Practice Enterprise Electronics 2

2022 – 2023

Thomas More
William Van Raemdonck

Inleiding

Voor Project Practice Enterprise 2 heb ik een audioversterker gemaakt. Deze versterker is zodanig opgebouwd dat hij zowel een mono-en stereosignaal kan versterken. Er zit ook een IC in die tooncontrole doet, die op zijn beurt wordt aangestuurd door een microcontroller.

Verder ben ik in de zomervakantie van vorig jaar al gestart aan dit project door een development bordje te maken voor de ATtiny828. Dit heb ik vooral gedaan omdat ik niet meer met de XC888 van vorig jaar wou werken. Dit bordje heeft mij ook toegelaten om al code te schrijven voor de PCB geleverd was. De details hierover staan ook in de bijlage.

Bijlage in aparte documenten:
https://github.com/WilliamVanRaemdonck/Project_Practise_Enterprise_2_Amplifier

> *“Do or do not. There is no try.”*
> ***Yoda - The Empire Strikes Back***

Dankwoord

Het maken van deze versterker tijdens het afgelopen academiejaar was een leerrijke periode. Ik had deze taak nooit zo goed afgerond zonder de hulp van een paar mensen.

Een bijzonder woord van dank gaat uit naar meneer Dams, die de tijd nam om constructieve feedback te geven. Zijn wijze raad als mentor heeft me geholpen om problemen te analyseren. Hij stelde me goeie vragen die ervoor gezorgd hebben dat ik oplossingsgericht kon werken.

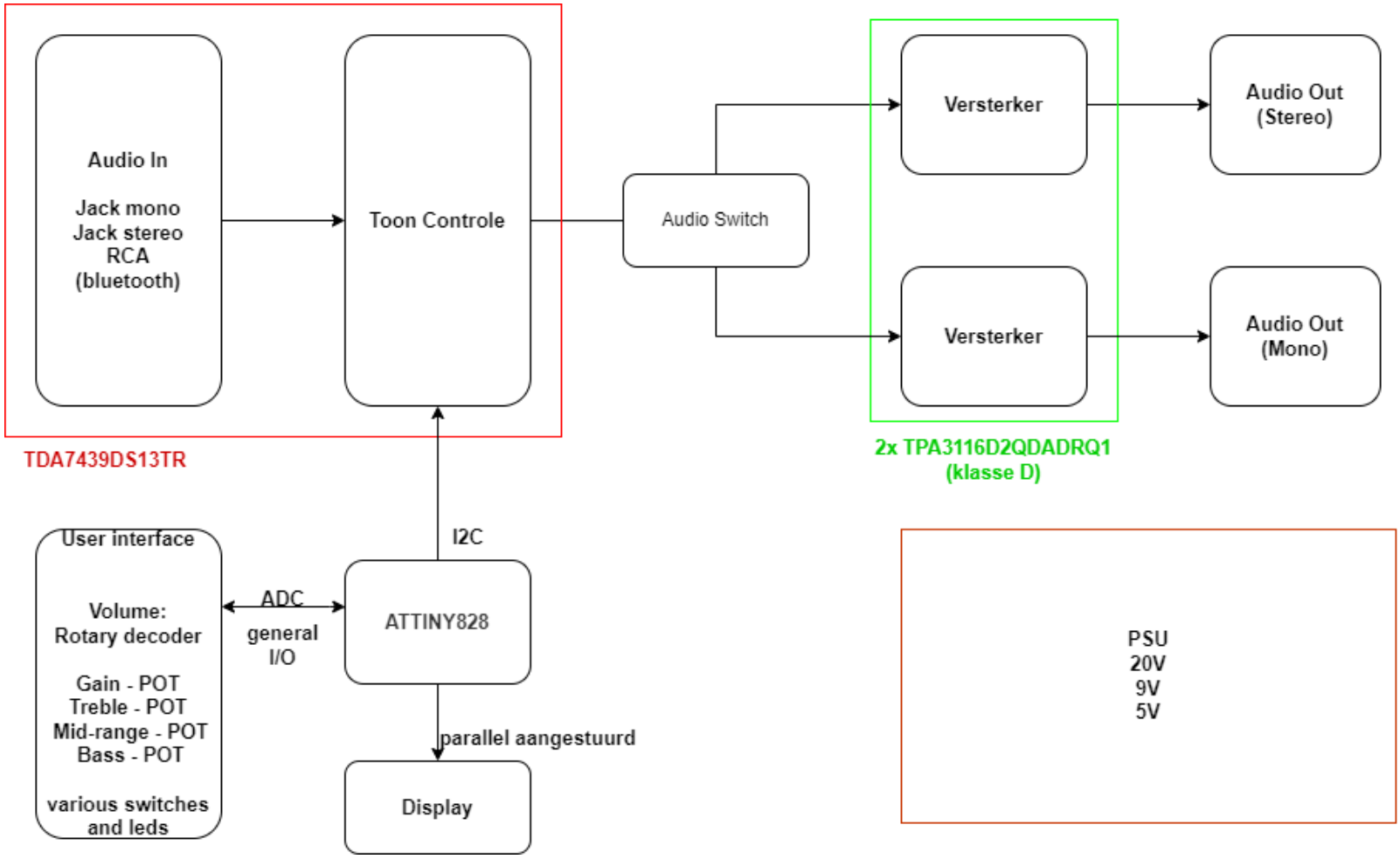
Ik wil eveneens meneer De Weerdt en meneer Pelgrims bedanken voor het delen van hun kennis.

Ook mijn vrienden en familie hebben me steeds aangemoedigd zodat ik me gesteund voelde om dit project tot een goed einde te brengen.

William Van Raemdonck

Contents

Blokschema


Audio In
Jack mono
Jack stereo
RCA
(bluetooth)
Toon Controle
TDA7439DS13TR
Audio Switch
Versterker
Versterker
Audio Out
(Stereo)
Audio Out
(Mono)
2x TPA3116D2QDADRQ1
(klasse D)
User interface
Volume:
Rotary decoder
Gain - POT
Treble - POT
Mid-range - POT
Bass - POT
various switches
and leds
ADC
general
I/O
ATTINY828
I2C
parallel aangestuurd
Display
PSU
20V
9V
5V

Uitleg Blokschema

De power supply levert 20V voor de versterker IC’s, 9V voor de tooncontrole IC en 5V voor de microcontroller en het lcd-paneel.

De tooncontrole IC wordt aangestuurd door een microcontroller via een software I2C verbinding.

De microcontroller gebruikt zijn ADC en I/O pinnen om de gain, bass, midrange, treble, volume waardes in te lezen. Hij gaat hier dan omzettingen op doen om daarna door te sturen naar de audiocontrole IC.

De display is aangestuurd via een 4 bit parallelle verbinding met de microcontroller.

Componenten

ATtiny828:

Dit is een 8 Bit controller die met een klok frequentie van 8Mhz.

Deze IC is vergelijkbaar met de XC888 dat we vorig jaar moesten gebruiken met het belangrijk verschil dat deze gemakkelijk in C geprogrammeerd kan worden en iets minder I/O pinnen heeft.

De belangrijkste delen van deze microcontroller zijn:
Hardware SPI.
Hardware I2C slave.
Full Duplex USART
2 timers waarvan één 16 bit.
32 ADC kanalen (10 bit resolutie).
20MHz kloksnelheid.

De IC wordt geprogrammeerd via de ICSP pinnen van een Arduino, je kan ook een AVR programmer kopen. Ik heb deze optie gekozen omdat deze gratis was.

Om code in deze controller te krijgen staat er een condensator op de reset pin van een Arduino nadat deze zelf geprogrammeerd is als ISP programmer. Hierna zal Microchip studio zijn code sturen naar AVRDUDE om de code naar de Arduino en dus ook de ATtiny te sturen. AVRDUDE kent de ATtiny828 niet maar er is een override optie zodat deze de code toch uploadt.

TDA7439DS:

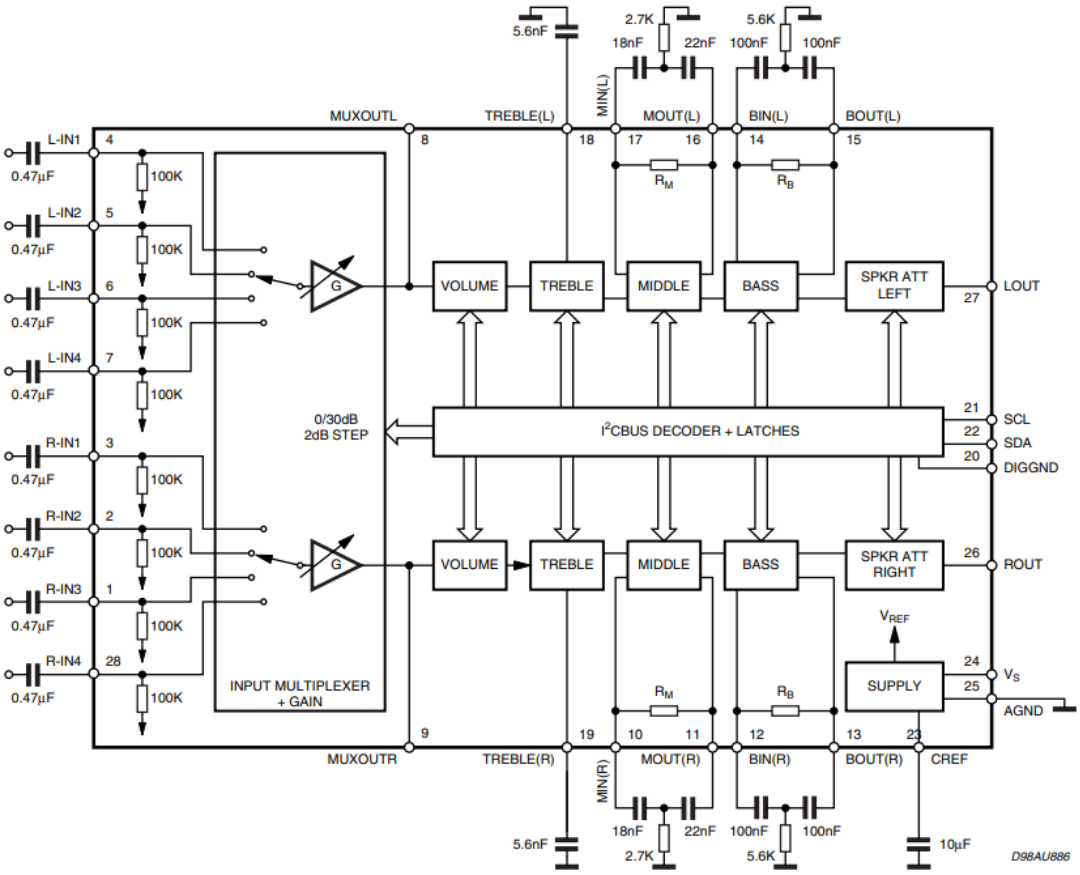
Deze IC gaat de audiocontrole doen, hij is aangestuurd via de ATtiny controller via een software I2C verbinding.
Er bestaan registers waarbij de waardes in de figuur hiernaast kunnen worden aangepast.

Function
Input selector
Input gain
Volume
Bass gain
Mid-range gain
Treble gain
Speaker attenuation, R
Speaker attenuation, L

Figuur 1: Registers

Bass, mid-range en treble kunnen ingesteld worden tussen -14 dB en +14 dB.
De input gain tussen 0 en 30 dB.
Volume tussen 0 en -40 dB.

Verder is er nog een input select register om één van de vier input signalen te selecteren.



Figuur 2: Waardes uit datasheet

TPA3116D2-Q1:

Twee van deze Ic's zitten in deze versterker. Één als stereo en één als mono versterker.
Deze IC heeft een systeem dat fouten detecteert. Het is aangesloten zodat fouten gecleared worden wanneer de fout verdwijnt.

7.3.8 Device Protection System

The TPA311xD2-Q1 family contains a complete set of protection circuits to make system design efficient as well as to protect the device against any kind of permanent failures due to short circuits, overload, overtemperature, and undervoltage. The $\overline{\text{FAULT}}$ pin signals if an error is detected according to Table 4:

Table 4. Fault Reporting

FAULT	TRIGGERING CONDITION (typical value)	$\overline{\text{FAULT}}$	ACTION	LATCHED OR SELF-CLEARING
Overcurrent	Output short or short to PVCC or GND	Low	Output high impedance	Latched
Overtemperature	T_j > 150°C	Low	Output high impedance	Latched
Too-high dc offset	DC output voltage	Low	Output high impedance	Latched
Undervoltage on PVCC	$V_{(PVCC)}$ < 4.5 V	–	Output high impedance	Self-clearing
Overvoltage on PVCC	$V_{(PVCC)}$ > 27 V	–	Output high impedance	Self-clearing

Gekozen instelling:

PVCC = 20V.

PV_{CC} (V)	PLIMIT VOLTAGE (V)(1)	R to GND	R to GVDD	OUTPUT VOLTAGE (V_{rms})
24 V	GVDD	Open	Short	17.9
24 V	3.3	45 kΩ	51 kΩ	12.67
24 V	2.25	24 kΩ	51 kΩ	9
12 V	GVDD	Open	Short	10.33
12 V	2.25	24 kΩ	51 kΩ	9
12 V	1.5	18 kΩ	68 kΩ	6.3

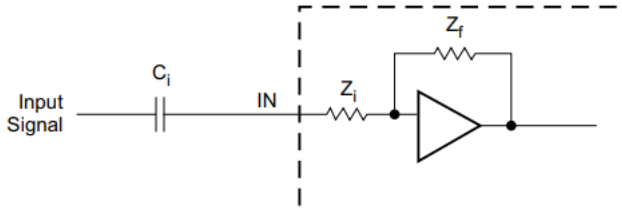
Gain = 26dB.

MASTER / SLAVE MODE	GAIN	R1 (to GND)(1)	R2 (to GVDD)(1)	INPUT IMPEDANCE
Master	20 dB	5.6 kΩ	OPEN	60 kΩ
Master	26 dB	20 kΩ	100 kΩ	30 kΩ
Master	32 dB	39 kΩ	100 kΩ	15 kΩ
Master	36 dB	47 kΩ	75 kΩ	9 kΩ
Slave	20 dB	51 kΩ	51 kΩ	60 kΩ
Slave	26 dB	75 kΩ	47 kΩ	30 kΩ
Slave	32 dB	100 kΩ	39 kΩ	15 kΩ
Slave	36 dB	100 kΩ	16 kΩ	9 kΩ

Input condensatoren.

Table 2. Recommended Input AC-Coupling Capacitors

GAIN	INPUT IMPEDANCE	INPUT CAPACITANCE	HIGH-PASS FILTER
20 dB	60 kΩ	1.5 µF	1.8 Hz
26 dB	30 kΩ	3.3 µF	1.6 Hz
32 dB	15 kΩ	5.6 µF	2.3 Hz
36 dB	9 kΩ	10 µF	1.8 Hz



Oscillator frequentie.

f_{OSC}	Oscillator frequency	AM[2:0] = 000	376	400	424	kHz
		AM[2:0] = 001	470	500	530	
		AM[2:0] = 010	564	600	636	
		AM[2:0] = 011	940	1000	1060	
		AM[2:0] = 100	1128	1200	1278	
		AM[2:0] = 101	Reserved			
		AM[2:0] = 110				
		AM[2:0] = 111				

Belangrijke waardes.

Vout ~ = 14,916V.
Rspeaker = 4Ω.
Imax = 3,729A.
Gain = 26dB.

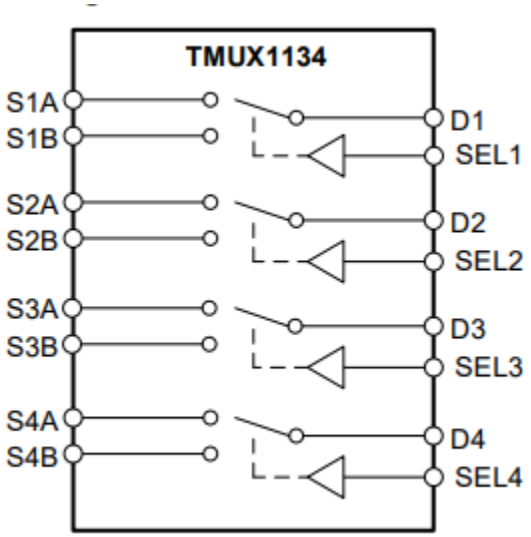
Koeling.

De datasheet zegt dat deze koelvin moet gebruikt worden.
Datasheet:
https://www.mouser.be/ProductDetail/984-ATS-TI1OP521C1R1

TMUX1134:

Deze IC schakelt van tussen de stereo-en mono versterker door het audiosignaal van de ene versterker naar de andere te sturen. Tegelijkertijd wordt de versterker IC die niet gebruikt wordt ook gemuted zodat de uitgang tevens ook hoog impedant gaat.

Het audiosignaal heeft een 3,8V dc-offset en het mute signaal zit tussen de 0 en 5V. Dus deze IC kan dit perfect aan. Verder gebruik ik deze IC ook omdat ik zo geen extra transistor of FET moest bijplaatsen om het MUTE-signaal te kunnen regelen.



Figuur 3: Audio switch

Display:

Deze display is parallel aangesloten aan de microcontroller. De 1e en 3e lijn zijn constante strings die tijdens initialisatie worden doorgestuurd.
De lijnindicator die het volume aanduidt wordt op runtime gemaakt. De string die de input aangeeft wordt geselecteerd met een look up table.
Voor de display in te stellen zijn de volgende instellingen gebruikt.
Function Set:

- 4 bit interface
- 2 lines
- Font size 8 x 5

De display heeft ook 2 andere lijnen: de enable en RS-lijn. De enable wordt gebruikt als een kloklijn om nieuwe data binnen te klokken. De RS-lijn bepaalt of je naar een data-of instructieregister schrijft. Verder is er ook nog de R/W lijn maar in dit geval hangt deze aan ground omdat er alleen maar geschreven moet worden.

Om bij te houden naar welke plaats op de display er geschreven moet worden wordt gebruik gemaakt van het DDRAM-adres. Startende van 0x80 tot 0xe7.

Figuur 4: Display

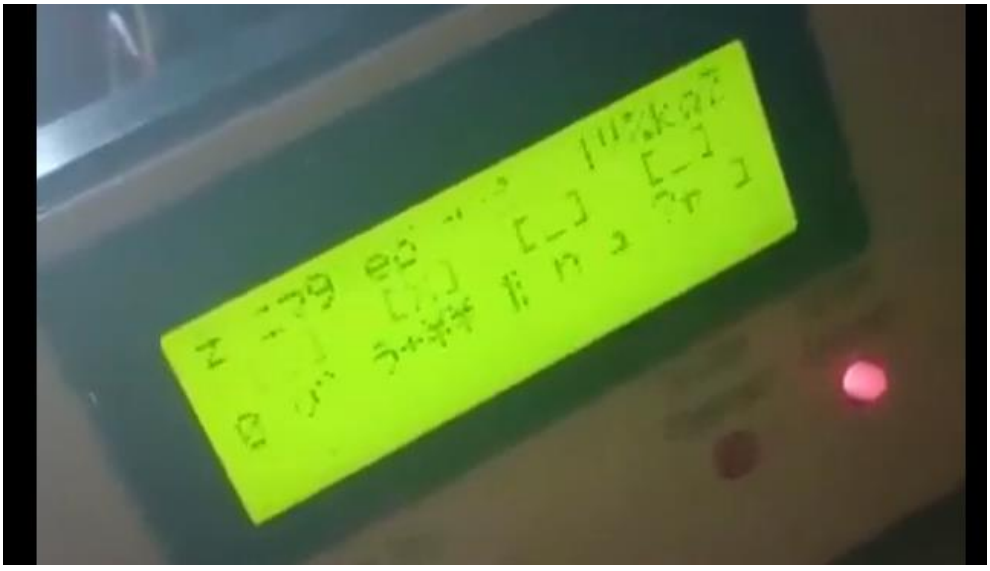
Problemen Display:

Tijdens het testen van de versterker is deze display wat onverklaarbaar gedrag beginnen vertonen. Hij begint willekeurige karakters op het scherm te tonen.
Dit gebeurt soms als de spanning heel snel op en afgezet wordt. Verder heb ik dit ook één keer zien gebeuren wanneer ik een audio input aansloot.

Om dit te verhelpen heb ik meer condensatoren bijgezet op de voedingslijnen van de display en power supply.

Verder dacht ik ook dat het misschien te maken had met de power supply die niet snel genoeg zijn spanning liet wegvallen. Ik had een extra load weerstand bijgezet op de 5V maar dit hielp niet. Verder zou dit ook willen zeggen dat de schakelende voeding ook geen nut meer heeft aangezien er toch vermogen verloren wordt door deze weerstand.

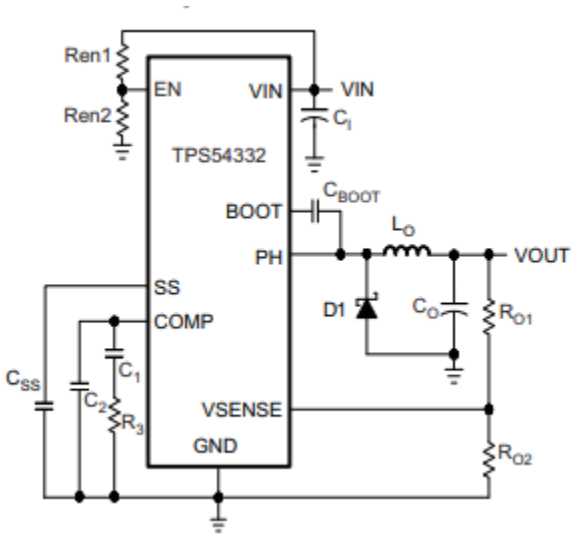
De laatste hypothese die ik heb is dat ik deze display in mijn eerste jaar elke dag mee naar school heb genomen. Ik fiets elke dag 1u over soms niet ideale wegen. Misschien is de display ergens kapot vanbinnen door deze constante schokken. Dit lijkt mij wel een beetje vergezocht.



Figuur 5: Foto anomalie

Power supply:

De power supply voorziet 20V, 9V en 5V voorzien. Voor de 5V en 9V zijn er schakelende spanningsregelaars.
Deze voeding is zodanig ontworpen dat deze ook andere spanningen kan leveren. De spanningsregelaars werken met een feedbacksysteem waarmee de uitgangsspanning kan worden ingesteld.
Verder kan de print ook afgebroken worden van de versterker PCB zodat deze in andere projecten kan worden gestoken.
De schakelende regelaar is de TPS54332DDA.



Figuur 6: Schema regelaar

Besluit

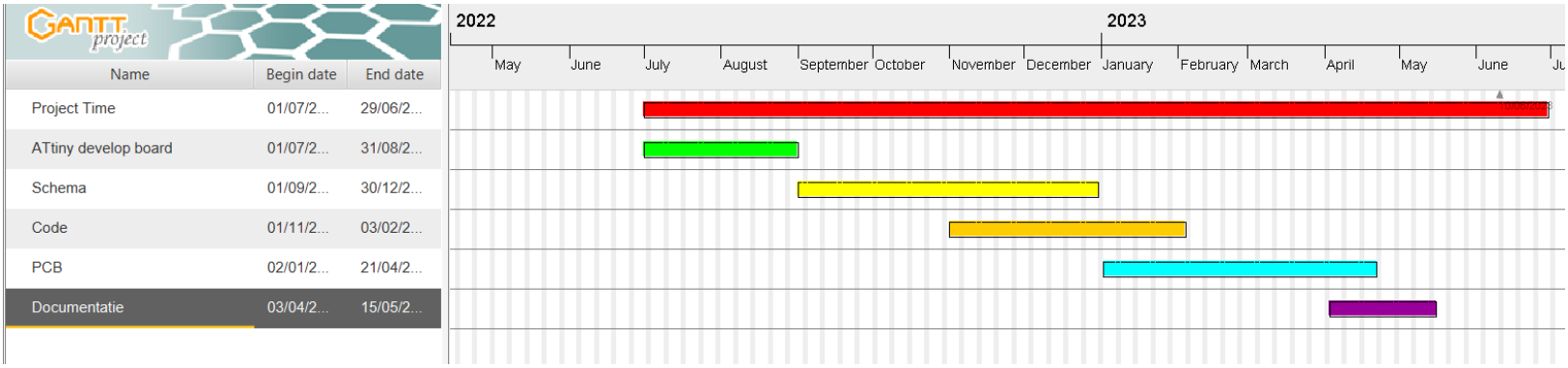
Dit was een zeer leerzaam project, vooral omdat ik niet veel van audioversterkers afwist voor ik aan dit project begon. Verder heb ik wel een paar puntjes dat ik anders zou doen naar de toekomst toe.

De PCB is redelijk groot en ik denk dat de Pcb's nog steeds modulair hadden kunnen zijn op een kleinere oppervlakte. De 8 debug-leds waren ook een beetje overbodig en zijn tijdens het maken van het project eigenlijk nooit gebruikt.

Voor de microcontroller en audiocontrole denk ik dat het ook handiger had geweest mocht er gebruikt zijn gemaakt van een DSP en een STM-controller. Dit had ook geholpen met een kleinere PCB en had uploaden van code iets gemakkelijker gemaakt.
De audiocontrole IC die ik gebruik is namelijk al 16 jaar oud.

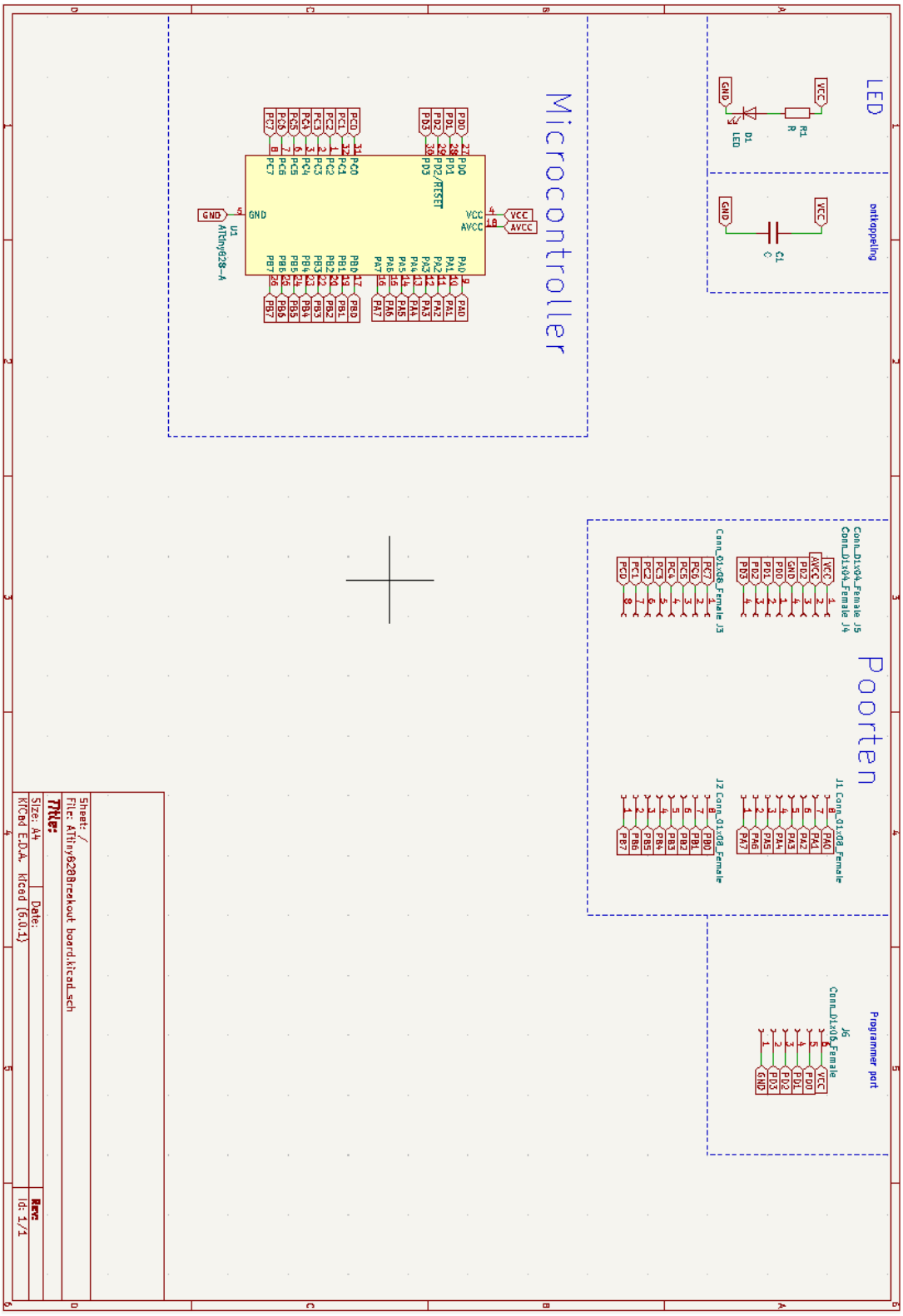
Bij het begin van dit project had ik ook de beslissing gemaakt om thuis eerst eens te kijken wat ik nog had liggen zodat ik niet alles nieuw moest bestellen. Wat ik hieruit ook geleerd heb is dat je best alle componenten die je hergebruikt eens nameet en test voordat je ze in een project steekt. Dit helpt ook om mogelijke fouten in het debug stadium van het project rapper op te sporen.

Gantt Chart

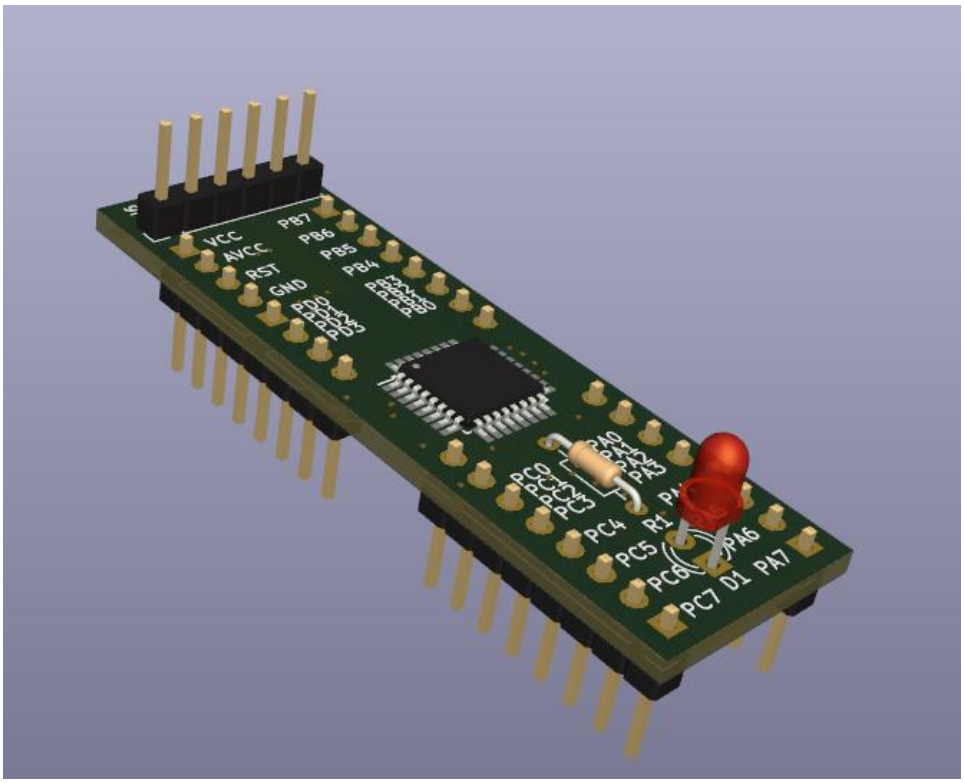

GANTT project
Name
Begin date
End date
Project Time
01/07/2...
29/06/2...
ATtiny develop board
01/07/2...
31/08/2...
Schema
01/09/2...
30/12/2...
Code
01/11/2...
03/02/2...
PCB
02/01/2...
21/04/2...
Documentatie
03/04/2...
15/05/2...
2022
2023
May
June
July
August
September
October
November
December
January
February
March
April
May
June

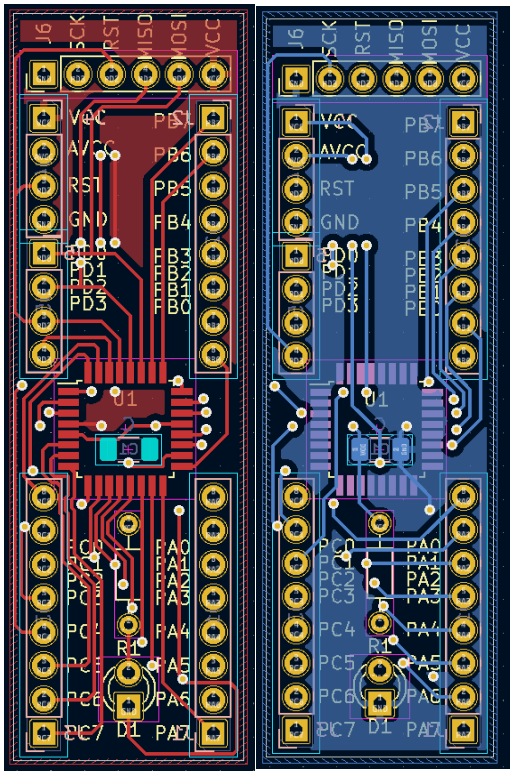
Test bord

Schema:



Pcb:


VCC
AVCC
RST
GND
PB7
PB6
PB5
PB4
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
R1
D1
PA6
PA7


J6
SCK
RST
MISO
MOSI
VCC
AVCC
GND
PB7
PB6
PB5
PB4
PD1
PD3
PB3
PB2
PB1
PB0
U1
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
R1
D1

Code

Main.c

```
/*
* Created: 25/10/2022 11:19:27
* Author : William VR
*/
/*includes*/
#include "main.h"

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <math.h>

#include "I2C/I2C.h"
#include "TDA/TDA.h"
#include "Display/Display.h"
#include "EEPROM/EEPROM.h"

//variables             input values
volatile uint8_t volume = 0x00;
uint8_t mux = 0x00;
uint8_t gain = 0x00;
uint8_t bass = 0x00;
uint8_t midRange = 0x00;
uint8_t treble = 0x00;
uint8_t muteLeds = 0x00;

int main(void)
{
        //clk
        initclk();

        //IO
        initIO();

        //ADC
        initADC();

        //TIMER1
        initTimer1();

        //I2C
        initI2C();

        //DISPLAY
        initDisplay();

        //ENABLE INTERRUPTS
        sei();

        //read volume value
        volume = EEPROM_read(0x00);

        //unmute TDA IC
        setTDAValue(CHIP_ADDRESS, SubAdr_Speaker_attenuation_L, 0x00);
        setTDAValue(CHIP_ADDRESS, SubAdr_Speaker_attenuation_R, 0x00);
```

```
	setTDAValue(CHIP_ADDRESS, SubAdr_Volume, 0x00);

	loadingScreen();


	while (1)
	{
		//read inputs
		gain = ReadADCPinValue(0b00001000);				//PB0
		bass = ReadADCPinValue(0b00001011);				//PB3
		midRange = ReadADCPinValue(0b00001100);		//PB4
		treble = 255 - ReadADCPinValue(0b00001101);	//PB5

		//mux PC2 PC3 PC4 PC5
		mux = 0;
		if (PINC & (1 << PINC3)) mux = 1;
		if (PINC & (1 << PINC4)) mux = 2;
		if (PINC & (1 << PINC5)) mux = 3;

		//TDA update
		setTDAValue(CHIP_ADDRESS, SubAdr_Input_selector, mux);
		setTDAValue(CHIP_ADDRESS, SubAdr_Input_gain, ((gain >> 4) & 0b00001111));
		setTDAValue(CHIP_ADDRESS, SubAdr_Volume, convert6bits((255 - volume)));
		setTDAValue(CHIP_ADDRESS, SubAdr_Bass_gain,  convert4bits(bass));
		setTDAValue(CHIP_ADDRESS, SubAdr_Mid_range_gain,  convert4bits(midRange));
		setTDAValue(CHIP_ADDRESS, SubAdr_Treble_gain,  convert4bits(treble));

		//Display update -> parallel
		updateDisplay(volume, mux);

		//write to EEPROM
		EEPROM_write(0x00, volume);
	}
}

//------------------------------------------------------------------------------	IO
void initIO(){
	//PORT A
	DDRA = 0xff;		//output
	PUEA = 0xff;		//Set pull ups
	PORTA = 0x00;		//write zero
	//PORT B
	DDRB = 0b11000000;		//output
	PUEB = 0xff;			//Set pull ups
	PORTB = 0b00000000;		//write zero
	//PORT C
	DDRC = 0b11000000;		//output
	PUEC = 0xff;			//Set pull ups
	PORTC = 0b00000000;		//write zero
}

//------------------------------------------------------------------------------	ADC
void initADC(){
	PRR		&= 0b11111110;		//turn power saving off
	ADCSRA |= 0b10000000;		//turn ADEN on
	ADMUXB &= 0b00000000;		//Voltage reference VCC
	ADCSRA |= 0b00000100;		//set Scaler 0100 => /16
	ADCSRB |= 0b00001000;		//left adjusted
}

uint8_t ReadADCPinValue(uint8_t ADCReadPin){
	ADMUXA = ADCReadPin;
```

```
	// Start ADC conversion
	ADCSRA |= (1 << ADSC);

	// Wait for ADC conversion to complete
	while (ADCSRA & (1 << ADSC));

	// Return ADC result
	return ADCH;
}



//------------------------------------------------------------------------------	Timer1
void initTimer1(void){
	PRR		&= 0b11110111;					//power saving off

	TCCR1B |= (1 << WGM12 );				// Configure timer 1 for CTC mode
	OCR1A = 0;

	TIMSK1 |= (1 << OCIE1A );				// Enable CTC interrupt
	TCCR1B |= (1 << CS11 ) | (1 << CS10 );		// prescaler of 64
	//TCCR1B |= (1 << CS00 );				// no prescaling
}

//------------------------------------------------------------------------------	MISC
// src: https://stackoverflow.com/questions/2602823/in-c-c-whats-the-simplest-way-to-reverse-the-order-of-bits-in-a-byte
uint8_t reverse(uint8_t b) {
	b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;
	b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
	b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
	return b;
}

void initclk(void){
	//Set prescaler to 1;
	CCP	= 0xD8;
	CLKPR = 0x00;
}

//------------------------------------------------------------------------------	ISR
ISR(TIMER1_COMPA_vect, ISR_BLOCK){
	static uint8_t volumeSwitchState = 0x00;

	/*
	0x00	READ
	0x01	IDLE	*/

	//rotary encoder
	switch (volumeSwitchState)
	{
		case 0x00:
		if(PINB & (1<<PINB2)){				// clk = 0?
			if(PINB & (1<<PINB1)){			// data = 0
				if(volume >= 5){
					volume -= 5;
				}
				volumeSwitchState = 0x01;
			}
			else{					// data = 1
				if(volume <= 250){
					volume += 5;
				}
```

```
                                        volumeSwitchState = 0x01;
                                }
                        }
                        break;

                        case 0x01:
                        if(PINB & (1<<PINB2)){                                  // clk still low?
                                //stay
                        }
                        else{
                                volumeSwitchState = 0x00;
                        }
                        break;
        }

}
```

Main.h

```
/*
 * Main.h
 * Created: 03/03/2023 12:22:17
 *  Author: William VR
 */


#ifndef MAIN_H_
#define MAIN_H_

/*defines*/
#define F_CPU 8000000UL
#define CHIP_ADDRESS 0x88

#include <avr/io.h>
#include <util/delay.h>

/*functions*/

//MISC
void	initIO();
void initclk(void);
uint8_t reverse(uint8_t);

//ADC
void	initADC();
uint8_t	ReadADCPinValue(uint8_t);

//TIMER0
void initTimer1(void);



#endif /* MAIN_H_ */
```

Display.c

```
/*
* Display.c
*
* Created: 03/03/2023 12:40:01
*  Author: William VR
*          https://www.farnell.com/datasheets/50586.pdf
*          pg 48
*
*   pin 4 RS                    -           PINC6
*   0 instruction register
*   1 data register
*
*   pin 5 always 0
*   0       writing
*
*   pin 6 enable        -           PINC7
*   X/0 enable writing
*   X/1 enable reading
*/

#include "../main.h"
#include "Display.h"
#include <string.h>
#include <stdio.h>

//vars
char inpStrText[] = "INPUT:            ";
char volStrText[] = "VOLUME:";

char loadingStrText[] = "      Loading      ";
char emptyStrText[] = "                   ";
char fullbarStrText[] = "YYYYYYYYYYYYYYYYYYYY_";

char* muxTable[4] = {
        " [X] [_] [_] [_] ",
        " [_] [X] [_] [_] ",
        " [_] [_] [X] [_] ",
        " [_] [_] [_] [X] "
};

char barStr[20];
char* muxStr;

uint8_t bars = 0x00;
uint8_t blanks = 0x00;
uint8_t first = 0x00;

void initDisplay(void){
        //init
        setRS();
        setEnable();
        PORTA = 0x00;
        //power on delay
        _delay_ms(500);

        clearEnable();
        clearRS();
        _delay_ms(500);
```

```
	//send function set 3 times
	sendByteByNibble(FUNCTION_SET);    //Function set
	_delay_ms(15);       //>4.1

	sendByteByNibble(FUNCTION_SET);    //Function set
	_delay_ms(5);        //>1.67ms

	sendByteByNibble(FUNCTION_SET);    //Function set
	_delay_ms(2);        //>1.67ms

	sendByteByNibble(FUNCTION_SET_4BIT);         //Function set         - 4bit mode
	_delay_ms(2);        //>1.67ms

	sendByteByNibble(CURSOR_ON_BLINK);           // display on, cursor on, blink on
	_delay_ms(2);        //>1.67ms

	sendByteByNibble(ENTRY_MODE);      // ready to write
	_delay_ms(2);        //>1.67ms

	sendByteByNibble(RETURN_HOME);     // return home
	_delay_ms(2);        //>1.67ms

	clearLCD();
}

void updateDisplay(uint8_t displayValueF, uint8_t mux){
	//1 VOLUME:                                                address: 0x80
	//2 ??????_____________  address: 0xc0
	//3 INPUT:                                                 address: 0x94
	//4 [] [] [?] []                                 address: 0xd4

	// volume string
	bars = displayValueF / 12; // 0 - 255 => 0 - 20 => / 12.75 ~ 12
	blanks = 20 - bars;

	//assemble string
	strcpy(barStr, "");
	for(uint8_t i = 0; i < bars; i++){
		strcat(barStr, "Y");
	}
	for(uint8_t i = 0; i < blanks; i++){
		strcat(barStr, "_");
	}

	//mux string
	muxStr = muxTable[mux];

	cursorHome();

	writeToDisplay(volStrText, strlen(volStrText), 0x80);

	// write entire bar on reset
	if(first == 0x00){
		writeToDisplay(fullbarStrText, strlen(fullbarStrText), 0xc0);
		first = 0x01;
	}
	else{
		writeToDisplay(barStr, strlen(barStr), 0xc0);
	}

	writeToDisplay(inpStrText, strlen(inpStrText), 0x94);
	writeToDisplay(muxStr, strlen(muxStr), 0xd4);
}
```

```
void writeToDisplay(char data[], uint8_t length, uint8_t DDRAMaddress){
	uint8_t addressF = DDRAMaddress;

	cursorHome();

	for(int i = 0; i < length;i++){
		sendByteByNibble(addressF);
		_delay_us(2);

		setRS();
		sendByteByNibble(data[i]);
		clearRS();

		addressF++;
	}
}

void loadingScreen(void){
	sendByteByNibble(CURSOR_ON_BLINK);		// display on, cursor on, blink on
	_delay_ms(2);		//>1.67ms

	sendByteByNibble(ENTRY_MODE);			// ready to write
	_delay_ms(2);		//>1.67ms

	clearLCD();
	cursorHome();

	writeToDisplay(emptyStrText, strlen(emptyStrText), 0x80);
	writeToDisplay(loadingStrText, strlen(loadingStrText), 0xc0);

	for(uint8_t i = 0; i < 20; i++){
		strcat(barStr, "Y");
		writeToDisplay(barStr, strlen(barStr), 0x94);
	}

	clearLCD();
	cursorHome();
}

void sendByteByNibble(char data){
	uint8_t input = 0x00;
	uint8_t leftNibble = 0x00;
	uint8_t rightNibble = 0x00;
	uint8_t mask = 0b11110000;

	//X to blok
	if(data == 0b01011001){			//data == Y ?
		leftNibble = 0xFF;
		rightNibble = 0xFF;
	}
	else{
		input = (uint8_t)data;

		leftNibble = (input & mask);

		input = (((uint8_t)data) << 4);
		rightNibble = (input & mask);
	}

	//data send
	PORTA = leftNibble;
```

```
        //pulse
        setEnable();
        _delay_us(50);
        clearEnable();
        _delay_us(50);

        //data send
        PORTA = rightNibble;

        //pulse
        setEnable();
        _delay_us(50);
        clearEnable();
        _delay_us(50);

        //default position
        clearEnable();
        clearRS();
        PORTA = 0x00;
}

void clearLCD(void){
        sendByteByNibble(CLEAR_DISPLAY);  // clear display
        _delay_ms(2);         //>1.67ms
}

void cursorHome(void){
        sendByteByNibble(RETURN_HOME);    // return home
        _delay_ms(2);         //>1.67ms
}

void setRS(void){
        PORTC |= 0b01000000;
}

void clearRS(void){
        PORTC &= 0b10111111;
}

void setEnable(void){
        PORTC |= 0b10000000;
}

void clearEnable(void){
        PORTC &= 0b01111111;
}
```

Display.h

```
/*
* display.h
* Created: 03/03/2023 12:40:13
*  Author: William VR
*/
#ifndef DISPLAY_H_
#define DISPLAY_H_
//port A DATA pins
//port C
//PA8 pin - 4 PC6           RS
//PA9 pin  -6 PC7 ENABLE
/*
1. Display clear
2. Function set:
DL = 1; 8-bit interface data
N = 0; 1-line display
F = 0; 5x8 dot character font

3. Display on/off control:
D = 0; Display off
C = 0; Cursor off
B = 0; Blinking off

4. Entry mode set:
I/D = 1; Increment by 1
S = 0; No shift
*/

#define FUNCTION_SET          0b00110000
#define FUNCTION_SET_4BIT     0b00100000
#define CURSOR_ON_BLINK       0b00001100
#define DISPLAY_ON            0b00000001
#define ENTRY_MODE            0b00000110
#define RETURN_HOME           0b00000010
#define CLEAR_DISPLAY         0b00000001
#define SET_DDRAM_ADDRESS0b10000000

void initDisplay(void);
void updateDisplay(uint8_t, uint8_t);
void writeToDisplay(char[], uint8_t, uint8_t); // data; length, DDRAM address
void loadingScreen(void);
void clearLCD(void);
void cursorHome(void);
void sendByteByNibble(char);
void setRS(void);
void clearRS(void);
void setEnable(void);
void clearEnable(void);
#endif /* DISPLAY_H_ */
```

EEPROM.c

```
/*
 * EEPROM.c
 *
 * Created: 17/04/2023 11:12:38
 *  Author: William VR
 */

#include "../main.h"
#include "EEPROM.h"
#include <avr/interrupt.h>

void EEPROM_write(uint8_t ucAddress, uint8_t ucData)
{
	cli();
	/* Wait for completion of previous write */
	while(EECR & (1<<EEPE));
	/* Set Programming mode */
	EECR = (0<<EEPM1)|(0<<EEPM0);
	/* Set up address and data registers */
	EEAR = ucAddress;
	EEDR = ucData;
	/* Write logical one to EEMPE */
	EECR |= (1<<EEMPE);
	/* Start eeprom write by setting EEPE */
	EECR |= (1<<EEPE);
	sei();
}

uint8_t EEPROM_read(uint8_t ucAddress)
{
	cli();
	/* Wait for completion of previous write */
	while(EECR & (1<<EEPE));
	/* Set up address register */
	EEAR = ucAddress;
	/* Start eeprom read by writing EERE */
	EECR |= (1<<EERE);
	/* Return data from data register */
	sei();
	return EEDR;
}
```

EEPROM.h

```
/*
 * EEPROM.h
 * Created: 17/04/2023 11:12:51
 *  Author: William VR
 */
#ifndef EEPROM_H_
#define EEPROM_H_

void EEPROM_write(uint8_t ucAddress, uint8_t ucData);
uint8_t EEPROM_read(uint8_t ucAddress);

#endif /* EEPROM_H_ */
```

I2C.c

```
/*
* I2C.c
*
* Created: 03/03/2023 12:39:19
*  Author: William VR
*/

#include "../main.h"
#include "I2C.h"

void initI2C(){
	sdaHigh();
	clkHigh();
	_delay_us(I2CSpeed);
}

void setTDAValue(uint8_t chipAddress, uint8_t subAddress, uint8_t data){
	startCom();
	sendI2C(chipAddress);
	sendI2C(subAddress);
	sendI2C(data);
	finishCom();
}

void sendI2C(uint8_t input){
	uint8_t shift = 0x00;
	uint8_t mask = 0b00000001;
	uint8_t result = 0x00;

	//send input
	shift = reverse(input);
	for(uint8_t index = 0; index < 8; index++){
		result = shift & mask;
		if(result == 0x01){
```

```
                        sdaHigh();
                }
                else{
                        sdaLow();
                }
                clkPulse();
                shift = (shift >> 1);           //shift right by one
        }
        //ACK
        sdaHigh();
        clkPulse();
}

void startCom(void){
        //start condition
        sdaLow();
        _delay_us(I2CSpeed);
        clkLow();
        _delay_us(I2CSpeed);
}

void finishCom(){
        // finish communication
        clkHigh();
        _delay_us(I2CSpeed);
        sdaHigh();
        _delay_us(I2CSpeed);
}

void clkPulse(void){
        clkHigh();
        _delay_us(I2CSpeed);
        clkLow();
        _delay_us(I2CSpeed);
}

void clkLow(void){
        PORTB &= 0b01111111;
}

void clkHigh(void){
        PORTB |= 0b10000000;
}

void sdaLow(void){
        PORTB &= 0b10111111;
}

void sdaHigh(void){
        PORTB |= 0b01000000;
}
```

I2C.h

```
/*
* I2C.h
* Created: 09/01/2023 15:32:19
*  Author: William VR
*/


#ifndef I2C_H_
#define I2C_H_

#define I2CSpeed 52       //BAUDRATE = 9600

/*function declaration*/
void       initI2C();
void       setTDAValue(uint8_t, uint8_t, uint8_t); //chip address, sub address, data
void       sendI2C(uint8_t);

void    startCom(void);
void       finishCom(void);

void       clkLow(void);
void       clkHigh(void);
void       sdaLow(void);
void       sdaHigh(void);

void       clkPulse(void);

#endif /* I2C_H_ */
```

TDA.c

```
/*
 * TDA.c
 * Created: 03/03/2023 12:39:33
 *  Author: William VR
 */

#include "../main.h"
#include "TDA.h"

/*
input = 0 -> 255

1111 1111 => XXXX 1111 shift 4
1111 1111 => XX11 1111 shift 6

*/

uint8_t steps[15] = {	0b00000000,
			0b00000001,
			0b00000010,
			0b00000011,
			0b00000100,
			0b00000101,
			0b00000110,
			0b00000111,
			0b00001110,
			0b00001101,
			0b00001100,
			0b00001011,
			0b00001010,
			0b00001001,
			0b00001000};


uint8_t convert4bits(uint8_t input) {
	uint8_t shift = ((input >> 4) & 0b00001111);

	return steps[shift] ;
}

uint8_t convert6bits(uint8_t input) {
	uint8_t shift = ((input >> 2) & 0x3F);

	if(shift > 0b00101000){
		shift = 0b00111000;	//mute
	}

	return shift;
}
```

TDA.h

```
/*
 * TDA.h
 *          Created: 15/12/2022 16:21:01
 *  Author: William VR
 */


#ifndef TDA_H_
#define TDA_H_

#define CHIP_ADDRESS 0x88

//Sub-address byte
#define SubAdr_Input_selector               0x00
#define SubAdr_Input_gain                   0x01
#define SubAdr_Volume                       0x02
#define SubAdr_Bass_gain                    0x03
#define SubAdr_Mid_range_gain               0x04
#define SubAdr_Treble_gain                  0x05
#define SubAdr_Speaker_attenuation_L 0x06
#define SubAdr_Speaker_attenuation_R 0x07
#define AUTO_INCREMENT                              0b00001000

uint8_t convert4bits(uint8_t);
uint8_t convert6bits(uint8_t);

#endif /* TDA_H_ */
```

BOM

Comment	Description	Designator	Footprint	Quantity
1725656	Header	Labels	PHOE-1725656-2	8
TSW-103-07-F-S	Header	Labels	SMTC-TSW-103-07-X-S	11
GBJ3510_T0_00601	Bridge Rectifier	BR1	GBJ3510_T0_00601	1
0805	Capacitor, Resistor	C1 => C97, R1 => R53	6-0805_N	140
86001037301O	Capacitor Polarised	C16, C29, C54, C57	CAPPRD250W52D630H1250	4
UFW1V472MHD	Capacitor Polarised	C70, C71, C76, C86, C87	CAPPRD750W80D1825H3750	5
B320A-13-F	DIODE SCHOTTKY 20V 3A SMA	D1, D2	FP-SMA-MFG	2
61301611821	Header	Display Header	61301611821	1
ATS-TI1OP-521-C1-R1	Hardware	H1, H2	ATSTI1OP521C1R1	2
ATTINY828-AU	Integrated Circuit	IC1	QFP80P900X900X120-32N	1
TDA7439DS13TR	Integrated Circuit	IC2	SOIC127P1032X265-28N	1
TPA3116D2QDADRQ1	Integrated Circuit	IC3, IC5	SOP65P810X120-33N	2
TMUX1134PWR	Integrated Circuit	IC4	SOP65P640X120-20N	1
TPS54332DDA	Integrated Circuit	IC6, IC7	SOIC127P600X170-9N	2
61300611121	Header	ICSP	61300611121	1
Inductor	Inductor	L1	2220	1
10uH	Inductor	L2, L3, L4, L5, L6, L7	INDPM110100X400N	6
2.5uF	SMD	L8, L9	710-74437346025	2
ELM70303GD	LED	LED1 => LED10	ELM70303GD	10
61300511121	Header	MUX, Volume	61300511121	2
3386P-1-103LF	Resistor	R9	3386P_1	1
5D2-18LCS	Thermistor	RT1	5D218LCS	1

Figuur 7: Bom lijst

Bibliografie

Github:
https://github.com/WilliamVanRaemdonck/Project_Practise_Enterprise_2_Amplifier

Datasheets:
ATtiny828:
https://ww1.microchip.com/downloads/en/DeviceDoc/doc8371.pdf

TPA3118d2:
https://www.ti.com/lit/ds/symlink/tpa3118d2-q1.pdf?HQS=dis-mous-null-mousermode-dsf-pf-null-wwe&ts=1674903938367&ref_url=https%253A%252F%252Fwww.mouser.be%252F

TDA7439DS:
https://www.mouser.be/datasheet/2/389/cd00010668-1796242.pdf

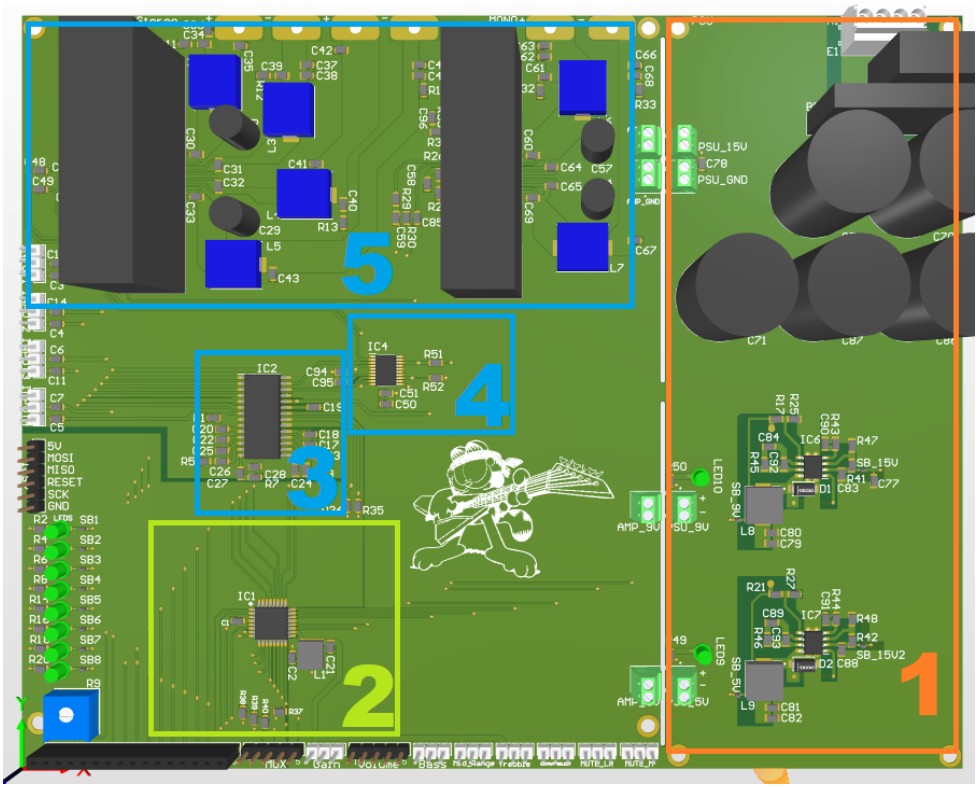
Display:
https://uk.beta-layout.com/download/rk/RK-10290_410.pdf

Spanningsregelaar:
https://www.mouser.be/ProductDetail/595-TPS54332DDA

Code:
Reverse bits in byte:
https://stackoverflow.com/questions/2602823/in-c-c-whats-the-simplest-way-to-reverse-the-order-of-bits-in-a-byte

Chat.gpt:
Algemene code optimalisatie.

Schema & PCB



Figuur 8: pcb-voorkant

Zone 1: Power Supply.

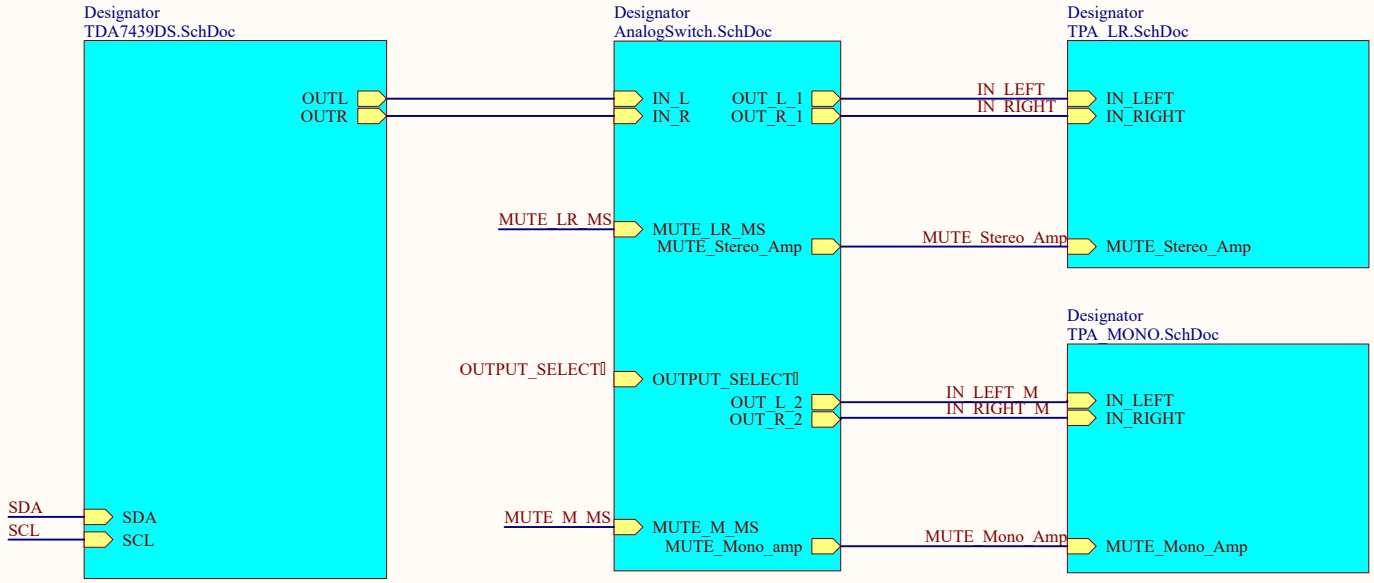
Zone 2: Microcontroller

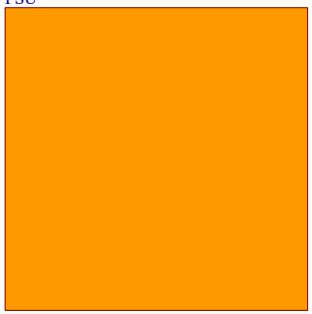
Zone 3: Audiocontrole IC.

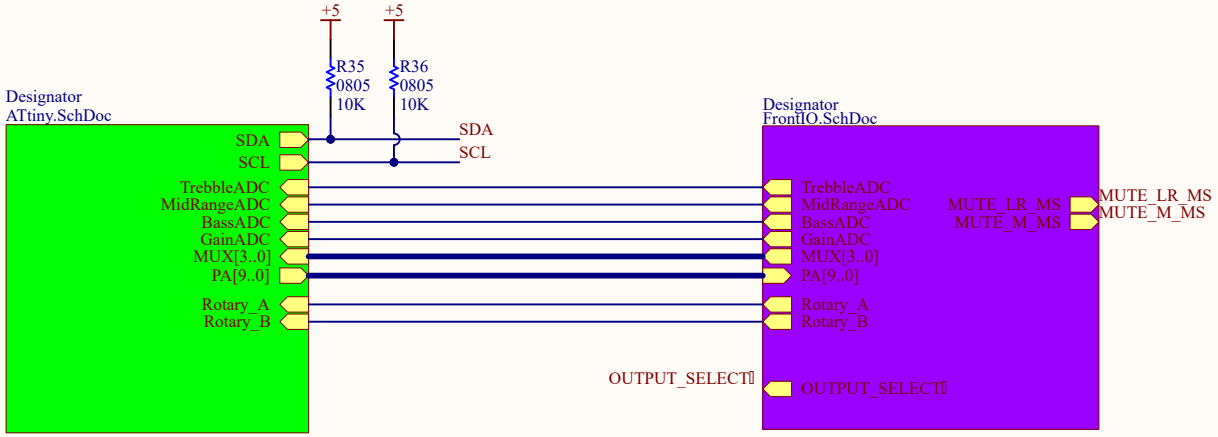
Zone 4: Audio switch

Zone 5: Mono/Stereo versterker.

Class-D Amp
50 W Stereo
100W Mono
Load 4Ohm
26dB Gain

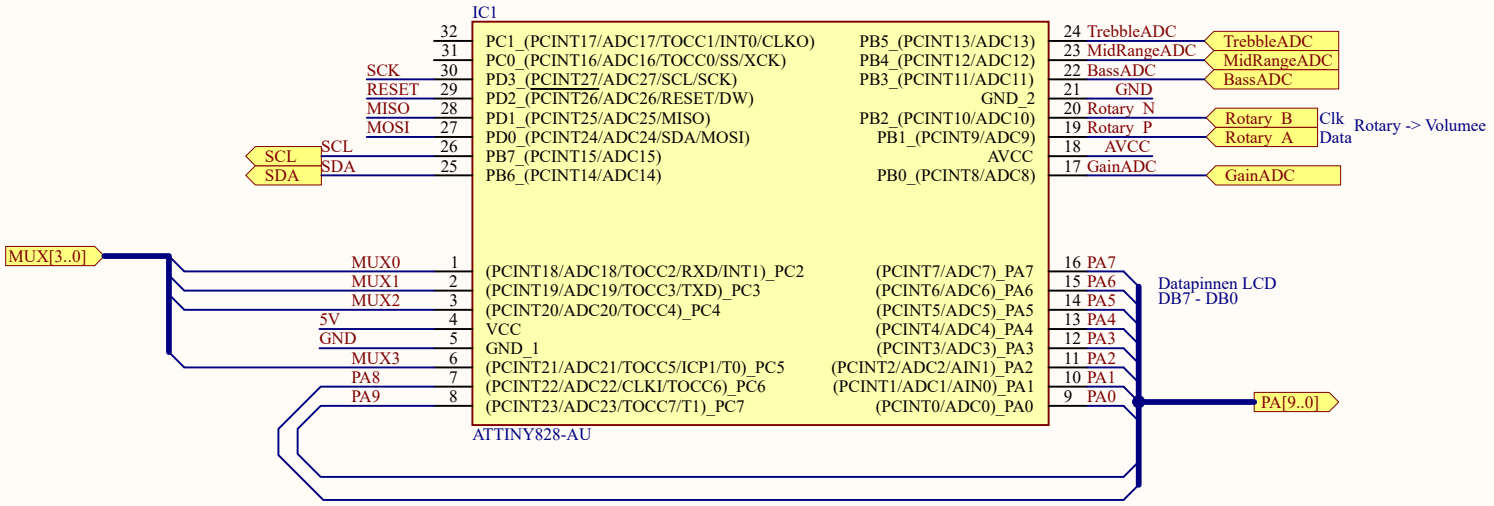




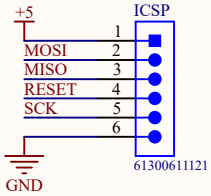


Title		
Size A4	Number	Revision
Date: 6/10/2023		Sheet of
File: G:\My Drive\..\Main.SchDoc		Drawn By:

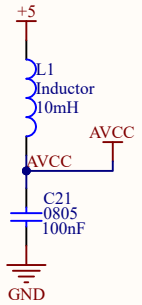
Microcontroller - ATtiny828



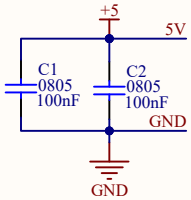
Programmer Port



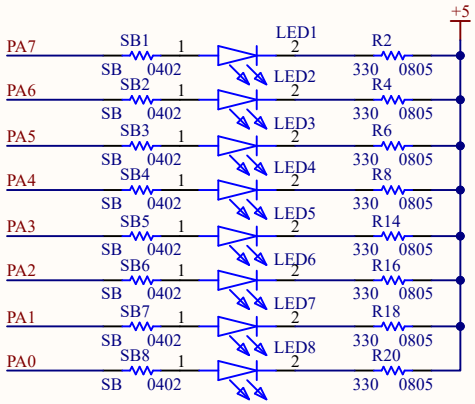
ADC filter



Ontkoppeling

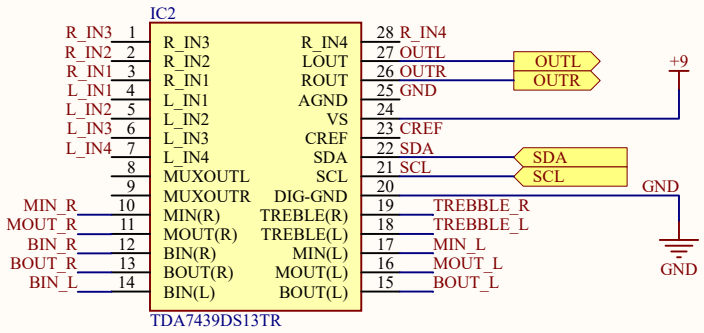


DEBUG LEDS

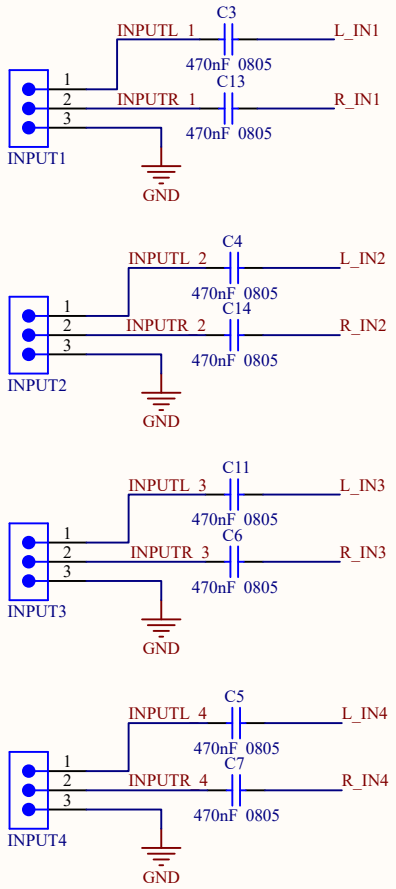


Title		
Size A4	Number	Revision
Date: 6/10/2023		Sheet of
File: G:\My Drive\..\ATtiny.SchDoc		Drawn By:

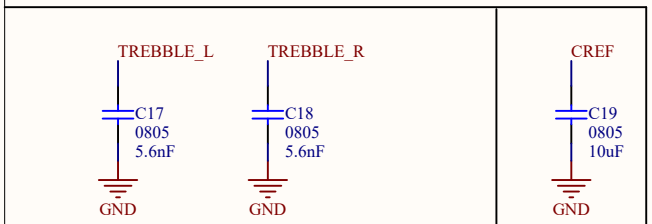
Audio controle

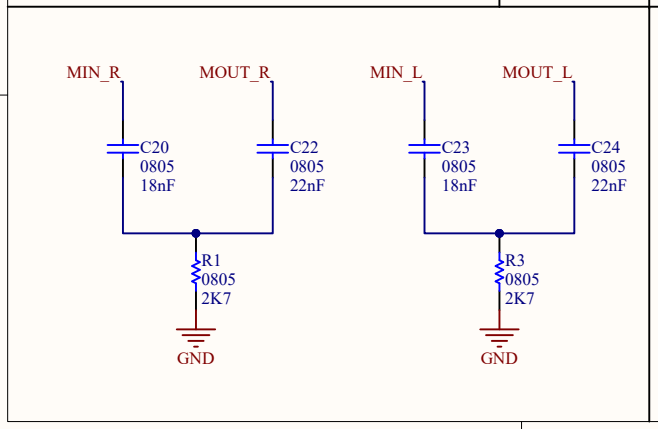


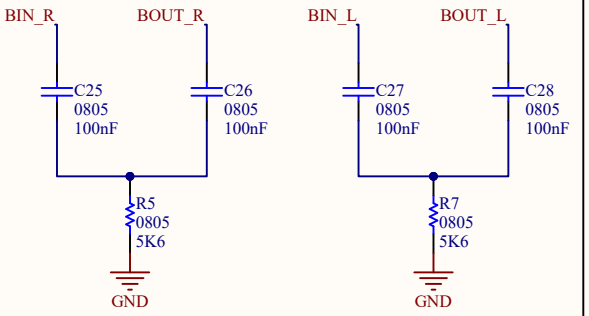
Input



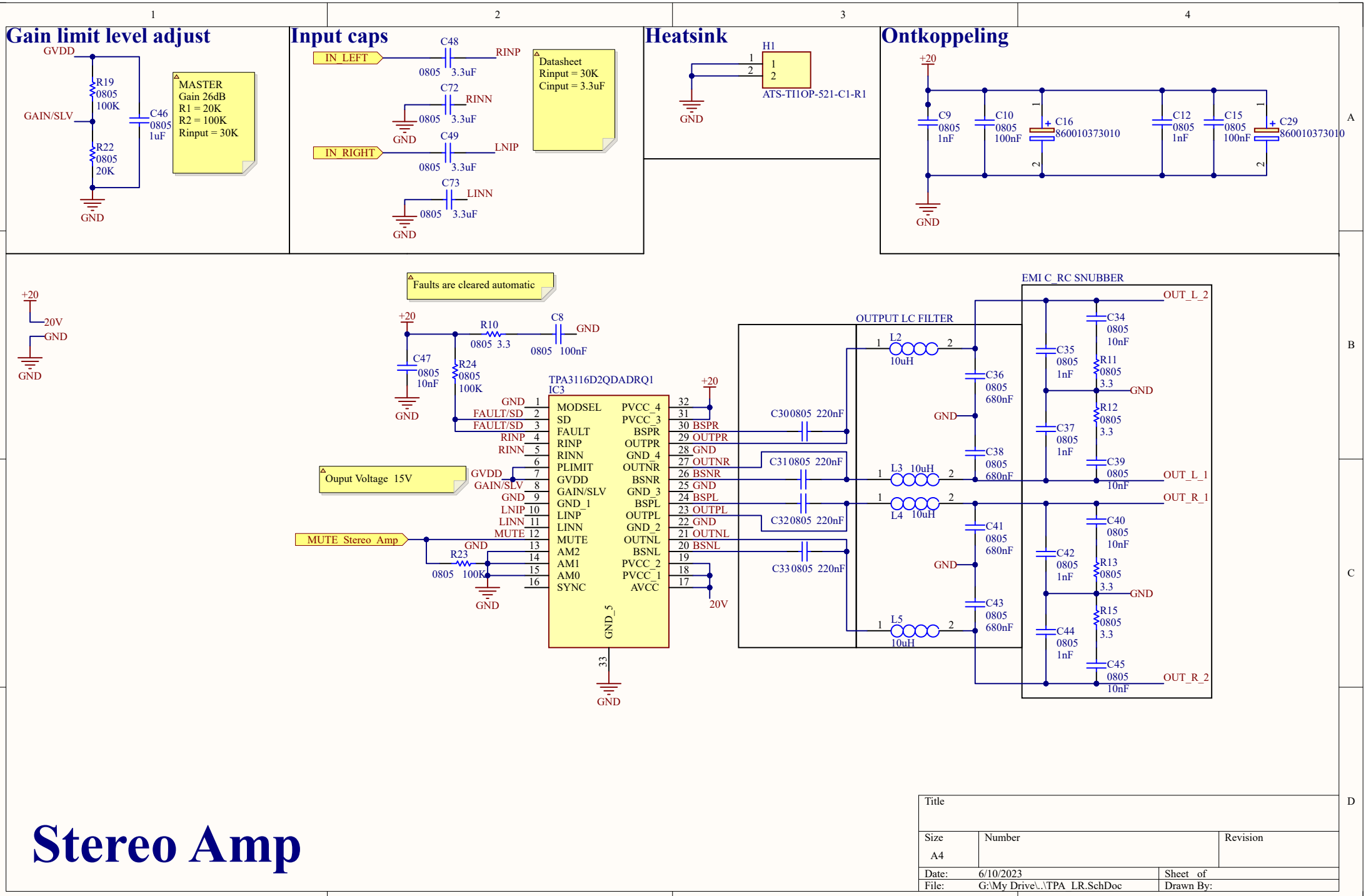
Filters





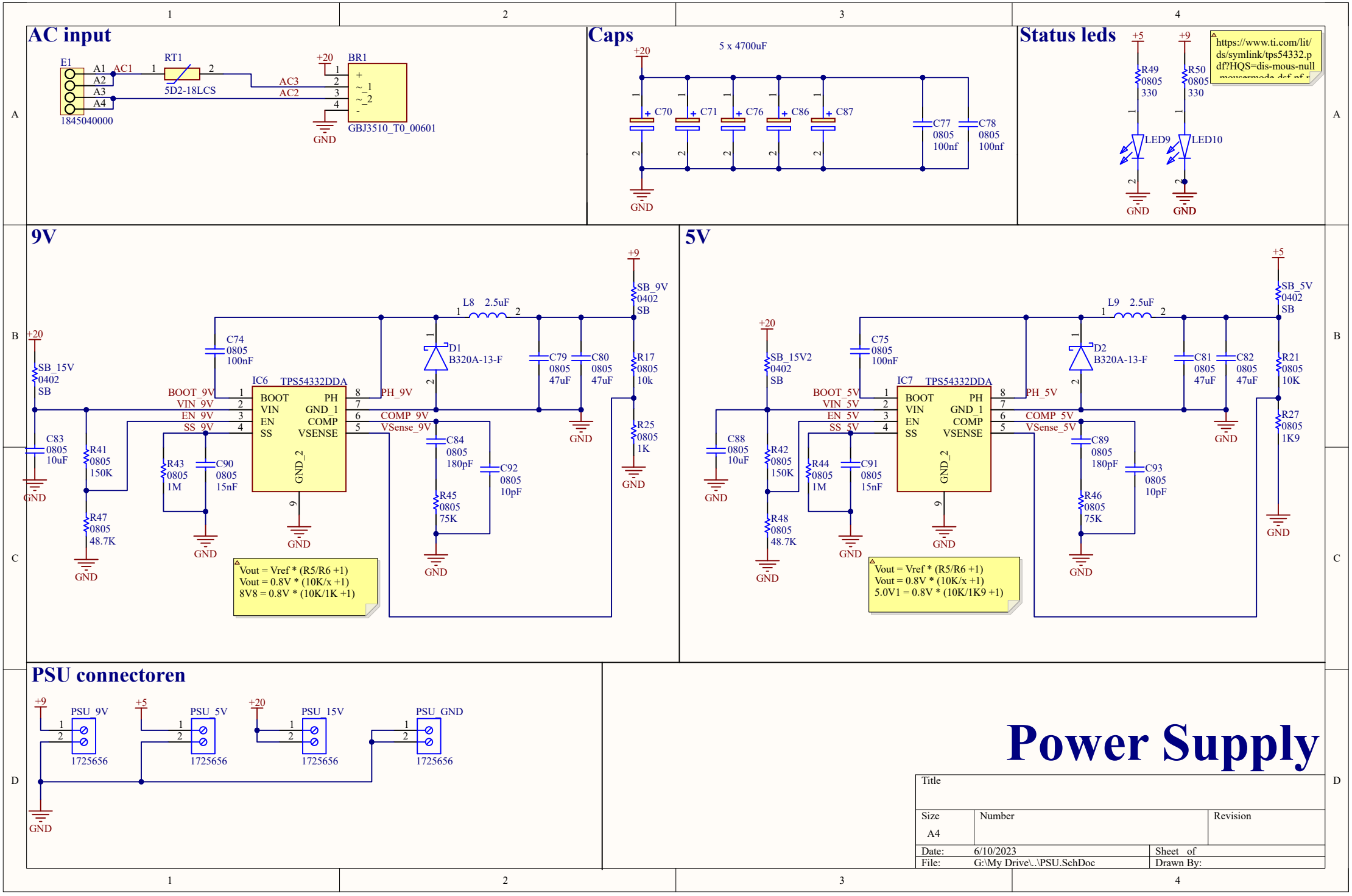


Title		
Size A4	Number	Revision
Date: 6/10/2023		Sheet of
File: G:\My Drive\..\TDA7439DS.SchDoc		Drawn By:



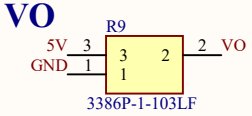
Stereo Amp

Title		
Size A4	Number	Revision
Date: 6/10/2023		Sheet of
File: G:\My Drive\..\TPA_LR.SchDoc		Drawn By:

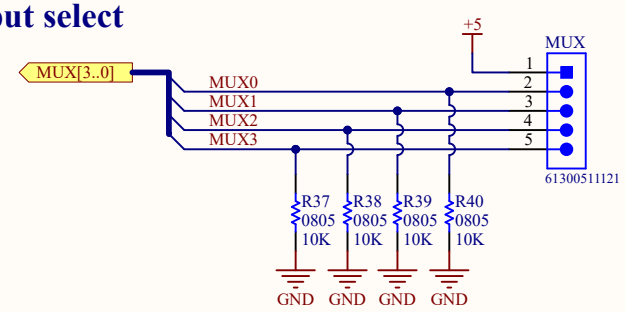

AC input
E1
A1
A2
A3
A4
1845040000
AC1
RT1
5D2-18LCS
AC3
AC2
+20
BR1
GBJ3510_T0_00601
GND
Caps
5 x 4700uF
+20
C70
C71
C76
C86
C87
C77
0805
100nf
C78
0805
100nf
GND
Status leds
+5
+9
R49
0805
330
R50
0805
330
LED9
LED10
GND
GND
https://www.ti.com/lit/ds/symlink/tps54332.pdf?HQS=dis-mous-null
9V
+20
SB_15V
0402
SB
C83
0805
10uF
GND
R41
0805
150K
R47
0805
48.7K
GND
C74
0805
100nF
IC6
TPS54332DDA
BOOT_9V
VIN_9V
EN_9V
SS_9V
BOOT
VIN
EN
SS
PH
GND_1
COMP
VSENSE
GND_2
PH_9V
COMP_9V
VSense_9V
R43
0805
1M
C90
0805
15nF
GND
GND
L8 2.5uF
D1
B320A-13-F
C79
0805
47uF
C80
0805
47uF
GND
+9
SB_9V
0402
SB
R17
0805
10k
R25
0805
1K
GND
C84
0805
180pF
C92
0805
10pF
R45
0805
75K
GND
Vout = Vref * (R5/R6 +1)
Vout = 0.8V * (10K/x +1)
8V8 = 0.8V * (10K/1K +1)
5V
+20
SB_15V2
0402
SB
C88
0805
10uF
GND
R42
0805
150K
R48
0805
48.7K
GND
C75
0805
100nF
IC7
TPS54332DDA
BOOT_5V
VIN_5V
EN_5V
SS_5V
BOOT
VIN
EN
SS
PH
GND_1
COMP
VSENSE
GND_2
PH_5V
COMP_5V
VSense_5V
R44
0805
1M
C91
0805
15nF
GND
GND
L9 2.5uF
D2
B320A-13-F
C81
0805
47uF
C82
0805
47uF
GND
+5
SB_5V
0402
SB
R21
0805
10K
R27
0805
1K9
GND
C89
0805
180pF
C93
0805
10pF
R46
0805
75K
GND
Vout = Vref * (R5/R6 +1)
Vout = 0.8V * (10K/x +1)
5.0V1 = 0.8V * (10K/1K9 +1)
PSU connectoren
+9
PSU_9V
1725656
+5
PSU_5V
1725656
+20
PSU_15V
1725656
PSU_GND
1725656
GND
Power Supply
Title
Size
A4
Number
Revision
Date: 6/10/2023
File: G:\My Drive\..\PSU.SchDoc
Sheet of
Drawn By:

I/O

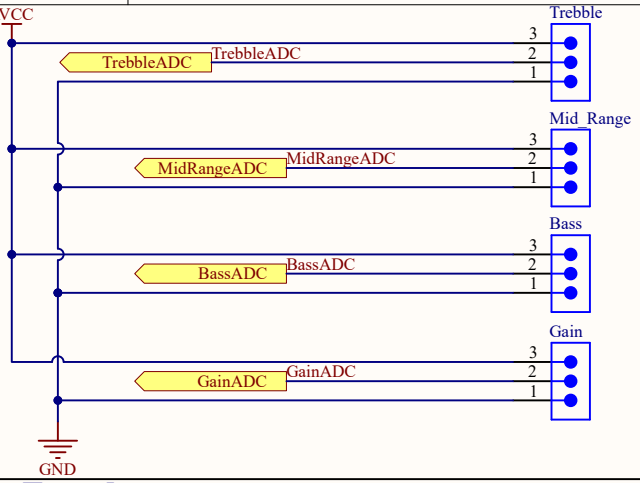
VO



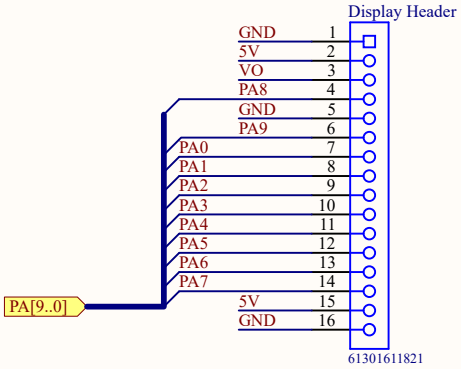
Input select



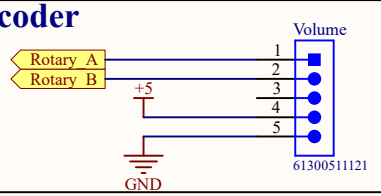
POT

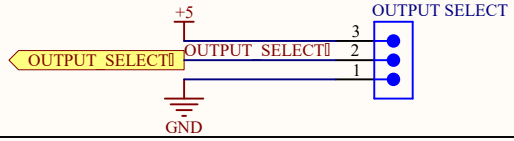


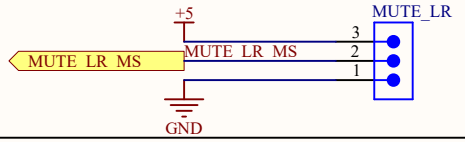
Display

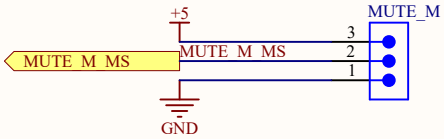


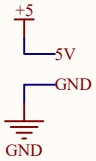
Rotary Encoder



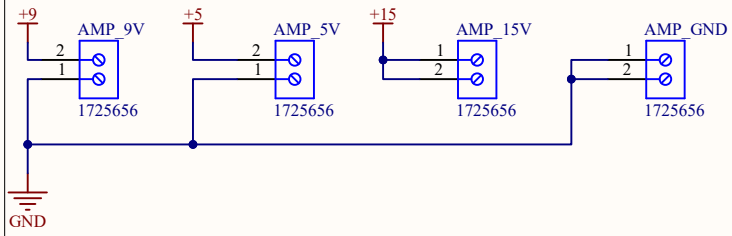




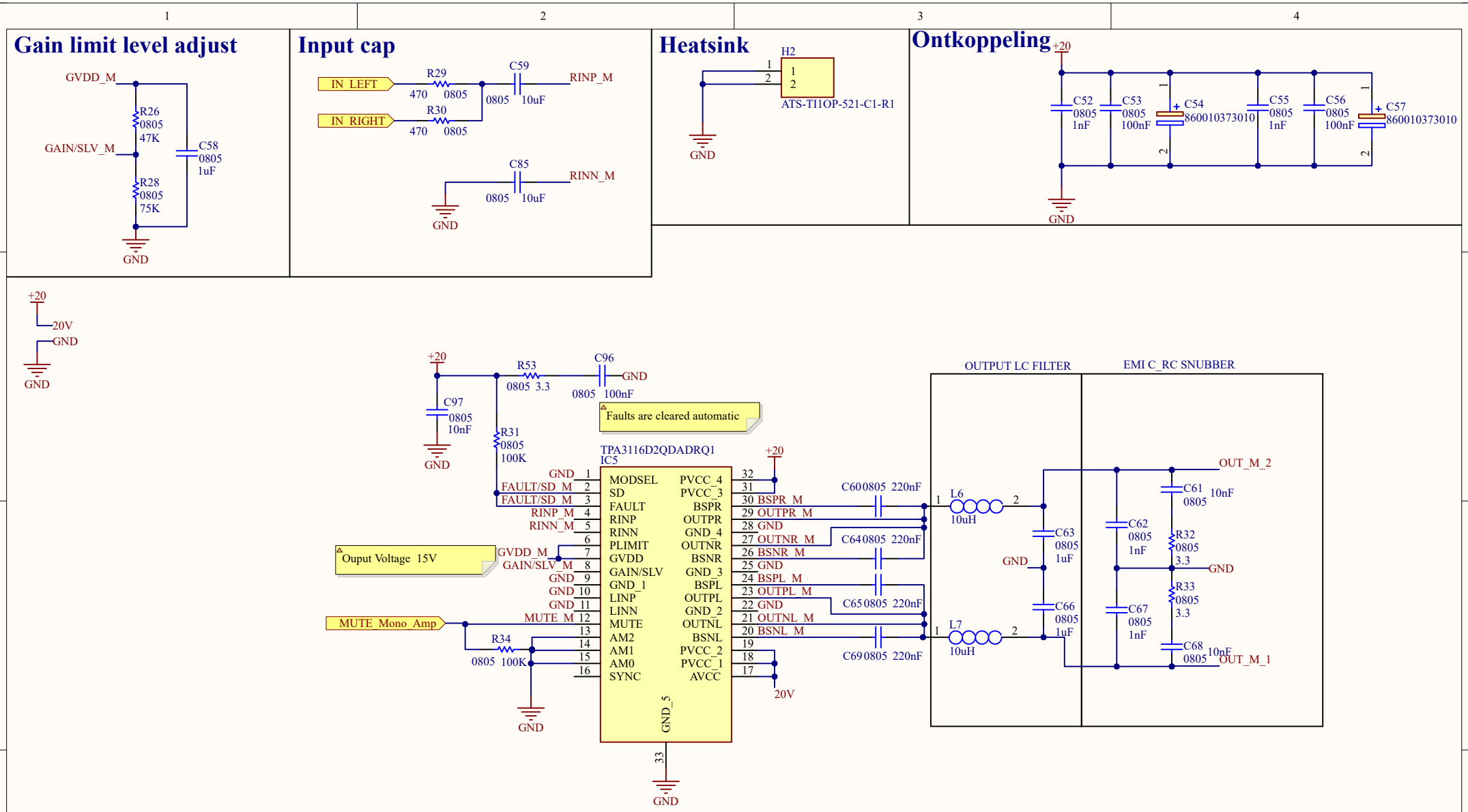




AMP connectoren

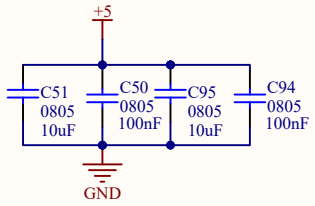


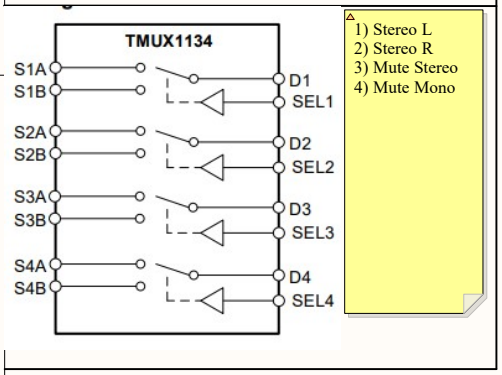
Title		
Size: A4	Number	Revision
Date: 6/10/2023		Sheet of
File: G:\My Drive\..\FrontIO.SchDoc		Drawn By:


Gain limit level adjust
GVDD_M
R26
0805
47K
GAIN/SLV_M
C58
0805
1uF
R28
0805
75K
GND
Input cap
IN_LEFT
R29
470 0805
C59
0805 10uF
RINP_M
IN_RIGHT
R30
470 0805
C85
0805 10uF
RINN_M
GND
Heatsink
H2
1
2
ATS-TI1OP-521-C1-R1
GND
Ontkoppeling
+20
C52
0805
1nF
C53
0805
100nF
C54
860010373010
C55
0805
1nF
C56
0805
100nF
C57
860010373010
GND
+20
20V
GND
GND
+20
R53
0805 3.3
C96
0805 100nF
GND
C97
0805
10nF
GND
Faults are cleared automatic
R31
0805
100K
TPA3116D2QDADRQ1
IC5
GND_1
FAULT/SD_M_2
FAULT/SD_M_3
RINP_M_4
RINN_M_5
6
GVDD_M_7
GAIN/SLV_M_8
GND_9
GND_10
GND_11
MUTE_M_12
13
14
15
16
MODSEL
SD
FAULT
RINP
RINN
PLIMIT
GVDD
GAIN/SLV
GND_1
LINP
LINN
MUTE
AM2
AM1
AM0
SYNC
PVCC_4 32
PVCC_3 31
30 BSPR_M
29 OUTPR_M
28 GND
27 OUTNR_M
26 BSNR_M
25 GND
24 BSPL_M
23 OUTPL_M
22 GND
21 OUTNL_M
20 BSNL_M
19
18
17
BSPR
OUTPR
GND_4
OUTNR
BSNR
GND_3
BSPL
OUTPL
GND_2
OUTNL
BSNL
PVCC_2
PVCC_1
AVCC
GND_5
33
+20
20V
GND
Ouput Voltage 15V
MUTE_Mono_Amp
R34
0805 100K
GND
C60 0805 220nF
C64 0805 220nF
C65 0805 220nF
C69 0805 220nF
OUTPUT LC FILTER
EMI C_RC SNUBBER
L6
10uH
L7
10uH
C63
0805
1uF
C66
0805
1uF
GND
C62
0805
1nF
C67
0805
1nF
C61
0805
10nF
R32
0805
3.3
GND
R33
0805
3.3
C68
0805
10nF
OUT_M_2
OUT_M_1
Mono Amp
Title
Size
A4
Number
Revision
Date: 6/10/2023
File: G:\My Drive\..\TPA_MONO.SchDoc
Sheet of
Drawn By:

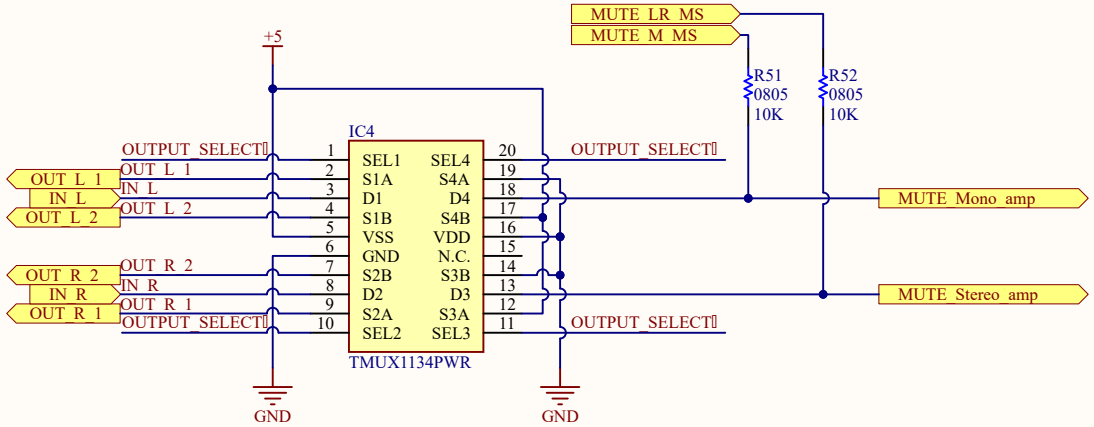
Audio Switch

ONTKOPPELING

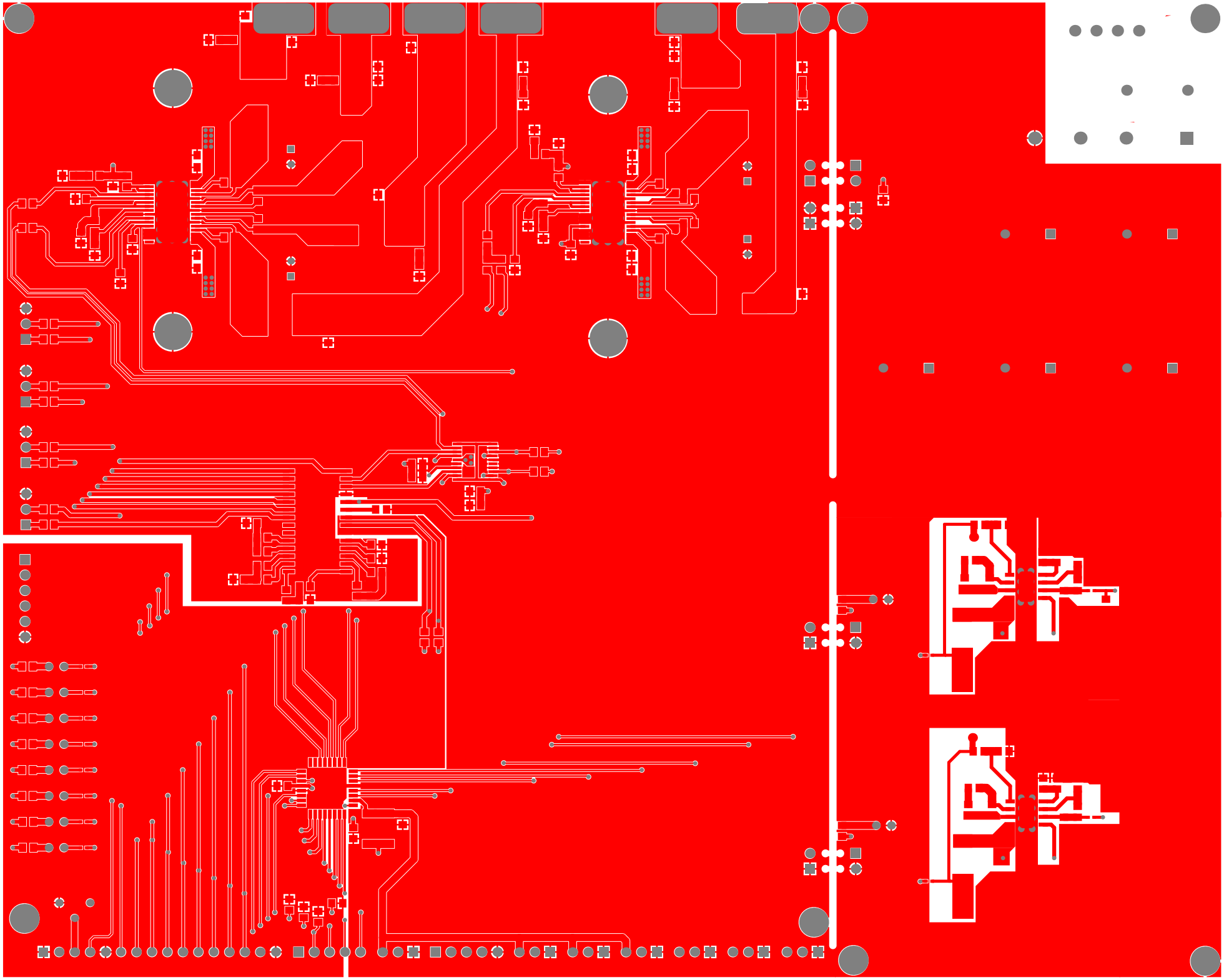


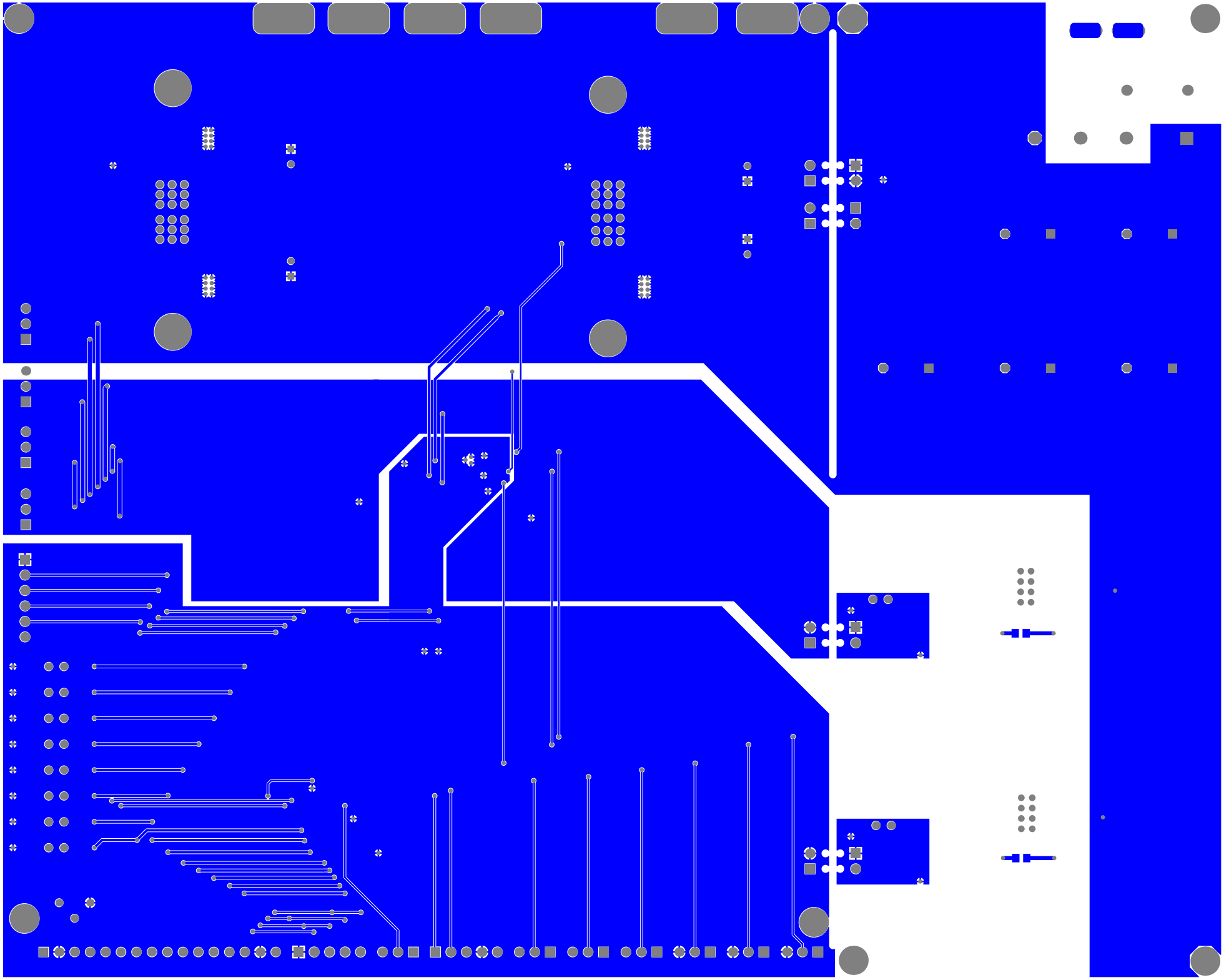


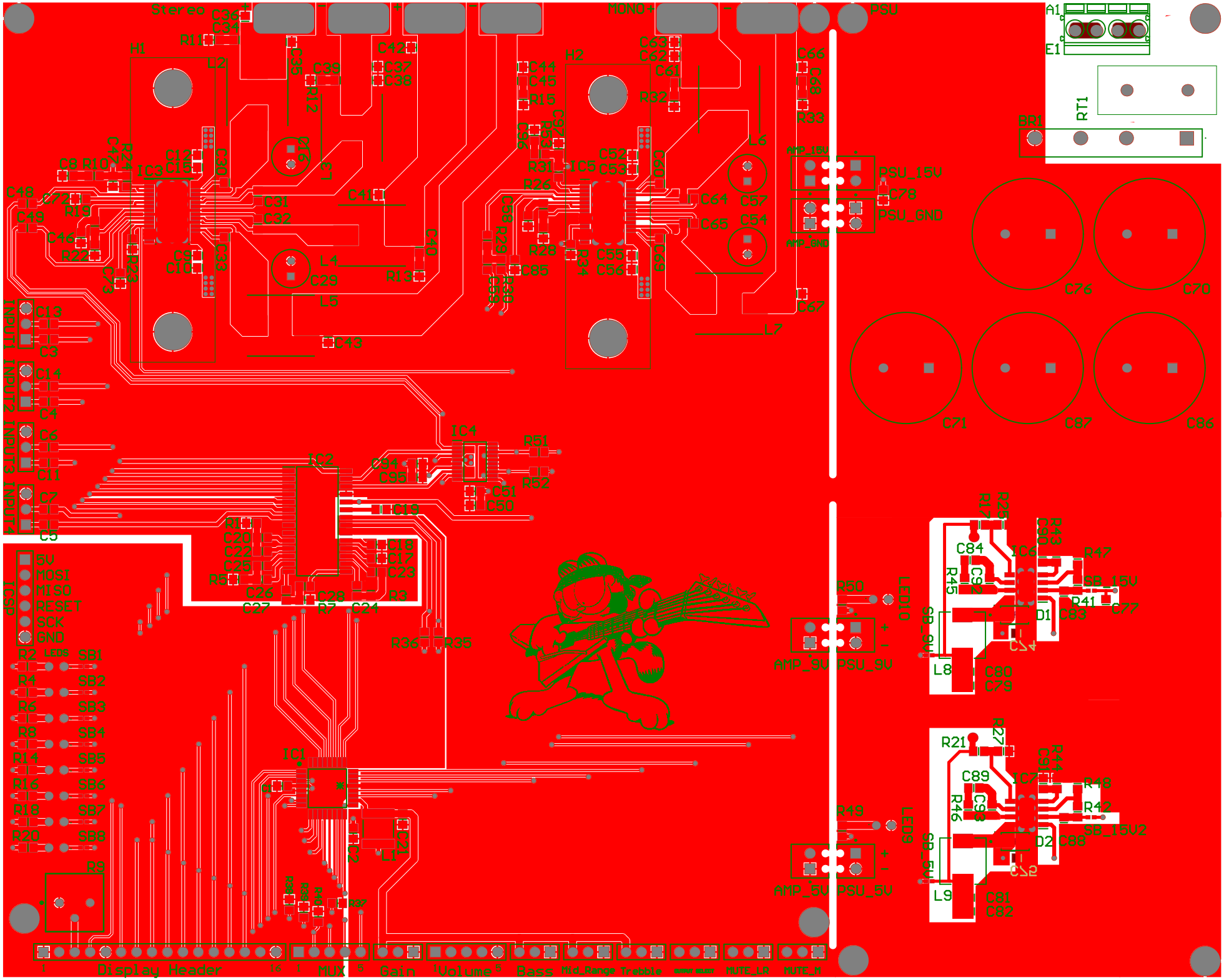
OUTPUT_SELECT OUTPUT_SELECT

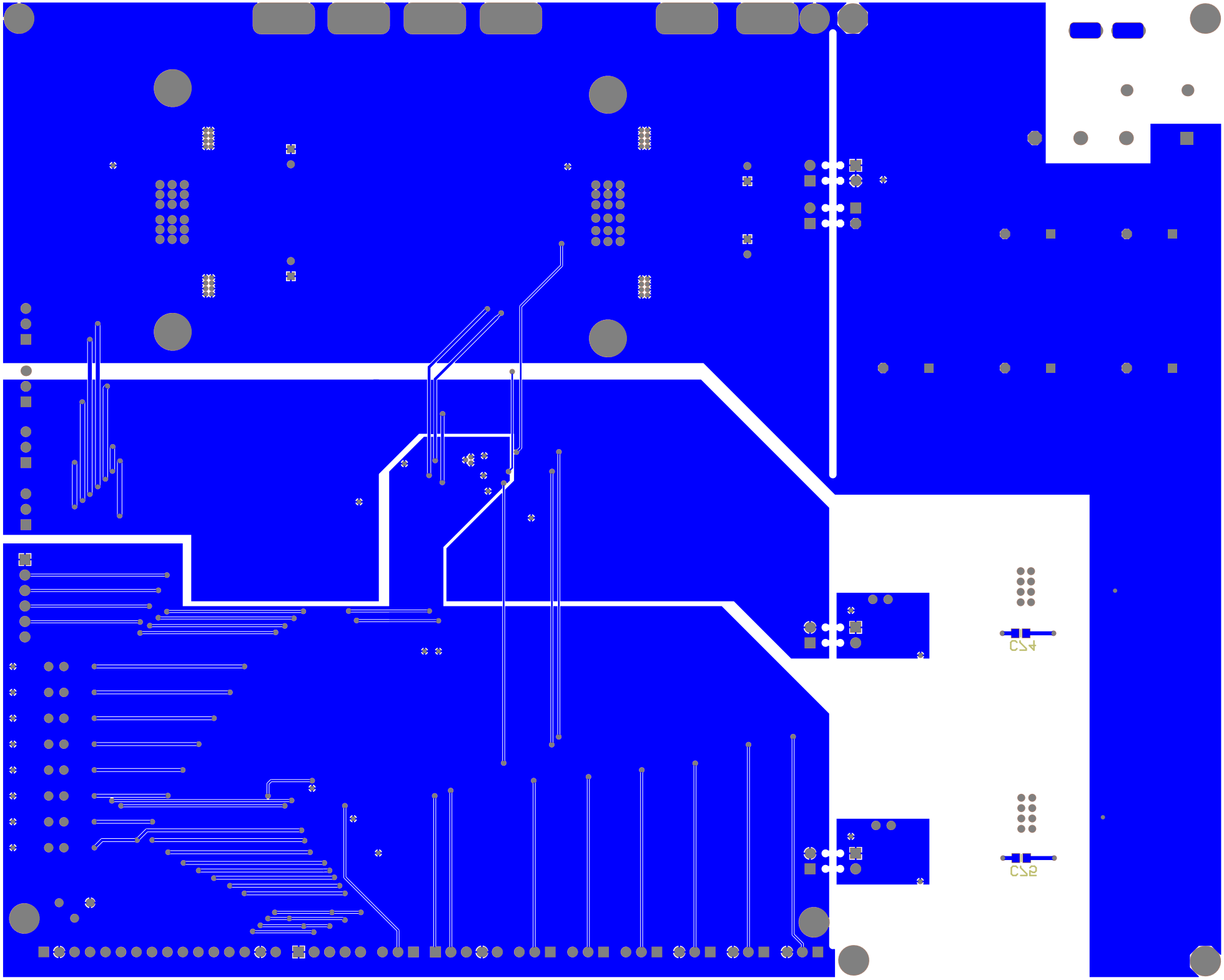


Title		
Size: A4	Number	Revision
Date: 6/10/2023		Sheet of
File: G:\My Drive\..\AnalogSwitch.SchDoc		Drawn By:






Stereo
MONO
PSU
H1
H2
IC1
IC2
IC3
IC4
IC5
IC6
IC7
L1
L2
L3
L4
L5
L6
L7
L8
L9
INPUT1
INPUT2
INPUT3
INPUT4
ICSP
5V
MOSI
MISO
RESET
SCK
GND
LEDS
Display Header
MUX
Gain
Volume
Bass
Mid_Range
Treble
MUTE_LR
MUTE_M
PSU_15V
PSU_GND
AMP_15V
AMP_GND
AMP_9V
PSU_9V
AMP_5V
PSU_5V
SB_15V
SB_15V2
SB_9V
SB_5V
LED0
LED9


C24
C25