Thomas More

Campus De Nayer

Jan De Nayerlaan 5

2860 Sint-Katelijne-Waver

Tel. (015) 31 69 44

*Practice Enterprise Electronics*

*Schooljaar 2020-2021*

**NAAM: Hannes Hugaert & Stijn Van den bossche**

#### **ONDERWERP: Muziekversterker**

**AFDELING: Elektronica – Embedded Hardware 2 | 2EE2**

**Elektronica – Embedded Software 2 |**

[1. Woord vooraf 3](#_Toc72528480)

[2. Hardware 3](#_Toc72528481)

[**2.1** **Blokschema** 3](#_Toc72528482)

[**2.2** **Schema** 6](#_Toc72528483)

[2.2.1 Communicaties 8](#_Toc72528487)

[2.2.2 Spanningsregelaars 10](#_Toc72528492)

[2.2.3 Digitale potentiometer(AD5204BRUZ10-REEL7) 10](#_Toc72528493)

[2.2.4 8 digit LED display driver (MAX7219CNG+) 11](#_Toc72528494)

[2.2.5 Versterking 14](#_Toc72528502)

[**2.3** **PSpice** 15](#_Toc72528503)

[2.3.1 High-Pass filter with gain 15](#_Toc72528505)

[2.3.2 Phano Preampfilter 16](#_Toc72528506)

[2.3.3 Active tone control 17](#_Toc72528507)

[2.3.4 Links en rechts balancer + volume aan de uitgang regelen 19](#_Toc72528508)

[**2.4** **PCB lay-out** 20](#_Toc72528509)

[3 Software 28](#_Toc72528510)

[**3.1 Flowchart** 28](#_Toc72528511)

[**3.3** **Listing** 29](#_Toc72528512)

[4 Bediening handleiding 37](#_Toc72528513)

[5 Besluit 39](#_Toc72528514)

[6 Bronnen 40](#_Toc72528515)

# **Woord vooraf**

Wij gaan een muziekversterker maken die voorzien is van 2 input-channels en een microfoon. Op deze versterker kunnen we de verschillende tonen regelen per kanaal (treble, medium en bass), alsook het volume van deze kanalen. Hier bovenop voorzien we ook een ‘balans’, om te regelen in welke mate beide kanalen worden doorgegeven, en ook een ‘master output’. Hiermee kiezen we het volume van het uiteindelijke uitgangssignaal. Onze sliders zijn niet de gewone, mechanische sliders die meestal aanwezig zijn op een mengpaneel, wij voorzien touchpads als sliders. Om ook het volume of de verschillende tonen te visualiseren, voorzien we ledjes naast de touchpads.

# **Hardware**

## **Blokschema**

**Audio output:** staat voor de luidsprekers die aan de uitgang komen, dezen moeten voorzien worden van een inwendige versterking omdat onze versterker amper stroom zal hebben aan de uitgang.

**2 AUDIO inputs:** Wij hebben ons mengpaneel voorzien van 2 inputkanalen waar wij muziek op kunnen laten spelen (via AUX).

Ook kunnen we op lijn 1 eventueel een microfoon steken, zodat over de muziek heen te praten is, indien gewenst.

**5V DC:** micro USB, dit is de voeding voor de versterker.

**5V DC naar 3.3 V DC:** dit is de naam van de component 173950336, het is een step-down die 5V omzet naar +3.3 V DC. Dit wordt gebruikt om de microcontroller en de capacitieve touchsensor te voeden.

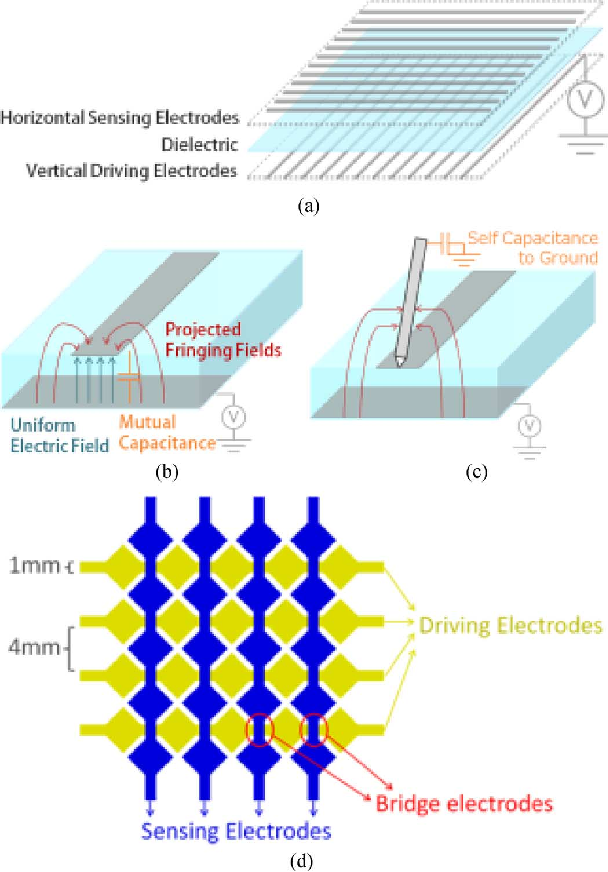
**Microcontroller:** De gebruikte microcontroller is eenSTM32F411CEU7, om hiermee te communiceren wordt een st-link V2 gebruikt, zodat we deze kunnen programmeren. Onze Microcontroller is van de SMT32F4x familie en die wordt gevoed met 3.3V en we willen ook ongeveer 40 I/O. En onze heeft er 49 dus dat is zeer goed.

**8-digit LED displaydrivers:** deze chip heet MAX7219CNG+. Dit is een chip om 8 segment displays aan te sturen, wij gebruiken een ‘matrix’ van leds, waarvoor deze drivers ook te gebruiken zijn.

**LED’s:** Dezen laten het huidig geselecteerde volume en de stand van de verschillende tonen zien. Er zijn 132 leds, om dit genoeg in detail weer te geven.

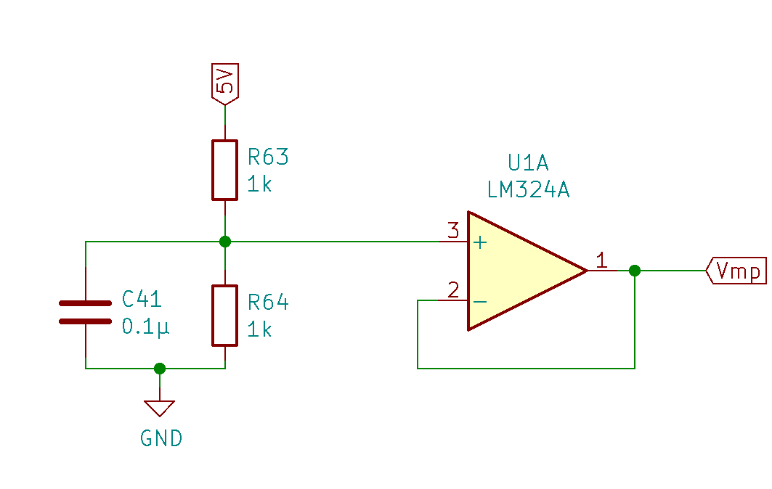
**Capacitieve tastsensor:** deze sensor heet de MTCH6301-I\_ML, het leest capacitieve aanrakingen afkomstig van de touchpads, zet deze om naar coordinaten, en verstuurt deze via een I²C signaal. Deze waardes worden wel opnieuw geïnterpreteerd, aangezien de gebruikte touchpads niet allemaal samen staan, en het dus niet één compleet vlak is. We hebben ook gekozen voor capacitief omdat dat geen corrosie geeft op de sliders.

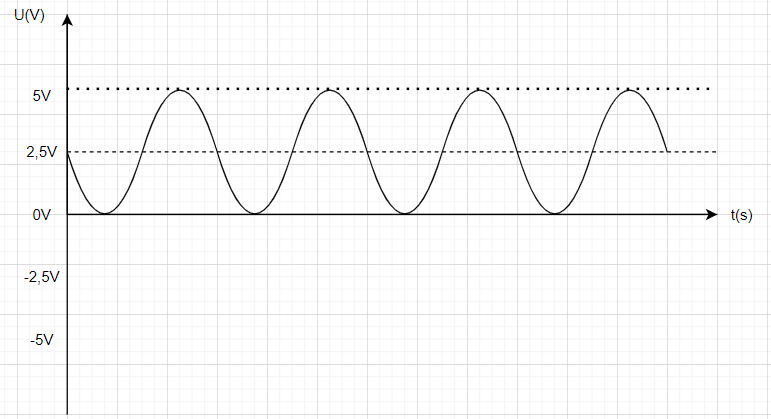
**Touchpads:** dit zijn geleiders die op een print door middel van aanraking een capacitief signaal doorsturen. Deze touchpads dienen als sliders van het mengpaneel. Ze zijn zelf ontworpen, om hier een grote kost uit te sparen, en zelf de vorm en grootte te kunnen bepalen.



**Digitale potentiometers:** onze potentiometers hebben de naam AD5204BRUZ10-REEL7. Tijdens het kiezen hoe we de potentiometers zouden aansturen, bleek dat het makkelijker zou zijn om digitaal gestuurde potentiometers te werken, anders hadden we zelf mechanische sliders moeten voorzien, wat veel duurder ging uitkomen.

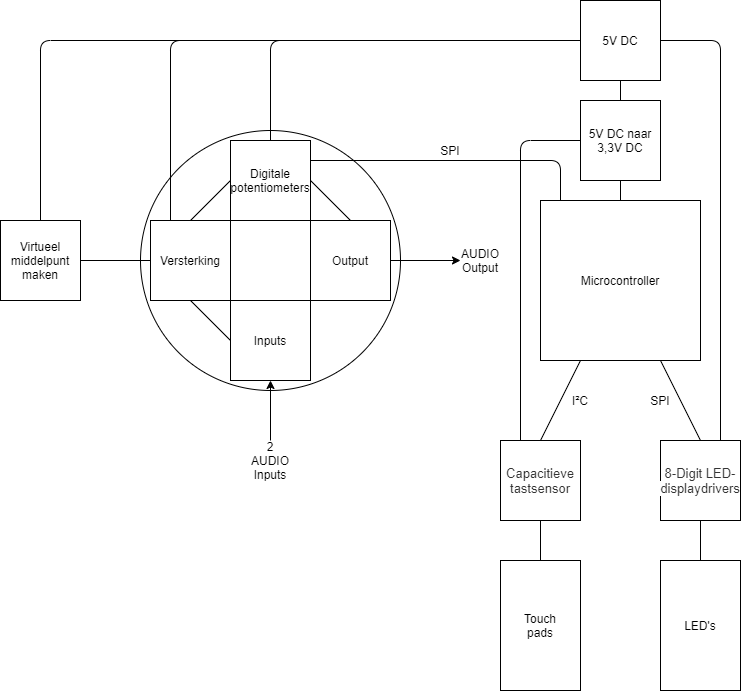
**Versterken:** We gebruiken de opamps LM324A. Ze vormen de basis van de versterkingcircuits.

**Virtueel middelpunt:** Door problemen met de maximale spanning voor de potentiometers hebben we een virtuele grond moeten voorzien. De opamps hadden we namelijk initieel gevoed met -12V en +12V, maar de potentiometers zijn maar ontworpen tot 5V.

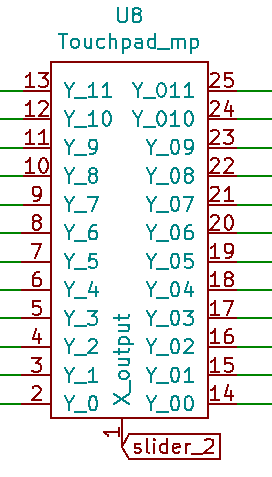


Om dit op te lossen hebben we een virtuele grond op 2.5V voorzien, en de opamps gevoed met 0V en 5V. Dit is verwezenlijkt met een eenvoudige spanningsdeler, op aanraden van de docent hebben we 1kΩ weerstanden hiervoor gebruikt.

Hierdoor gaat de uitgang van de opamps fluctueren tussen 0V en 5V. Het uitgangssignaal zou hierdoor niet beïnvloed moeten zijn, zij het dat dit mogelijks te stil is. Dit is echter makkelijk op te lossen met een actieve luidspreker.

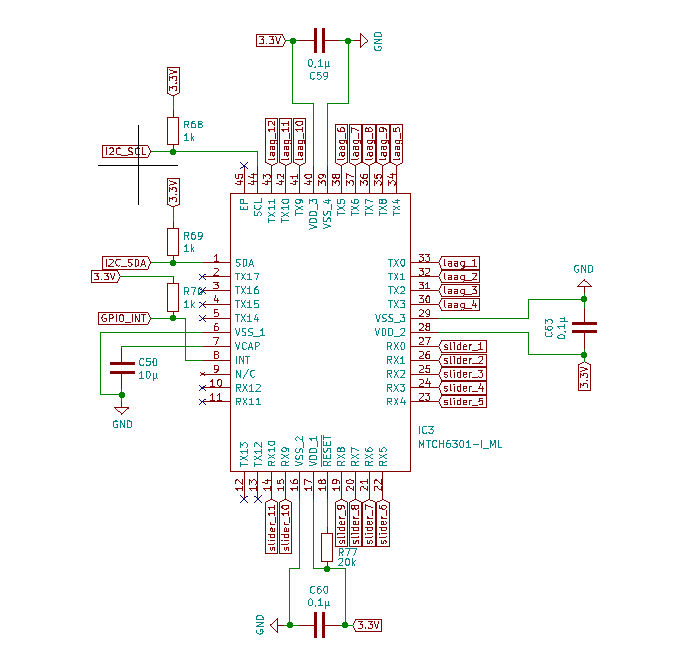


## **Schema**



**Touchpad:**

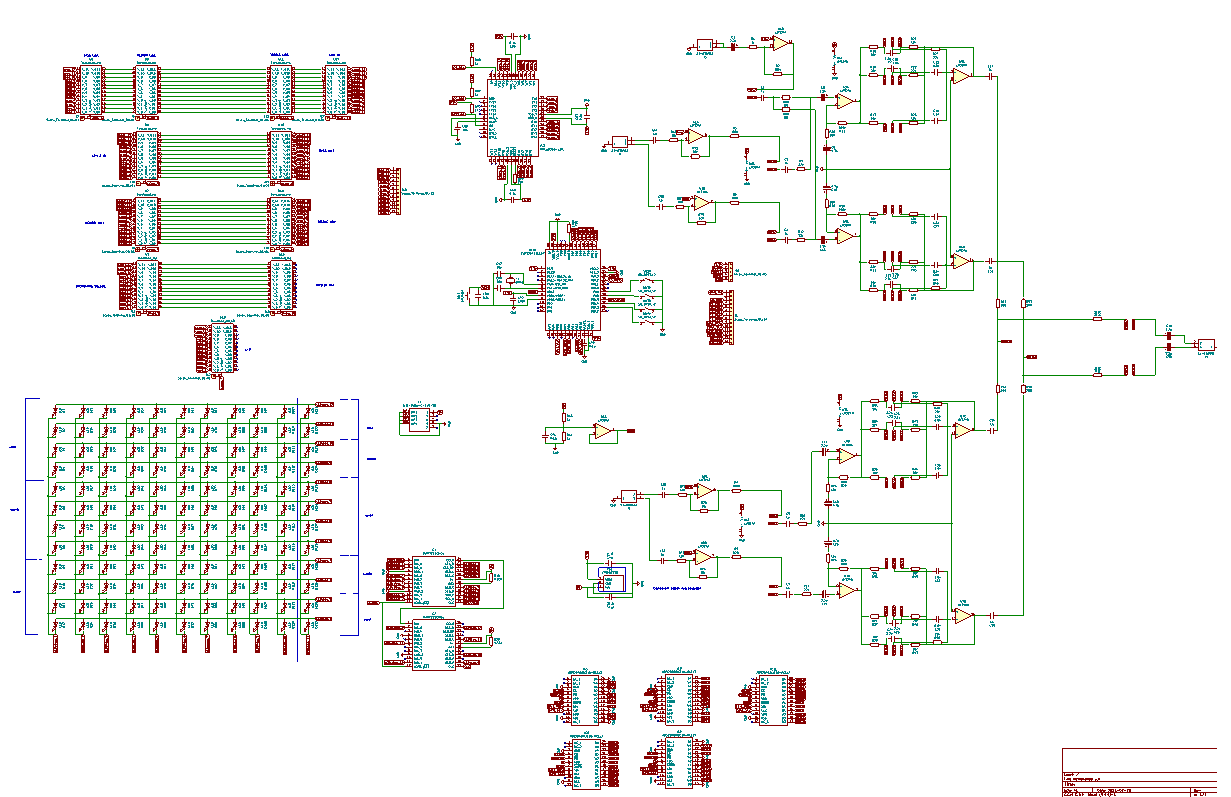
Links is het symbool van onze sliders afgebeeld. Dezen zijn zelf ontworpen, aangezien we deze rechtsreeks hebben verwerkt in onze PCB. Vooral om de footprint hiervan te maken moesten we rekening houden met veel verschillende parameters, zijnde hoe ver de verschillende vlakjes zich van elkaar konden en moesten bevinden, de dikte van verbindingslijnen hiertussen, grootte van deze vlakjes en dergelijke. Dit om te verzekeren dat een aanraking van deze touchpads correct gedetecteerd kan worden.



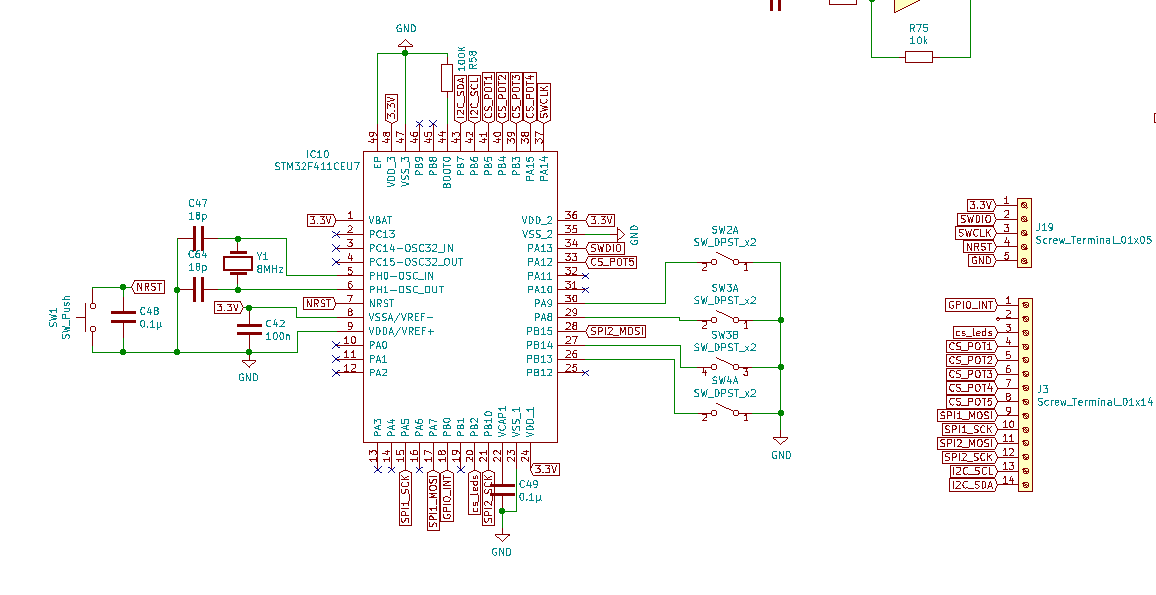
**MTCH6301-I\_ML:** dit is de IC die de signalen op de touchpad’s uitleest en omzet in data die over I²C naar de microcontroller wordt verstuurd.

Deze condensatoren zorgen ervoor dat er geen storing op de 3.3V zit, gebruikt als decoupling condensatoren.

Pull-up weerstand gebruikt om de INT lijn hoog te trekken.



Micro usb



**STM32F411CEU7:** Deze microcontroller is voorzien van 512KB Flash en 128 KB RAM en is een ARM cortex die op 32 bit werkt. Deze microcontroller is voorzien van 49 pinnen wat ruim genoeg is voor onze uitwerking. Deze microcontroller moet gevoed worden op een spanning tussen 1.7 en 3.6 dus hebben we gekozen voor het midden ongeveer 3.3V.

We hebben ons bord voorzien van externe pinnen, dit als noodoplossing in het geval er iets met de microcontroller zou gebeuren, of om makkelijk signalen te kunnen meten.

Wij hebben een reset knop voorzien op de microcontroller zodat we altijd opnieuw het programma kunnen laten afspelen. De reset moet voor zien worden van condensator zodat we geen bounce krijgen, zie oscillatie.

De microcontroller moet een kristal oscillatie hebben tussen 4 en 26 MHz. Wij hebben de onze voorzien met een kristal van 8 MHz.

Op de microcontroller zijn er twee poorten voorzien voor een oscillatie, wij hebben gekozen voor PH0 en PH1 omdat de andere een oscillatie van 32 kHz maximum mag hebben, met het gebruik van een RTC, deze bevind zich op pinnen PC14 en PC15.

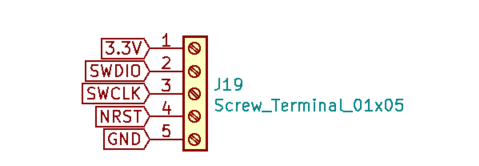
Dit is links naast de microcontroller te zien bovenaan deze pagina.



## Communicaties

Het protocol dat we gebruiken om onze microcontroller te programmeren is seriële communicatie. Dit via de ST-LINK V2. De benodigde pinnen hiervoor zijn te zijn hieronder op de voorziene connector.

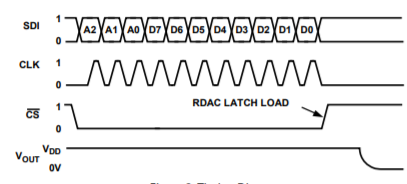


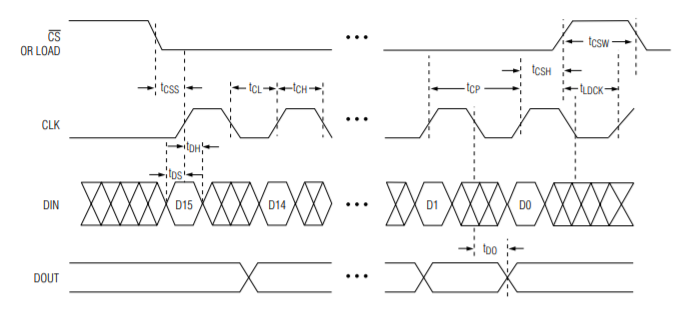
.

De pin SWCLK van de ST link V2 moet op pin PA14 van de STM32F411CEU7 komen. De pin SWDIO van de ST link V2 moeten op pin PA13 van de STM32F411CEU7 komen.

**SPI**

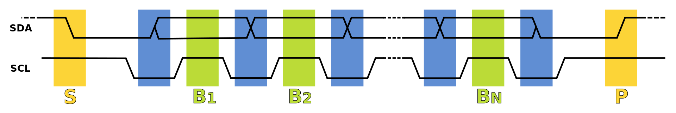
Bij SPI hangen er verschillende slaves aan dezelfde data- en kloklijn, daarom is er per slave ene aparte Chip Select (CS) lijn. Wanneer deze voor een bepaalde slave laag wordt getrokken, is de data op de datalijn (MOSI) bestemt voor deze slave, en gaat hij deze ontvangen en verwerken wanneer de CS-lijn terug hoog gaat.

* **Digitale potentiometer:** Als je CS laag gemaakt hebt kan je de 16 bit data beginnen inlezen. De digitale potentiometers beginnen de adressen en de data in te lezen als de CLK hoog is. Als alle adressen en data zijn ingelezen moet CS terug hoog worden en zal de weerstandswaarde van de potentiometer veranderen.
* **8-Digit LED Display Driver:** Hier gebeurt exact hetzelfde als bij de digitale potentiometer, de data wordt ook ingelezen op een rising edge van de klok.



**I²C**

De IC voor onze touchpads word aangestuurd met I²C. Dit protocol wordt over 2 lijnen gestuurd, de lijn SDA (data line) en SCL (clock line).

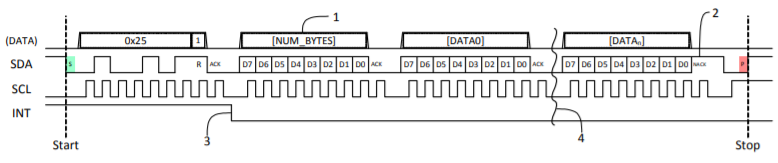


**Start:** De chip begint te starten met lezen van data als de SDA laag wordt en SCL hoog is (startconditie), daarna moet SCL eerst laag worden en dan kan de data beginnen ingelezen worden. Telkens wanneer SCL hoog wordt, wordt de data ingelezen.

De stopconditie bestaat er uit om SDA opnieuw hoog te trekken als SCL hoog is.

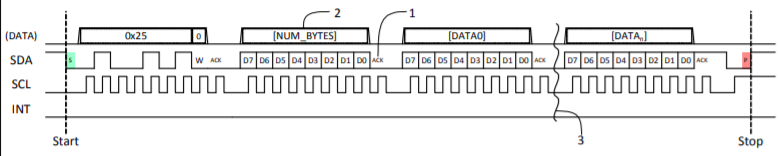
**Read:** De eerste byte data dat binnen gelezen wordt is het adres, hierna wordt de INT lijn opnieuw laag getrokken door de IC, indien er een correcte match is van adres (deze interrupt lijn is telkens hoog als er data klaar is om te verzenden).

Master read



**Write:** De eerste data dat binnen gelezen wordt is ook hier weer het adres. Hier blijft INT laag, omdat de master gaat schrijven naar de slave, en dit dus niet gebeurt op vraag van de slave.

Master write

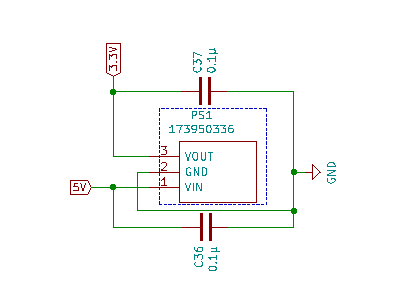




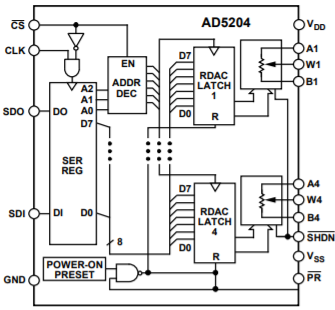
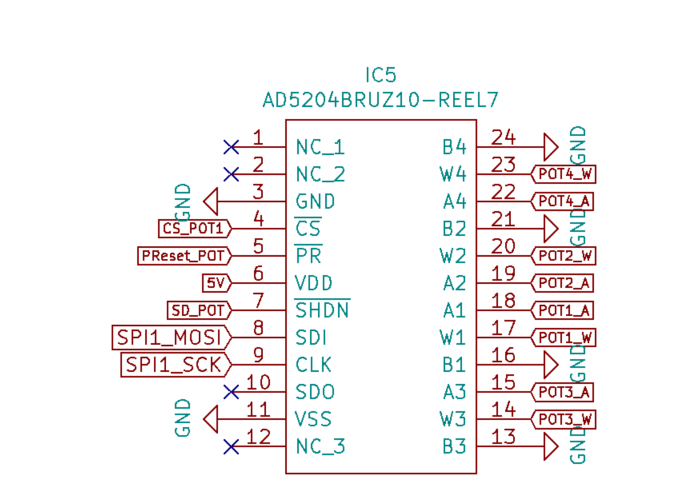
## Spanningsregelaars

**173950336:** is een Step Down die 5V DC omzet naar 3,3V DC en heeft een uitgangsstroom  
van 500 mA.

De condensatoren dienen voor spanningen af te vlakken, zo zitten we zonder ruis op de beide spanningen.



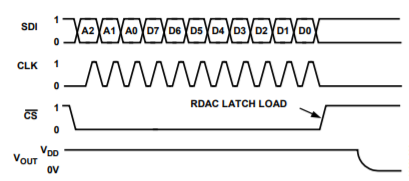
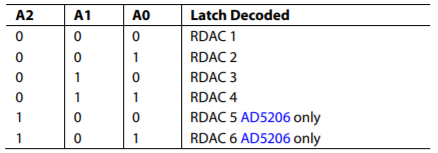
## Digitale potentiometer(AD5204BRUZ10-REEL7)



**CS:** Zorgt ervoor dat dat de waardes maar ingelezen worden als deze laag is, als alle data is ingeklokt in het shiftregister.

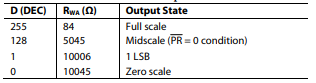
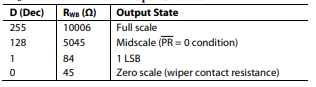
**CLK:** Als de Clock hoog is zal de data doorgeshift worden in het shiftregister.

**SDI:** Hier worden alle bits ingelezen naar het shiftregister. De eerste 3 bits zijn telkens het register adres, de rest zijn de 8 data bits (of 1 byte). Als alle data is uitgelezen en CS wordt terug hoog zal de data doorgeshift worden naar het correcte register.



**SDO:** Dient om meerdere componenten achter elkaar te schakelen in daisy-chain, dit gebruiken we echter niet.

**An, Wn, Bn:** zijn de pinnen van de potentiometer.



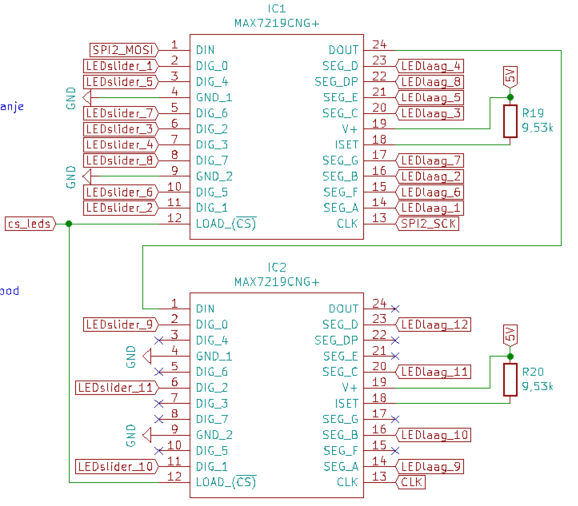
**PR:** Zorgt ervoor dat de waardes resetten als het laag is en terug op de beginwaarden komen te staan.

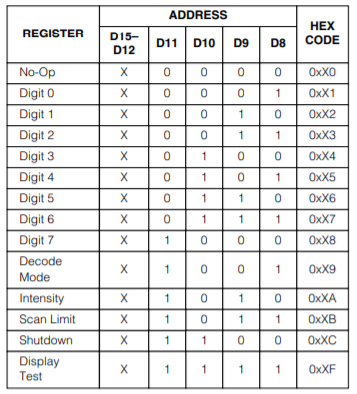
**SHDN:** is de shutdown van de potentiometers zelf. Het zet de potentiometers af als het signaal laag is.

**Vdd en Vss:** De Vdd is de spanning die de potentiometer gaat voeden en is ook de spanning die er maximum door de potentiometers mag vloeien. Vss is de minimum spanning die er door de potentiometers mag vloeien. Maar waarde Vdd moet tussen de 3V a 5V DC liggen, Vss moet tussen de 0V a -2,7V DC liggen. Het verschil tussen Vdd en Vss mag niet hoger zijn dan 5,5V.

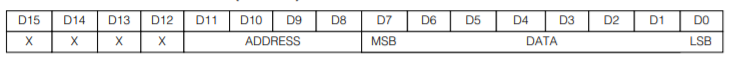
## 8 digit LED display driver (MAX7219CNG+)

Deze IC wordt vaak gebruik voor 7 segment displays, maar je kan het ook voor andere displays of matrixen gebruiken. Zo hebben wij zelf onze LED-matrix gemaakt om zo de geluidsniveaus te visualiseren. Je moet de anodes en de kathodes samen verbinden om een matrix te maken.

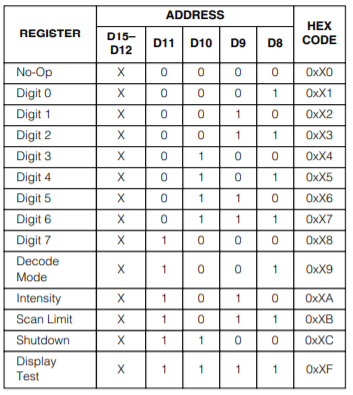
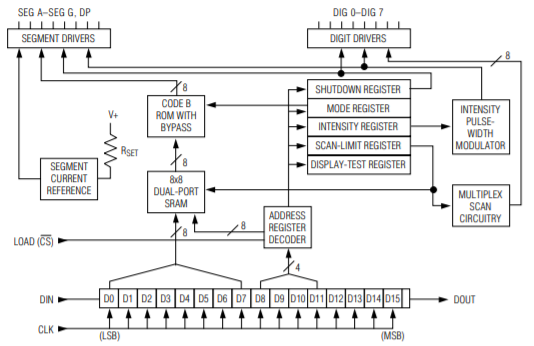




De communicatie naar deze chip werkt het zelfde als de potentiometers, ook via SPI. Er moeten 4 bits voorzien worden om adressen door te geven, dit is bit 11, 10 , 9 en 8. Daarna heb je ook de data, dit zijn8 bit, bit 7(MSB) tot bit 0(LSB).



Elk register heeft een specifieke functie, dit is te zien in de tabel hieronder. Er zijn enkel 4 bits nodig voor deze adressen, de eerste 4 bits van de adresbyte worden dus niet gebruikt, en mogen eender welke waarde hebben.



Er zijn 8 registers om verschillende ‘digits’ aan te sturen, in ons geval zijn dit kolommen met LEDs, per slider een kolom. Andere registers worden gebruikt om verschillende andere instellingen aan te passen, hier is meer uitleg over gegeven in het codesegment.

Onze LED matrix bestaat uit 11 kolommen en 12 rijen, in totaal 132 LEDs. Elke chip kan 8 rijen en 8 kolommen aansturen, voor een totaal van 64 LEDs. Ons originele plan was een tweede driver in daisy-chain te schakelen, en deze de overige rijen en kolommen aan te laten sturen. Hier hebben we echter een fout gemaakt, deze twee componenten gebruiken niet dezelfde klok om over hun matrix te scannen, dus gaan de LEDs die liggen op een rij en kolom van een verschillende driver nooit op hetzelfde moment correct aangestuurd worden. Om dit op te lossen hadden we het tweede deel LEDs allemaal op de tweede driver moeten aansluiten, voor een totaal van 128 LEDs.

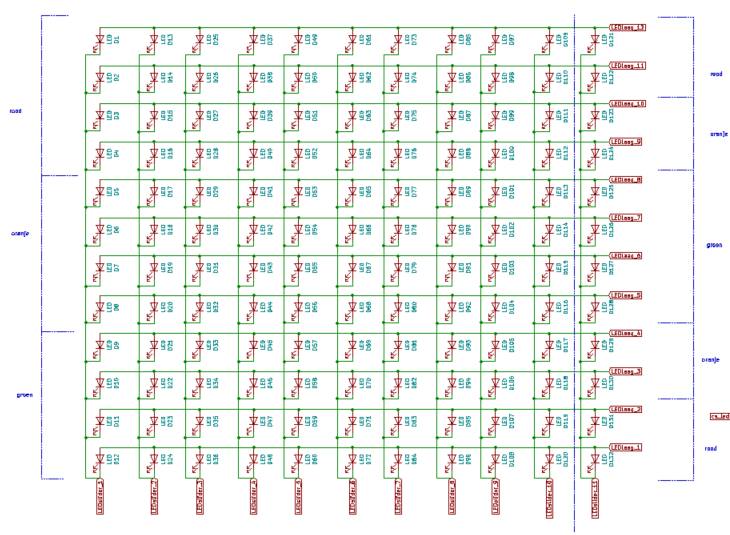
De eerste 8 rijen in combinatie met de eerste 8 kolommen zijn wel allemaal verbonden met de eerste LED driver, dus dezen zijn wel nog correct aan te sturen.

Omdat wij ook verschillende kleuren gebruiken om aan te duiden op welke volume we zitten hebben we gekozen om de kleuren groen, oranje en rood te gebruiken.

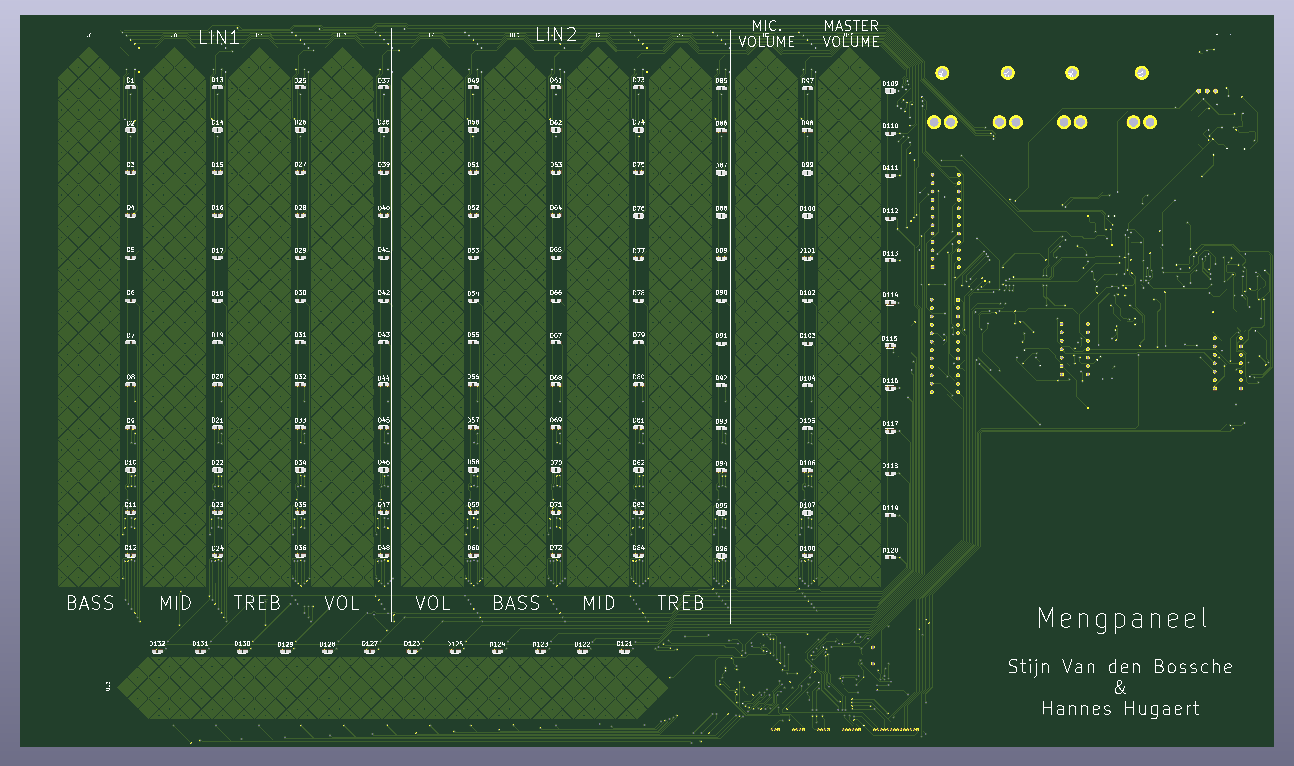
Groen betekent dat het geluid of regel circuit nog rustig staat.

Oranje betekent dat het volume of regel circuit al luider en zwaarder wordt.

Rood betekent dat het volume of regel circuit op het luidste en zwaarste staat.



Elke LEDslider heeft zijn eigen slider waar de LEDs rechts naast komen staan. Dit is ook te zien op de volgende pagina op de PCB.





## Versterking

Onze PCB is ook nog voorzien van een volledig circuit om de volumes, bassen, mediums en trebles te regelen op 2 lijnen waar muziek op komt en ook op lijn 1 waar je ook kan opteren om een microfoon op aan te sluiten. In het volgende hoofdstuk is aan de hand van grafieken te zien hoe elke versterking werkt.

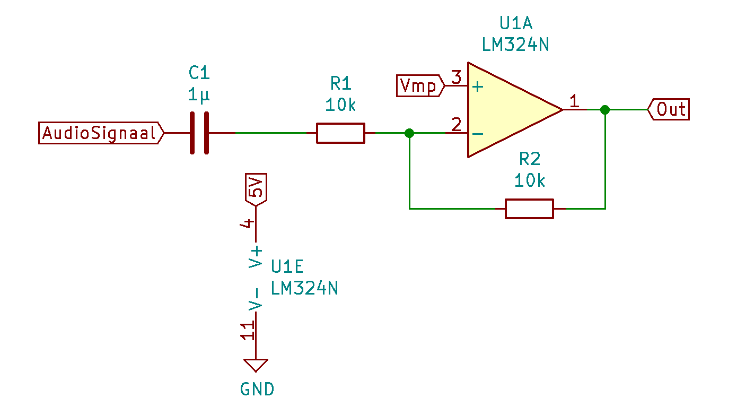
## **PSpice**



### High-Pass filter with gain

Dit is een inverterende versterker met een condensator zodat alleen AC signaal binnen gelezen wordt. Het signaal wordt geïnverteerd. Naarmate de frequentie stijgt, stijgt de versterking ook aan de uitgang.

Vout = - . Vaudiosignaal + Vmp . ( )



Vmax ongeveer = 0,5V

* Vout = - . 0.5 + 2.5 . ( )
* Vout = 2 V

Vgem 0V

* Vout = - . 0 + 2.5 . ( )
* Vout = 2,5 V

Vmin -0.5V

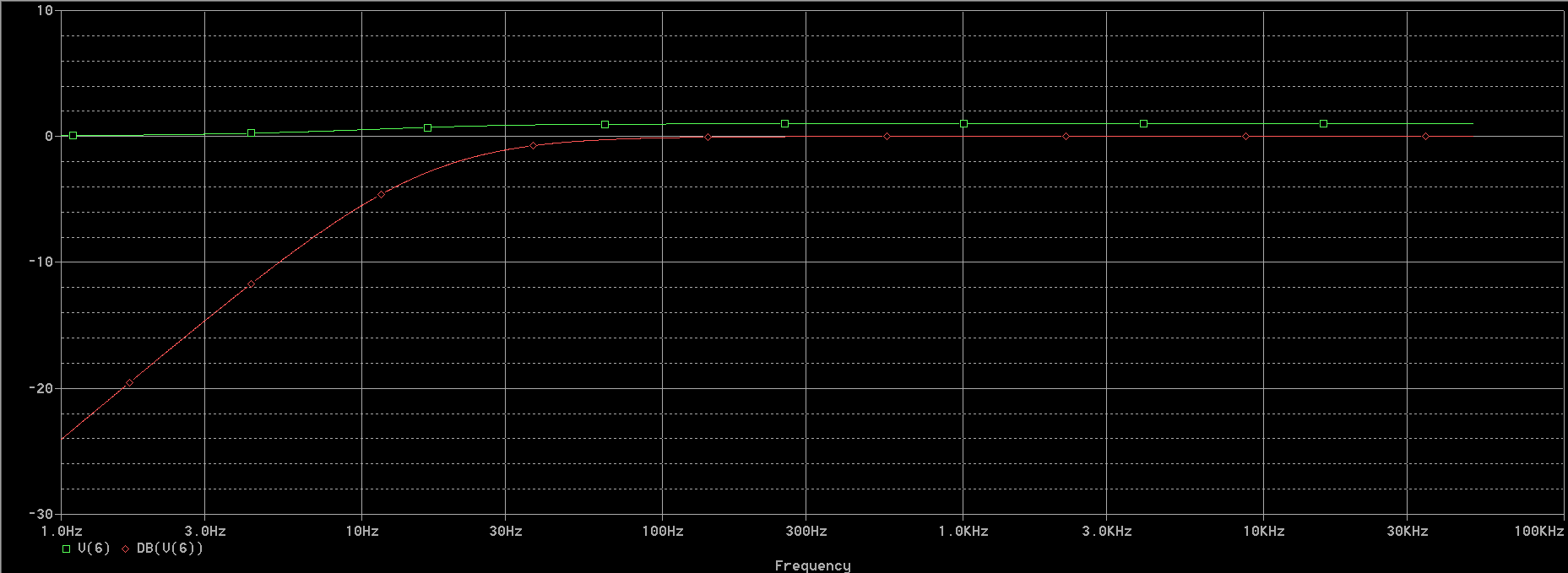
* Vout = - . (-0.5) + 2.5 . ( )
* Vout = 3 V

A = 1 +

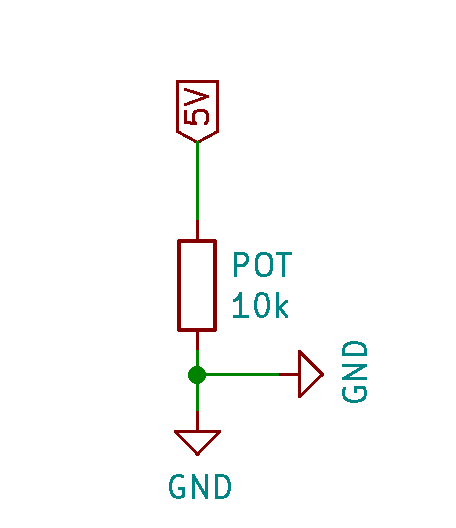
* A = -
* A = -1

fo =

* fo =
* fo = 15,195 Hz



### Phano Preampfilter

Dit is normaal een Phano Preampfilter maar wij hebben het shunting netwerk veranderd in een potentiometer, met daar achter condensatoren geplaatst. Dit zorgt ervoor dat de versterking wordt verhoogd afhankelijk van de potentiometer.

Als de potentiometer meer naar de maximum waarde staat, komt het dichter bij de max 5V spanning waardoor je die waarde binnen leest en dan zorgt de waarde van de potentiometer voor de Max vermogen overdracht.

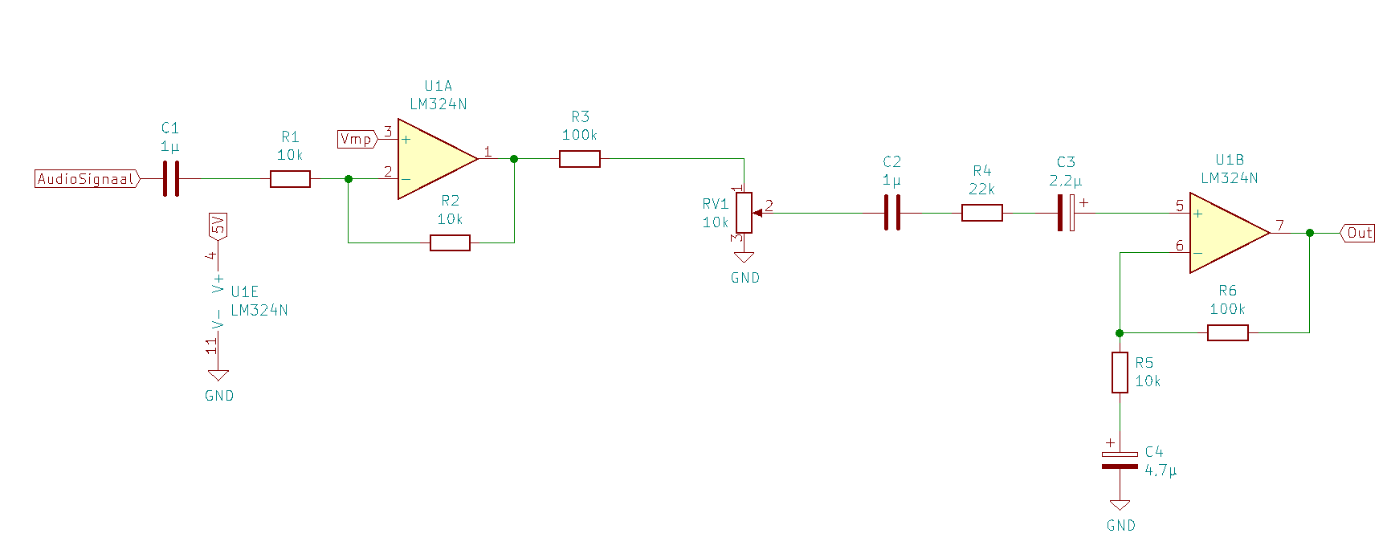
C1 op de tekening geeft het laag frequentie breek punt om DC en sub audio frequentie te onderdrukken. Als we de transfert functie willen berekenen moeten we C1 weglaten want daar zijn we niets mee bij deze berekening.

f1 =

* f1 = = 3,386 Hz

H = 1 +

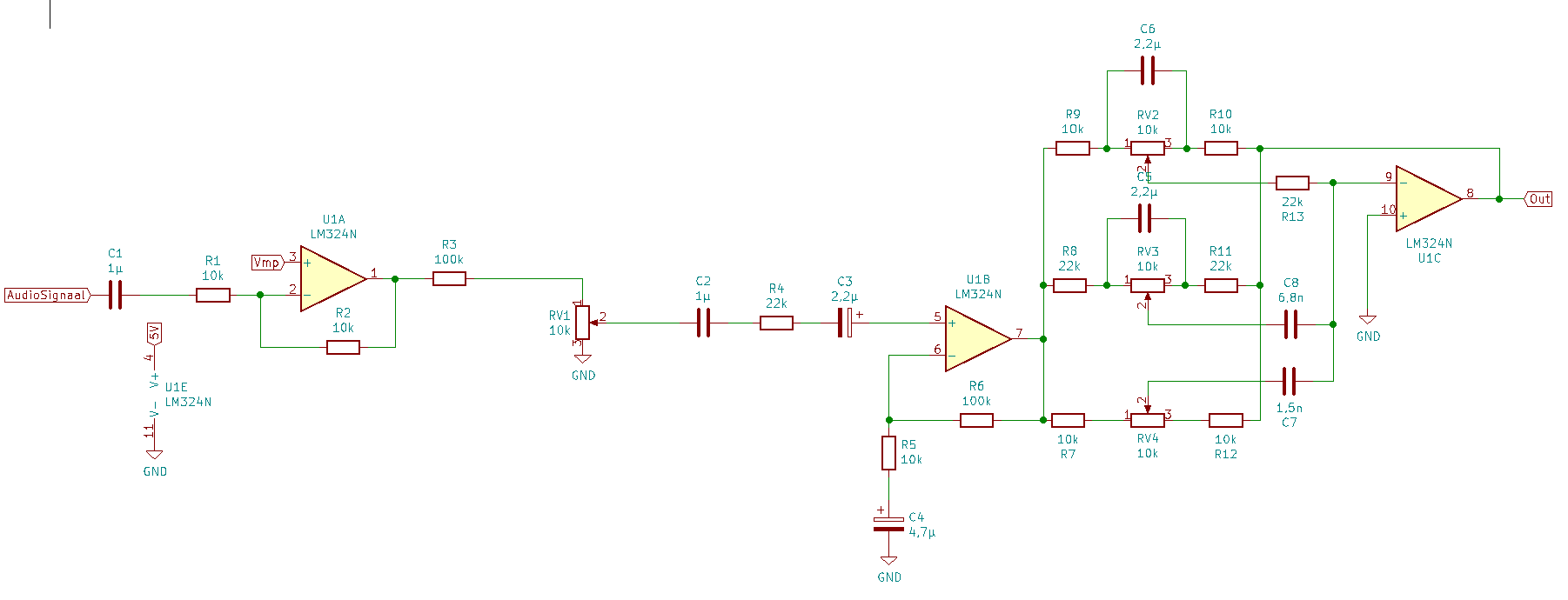
* H = 1 + = …



### Active tone control

Nu gebruiken we de Baxandall schakeling, en sturen van het medium, dit zorgt er voor om de bass, medium en treble aan te sturen onafhankelijk van de gain.

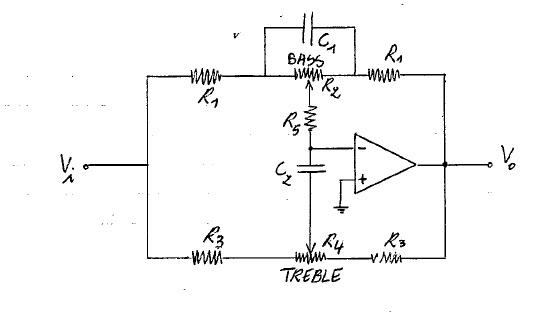
De BASS bestuurt alle lage tonen, ofwel te versterken of te dempen. MEDIUM zorgt voor het regelen van de middenfrequenties. Met de TREBLE kan je de hoge frequenties verhogen of verlagen.

BASS

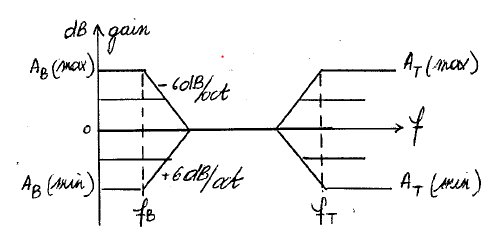
MEDIUM

TREBLE

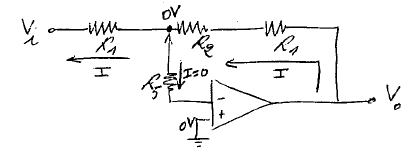
Wij wouden echt een mengpaneel hebben, dus wouden we bas, medium en treble apart kunnen aanpassen. Maar omdat we MEDIUM nog niet gezien hebben, was hier wat opzoekwerk voor nodig. Zo hebben we gevonden tot de conclusie dat MEDIUM eigenlijk een combo is van BASS en TREBLE.



Als we de BASS willen berekenen en de input frequentie is lager dan de fB kan men de condensator als een open keten beschouwen.



BASS schakeling



Hier wordt de potentiometer afgebeeld als het volledig op minimum staat. Samen met de docent hebben we besproken welke kringen het meeste stroom zouden trekken toen wij problemen hadden met onze potentiometer maar dat probleem vertellen we in het hoofdstuk datasheets. De baxandall schakeling zou het meeste stroom gebruiken, dus wou ik de stroom van de bas meten als het op minimum of maximum staat.

Potentiometer op minimum:

* Iminimum = =

Versterking:

* ABmin =
* 2

Potentiometer op maximum:

* Imaximum = =

Versterking:

* ABmax =
* 0,5

De frequentie van de BASS kan je berekenen met de formule:

* fB =
* 7,2343 Hz

Om de TREBLE te berekenen zullen op de hoge frequenties de condensators zich gedragen als een kortsluiting, waardoor de gain nu gecontroleerd wordt door de treble-potentiometer.

Potentiometer op minimum:

* Iminimum = =

Versterking:

* ATmin =
* 0,15625

Potentiometer op maximum:

* Imaximum = =

Versterking:

* ATmax =
* 6,4

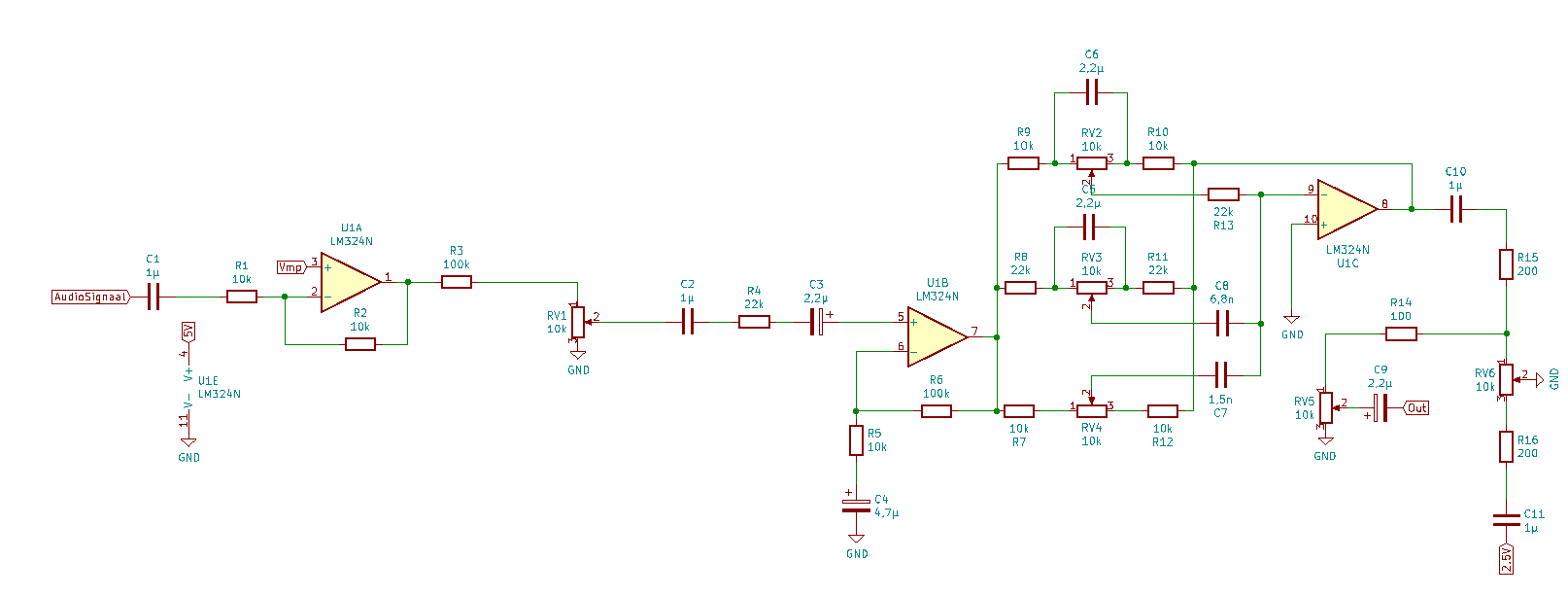
De frequentie van de TREBLE kan je berekenen met de formule:

* fT =
* 10610,32 Hz

### Links en rechts balancer + volume aan de uitgang regelen

**Balance of L-R:**benadruk de linker- of rechter speaker

Door dat we het beweegbaar deel van de potentiometer aan de ground zetten en de twee signalen op het einde van het linker- en rechterdeel van de potentiometer zetten, is er eigenlijkeen pull down weerstand aan de twee uitgangsaudiosignalen. Die zorgt er dan voor dat je de spanning gelijkt houdt tussen de twee audiosignalen. Hoe kleiner de weerstand waarde is hoe lager je spanning aan de kant met de kleinste weerstandswaarde van de pull down weerstand.



## **PCB lay-out**

Onze PCB is voorzien van verschillende componenten zoals weerstanden, condensatoren, LED’s en nog veel meer. Hier zie je welke component er op welke plaats zou meten komen.

Afbeelding met tafel

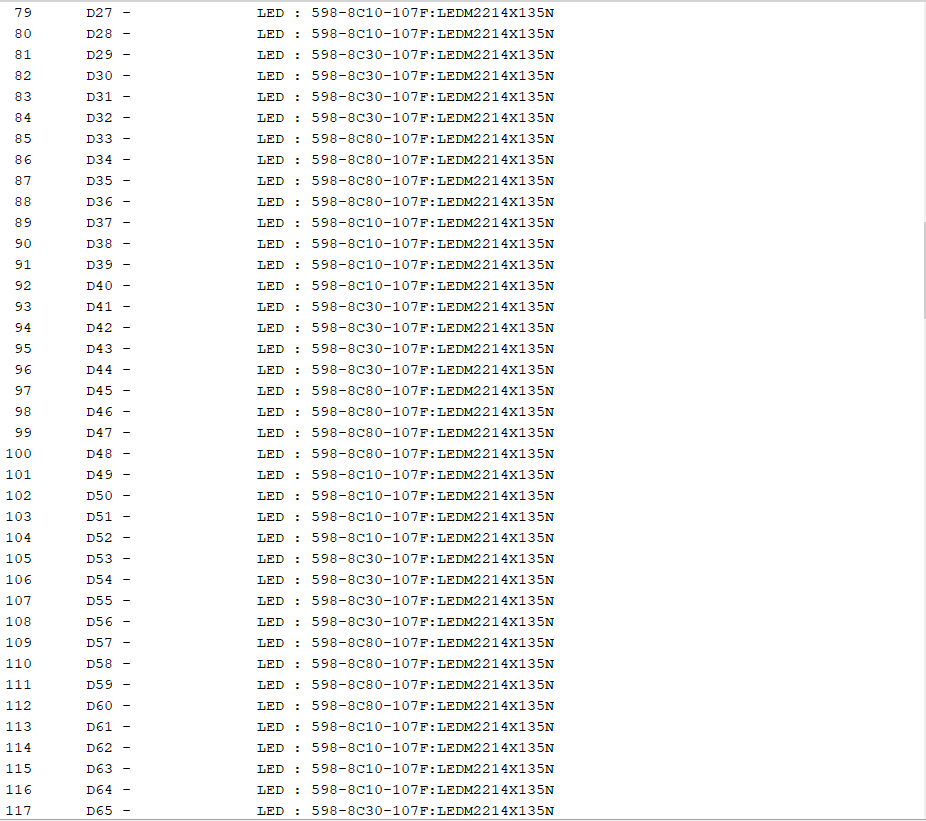
Automatisch gegenereerde beschrijving

Zoals je kan zien zijn de condensatoren niet allemaal van dezelfde reeks en hebben niet dezelfde grootte, dit is natuurlijk een schoonheidsfoutje die we zeker konden verbeteren. Maar we hebben naar condensatoren zitten zoeken die in stok waren en een lage tolerantie hadden.

Afbeelding met tafel

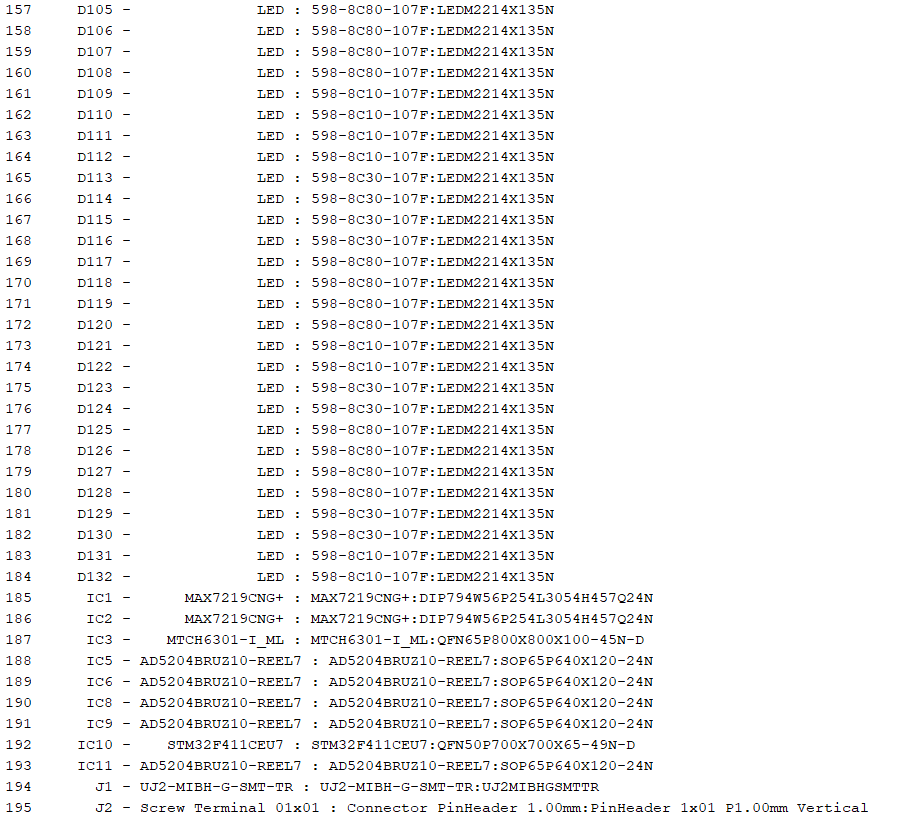
Automatisch gegenereerde beschrijving

Al onze leds zijn van de zelfde reeks en grootte omdat deze nog genoeg in voorraad waren en het oogt ook mooier als ze allemaal hetzelfde zijn. Wel zijn er verschillende kleuren, namelijk groen, oranje en rood.



Afbeelding met tafel

Automatisch gegenereerde beschrijving



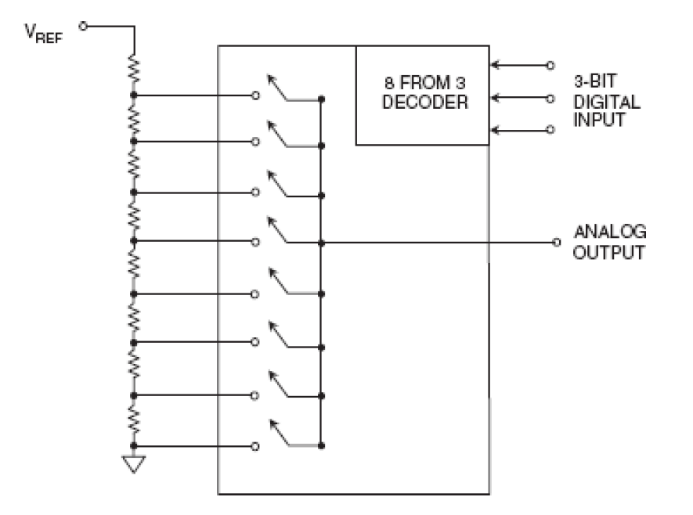
We hebben gekozen om de MAX7219CNG+ te gebruiken omdat dit een betrouwbare led driver is maar eigenlijk is het een 8-Digit LED Display Driver. Normaal zou het dus voorzien worden voor een 7 segment display, maar als je ze op de juiste manier zou aansluiten en programmeren kan je er ook een LED matrix van maken.

De MTCH6301-I\_ML is de IC die wij gekozen hebben om onze touchpad sliders mee aan te sturen als er een aanraking is met de sliders. Dit is capacitatieve touch, omdat deze beter zijn voor het bord en voor de verbindingen op het bord. Er waren veel mogelijke oplossingen om deze sliders te maken maar dit had het meeste voordelen.

Wij hebben gekozen voor de STM32F411CEU7 als microcontroller omdat we niet al te veel I/O nodig hebben, deze heeft 34 I/O. Qua RAM en FLASH geheugen hadden we niet echt zware criteria. We hebben ook voor deze moeten kiezen om dat er heel veel microcontrollers niet meer verkrijgbaar waren die aan onze eisen voldeden.

AD5204BRUZ10-REEL7 is een digitale potentiometer die voorzien zien is van 4 potentiometers. Dit kwam voordeliger uit voor ons dan met een IC met 6 potentiometers. Er waren veel verschillende soorten maar deze potentiometer had het voordeel dat het voorzien was met interne weerstanden in de chip waardoor je een soort van laddernetwerk hebt en dan krijg je een soort van echte potentiometer.





Afbeelding met tafel

Automatisch gegenereerde beschrijving

Afbeelding met tafel

Automatisch gegenereerde beschrijving

Afbeelding met tafel

Automatisch gegenereerde beschrijving

De touchpad hebben we zelf ontworpen, en zou moeten voldoen aan de normen die opgelegd worden in de datasheet van de controller om correct te werken.

LM324A zijn de opamps die we gebruiken op school en dat is ook heel gemakkelijk om onze oscilloscoop aan te hangen om te kijken hoe onze sinus verloopt.

We hebben ook gekozen voor een 8 MHz clock omdat die tussen de mogelijke clock tijden ligt die de microcontroller aan kan.

# **Software**

## **3.1 Flowchart**

Diagram, text

Description automatically generated

## **Listing**

De code is geschreven met het programma ‘System Workbench for STM32’, in C. Er is gebruik gemaakt van CubeMX om instellingen van gebruikte protocollen en GPIO correct in te stellen, en vanuit deze gegenereerde code is er verder gewerkt.

De volledige, en laatste versie van de code is te vinden op onze github repository, onder de map /Microcontrollercode/Muziekversterker\_v3/

Volledige link: <https://github.com/Hannes12072000/Muziekversterker_Hannes_Hugaert_and_Stijn_Van_den_Bossche/tree/main/Microcontrollercode/Muziekversterker_v3>

Om te beginnen worden er enkele globale variabelen bijgehouden, dezen zijn hieronder te zien.

uint8\_t ledValues[11];

uint8\_t potValues[20];

uint16\_t touchCoordinates[2];

**int** isTouchInitialized = 0;

De belangrijkste variabelen hiervan zijn de eerste drie. Die worden namelijk gebruikt om de huidige waardes van de LEDs (welke LEDs aan zouden moeten zijn), de huidige waardes van de potentiometers, en de coördinaten van de laatste aanraking van de touchpads bij te houden. Ze worden dan aangepast wanneer er een aanraking wordt gedetecteerd, en hierna uitgestuurd via SPI naar de relevante IC’s.

//Initialization of Potvalues to 0

**for**(**int** i=0;i<20;i++){

potValues[i]=0x00;

}

//L-R balance default op mid zetten

potValues[17]=128;

//initialization of LedValues to 0

**for**(**int** i=0;i<11;i++){

ledValues[i]=0x00;

}

initializePeripherals();

initializeLedDriver();

enableInterrupt();

Vervolgens wordt er in de main-functie enkele zaken geïnitialiseerd. Beide variabelen hierboven vermeld krijgen hier een startwaarde. Hierna worden enkele functies opgeroepen om verschillende IC’s correct in te stellen. Deze functies zijn verder verduidelijkt hieronder. Verder is ook te zien dat de functie enableInterrupt pas wordt opgeroepen na deze eerste initialisaties. Dit omdat de start-up sequence van de touchcontroller voorschrijft om eerst de nodige I²C initialisaties uit te voeren, en vervolgens, op een trigger van het interrupt hiervan afkomstig, om de instellingen van de touch controller naar de relevante registers te schrijven. (Zie afbeelding).



De overige nodige initialisaties zijn verder afgehandeld in het interrupt, dit wordt verder besproken.

**void** **initializePeripherals**(){

HAL\_GPIO\_WritePin(CS\_LEDS\_GPIO\_Port,CS\_LEDS\_Pin,*GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(CS\_POT1\_GPIO\_Port,CS\_POT1\_Pin,*GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(CS\_POT2\_GPIO\_Port,CS\_POT2\_Pin,*GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(CS\_POT3\_GPIO\_Port,CS\_POT3\_Pin,*GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(CS\_POT4\_GPIO\_Port,CS\_POT4\_Pin,*GPIO\_PIN\_SET*);

HAL\_GPIO\_WritePin(CS\_POT5\_GPIO\_Port,CS\_POT5\_Pin,*GPIO\_PIN\_SET*);

}

**void** **initializeLedDriver**(){

//LED driver (MAX7219)

uint8\_t data[4];

//set decode mode for both chips to no decode

data[0]=0x09;

data[1]=0x00;

data[2]=0x09;

data[3]=0x00;

writeToLedsRegister(data);

//Intensity setting; register 0x0A

//by default both set to max intensity, can be adjusted if some are brighter than others

data[0]=0x0A;

data[1]=0x0F;

data[2]=0x0A;

data[3]=0x0F;

writeToLedsRegister(data);

/\*\*scan-limit mode, set to 8 digits for first one, 3 for other one,

\* for a total of 11 "digits".

\* Each "digit" represents a slider with leds,

\* every segment represents a vertical led layer

\*/

data[0]=0x0B;

data[1]=0x07;

data[0]=0x0B;

data[1]=0x02;

writeToLedsRegister(data);

//enabling shutdown register

data[0]=0x0C;

data[1]=0x01; //writing 0x01 to enable

data[2]=0x0C;

data[3]=0x01;

writeToLedsRegister(data);

}

De functie initializePeripherals wordt simpelweg gebruikt om zeker te zijn dat alle CS-pinnen, gebruikt voor SPI-communicatie, om te beginnen zeker hoog zijn.

De functie initializeLedDriver gaat dan de benodigde parameters schrijven naar de led-drivers. Deze commandos zijn telkens 16 bits, eerste 8 bit zijn het relevante register, de tweede 8 bit zijn de data die naar dit register geschreven wordt. (Zie pagina 12 voor verschillende registerfuncties). We gebruiken twee leddrivers die gedaisychained zijn. Daarom worden er 2 maal 16 bits, voor een totaal van 32 bits, of 4 byte, doorgestuurd via SPI.

Het eerste commando stuurt naar het Decode Mode register, om in te stellen dat de data niet gedecodeerd moet worden, aangezien het geen 7-segment display is dat aangestuurd wordt.

Hierna wordt het Intensity Register ingesteld zodat alle LEDs op maximale helderheid zitten.  
Het Scan-Limit register stelt in hoeveel ‘digits’ er worden gebruikt per controller, in ons geval komt dit overeen met de kolommen LEDs, per slider.

Ten laatste wordt het Shutdown register zo ingesteld dat de chip uit shutdown gaat, en actief wordt.

**void** **HAL\_GPIO\_EXTI\_Callback**(uint16\_t GPIO\_Pin){

**if**(GPIO\_Pin == TOUCH\_INT\_Pin){

//first time interrupt is generated is for initialization for touchcontroller

**if**(!isTouchInitialized){

initializeTouchController();

isTouchInitialized = 1;

}**else**{

//interrupt generated by touchcontroller, should read out last touch now

getCoordinatesLastTouch(touchCoordinates);

//use touchcoordinates to write to LEDs & pots

processTouch();

//converting touchCoordinates to usable values for LEDs & pots

}

}

}

De functie HAL\_GPIO\_EXTI\_Callback handelt het interrupt af, afkomstig van de touchcontroller. De eerste maal dat dit wordt opgeroepen, gebeurt nadat de touchcontroller is opgestart en klaar is om ingesteld te worden.

In de functie initializeTouchController worden de registers dan ingesteld. In alle andere gevallen is de initialisatie al gebeurd op dat moment, en wordt een aanraking verwerkt door de functies getCoordinatesLastTouch en processTouch. Dezen zijn verder uitgelegd op pagina 33.

**void** **initializeTouchController**(){

//Touch controller (MTCH6301)

uint8\_t data[5];

//following guide to initialize component

//disable touch cmd

data[0]=0x55;

data[1]=0x01; //# of bytes to send

data[2]=0x01; //disable touch cmd

writeToTouchController(data,3);

//write parameters --> write to registers

//RX channels

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x00; //index location (see datasheet registers) here: general : 0x00

data[4]=0x01; //offset location here: RX channels : 0x01

data[5]=0x0B; //value to write here: 11 -> : 0x0B

writeToTouchController(data,6);

//TX channels

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x00; //index location (see datasheet registers) here: general : 0x00

data[4]=0x02; //offset location here: TX channels : 0x02

data[5]=0x0C; //value to write here: 12 -> : 0x0C

writeToTouchController(data,6);

//RX scaling <7:0>

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x00; //index location (see datasheet registers) here: general : 0x00

data[4]=0x04; //offset location here: RX scaling <7:0> : 0x04

data[5]=0x45; //value to write here: 5957 -> 0x1745 -> : 0x45

writeToTouchController(data,6);

//RX scaling <15:8>

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x00; //index location (see datasheet registers) here: general : 0x00

data[4]=0x05; //offset location here: RX scaling <15:8> : 0x05

data[5]=0x17; //value to write here: 5957 -> 0x1745 -> : 0x17

writeToTouchController(data,6);

//TX scaling <7:0>

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x00; //index location (see datasheet registers) here: general : 0x00

data[4]=0x06; //offset location here: TX scaling <7:0> : 0x06

data[5]=0x55; //value to write here: 5461 -> 0x1555 -> : 0x55

writeToTouchController(data,6);

//TX scaling <15:8>

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x00; //index location (see datasheet registers) here: general : 0x00

data[4]=0x07; //offset location here: TX scaling <15:8> : 0x07

data[5]=0x15; //value to write here: 5461 -> 0x1555 -> : 0x15

writeToTouchController(data,6);

//Sensormapping

uint8\_t registermap[12] ={0x00,0x01,0x02,0x03, 0x04,0x05,0x06,0x07, 0x08,0x09,0x0A,0x0B};

// RX map = {8,7,3,10, 6,1,5,0, 4,9,2}

uint8\_t RX\_mapping[11] ={0x08,0x07,0x03,0x0A, 0x06,0x01,0x05,0x00, 0x04,0x09,0x02};

// TX map = {13,6,3,2, 4,30,29,28, 7,14,15,16}

uint8\_t TX\_mapping[12] ={0x0D,0x06,0x03,0x02, 0x04,0x1E,0x1D,0x1C, 0x07,0x0E,0x0F,0x10};

//RX Mapping

**for**(**int** i=0; i<11 ;i++){

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x01; //index location (see datasheet registers) here: sensor map RX : 0x01

data[4]=registermap[i]; //offset location here: RX map i : see registermap above

data[5]=RX\_mapping[i]; //value to write here: RX value i : see RX mapping above

writeToTouchController(data,6);

}

//TX Mapping

**for**(**int** i=0; i<12 ;i++){

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x02; //index location (see datasheet registers) here: sensor map TX : 0x02

data[4]=registermap[i]; //offset location here: TX map i : see registermap above

data[5]=TX\_mapping[i]; //value to write here: TX value i : see TX mapping above

writeToTouchController(data,6);

}

//Decoding :

//Flip State

data[0]=0x55;

data[1]=0x04; //# of bytes to send

data[2]=0x15; //write to register cmd

data[3]=0x30; //index location (see datasheet registers) here: decoding : 0x30

data[4]=0x02; //offset location here: flipstate : 0x02

data[5]=0x04; //value to write here: 0b100 : 0x04

writeToTouchController(data,6);

//Possibly extra values to change: subject to change

//send scan baseline command

data[0]=0x55;

data[1]=0x01; //# of bytes to send

data[2]=0x14; //scan baseline cmd

writeToTouchController(data,3);

//enable touch cmd

data[0]=0x55;

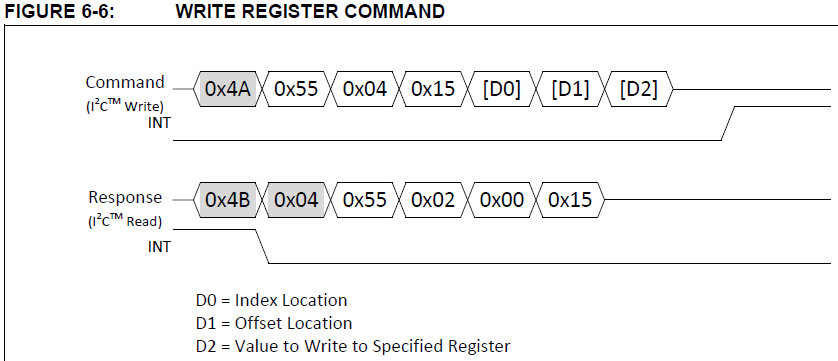
data[1]=0x01; //# of bytes to send

data[2]=0x00; //enable touch cmd

writeToTouchController(data,3);

}

Op de vorige twee pagina’s staat de code gebruikt om de touch controller te initialiseren. Een groot deel van deze code is herhaaldelijk verschillende waardes schrijven naar verscheidene registers, om instellingen juist in te stellen voor het beoogde gebruik. De manier om naar registers te schrijven van deze controller bestaat uit verschillende opeenvolgende bytes, conform de datasheet.

****

De eerste byte is het adres van de slave, telkens 0x4A, dit inclusief de bit die duidt op write/read. Vervolgens een vaste byte gebruikt om naar registers te schrijven, 0x55. De derde byte beduidt het aantal hierop volgende bytes. De laatste 3 bytes zijn telkens een register adres, de offset binnen in dit register, en de waarde die hiernaar geschreven dient te worden.

Op deze manier worden de volgende parameters ingesteld: het aantal RX channels (voor ons aantal sliders), aantal TX channels (aantal onderverdelingen per slider), RX en TX Scaling (Dezen zijn afhankelijk van het aantal gebruikte channels, exacte waarde per gekozen aantal channels is vermeld in de datasheet), de gebruikte mapping van TX en RX channels (dezen zijn niet normaal oplopend, deze waardes waren terug te vinden in de datasheet) en flipstate (richting van beide assen, en of de assen al dan niet geïnverteerd zijn).

Hierna wordt er naar een baseline gescand, en de touch terug enabled, conform de gebruikelijke start-up sequence vermeld op pagina 29.

**void** **getCoordinatesLastTouch**(uint16\_t touchCo[]){

uint8\_t databuf[20];

readTouches(databuf,6); //6 expected

/\*\*byte 0 = amount total bytes

\* byte 1 = TOUCHID, not really relevant

\* byte 2 = part of touch X coordinate

\* byte 3 = other part of touch X coordinate

\* byte 4 = part of touch Y coordinate

\* byte 5 = other part of touch Y coordinate

\*/

//writing X out

touchCo[0] = (((uint16\_t)databuf[3])<<7) + ((uint16\_t)databuf[2]);

//writing Y out

touchCo[1] = (((uint16\_t)databuf[5])<<7) + ((uint16\_t)databuf[4]);

}

**void** **processTouch**(){

/\* no arguments needed, bc all values used and written to

\* are saved as global variables.

\* That being:

\* touchCoo, ledValues & potValues

\*/

//depending on these values, needs to be converted into led values & pot values

//Touchcontroller get values from 0->4096 for both axis

**int** i=0; //used to loop

**int** j=0;

uint8\_t slider=50; //give a random value that will never be reached

uint8\_t sliderheight=50;

//checking for slider with loop

**while**(i<11 && slider==50){

**if**(i\*TOUCHSCALING\_X < touchCoordinates[0] && touchCoordinates[0] < (i+1) \* TOUCHSCALING\_X){

slider = i;

}

i++;

}

//default value if something went wrong

**if**(slider==50){

slider=0;

}

//checking for heightvalue with loop

**while**(j<12 && sliderheight ==50){

**if**(j\*TOUCHSCALING\_Y < touchCoordinates[1] && touchCoordinates[1] <(j+1)\*TOUCHSCALING\_Y){

sliderheight = j;

}

j++;

}

//default value if something went wrong

**if**(sliderheight==50){

sliderheight=0;

}

//Setting LED values to relevant value

**switch**(sliderheight){

**case** 0:

ledValues[slider]=0b00000001;

**break**;

**case** 1:

ledValues[slider]=0b00000011;

**break**;

**case** 2:

ledValues[slider]=0b00000111;

**break**;

**case** 3:

ledValues[slider]=0b00001111;

**break**;

**case** 4:

ledValues[slider]=0b00011111;

**break**;

**case** 5:

ledValues[slider]=0b00111111;

**break**;

**case** 6:

ledValues[slider]=0b01111111;

**break**;

**case** 7:

ledValues[slider]=0b11111111;

**break**;

//for cases higher, also just write max value

**case** 8:

ledValues[slider]=0b11111111;

**break**;

**case** 9:

ledValues[slider]=0b11111111;

**break**;

**case** 10:

ledValues[slider]=0b11111111;

**break**;

**case** 11:

ledValues[slider]=0b11111111;

**break**;

**default**:

ledValues[slider]=0b00000000;

**break**;

}

//setting potValues to relevant values

**switch**(slider){

**case** 0: //BASS LINE1 -> POT6 & POT11

potValues[5]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[10]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 1: //MID LINE1 -> POT7 & POT10

potValues[6]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[9]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 2: //TREBLE LINE1 -> POT8 & POT9

potValues[7]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[8]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 3: //VOL LINE1 -> POT2 & POT3

potValues[1]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[2]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 4: //VOL LINE2 -> POT4 & POT5

potValues[3]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[4]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 5: //BASS LINE2 ->POT12 & POT17

potValues[11]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[16]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 6: //MID LINE2 -> POT13 & POT16

potValues[12]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[15]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 7: //TREBLE LINE2 -> POT14 & POT15

potValues[13]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[14]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 8: //MIC VOL -> POT1

potValues[0]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 9: //MASTER OUT -> POT19 & POT20

potValues[18]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

potValues[19]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**case** 10: //BALANCE L-R -> POT18

potValues[17]= POTVALUE\_SCALING \* (uint8\_t)sliderheight;

**break**;

**default**:

//niks doen

**break**;

}

//Call functions that handle writing to IC's

setLedValues();

setPotValues();

}

De functies getCoordinatesLastTouch en processTouch worden telkens opgeroept in het interrupt gegenereerd door de touch controller. Eerst wordt er in getCoordinatesLastTouch de waarden ontvangen, en opgeslagen. In de functie erna worden deze coördinaten dan verwerkt en de LEDs en potentiometers afhankelijk van deze waarden aangepast. De touchcontroller geeft coördinaten door gaande van 0 tot 4096, dus deze waarden worden herleid naar een overeenkomstige waarde voor één van de 11 sliders, en één van de overeenkomstige 12 onderverdelingen van deze slider. Wanneer deze twee waardes gevonden zijn, gebeurt er een conversie naar een handig formaat van opslag voor de waarden die moeten ingesteld worden voor de LEDs. De LEDs worden namelijk opgeslagen als een array van uint8\_t, met elk element van deze array zijnde een kolom LEDs. Per kolom zijn er 8 LEDs aan te sturen, elk 1 bit van deze uint8\_t. Oorspronkelijk was het de bedoeling om 12 LEDs per kolom aan te sturen, maar ten gevolge van het incorrect interpreteren en aansluiten van de LED drivers kunnen we maar 8 kolommen van 8 LEDs aansturen.

Op een vergelijkbare manier worden hierna de beoogde waarden voor de potentiometers aangepast. Per slider zijn er twee potentiometers die aangepast dienen te worden om de versterking aan te passen. De waarden worden eerst nog herschaald naar een waarde tussen 0 en 255 (uint8\_t), aangezien de potentiometers hiermee aan te sturen zijn.

Nadat de touchcoördinaten zijn omgezet naar overeenkomstige waarden voor de LEDs en voor de potentiometers, worden deze waardes, opgeslagen in de globale variabelen, verstuurd naar de IC’s met behulp van de functies setLedValues en setPotValues.

**void** **setLedValues**(){

//Decoding led values to instructions for led drivers

/\* Noticed a flaw in way led-matrix is set up,

\* will very likely not be possible to drive LEDs of sliders 9,10 or 11,

\* neither will LED layers 9,10,11 and 12 be drivable.

\* There will still be LEDs drivable on Sliders 1-8,

\* and of those Led layers 1-8

\*

\*

\* Data stored is in order:

\* 7:0 (led layers)

\* D7,D0,D1,D2, D3,D4,D5,D6

\*/

/\* looping over different ledsliders

\* LedSlider Mapping : see below

\*/

uint8\_t data[4]; //used to write data out

uint8\_t ledSliderMap[8]= {0x01,0x02,0x03,0x04, 0x05,0x06,0x07,0x08};

**for**(**int** ledSlider=0;ledSlider<8;ledSlider++){

//easiest to write these manually

uint8\_t ledValue=0x00;

ledValue = ledValue | ( ledValues[ledSlider] & 0b10000000); //writing bit 7

ledValue = ledValue | ((ledValues[ledSlider] & 0b00000001)<<6); //writing bit 6

ledValue = ledValue | ((ledValues[ledSlider] & 0b00000010)<<4); //writing bit 5

ledValue = ledValue | ((ledValues[ledSlider] & 0b00000100)<<2); //writing bit 4

ledValue = ledValue | ( ledValues[ledSlider] & 0b00001000); //writing bit 3

ledValue = ledValue | ((ledValues[ledSlider] & 0b00010000)>>2); //writing bit 2

ledValue = ledValue | ((ledValues[ledSlider] & 0b00100000)>>4); //writing bit 1

ledValue = ledValue | ((ledValues[ledSlider] & 0b01000000)>>6); //writing bit 0

data[0]=ledSliderMap[ledSlider];

data[1]=ledValue;

data[2]=0x00; //no-op code, second controller doesn't need to display bc of problem noted above

data[3]=0x00;

writeToLedsRegister(data);

}

}

**void** **setPotValues**(){

uint8\_t data[2];

uint8\_t addressMap[4]={0b00000000, 0b00000001, 0b00000010, 0b00000011};

**for**(**int** pot=0;pot<5;pot++){

**for**(**int** pot\_comp=0;pot\_comp<4;pot\_comp++){

data[0]=addressMap[pot\_comp];

data[1]=potValues[pot\*4 + pot\_comp];

writeToPotentiometers(data,2,pot+1);

}

}

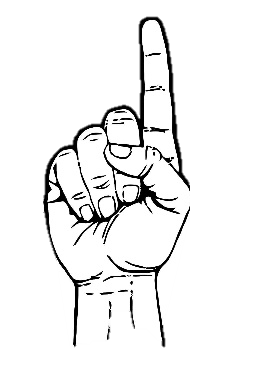
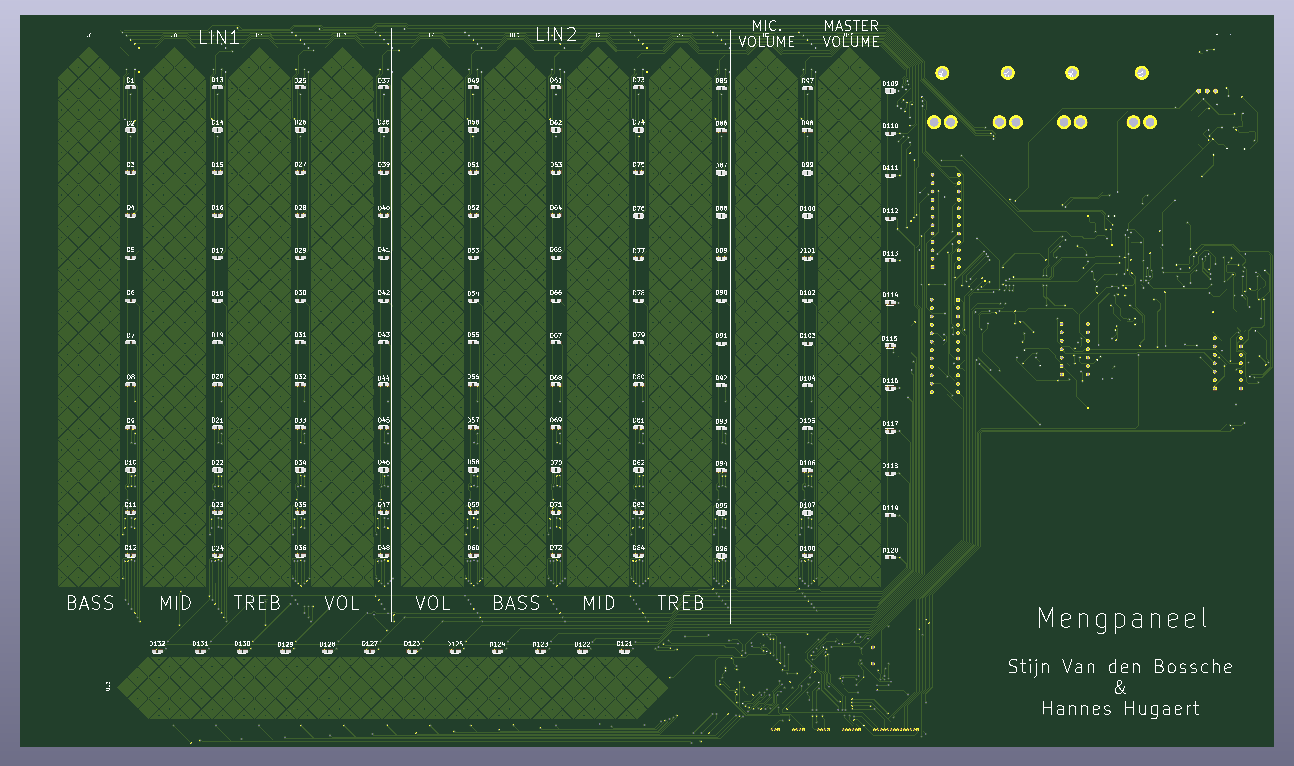
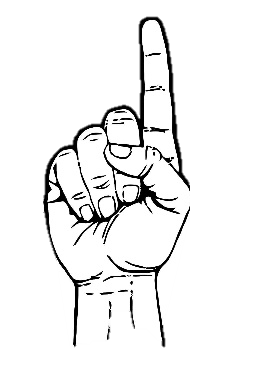
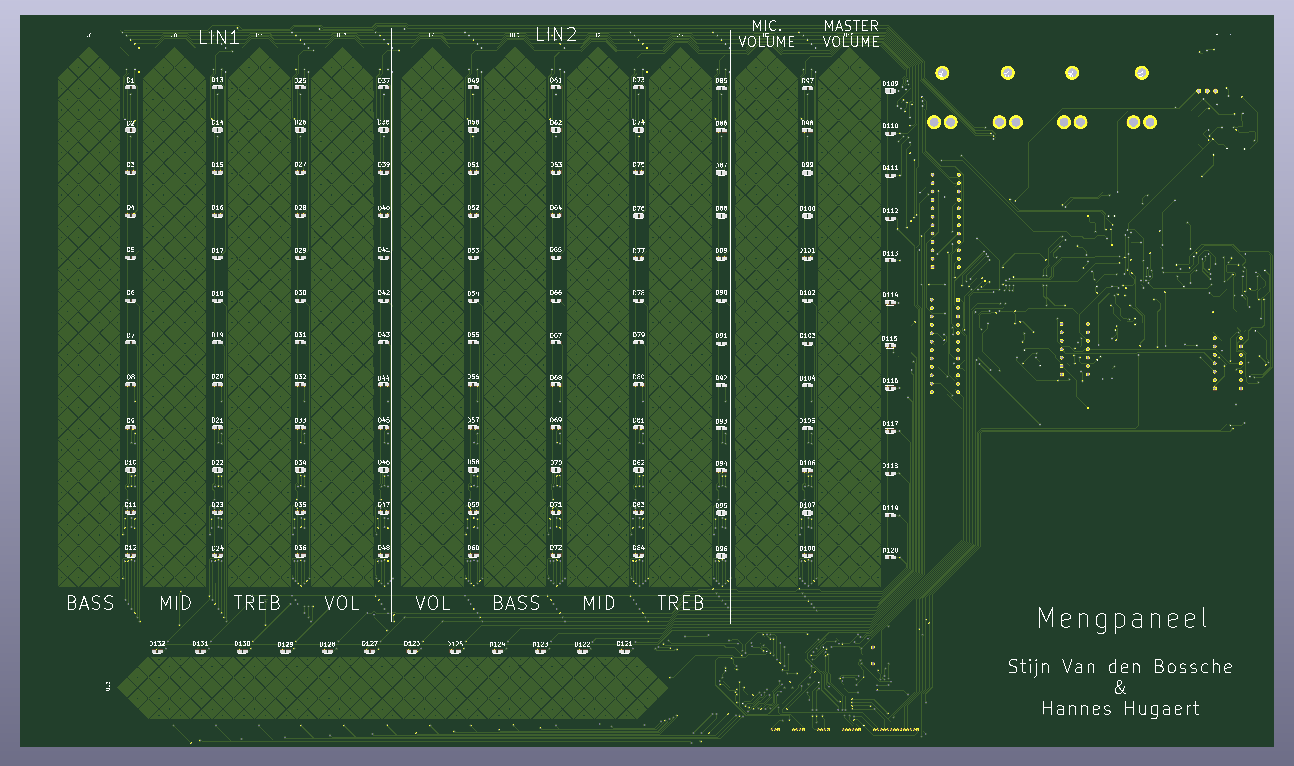
}

Om de LED waardes te schrijven, gebeurt eerst nog een conversie. De waardes zijn immers opgeslagen in een 8bit formaat, MSB is hiervan de hoogste LED per slider. Dezen zijn echter verschillend verbonden op de PCB. De overeenkomstige segmenten op een 7-segment display zijn DP, en vervolgens secties A-G, in MSB-formaat, zoals opgeslagen in de globale variabele. Op de PCB zijn ze aangesloten als DP als MSB, en vervolgens secties G-A, dus deze conversie corrigeert dit.

# **Bediening handleiding**

Ons mengpaneel werkt via touchpads, dus kan je door je vinger op de touchpad de stand van de bass, medium en treble of volumes regelen. Als je je vinger vanonder zet dan staat de potentiometer op 0% als je je vinger gaat bewegen naar boven gaat de stand van de vingers bepalen welke stand je potentiometer komt te staan en de LEDs lichten ook op om de huidige stand weer te geven. Helemaal bovenaan staat de potentiometer op 100%.

We hebben 2 lijnen voorzien voor audio, deze twee lijnen worden op het einde bij elkaar gevoegd, op deze manier kan men verschillende audiosignalen met elkaar mixen. Ook kan je via lijn 1 een microfoon plaatsen in plaats van een audiosignaal.



# **Besluit**

Aanvankelijk verliep alles goed op schema, we hadden vrij snel een idee uitgewerkt, en algemeen hoe we alles zouden aansturen en visualiseren. Er waren verschillende manieren om ons audiosignaal te verwerken, maar dit doen via digitale potentiometers leek ons van in het begin het beste, als we dit analoog hadden gedaan, hadden we zeer waarschijnlijk veel duurdere en accurate componenten moeten gebruiken, onder andere als slider, om te verzekeren dat de geluidskwaliteit goed ging zijn. Daarom kozen we ervoor om deze verwerking in software uit te voeren.

Ook in het bedienen waren er verschillende keuzes, onder andere mechanische sliders. Dezen hadden ook vrij duur uitgedraaid, dus hebben we geopteerd om voor capacitatieve touch te kiezen. Dit leek ons de beste optie, hierdoor waren er geen blootgestelde pads die eventueel kapot konden gaan, of slecht contact konden hebben. Hiervoor hebben we ook vrij snel een IC voor gevonden om dit aan te sturen.

De touchpads hebben we zelf ontworpen en op één grote print gezet, aangezien dit mooier ging zien qua uitzicht. Als we allemaal verschillende, kleinere touchpads hadden gemaakt, moesten dezen ook sowieso nog aan elkaar worden gemonteerd, dit had nog extra moeilijkheden kunnen veroorzaken.

Tijdens de paasvakantie was de eerste versie van de print klaar, en na de paasvakantie waren al onze zichtbare fouten weg. Dus gingen we onze print bestellen tot we zelf nog een foutje vonden met de potentiometers, dus hebben we dezen nog moeten aanpassen zodat ze zouden werken op de virtuele grond. Hierdoor waren we de week na de paasvakantie klaar en hebben we onze PCB besteld, samen met nog enkele missende componenten. We hebben drie weken moeten wachten op onze PCB omdat die vastzat bij het verzenden, hier zijn we echter veel kostbare tijd kwijtgeraakt. We hebben het grootste deel thuis gesoldeerd en in de les getest.

Tijdens het testen hebben we echter nog enkele grote problemen ondervonden, onder andere de voeding van de PCB was niet correct verbonden. Ook kregen sommige componenten niet genoeg voedingsspanning. Tijdens het uitzoeken waar hier ergens het probleem zat, is onze microcontroller ook stuk gegaan. Daardoor zijn we verder gegaan met gebruiken van het STM32F476G-DISCO bordje om alle componenten aan te sturen.

Hannes Hugaert heeft de hardware gedaan, Stijn Van den bossche heeft de software geschreven. Het concept en uitwerken van de opdracht is samen gebeurd.

# **Bronnen**

Al onze datasheet schema’s en tabellen hebben we gehaald van mouser:

<https://www.mouser.be/>

De I²C pulsen zijn van:

<https://nl.wikipedia.org/wiki/I%C2%B2C-bus>