PCB

# CHAPTER 3

## 3D CAD DESIGN WITH SOLIDWORKS

---

In order to compact this project and to make it look prettier, a 3D CAD design has been built. This 3D CAD design will be then printed by a 3D printed. In this chapter are introduced different step followed to design the model as well as the relevant technical data. For this project, the chosen program was SolidWorks.

### 3.1 WHAT IS SOLIDWORKS?

SolidWorks is a CAD Software (computer aided design) for mechanical 3D modelling, now developed by SolidWorks Corp., a subsidiary of Dassault Systèmes SA (Suresnes, France) for the Microsoft Windows operating system. Its first version was launched in 1995 with the purpose of making CAD technology more accessible.



The program allows modelling pieces and assemblies and obtain technical drawings and other information necessary for production. It is a program that works based on new modelling techniques with CAD systems. The process is to transfer the mental idea of the designer to the CAD system, "building virtually" the piece or assembly. Than all extractions (plans and swap files) are made automatically.

### 3.2 3D CAD LED MATRIX BOX DESIGNS

If it is the first time you are going to use SolidWorks, you can find very good tutorials of how to use it, in YouTube. As no one taught me how to use this software, all I know about it, I learnt it watching the tutorials in YouTube and was enough to do my 3D design.

Remember that this design has to contain the following devices:

- LED Matrix
- Arduino Due
- PCB
- 2x Additional Buttons
- 1x Microphone

Therefore the basic parameters of the design are the following:

- **Case Size:** 182x70x200mm. We designed a box with a bigger size than the LED matrix to have enough space to put inside the LED matrix and the holes for the buttons and the microphone on the front side. The cables, PCB and Arduino are hidden in the back side.
- **Holes:** We made three holes, two for the mode buttons ( $r=4.5\text{mm}$ ) and one for the microphone (5mm). Also we made a 3x3 round holes matrix in the base where the LED matrix will be support to save some plastic material of the 3D printer.
- **Inclination:** We inclined  $15^\circ$  the front side to have better view of the LED lights.

To create a new project in SolidWorks open the program and click in New Document. After that a new window will appear on our screen and we will have to choose one of the three options. To make a simple 3D design we chose the first option that appears on the top “3D representation of a single design component”.

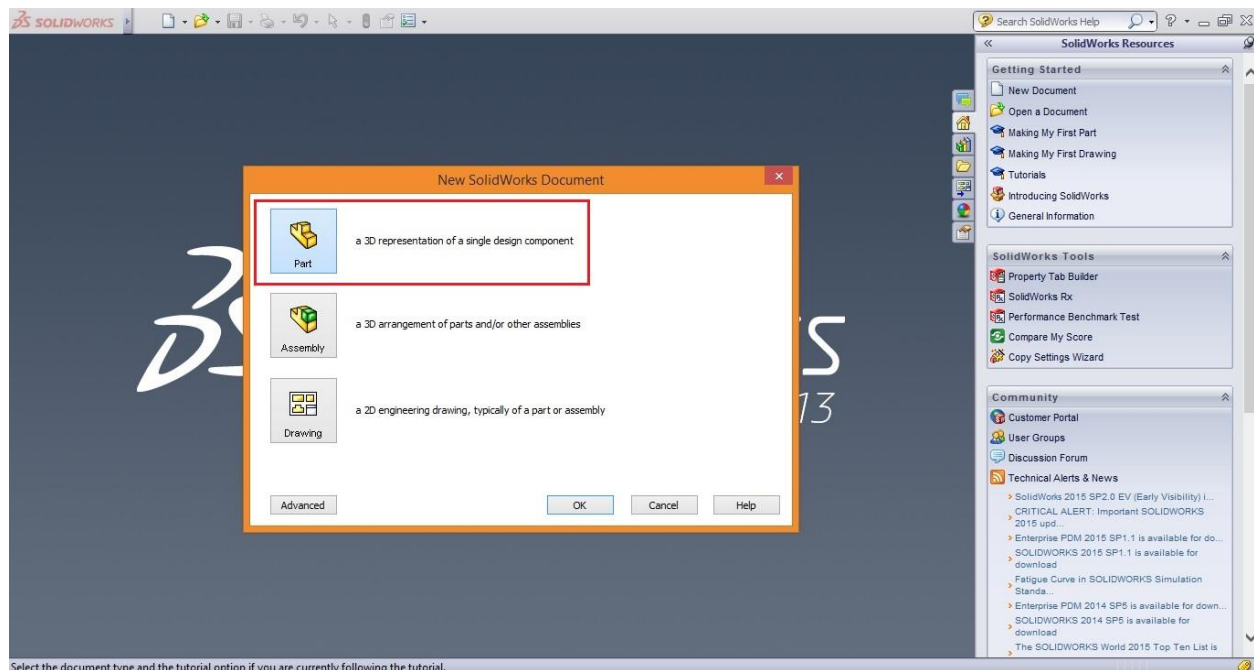


Figure 30. SolidWorks Software

## FIRST DESIGN:

In this design we did it very simple. We created a sketch with a square of 182x182mm. Then we extruded the square 100mm to form a kind of cube. Later we shelled the back side of the cube and on the front side made two steps, one for the base of the LED matrix and other for the glass that covers the LED matrix. To finish we made the holes for Arduino connections, buttons and microphone. In the picture we can see the result.

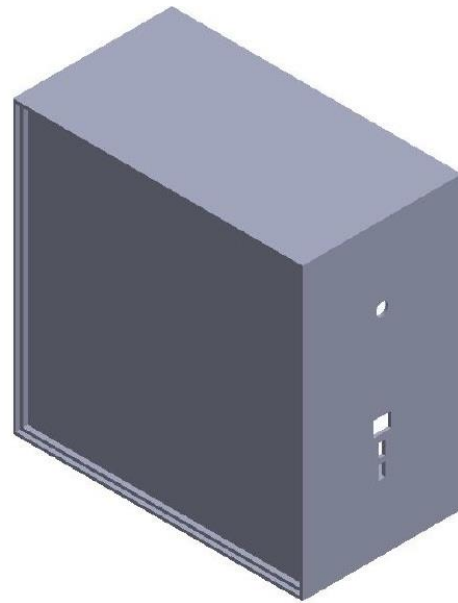


Figure 31. First 3D LED matrix case design

## SECOND DESIGN:

In our second design we followed more or less the same steps that in the first design. We created a sketch with a rectangle 182x200mm. Then we extruded the rectangle 70.3mm to form a kind of cube and we shelled the back side of the cube. The different in this design is that we inclined a little bit the LED matrix base and the buttons and microphone are situated in the front side. We thought that with this small inclination the visual effect of the LED Matrix will be nicer and the microphone in the front side would work properly.

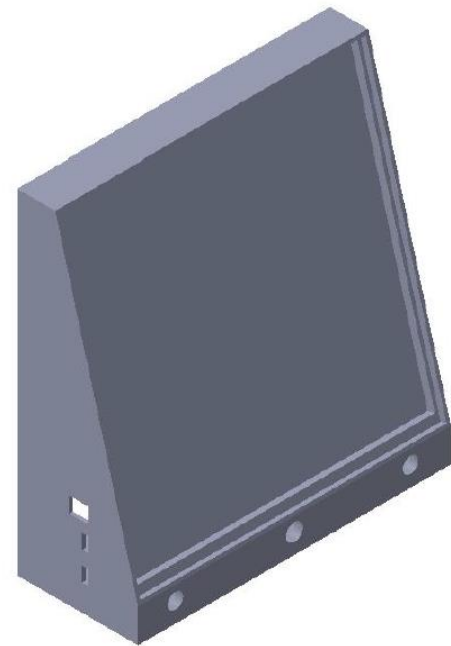


Figure 32. Second 3D LED matrix case design

### THIRD DESIGN

In our final design we did a small improvement. The design is equal as the second one but we made a round holes matrix to save some material of the 3D printer.

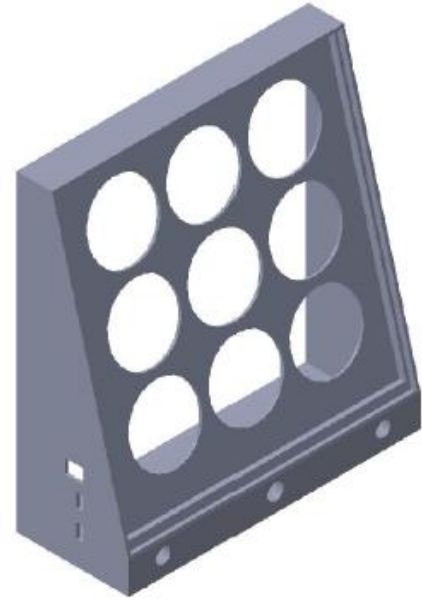


Figure 33. Final 3D LED Matrix Box Design

# CHAPTER 4

## 3D PRINTER

---

The printer used was the X400 from German RepRap. This is a company which produces big and high quality in Germany. Philosophy of RepRap is open source and open hardware, so everybody is able to build an own printer. What makes the different between the X400 and the others low cost printer is the size of them, almost double than biggest printers in market.

Technical Specifications:

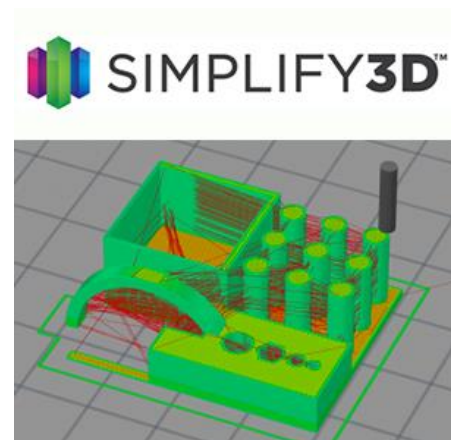
- Build Volume: 400 x 400 x 350 mm
- Overall size: 700 x 770 x 700 mm
- Print Volume: 56 l
- Layer thickness: min 0.1mm
- Weight: 65 kg
- Material: PLA, ABS, PP, PVA
- nozzle size: 0.3, 0.4, 0.5 mm
- Accuracy: 0.1mm
- Confection material: 3mm
- Power Consumption: 50W
- Extruder: Dual Extruder
- Extruder Temperature: max 275°C



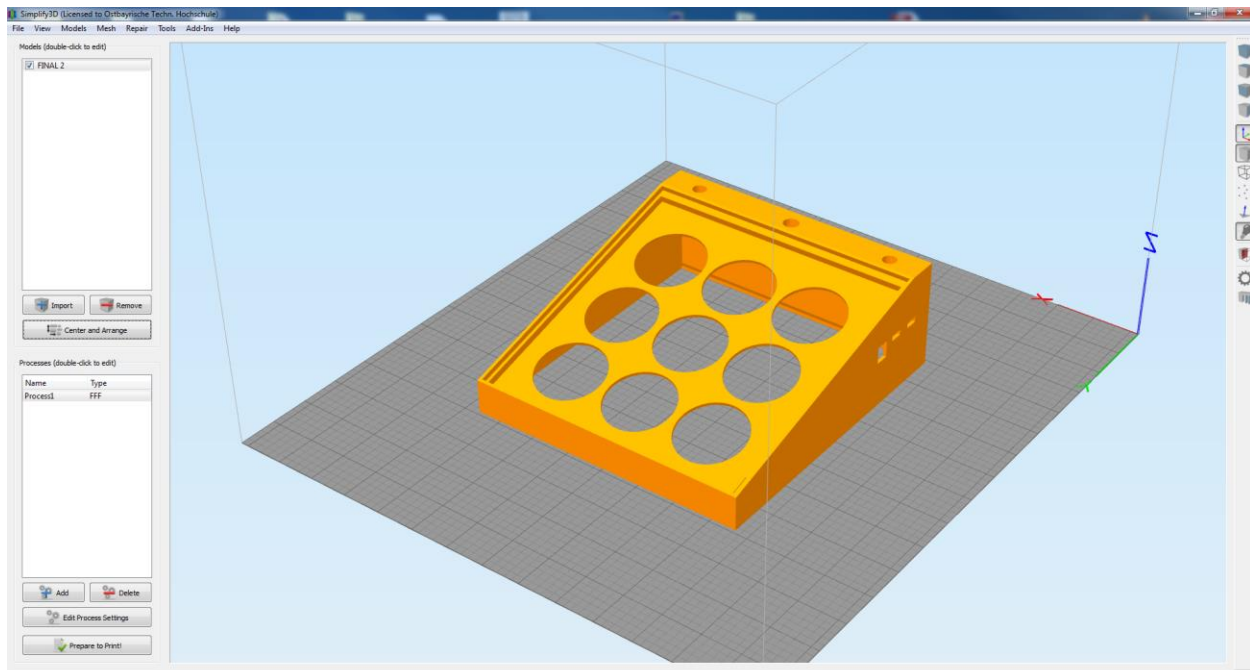
Figure 34. X400 3D Printer

## 4.1 PRINTER SOFTWARE

In order to control the printer and communicate with it per serial port, the used software was Simplify 3D. There are others software and many of them are free, but as Simplify3D is software specific for German RepRap and OTH Regensburg bought the licenses; this was the proper software to use.







**Figure 35.** Simplify 3D Software

## 4.2 PRINTER SLICER

The Slicer is the software in charge of translating the CAD (STL) design into the machine code (Gcode). There are many different kind of slicer, like Skinforge or Slic3r among others. For this particular case has been used Slic3r, as it has been proof that it is the better for this purpose because of its easy usage and quality of Gcode parameters generation.

Main parameters of Slic3r:

- Extruder Temperature: 200°C for PLA
- Bed: Temperature: 70°C for a better adherence of the printed part
- Layer height: 0.25 mm for a good ratio between speed and quality.
- Infill: 0.25 to spare material
- Brim: 3mm to provide better adherence.
- Support material:
  - Pattern: Rectilinear
  - Pattern Spacing: 2 mm
  - Overhang support: 45°
  - Interface layers: 1

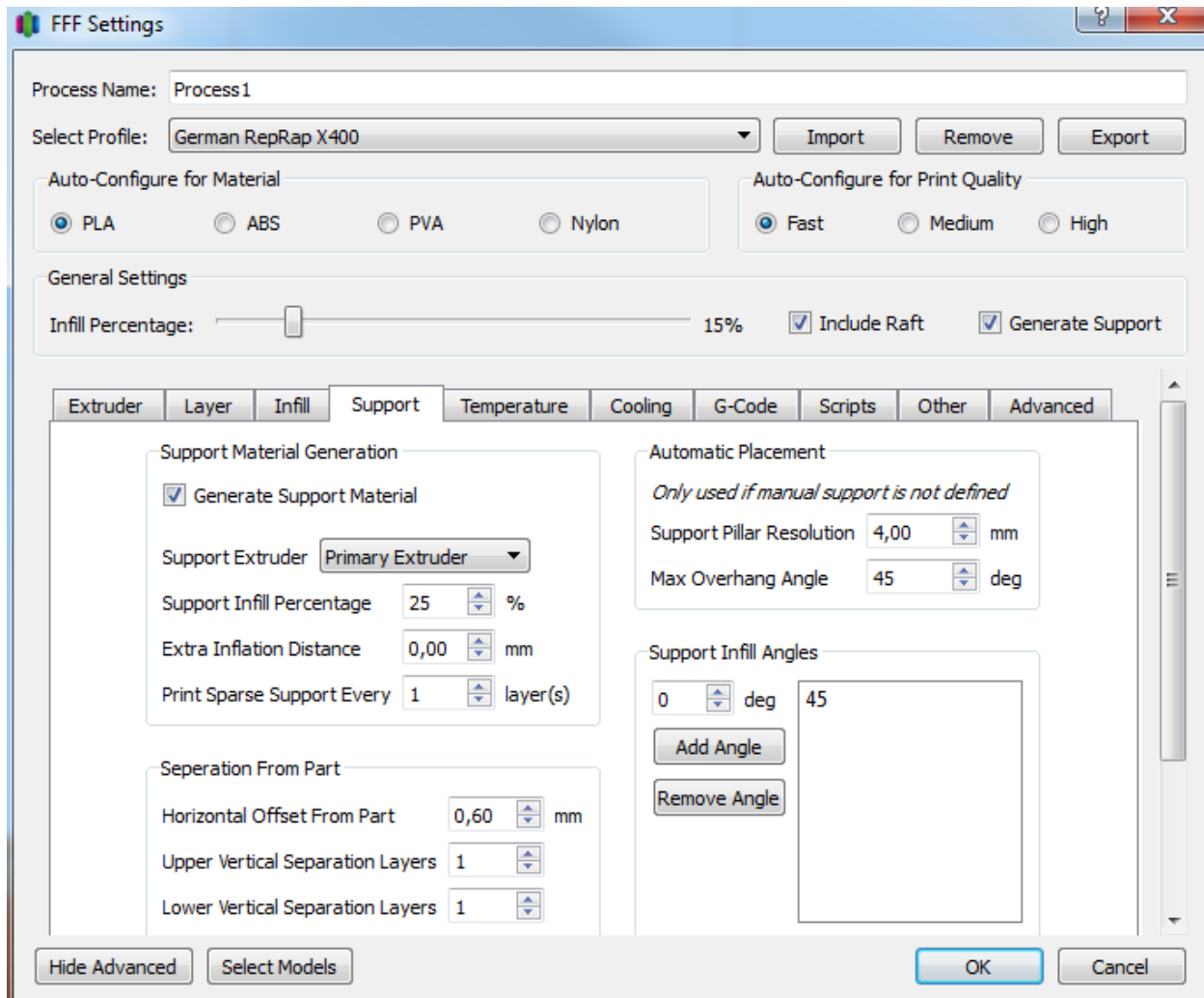


Figure 36. 3D Printer Parameters

## 4.3 RESULTS

- **First trial:** Huge print requires lot of time. The energy supply failed on the extruder at 50% of printing, then cold extrusion prevent made unable to keep printing.
- **Second trial:** As supply energy failed, recalibration of extruders after reparation. The calibration was wrong breaking adherence sheet of the plate. Reparation took 2 days.

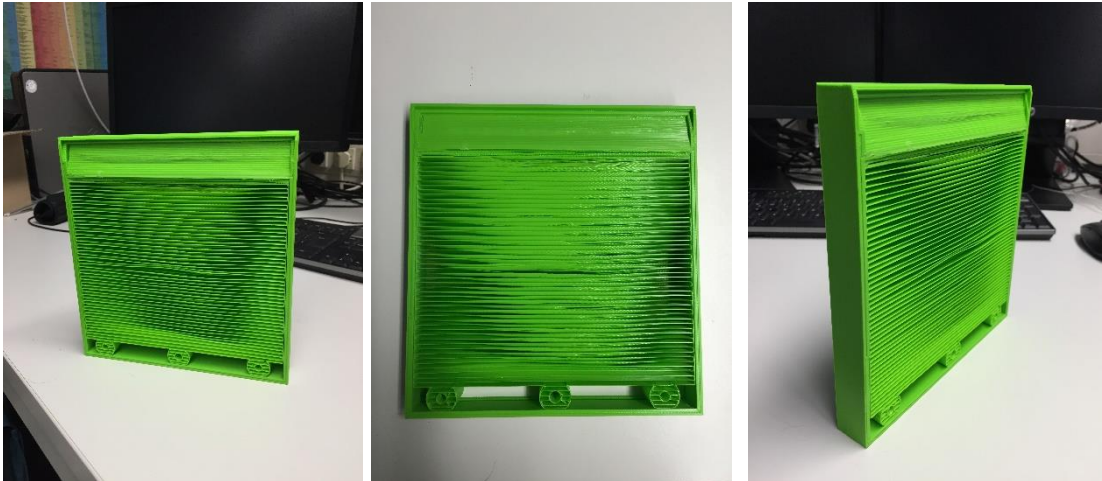
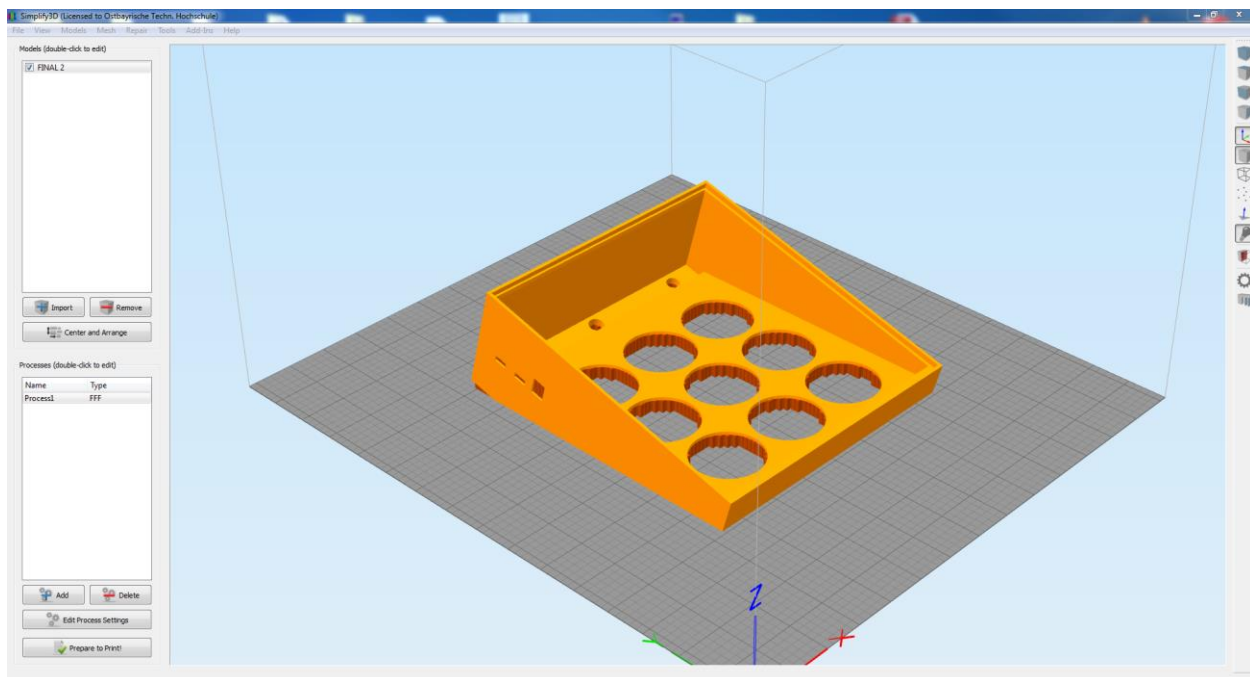


Figure 37. Failed Printout

- **Third trial:** With one of the failed print I notice that the printer needed to create a big support base to make the top of the box and the measures were a bit smaller than in the software. Because of that I decide to crate the third design where I modified some parameters, I made the round holes and I put down the front side to save material.



With the modifications made we reduce the printing time from 42h to 18h and at the end we success.

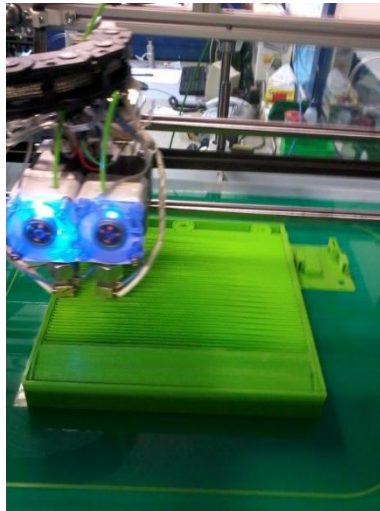
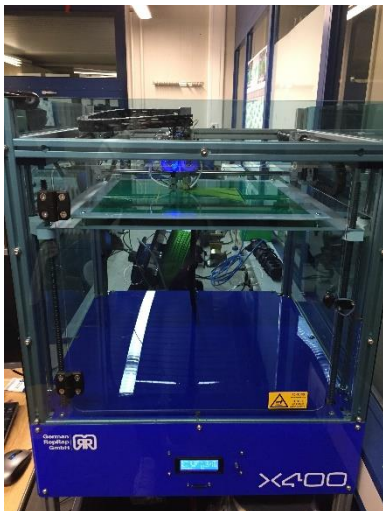
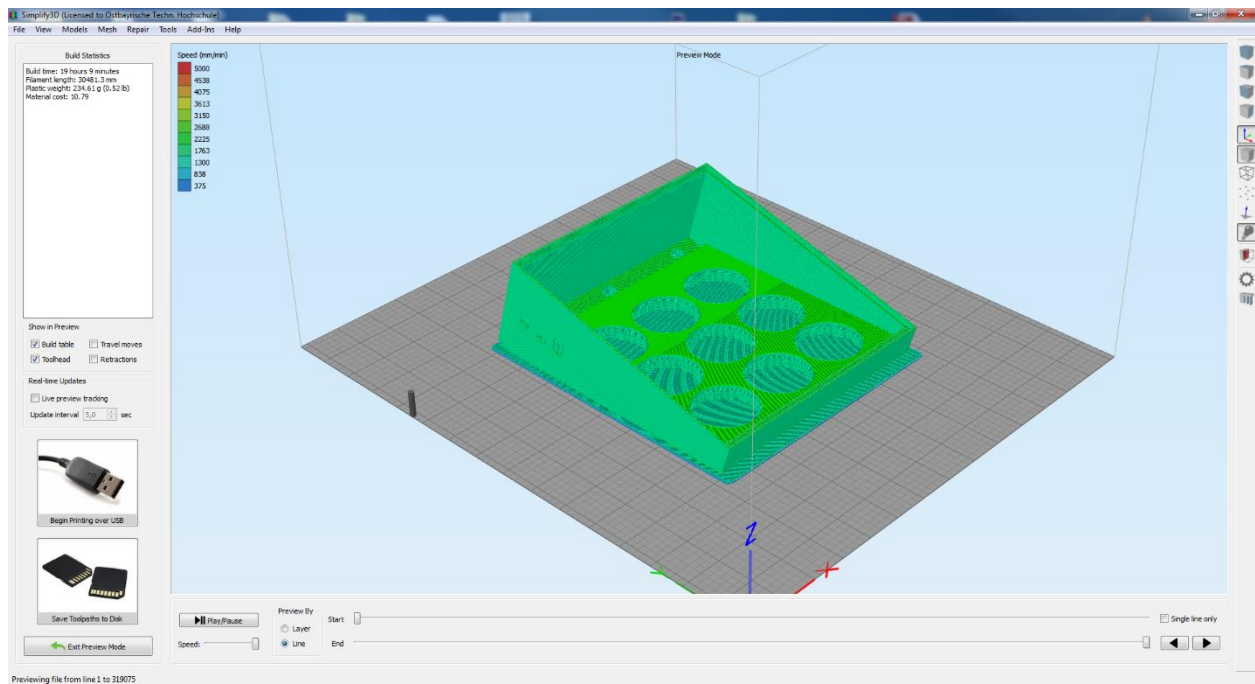


Figure 38. Third Trial Results

# CHAPTER 5

## APPLICATION

---



Building our project we realised that we can use the LED Matrix for different things. Depending on the programming we can do:

- Night lamp
- KID lamp
- Decoration lamp
- VU meter
- Digital Clock
- LED animations
- Equalizer

## 5.1 APPLICATION MODES

In our Arduino Software we created a basic program with different functions. With two buttons and a microphone we have created three different modes.

- **Mode 1:** When the buttons are not pushed, in the LED Matrix appear an animated digital VU meter with three different colours that indicates the intensity of the volume.
  - Green: low
  - Yellow: medium
  - Red: high
- **Mode 2:** It's activated when green button is pushed. This mode is interrupt Mode 1 and red heart appears in the LED Matrix for 2 seconds.
- **Mode 3:** It's activated when the orange button is pushed. This mode interrupt Mode 1 and a yellow flash appear in the LED Matrix for 2 seconds.



Figure 40: Application Modes

## 5.2 PROGRAMMING USER FUNCTIONS

**Function:** Adafruit\_NeoPixel

**Description:**

We use this function to declare a NeoPixel object. We will refer to “Adafruit\_NeoPixel strip” to control the strip or matrix of pixels.

**Syntax:** Adafruit\_NeoPixel strip = Adafruit\_NeoPixel (PIXEL\_COUNT, PIXEL\_PIN, NEO\_GRB + NEO\_KHZ800)

**Parameters:**

PIXEL\_COUNT: number of pixels in the matrix.

PIXEL\_PIN: pin number to which the pixels matrix is connected.

NEO\_GRB+NEO\_KHZ800: A value indicating the type of NeoPixels that are connected.

**Return:**

None

**Function:** Serial.begin()

**Description:**

Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate. An optional second argument configures the data, parity, and stop bits. The default is 8 data bits, no parity, one stop bit.

**Syntax:**

Serial.begin(speed)

**Parameters:**

speed: data rate in bits per second for serial data transmission.

**Return:**

None

**Function:** pinMode()

**Description:**

Configures the specified pin to behave either as an input or an output.

**Syntax:**

pinMode(pin,mode)

**Parameters:**

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT, or INPUT\_PULLUP.

**Returns:**

None

**Function:** strip.begin()

**Description:**

The data pin is prepared for NeoPixel output

**Syntax:**

strip.begin()

**Parameters**

None

**Return:**

None

**Function:** strip.show()

**Description:**

Initialize all pixels to "off"

**Syntax:**

Strip.show()

**Parameters**

None

**Return:**

None



**Function attachInterrupt()****Description:**

Specifies a named Interrupt Service Routine (ISR) to call when an interrupt occurs. Replaces any previous function that was attached to the interrupt.

The Arduino Due board has powerful interrupt capabilities that allows you to attach an interrupt function on all available pins. You can directly specify the pin number in attachInterrupt().

**Syntax**

```
attachInterrupt(interrupt, ISR, mode)  
attachInterrupt(pin, ISR, mode)
```

**Parameters**

interrupt: The number of the interrupt(int)

pin: the pin number (Arduino Due only)

ISR: the ISR to call when the interrupt occurs; this function must take no parameters and returns nothing.

mode: defines when the interrupt should be triggered. Five constants are predefined as valid values:

LOW: to trigger the interrupt whenever the pin is low.

CHANGE: to trigger the interrupt whenever the the pin changes the value.

RISING: to trigger the interrupt when the pins goes from high to low

FALLING: when pin goes from high to low

HIGH: to trigger the interrupt whenever the pin is high

**Return:**

None

**Function:** millis()

**Description:**

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days.

**Syntax:**

unsigned long startMillis= millis()

**Parameters:**

None

**Return:**

Number of milliseconds since the program started (*unsigned long*)

**Function:** analogRead()

**Description:**

Reads the value from the specified analog pin. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

**Syntax:**

analogRead(pin)

**Parameters:**

pin: the number of the analog input pin to read from.

**Return:**

int (0 to 1023)

**Function:** print()**Description:**

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character.

**Syntax:**

```
Serial.print(val)  
Serial.print(val, format)
```

**Parameters:**

val: the value to print - any data type

format: specifies the number base (for integral data types) or number of decimal places (for floating point types)

**Return:**

size\_t (long): print() returns the number of bytes written, though reading that number is optional

**Function:** setBrightness()**Description:**

The overall brightness of all the LEDs can be adjusted using setBrightness().

**Syntax:**

```
strip.setBrightness(brightness value);
```

**Parameters:**

Brightness value: This takes a single argument, a number in the range 0 (off) to 255 (max brightness). For example, to set a strip to 1/4 brightness:

**Return:**

None

**Function:** setPixelColor**Description:**

Is the way to set the color of a pixel. The first argument – n in this example – is the pixel number along the strip, starting from 0 closest to the Arduino. If you have a strip of 30 pixels, they're numbered 0 through 29. It's a computer thing. You'll see various places in the code using a for loop, passing the loop counter variable as the pixel number to this function, to set the values of multiple pixels.

**Syntax:**

```
strip.setPixelColor(n, red, green, blue);
```

**Parameters:**

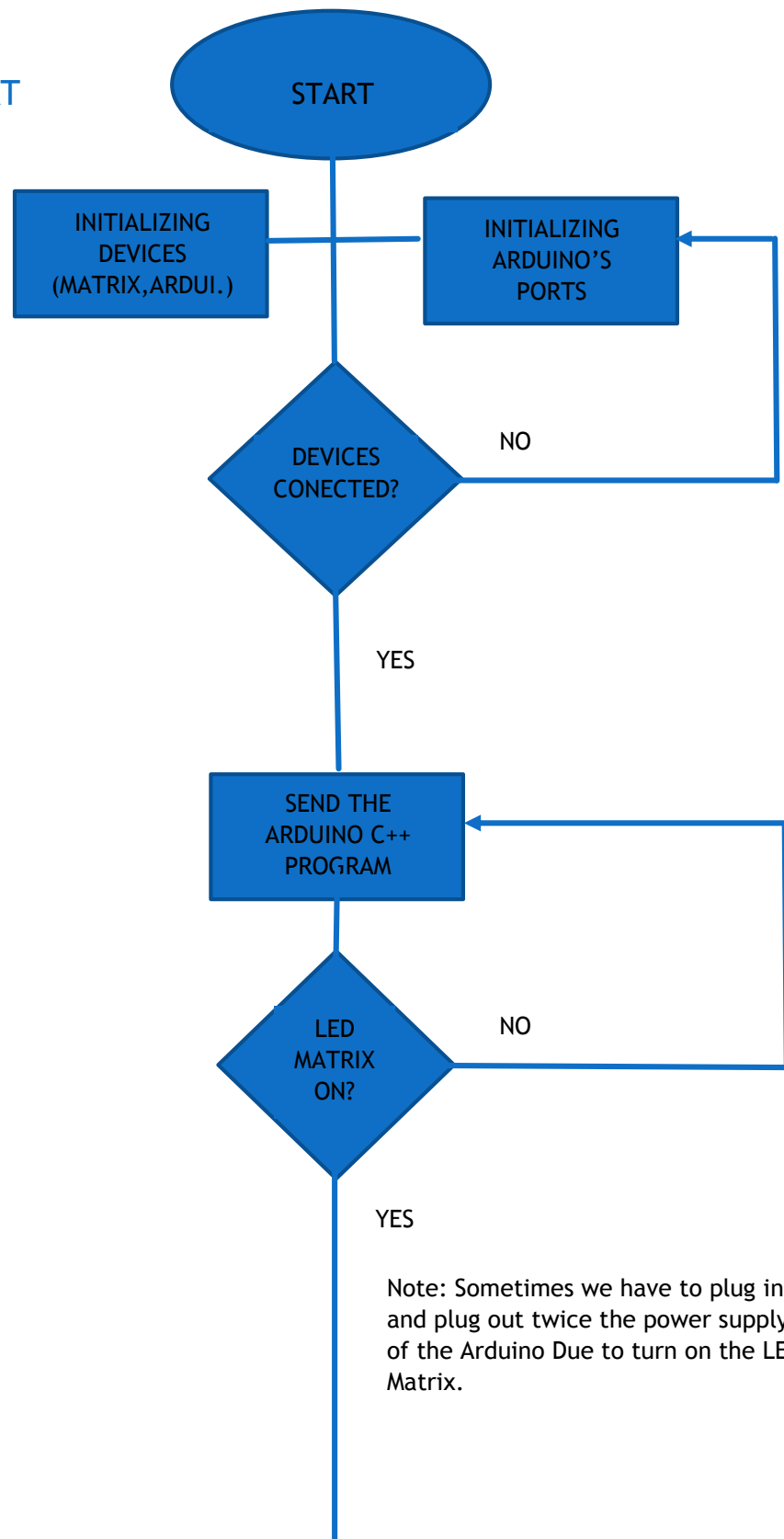
n: pixel number along the strip

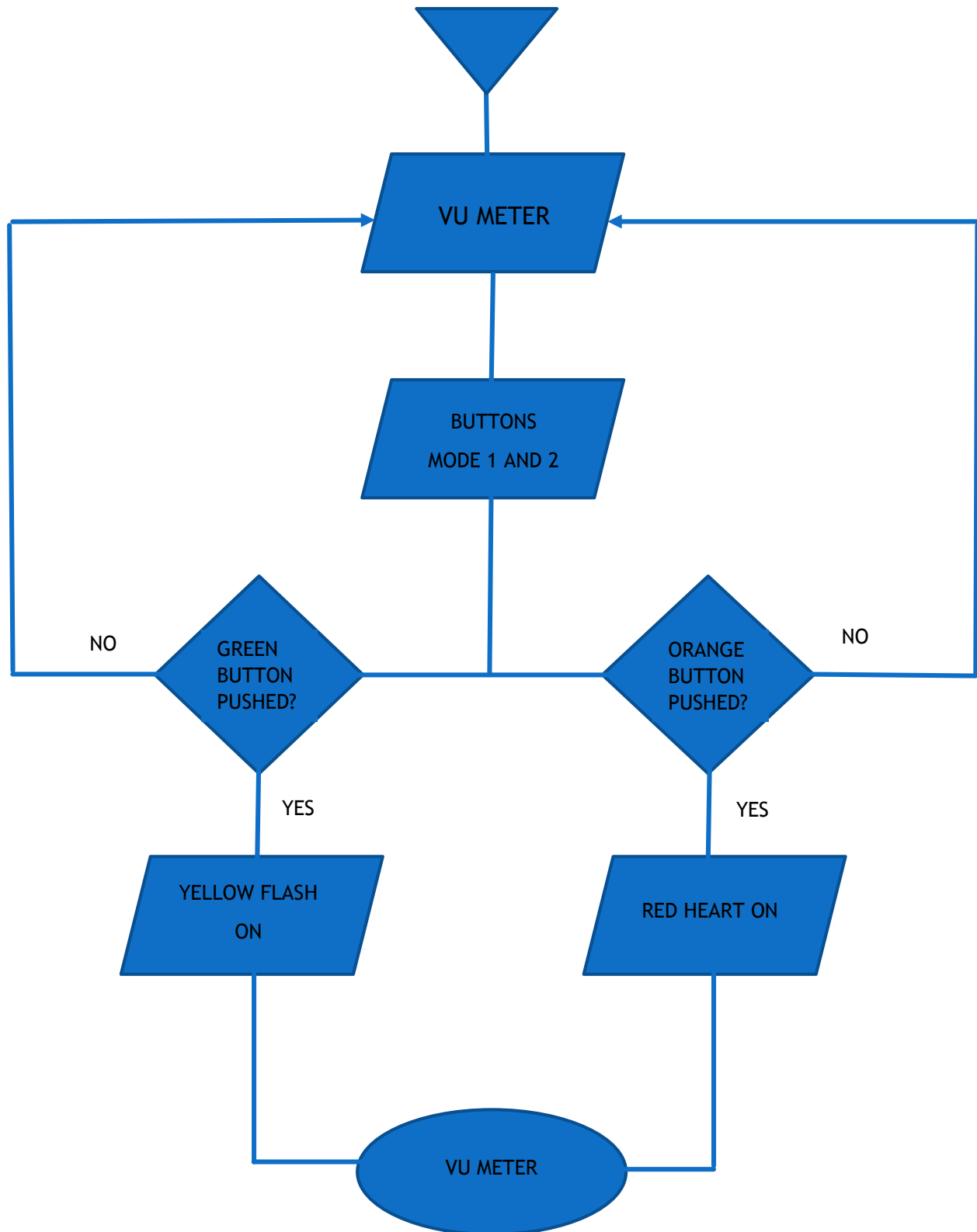
(red, green, blue): The three arguments are the pixel color, expressed as red, green and blue brightness levels, where 0 is dimmest (off) and 255 is maximum brightness.

**Return:**

None

## 5.2 FLOW CHART





# CHAPTER 6

## CONCLUSIONS AND VALUATIONS

---

## 6.1 CONCLUSION AND PROBLEMS

Now that our project is completed, is time to analyse carefully the problems we have found on the way and the conclusions over them.

As I am a Student of electrical engineering, during my studies I have done very few about programming and electronics. I decided to do a project associated with this topics and on the way, I found problems that I was able to solve with help of some colleagues, searching and analysing information. The most important is that I learnt things I didn't know yet.

The main idea of the project was to develop an application for a LED Matrix. At the end, I decided to do a lighting application with three different modes. The principal idea was to create a kind of lamp with a digital equalizer, and then with two additional buttons create other modes, a yellow flash for boys and the red heart for girls.

To start with it, first I had to see some tutorial and study a little bit about C++ Programming. Once done, I did some practices with simple codes and then I began with our application code. For it, also we had to check the Adafruit NeoPixel library that we had to use for the LED Matrix. We studied how the LED matrix works and we made the codes for the flash and the heart. Unfortunately we run out of time and was not possible to make the equalizer. Instead the equalizer, we made a code for a digital VU Meter that simulates an equalizer.

Also we had to design a PCB for our Application. For it, I had to see some tutorials in you tube of EAGLE software. When I printed the PCB and I solded, for the first time in my life, the components to it, the PCB doesn't work properly. It took us some days until the problem was solved. One side of the condensator was not solded to the PCB and the AC signal coupling didn't work. Also, because of the distance, when we speak not always the VU works properly.

As in the PCB design, I have to watch some tutorials of SolidWorks software. With it we designed a Box for the LED Matrix that later would be printed with a 3D Printer. When the design was finished in SolidWorks, we had some problems with the 3D printer, but in the third trial the solid object was good printed. In one of the failed printouts I noticed that the parameters were reduce with the printer. So, before our third trial, I modified the value of some parameters and I made the design more efficiently saving material.

With the VU application sometimes the LEDs flicker very quickly. After searching some information on the internet, we discovered that each individual NeoPixel draws up to 60 milliamps at maximum brightness white (red + green + blue).



In actual use though, it's rare for all pixels to be turned on that way. When mixing colours and displaying animations, the current draw will be much less.

It's impossible to estimate a single number for all circumstances, but we've been using 1/3 this (20 mA per pixel) as a gross rule of thumb with no ill effects.

To estimate power supply needs, multiply the number of pixels by 20, then divide the result by 1,000 for the "rule of thumb" power supply rating in Amps. Or use 60 (instead of 20) if you want to guarantee an absolute margin of safety for all situations. For example:

256	NeoPixels	×	20	mA	÷	1,000	=	5.2	Amps	minimum
256	NeoPixels	×	60	mA	÷	1,000	=	15.4	Amps	minimum

In conclusion, for the properly working of the LEDs and the application we need a power supply between 5.2 - 15.4 Amps. The power supply that we used only had 1.0 Amps.

Also you must remember this rule not to kill your LEDs:

**Extra Amps = Good but extra Volts = bad**

## 6.2 VALUATIONS

During this semester, I have realised that electronic and programming is the technology for the future. This technology is very intensive and interesting, and knowing programming is possible to do a lot off new application for different devices then later can be improved with no high economic costs.

Regarding the project, it could be improved finishing the idea an equalizer. With the other two modes it should be nice to do one of the next applications:

- Digital alarm clock
- A lamp that is turn on/off with a clap
- Add thermometer to see the temperature on the LED Matrix

More complicate but not impossible would be programming a capacitive screen and crate a code only to turn on the LEDs sliding your fingers over them. That will make easier drawing forms and shapes in the LED Matrix.

# BIBLIOGRAPHY

---

1. Arduino Due board features  
<http://arduino.cc/en/Main/arduinoBoardDue>
2. Working with Adafruit Neo Pixel Library  
<https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library>
3. Programming of a LED Matrix with Adafruit Neo Pixel Library  
<https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library>
4. WS 2812b LED features  
<http://www.world-semi.com/en/>
5. Tutorials EAGLE Software  
[https://www.youtube.com/results?search\\_query=tutoria+eagle](https://www.youtube.com/results?search_query=tutoria+eagle)
6. Tutorials SolidWorks Software  
<https://www.youtube.com/results?q=tutorial+solidworks+2013>
7. Tutorials Arduino Software(C++)  
[https://www.youtube.com/results?search\\_query=tutorial+c%2B%2B](https://www.youtube.com/results?search_query=tutorial+c%2B%2B)
8. 3D X400 Printer  
<https://shop.germanreprap.com/en/x400-standard>
9. Op Amp and Microphone Electric Circuit Information  
[http://es.wikipedia.org/wiki/Amplificador\\_operacional](http://es.wikipedia.org/wiki/Amplificador_operacional)  
<http://electroniciayciencia.blogspot.de/2010/05/preamplificador-microfono-electret.html>  
<http://www.electronicafacil.net/tutoriales/AMPLIFICADOR-INVERSOR.php>
10. VU Meter Information  
<http://playingwitharduino.blogspot.de/2011/12/vumetro-leds-al-ritmo-de-la-musica.html>  
<https://geekytheory.com/vu-meter-con-arduino-2/>
11. TLV 2772A T.I. Amplifier  
<http://www.ti.com/product/tlv2772a>

# **ANNEX 1**

## **ADAFRUIT NEOPIXEL LIBRARY**

---

This file is part of the Adafruit NeoPixel library.

NeoPixel is free software: you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

NeoPixel is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with NeoPixel. If not, see <http://www.gnu.org/licenses/>.

-----\*/

```
#ifndef ADAFRUIT_NEOPIXEL_H
#define ADAFRUIT_NEOPIXEL_H
```

```
#if (ARDUINO >= 100)
#include <Arduino.h>
#else
#include <WProgram.h>
#include <pins_arduino.h>
#endif
```

```
// 'type' flags for LED pixels (third parameter to constructor):
#define NEO_RGB      0x00 // Wired for RGB data order
#define NEO_GRB      0x01 // Wired for GRB data order
#define NEO_BRG      0x04

#define NEO_COLMASK  0x01
#define NEO_KHZ800    0x02 // 800 KHz datastream
#define NEO_SPDMASK   0x02
// Trinket flash space is tight, v1 NeoPixels aren't handled by default.
// Remove the ifndef/endif to add support -- but code will be bigger.
// Conversely, can comment out the #defines to save space on other MCUs.
#ifndef __AVR_ATtiny85__
#define NEO_KHZ400    0x00 // 400 KHz datastream
#endif
```

```
class Adafruit_NeoPixel {

public:

    // Constructor: number of LEDs, pin number, LED type
    Adafruit_NeoPixel(uint16_t n, uint8_t p=6, uint8_t t=NEO_GRB + NEO_KHZ800);
    ~Adafruit_NeoPixel();

    void
        begin(void),
        show(void),
        setPin(uint8_t p),
        setPixelColor(uint16_t n, uint8_t r, uint8_t g, uint8_t b),
        setPixelColor(uint16_t n, uint32_t c),
        setBrightness(uint8_t),
        clear();
    uint8_t
        *getPixels(void) const,
        getBrightness(void) const;
    uint16_t
        numPixels(void) const;
    static uint32_t
        Color(uint8_t r, uint8_t g, uint8_t b);
    uint32_t
        getPixelColor(uint16_t n) const;
    inline bool
        canShow(void) { return (micros() - endTime) >= 50L; }

private:

    const uint16_t
        numLEDs,      // Number of RGB LEDs in strip
        numBytes;      // Size of 'pixels' buffer below
    uint8_t
        pin,           // Output pin number
        brightness,
        *pixels,        // Holds LED color values (3 bytes each)
        rOffset,        // Index of red byte within each 3-byte pixel
        gOffset,        // Index of green byte
        bOffset;        // Index of blue byte
    const uint8_t
        type;          // Pixel flags (400 vs 800 KHz, RGB vs GRB color)
```

```
uint32_t
    endTime;          // Latch timing reference
#ifdef __AVR__
    const volatile uint8_t
        *port;         // Output PORT register
    uint8_t
        pinMask;       // Output PORT bitmask
#endif

};

#endif // ADAFRUIT_NEOPIXEL_H
```

# **ANNEX 2**

## **APPLICATION CODE**

---

```
#include <Adafruit_NeoPixel.h> // To recognise Adafruit_NeoPixel library functions.

#define BUTTON_PIN7 7 // The green button is define in PIN 7

#define BUTTON_PIN2 2 //The orange button is define in PIN 2

#define PIXEL_PIN 6 //The data pin is define in PIN 6

#define PIXEL_COUNT 256 // The n° of LED we are going to use are define

int i = 0,j,h,k;

// we declare a NeoPixel object. We will refer to "Adafruit_NeoPixel strip" to control the strip of pixels

Adafruit_NeoPixel strip = Adafruit_NeoPixel(PIXEL_COUNT, PIXEL_PIN, NEO_GRB + NEO_KHZ800);

const int sampleWindow = 20; // It is a window in msSample window width (50 mS = 20Hz)
unsigned int sample;

/*The setup() function is called when a sketch starts.
Use it to initialize variables,pin modes, start using libraries, etc.
The setup function will only run once, after each powerup or reset of the Arduino board*/

void setup()
{
    //Opens serial port, sets data rate to 9600 bps
    Serial.begin(9600);

    //Set the digital pin 2 and 7 as pull up
    pinMode(BUTTON_PIN2, INPUT_PULLUP);
    pinMode(BUTTON_PIN7, INPUT_PULLUP);
}
```



```
//Replaces any previous function that was attached to the interrupt with pin 7 and 2
attachInterrupt(7, flash, HIGH);
attachInterrupt(2, heart,HIGH);

//The data pin is prepared for NeoPixel output
strip.begin();

//We introduce LED brightness value and we Initialize all pixels to "off"
strip.setBrightness(255);
strip.show();
}

/*After creating a setup() function, which initializes and sets the initial values,
the loop() function does precisely what its name suggests, and loops consecutively,
allowing your program to change and respond.*/

void loop()
{

    /*With the next functions we will see the signal
    of the microphone on a sampling window*/

    unsigned long startMillis= millis(); //We start sampling window
    unsigned int peakToPeak = 0; // Level peak to peak

    unsigned int signalMax = 0;
    unsigned int signalMin = 1024;

    while (millis() - startMillis < sampleWindow) // We recive data with an speed of 50ms
    {
```

sample = `analogRead(0)`; // The output of the amplifier is connected to the analog A0 input of Arduino

```

    if (sample < 1024)
    {
        if (sample > signalMax)
        {
            signalMax = sample; // Max. Signal, sampling
        }
        else if (sample < signalMin)
        {
            signalMin = sample; // Min. Signal, sampling
        }
    }
}

peakToPeak = signalMax - signalMin; // Max - Min = range peak to peak
Serial.print(peakToPeak);
double volts = (peakToPeak * 5) / 1024; // To convert to Volts

/*Now we define the code to create the VU Meter.
We have created five bands with different parameters*/

//BANDA 1

if (peakToPeak>1){ //Peaktpeak>1 the LED will turn on in green
    strip.setPixelColor(16, 0, 255, 0);
    strip.setPixelColor(17, 0, 255, 0);
    strip.setPixelColor(46, 0, 255, 0);
    strip.setPixelColor(47, 0, 255, 0);
}

else{ //peaktpeak<1 The LED will turn off
    strip.setPixelColor(16, 0, 0, 0);
    strip.setPixelColor(17, 0, 0, 0);
    strip.setPixelColor(46, 0, 0, 0);
    strip.setPixelColor(47, 0, 0, 0);
}

```

```
}
```

```
if (peakToPeak>250){  
    strip.setPixelColor(18, 0, 255, 0);  
    strip.setPixelColor(19, 0, 255, 0);  
    strip.setPixelColor(44, 0, 255, 0);  
    strip.setPixelColor(45, 0, 255, 0);
```

```
}
```

```
else{  
    strip.setPixelColor(18, 0, 0, 0);  
    strip.setPixelColor(19, 0, 0, 0);  
    strip.setPixelColor(44, 0, 0, 0);  
    strip.setPixelColor(45, 0, 0, 0);
```

```
}
```

```
if (peakToPeak>260){  
    strip.setPixelColor(20, 0, 255, 0);  
    strip.setPixelColor(21, 0, 255, 0);  
    strip.setPixelColor(42, 0, 255, 0);  
    strip.setPixelColor(43, 0, 255, 0);
```

```
}
```

```
else{  
    strip.setPixelColor(20, 0, 0, 0);  
    strip.setPixelColor(21, 0, 0, 0);  
    strip.setPixelColor(42, 0, 0, 0);  
    strip.setPixelColor(43, 0, 0, 0);
```

```
}
```

```
if (peakToPeak>270){  
    strip.setPixelColor(22, 204, 204, 0);  
    strip.setPixelColor(23, 204, 204, 0);
```

```
strip.setPixelColor(40, 204, 204, 0);  
strip.setPixelColor(41, 204, 204, 0);  
}
```

```
else{  
    strip.setPixelColor(22, 0, 0, 0);  
    strip.setPixelColor(23, 0, 0, 0);  
    strip.setPixelColor(40, 0, 0, 0);  
    strip.setPixelColor(41, 0, 0, 0);  
}
```

```
if (peakToPeak>280){  
    strip.setPixelColor(24, 204, 204, 0);  
    strip.setPixelColor(25, 204, 204, 0);  
    strip.setPixelColor(38, 204, 204, 0);  
    strip.setPixelColor(39, 204, 204, 0);  
}
```

```
else{  
    strip.setPixelColor(24, 0, 0, 0);  
    strip.setPixelColor(25, 0, 0, 0);  
    strip.setPixelColor(38, 0, 0, 0);  
    strip.setPixelColor(39, 0, 0, 0);  
}
```

```
if (peakToPeak>290){  
    strip.setPixelColor(26, 204, 204, 0);  
    strip.setPixelColor(37, 204, 204, 0);  
}  
else{  
    strip.setPixelColor(26, 0, 0, 0);  
    strip.setPixelColor(37, 0, 0, 0);  
}
```

```
if (peakToPeak>300){  
    strip.setPixelColor(27, 255, 128, 0);  
    strip.setPixelColor(28, 255, 128, 0);  
    strip.setPixelColor(35, 255, 128, 0);  
    strip.setPixelColor(36, 255, 128, 0);  
}  
else{  
    strip.setPixelColor(27, 0, 0, 0);  
    strip.setPixelColor(28, 0, 0, 0);  
    strip.setPixelColor(35, 0, 0, 0);  
    strip.setPixelColor(36, 0, 0, 0);  
}
```

```
if (peakToPeak>310){  
    strip.setPixelColor(29, 255, 128, 0);  
    strip.setPixelColor(34, 255, 128, 0);  
}  
else{  
    strip.setPixelColor(29, 0, 0, 0);  
    strip.setPixelColor(34, 0, 0, 0);  
}
```

```
if (peakToPeak>320){  
    strip.setPixelColor(30, 255, 0, 0);  
    strip.setPixelColor(33, 255, 0, 0);  
}  
else{  
    strip.setPixelColor(30, 0, 0, 0);  
    strip.setPixelColor(33, 0, 0, 0);  
}
```

```
if (peakToPeak>330){  
    strip.setPixelColor(31, 255, 0, 0);  
    strip.setPixelColor(32, 255, 0, 0);  
}  
else{  
    strip.setPixelColor(31, 0, 0, 0);  
    strip.setPixelColor(32, 0, 0, 0);  
}
```

```
//BANDA 2
```

```
if(peakToPeak>1){  
    strip.setPixelColor(78, 0, 255, 0);  
    strip.setPixelColor(79, 0, 255, 0);  
    strip.setPixelColor(80, 0, 255, 0);  
    strip.setPixelColor(81, 0, 255, 0);  
}  
else{  
    strip.setPixelColor(78, 0, 0, 0);  
    strip.setPixelColor(79, 0, 0, 0);  
    strip.setPixelColor(80, 0, 0, 0);  
    strip.setPixelColor(81, 0, 0, 0);  
}
```

```
if (peakToPeak>350){  
    strip.setPixelColor(76, 0, 255, 0);  
    strip.setPixelColor(77, 0, 255, 0);  
    strip.setPixelColor(82, 0, 255, 0);  
    strip.setPixelColor(83, 0, 255, 0);  
}  
else{
```

```
strip.setPixelColor(76, 0, 0, 0);  
strip.setPixelColor(77, 0, 0, 0);  
strip.setPixelColor(82, 0, 0, 0);  
strip.setPixelColor(83, 0, 0, 0);  
}
```

```
if (peakToPeak>370){  
    strip.setPixelColor(74, 0, 255, 0);  
    strip.setPixelColor(75, 0, 255, 0);  
    strip.setPixelColor(84, 0, 255, 0);  
    strip.setPixelColor(85, 0, 255, 0);  
}
```

```
else{  
    strip.setPixelColor(74, 0, 0, 0);  
    strip.setPixelColor(75, 0, 0, 0);  
    strip.setPixelColor(84, 0, 0, 0);  
    strip.setPixelColor(85, 0, 0, 0);  
}
```

```
if (peakToPeak>390){  
    strip.setPixelColor(72, 204, 204, 0);  
    strip.setPixelColor(73, 204, 204, 0);  
    strip.setPixelColor(86, 204, 204, 0);  
    strip.setPixelColor(87, 204, 204, 0);  
}
```

```
else{  
    strip.setPixelColor(72, 0, 0, 0);  
    strip.setPixelColor(73, 0, 0, 0);  
    strip.setPixelColor(86, 0, 0, 0);  
    strip.setPixelColor(87, 0, 0, 0);  
}
```

```
if (peakToPeak>410){  
    strip.setPixelColor(70, 204, 204, 0);  
    strip.setPixelColor(71, 204, 204, 0);  
    strip.setPixelColor(88, 204, 204, 0);  
    strip.setPixelColor(89, 204, 204, 0);  
}  
else{  
    strip.setPixelColor(70, 0, 0, 0);  
    strip.setPixelColor(71, 0, 0, 0);  
    strip.setPixelColor(88, 0, 0, 0);  
    strip.setPixelColor(89, 0, 0, 0);  
}
```

```
if (peakToPeak>430){  
    strip.setPixelColor(69, 204, 204, 0);  
    strip.setPixelColor(90, 204, 204, 0);  
}  
else{  
    strip.setPixelColor(69, 0, 0, 0);  
    strip.setPixelColor(90, 0, 0, 0);  
}
```

```
if (peakToPeak>450){  
    strip.setPixelColor(67, 255, 128, 0);  
    strip.setPixelColor(68, 255, 128, 0);  
    strip.setPixelColor(91, 255, 128, 0);  
    strip.setPixelColor(92, 255, 128, 0);  
}  
else{  
    strip.setPixelColor(67, 0, 0, 0);  
    strip.setPixelColor(68, 0, 0, 0);  
    strip.setPixelColor(91, 0, 0, 0);  
}
```



```
strip.setPixelColor(92, 0, 0, 0);  
}  
  
if (peakToPeak>470){  
    strip.setPixelColor(66, 255, 128, 0);  
    strip.setPixelColor(93, 255, 128, 0);  
}  
else{  
    strip.setPixelColor(66, 0, 0, 0);  
    strip.setPixelColor(93, 0, 0, 0);  
}  
  
if (peakToPeak>490){  
    strip.setPixelColor(65, 255, 0, 0);  
    strip.setPixelColor(94, 255, 0, 0);  
}  
else{  
    strip.setPixelColor(65, 0, 0, 0);  
    strip.setPixelColor(94, 0, 0, 0);  
}  
  
if (peakToPeak>510){  
    strip.setPixelColor(64, 255, 0, 0);  
    strip.setPixelColor(95, 255, 0, 0);  
}  
else{  
    strip.setPixelColor(64, 0, 0, 0);  
    strip.setPixelColor(95, 0, 0, 0);  
}
```

//BANDA 3

```
if (peakToPeak>1){
    strip.setPixelColor(112, 0, 255, 0);
    strip.setPixelColor(113, 0, 255, 0);
    strip.setPixelColor(142, 0, 255, 0);
    strip.setPixelColor(143, 0, 255, 0);
}
else{
    strip.setPixelColor(112, 0, 0, 0);
    strip.setPixelColor(113, 0, 0, 0);
    strip.setPixelColor(142, 0, 0, 0);
    strip.setPixelColor(143, 0, 0, 0);
}

if (peakToPeak>160){
    strip.setPixelColor(114, 0, 255, 0);
    strip.setPixelColor(115, 0, 255, 0);
    strip.setPixelColor(140, 0, 255, 0);
    strip.setPixelColor(141, 0, 255, 0);
}
else{
    strip.setPixelColor(114, 0, 0, 0);
    strip.setPixelColor(115, 0, 0, 0);
    strip.setPixelColor(140, 0, 0, 0);
    strip.setPixelColor(141, 0, 0, 0);
}

if (peakToPeak>180){
    strip.setPixelColor(116, 0, 255, 0);
    strip.setPixelColor(117, 0, 255, 0);
    strip.setPixelColor(138, 0, 255, 0);
    strip.setPixelColor(139, 0, 255, 0);
}
else{
```

```
strip.setPixelColor(116, 0, 0, 0);
strip.setPixelColor(117, 0, 0, 0);
strip.setPixelColor(138, 0, 0, 0);
strip.setPixelColor(139, 0, 0, 0);
}

if (peakToPeak>200){
    strip.setPixelColor(118, 204, 204, 0);
    strip.setPixelColor(119, 204, 204, 0);
    strip.setPixelColor(136, 204, 204, 0);
    strip.setPixelColor(137, 204, 204, 0);
}
else{
    strip.setPixelColor(118, 0, 0, 0);
    strip.setPixelColor(119, 0, 0, 0);
    strip.setPixelColor(136, 0, 0, 0);
    strip.setPixelColor(137, 0, 0, 0);
}

if (peakToPeak>220){
    strip.setPixelColor(120, 204, 204, 0);
    strip.setPixelColor(121, 204, 204, 0);
    strip.setPixelColor(134, 204, 204, 0);
    strip.setPixelColor(135, 204, 204, 0);
}
else{
    strip.setPixelColor(120, 0, 0, 0);
    strip.setPixelColor(121, 0, 0, 0);
    strip.setPixelColor(134, 0, 0, 0);
    strip.setPixelColor(135, 0, 0, 0);
}

if (peakToPeak>230){
    strip.setPixelColor(122, 204, 204, 0);
```

```
    strip.setPixelColor(133, 204, 204, 0);
}
else{
    strip.setPixelColor(122, 0, 0, 0);
    strip.setPixelColor(133, 0, 0, 0);
}

if (peakToPeak>240){
    strip.setPixelColor(123, 255, 128, 0);
    strip.setPixelColor(124, 255, 128, 0);
    strip.setPixelColor(132, 255, 128, 0);
    strip.setPixelColor(131, 255, 128, 0);
}
else{
    strip.setPixelColor(123, 0, 0, 0);
    strip.setPixelColor(124, 0, 0, 0);
    strip.setPixelColor(132, 0, 0, 0);
    strip.setPixelColor(131, 0, 0, 0);
}

if (peakToPeak>250){
    strip.setPixelColor(125, 255, 128, 0);
    strip.setPixelColor(130, 255, 128, 0);
}
else{
    strip.setPixelColor(125, 0, 0, 0);
    strip.setPixelColor(130, 0, 0, 0);
}

if (peakToPeak>260){
    strip.setPixelColor(126, 255, 0, 0);
    strip.setPixelColor(129, 255, 0, 0);
}
else{
```

```
strip.setPixelColor(126, 0, 0, 0);  
strip.setPixelColor(129, 0, 0, 0);  
}
```

```
if (peakToPeak>270){  
    strip.setPixelColor(127, 255, 0, 0);  
    strip.setPixelColor(128, 255, 0, 0);  
}  
else{  
    strip.setPixelColor(127, 0, 0, 0);  
    strip.setPixelColor(128, 0, 0, 0);  
}
```

```
//BANDA 4
```

```
if(peakToPeak>1){  
    strip.setPixelColor(174, 0, 255, 0);  
    strip.setPixelColor(175, 0, 255, 0);  
    strip.setPixelColor(176, 0, 255, 0);  
    strip.setPixelColor(177, 0, 255, 0);  
}  
else{  
    strip.setPixelColor(174, 0, 0, 0);  
    strip.setPixelColor(175, 0, 0, 0);  
    strip.setPixelColor(176, 0, 0, 0);  
    strip.setPixelColor(177, 0, 0, 0);  
}
```

```
if (peakToPeak>200){  
    strip.setPixelColor(172, 0, 255, 0);  
    strip.setPixelColor(173, 0, 255, 0);  
    strip.setPixelColor(178, 0, 255, 0);  
    strip.setPixelColor(179, 0, 255, 0);  
}
```

```
else{
    strip.setPixelColor(172, 0, 0, 0);
    strip.setPixelColor(173, 0, 0, 0);
    strip.setPixelColor(178, 0, 0, 0);
    strip.setPixelColor(179, 0, 0, 0);
}

if (peakToPeak>240){
    strip.setPixelColor(170, 0, 255, 0);
    strip.setPixelColor(171, 0, 255, 0);
    strip.setPixelColor(180, 0, 255, 0);
    strip.setPixelColor(181, 0, 255, 0);
}
else{
    strip.setPixelColor(170, 0, 0, 0);
    strip.setPixelColor(171, 0, 0, 0);
    strip.setPixelColor(180, 0, 0, 0);
    strip.setPixelColor(181, 0, 0, 0);
}

if (peakToPeak>280){
    strip.setPixelColor(168, 204, 204, 0);
    strip.setPixelColor(169, 204, 204, 0);
    strip.setPixelColor(182, 204, 204, 0);
    strip.setPixelColor(183, 204, 204, 0);
}

else{
    strip.setPixelColor(168, 0, 0, 0);
    strip.setPixelColor(169, 0, 0, 0);
    strip.setPixelColor(182, 0, 0, 0);
    strip.setPixelColor(183, 0, 0, 0);
}
```

```
if (peakToPeak>320){
    strip.setPixelColor(166, 204, 204, 0);
    strip.setPixelColor(167, 204, 204, 0);
    strip.setPixelColor(184, 204, 204, 0);
    strip.setPixelColor(185, 204, 204, 0);
}
else{
    strip.setPixelColor(166, 0, 0, 0);
    strip.setPixelColor(167, 0, 0, 0);
    strip.setPixelColor(184, 0, 0, 0);
    strip.setPixelColor(185, 0, 0, 0);
}

if (peakToPeak>360){
    strip.setPixelColor(165, 204, 204, 0);
    strip.setPixelColor(186, 204, 204, 0);
}
else{
    strip.setPixelColor(165, 0, 0, 0);
    strip.setPixelColor(186, 0, 0, 0);
}

if (peakToPeak>400){
    strip.setPixelColor(163, 255, 128, 0);
    strip.setPixelColor(164, 255, 128, 0);
    strip.setPixelColor(187, 255, 128, 0);
    strip.setPixelColor(188, 255, 128, 0);
}
else{
    strip.setPixelColor(163, 0, 0, 0);
    strip.setPixelColor(164, 0, 0, 0);
    strip.setPixelColor(187, 0, 0, 0);
    strip.setPixelColor(188, 0, 0, 0);
}
```

```
if (peakToPeak>450){  
    strip.setPixelColor(162, 255, 128, 0);  
    strip.setPixelColor(189, 255, 128, 0);  
}  
else{  
    strip.setPixelColor(162, 0, 0, 0);  
    strip.setPixelColor(189, 0, 0, 0);  
}
```

```
if (peakToPeak>500){  
    strip.setPixelColor(161, 255, 0, 0);  
    strip.setPixelColor(190, 255, 0, 0);  
}  
else{  
    strip.setPixelColor(161, 0, 0, 0);  
    strip.setPixelColor(190, 0, 0, 0);  
}
```

```
if (peakToPeak>600){  
    strip.setPixelColor(160, 255, 0, 0);  
    strip.setPixelColor(191, 255, 0, 0);  
    //strip.show();  
}  
else{  
    strip.setPixelColor(160, 0, 0, 0);  
    strip.setPixelColor(191, 0, 0, 0);  
    //strip.show();  
}
```

```
//BANDA 5
```

```
if(peakToPeak>1){
```



```
strip.setPixelColor(208, 0, 255, 0);
strip.setPixelColor(209, 0, 255, 0);
strip.setPixelColor(238, 0, 255, 0);
strip.setPixelColor(239, 0, 255, 0);
}
else{
strip.setPixelColor(208, 0, 0, 0);
strip.setPixelColor(209, 0, 0, 0);
strip.setPixelColor(238, 0, 0, 0);
strip.setPixelColor(239, 0, 0, 0);
}

if (peakToPeak>350){
strip.setPixelColor(210, 0, 255, 0);
strip.setPixelColor(211, 0, 255, 0);
strip.setPixelColor(236, 0, 255, 0);
strip.setPixelColor(237, 0, 255, 0);
}
else{
strip.setPixelColor(210, 0, 0, 0);
strip.setPixelColor(211, 0, 0, 0);
strip.setPixelColor(236, 0, 0, 0);
strip.setPixelColor(237, 0, 0, 0);
}

if (peakToPeak>370){
strip.setPixelColor(212, 0, 255, 0);
strip.setPixelColor(213, 0, 255, 0);
strip.setPixelColor(234, 0, 255, 0);
strip.setPixelColor(235, 0, 255, 0);
}
else{
strip.setPixelColor(212, 0, 0, 0);
strip.setPixelColor(213, 0, 0, 0);
```

```
strip.setPixelColor(234, 0, 0, 0);
strip.setPixelColor(235, 0, 0, 0);
}

if (peakToPeak>390){
    strip.setPixelColor(214, 204, 204, 0);
    strip.setPixelColor(215, 204, 204, 0);
    strip.setPixelColor(232, 204, 204, 0);
    strip.setPixelColor(233, 204, 204, 0);
}
else{
    strip.setPixelColor(214, 0, 0, 0);
    strip.setPixelColor(215, 0, 0, 0);
    strip.setPixelColor(232, 0, 0, 0);
    strip.setPixelColor(233, 0, 0, 0);
}

if (peakToPeak>410){
    strip.setPixelColor(216, 204, 204, 0);
    strip.setPixelColor(217, 204, 204, 0);
    strip.setPixelColor(230, 204, 204, 0);
    strip.setPixelColor(231, 204, 204, 0);
}
else{
    strip.setPixelColor(216, 0, 0, 0);
    strip.setPixelColor(217, 0, 0, 0);
    strip.setPixelColor(230, 0, 0, 0);
    strip.setPixelColor(231, 0, 0, 0);
}

if (peakToPeak>430){
    strip.setPixelColor(218, 204, 204, 0);
    strip.setPixelColor(229, 204, 204, 0);
}
```

```
else{
    strip.setPixelColor(218, 0, 0, 0);
    strip.setPixelColor(229, 0, 0, 0);
}

    if (peakToPeak>450){
        strip.setPixelColor(219, 255, 128, 0);
        strip.setPixelColor(220, 255, 128, 0);
        strip.setPixelColor(227, 255, 128, 0);
        strip.setPixelColor(228, 255, 128, 0);
    }
    else{
        strip.setPixelColor(219, 0, 0, 0);
        strip.setPixelColor(220, 0, 0, 0);
        strip.setPixelColor(227, 0, 0, 0);
        strip.setPixelColor(228, 0, 0, 0);
    }

    if (peakToPeak>470){
        strip.setPixelColor(221, 255, 128, 0);
        strip.setPixelColor(226, 255, 128, 0);
    }
    else{
        strip.setPixelColor(221, 0, 0, 0);
        strip.setPixelColor(226, 0, 0, 0);
    }

    if (peakToPeak>490){
        strip.setPixelColor(222, 255, 0, 0);
        strip.setPixelColor(225, 255, 0, 0);
    }
    else{
        strip.setPixelColor(222, 0, 0, 0);
```

```
strip.setPixelColor(225, 0, 0, 0);

}

if (peakToPeak>510){
  strip.setPixelColor(223, 255, 0, 0);
  strip.setPixelColor(224, 255, 0, 0);

}
else{
  strip.setPixelColor(223, 0, 0, 0);
  strip.setPixelColor(224, 0, 0, 0);
  strip.show();
}

Serial.print("\t");
  Serial.println(volts);
if (volts>0.4){          // Si el valor leído supera al umbral damos una alarma visual
}

else if (volts<0.4){    // De lo contrario, no damos ninguna alarma
}

}

//With the next functions we activate the yellow flash when we push the green button
void flash(){

  //We reset all the LED before the flash not to mix the it with the VU meter
  for(i=0;i<256;i++){

    strip.setPixelColor(i, 0, 0, 0);
```

```
}  
strip.show();
```

```
//We define the LED that form the Flash
```

```
strip.setPixelColor(51, 204, 204, 0);  
strip.setPixelColor(77, 204, 204, 0);  
strip.setPixelColor(76, 204, 204, 0);  
strip.setPixelColor(81, 204, 204, 0);  
strip.setPixelColor(82, 204, 204, 0);  
strip.setPixelColor(83, 204, 204, 0);  
strip.setPixelColor(84, 204, 204, 0);  
strip.setPixelColor(90, 204, 204, 0);  
strip.setPixelColor(100, 204, 204, 0);  
strip.setPixelColor(101, 204, 204, 0);  
strip.setPixelColor(106, 204, 204, 0);  
strip.setPixelColor(107, 204, 204, 0);  
strip.setPixelColor(108, 204, 204, 0);  
strip.setPixelColor(109, 204, 204, 0);  
strip.setPixelColor(110, 204, 204, 0);  
strip.setPixelColor(111, 204, 204, 0);  
strip.setPixelColor(114, 204, 204, 0);  
strip.setPixelColor(113, 204, 204, 0);  
strip.setPixelColor(115, 204, 204, 0);  
strip.setPixelColor(116, 204, 204, 0);  
strip.setPixelColor(117, 204, 204, 0);  
strip.setPixelColor(118, 204, 204, 0);  
strip.setPixelColor(124, 204, 204, 0);  
strip.setPixelColor(123, 204, 204, 0);  
strip.setPixelColor(122, 204, 204, 0);  
strip.setPixelColor(130, 204, 204, 0);  
strip.setPixelColor(131, 204, 204, 0);  
strip.setPixelColor(132, 204, 204, 0);  
strip.setPixelColor(133, 204, 204, 0);
```

```
strip.setPixelColor(136, 204, 204, 0);
strip.setPixelColor(137, 204, 204, 0);
strip.setPixelColor(140, 204, 204, 0);
strip.setPixelColor(141, 204, 204, 0);
strip.setPixelColor(147, 204, 204, 0);
strip.setPixelColor(152, 204, 204, 0);
strip.setPixelColor(151, 204, 204, 0);
strip.setPixelColor(155, 204, 204, 0);
strip.setPixelColor(152, 204, 204, 0);
strip.setPixelColor(155, 204, 204, 0);
strip.setPixelColor(154, 204, 204, 0);
strip.setPixelColor(157, 204, 204, 0);
strip.setPixelColor(158, 204, 204, 0);
strip.setPixelColor(160, 204, 204, 0);
strip.setPixelColor(161, 204, 204, 0);
strip.setPixelColor(164, 204, 204, 0);
strip.setPixelColor(165, 204, 204, 0);
strip.setPixelColor(166, 204, 204, 0);
strip.setPixelColor(167, 204, 204, 0);
strip.setPixelColor(187, 204, 204, 0);
strip.setPixelColor(186, 204, 204, 0);
strip.setPixelColor(185, 204, 204, 0);
strip.setPixelColor(196, 204, 204, 0);
strip.setPixelColor(197, 204, 204, 0);
strip.setPixelColor(219, 204, 204, 0);
strip.setPixelColor(191, 204, 204, 0);
strip.show();
```

```
//The flash is on for 2 sec. and then the application turns to the VU Meter
```

```
delayMicroseconds(900000);
```

```
strip.setPixelColor(51, 0, 0, 0);
```

```
strip.setPixelColor(77, 0, 0, 0);
```

```
strip.setPixelColor(76, 0, 0, 0);  
strip.setPixelColor(81, 0, 0, 0);  
strip.setPixelColor(82, 0, 0, 0);  
strip.setPixelColor(83, 0, 0, 0);  
strip.setPixelColor(84, 0, 0, 0);  
strip.setPixelColor(90, 0, 0, 0);  
strip.setPixelColor(100, 0, 0, 0);  
strip.setPixelColor(101, 0, 0, 0);  
strip.setPixelColor(106, 0, 0, 0);  
strip.setPixelColor(107, 0, 0, 0);  
strip.setPixelColor(108, 0, 0, 0);  
strip.setPixelColor(109, 0, 0, 0);  
strip.setPixelColor(110, 0, 0, 0);  
strip.setPixelColor(111, 0, 0, 0);  
strip.setPixelColor(114, 0, 0, 0);  
strip.setPixelColor(113, 0, 0, 0);  
strip.setPixelColor(115, 0, 0, 0);  
strip.setPixelColor(116, 0, 0, 0);  
strip.setPixelColor(117, 0, 0, 0);  
strip.setPixelColor(118, 0, 0, 0);  
strip.setPixelColor(124, 0, 0, 0);  
strip.setPixelColor(123, 0, 0, 0);  
strip.setPixelColor(122, 0, 0, 0);  
strip.setPixelColor(130, 0, 0, 0);  
strip.setPixelColor(131, 0, 0, 0);  
strip.setPixelColor(132, 0, 0, 0);  
strip.setPixelColor(133, 0, 0, 0);  
strip.setPixelColor(136, 0, 0, 0);  
strip.setPixelColor(137, 0, 0, 0);  
strip.setPixelColor(140, 0, 0, 0);  
strip.setPixelColor(141, 0, 0, 0);  
strip.setPixelColor(147, 0, 0, 0);  
strip.setPixelColor(152, 0, 0, 0);  
strip.setPixelColor(151, 0, 0, 0);
```

```
strip.setPixelColor(155, 0, 0, 0);
strip.setPixelColor(152, 0, 0, 0);
strip.setPixelColor(155, 0, 0, 0);
strip.setPixelColor(154, 0, 0, 0);
strip.setPixelColor(157, 0, 0, 0);
strip.setPixelColor(158, 0, 0, 0);
strip.setPixelColor(160, 0, 0, 0);
strip.setPixelColor(161, 0, 0, 0);
strip.setPixelColor(164, 0, 0, 0);
strip.setPixelColor(165, 0, 0, 0);
strip.setPixelColor(166, 0, 0, 0);
strip.setPixelColor(167, 0, 0, 0);
strip.setPixelColor(187, 0, 0, 0);
strip.setPixelColor(186, 0, 0, 0);
strip.setPixelColor(185, 0, 0, 0);
strip.setPixelColor(196, 0, 0, 0);
strip.setPixelColor(197, 0, 0, 0);
strip.setPixelColor(219, 0, 0, 0);
strip.setPixelColor(191, 0, 0, 0);
strip.show();

}

void heart(){

    for(i=0;i<256;i++){

        strip.setPixelColor(i, 0, 0, 0);
    }

    strip.show();

    strip.setPixelColor(4, 255, 0, 0);
    strip.setPixelColor(5, 255, 0, 0);
    strip.setPixelColor(6, 255, 0, 0);
```



```
strip.setPixelColor(23, 255, 0, 0);  
strip.setPixelColor(7, 255, 0, 0);  
strip.setPixelColor(3, 255, 0, 0);  
strip.setPixelColor(34, 255, 0, 0);  
strip.setPixelColor(29, 255, 0, 0);  
strip.setPixelColor(41, 255, 0, 0);  
strip.setPixelColor(53, 255, 0, 0);  
strip.setPixelColor(61, 255, 0, 0);  
strip.setPixelColor(66, 255, 0, 0);  
strip.setPixelColor(75, 255, 0, 0);  
strip.setPixelColor(92, 255, 0, 0);  
strip.setPixelColor(83, 255, 0, 0);  
strip.setPixelColor(100, 255, 0, 0);  
strip.setPixelColor(109, 255, 0, 0);  
strip.setPixelColor(113, 255, 0, 0);  
strip.setPixelColor(122, 255, 0, 0);  
strip.setPixelColor(133, 255, 0, 0);  
strip.setPixelColor(134, 255, 0, 0);  
strip.setPixelColor(143, 255, 0, 0);  
strip.setPixelColor(145, 255, 0, 0);  
strip.setPixelColor(155, 255, 0, 0);  
strip.setPixelColor(163, 255, 0, 0);  
strip.setPixelColor(173, 255, 0, 0);  
strip.setPixelColor(179, 255, 0, 0);  
strip.setPixelColor(189, 255, 0, 0);  
strip.setPixelColor(194, 255, 0, 0);  
strip.setPixelColor(203, 255, 0, 0);  
strip.setPixelColor(213, 255, 0, 0);  
strip.setPixelColor(221, 255, 0, 0);  
strip.setPixelColor(226, 255, 0, 0);  
strip.setPixelColor(252, 255, 0, 0);  
strip.setPixelColor(247, 255, 0, 0);  
strip.setPixelColor(233, 255, 0, 0);  
strip.setPixelColor(248, 255, 0, 0);
```

```
strip.setPixelColor(249, 255, 0, 0);  
strip.setPixelColor(250, 255, 0, 0);  
strip.setPixelColor(251, 255, 0, 0);  
strip.show();
```

```
delayMicroseconds(900000);
```

```
strip.setPixelColor(4, 0, 0, 0);  
strip.setPixelColor(5, 0, 0, 0);  
strip.setPixelColor(6, 0, 0, 0);  
strip.setPixelColor(23, 0, 0, 0);  
strip.setPixelColor(7, 0, 0, 0);  
strip.setPixelColor(3, 0, 0, 0);  
strip.setPixelColor(34, 0, 0, 0);  
strip.setPixelColor(29, 0, 0, 0);  
strip.setPixelColor(41, 0, 0, 0);  
strip.setPixelColor(53, 0, 0, 0);  
strip.setPixelColor(61, 0, 0, 0);  
strip.setPixelColor(66, 0, 0, 0);  
strip.setPixelColor(75, 0, 0, 0);  
strip.setPixelColor(92, 0, 0, 0);  
strip.setPixelColor(83, 0, 0, 0);  
strip.setPixelColor(100, 0, 0, 0);  
strip.setPixelColor(109, 0, 0, 0);  
strip.setPixelColor(113, 0, 0, 0);  
strip.setPixelColor(122, 0, 0, 0);  
strip.setPixelColor(133, 0, 0, 0);  
strip.setPixelColor(134, 0, 0, 0);  
strip.setPixelColor(143, 0, 0, 0);  
strip.setPixelColor(145, 0, 0, 0);  
strip.setPixelColor(155, 0, 0, 0);  
strip.setPixelColor(163, 0, 0, 0);  
strip.setPixelColor(173, 0, 0, 0);  
strip.setPixelColor(179, 0, 0, 0);
```

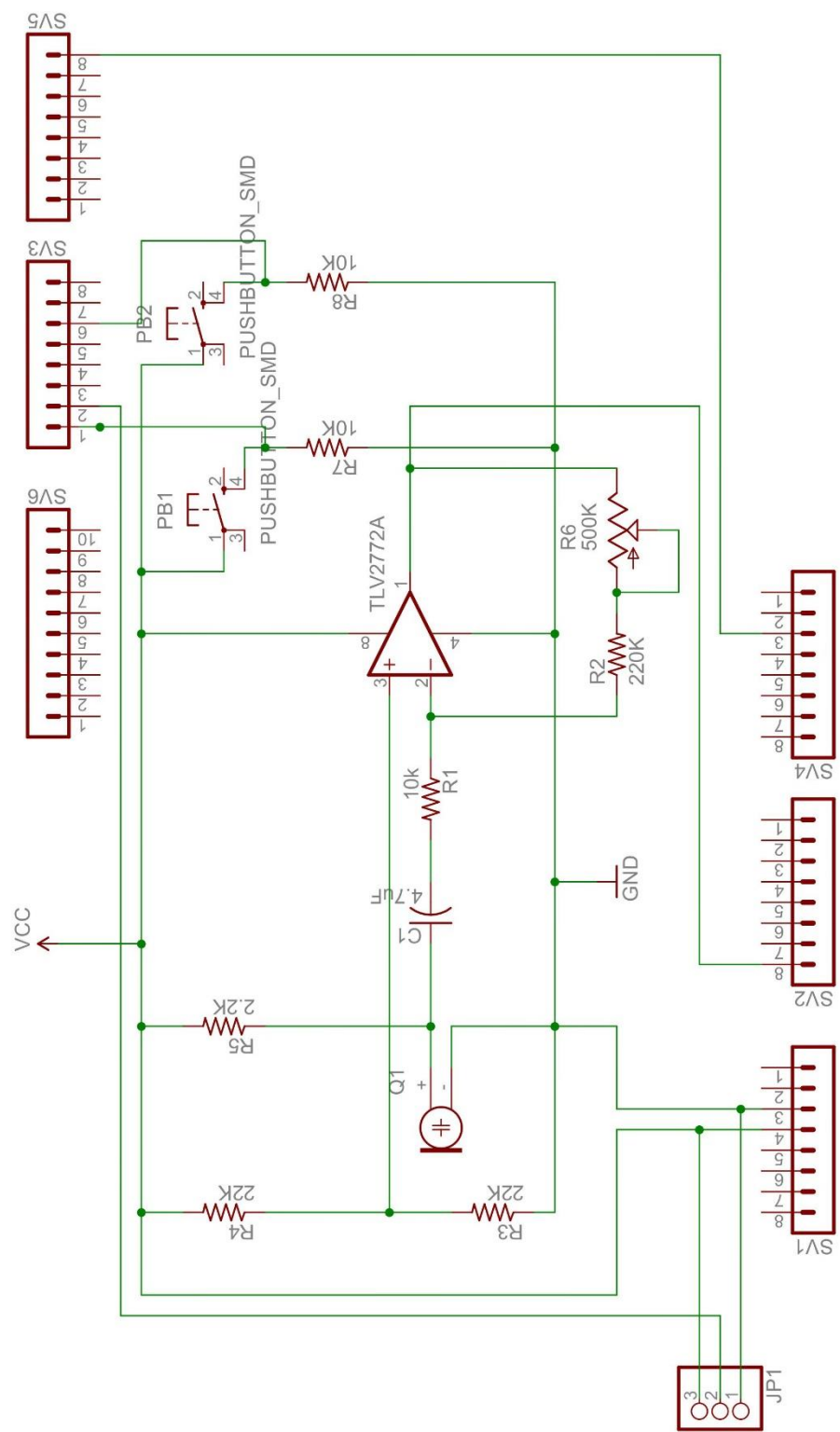
```
strip.setPixelColor(189, 0, 0, 0);  
strip.setPixelColor(194, 0, 0, 0);  
strip.setPixelColor(203, 0, 0, 0);  
strip.setPixelColor(213, 0, 0, 0);  
strip.setPixelColor(221, 0, 0, 0);  
strip.setPixelColor(226, 0, 0, 0);  
strip.setPixelColor(252, 0, 0, 0);  
strip.setPixelColor(247, 0, 0, 0);  
strip.setPixelColor(233, 0, 0, 0);  
strip.setPixelColor(248, 0, 0, 0);  
strip.setPixelColor(249, 0, 0, 0);  
strip.setPixelColor(250, 0, 0, 0);  
strip.setPixelColor(251, 0, 0, 0);  
strip.show();  
}
```

# **ANNEX 3**

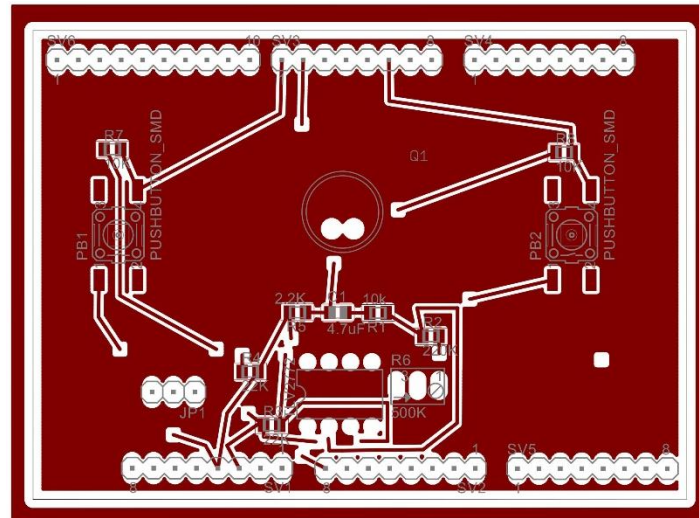
## **VU METER PCB**

---

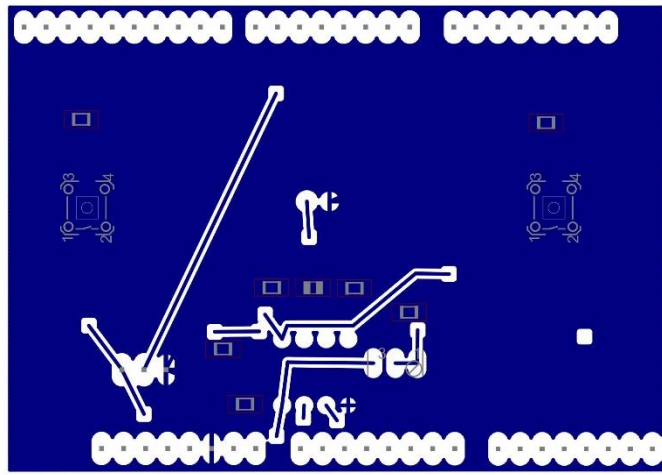
A. SCHEMATIC



## B. TOP LAYER



### C. BOTTOM LAYER

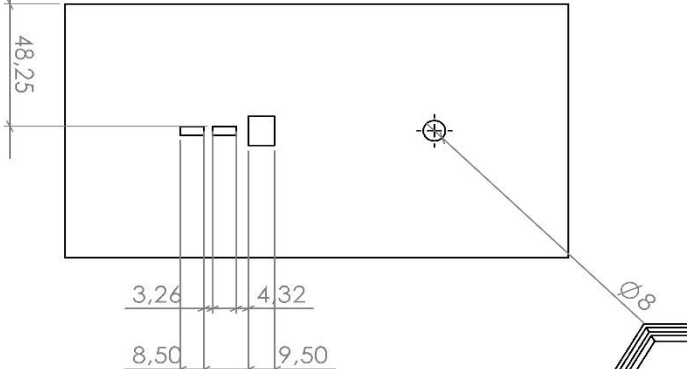
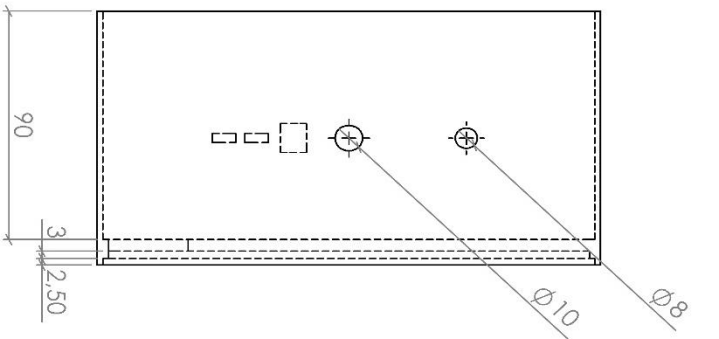
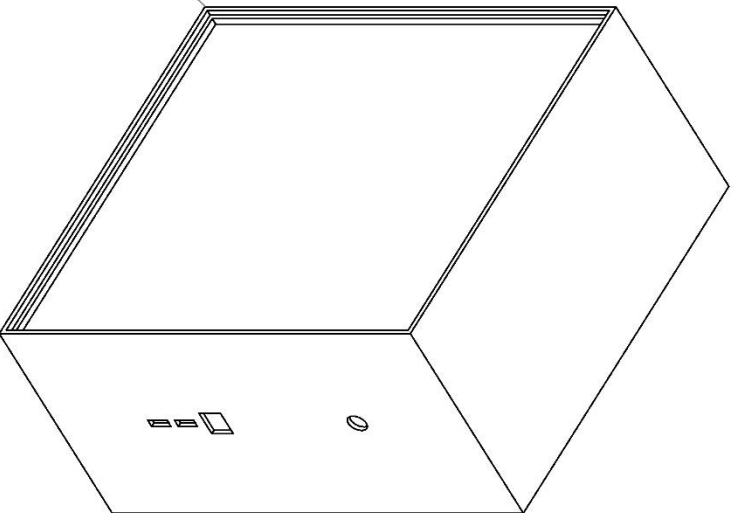
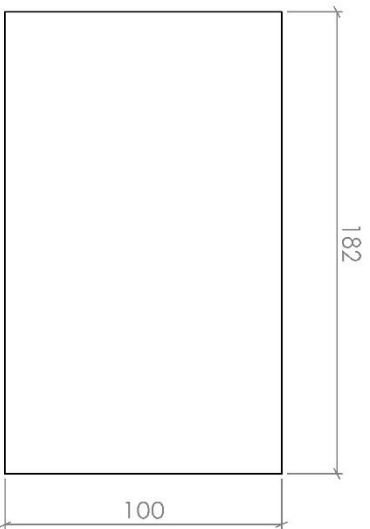


# ANNEX 4

## 3D CAD PLANS

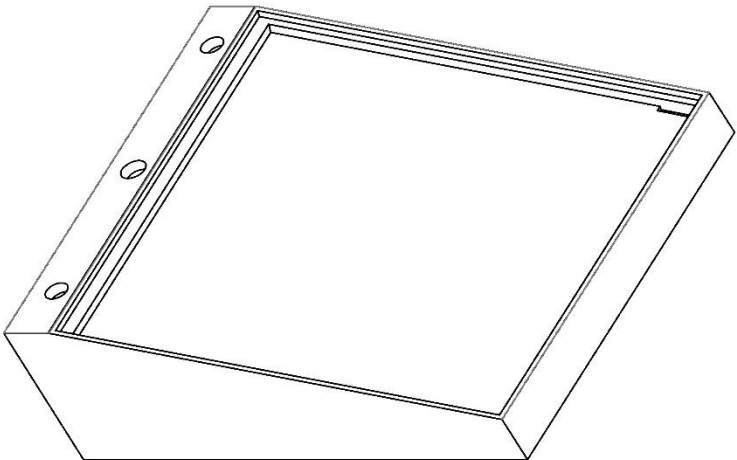
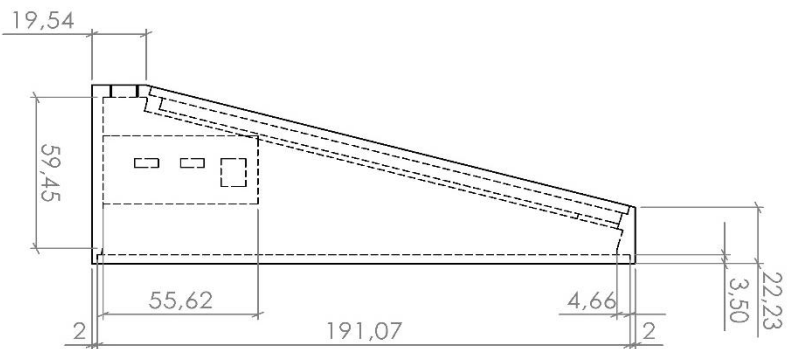
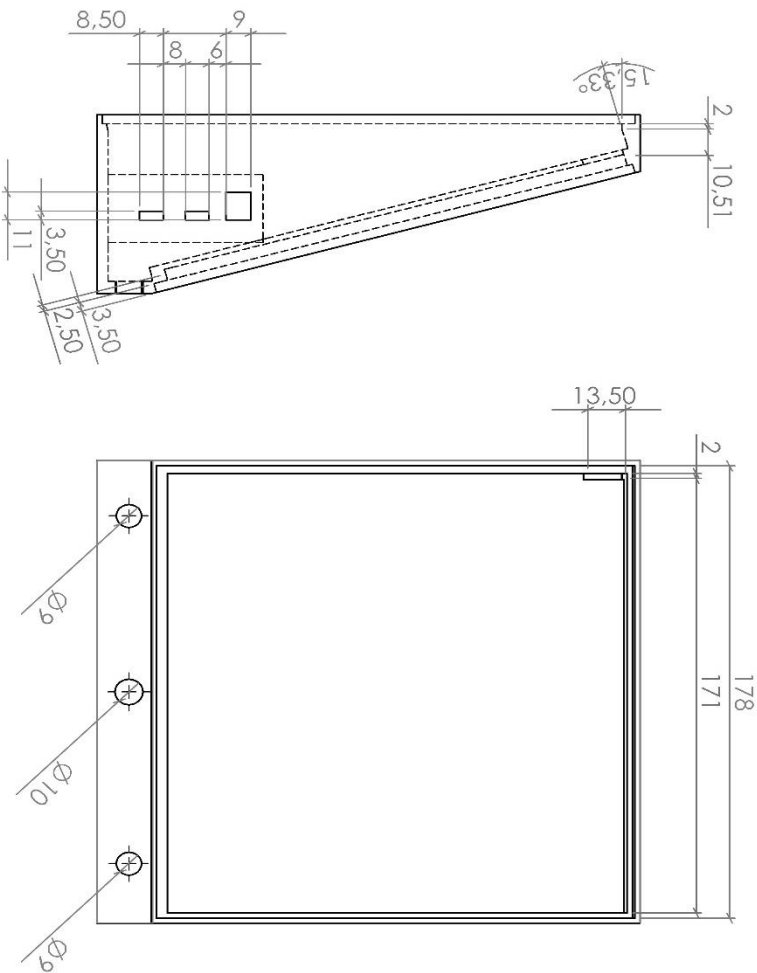
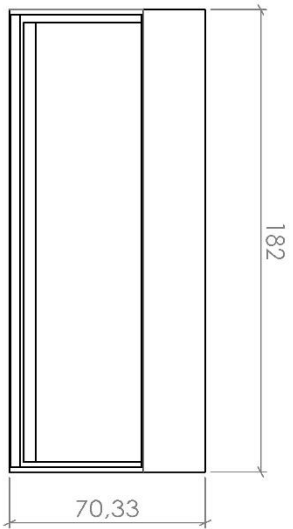
---





SÍ NO SE INDICA LO CONTRARIO: ACABADO: LAS COTAS SE EXPRESAN EN MM				FEEDBACK Y REVISIÓN	
TOLERANCIAS: LINEAL: ANGULAR:				REVISIÓN	
NOMBRE	FECHA	FECHA			
N.º DISEÑO	27.01.2015				
VERIF.					
APROB.					
PASA.					
CAUD.					
MATERIAL: Acrylonitrile butadiene styrene (ABS)					
N.º DE DISEÑO					
1					
ESCALA: 1:2					
HOJA 1 DE 3					
A3					

First Design  
Front view, cross-section and raised view



SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERIOR: INTERIORES: ANTEPIED:				ACABADO:				REPARAR Y COMPLETAR VÍAS			
DELL.	NOMBRE	FECHA	FECHA	DELL.	NOMBRE	FECHA	FECHA	DELL.	NOMBRE	FECHA	FECHA
VERE.				VERE.				VERE.			
ASER.				ASER.				ASER.			
FABR.				FABR.				FABR.			
CALD.				CALD.				CALD.			
				MATERIAL:				N.º DE DIBUJO			
				Acrylonitrile butadiene styrene (ABS)				2			
				ESCALA: 1:2				HOJA 2 DE 3			
								A3			

Second Design  
Front view, cross-section and raised view





