10 Ch Analog Spectrum Analyzer

Analog Analyzer with digital Visualization

10 Channel
PIXEL LEDS
40Hz - 16Khz
Line in
Microphone in
Different modes
Webserver live

Building Manual

M.Donners 13 Dece,ber 2021

Table of Contents

M Donners Document version 1.0 December 2021

1. Disclaimer and safety

I, Mark Donners, The Electronics Engineer, may or may not endorse various Do-It-Yourself (DIY) projects and all DIY projects are purely “at your own risk”. As with any DIY project, unfamiliarity with the tools and process can be dangerous. Posts should be construed as theoretical advice only.

If you are at all uncomfortable or inexperienced working on these projects (especially but not limited to electronics and mechanical), please reconsider doing the job yourself. It is very possible (but not likely) on any DIY project to damage belongings or void your property insurance, create a hazardous condition, or harm or even kill yourself or others.

I will not be held responsible for any injury due to the misuse or misunderstanding of any DIY project.

By using the information provided by me, (Website, YouTube, Facebook and other social media), you agree to indemnify me, affiliates, subsidiaries and their related companies for any and all claims, damages, losses and causes of action arising out of your breach or alleged breach of this agreement(disclaimer).

The materials on this site are distributed “as is” and appear on the site without express or implied warranties of any kind, except those required by the relevant legislation. In particular, I make no warranty as to the accuracy, quality, completeness or applicability of the information provided.

The information provided is for entertainment and promotional purposes only. You may not rely on any information and opinions expressed in it for any other purpose.

Disclaimer short version:

This is a DIY project, use any provided information and/or materials at your own risk! I am not responsible for what you do with it!

2. About this project

This document is related to the 10 channel Analog Spectrum analyser. The analogue back end uses 10 band pass filters to divide the input signal into 10 frequency bands.

A ESP32 controllers reads the amplitude of each band and visualizes it using a Pixel LED Matrix. On top of that, the ESP32 runs a webserver to display a live graph of the current analysis. This enables you to visualize your spectrum analyser on any mobile device or PC within your network.

You can connect your audio signal by using the audio input or you can use the microphone input to connect a small condenser microphone. Although using the microphone will limit the frequency response because of its limitations.

The input sensitivity can by adjusted just like brightness and peak hold time. When it does not receive any input signal, after a while, it will go to fire mode in which some leds/display will light up like a fire.

The PCB can be purchased at my Tindie web shop. The firmware (Arduino Sketch) is open source and you can modify it to your needs.

3. Operation

You can use the microphone in to connect a small condenser microphone or you can connect your audio device to the line input connectors. Although the signal from the microphone is amplified on the PCB, it might not be strong enough. Depending on your microphone, you can adjust resistor R52; decreasing it's value will amplify the signal more. In my prototype I replaced it with a resistor of 0 Ohm (I shorted it). However, when using a different mic, I had to increase it again to 20K. So it all depends on your mic.

Mode button

The mode button has 3 functions:

Short press: change pattern(mode), there are 12 available patterns from which the last one is a fire screensaver.

Fast triple press: The VU meter that is displayed on the top row can be disabled/enabled

Pressed/ hold while booting: This will reset your stored WIFI settings. In case you need to change your WIFI settings or in case your system keeps rebooting, this is where to start!

Select Button

The select button has 3 functions:

Short press: Toggle between line-in and microphone input.

Long press: Press for 3 seconds to toggle "the auto change patterns" mode. When enabled, the pattern that is shown changes every few seconds. Also, when the button is pressed long enough, the Dutch national Flag will be shown. That's how you know you've pressed long enough!

Double press: The direction of the falling peak will change.

Brightness Potmeter

You can use this to adjust the overall brightness of all leds / display. WARNING:Make sure you use a power supply to match the current for the brightness that you set. For sure, the ESP32 onboard regulator cannot handle all leds at full brightness. It is best to use an external powersupply that can handle 4 to 6 A. If you are using the USB cable that is connected to the ESP32, you might end up with a burning sensation coming from the ESP32 Board.

Peak Delay Potmeter

You can use this to adjust the time it takes for a peak to fall down to / rise up from the stack

Sensitivity Potmeter

You can use this to adjust the sensitivity of the input. It's like cranking up the volume for lower signal inputs.

Serial Monitor

The serial monitor is your friend, it displays all info on booting, including your web server IP address.

Webserver

After uploading your sketch and booting for the first time, the ESP32 will serve as an access point. Use your labtop, cellphone etc, to connect to this wifi access point. You will be directed to the webpage of the WIFI manager to setup your WIFI access. It will be remembered on your next boot. If you need to change it, see text 'Mode button'.

When set up properly, you can access the webpage with a live view of your spectrum analyzer. You can use the Serial monitor to find the ip-address, it is shown in the boot log.

4. Tools needed.

You will be working with low voltage and you should know your way around basic electronics. Using the wrong voltage or polarity might not kill you but it will destroy your project in an instance!

Being successful in building this device requires some adequate skills and tools. You should be able to solder small components on a PCB which will require a soldering station with a small tip as you will be handling components like 0805 sized although small, if you have a steady hand, you should be able to solder them onto the board. If this is beyond your possibilities, then you should buy the pcb version that has all SMD components pre-installed.

Furthermore, you will be programming the microcontroller by using the Arduino IDE environment. Although the steps to do so will be described in this document, it is advisable for you to get to know this Arduino software.

And yes, you will need some basic tools like a screwdriver 😉

5. Hardware

5.1. Main PCB

The pcb is available at my Tindie store here:

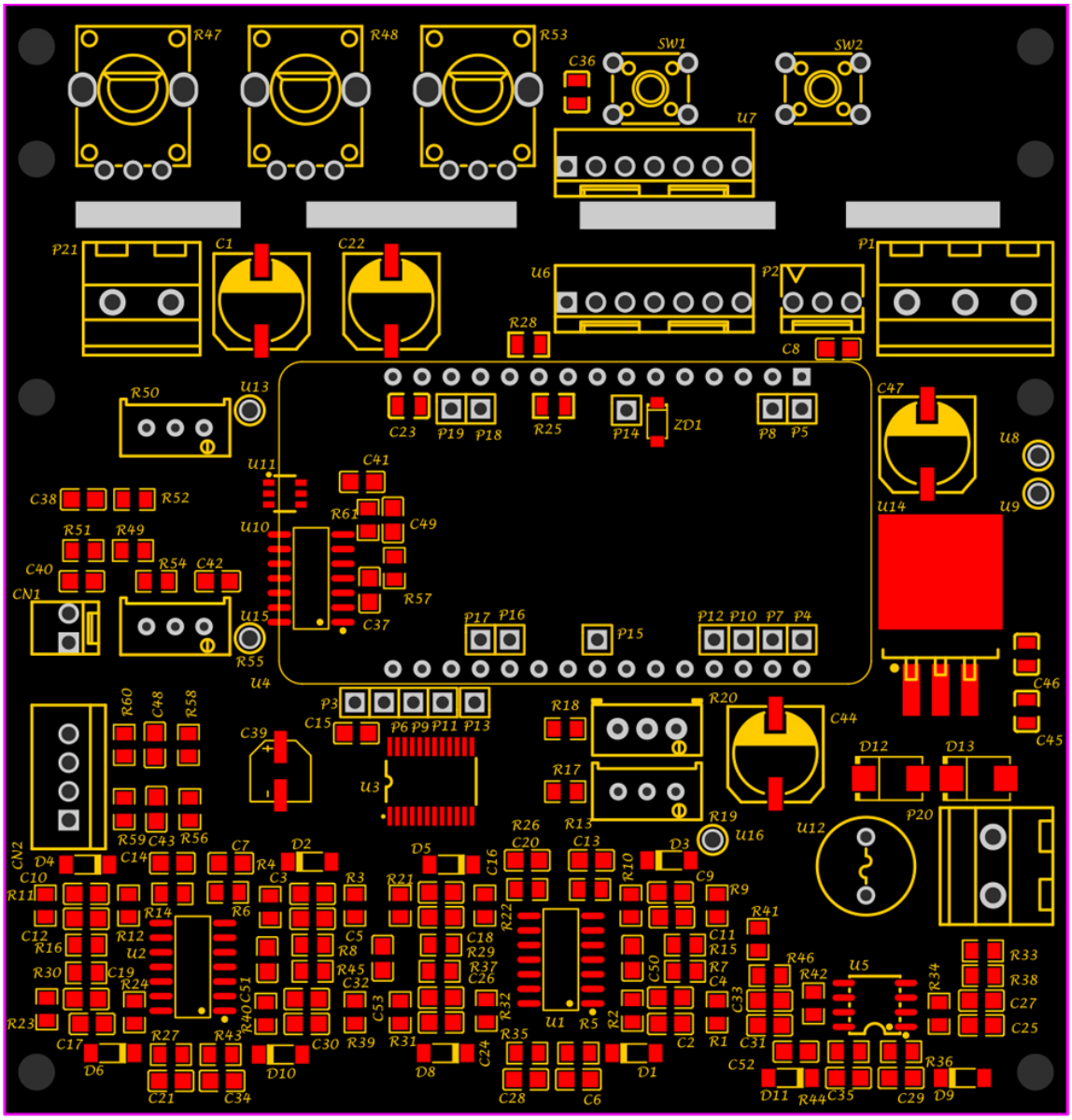
https://www.tindie.com/products/markdonners/pcb-10-channel-analog-spectrum-analyzer/

The PCB has all SMD components pre-installed and all you need to add is the ESP32 board and a hand full of through hole components.

Shipping is possible within the EU and some other global destinations. Take note that shipping outside the EU involves import Tax, declarations of conformity and other complicated stuff. If you are a resident of a country that I don't ship to, you can always order a pcb directly from a PCB supplier near you and assemble one yourself. The Gerber productions files you need for that are available for download, although it might be more expensive than what you are hoping for. If you need 1 or 2 PCB's, a Tindie order is the cheapest way to go. You can always file a "Ship to my country" request on Tindie and I'll see what I can do.

My shipping policy might change over time so always check the Tindie store to see if I am shipping to your country. ADDITION: I have successfully shipped to UK, USA and other non-EU countries but handling the tax, import fee and so on is on you! Check out the Tindie store to see what is possible for your country.

5.1.1. Assembly



Assembly notes:

If you didn't buy the PCB with pre-assembled SMD components then you'll have to solder those on yourself. It's possible when you use the right tools but let me remind you that soldering SMD components in this scale is not for beginners

Here are some links for some of the used components. A more detailed list of components is available on one of the following pages of this document.

Microphone
A used a simple electret microphone like this one:

https://nl.aliexpress.com/item/32961327636.html?spm=a2g0o.productlist.0.0.40156749GH0jMh&algo_pvid=b3873d4e-2d66-4844-b423-45a81e07bc15&algo_expid=b3873d4e-2d66-4844-b423-45a81e07bc15-0&btsid=2100bdd516132465203027466e5419&ws_ab_test=searchweb0_0,searchweb201602_,searchweb201603_

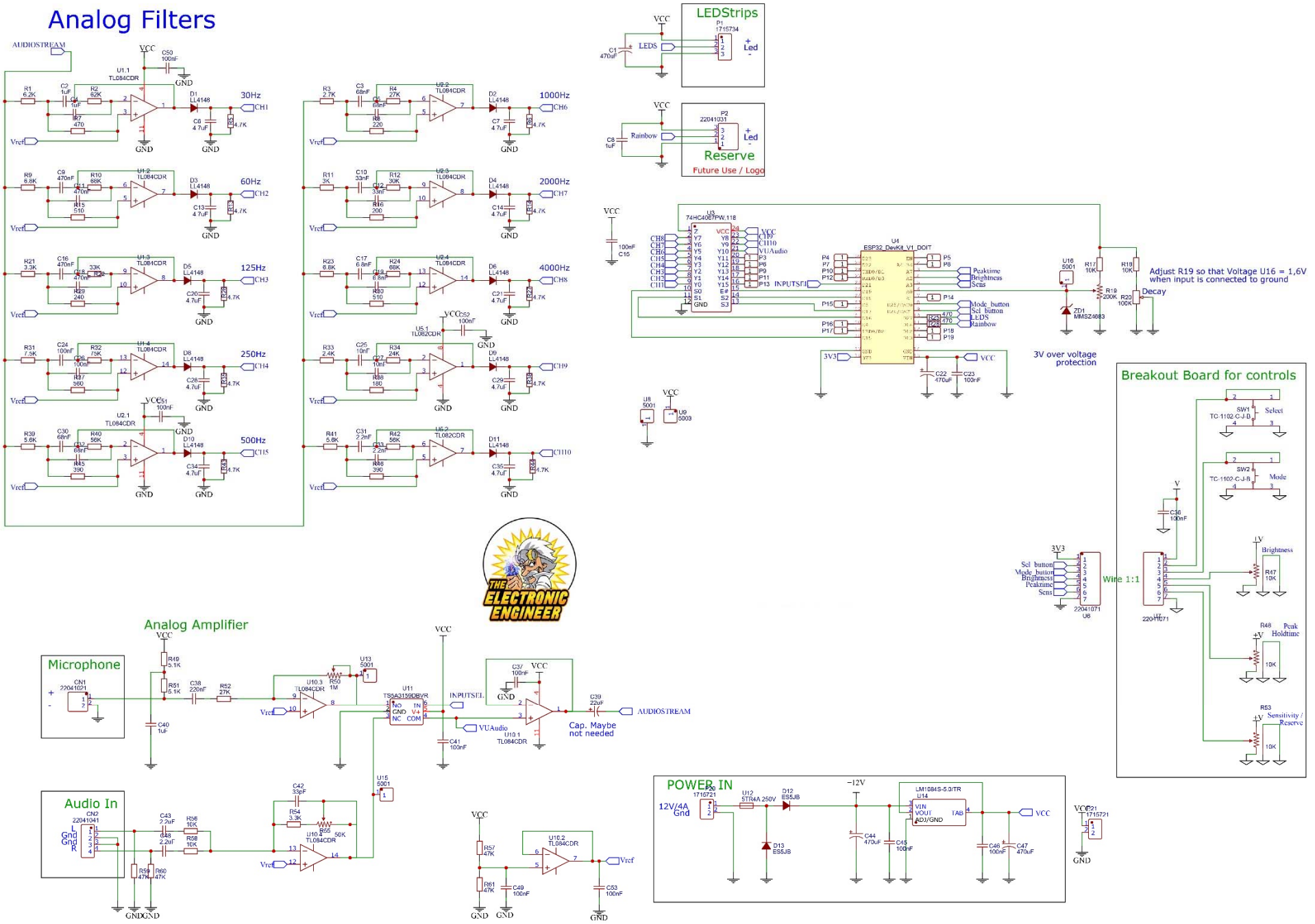
Power Supply
You will be needing a power supply of DC 12V that can handle 4 to 6A. (Current depends on the number of panels or leds you are planning to connect. Make sure your power supply is stable and doesn't have noise on the output.

You'll also need some connectors for audio and power and some wire to connect all. Regarding the power to the leds, use a wire that can handle a current of 4A Depending on what you planning, thinner wire can do the job as well.

Pixel LEDS / LEDstrip
Also, you'll need to order a few meters of Pixel Ledstrip WS2812. Make sure you order the one that works with your design. (number of pixels per meter varies) I used 74Leds/meter version.

You could use Pixelled matrix as well but depending on how it is wired (zig zag or not) you might have to change the code in LEDDRIVER.h

5.1.2. Schematic



5.1.3. PCB Part list main PCB

The complete part list for the main PCB is here:

ID	Name	Designator	Footprint	Quantity	Manufacturer Part	Manufacturer	Supplier	Supplier Part
1	5001	U16,U13,U15,U8	TEST-TH_BD2.54-P1.39	4	5001	Keystone	LCSC	C238122
2	470	R25,R28,R7	R0805	3	0805W8F4700T5E	UniOhm	LCSC	C17710
3	22041071	U7,U6	CONN-TH_22041071	2	22041071	MOLEX	LCSC	C293439
4	100nF	C53,C37,C26,C24,C49,C41,C15,C50,C51,C52,C46,C45,C36,C23	C0805	14	CC0805KRX7R9BB104	YAGEO	LCSC	C49678
5	1715734	P1	CONN-TH_3P-P5.08_1715734	1	1715734	Phoenix Contact	LCSC	C480520
6	1715721	P20,P21	CONN-TH_P5.08_1715721	2	1715721	Phoenix Contact	LCSC	C480516
7	TL084CDR	U10,U2,U1	SOIC-14_L8.7-W3.9-P1.27-LS6.0-BL	3	TL084CDR	TI	LCSC	C8956
8	TL082CDR	U5	SOIC-8_L5.0-W4.0-P1.27-LS6.0-BL	1	TL082CDR	TI	LCSC	C9385
9	5TR4A 250V	U12	FUSE-TH_BD8.5-P5.08-D1.0	1	5TR4A 250V	XC Elec(Shenzhen)	LCSC	C140489
10	TS5A3159DBVR	U11	SOT-23-6_L2.9-W1.6-P0.95-LS2.8-BR	1	TS5A3159DBVR	TI(Tex as Instruments)	LCSC	C92485
11	5003	U9	TEST-TH_BD2.54-P1.39	1	5003	Keystone	LCSC	C238124
12	74HC4067PW,118	U3	TSSOP-24_L7.8-W4.4-P0.65-LS6.4-BL	1	74HC4067PW,118	Nexperia	LCSC	C179326
13	TC-1102-C-J-B	SW2,SW1	KEY-TH_4P-L6.0-W6.0-P4.50-LS6.5	2	TC-1102-C-J-B	XKB Enterprise	LCSC	C381016
14	1M	R50	RES-ADJ-TH_3P-L10.0-W10.0-P2.50-BL-BS	1	3296W-1-105	Chengdu Guosheng Tech	LCSC	C118944
15	27K	R52,R4 (R52 might need to be changed, see text)	R0805	2	0805W8F2702T5E	UniOhm	LCSC	C17593
16	5.1K	R51,R49	R0805	2	0805W8F5101T5E	UniOhm	LCSC	C27834
17	10K	R48,R47,R53	RES-TH_RK09D1130C2P	3	RK09D1130C2P	ALPS Electric	LCSC	C361173
18	100K	R20	RES-ADJ-TH_3P-L9.5-W4.9-P2.50-L_3296W	1	3296W-1-104	Chengdu Guosheng Tech	LCSC	C118963
19	50K	R55	RES-ADJ-TH_3P-L9.5-W4.85-P2.50-BL-BS	1	3296W-1-503	Chengdu Guosheng Tech	LCSC	C118911
20	200K	R19	RES-ADJ-TH_3P-L9.5-W4.85-P2.50-BL-BS	1	3296W-1-204	Chengdu Guosheng Tech	LCSC	C118942
21	47K	R61,R60,R59,R57	R0805	4	0805W8F4702T5E	UniOhm	LCSC	C17713
22	10K	R58,R56,R18,R17	R0805	4	0805W8F1002T5E	UniOhm	LCSC	C17414
23	3.3K	R54,R21	R0805	2	0805W8F3301T5E	UniOhm	LCSC	C26010
24	390	R46,R45	R0805	2	0805W8F3900T5E	UniOhm	LCSC	C17655
25	4.7K	R44,R43,R36,R35,R27,R26,R14,R13,R6,R5	R0805	10	0805W8F4701T5E	UniOhm	LCSC	C17673
26	56K	R42,R40	R0805	2	0805W8F5602T5E	UniOhm	LCSC	C17756
27	5.6K	R41,R39	R0805	2	0805W8F5601T5E	UniOhm	LCSC	C4382
28	180	R38	R0805	1	0805W8F1800T5E	UniOhm	LCSC	C25270
29	560	R37	R0805	1	0805W8F5600T5E	UniOhm	LCSC	C28636
30	24K	R34	R0805	1	0805W8F2402T5E	UniOhm	LCSC	C17575
31	2.4K	R33	R0805	1	0805W8F2401T5E	UniOhm	LCSC	C17526
32	75K	R32	R0805	1	0805W8F7502T5E	UniOhm	LCSC	C17819
33	7.5K	R31	R0805	1	0805W8F7501T5E	UniOhm	LCSC	C17807
34	510	R30,R15	R0805	2	0805W8F5100T5E	UniOhm	LCSC	C17734

35	240	R29	R0805	1	0805W8F2400T5E	UniOhm	LCSC	C17572
36	68K	R24,R10	R0805	2	0805W8F6802T5E	UniOhm	LCSC	C17801
37	6.8K	R23,R9	R0805	2	0805W8F6801T5E	UniOhm	LCSC	C17772
38	33K	R22	R0805	1	0805W8F3302T5E	UniOhm	LCSC	C17633
39	200	R16	R0805	1	0805W8F2000T5E	UniOhm	LCSC	C17540
40	30K	R12	R0805	1	0805W8F3002T5E	UniOhm	LCSC	C17621
41	3K	R11	R0805	1	0805W8F3001T5E	UniOhm	LCSC	C17661
42	220	R8	R0805	1	0805W8F2200T5E	UniOhm	LCSC	C17557
43	2.7K	R3	R0805	1	0805W8F2701T5E	UniOhm	LCSC	C17530
44	62K	R2	R0805	1	0805W8F6202T5E	UniOhm	LCSC	C17783
45	6.2K	R1	R0805	1	0805W8F6201T5E	UniOhm	LCSC	C17767
46	Header-Male-2.54_1x1	P19,P18,P17,P16,P15,P14,P13,P12,P11,P10,P9,P8,P7,P6,P5,P4,P3	HDR-TH_1P-P2.54-V-M	17	Header-Male-2.54_1x1	ReliaPro	LCSC	C81276
47	22041031	P2	CONN-TH_3P-P2.54_22041031	1	22041031	MOLEX	LCSC	C293437
48	ES5JB	D13,D12	SMB_L4.6-W3.6-LS5.3-RD	2	ES5JB	Shandong Jingdao Microelectronics	LCSC	C123908
49	LL4148	D11,D10,D9,D8,D6,D5,D4,D3,D2,D1	LL-34_L3.5-W1.5-RD-1	10	LL4148	SEMTECH	LCSC	C9808
50	22041041	CN2	CONN-TH_22041041	1	22041041	MOLEX	LCSC	C185196
51	22041021	CN1	CONN-TH_22041021	1	22041021	MOLEX	LCSC	C185215
52	22uF	C39	CAP-SMD_BD5.0-L5.3-W5.3-FD	1	RVT1E220M0505	HONOR	LCSC	C15848
53	1uF	C40,C8,C4,C2	C0805	4	CL21B105KBFNNNE	SAMSUNG	LCSC	C28323
54	220nF	C38	C0805	1	CL21B224KBFNNNE	SAMSUNG	LCSC	C5378
55	2.2uF	C48,C43	C0805	2	0805F225M500NT	FH	LCSC	C49217
56	33pF	C42	C0805	1	CL21C330JBANNNC	SAMSUNG	LCSC	C1814
57	4.7uF	C35,C34,C29,C28,C21,C20,C14,C13,C7,C6	C0805	10	CL21A475KAQNNNE	SAMSUNG	LCSC	C1779
58	2.2nF	C33,C31	C0805	2	CL21C222JBFNNNE	SAMSUNG	LCSC	C28260
59	68nF	C32,C5,C3,C30	C0805	4	0805B683K500NT	FH	LCSC	C1756
60	10nF	C27,C25	C0805	2	CL21B103KBANNNC	SAMSUNG	LCSC	C1710
61	6.8nF	C19,C17	C0805	2	0805B682K500NT	FH	LCSC	C1755
62	470nF	C18,C16,C11,C9	C0805	4	CL21B474KBFNNNE	SAMSUNG	LCSC	C13967
63	33nF	C12,C10	C0805	2	0805B333K500NT	FH	LCSC	C1739
64	470uF	C47,C1,C44,C22	CAP-SMD_BD8.0-L8.3-W8.3-RD	4	VZH471M1CTR-0810	LELON	LCSC	C164069
65	MMSZ4683	ZD1	SOD-123_L2.8-W1.8-LS3.7-RD	1	MMSZ4683	CJ	LCSC	C21512
66	LM1084S-5.0/TR	U14	TO-263-3_L8.6-W10.2-P2.54-LS14.4-TL	1	LM1084S-5.0/TR	HGSEMI	LCSC	C259973
67	ESP32_DevKit_V1_DOIT	U4	ESP32_DEVKIT_V1_DOIT	1				

REMEMBER: MOST COMPONENTS ARE STANDARD COMPONENTS AND CAN BE REPLACED BY OTHER TYPE/ BRAND ETC. USE THIS LIST AS A GUIDE NOT AS AN ABSOLUTE MUST!

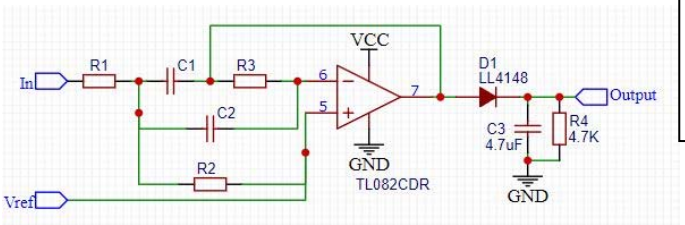
5.2. Electronics explained.

The Electronics can be divided into several sections. Of course, there is a power supply and some standard connectors for audio input and power and there are a few standard switches. Nothing scary about that at all.

Connector P20 is used to connect a 12V power supply. The circuit is protected by a fuse and two power diodes. If the power is connected reversed, D13 will conduct so that the current will rise beyond the capabilities of the fuse. While D12 will protect the circuit from this reversed voltage, the fuse will blow. However, under normal circumstances, the power source is connected correctly and while passing a few capacitors to smooth out any power supply noise or instabilities, the 12V is offered to the onboard regulator U14. U14 converts the 12V power to 5V and feeds the ESP32 board and some of our onboard features like the pre-amplifier, the bandfilters and the led strip.

The user interface consists of 3 potentiometers and two switches. The potmeters control Sensitivity, brightness and Peak time delay while the switches are used to set mode, input and auto-change mod and more.

The PCB has room for a dual pre-amplifier. One can be used to connect a microphone and the other can be used for the line in. You can adjust the output of each pre-amplifier by adjusting the potmeter R50 (for microphone) or R55 for Line-in. The output of both amplifier goes to an analog switch U11. You can use the select button to toggle between the inputs. The output of the Analog switch is buffered by U10.1. It's the output (AUDIOSTREAM) of this buffer that feeds all the bandfilters. Each bandfilter has a specific band frequency. The output signal of each bandfilter is rectified by a diode and smoothened by a capacitor.

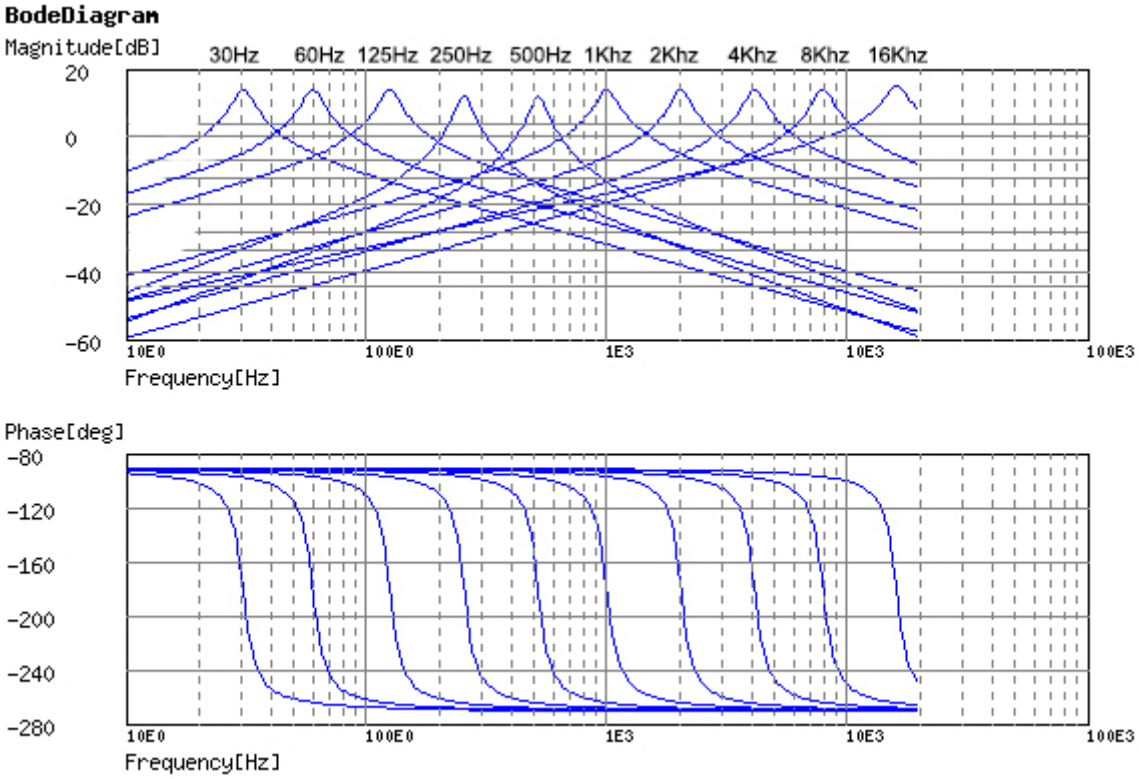


To calculate, I used this tool:

http://sim.okawa-denshi.jp/en/OPtazyuLowkeisan.htm

R1	R2	R3	C1 [uF]	C2 [uF]		Gain		Fcentre [Hz]	Optimum [Hz]	Deviation [Hz]	Deviation %	Q
6200	470	62000	1	1		-5		31	30	1	1,93	5,96
6800	510	68000	0,47	0,47		-5		60	60	0	-0,63	5,99
3300	240	33000	0,47	0,47		-5		125	125	0	-0,30	6,07
7500	560	75000	0,1	0,1		-5		255	250	5	1,83	6,00
5600	390	56000	0,068	0,068		-5		518	500	18	3,59	6,20
2700	220	27000	0,068	0,068		-5		999	1000	-1	-0,13	5,76
3000	200	30000	0,033	0,033		-5		2034	2000	34	1,68	6,32
6800	510	68000	0,0068	0,0068		-5		4121	4000	121	3,02	5,99
2400	180	24000	0,01	0,01		-5		7939	8000	-61	-0,76	5,99
5600	390	56000	0,0022	0,0022		-5		16010	16000	10	0,06	6,20

There is a Bode Diagram on the next page



All the output signals of the bandfilters are connected to an analog multiplexor U3. In the software you can select what channel you want to connect to the the resistor network R17/R19 R18/R20. The diode ZD1 is there to protect the ESP32 from over voltage. R19 is used to adjust the amplitude of the signal offered to the ESP32 while R20 can be used to delay the decay of the output signal, resulting in a slower/faster response.

The multiplexor is also connected to the output of the analog switch. This can be used to measure the audio level of the input signal or to do a FFT analysis of the input signal.

Last but not least, there is U10.2. This is used to create the Vref signal. The amplitude of Vref is VCC/2 so that should be around 2,5V

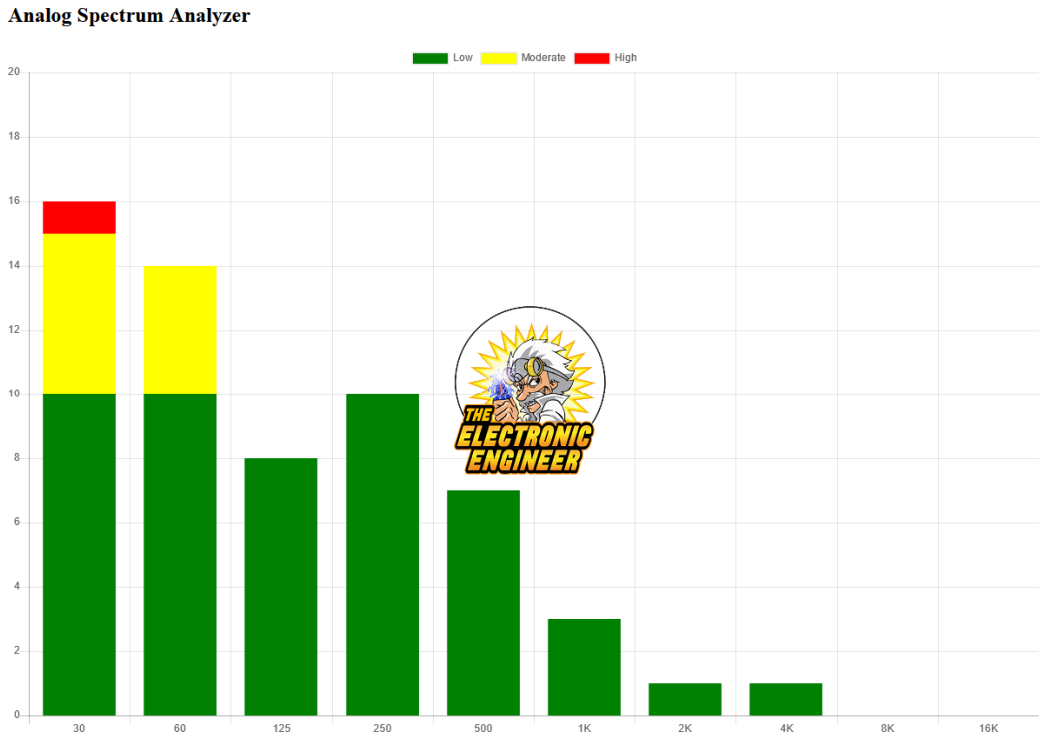
5.3. Visualisation

The visualization of the bandfilters is done by the ESP32, driving a string of LEDS. I used WS2812 based Ledstrip that I wired in ZIG ZAG. You can also use a ready made led matrix but you'll have to adjust the directions of the led accordingly .

LEDDRIVER.H

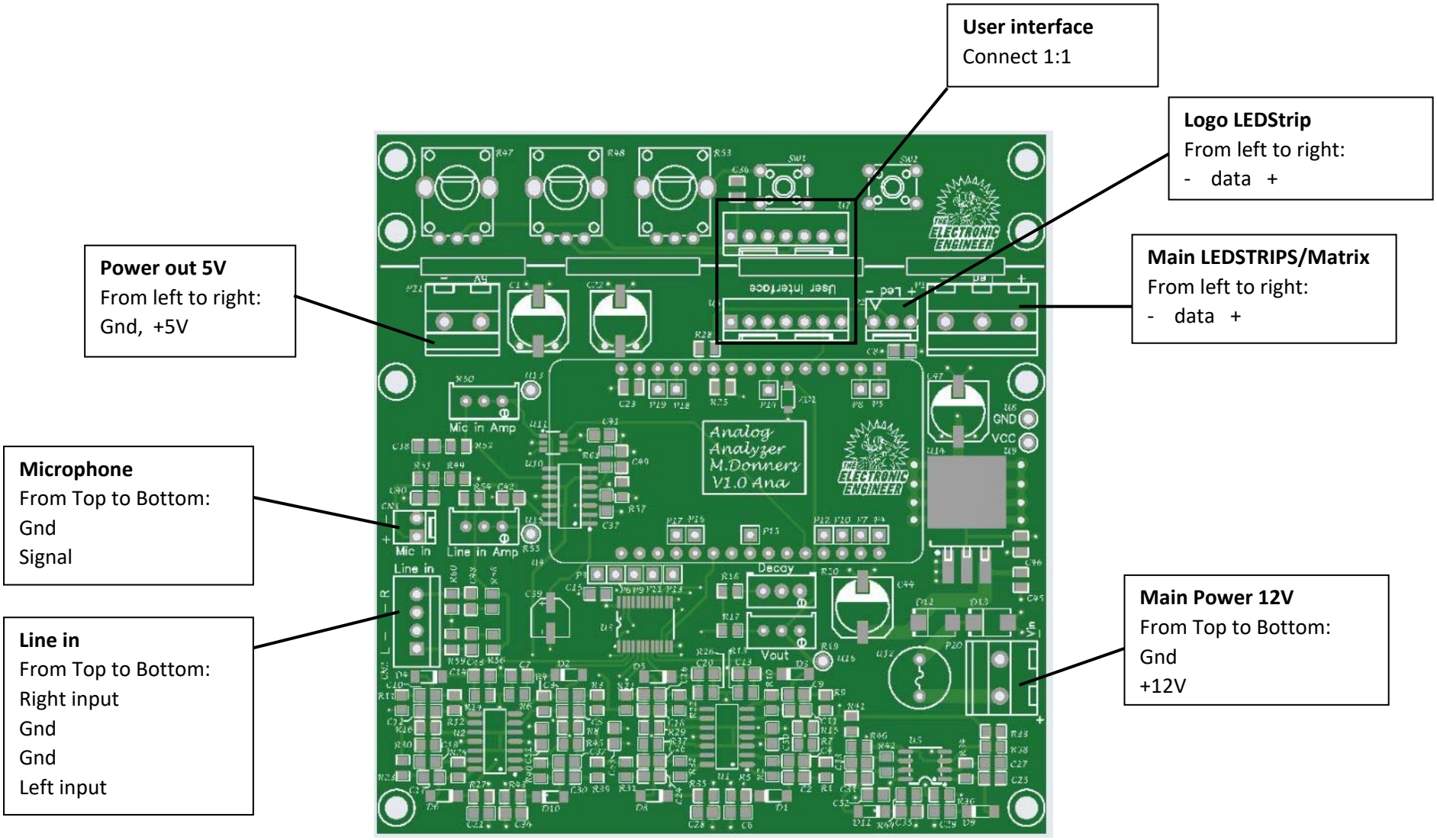
```
FastLED_NeoMatrix *matrix = new FastLED_NeoMatrix(leds, kMatrixWidth, kMatrixHeight,
NEO_MATRIX_BOTTOM     + NEO_MATRIX_LEFT +
NEO_MATRIX_COLUMNS     + NEO_MATRIX_PROGRESSIVE +
NEO_TILE_TOP + NEO_TILE_LEFT + NEO_TILE_ROWS);
```

The ESP32 also runs a webinterface that shows a live display of the analysis. In short, you can use any webbrowser as a display.



5.4. Wiring tables

The wiring is not that spectacular. I used shielded wire to connect the microphone and the audio input and I used some general wire for everything else. Give some extra attention to the power lines that feed the LED Strips. It you are using more then 200 leds, you might want to feed power to each en every strip. I used 180 leds so I decided to power up the first strip and hook it up to the next etc. (series).



6. Software

The software uses both cores of the ESP32. Core 0 is used for running/processing the webserver and updating the rainbow appearance of the Logo ledstrip. Core 1 is used for everything else.

6.1. Core 1

You can find the code for core 0 in function Task1code(void * pvParameters)

It does nothing more then run a loop, waiting for the webtoken to be set, indicating that new data is waiting. If there is data, it will be processed on the HTML page of the webserver. Also, every 50 milliseconds, the logo is updated with a new color.

Now, that was everything for core 0. It doesn't look like much, but I put it there for a reason. Whenever the webserver experiences a wifi signal lost, (for example because your hand is covering the circuit when operating), the wm.process() function would wait for the signal to recover. When that happens everything else will freeze because the program is waiting and waiting…. Now, by moving this function to another core, separated from the main program, the freeze will still happen but it will only effect core 0. When the Wifi is stagnant, the webserver HTML page will freeze but the visualization of the pixelleds will continue.

6.2. Core 2

The software running on core two can be divided into several functions. The most important once, are documented here to help you understand how the program works.

6.2.1. Setup()

This runs only on startup and it does some initialization.

Purpose: create a task that will be executed in the Task1code() function, with priority 1 and executed on core 0 this will run the webinterface datatransfer.
xTaskCreatePinnedToCore(Task1code, /* Task function. */ "WebserverTask", /* name of task. */ 10000, /* Stack size of task */ NULL, /* parameter of the task */ 4, /* priority of the task */ &WebserverTask, /* Task handle to keep track of created task */ 0); /* pin task to core 0 */
This is needed to take control of what task is running on what core. In this case I assigned the Tak1Code{} function to core 0. By default, the main loop is running on core 1

Serial.begin(115200); → This will initiate the serial port for debugging and feedback. This is followed by some first debugging information like what core the main program is running on, version etc.

Purpose: Setup of input buttons

```
 ModeBut.begin();
 SelBut.begin();

 // Attach callback to buttons
 ModeBut.onPressed(onPressed);
 SelBut.onPressed(onPressed2);
 SelBut.onPressedFor(LONG_PRESS_MS, startAutoMode);
 ModeBut.onSequence(3, 600, VUMeterToggle);
 SelBut.onSequence(2, 500, ChangePeakDirection);
```

This is related to code outside the setup loop, in fact, I placed it right above:

```
// Two buttons
EasyButton ModeBut(MODE_BUTTON_PIN);
EasyButton SelBut(SELECT_BUTTON_PIN);

// Mode button 1 short press
void onPressed() {
 Serial.println("Mode Button has been pressed!");
 buttonPushCounter = (buttonPushCounter + 1) % NumberOfModes;
 FastLED.clear();
}
// Select button 1 short press
void onPressed2() {
 Serial.println("Select Button has been pressed!");
 InputLine = !InputLine;
 SetInput(InputLine);
 Serial.printf("inputline set to %d", InputLine);
}
//called when mode button is pressed 3 times within 600ms
void VUMeterToggle() {
 VUMeter = !VUMeter;
 Serial.println("Select Button has been pressed 3x!");
}
//called when select button is pressed 2 times within 500ms
void ChangePeakDirection() {
 PeakDirection = !PeakDirection;
}
```

It attaches a functions to each button, depending on the use..single press, long press , double press etc. These library functions are form #include <EasyButton.h>

Next, I defined some critical output pins as being output. The actual pin definition is in the settings.h file

Purpose: Reset Wifi stored settings if needed, load html webpage and start the webserver

```
if (digitalRead(MODE_BUTTON_PIN) == 0) {
   Serial.println("button pressed on startup, WIFI settings will be reset");
   wm.resetSettings();
 }
 //Try to connect WiFi, then create AP but if no success then don't block the program
 wm.setConfigPortalBlocking(false);
 wm.autoConnect("ESP32_AP", "");

 // this will load the actual html webpage to be displayed
 server.on("/", []() { server.send_P(200, "text/html", webpage); });

 // now start the server
 server.begin();
 webSocket.begin();
 webSocket.onEvent(webSocketEvent);
```

If mode button is pressed and hold during startup, the Wifi manager will be called to reset the wifi settings. If you decide not to setup wifi, you can skip it and the system will run without wifi.

To complete the setup() function, I call SetupLEDSTRIP() and SetInput(InputLine)

Purpose: Setup the LEDstrip / Matrix

```
void SetupLEDSTRIP(void) {
  FastLED.addLeds<CHIPSET, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS).setCorrection(TypicalSMD5050);
  FastLED.addLeds<CHIPSET, LED_PIN_LOGO, COLOR_ORDER>(Logo, N_PIXELS_LOGO).setCorrection(TypicalSMD5050);
  FastLED.setMaxPowerInVoltsAndMilliamps(LED_VOLTS, MAX_MILLIAMPS);
  FastLED.setBrightness(BRIGHTNESS);
  FastLED.clear();
}
```

You can change Chipset, led pin color order etc. in the settings.h file

Purpose: Setup the input line, microphone or line-in

```
// inputSelect  use: SetInput(MIC) or SetInput(LINE)
void SetInput(int inputpin) {
  digitalWrite(InputSelPin, inputpin);
  if (inputpin==MIC)noise=noisemic;
  else noise= noiseline;
}
```

6.2.2. Void loop()

The main loop can be devided into tree global goals:

1. Handle user interface
2. Get the latest analog signals from each band filter and store each value in a frequency bin.
3. Do some filtering to smooth out the bins
4. Process the VU meter value
5. Process the bin values into bar height.
6. Move Peak to new position
7. Check for input, if no input go to firemode. If there is signal, wake up if needed
8. Call the actual routines to display the LED patterns. For each mode, there is a different routine

After that, some overhead is handled:

Purpose: display new peak position, rise or fall if needed

```
EVERY_N_MILLISECONDS(Fallingspeed) {
  for (byte band = 0; band < numBands; band++) {
   if (PeakFlag[band] == 1) {
    PeakTimer[band]++;
    if (PeakTimer[band] > PEAKDELAY) {
     PeakTimer[band] = 0;
     PeakFlag[band] = 0;
    }
   }
   else if ((peak[band] > 0) && (PeakDirection == up)) {
    peak[band] += 1;
    if (peak[band] > (kMatrixHeight + 10))peak[band] = 0;
   } // when to far off screen then reset peak height
   else if ((peak[band] > 0) && (PeakDirection == down)) {
    peak[band] -= 1;
   }
  }
  colorTimer++;
 }
```

This is based on number of milliseconds as set in the parameter Fallingspeed in settings.h. On the set interval, the peak will fall or rise, depending on the new signal compared to the old.

Purpose: Setup the LEDstrip / Matrix

```
EVERY_N_SECONDS(SecToChangePattern) {
  if (autoChangePatterns) {
   buttonPushCounter = (buttonPushCounter + 1) % 12;
  }
 }
```

This is based on number of milliseconds as set in the parameter SecToChangePattern in settings.h. On the set interval, the next pattern will be shown if autoChangePatterns is enabled

FastLED.show() will enable the new Led pattern to be shown on the ‘led display’.

Webtoken = true → this is important flag. It is set just before the whole thing (loop) repeats and it is used by core 0. Core 0 also runs in a loop but it repeats a lot more often then the loop in core 1. However, core 0 will only update the webdata is this flag is set. After sending the data, core0 will reset this flag.

Then, the whole thing repeats.

Step 1: Handle user interface

```
// read potmeters and process
PEAKDELAY = map(analogRead(PEAKDELAYPOT), 0, 4095, 100, Fallingspeed);
BRIGHTNESS = map(analogRead(BRIGHTNESSPOT), 0, 4095, BRIGHTNESSMAX, BRIGHTNESSMIN);
SENSE = map(analogRead(SENSEPOT), 0, 4095, 1, InputBoost);

FastLED.setBrightness(BRIGHTNESS);
// Continuously update the button state.
ModeBut.read();
SelBut.read();
```

This does a few things: read the potmeters, map the values to a useable range and store in PEAKDELAY, BRIGHTNESS and SENSE. Also the brightness for the FastLED is applied here. The status of the buttons is read and processed.

Step 2: Get the latest analog signals from each band filter and store each value in a frequency bin.

```
Analyser_ALL(); // call function to read all channels to fill FreqBinsNew[]
```

The actual function is this:

```
void Analyser_ALL() {
 int tempADC = 0;
 for (int channelcnt = 0; channelcnt < numBands; channelcnt++) {
  digitalWrite(MP_E, 1); // multiplex disabled
  digitalWrite(S0, channelcnt & 0b0001);
  digitalWrite(S1, channelcnt & 0b0010);
  digitalWrite(S2, channelcnt & 0b0100);
  digitalWrite(S3, channelcnt & 0b1000);
  digitalWrite(MP_E, 0); // Multiplex enabled
  tempADC = analogRead(ADC_IN);
  FreqBinsNew[channelcnt] = max(tempADC - ADC_OFFSET - noise, 1); // remove offset
 }
 // now read the audio level for VU
 digitalWrite(MP_E, 1); // multiplex disabled
 digitalWrite(S0, 10 & 0b0001);
 digitalWrite(S1, 10 & 0b0010);
 digitalWrite(S2, 10 & 0b0100);
 digitalWrite(S3, 10 & 0b1000);
 digitalWrite(MP_E, 0); // Multiplex enabled
 tempADC = analogRead(ADC_IN);
 VUadc = abs(VU_OFFSET - tempADC);
 VUadc = max(VUadc - noise, 0);
}
```

The function Analyser_ALL is called to read the analog value on each Band filter. Also, the VU meter value is determined by finding the highest value of all bands.

Step 3: Do some filtering to smooth out the bins

```
 float averageSum = 0;
 for (int cnt = 0; cnt < numBands; cnt++) {
  if (FreqBinsNew[cnt] < FreqBinsOld[cnt]) {
   FreqBins[cnt] = max(FreqBinsOld[cnt] - Speedfilter, FreqBinsNew[cnt]);
  }
  else if (FreqBinsNew[cnt] > FreqBinsOld[cnt]) {
   FreqBins[cnt] = FreqBinsNew[cnt];
  }
  FreqBinsOld[cnt] = FreqBins[cnt];
  // using this same loop to do some processing like scaling the freqbins to matrix height
  // if VUMeter is displayed then matrix height -1
  if (VUMeter == 1)  FreqBins[cnt] = map(FreqBins[cnt], 0, SENSE, 0, kMatrixHeight - 1);
  else FreqBins[cnt] = map(FreqBins[cnt], 0, SENSE, 0, kMatrixHeight);
 }
```

Process the read data and if the newer value < old value smootly decrease the stack by number set in Speedfilter. This will make the display jump around less and it looks less hectic.

Step 4: Process the VU meter value

```
// Value for VU meter processing
gVU = (gVU * 400) / SENSE; // this makes the VU scale depend on the Sense potmeter
if (VUadc < oldVU)gVU--;
else gVU = VUadc;
oldVU = VUadc;
if (abs(gVU) > DemoTreshold)LastDoNothingTime = millis(); // if there is signal in any off the bands[>treshhold] then no demo mode
```

This will simply update the gVU value. If the new value is > DemoTreshold, it will also reset a timer that is used to determine how long the input signal was low, so the system can go into firemode if needed.

Step 5: Process the bin values into bar height.

```
// Small amount of averaging between frames
barHeight = ((oldBarHeights[band] * 1) + barHeight) / 2;
```

Yes, it's realy that simple!

Step 6: Move Peak to new position

```
// Move peak up
  if (barHeight > peak[band]) {
    peak[band] = min(TOP, barHeight);
    PeakFlag[band] = 1;
  }
```

The barheight is set if needed.

Step 7: Check for input, if no input go to firemode. If there is signal, wake up if needed

```
// if there hasn't been much of a input signal for a longer time ( see settings ) go to demo mode
if ((millis() - LastDoNothingTime) > DemoAfterSec && DemoFlag == false)
{
  DemoFlag = true;
  // first store current mode so we can go back to it after wake up
  DemoModeMem = buttonPushCounter;
  AutoModeMem = autoChangePatterns;
  autoChangePatterns = false;
  buttonPushCounter = 12;
  Serial.println("Automode is turned of because of demo");
}
// Wait,signal is back? then wakeup!
else if (DemoFlag == true &&   (millis() - LastDoNothingTime) < DemoAfterSec   )
{
  // while in demo the democounter was reset due to signal on one of the bars.
  // So we need to exit demo mode.
  buttonPushCounter = DemoModeMem; // restore settings
  Serial.printf ("automode setting restored to: %d", AutoModeMem);
  autoChangePatterns = AutoModeMem; // restore settings
  DemoFlag = false;
  FastLED.clear();
}
```

A timer is running and every time there is an input signal, the timer is reset. Without input signal, when the timer = DemoAfterSec, as set in Settings.h, the firemode will start.
If the signal is back, firemode will exit.

Step 8: Call the actual routines to display the LED patterns. For each mode, there is a different routine

Using a switch function, depending on the mode 0 to 12, a function is called to display the frequency data, according to a pattern set in the functions. If applicable, a separate function is called to display peaks.

There are too many to display all but here is the code for mode 1:

```
TriBarLS(band, barHeight);
TriPeakLS(band);


The actual functions of each mode are stored in PatternsLedstrip.h
//************ Mode 1 ***********
void TriBarLS(int band, int barHeight) {
  int xStart = offsetBar + (BAR_WIDTH * band);
  for (int x = xStart; x < xStart + BAR_WIDTH; x++) {
    for (int y = TOP; y >= 0; y--) {
      if (y >= TOP - barHeight) {
        if (y < (kMatrixHeight / 3)) matrix -> drawPixel(x, y, CHSV(TriBar_Color_Top));   //Top red
        else if (y > (1 * kMatrixHeight / 2)) matrix -> drawPixel(x, y, CHSV(TriBar_Color_Bottom)); //green
        else matrix -> drawPixel(x, y, CHSV(TriBar_Color_Middle));     //yellow
      }
      else {

        matrix->drawPixel(x, y, CRGB(0, 0, 0).fadeToBlackBy(200)); // make unused pixel in a band black
      }
    }
  }
}

void TriPeakLS(int band) {
  int xStart = offsetBar + (BAR_WIDTH * band);
  int peakHeight = TOP - peak[band] - 1;
  for (int x = xStart; x < xStart + BAR_WIDTH; x++) {
    if (peakHeight < 4) matrix -> drawPixel(x, peakHeight, CHSV(TriBar_Color_Top_Peak)); //Top red
    else if (peakHeight > 8) matrix -> drawPixel(x, peakHeight, CHSV(TriBar_Color_Bottom_Peak)); //green
    else matrix -> drawPixel(x, peakHeight, CHSV(TriBar_Color_Middle_Peak)); //yellow
  }
}
```

Take a look at PattersLedstrip.h for more examples

6.2.3. getData()

Function: Update the webinterface HTML page with new data

```
void getData() {

  String json = "[";
  for (int i = 0; i < 10; i++) {
    if (i > 0) {
      json += ", ";
    }
    json += "{\"bin\":";
    json += "\"" + labels[i] + "\"";
    json += ", \"value\":";
    json += String(FreqBins[i]);
    json += "}";
  }
  json += "]";
  webSocket.broadcastTXT(json.c_str(), json.length());
}
```

In order to see the bargraph actually move, you'll need to feed it with data. That is what this does.

6.2.4. Matrix_Flag()

Function: To display the Dutch National Flag on the Led matrix

```
void Matrix_Flag() {

 int ledcounter = 0;
 for (int i = 0; i < kMatrixWidth / 2; i++) {
  for (int j = 0; j < kMatrixHeight / 3; j++) {
   leds[ledcounter] = CRGB::Blue;
   ledcounter++;
  }
  for (int j = 0; j < kMatrixHeight / 3; j++) {
   leds[ledcounter] = CRGB::White;
   ledcounter++;
  }
  for (int j = 0; j < kMatrixHeight / 3; j++) {
   leds[ledcounter] = CRGB::Red;
   ledcounter++;
  }
  // if number of rows can not be devided by 3 then compensate
  for (int i = 0; i < (kMatrixHeight - (kMatrixHeight / 3) * 3); i++) {
   leds[ledcounter] = CRGB::Red;
   ledcounter++;
  }
  // if number of rows can not be devided by 3 then compensate
  for (int i = 0; i < (kMatrixHeight - (kMatrixHeight / 3) * 3); i++) {
   leds[ledcounter] = CRGB::Red;
   ledcounter++;
  }

  for (int j = 0; j < kMatrixHeight / 3; j++) {
   leds[ledcounter] = CRGB::Red;
   ledcounter++;
  }
  for (int j = 0; j < kMatrixHeight / 3; j++) {
   leds[ledcounter] = CRGB::White;
   ledcounter++;
  }
  for (int j = 0; j < kMatrixHeight / 3; j++) {
   leds[ledcounter] = CRGB::Blue;
   ledcounter++;
  }

 }
 FastLED.show();
}
```

This is called upon when pressing the sel buttons for 3 seconds.It displays the Dutch Flag to indicate that the autochange functions was enabled/disabled.

6.3. Setting up your Led matrix

Depending on how you wire up your ledstrips or what matrix you are using, you might have to change the following settings. If your system is showings bars up side down or right to left where you expect left to right take a look here:

see https://github.com/marcmerlin/FastLED_NeoMatrix for Tiled Matrixes, Zig-Zag and so forth

```
FastLED_NeoMatrix *matrix = new FastLED_NeoMatrix(leds, kMatrixWidth, kMatrixHeight,
  NEO_MATRIX_BOTTOM     + NEO_MATRIX_RIGHT +
  NEO_MATRIX_COLUMNS    + NEO_MATRIX_ZIGZAG +
  NEO_TILE_TOP + NEO_TILE_LEFT + NEO_TILE_ROWS);
```

6.3.1. DrawVUMeter(), DrawVUPixels()

Function: Update the Vu meter display

```
void DrawVUMeter(int yVU) {
 static int iPeakVUy = 0;                    // size (in LED pixels) of the VU peak
 static unsigned long msPeakVU = 0;          // timestamp in ms when that peak happened so we know how old it is
 const int MAX_FADE = 256;
 matrix->fillRect(0, yVU, matrix->width(), 1, 0x0000);
 if (iPeakVUy > 1) {
  int fade = MAX_FADE * (millis() - msPeakVU) / (float) 1000;
  DrawVUPixels(iPeakVUy,   yVU, fade);
 }
 int xHalf ;
 if (DoubleVU == true)xHalf = (PANE_WIDTH / 2); // - 1;
 else xHalf = PANE_WIDTH;
 int bars  = map(gVU, 0, MAX_VU, 1, xHalf);
 bars = min(bars, xHalf);
 //  bars = min(bars, xHalf*2);
 if (bars > iPeakVUy) {
  msPeakVU = millis();
  iPeakVUy = bars;
 }
 else if (millis() - msPeakVU > 1000)iPeakVUy = 0;
 for (int i = 0; i < bars; i++)DrawVUPixels(i, yVU);
}
```

This invokes another function:

```
void DrawVUPixels(int i, int yVU, int fadeBy = 0) {
 CRGB VUC;
 if (i > (PANE_WIDTH - 3)) {
  VUC.r = 255;
  VUC.g = 0;
  VUC.b = 0 ;
 }
 else if (i > (PANE_WIDTH / 5)) {
  VUC.r = 255;
  VUC.g = 255;
  VUC.b = 0;
 }
 else { // green
  VUC.r = 0;
  VUC.g = 255;
  VUC.b = 0;
 }
 if (DoubleVU == true) {
  // dbgprint("i=%d", i);
  if (i > ((PANE_WIDTH / 2) - 2)) { // little overrule
   VUC.r = 255;
   VUC.g = 0;
   VUC.b = 0 ;
  }
  int xHalf = matrix->width() / 2;
  matrix->drawPixel(xHalf - i - 1, yVU, CRGB(VUC.r, VUC.g, VUC.b).fadeToBlackBy(fadeBy));
  matrix->drawPixel(xHalf + i,   yVU, CRGB(VUC.r, VUC.g, VUC.b).fadeToBlackBy(fadeBy));
 }
 else {
  matrix->drawPixel( i,   yVU, CRGB(VUC.r, VUC.g, VUC.b).fadeToBlackBy(fadeBy));
 }
}
```

This is what actually displays the VU meter if VUMeter is enabled in setting.h

CRGB Logo[N_PIXELS_LOGO];

// See manual if you need to change these settings

6.4. Settings

The following settings can be changed. The name in the top of each table gives the name of the file where to find it.

Settings.h

```
// Ledstrip / matrix settings
#define CHIPSET          WS2812B              // LED strip type
#define LED_PIN          27                   // LED strip data
#define COLOR_ORDER      GRB                  // If colours look wrong, play with this
#define LED_VOLTS        5                    // Usually 5 or 12
#define MAX_MILLIAMPS    2000                 // Careful with the amount of power here if running off USB port
#define offsetBar  0                          //if you need the whole display to start on a different column, you can give an offset.
const int kMatrixWidth =   10; //128;        // Matrix width --> number of columns in your led matrix
const int kMatrixHeight =  18 ;//64;         // Matrix height --> number of leds per column

// Logo
#define LED_PIN_LOGO     14                  // NeoPixel LED strand is connected to this pin
#define N_PIXELS_LOGO    12                   // Number of pixels in Logo strand
long   firstPixelHue =   0;                  // logo only
#define HueSpread        5                   // Hue spread between two pixels: 255/N_PIXELS_LOGO to spread the total hue over
                                              // your leds other options: you can also put down any number 0-255

//Options Change to your likings
// first the options related to visualisation
#define BottomRowAlwaysOn  1          // if set to 1, bottom row is always on. Setting only applies to LEDstrip not HUB75
#define Fallingspeed      30          // Falling down factor that effects the speed of falling tiles
int    PEAKDELAY =       60;          // Delay before peak falls down to stack. Overruled by PEAKDEALY Potmeter
#define SecToChangePattern  10        // number of seconds that pattern changes when auto change mode is enabled
boolean VUMeter =         1;          // if 1 then VU is visible, if 0 then it's off. can be overrules by tripple press on mode button
boolean DoubleVU =        true;       // use a double vu meter from center to L and R, of when false, use single from L to R
#define Speedfilter       20          // slowdown factor for columns to look less 'nervous' The higher the quicker
#define BRIGHTNESSMAX     100         // Max brightness of the leds...carefull...to bright might draw to much amps!
int    BRIGHTNESSMIN =    1;          // Min brightness
int    BRIGHTNESS =       50;         // Default brightness, however, overruled by the Brightness potmeter

//amplification and signal related stuff
#define MAX_VU            1500        // How high our VU could max out at.  Arbitarily tuned.
#define InputBoost        1000        // this is the SEnse potmeter amplification. Higher number for low level input
int    noisemic =        300;
int    noiseline =       300;
#define ADC_OFFSET        1680
#define VU_OFFSET         2200
// other stuff
int    buttonPushCounter = 0;          // This number defines what pattern to start after boot (0 to 12)
bool   autoChangePatterns =false;      // After boot, the pattern will not change automatically.
#define DemoAfterSec      6000         // if there is no input signal during this number of milli seconds, the unit will go to demo mode
#define DemoTreshold      200          // this defines the treshold that will get the unit out of demo mode
#define NumberOfModes     13           // this is the number of modes. If you add modes, increase this number also.
#define LONG_PRESS_MS     3000         // Number of ms to count as a long press on the switch

// Debug features should default be off!
int    DEBUG =           0 ;           // When debug=1, extra information is printed to serial port. Turn of if not needed--> DEBUG=0
#define DEBUG_BUFFER_SIZE  100         // Debug buffer size

// Label for Frequency columns in webbrowser
String labels[] = {"30", "60", "125", "250", "500", "1K", "2K", "4K", "8K", "16K"};


/*****************************************************************************
  Colors of bars and peaks in different modes, changeable to your likings
 *****************************************************************************/
```

Please see the actual setting.h file to change these values

6.5. Files

The firmware can be divided into 6 files

AnalogAnalyser1.0.ino → main function and code
LEDDRIVER.H -> All stuff to setup your led matrix, zigzag, left to right, right to left etc.
PatternsLedstrip.h → Here you'll find all the patterns
Settings.h → here are all the settings including options and colors
Webstuf.h→ this is the array webpage[] containing your actual webpage.
Fire.h → this is the code for the firemode screensaver

You can download the latest sketch here:

https://github.com/donnersm/Analog-Analyzer/tree/main/Firmware

7. Programming your Arduino

I used the Arduino IDE. It is freely available online and it does the job. However, I recently stumbled on something called Sloeber Beryllium which is a great tool that offers a much better compiler interface. However, it has a bit of a learning curve but I promise, it's worth it! Why don't you check it out? You can also use Visual Studio or some other great IDE. However, it is important the right library and it is best not to install what you don't need as it might give you errors when compiling. Make sure that your Arduino IDE is set for using the ESP32. If you don't know how to do so, google or look it a youtube video. There are some very clear instructions and setting up the IDE is not hard. You can do it! In a nutshell, it comes down to this:

1. In the Ide preferences window, look for the line: Additional Boards Manager and add the following line; https://dl.espressif.com/dl/package_esp32_index.json
2. Go to your board manager and look for ESP32 and install the ESP32 from Espressif Systems.
3. Select the correct board before you compile and you are good to go

7.1. Here a few libraries that you'll need for sure:

Here are the libraries that you will need to install using the Arduino Library manager. These I the versions I used. Higher version might work just fine.

FastLED_NeoMatrix	at version 1.1
Framebuffer_GFX	at version 1.0
FastLED	at version 3.4.0
Adafruit_GFX_Library	at version 1.10.4
EasyButton	at version 2.0.1
WiFi	at version 1.0
WebServer	at version 1.0
WebSockets	at version 2.1.4
WiFiClientSecure	at version 1.0
Ticker	at version 1.1
WiFiManager	at version 2.0.5-beta
Update	at version 1.0
DNSServer	at version 1.1.0
Adafruit_BusIO	at version 1.7.1
Wire	at version 1.0.1
SPI	at version 1.0
FS	at version 1.0

Remark: I had some trouble compiling when I started. Turned out that Arduino IDE had many libraries activated and it decided to choose the wrong ones whenever it had to choose between libraries. I solved it by uninstalling the Arduino IDE and re-installing it from scratch.

Also, since some libraries are included with others, maybe this helps. Try sticking to these first:

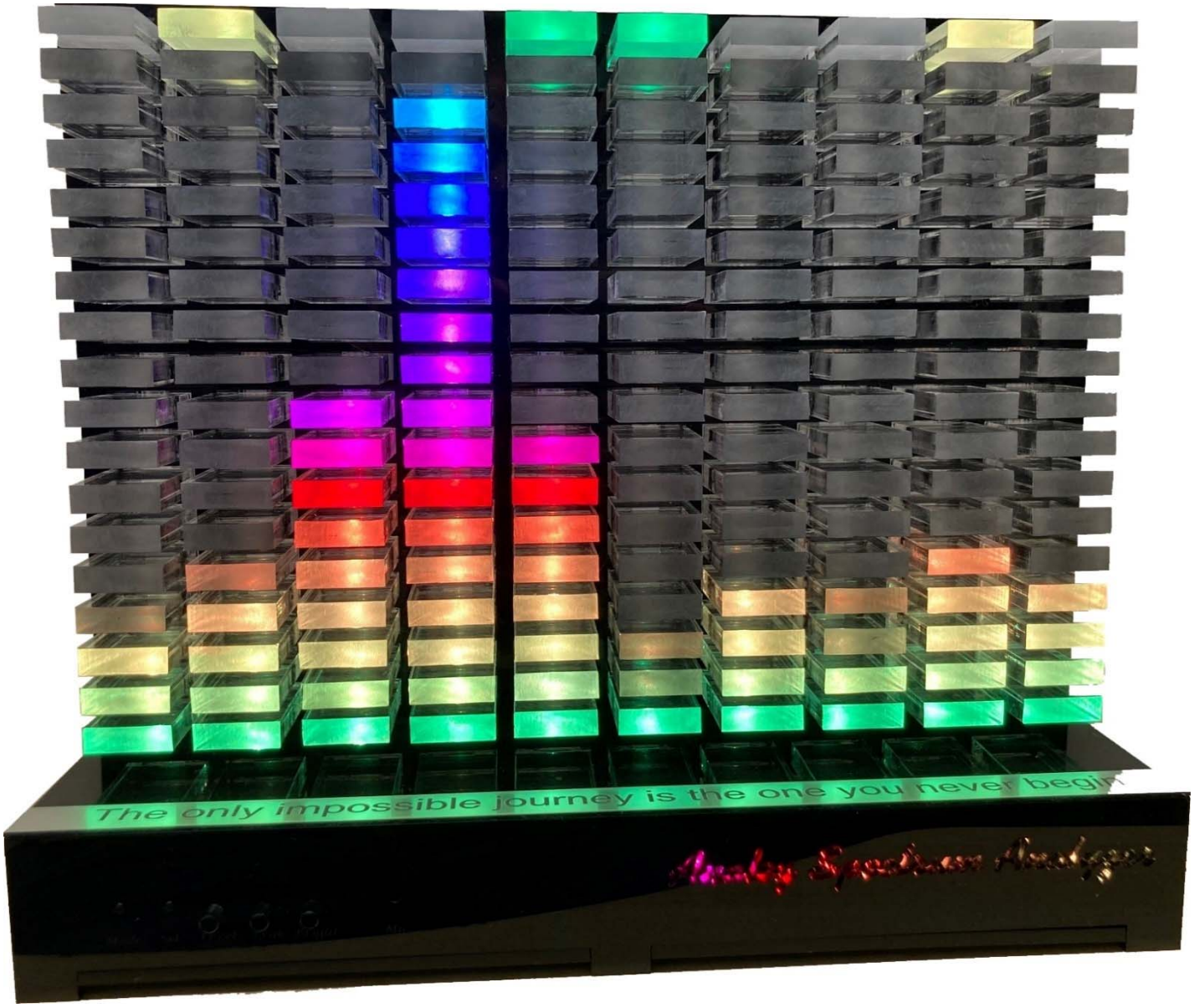
```
#include <FastLED_NeoMatrix.h>
#include <Adafruit_GFX.h>
#include <pixeltypes.h>
#include <EasyButton.h>
#include <WiFi.h>
#include <WebServer.h>
#include <WebSocketsServer.h>
#include <Ticker.h>
#include <WiFiManager.h>
```

8. Assembling your analyzer

Assuming that you want to build the same awesome setup as I did, the Autocad files on my github may come in handy

ww.gthub.nl

Using these files, you should be able to create something awesome like this:



It's just a matter of laser cutting all the parts. Everything except two things are made of 5mm black acrylic.

The tiles that light up are made of 10mm transparent acrylic and the layer in between the backpanel and frontpanel; the layer that the ledstrips are played in; is made of 3 mm transparent acrylic. You can also use black acrylic for that if you like but with transparent acrylic, the sides of the back will light up as well and I liked it that way.

All parts are glued together with Acryfix glue (1R0192) but other glue for acrylic might work just fine. The tricky part is not to use to much glue.

When placing the tiles, work 1 row at a time. I used an Aluminium L shape profile to line them all up and I left it in place until the glue was hardened enough. Only then, I moved on the next row.

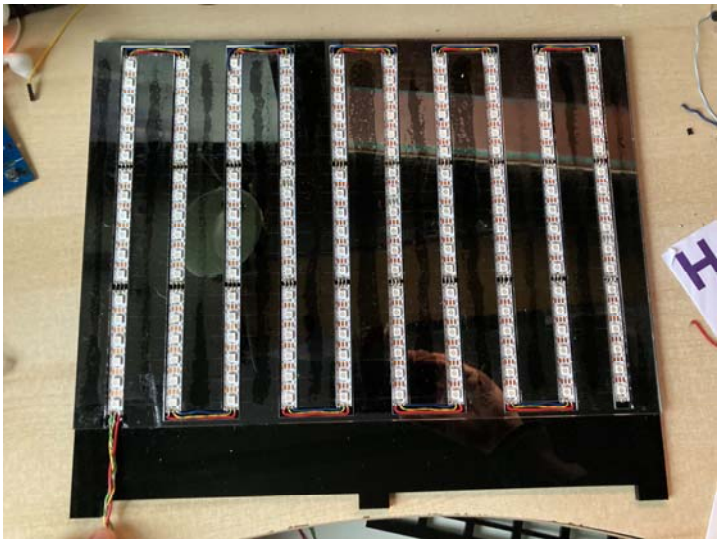
In the back, I drilled some holes for power and audio input and a switch. I didn't include those holes in the laser cut design because I didn't now where I wanted those at the time. That's why I drilled them after cutting but before assembly!

Here are some photo's to help you on your way.

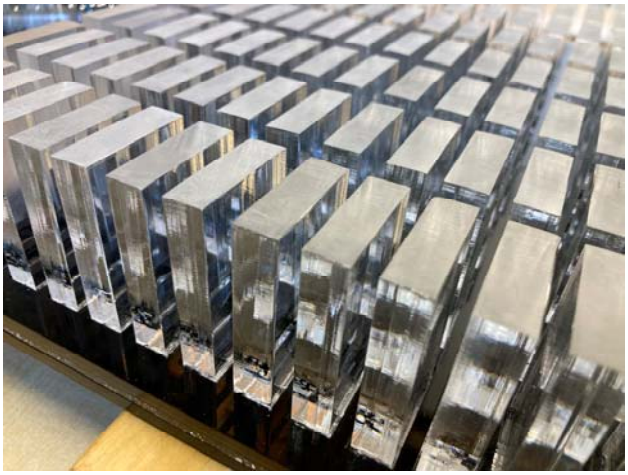
All the parts (Tiles not included) before gluing everything together

Inner plate (3mm) with spacing for ledstrips

All parts glued together (Tiles not included)



Led strips in place. Take note of how I divided each strip into 3 sections. These 'made in China' strips are very affordable unfortunately; cheap price also means cheaper quality. I ordered 3 sets of Ledstrip (74 leds /mtr) and all three where different. Somehow, they all have a tolerance in spacing. So before you attach the ledstrip to the backplate, make sure the leds line up with the tiles. In my AutoCAD file, I included a raster that you can engrave to help you. You do have some margin as the tiles are 10mm in height. That's why I line them up in sections of 8.



Glued Tiles

Glue one row at a time using a L shape profile ruler

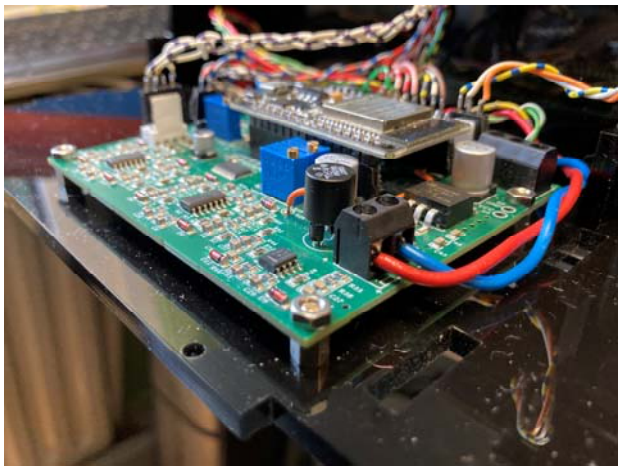
The operating panel in place. Take note how the tactile switches are long enough to stick out of the 5mm acrylic, just enough to operate it.

The operating panel is hold in place by a little acrylic ‘left over’ glue to the housing. Some foam tape in between will do the job. The microphone is carefully glued to the front plate.

The logo from the outside

The Ledstrip for the logo is attached to a ‘left over’ acrylic plate that has been glued to the housing.

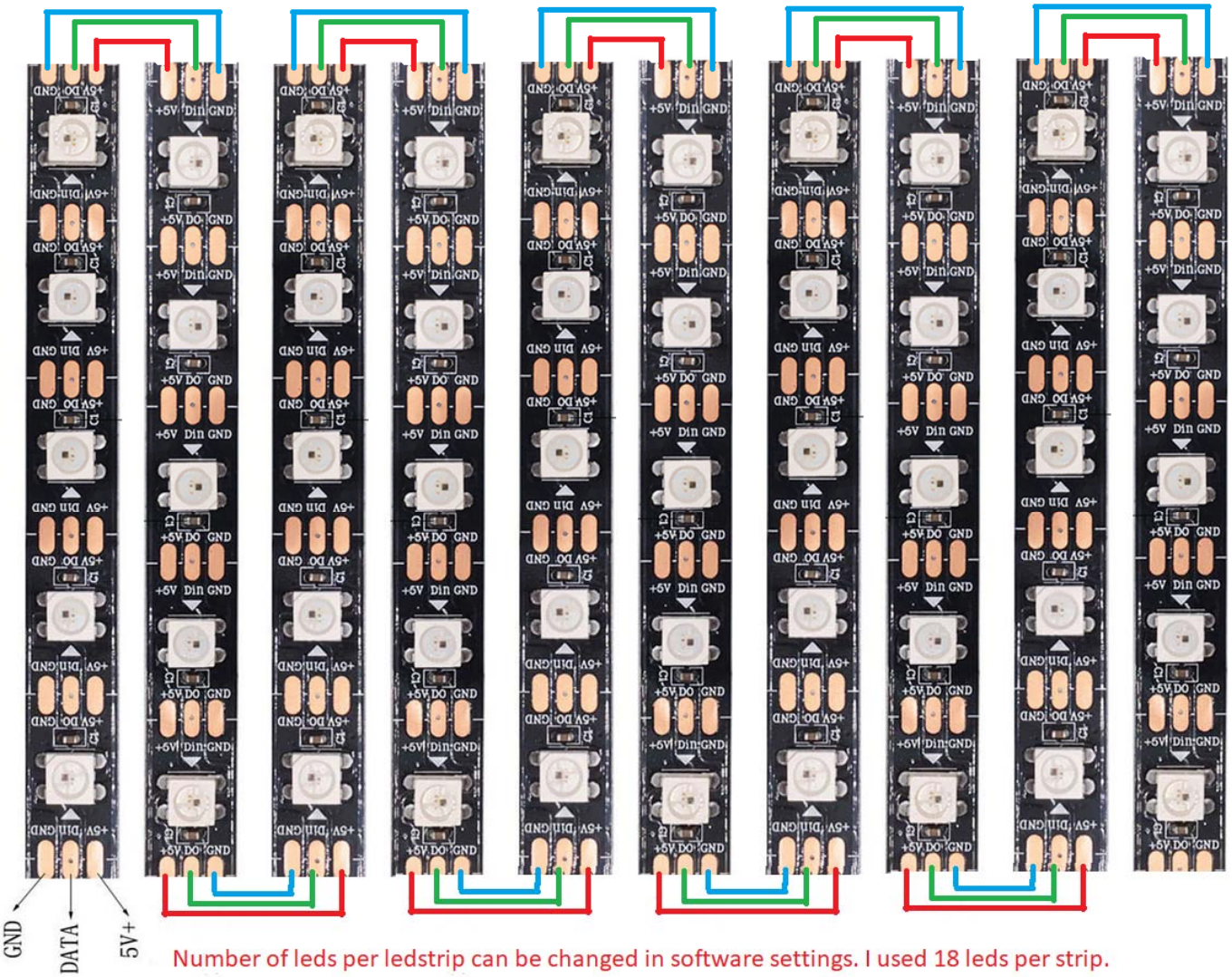
The placement of the PCB on the bottom plate. Also visible are the power switch and connectors for power and audio. The wires still need to be attached with some Ty-raps.



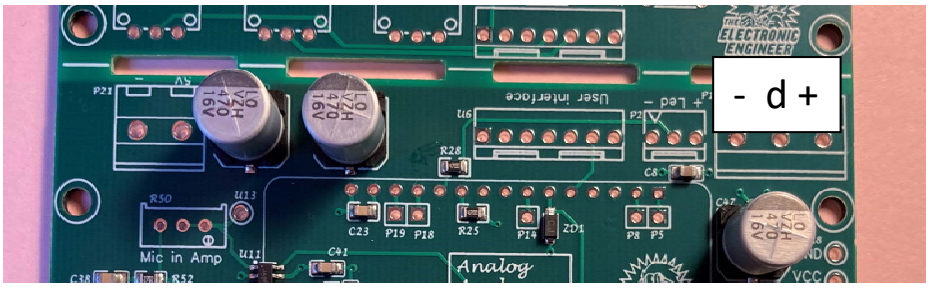
The PCB from another angle. Don’t mind the orange wire on the PCB. It was only there for experimenting.

8.1. Wiring up the Ledstrips

Take note, my way of wiring is not the only way but if you decide to deviate from is, you'll need to adjust the LEDSTRIP setup. See software section for details.



The connection to the PCB is made on the left. Connect this to connector P1 of the PCB The denter pin is de data pin



A single ledstrip with number of leds that can be set in setting.h, can be connected to P2. It will show an ever changing rainbow. Use it for illuminating the logo.

9. Trouble shooting and a word of thanks

You probably got hands on this document as a result of watching a video on my youtube channel, right?

Well, if you did, I would appreciate a like, a subscribe and a comment if you like. Feedback and pat on the shoulder is what keeps me going and it motivates me to put more stuff in the public domain.

https://www.youtube.com/channel/UCm5wy-2RoXGjG2F9wpDFF3w

Thank you for taking the time to read about, and hopefully build, this project