UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Peter Dolenc

Vizualizacija zvoka

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Patricio Bulić

Ljubljana 2013

Rezultati diplomskega dela so intelektualna lastnina avtorja in Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00104/2013
Datum: 12.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **PETER DOLENC**

Naslov: **VIZUALIZACIJA ZVOKA**
SOUND VISUALISATION

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Izdelajte nizkotonski zvočnik z vgrajeno vizualizacijo zvoka s svetlečimi diodami LED. Frekvenčno analizo zvoka naj opravi vgrajen sistem na osnovi mikrokrmilnika STM32F4 z jedrom ARM Cortex M4. Na osnovi frekvenčne analize naj mikrokrmilnik ustvari svetlobne efekte ter krmili polje svetlečih diod. Z uporabo svetlečih diod realizirajte tri vrste svetlobnih efektov: večbarvno osvetlitev prostora nizkotonskega zvočnika, prikazovanje vzorcev ali drsečega besedila v matriki LED ter efekt bliskavice.

Mentor:
izr. prof. dr.Patricio Bulić

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Peter Dolenc, z vpisno številko 63090040, sem avtor diplomskega dela z naslovom:

Vizualizacija zvoka

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 16. septembra 2013 Podpis avtorja:

Zahvaljujem se mentorju izr. prof. dr. Patriciu Buliću za pomoč in potrpežljivost pri pisanju diplomskega dela.

Posebej se zahvaljujem prijatelju Juretu Zdovcu za izčrpno pomoč pri načrtovanju in izdelavi vezja.

Zahvaljujem pa se tudi staršem ter punci Maruši, da so mi stali ob strani v obdobju študija.

Kazalo

Seznam uporabljenih kratic

LED	Light Emitting Diode
ADC	Analog-to-digital Converter
THD	Total Harmonic Distortion
MDF	Medium-density Fiberboard
RGB	Red Green Blue
FPU	Floating Point Unit
RAM	Random Access Memory
USB	Universal Serial Bus
MOSFET	Metal–Oxide–semiconductor Field-effect Transistor
UTP	Unshielded Twisted Pair
DMA	Direct Memory Access
EXTI	External Interrupt
TIM	Timer
FFT	Fast Fourier Transformation
DFT	Discrete Fourier Transformation
GIF	Graphics Interchange Format

Povzetek

V diplomskem delu je predstavljena izdelava ohišja nizkotonskega zvočnika, katerega funkcionalnost je razširjena s sistemom za svetlobne efekte in vizualizacije, ki se ujemajo s predvajano glasbo. Za ta namen je bilo na ohišje nizkotonskega zvočnika postavljenih več različnih sklopov svetilnih elementov sestavljenih iz diod LED (*Light Emitting Diode*). Za krmiljenje svetilnih elementov je bil napisan namenski program, ki teče na razvojni ploščici STM32F4-Discovery z jedrom ARM Cortex-M4, uporabljeno pa je bilo tudi dodatno izdelano vezje. Naloga programa je vzorčenje zvoka in analiza prisotnih frekvenčnih spektrov ter generiranje svetlobnih efektov, ki ustrezajo obdelanim podatkom. Pri načrtovanju programa je bila arhitektura načrtna tako, da omogoča preprosto dodajanje novih animacij.

Ključne besede

STM32F4-Discovery, nizkotonski zvočnik, svetlobni efekti, mikrokrmilnik, vgrajen sistem, Diskretna Fouriereva Transformacija, vizualizacije.

Abstract

This thesis contains a description of a construction of subwoofer case that has an extra functionality of being able to produce special visual effects and display visualizations that match the currently playing sound. For this reason, multiple lighting elements made out of LED (Light Emitting Diode) diodes were installed onto the subwoofer case. The lighting elements are controlled by dedicated software that was also developed. The software runs on STM32F4-Discovery evaluation board inside a microcontroller that has an ARM Cortex-M4 core. For the purposes of powering the lighting system, an external circuit board was also designed. The software on the microcontroller is programmed to sample the audio signal and perform the spectral analysis and then use the results of this analysis to generate visual effects and visualizations. Special care was taken while designing the software architecture so that new visualizations are easy to develop.

Key Words

STM32F4-Discovery, subwoofer, visual effects, microcontroller, embedded system, Discrete Fourier Transformation, visualizations

Poglavje 1

1 Uvod

V diplomskem delu, ki ga berete, je predstavljena izdelava avtomobilskega nizkotonskega zvočnika, katerega funkcionalnost je razširjena s sistemom za svetlobne efekte in vizualizacije, ki se ujemajo s predvajano glasbo.

Za ta namen so bili na prednjo stran nizkotonskega zvočnika postavljeni različni sklopi svetilnih elementov, sestavljenih iz različnih diod LED (*Light Emitting Diode*). Glede na vrsto svetlobnih efektov, ki jih posamezni sklop proizvaja, lahko razlikujemo tri načine osvetljevanja: močno osvetljevanje celotne prednje strani v spreminjajočih barvah, osvetljevanje z matriko diod LED na prednji strani nizkotonskega zvočnika z 240 diodami LED ter osvetljevanje z zelo močno LED bliskavico v zračniku nizkotonskega zvočnika. Za potrebe krmiljenja diod LED je bilo izdelano tudi dodatno vezje, na katerem najdemo tranzistorska stikala za krmiljenje posameznih zmogljivejših diod LED ter polje pomikalnih registrov, ki skrbijo za prikaz pravilne *slike* na matriki diod LED na sprednjem delu nizkotonskega zvočnika.

Za obdelavo podatkov o trenutno predvajani glasbi in ustvarjanju svetlobnih efektov ter vizualizacij je poskrbljeno v notranjosti nizkotonskega zvočnika. Tja je vgrajena razvojna ploščica STM32F4-Discovery s procesorjem ARM Cortex M4 [1]. Na njej teče namensko razvit program, ki ga bomo podrobno opisali v drugi polovici moje diplomske naloge.

Program na razvojni ploščici STM32F4-Discovery ima tako več nalog. Njegova prva naloga je zajemanje zvoka, ki do razvojne ploščice pride v analogni obliki. Naloga razvojne ploščice je vzorčenje in digitalizacija, kar pomeni, da odčitava vrednost signala in ga pretvarja v digitalno obliko, pri čemer sodeluje vgrajen analogno-digitalni pretvornik (*Analog-to-digital Converter*). Program zbere 1024 vzorcev, ki jih po plastoviti arhitekturi programa pošlje v obdelavo. To zaporedje vzorcev nato obdela s pomočjo algoritma za izvajanje hitre Fouriereve

transformacije. Rezultat obdelave je spektralna sestava zvoka v danem trenutku. Program te podatke uporabi pri svoji drugi nalogi, ki se izvaja v drugem neodvisnem modulu programa – generiranju svetlobnih efektov in animacij.

Program je bil zasnovan v večplastni arhitekturi, ki naloge programa loči na štiri plasti: abstrakcijo strojne opreme, življenjske funkcije programa, vzorčenje in analizo zvoka in aplikacijsko plast. Vse plasti skupaj tvorijo ogrodje, ki ponuja orodja in strukturo za preprosto pisanje *vizualizacijskih aplikacij*. Vizualizacijska aplikacija je v resnici razred, v katerem je na visokem nivoju definiran način, kako se program odzove na dane podatke o glasbi, oziroma kakšna bo vizualizacija zvoka. Struktura programa omogoča pisanje in dodajanje več med sabo povsem različnih načinov vizualiziranja glasbe na preprost način v zelo kratkem času. Ogrodje pa s pomočjo vgrajenih gumbov na nizkotonskem zvočniku celo dovoljuje izbiranje *vizualizacijske aplikacije* kar med delovanjem.

Poglavje 2

2 Izdelava nizkotonskega zvočnika

Ideja o izdelavi kompleksnega vizualizatorja zvoka je prišla ob nakupu nove avtomobilske avdio opreme. Zato bomo v tem poglavju na kratko predstavili uporabljeno avdio opremo in namenili nekaj besed tudi samemu ohišju nizkotonskega zvočnika. V nadaljevanju poglavja pa bomo pojasnili zamisli o tipih osvetljevanja nizkotonskega zvočnika, ki bi se jih dalo z ustreznim kontroliranjem spremeniti v vizualizacije.

2.1 Uporabljene avdio komponente

V nizkotonskem zvočniku sta uporabljeni naslednji avdio komponenti:

nizkotonski zvočnik *RE Audio SX-10 D2* [2]**:**

- deklarirana moč pod stalno obremenitvijo: 1000W (*Thermal power handling*)
- dve navitji z impedanco 4 Ohm, ki omogočata vzporedno vezavo za priklop z skupno impedanco 2 Ohm;

in ojačevalec *Ground Zero GZTA 1.1200DX* [3], ki je enokanalni ojačevalec, ki premore ojačati sinusni signal do skupne moči 1750W[1] ob priklopu bremena z impedanco 2 Ohm.

2.2 Ohišje nizkotonskega zvočnika

Nizkotonski zvočnik brez ohišja zveni zelo slabo, poleg tega pa se hitro pokvari zaradi nekontroliranega gibanja. Ohišje mu namreč zagotavlja resonanco (ojačitev zvoka) v predvidenem območju delovanja, ter kontrolo (zrak, ki ga izpodriva skozi zračnik[2] ohišja, deluje kot zračna vzmet). Zato je zelo pomembno, da so

[1] Specifikacija moči 1750W velja ob doseganju maksimalno 10% skupnega harmoničnega popačenja (Total Harmonic Distortion) [30].
[2] Zračnik ohišja bi lahko poimenovali tudi *bass reflex* cev. Angleški izraz za zračnik pa je subwoofer port ali subwoofer vent.

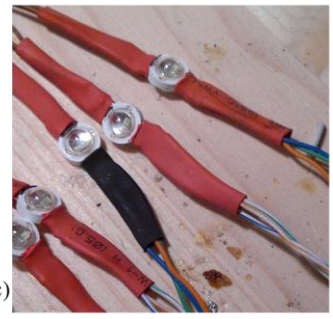
volumen ohišja, dolžina in prostornina zračnika pravilno dimenzionirani in v pravih razmerjih med seboj.

Ohišje nizkotonskega zvočnika je bilo v osnovi narejeno iz plošče MDF (*Medium-density fiberboard*) [4] v debelini 19 mm. Material odlikuje visoka trdnost in togost, ki sta potrebni lastnosti za tovrstno ohišje, saj ne želimo, da bi ohišje proizvajalo kakšne dodatne vibracijske zvoke. Ohišje je bilo na koncu še prekrito s tapisonom, ki se je najbolje ujemal s tistim, ki je prvotno uporabljen za tapeciranje avtomobilskega prtljažnega prostora.

2.3 Izbira izgleda in tipa svetlobnih efektov

Avtomobil ima za lastne potrebe lasten vir napajanja. Vsi dodatni porabniki električne energije v avtomobilu si tako delijo kapaciteto akumulatorja oziroma generatorja, odvisno od tega, če je motor avtomobila zagnan, zato so za svetlobne efekte zelo primerna svetila, ki imajo zelo dobro razmerje med svetilnostjo in porabo električne energije. Zaradi tipa električnega vira, ki zmore zagotavljati 12 V napetost (akumulator) oziroma 14,4 V (alternator) napetost in enosmeren tok, so najboljša izbira za osvetljevanje diode LED (*Light Emitting Diode*) [5].

Za zagotovitev vseh možnih svetlobnih efektov so bili izbrani trije povsem različni načini osvetljevanja. Vsak od teh načinov uporablja tudi povsem drugačne diode LED. Tudi sama oblika ohišja nizkotonskega zvočnika je bila prirejena vsem trem načinom osvetljevanja.

a)  b)  c) 

Slika 1: Trije različni tipi diod LED za tri različne tipe osvetljevanja.
*Na sličici **a)** vidimo diodo LED z visoko močjo (kar 10W).*
*Na sličici **b)** vidimo običajne enobarvne diode LED (v štirih različnih barvah) z odrezanim vrhom glavice za čim večjo razpršitev svetlobe.*
*Na sličici **c)** pa so vidne diode LED z visoko močjo, ki imajo tri barvne komponente za mešanje barv (RGB).*

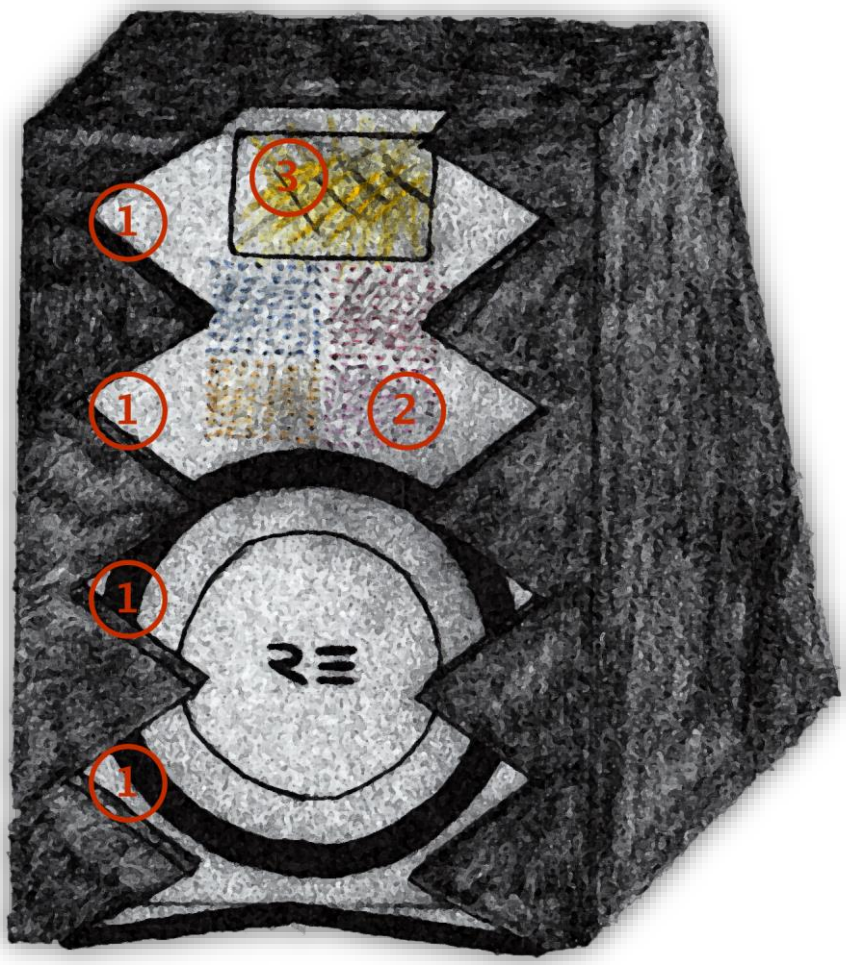
K obliki ohišja je bil dodan nekakšen predprostor globine nekaj centimetrov. Ta predprostor služi kot zaščita zvočniku pred predmeti v prtljažniku in je precej stalna praksa pri izdelovalcih tovrstnih ohišij. V predprostor bo nameščen prvi tip osvetljevanja.

2.3.1 Večbarvno osvetljevanje predprostora

Namen tega prvega načina osvetljevanja je močna osvetlitev celotnega nizkotonskega zvočnika po celotni višini v vseh mogočih barvah. Za ta namen bodo uporabljene diode *RGB LED* z visoko močjo. Pod lečo diode RGB LED se v resnici nahajajo tri diode LED: rdeča, zelena in modra. Od tod oznaka RGB (*Red Green Blue*). Vsaka od treh diod LED je dioda LED z visoko močjo in zmore približno 1W moči. Moč vseh treh diod LED skupaj oziroma ene enote RGB LED je tako deklarirana kot 3W. S kombinacijo rdeče, zelene in modre diode LED in reguliranja svetilnosti vsake izmed njih je možno prikazati praktično vse barve (razen črne, ki je v tem primeru ugasnjena dioda) – podobno, kot to počno zasloni in televizije. Na sliki 2 je s številko 1 označeno mesto, kjer osvetljevanje predprostora od strani postane vidno. Diode RGB LED z visoko močjo so namreč nameščene za rob. Diode RGB LED so vidne tudi na sliki 1.c.

2.3.2 LED matrika 12x20

Drug način osvetljevanja pa je matrika običajnih diod LED. Razporejene so v 10 vrstic po 24 diod LED. Natančneje jih sestavljajo štirje kvadranti v dimenzijah 5x12, pri čemer je vsak kvadrant drugačne barve. Barve vsakega kvadranta so sicer fiksne in enobarvne. LED matrika je zelo primerna za prikazovanje drsečega besedila ter drugih vzorcev. Nahaja se nad zvočnikom in pod zračnikom nizkotonskega zvočnika, kot je razvidno na sliki 2, označeno s številko 2. Uporabljene diode LED so vidne na sliki 1.b.



Slika 2: Oblika in tipi osvetljevanja ohišja nizkotonskega zvočnika.

2.3.3 Močna bliskavica LED

Kot zadnji način osvetljevanja pa je dodana zelo močna bliskavica LED. Ob vklopu odda 1400 lumnov slepeče bele svetlobe, za kar poskrbi 10W dioda LED z visoko močjo. Primerna je za kratke pulze svetlobe in zagotavlja vidnost od daleč. Ker je neposreden pogled v vklopljeno diodo zelo neprijeten je nameščena v zračnik nizkotonskega zvočnika, kakor je razvidno pod številko tri na sliki 2. Izgled same diode pa je razviden na sliki 1.a.

Poglavje 3

3 Priprava elektronskih elementov

V tem poglavju je opisana izdelava, priprava in/ali vgradnja posameznih zaključenih elektronskih enot oziroma naprav, kot so, na primer: razvojna ploščica, celotna matrika diod LED, vezje za krmiljenje te matrike, razvojna procesorska ploščica itd. Na koncu pa bomo vse naprave povezali v en sistem.

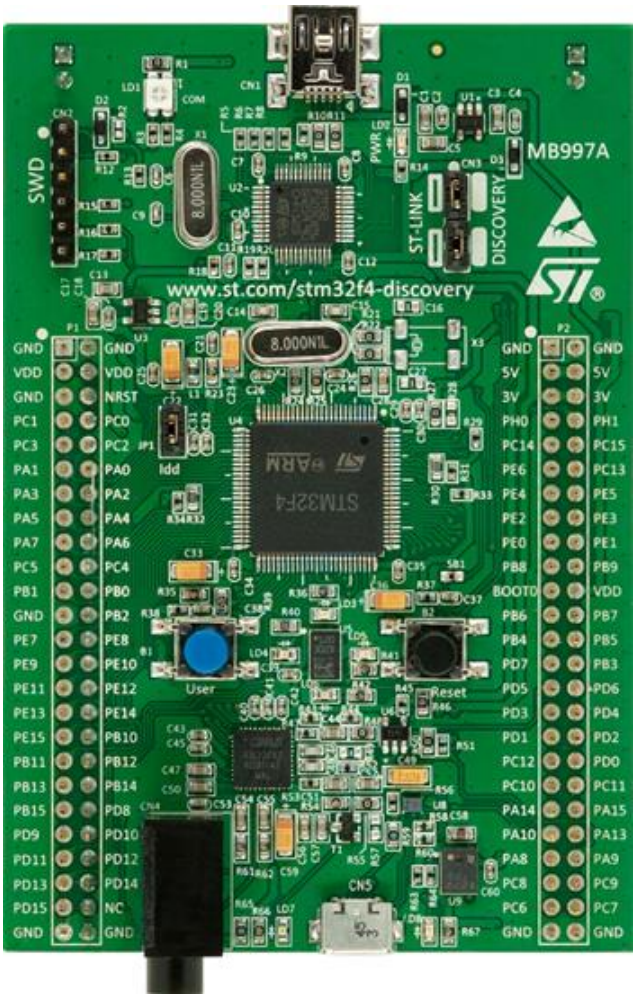
3.1 Razvojna ploščica STM32F4-Discovery

Razvojna ploščica STM32F4-Discovery [1] je zelo priljubljena razvojna ploščica, predvsem zaradi nizke cene za ponujeno zmogljivost in obilico perifernih naprav na sami ploščici.

Na razvojni ploščici STM32F4-Discovery se nahaja mikrokrmilnik STM32F407VGT6 [6] z 32-bitnim jedrom ARM Cortex-M4F, 1 MB trajnega pomnilnika *flash* in 192 KB pomnilnika RAM (*Random Access Memory*). Mikrokrmilnik ima tudi enoto FPU (*Floating Point Unit*) za računanje s plavajočo vejico. Na sami razvojni ploščici je vgrajen tudi programator in razhroščevalnik za mikrokontroler - ST-LINK/V2 [7], ki omogoča preprosto programiranje in razhroščevanje preko vgrajenega mini-USB *(Universal Serial Bus*) vhoda. Če je ploščica priklopljena preko povezave USB, potem dodatno napajanje ni potrebno (v kolikor v sistem nimamo povezanih dodatnih porabnikov). V nasprotnem primeru pa moramo zagotoviti 3V ali 5V vir napajanja. Med drugim pa so na voljo še naslednje naprave: 3-osni merilec pospeškov (*accelerometer*), avdio izhod s sintetizatorjem zvoka, mikrofon, dve tipki, 8 informativnih diod LED ter micro-USB vhod. Na razvojni ploščici najdemo tudi 100 priključnih pinov, ki ustrezajo 100 nožicam mikrokrmilnika. Oblika mikrokrmilmika imata zato posebno ime LQFP100 (*Low profile Quad Flat Package 100 pins*).

Za razvojno ploščico je na uradni strani na voljo kvalitetna dokumentacija ter precej demonstrativnih projektov s primeri uporabe različnih naprav in

mehanizmov. Na voljo je tudi celotna knjižnica funkcij, ki zagotovijo abstrakcijo strojne opreme - STM32F4-Discovery Firmware Library (STSW-STM32068) [8].



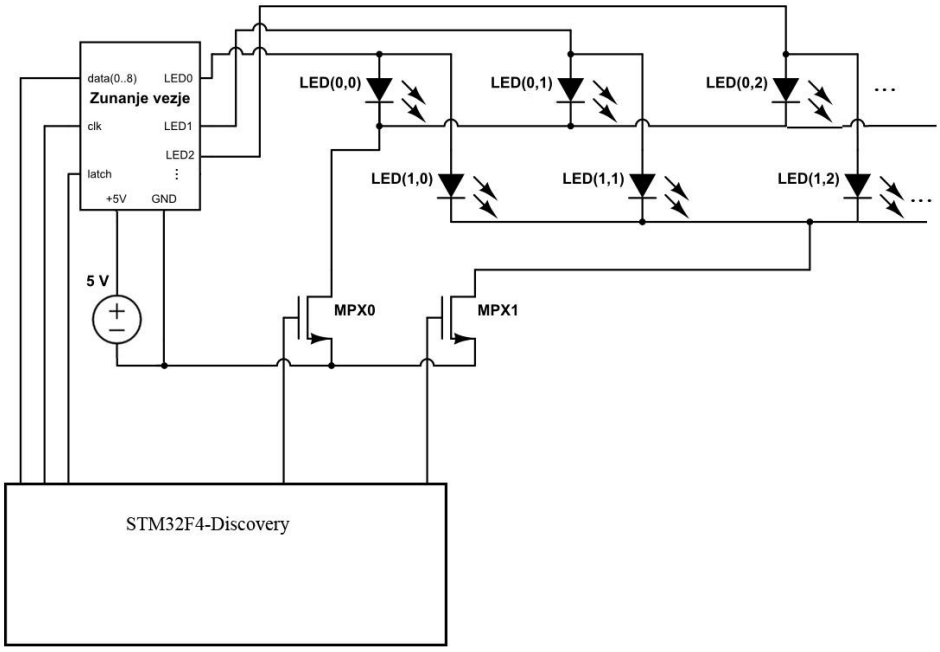
Slika 3: Razvojna ploščica STM32F4-Discovery z mikrokrmilnikom ki ima jedro ARM Cortex M4F [1].

3.2 Vezje za krmiljenje LED matrike

Ob priklopu diod LED z visoko močjo je bil za krmiljenje uporabljen pristop, pri katerem ena nogica na razvojni ploščici krmili eno diodo LED z visoko močjo. Pri matriki diod LED 12x20 (opisani v poglavju 2.3.2) pa je zaradi velikega števila diod LED, to znaša 240, ta pristop neprimeren. Za krmiljenje celotne matrike vseh 240 diod LED je bilo izdelano dodatno zunanje vezje, po večini sestavljeno iz

polja pomikalnih registrov. Vezje se tako krmili serijsko in ne zasede veliko nožic razvojne ploščice STM32F4-Discovery. Število potrebnih pomikalnih registrov pa je bilo tudi zmanjšano za polovico z uporabo tehnike multipleksiranja.

Multipleksiranje po definiciji pomeni zmanjšanje števila prenosnih kanalov za kontroliranje večjega števila signalov [9].



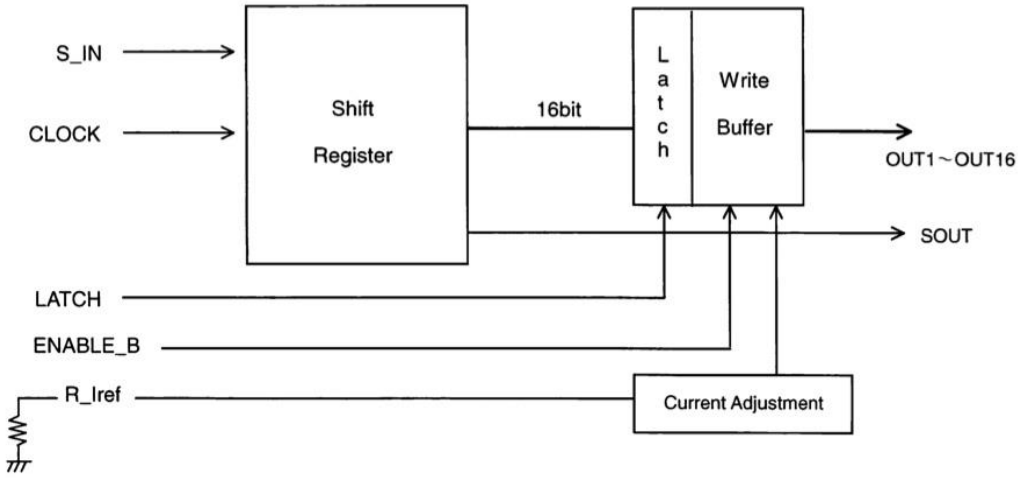
Slika 4: Shematski prikaz multipleksiranja. Z zunanjim vezjem izbiramo stolpec v matriki diod LED, z izmenično vklopljenima tranzistorskima stikaloma pa vrstico. Shema je narisana z brezplačnim orodjem Circuitlab [10].

V našem primeru se število prenosnih kanalov med zunanjim vezjem in matriko diod LED zmanjša za 2. To dosežemo tako, da krmilimo po dve vrstici diod LED naenkrat, oziroma ju krmilimo izmenično. Podatki o tem, katera dioda LED mora biti v danem trenutku vklopljena, prihajajo iz vezja s pomikalnimi registri. Vezje na svoje izhode izmenično nastavlja podatke o prižganih diodah LED v lihi in v sodi vrstici. Izhod iz vezja je zvezan na anode (+ *nogice*) obeh vrstic diod LED. To pomeni, da so anode posameznih parov diod LED v sodi in lihi vrstici povezane skupaj. Tako podatek iz vezja s pomikalnimi registri skuša prižgati določene diode LED tako v sodi kot v lihi vrstici naenkrat. Ker pa želimo

kontrolirati vsako diodo LED v vsaki vrstici posebej, pa s pomočjo katod (- nogic) nadziramo, katera vrstica diod LED ima možnost za sklenjen električni krog. To dosežemo tako, da uporabimo dve tranzistorski stikali: eno za sode vrstice in eno za lihe vrstice, ki ju vedno vklapljamo izmenično, in nikoli obe hkrati. Vklapljanje tranzistorskih stikal je sinhronizirano s podatki, ki jih vezje s pomikalnimi registri da na svoje izhode. Tako se tranzistorsko stikalo za vklop sode vrstice vključi takrat, ko vezje s pomikalnimi registri na svoje izhode pripravi podatke za vklopljene diode LED iz sode vrstice ter seveda obratno za lihe vrstice. Da lahko s tranzistorskim stikalom kontroliramo celotno vrstico diod LED, so vse anode v vrstici povezane skupaj s tranzistorskim stikalom. Mehanizem si lahko predstavljamo tudi kot mehanizem za izbiranje prižgane diode LED, kjer najprej izberemo vrstico, kjer želimo prižgati diodo LED, in nato izberemo še stolpec. Multipleksiranje za 3 diode LED na vrstico je narisano na sliki 4. Izmenično kontroliranje sode in lihe vrstice diod LED v matriki se seveda izvaja tako hitro, da človeško oko ne zazna utripanja in ustvari iluzijo, da so vse diode LED kontrolirane istočasno. Namesto faktorja 2 bi lahko uporabili tudi večji faktor, na primer 4 ali 8, in tako zmanjšali potrebo po tolikih pomikalnih registrih. Vendar bi to storili na račun svetilnosti diod LED, saj bi pri multipleksiranju s faktorjem 8 to pomenilo, da bi bila vsaka dioda LED v resnici prižgana le 1/8 časa.

3.2.1 Uporabljeni elementi

Poleg preprostih analognih elementov, kot so uporniki, kondenzatorji itd., so v vezju uporabljeni predvsem digitalni elementi. Najbolj kompleksen med njimi je 16-bitni serijsko-paralelni pomikalni register BD7851FP-E2 proizvajalca ROHM [11]. BD7851FP-E2 ima serijski vhod in 16 paralelnih izhodov. Deluje tako, da vsako urino periodo na serijskem vhodu, na podatkovni (*data*) nogici, sprejema vrednost enega od izhodov. Vendar pa se stanje izhoda ne spremeni še takoj. Za spremembo stanja je potrebno za nekaj časa dvigniti signal *latch* v visoko stanje, takrat pa se zgodi prepis vseh 16 stanj izhodov. Nato se zopet lahko začne zbiranje vrednosti izhodov za vsako posamezno od 16 nogic. Delovanje lahko preučimo tudi na proizvajalčevi shemi za ta element na sliki 5.



Slika 5: Proizvajalčeva shema delovanja BD7851FP-E2 [11].

V našem vezju upravljamo tudi z urinim signalom. Ta namreč ni fiksen, ampak ga dvigamo in spuščamo le takrat, ko pošiljamo podatek o izhodu po podatkovni nogici. Tako celoten postopek izgleda takole:

1. Signal ure v nizko stanje.
2. Za vsako nogico:
    a. Željeni izhod na vhodno podatkovno nogico.
    b. Signal ure v visoko stanje.
    c. Signal ure v nizko stanje.
3. Dvigni signal *latch* v visoko stanje.
4. Spusti signal *latch* v nizko stanje.

Naslednja dva uporabljena elementa sta tranzistorja tipa MOSFET [12] (*Metal–Oxide–semiconductor Field-effect Transistor*). Tranzistor je polprevodniški elektronski element, ki deluje kot nastavljiv ventil. Značilno zanj je, da ima tri priključke: vrata (*gate*), izvir (*source*) in ponor (*drain*). Odvisno od tipa tranzistorja lahko s pomočjo napetosti na vratih nadziramo pretok električnega toka med izvirom in ponorom [13].

Prvi je tranzistor IRFML8244TRPBF [14] proizvajalca International Rectifier. Vrsta tranzistorja je MOSFET tipa N-kanal. Tip N-kanal pomeni, da se z višanjem napetosti na vratih tranzistorja viša tok med izvorom in ponorom. Uporabljeni so za nadzor pretoka električnega toka za posamezne diode RGB LED z visoko

močjo, ki so uporabljene za osvetljevanje predprostora. Dodatni tehnični podatki so vidni v tabeli Tabela 3.1.

Deklariran maksimalni tok	5,8 A
Maksimalna napetost na ponoru	25V
Napetost na izvoru	1,7 V – 20 V
Število nožic	3
Območje delovanja	-55°C – 150°C

Tabela 3.1:Tehnični podatki za manj zmogljivi tranzistor [14].

Drugi uporabljen tranzistor pa je PSMN9R0-25MLC [15] proizvajalca NXP. Tudi ta tranzistor je tipa MOSFET N-kanal. Bistvena razlika pa je v tem, da je tokovno bolj zmogljiv. Zato je uporabljen pri *multipleksiranju* matrike diod LED, kjer skoz njega naenkrat lahko teče tok za napajanje 120 diod LED, ter za kontrolo močne LED bliskavice. Tehnični podatki so vidni v tabeli Tabela 3.2.

Deklariran maksimalni tok	55 A
Maksimalna napetost na ponoru	25V
Napetost na izvoru	1,5 V – 20 V
Število nožic	4
Območje delovanja	-55°C – 175°C

Tabela 3.2:Tehnični podatki za bolj zmogljivi tranzistor [15].

3.2.2 Načrtovanje vezja

Naloge, povezane s krmiljenjem LED matrike 12x20, ki jih želimo zaupati zunanjemu vezju, so:

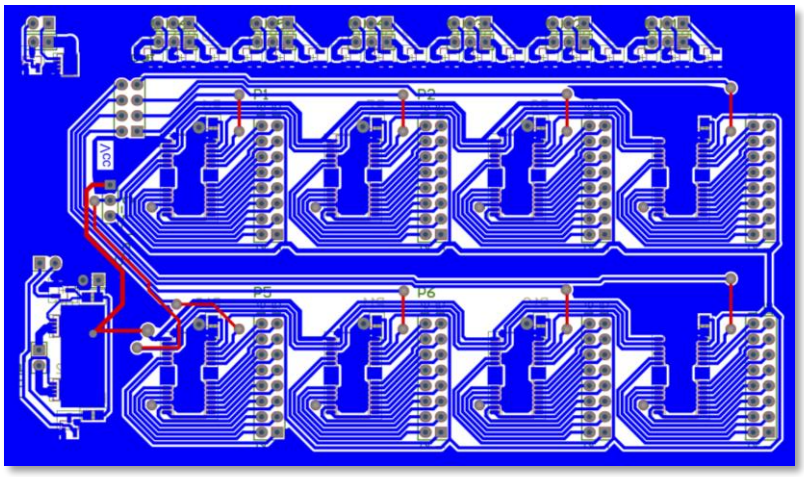
1. Krmiljenje celotne 12x20 LED matrike s pomočjo serijskega prenosa podatkov med razvojno ploščico STM32F4-Discovery in zunanjim vezjem.

2. Tokovno krmiljenje diod LED v matriki in multipleksiranje s faktorjem 2.

Poleg tega pa bomo na vezje dodali tudi tranzistorska stikala za krmiljenje diod RGB LED z visoko močjo in za krmiljenje močne LED bliskavice:

3. Napajanje in krmiljenje vseh diod RGB LED z visoko močjo, pri čemer naj se krmili vsaka barva posebej. Tukaj ni multipleksiranja.
4. Napajanje in krmiljenje močne bliskavice LED. Ta za napajanje potrebuje 12V.

Za potrebe prve in druge naloge, krmiljenja in napajanja 12x20 LED matrike, Prva in druga naloga, krmiljenje in napajanje 12x20 LED matrike, imamo na voljo 8 serijsko paralelnih 16-bitnih pomikalnih registrov BD7851FP-E2.Vsak tak čip ima 16 izhodov, kar znese zmožnost krmiljenja 128 diod LED. Kapaciteto krmiljenja diod LED pa še povečamo z uporabo tehnike, ki smo jo že opisali na začetku poglavja, multipleksiranjem. To pomeni, da lahko z vsemi osmimi 16-bitnimi pomikalnimi registri BD7851FP-E2 s stopnjo multipleksiranja 2 krmilimo 256 diod LED. Za zagotavljanje ustreznega tokovnega krmiljenja so v vezje dodani referenčni upori, ki jih zahteva 16-bitni serijsko-paralelni pomikalni register BD7851FP-E2, saj s tem nadzorujemo, s kolikšnim tokom naj poganja diode LED.



Slika 6: S programom Altium Designer [16] generirano vezje. V sredinskem delu vezja so vidni 16-bitni serijsko-paralelni registri za krmiljenje LED matrike. Na vrhu pa so v skupinah po 3 zbrani tranzistorji za krmiljenje RGB diod LED.

Tretja naloga je krmiljenje in napajanje diode RGB LED z visoko močjo. Vsaka barva pri vsaki od šestih diod RGB LED je vedno vezana na napetost. Vsak priklop za maso pa je speljan preko lastnega tranzistorja MOSFET. Odprtost vsakega tranzistorja nadzorujemo z lastnim signalom, ki je direktno povezan na razvojno ploščico STM32F4-Discovery. Vezje je narejeno tako, da je tranzistor lahko do konca odprt ves čas, saj je za tokovno krmiljenje poskrbljeno z dodatnimi uporniki. Podrobnosti o uporabljenih upornikih bomo spoznali v poglavju 3.4.

Četrta naloga, krmiljenje in napajanje močne bliskavice LED je podobno kot krmiljenje diod RGB LED, le da je v tem primeru uporabljen zmogljivejši tranzistor MOSFET. Vezje pa je načrtano tako, da je tranzistor vključen v ločen električni krog na zunanjem vezju, ki ima napetost 12V, ki je potrebna za napajanje močne bliskavice LED. Napetost na ostalih komponentah v vezju je sicer 5V.

Shema izdelanega vezja se nahaja v dodatnem poglavju.

3.2.3 Izdelava vezja

Fizična izdelava vezja je precej zahteven postopek, posebej za začetnika na tem področju, saj potrebuje veliko mero natančnosti zaradi majhnih povezav, pa tudi previdnosti, saj se v procesu uporabljajo tudi nekatere precej nevarne snovi.

Osnova za izdelavo vezja je prazna plošča iz ognjevarnega materiala, ki je prevlečena s plastjo bakra. Proizvajalci večkrat uporabljajo ime *Pertinax*, vendar gre v resnici za ploščico na papirni osnovi, ki je spojena z umetno smolo [17]. Ploščico je pred postopkom potrebno ustrezno pripraviti. Prvi korak je fino brušenje bakrene plasti pod vodo. Ploščico posušimo. Sedaj bakreno plast prelakiramo s foto lakom. Uporabljen je bil lak *Positiv 20*, ki se mora na zraku sušiti 24 ur.

Naslednji sklop se imenuje priprava filma vezja. S programom *Altium Designer* [16], kjer je bilo vezje načrtano, generiramo sliko vseh povezav v vezju, ki jo moramo nato zrcalno natisniti na prosojnico. Priporočljiva je uporaba laserskega tiskalnika.

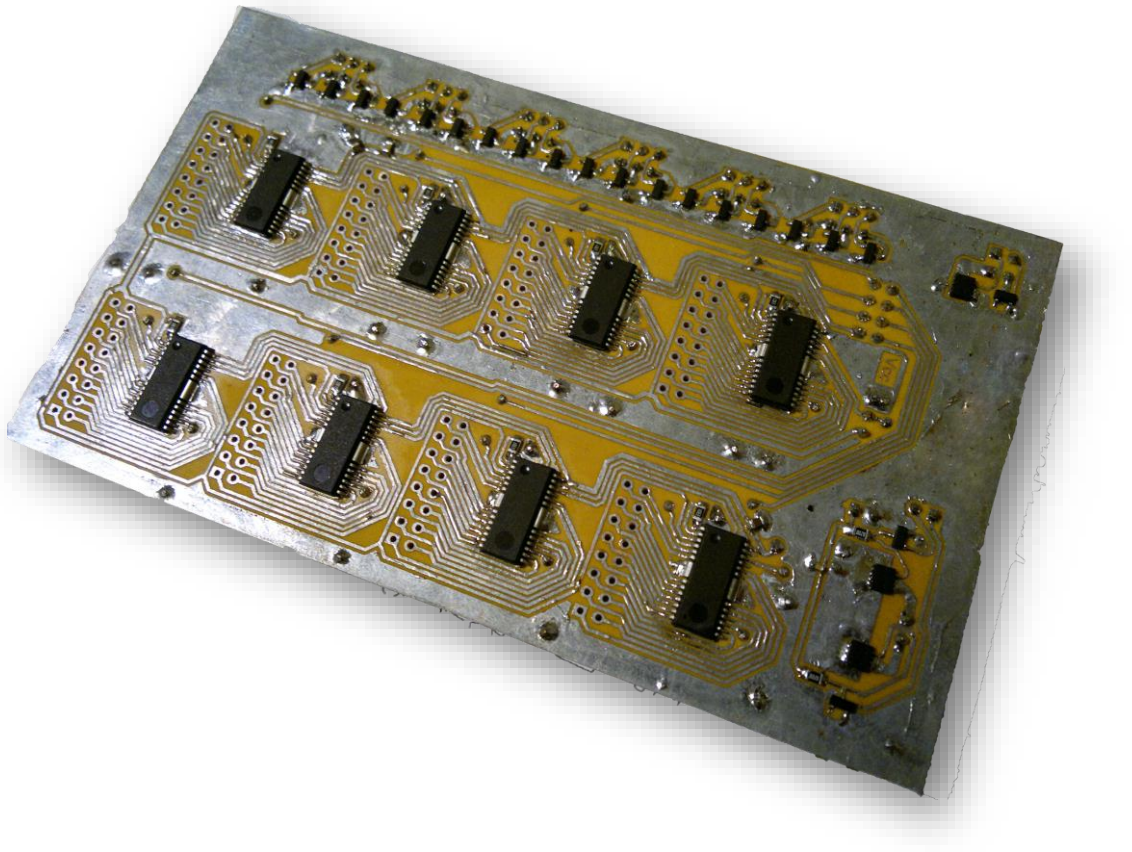
Sledi izpostavljanje ploščice UV svetlobi. Med UV lučko in bakreno ploščico, premazano s foto lakom, vstavimo film vezja. Svetloba reagira s foto lakom, vendar se mesta, kjer so povezave v vezju, nikoli ne osvetlijo.

Vezje moramo sedaj razviti. To storimo tako, da celotno vezje potopimo v raztopino natrijeve baze (NaOH). Vezje lahko v raztopini malce premikamo, da pohitrimo postopek. Foto lak se na mestih, kjer je bil obsevan z UV svetlobo, odstrani.

Zadnji korak pa je jedkanje. To je kemijsko vzpodbujen proces odstranjevanja bakra na mestih, kjer na bakreni plošči ni foto laka. Za jedkanje previdno pripravimo raztopino 50% vode, 35% klorovodikove kisline (HCl) in 15% vodikovega peroksida (H_2O_2). Vezje moramo zelo previdno potopiti v pripravljeno raztopino, saj je reakcija precej burna. Raztopina se takoj začne peniti. Ko se penjenje konča, je proces jedkanja končan. Vezje vzamemo iz raztopine za jedkanje in ga speremo pod vodo. Ves bakren premaz, ki ni bil zaščiten s foto lakom, bi moral biti odstranjen. Preostali foto lak na koncu odstranimo z razredčilom.

Sedaj so vse povezave v vezju končane. Potrebno pa je še zvrtati luknje na mestih, kjer dodajamo povezave, ter prispajkati vse elemente.

Slika 7 prikazuje povsem končan izdelek.

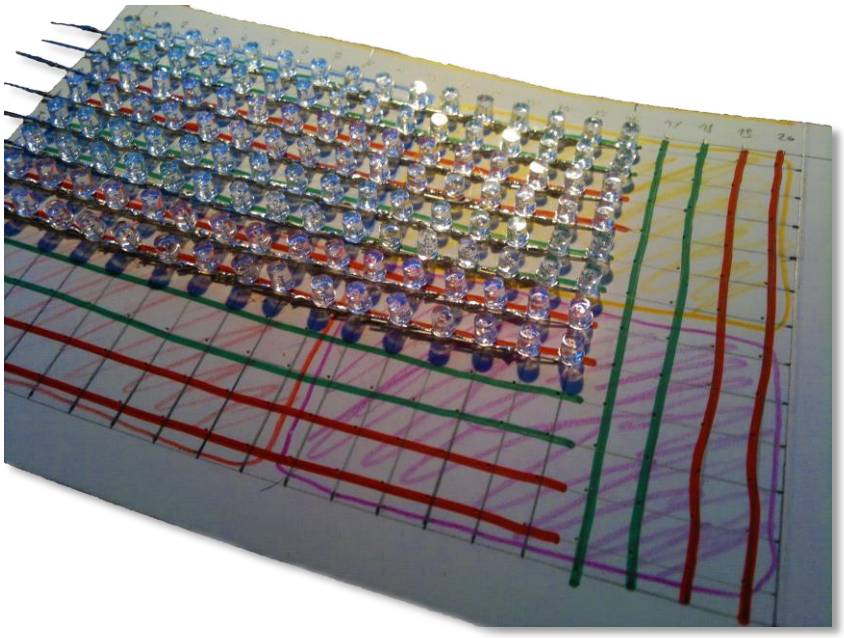


Slika 7: Dokončano vezje za krmiljenje diod LED..

3.3 LED matrika 12x20

LED matriko sestavlja 240 diod LED v 4 različnih barvnih kvadrantih. Vsak kvadrant vsebuje 60 diod LED enake barve. Kvadrant ima tako dimenzije 5x12. Uporabljene diode LED, naročene iz Kitajske, v pakiranju po 100, so visoko svetleče z ravno odrezano kapico za čimbolj razpršen curek svetlobe.

Prva faza pri izdelovanju LED matrike je bila priprava podlage, na katero bo matrika pritrjena. Na podlagi je jasno označeno, katere diode LED so povezane skupaj ter na katerega od 16-bitnih serijsko-paralelnih pomikalnih registrov se izvede priklop. Kot smo zapisali v poglavju 3.2.2, je bilo potrebno skleniti katode vseh diod LED v vrstici ter skleniti anodi po dveh vertikalnih sosedov naenkrat.



Slika 8: Napol dokončano povezovanje diod LED na podlagi za matriko.

Ko so bile diode LED pritrjene na podlago in ustrezno povezane, je sledilo prispajkovanje vseh 120 povezav. Za povezovanje so bili zaradi cenovne dostopnosti in prijetnega rokovanja z njimi uporabljeni UTP (*Unshielded Twisted Pair*) kabli. Najprej so bile povezave prispajkane na vezje. Tako povezani UTP kabli so bili porinjeni skozi prednjo stran ohišja nizkotonskega zvočnika, kjer so bili prispajkani še na pare diod LED na matriki. Anode diod LED, povezane v sode in lihe vrstice, pa so bile – lihe posebej in sode posebej - povezane na pripravljena tranzistorja MOSFET na vezju, s katerima bo izvedeno multipleksiranje.

3.4 RGB osvetlitev predprostora

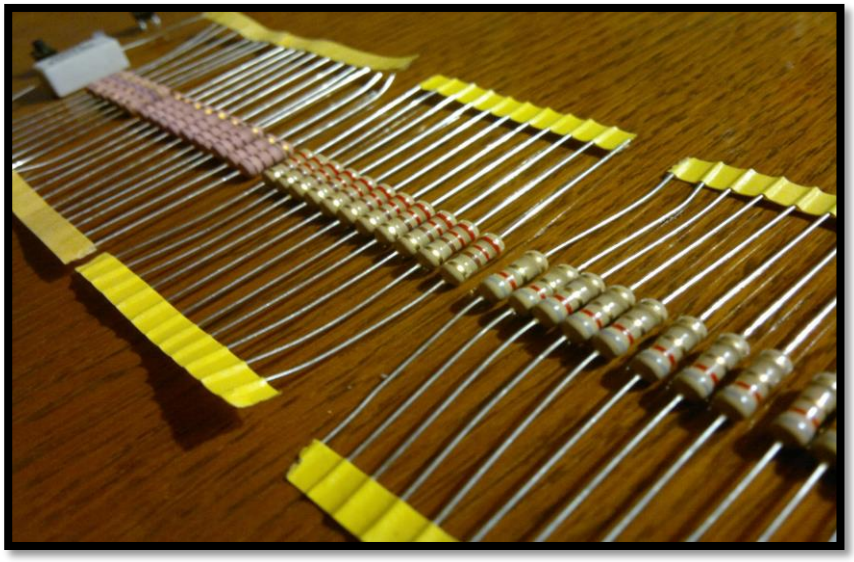
Pri diodah LED z visoko močjo je dodaten problem, ki ga je potrebno rešiti pri vgrajevanju, odvajanje toplote. S tem namenom je bil na prednjo stran nizkotonskega zvočnika v predprostor po celi dolžini na obeh straneh nameščen trak iz bakra, na katerega so bile potem pritrjene diode. Velik kos bakra zagotavlja hitro odvajanje toplote. Anode diod LED so bile povezane na +5 V napajanje. Katode pa so bile posamično povezane na ponor ustreznega tranzistorja MOSFET na pripravljenem zunanjem vezju. Za potrebe tokovnega krmiljenja so bili v

električne kroge dodani ustrezni uporniki. Upornost upornika za tokovno krmiljenje posamezne barvne diode LED celotne enote RGB LED je za vsako posamezno barvo drugačna, saj se nazivne napetosti in moči posameznih barvnih diod LED razlikujejo med sabo. Uporniki so vidni na sliki 10. Uporabljeni so uporniki zmogljivosti 2W, njihove upornosti pa so prikazane v tabeli Tabela 3.3.

Barva diode LED	**Nazivna napetost**	**Nazivni tok**	**Uporabljen upornik**
Rdeča	2,2 (1,9 – 3,1) V	385 mA	8,2 Ω
Zelena	3,5 (2,8 – 4,0) V	350 mA	4,7 Ω
Modra	3,5 (2,8 – 4,0) V	350 mA	4,7 Ω

Tabela 3.3: Izračunana upornost potrebnega upornika za tokovno regulacijo posameznih diod LED glede na barvo. Za računanje je bil uporabljen kalkulator [18].

Slika 9: Dioda RGB LED z visoko močjo, nameščena na bakren hladilnik.



Slika 10: Uporniki za tokovno krmiljenje diod RGB LED zmogljivosti 2W.

3.5 Močna LED bliskavica

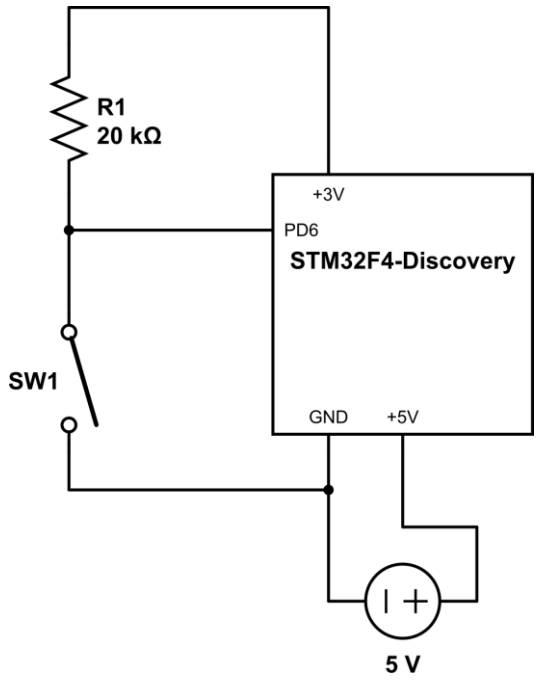
Tudi močna bliskavica LED je dioda LED z visoko močjo, v tem primeru 10W, torej tudi zanjo velja problem odvajanja toplote. Podobno kot diode RGB LED, je bila tudi ta povezana na ustrezen ponor tranzistorja MOSFET na zunanjem vezju. Za močno bliskavico LED je bil na vezju pripravljen elektrini krog z napetostjo 12V, saj je 12V nazivna napetost uporabljene diode LED. Za tokovno krmiljenje je bil v krog dodan tudi keramični 2 Ω upornik z zmogljivostjo 5W, viden na sliki 10 v zgornjem levem kotu.

Slika 11: Bliskavica LED, ki je dioda LED z visoko močjo, kar 10W, skrita pod pleksi steklom, ki je zadnja stena zračnika v nizkotonskem zvočniku.

Skupaj z aluminijastim hladilnikom je bila bliskavica vgrajena na zadnjo stranico zračnika nizkotonskega zvočnika ter prekrita s pleksi steklom v izogib proizvajanju hrupa v zračniku ob delovanju zvočnika.

3.6 Gumbi

Na prednjo stran ohišja so bili nameščeni tudi štirje preprosti gumbi - tipke. Njihov priklop je bil na razvojno ploščico STM32F4-Discovery izveden s pomočjo kroga z zunanjim *pull-up* uporom z upornostjo 20.000 Ω, kot je razvidno iz sheme na sliki 12. Tako ob sproščenem gumbu vhod na razvojni ploščici STM32F4-Discovery zaznava vrednost 1, ob pritisnjenem gumbu pa zaznava 0, saj se napetost iz *pull-up* upora zaradi kratkega stika z maso izniči.



Slika 12: Shema priklopa tipke na razvojno ploščico STM32F4-Discovery z uporabo pull-up upora.

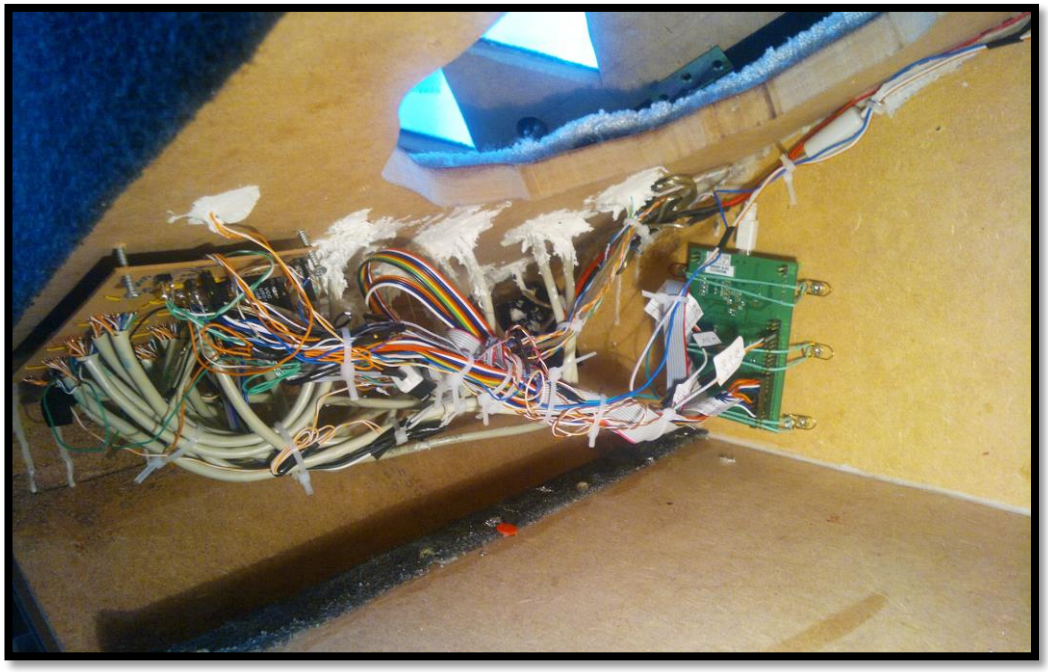
3.7 Napajanje

Sistem je napajan iz avtomobilskega akumulatorja, ki ima 12 V izhodno napetost pri ugasnjenem motorju in 14,4 V izhodno napetost pri prižganem. Ker pa skoraj

vse komponente potrebujejo 5 V napetost ali manj, pa je bil vgrajen tudi kupljen 50 W *step-down* pretvornik iz 12 V na 5 V. Model pretvornika je Car Power Technology 50W DC-DC Converter [19].

3.8 Povezovanje celega sistema

V posameznih podpoglavjih 3. poglavja smo se seznanili s pripravo posameznih komponent. Vse svetilne komponente smo že povezali z vezjem. Preostane nam le še pritrditev vezja na notranjo stran ohišja nizkotonskega zvočnika, pritrditev razvojne ploščice na notranjo stran nizkotonskega zvočnika in povezave med njima ter povezave med napajanjem. Omeniti velja, da je bil v razvojno ploščico STM32F4-Discovery priključen podaljšek USB kabla, katerega drugi konec je bil speljan iz ohišja nizkotonskega zvočnika. Tako bo programiranje mogoče tudi brez razdiranja ohišja nizkotonskega zvočnika. To poglavje je nekakšen zaključek priprave strojne opreme za sistem, saj je sedaj čas za programiranje razvojne ploščice STM32F4-Discovery. Na sliki 13 je razvidna pritrditev razvojne ploščice STM32F4-Discovery in zunanjega vezja v notranjost ohišja nizkotonskega zvočnika.



Slika 13: Vezje in razvojna ploščica STM32F4-Discovery z vsemi povezavami pričvrščeni na notranji strani prednje stranice ohišja nizkotonskega zvočnika.

Poglavje 4

4 Arhitektura programa

Pri določanju arhitekture programa sem na prvo mesto postavil zahtevo po preprostem programiranju novih animacij na čim bolj visokem nivoju[3]. Zato se je arhitektura programa v osnovi ločila na *ogrodje* in na *vizualizacijske aplikacije*.

Vizualizacijska aplikacija je v resnici preprost razred, kateri ogrodje ponuja veliko orodij za hitro kreiranje novih animacij. Gre za animacijske vrste ter pomočnike animiranja za vse vrste svetlobnih učinkov na vseh tipih osvetlitev. Ogrodje vizualizacijski aplikaciji vsiljuje tudi strukturo s pomočjo vmesnika, ki ga mora vizualizacijska aplikacija implementirati. V tem vmesniku so že definirani privzeti odzivi na vse dogodke, ki se lahko zgodijo (npr. prejem novih podatkov iz plasti za analizo zvoka ali pa odziv na pritisnjen gumb ali pa odziv na enega od časovnikov). Prav tako je že instanciranih nekaj osnovnih animacijskih komponent (npr. vrste za risanje na matriko). Ogrodje v programski zanki tudi nadzoruje življenjski cikel vizualizacijske aplikacije. Odloči se, kdaj jo bo instancirala, zagnala, jo ustavila, izbrisala instanco in instancirala novo. Ogrodje omogoča *reset* aplikacije in sprehajanje med aplikacijami.

Analiza vseh funkcij, ki jih bo imelo ogrodje, je pokazala, da bo program imel veliko različnih nalog, ki jih lahko združimo v zaokrožene celote, kot na primer vzorčenje in obdelava zvoka, krmiljenje strojne opreme ter generiranje vizualizacij. Marsikatere od teh nalog bo moral opravljati povsem neodvisno od drugih, a vendar sočasno z drugimi. Tipičen primer sta obdelava zvoka na eni strani ter animirane vizualizacije na drugi, čeprav sta to dve nalogi, ki si sicer enosmerno izmenjujeta podatke. Vendar je pomembno, da obe tečeta neprekinjeno ena ob drugi.

Deloma zaradi potrebe po neodvisnosti, deloma pa zaradi zelo dobrih izkušenj z bolj striktno strukturiranim načinom programiranja, sem se odločil, da ogrodje razdelim na štiri plasti:

[3] V tem primeru je z visokim nivojem mišljeno, da se pri programiranju ukvarjamo s problemi animiranja grafičnih objektov. Probleme na nižji ravni, kot so nadzorovanje posamezne diode LED ali pa krmiljenje le- teh, pa prepustimo ogrodju.

- **Plast za abstrakcijo strojne opreme** poskrbi, da je upravljanje s strojno opremo preprosto. Za upravljanje ji moramo posredovati le najbolj nujne podatke, za vse ostale podrobnosti pa poskrbi sama.
- **Plast za izvajanje življenjskih funkcij programa** skrbi za ključne procese, ki so starši vseh ostalih procesov. Reprezentativna primera sta programska zanka v metodi main() ter odziv na proženje sistemskega časovnika.
- **Plast za vzorčenje in obdelavo zvoka** skrbi za zajemanje vzorcev zvoka in njihovo obdelavo, na primer spektralno analizo in zaznavanje tempa.
- **Aplikacijska plast** definira pravila in načine uporabe nižjih plasti. Vsebuje kopico animacijskih modulov, ki so na voljo za preprosto uporabo.

Celotna struktura je vidna tudi na sliki 14. Podrobnejše funkcije vsake plasti bom posebej razložil v lastnem podpoglavju.

Za realizacijo tako razbite in plastovite arhitekture sem se programiranja lotil objektno v jeziku C++. Objektno programiranje pri vgrajenih sistemih običajno ni najbolj tipična izbira, predvsem zaradi večje porabe pomnilnika in slabše zmogljivosti.

Slika 14: Plastovita arhitektura programa. Na vrhu vseh plasti, ki tvorijo ogrodje, domujejo vizualizacijske aplikacije.

4.1 Plast za abstrakcijo strojne opreme

Plast za abstrakcijo strojne opreme se nahaja na samem dnu programa, saj jo višje plasti uporabljajo za interakcijo s strojno opremo. Direktne interakcije s strojno opremo v višjih plasteh praviloma ni. V napisanih modulih smo se posvetili predvsem abstrakciji svetilnih naprav, ki smo jih želeli uporabljati kot zunanje naprave, in ovijanju, konfiguraciji ter poenostavitvi uporabe strojne opreme. Razvita plast za abstrakcijo strojne opreme pa s strojno opremo ne komunicira direktno, temveč za komunikacijo uporablja uradno knjižnico STM32F4-Discovery Firmware Library (STSW-STM32068) [8]. Naloge te plasti so naslednje:

1. Doseganje čim večje neodvisnosti od uporabljene strojne opreme. Ob zamenjavi uporabljene razvojne ploščice bi bilo v programu potrebno prirediti to plast in uporabljeno uradno STM32F4-Discovery Firmware knjižnico. Ostale plasti pa bi ostale nedotaknjene. Manj ekstremen primer pa je recimo premik priključka iz ene vhodno-izhodne nogice na drugo. V tem primeru pa zadostuje le manjša sprememba v navedeni konfiguraciji, ki je tudi del te plasti.
2. Poenostavljena interakcija s strojno opremo. Plast za abstrakcijo strojne opreme poskrbi za vse potankosti krmiljenja s strojno opremo ter višjim plastem ponudi kar se da preprosto upravljanje s strojno opremo.
3. Logično modeliranje naprav, ki jih krmilimo preko univerzalnih vhodno-izhodnih nožic. Plast za abstrakcijo nam denimo ponuja LED matriko kot eno izmed strojnih enot.

4.1.1 Abstrakcija strojnih enot

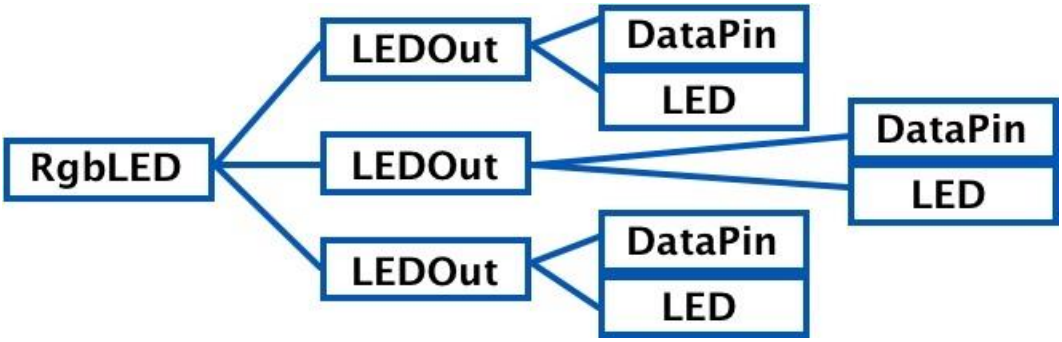
Abstrakcija strojnih enot je namenjena modeliranju zunanjih naprav oziroma svetilnih elementov, s katerimi manipuliramo preko izhodnih nožic razvojne ploščice STM32F4-Discovery. Te naprave želimo abstrahirati in za njih izdelati objekte, ki se navzven predstavljajo s funkcijami, ki se zdijo intuitivne za modeliran objekt (npr. določanje stanja in intenzitete ali pa barve diode LED), v ozadju pa same poskrbijo za primerno komunikacijo z napravo in tvorjenjem signalov za izvedbo željene funkcije.

Abstrakcija se začne pri razredih *DataPin* in *DataPort*, ki modelirata podatkovne nožice in podatkovna vrata na ARM ploščici. *DataPin* objekt predstavlja eno

nožico na ploščici. Določimo lahko tip (vhodni, izhodni, analogni), mu nastavljamo vrednost izhoda ali pa beremo. Nožica (*DataPin*) ve, katerim vratom (*DataPort*) pripada, saj drži referenco na vrata. Več o tem, zakaj je to pomembno, pa v poglavju o čarovnikih za inicializacijo 4.1.2.

Instanca razreda *LED* predstavlja en primerek diode LED. Stopnja abstrakcije je tukaj najvišja, saj ni pomembno, kakšnega tipa je dioda LED. Ima stanje (aktivno, neaktivno) ter intenziteto.

Razred, ki povezuje instanci obeh zgoraj omenjenih razredov, pa je razred *LEDOut*, ki vsebuje instanco razreda *LED*, poleg tega pa vsebuje tudi referenco na *DataPin* objekt, torej na nogico, preko katere se (preko tranzistorja MOSFET) dioda LED krmili. Ta razred vsebuje tudi metodo *sendOut()*, katera proži , da se stanje objekta *LED* prenese na nogico. Metoda *sendOut()* naj bi se klicala zelo pogosto iz ene od prekinitev časovnika. V razredu *LEDOut* je razrešena tudi problematika intenzitete, saj razred namreč shranjuje tudi zaporedno številko osvežitve. Razred tudi poskrbi, da bi bilo osvetljevanje čimbolj enakomerno in bi bilo utripanje čim manj opazno tudi pri nizkih intenzitetah in počasnejšem osveževanju[4]. Ta razred se uporablja za močno bliskavico LED.



Slika 15: Gnezdenje objektov v objekt RgbLED.

[4] Intenziteta, ki je število med 0 in 100 se primerja s številko osvežitve, ki je ostanek pri deljenju s 100. Za čimbolj enakomerno osveževanje se številki osvežitve, pred deljenjem prišteje številu 50 najbližje praštevilo – 47. Ta zagotovi, da se pri npr. 50% intenziteti diode LED približno enakomerno vklaplja in izklaplja. Saj bi bila v nasprotnem primeru dioda LED prvih 50 osvežitev ugasnjena, naslednjih 50 pa prižgana, kar bi človeško oko lahko zaznalo kot utripanje.

Razred, ki ovija kar tri instance razreda *LEDOut*, pa je razred *RgbLED*. Ta predstavlja načrt za objekt, ki ustreza eni diodi RGB LED za osvetljevanje predprostora. Vsaka instanca vsebuje 3 instance *LEDOut* objektov, saj je tudi fizično dioda RGB LED sestavljena iz treh diod LED. Za lažjo predstavo je v pomoč slika 15, ki prikazuje, kateri objekti so ugnezdeni v instanco razreda *RgbLED*. Dodatna zmožnost razreda *RgbLED* pa je seveda nastavljanje barve ene enote RGB LED. Barvo podamo kot parameter metodi *setColor()*, ki sprejme celoštevilski parameter, ki predstavlja barvo. Zadnji dve mesti cifre ustrezata procentu modre barve, 3. in 4. mesto procentu zelene, ter 5. in 6. mesto procentu rdeče barve. Intenziteta vsake barve je tako lahko regulirana na območju od 1-100, kar skupaj pomeni zmožnost generiranja milijon različnih oziroma različno intenzivnih barv. Takoj ko metoda razpozna posamezne barve iz podanega parametra, jih pretvori v intenzitete, ki jih nastavi vsem trem ugnezdenim instancam razreda *LEDOut*. Tudi ta razred vsebuje metodo *sendOut()*, ki pa sicer le pokliče *sendOut()* na vseh treh ugnezdenih *LEDOut* objektih.

Razred *LEDPanel* pa je najbolj kompleksen razred v celotni plasti abstrakcije strojne opreme. Njegova naloga je krmiljenje 16-bitnih serijsko-paralelnih pomikalnih registrov in prikaz stanja notranjega polja objektov *LED* na matriki diod LED. Vsebuje reference na *DataPin* objekte za nožice, ki komunicirajo z zunanjim vezjem za krmiljenje LED matrike. To so podatkovne nožice za dovod podatkov v 16-bitne pomikalne registre, nožica za urin signal in signal za zamenjavo vrednosti (*latch*) ter signala za multipleksiranje.

Poleg referenc na nožice za tvorjenje signalov pa ta razred vsebuje tudi tri matrike:

1. Matrika *LED* objektov velikosti 12x20. Ta matrika je edina vidna navzven in je razkrita za uporabo na višjih plasteh. S pomočjo manipulacije te matrike programer določa stanje fizičnih diod LED v matriki.
2. Matrika *boolean* vrednosti enake velikosti, ki določa aktivne diode LED pri naslednjem ciklu osveževanja. Ta matrika se re-kreira ob vsaki (drugi) osvežitvi. V njej je s podobno tehniko kot v objektu *LEDOut* poskrbljeno za ustrezno odražanje intenzitete diod LED skozi več ciklov osveževanja.
3. Matrika *boolean* vrednosti polovične velikosti, dimenzij 8x16. Ta matrika pa drži podatke za posamezne diode LED, ki so priklopljene na vsakega od osmih 16-bitnih serijsko-paralelnih pomikalnih registrov. Ta matrika je

direktno pripravljena za preslikovanje stanja na zunanje vezje. Za generiranje te matrike je potrebno ustrezno permutiranje.

Kot je razvidno iz slike 16, se zadnji dve matriki kreirata tik pred osveževanjem. Razred sicer omogoča tudi nekaj dodatnih funkcionalnosti, kot je na primer ugašanje vseh LED diod.

```
/* Send the bianry matrix data to shift registers */
/* Create binary matrix should be called before it (at least every other call) */
void LEDPanel::sendOut() {

    if (this->oddLine)
        this->createBinaryMatrix();

    LEDPanelPermutations::permutationMap(this->outputMatrix, this->binaryMatrix, this->oddLine);

    this->pinClk->clear();
    this->pinLatch->clear();

    //16x times send one bit to all 8 registers and cycle clock inbetween
    for (int i = 0; i < 16; i++) {

        for (int j = 0; j < 8; j++) {
            if (this->outputMatrix[j][i] > 0)
                this->pinData[j]->set();
            else
                this->pinData[j]->clear();
        }

        this->pinClk->set();
        this->signalCycleDelay();
        this->pinClk->clear();

    }

    this->pinMpx[0]->clear();
    this->pinMpx[1]->clear();

    //Latch!
    this->pinLatch->set();
    this->signalCycleDelay();
    this->pinLatch->clear();

    //Multiplexing
    if (this->oddLine)
        this->pinMpx[0]->clear();
    else
        this->pinMpx[1]->clear();

    this->oddLine = !this->oddLine;

}
```

Slika 16: Programska koda metode sendOut(), ki stanje notranjega polja objektov LED preslika v enako stanje na fizični LED matriki

4.1.2 Inicializatorji strojne opreme

Inicializatorji strojne opreme so preprosti razredi, ki na podlagi nekaj parametrov inicializirajo strojno opremo.

- *AudioSamplingInitialization* – vsebuje metodo za inicializiranje DMA (*Direct Memory Access*) krmilnika, za inicializiranje ADC (*Analog-to-digital converter*) pretvarjalnika, za vzpostavitev povezave med njima ter za konfiguriranje prekinitev ob končanem prenosu bloka. Vsebuje pa tudi metodo za pričetek pretvarjanja analognega signala v digitalno obliko in za pričetek DMA prenosa.
- *ButtonInitializations* – vsebuje metode za inicializacijo vsakega od štirih gumbov vključno s konfiguracijo EXTI (*External Interrupt*) prekinitvenih kanalov, ki samodejno zaznajo pritisk gumba in prožijo prekinitev.
- *TimerInitializations* – vsebuje metode za inicializacijo sistemske ure (*sysTick*) ter treh uporabljenih časovnikov TIM (*Timer*) in konfiguracijo prekinitev, ki jih prožijo.
- *DataPortInit* – inicializacija podatkovnih vrat in nožic (ki se v resnici inicializirajo v sklopu vrat). Razred prebere podatke o vseh uporabljenih nožicah iz konfiguracije. Vsaka nožica nosi podatek o pripadnosti vratom ter podatek o tipu (vhodna, izhodna, analogna). S pomočjo teh podatkov se sestavijo maske za inicializiranje vhodnih, izhodnih ter analognih nožic na vsakih vratih posebej. Nato se zgodi inicializacija. Ob menjavanju uporabljenih nožic nam torej ni potrebno skrbeti za inicializacijo.

4.1.3 Konfiguracija strojne opreme

Konfiguracija, ki je sicer manjša množica statičnih razredov, predstavlja nekakšen popis priklopov. V njej so definirane v prvi vrsti vse uporabljene nožice in podatkovna vrata. V drugi vrsti pa vse priključene svetilne naprave in vsi signali ter podatki o tem, katera izhodna nožica je uporabljena zanje.

Dodatno so v dodatnem razredu opisane permutacije LED matrike, kar pomen,i da je opisano, katera dioda LED je priključena na katero nožico katerega 16-bitnega serijsko paralelnega pomikalnega registra.

4.2 Plast za izvajanje življenjskih funkcij programa

Ta plast drži v rokah vse vajeti programa. Če v neki točki program v načinu za razhroščevanje ustavimo in pregledamo klice zbrane na skladu, bomo v vsakem primeru videli, da je klic čisto na vrhu prišel iz te plasti. V tej plasti so zbrane vse vstopne točke v program. To je poleg začetne točke v metodi main() tudi vstop iz vseh prekinitev. Poleg tega pa ta plast povsem nadzoruje delovanje vizualizacijske aplikacije, ki jo instancira in požene.

4.2.1 Programska zanka

Programska zanka je zanka, v kateri se program zadržuje med tem ko čaka na dogodke. Kroženje v programski zanki preprečuje, da bi se program izvršil do konca in končal izvajanje. V našem primeru je programska zanka realizirana na dveh stopnjah; v metodi *main()* in tudi kot del vsake vizualizacijske aplikacije.

Programska zanka v metodi *main()* ponavlja naslednji cikel:

1. Instanciraj vizualizacijsko aplikacijo.
2. Zaženi jo in ji predaj kontrolo.
3. V primeru, da se kontrola vrne, počisti trenutno vizualizacijsko aplikacijo.
4. Poženi aplikacijo za izbiro naslednje vizualizacijske aplikacije.

V koraku 2 se vizualizacijski aplikaciji preda kontrola. To pomeni, da se pokliče metoda *start()*, iz katere se program ne vrne, vse dokler izvajanje vizualizacijske aplikacije ni zaključeno. Vizualizacijska aplikacija se med tem časom ohranja pri življenju tako, da se vrti v lastni programski zanki. Najbolj običajen način uporabe je, da to programsko zanko prekine le pritisk na enega od gumbov, ki mu je dodeljena funkcija zapiranja aplikacije.

4.2.2 Razpečevalec prekinitev

Ko se zgodi prekinitev, prekinitveni vektor prekinitev takoj usmeri v razpečevalca prekinitev. Razpečevalec prekinitev je programski modul, ki smo ga razvili z namenom, da sprejema vse prekinitve od vseh naprav, ki so konfigurirane da jih prožijo.

Običajen odziv je, da razpečevalec poskrbi za prekinitev do te mere, da ponastavi bit za čakajočo prekinitev in nastavi prekinitev, da se znova sproži, če tip naprave to zahteva (npr. časovniki). Sicer pa zahtevo po prekinitvi le poda naprej v aplikacijsko plast.

Razlog za tako strukturo leži v razdelitvi programa na ogrodje in vizualizacijske aplikacije. V resnici so vizualizacijske aplikacije tiste, ki morajo izvesti pravi odziv na prekinitev. Vendar se te lahko zamenjujejo ali pa se lahko celo zgodi, da v določenem času ni instancirana nobena vizualizacijska aplikacija. Zato se vse prekinitve zberejo na ogrodju, v razpečevalcu prekinitev, nakar se ustrezno posredujejo trenutno aktivni vizualizacijski aplikaciji.

Prekinitve, za katere poskrbi razpečevalec:

- Iztekla se je čakalna doba sistemske ure (*sysTick*).
- Iztekla se je čakalna doba enega od 12 konfiguriranih časovnikov (na treh napravah TIM2, TIM3 in TIM4).
- Vzorec glasbe je bil zajet in je pripravljen za obdelavo.
- Vzorec glasbe je bil obdelan.
- Pritisnjen je bil eden od gumbov.

4.3 Plast za vzorčenje in analizo zvoka

Naloga te plasti je, da vzorči zvočni signal, ki je priključen na nožico razvojne ploščice STM32F4-Discovery, vzorce obdela – izvede spektralno analizo – ter oskrbi in opozori aplikacijsko plast, da so novi podatki o prisotnosti zvočnih spektrov na voljo.

4.3.1 Vzorčenje zvoka

Vzorčenje zvoka je način pretvarjanja zvoka ali analognega električnega signala zvoka v digitalni signal. To storimo tako, da v določenih časovnih intervalih odčitavamo vrednost signala. Pri tem pa je pomembno, kako pogosto signal odčitavamo in kako natančno ga odčitavamo [20].

Pogostost odčitavanja signala ima direkten vpliv na razpon frekvenc, ki jih s takim vzorčenjem še lahko zajamemo. Nyquist-Shannonov teorem [21] pravi, da je za zajem valovanja potrebno signal vzorčiti vsaj z dvakratno frekvenco

osnovnega signala. Ob upoštevanju, da človeško uho zaznava frekvence do približno 20 kHz [22], to pomeni, da vzorčenje hitrejše od 40.000 vzorcev na sekundo ni preveč smiselno, če imamo v mislih reprodukcijo zvoka, kjer bo ciljni poslušalec človek.

Natančnost odčitavanja signala pa pomeni, s kolikimi biti bomo zapisali odčitane vzorce. Pokazano je bilo, da za človeško uho zadostuje zapis signala s 16-bitnim številom. To pomeni, da lahko vzorce opišemo z 65.536 različnimi vrednostmi. Kot studijska kvaliteta pa sicer velja 24-bitni zapis [23].

Za vzorčenje signala za potrebe vizualizacije zvoka je bil prvotni cilj, da se pokrije celoten človeški slišni spekter. To mi je tudi skoraj povsem uspelo.

Razvojna ploščica STM32F4-Discovery s pomočjo konfiguriranega ADC – analogno-digitalnega pretvornika vzorči zvočni signal. Zmogljivost analogno-digitalnega konverterja je 12 bitov. Če bi želeli reproducirati tako vzorčen zvok, bi človeško uho verjetno zaznalo, da reprodukcija ni povsem *čista*. Vendar za potrebe vizualiziranja zvoka to zadostuje. ADC je konfiguriran tako, da ob vsakem zajetem vzorcu, ki ga zapiše v svoj register, pošlje DMA krmilniku signal, da ima pripravljen nov podatek. DMA krmilnik je krmilnik, ki omogoča direkten dostop do pomnilnika. Omogoča, da naprave dostopajo (berejo in pišejo) v glavni pomnilnik brez sodelovanja procesorja [24]. V našem primeru DMA krmilnik zapisuje vzorce, ki jih generira ADC v vnaprej pripravljen podatkovni blok v pomnilniku, ki predstavlja polje vzorcev (12 bitnih celih števil). To se dogaja povsem asinhrono. Med tem časom procesor obdeluje podatke iz prejšnjega okna vzorčenja. Šele ko se blok zapolni do konca, DMA krmilnik proži prekinitev, ki v programu sproži obdelavo pravkar zajetih podatkov.

Okno vzorčenja je število vseh vzorcev, ki jih naberemo, preden se odločimo, da jih bomo obdelali. V našem primeru je okno vzorčenja velikosti 1024. Velikost rezerviranega pomnilniškega bloka pa je sicer 2x večja, saj je program sestavljen tako, da izmenično nabira in obdeluje podatke iz nasprotnih polovic bloka.

Z oknom vzorčenja dolžine 1024 se je zgornja meja za odzivno delovanje izkazala kot približno 35 vzorčenih in obdelanih oken na sekundo. To pomeni vzorčenje s frekvenco približno 35,8 kHz, kar ob upoštevanju Nyquist-Shannonovega teorema pomeni, da smo uspešno vzorčili frekvence do skoraj 18 kHz, kar povsem zadostuje, saj človek zelo hitro gluši za frekvence pri koncu slišnega območja.

4.3.2 Hitra Fouriereva Transformacija (FFT)

Hitra Fouriereva transformacija (*Fast Fourier Transformation*) [25] je algoritem, ki zelo hitro izvede diskretno Fourierevo transformacijo in njeno inverzno transformacijo.

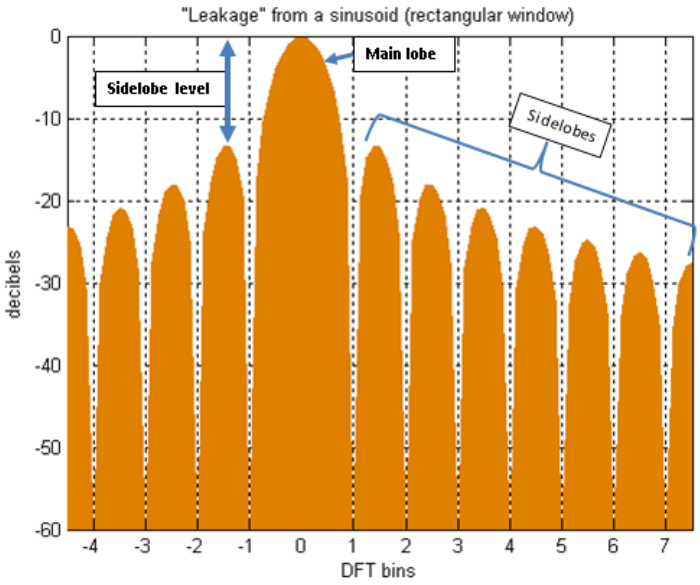
Diskretna Fouriereva transformacija je transformacija, ki signal, vzorčen v časovnem prostoru (zapis signala z amplitudo v trenutkih vzorčenja), pretvori v signal, zapisan v frekvenčnem prostoru (signal je zapisan kot množica amplitud posameznih frekvenc).

Transformacija velja za diskretno, saj število vhodnih podatkov v transformacijo ustreza številu izhodnih podatkov. Izhodni podatki iz diskretne Fourierеve transformacije pa so frekvenčni spektri (*Frequency bins*). Po Nyquist-Shannonovem teoremu znaša najvišja frekvenca, ki jo še zaznamo z vzorčenjem, ravno polovico hitrosti vzorčenja. Tako se frekvenčni spektri enakomerno razporedijo na intervalu med nič in polovično frekvenco vzorčenja. Vsak frekvenčni spekter ima tako podatek o povprečni amplitudi zelo ozkega območja frekvenc.

To pomeni, da je v primeru, da se transformacija izvaja nad podatki, zajetimi v enem oknu vzorčenja, število frekvenčnih spektrov oziroma ozkost frekvenčnega spektra pri določanju povprečne amplitude odvisna od dolžine okna.

4.3.3 Okenske funkcije

Ena od težav pri uporabi DFT (*Discrete Fourier Transformation*) je tudi ta, da je Fouriereva transformacija namenjena obdelavi neskončnih signalov. Ker pa želimo pognati transformacijo na zelo kratkem intervalu, pa lahko pride do posebnega pojava frekvenčnega puščanja (*frequency leakage*). Pojav je ilustriran na sliki 17. Če pojav na hitro razložimo na primeru, to pomeni, da so tudi pri izvajanju transformacije nad čistim sinusnim signalom z eno samo določeno frekvenco v rezultatu poleg ustreznega frekvenčnega spektra, deloma prisotni tudi sosednji frekvenčni spektri.



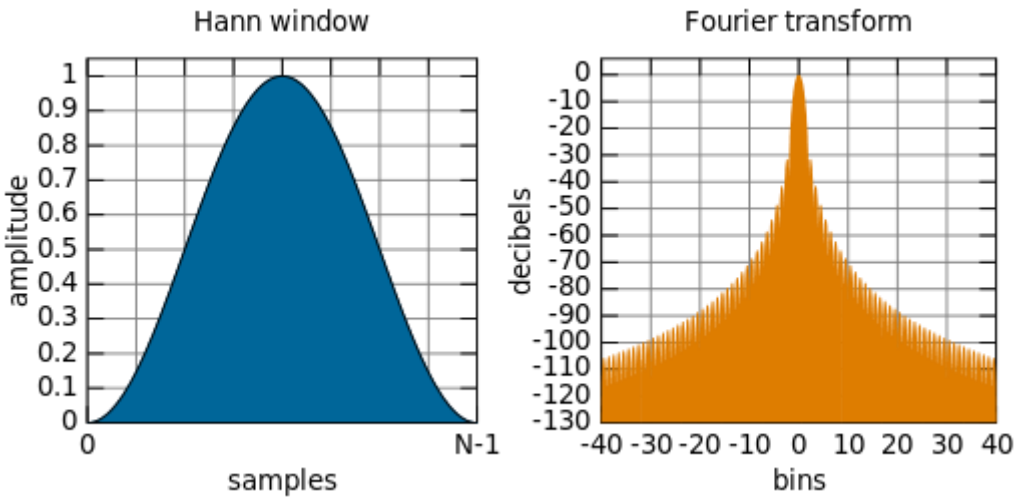
Slika 17:V frekvenčnem spektru z oznako 0 je prikazana amplituda sinusoide za katero je bila narejena transformacija. Amplitude v ostalih spektrih pa predstavljajo frekvenčno puščanje, saj v izvornem signalu te frekvence niso bile prisotne [26].

Eden od načinov, kako se lahko spopademo s to težavo, je ta, da nad oknom vzorcev pred transformacijo poženemo okensko funkcijo. Okenska funkcija je funkcija, ki signal popravi tako, kot da bi se začel čisto potihoma (z majhnimi amplitudami) in tako tudi končal. Za okensko funkcijo velja, da je njena vrednost izven območja okna enaka nič. Ena izmed bolj primernih je Hanningova okenska funkcija [26]. Kakor vidimo v enačbi 18, se vsak vzorec pomnoži z ustreznim kosinusom, ki se računa kar nad razmerjem zaporedne številke vzorca in številom vseh vzorcev.

$$w(n) = 0.5\ \left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$$

Slika 18:Enačba funkcije Hanningovega okna [26]..

Rezultat Hanningove funkcije je tako signal, ki na začetku in na koncu pojema. Na sliki 19 tako lahko vidimo zmanjšan vpliv frekvenčnega puščanja



Slika 19: Graf Hanningovega okna in graf zmanjšanega vpliva frekvenčnega puščanja okrog vodilne frekvence [26].

4.3.4 Analiza frekvenčnih spektrov

Vzorčenju zvoka sledi analiza frekvenčnih spektrov. Vhodni podatki predstavljajo zajeto okno vzorcev avdio signala. Ta proces poteka v naslednjih korakih:

1. Umik podatkov iz pomnilniškega bloka, ki ga uporablja DMA krmilnik.
2. Priprava podatkov z okensko funkcijo. Uporabi se Hanningovo okno.
3. Priprava podatkovnih struktur, ki jih algoritem potrebuje za izvedbo hitre Fouriereve transformacije.
4. Izvedba realne hitre Fouriereve transformacije. Rezultat ima obliko parov realne (amplituda) in imaginarne (fazni zamik) komponente. Za računanje se uporabi na prosto dostopna knjižnica, ki velja za zelo optimizirano. Njen avtor je Takuya Ooura iz Raziskovalnega inštituta za matematične vede Kjotske univerze [27].
5. Velikost amplitude je izračunana s pomočjo korenjenja seštetih kvadratov obeh komponent.
6. Amplitude so razvrščene v 20 spektralnih območij, za vsako območje je izračunano povprečje. Število vključenih amplitud oziroma spektrov v vsako območje eksponentno narašča. Tako sta prvi dve območji območji

od približno 70-105 Hz in 105-175 Hz, medtem ko je zadnje območje precej širše, saj se razpenja od 17 do 18.6 kHz.[5]

7. Nad spektralnimi območji je izveden logaritem, ki gibanje vrednosti pogladi in prepreči preveč izrazito nihanje, saj se v praksi izkaže, da so v glasbi v večji meri ves čas prisotni skoraj vsi frekvenčni spektri.
8. Za prikaz samo razlik in nihanj v posameznih amplitudnih skupinah je od vseh skupin odšteto 80% drsečega povprečja. Drseče povprečje se počasi prilagaja, saj se ob vsakem zajemu podatkov malce prilagodi povprečju. Tak sistem ne zaduši prikaza energijskih sunkov glasbe, obenem pa je sistem prilagodljiv za jakost glasbe. Ob spremembi jakosti se tako sistem v nekaj sekundah povsem samodejno prilagodi.

Kot dodatek k 8. točki je bil izdelan še dodaten način učenja, ki je aktiviran ob zagonu ali sprožen na enega od gumbov. V tem načinu se drseče povprečje spreminja v velikih korakih kar omogoči takojšnjo prilagoditev na glasnost. Tako se hitro vzpostavi relevantna začetna vrednost glasnosti, način učenja pa se po 10s zaključi.

Ko so vsi podatki pripravljeni, se preko razpečevalca prekinitev v plasti za življenjske funkcije programa pokliče prekinitev, da so glasbeni podatki pripravljeni za prikaz. Ogrodje to posreduje vizualizacijski aplikaciji, ta pa se sama odloči, kaj bo storila s podatki.

[5] To ustreza porazdelitvi, ki jo najdemo pri večino avdio izenačevalnikih v priljubljenih predvajalnikih glasbe. Razlog za tako porazdelitev pa je v tem, da smo ljudje manj občutljivi za zelo visoke tone, saj jih slabše slišimo in težje razločimo. Medtem ko zelo dobro razlikujemo med zelo nizkimi in dobro med srednjimi toni.

4.4 Aplikacijska plast

Aplikacijska plast poskrbi za dvig nivoja programiranja animacij, saj vizualizacijskim aplikacijam nudi orodja za lažjo kreacijo in izvajanje animacij. Poleg dviga nivoja programiranja pa vizualizacijskim aplikacijam tudi vsili strukturo s pripravljenim vmesnikom.

4.4.1 Vmesnik za vizualizacijske aplikacije

Razred *Application* v resnici ni čisto pravi vmesnik, saj vmesnikov, takih kot so na primer v Javi, C++ ne pozna. Gre torej le za razred z virtualnimi metodami, ki je implicitno namenjen temu, da ga vizualizacijska aplikacija razširi.

Definicija razreda *Application* je vidna na sliki 20, njegova vsebina je naslednja:

- Konstruktor in destruktor. Vsaka vizualizacijska aplikacija mora za seboj počistiti vse podatke iz kopice (*heap*).
- Metoda *start()*. V tej metodi mora virtualizacijska aplikacija imeti implementirano aplikacijsko zanko. V primeru da ne vsebuje re-definicije te metode pa se uporabi generična.
- Metodi *quit()* in *stop()*. Metodi prekineta programsko zanko in uničita objekt.
- Metode časovnika. Vsaka aplikacija ima možnost implementirati katerokoli od metod časovnika. Na voljo je 8 metod, ki se prožijo na zelo različne časovne intervale vse od 1/64 sekunde pa do 5 sekund.
- Metode za odzive na gumbe. Na voljo so tudi 4 metode za odzive na gumbe. Vizualizacijska aplikacija lahko implementira katero koli od njih, vendar se priporoča, da se gumbu s funkcijo *quit* prepusti njegova generična funkcionalnost.
- Metodi za prejem novih podatkov iz modula za analizo zvoka. Razpečevalec prekinitev preko ogrodja opozori virtualizacijsko aplikacijo vsakič, ko so bili zajeti novi podatki in tudi vsakič, ko so bili podatki obdelani in so na voljo podatki o trenutno prisotnih spektrih.

Razširjena oblika razreda *Appication* poimenovan *AnimatedApplication* pa poleg vsega naštetega vsebuje še:

- Instanci sinhronizatorja za LED matriko in animatorja za animiranje RGB osvetlitve v predprostoru. Vsaka animirana vizualizacijska aplikacija jih namreč potrebuje.
- Razred v programski zanki ne čaka, ampak obdeluje podatke o analiziranih spektrih. Izvaja tudi analizo tempa.
- Razred dodaja še en dodaten dogodek, ki ga sproži interno. To je dogodek newBeat(), ki označuje dogodek udarca. Vizualizacijska aplikacija pa se lahko odzove nanj.

```
class Application {

    public:

    Application();
    virtual ~Application();

    virtual void start();
    virtual void stop();

    virtual void r64second();
    virtual void r32second();
    virtual void r16second();
    virtual void r8second();

    virtual void quarterSecond();
    virtual void halfSecond();
    virtual void oneSecond();
    virtual void fiveSeconds();

    virtual void buttonLeft();
    virtual void buttonQuit();
    virtual void buttonRight();
    virtual void buttonConfirm();

    virtual void newSpectrumData(double* spectra, int spectraCount);
    virtual void newSampledSoundData();

    virtual void quit();

    protected:

    bool quitApplication;



};
```

Slika 20: Glava definicije razreda Application, ki služi kot vmesnik za vsako instanco vizualizacijske aplikacije.

4.4.2 Izvrševalci animacij

Animiranje svetlobnih efektov na LED matriki in na RGB osvetlitvi predprostora je poenostavljeno z izvrševalcema animacij; razredoma ki proces animiranja poenostavita in izboljšata.

Za animiranje LED matrike 12x20 je realiziran sinhronizator sprememb, to je razred *PanelChangesSynchronizer*. Omogoča, da se mu doda objekt vrste tipa *PanelChangesQueue,* kateri lahko nastavimo hitrost izvajanja animacije. Tako lahko kopico sprememb, ki jih zahteva naša animacija, dodamo v obliki objekta *PanelChangesSet* v vrsto in spremembe se bodo ob primernem času izvršile. Bistvena prednost tega sinhronizatorja je torej to, da lahko v vrsto v hipu vnesemo celotno animacijo, četudi se bo ta prikazovala še npr. naslednji 2 sekundi.

Za animiranje RGB osvetlitve predprostora je na voljo animator. Animiranje predprostora z diodami RGB LED je precej drugačna naloga. Diod je precej manj, vendar omogočajo prav toliko možnosti za animiranje, saj jim je mogoče spreminjati barvo. Razred *RGBAnimator*, ki je nekakšna izvrševalna vrsta za animiranje diod RGB LED; nanjo lahko pripnemo večje število objektov tipa *RGBEffect*. Efekti se odzivajo na dva dogodka, dogodek za posodobitev stanja (npr. sprememba barve ob udarcu) ter dogodek za izris stanja (za potrebe primerno hitrega osveževanja).

4.4.3 Animacijski efekti

Aplikacijska plast ponuja nekaj razredov, ki uporabljajo zgoraj opisane izvrševalce animacij, ter ponujajo pomočniška orodja za hitro izdelavo animacij ali pa kar animacije same.

Za animiranje LED matrike 12x20 sta na voljo pomočnik za generiranje tekstovnega napisa in pomočniški razred za ustvarjanje pametnega animiranega objekta.

PanelTextWriter je pomočniški razred, ki kot vhodni podatek sprejme niz in ga zna drseče prikazati na LED matriki 12x20. Možno je tudi nastaviti hitrost animacije. Razred pa niz črko po črko preko transformacijskega polja, pretvori v nize sprememb na LED matriki in jih preda sinhronizatorju za izris.

SmartPanelObject je pomočniški razred, na katerem temelji večino preprostejših animacij. S pomočjo tega razreda je možno hitro izdelati animacije z *narisanimi*

objekti, ki lahko zavzamejo končno število stanj (npr. stolpec, ki ponazori intenziteto spektra). Objekt animiramo podobno kot GIF (*Graphics Interchange Format*) sliko. Na začetku, ko objekt ustvarimo, mu določimo vsa možna stanja, ki jih lahko zavzame. Stanja mu podamo v obliki matrik, ki imajo enako velikost, kot je velikost animiranega objekta – *risbe*. Uporaba takega pametnega objekta pa je zelo preprosta. S pomočjo oštevilčenih stanj menjamo izgled objekta. Na voljo je tudi funkcionalnost, ki med objektoma preklopi z animiranjem tako, da izriše tudi vsa vmesna stanja (v kolikor stanji nista sosednji).

Za animiranje RGB predprostora nizkotonskega zvočnika imamo na voljo nekaj razredov z efekti. Vsem je skupno to, da so zmožni komunikacije z animatorjem (*RGBAnimator*) in da jim je ob začetku, ko objekte ustvarimo, potrebno določiti, katere barve naj uporabijo za animiranje.

RGBSemaphor je razred za generiranje efektov, ki premore več različnih efektov. Vsi efekti zamenjajo svoje stanje ob klicu funkcije *update(),* ki sproži posodobitev stanja. Glede na to, kakšno spremembo povzroči ta funkcija, lahko omenimo nekaj bolj zanimivih efektov:

- Predprostor osvetljujejo različne barve, ki glede na posodobitev stanja krožijo v smeri urinega kazalca.
- Predprostor osvetljuje ena barva, ki se zamenja ob posodobitvi.
- Predprostor osvetljuje ena barva, ki se zamenja ob posodobitvi. Vsako drugo stanje pa je povsem ugasnjena osvetlitev.
- Efekt štetja. Ob posodobitvah se najprej prižge levo krilo, nato samo desno, nato zopet levo in nato oba.

RGBWaterfall je razred, ki predstavlja slap barv. Razredu podamo dve ali več barv, ta pa barve počasi prelije od prve barve k drugi ter potem k tretji itd. V vmesnih stanjih zna sam od sebe generirati vmesne barve. Barve se osvežijo ob klicu funkcije za posodobitev.

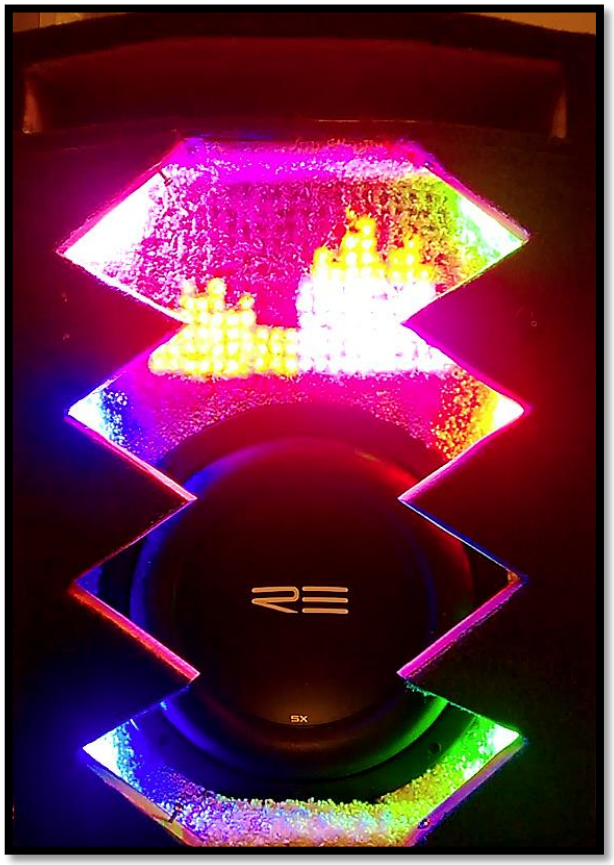
4.5 Vizualizacijske aplikacije

V zadnjih poglavjih so bile podrobno predstavljene vse plasti programa, ki skupaj tvorijo ogrodje. Uporabniki ogrodja pa so vizualizacijske aplikacije, ki delujejo kot preproste aplikacije ali programčki v svojem zaprtem notranjem okolju.

Za kreacijo nove vizualizacijske aplikacije je vse, kar moramo narediti to, da napišemo nov razred, ki razširja bodisi *Application* ali pa *AnimatedApplication* razred ter aplikacijo dodamo na izbiro v izbiralca aplikacij, tako da obstaja način, da jo poženemo.

4.5.1 Pozdravna aplikacija

Pozdravna vizualizacijska aplikacija je zelo preprosta aplikacija. Njen namen je, da predstavi sistem in pozdravi poslušalce. S tem namenom izpiše (z uporabo pomočnika za kreacijo animiranega teksta) pozdravno besedilo na LED matriko 12 x 20. Ko se besedilo izpiše, se aplikacija zapre.



Slika 21: Stolpični vizualizator zvoka v delovanju.

4.5.2 Stolpčni vizualizator zvoka

To je vizualizacijska aplikacija, ki vizualizira spektralno sestavo glasbe in glasbene udarce.

Spektralno sestavo vizualizira na najbolj klasičen način s pomočjo stolpčnega spektrografa. Gre za ažurni prikaz intenzitete vsake od spektralnih skupin, pri čemer je intenziteta vsake skupine ponazorjena z 12 stopenjskim stolpcem. Gledalec tako lahko opazuje sestavo zvoka in jo v mislih poveže s tistim, kar sliši. Za programiranje stolpcev so bili uporabljeni pametni matrični objekti (*SmartPanelObject*), za katere je bilo izdelanih vseh 13. stanj.

Glasbene udarce pa animira RGB osvetlitev predprostora. Sprogramirana je tako, da ob udarcu spremeni barvo osvetlitve predprostora ter jakost postavi na 100%. Takoj zatem hiter časovnik jakost počasi niža do stopnje 30%, kar povzroči nihanje intenzitete v ritmu glasbe. Kombinacija obojega poslušalcu da občutek intenzitete udarcev.

Stolpčni vizualizator zvoka je viden na sliki 21.

4.5.3 Vizualizator z udrihajočimi puščicami

To je vizualizacijska aplikacija, ki se odziva na štiri skupine glasbenih spektrov ter na glasbene udarce. Prikazuje pa tudi nekaj animacij, ki so nepovezane z glasbo.

Skupine spektrov so združene v štiri skupine, pri čemer vsaka skupina animira svoj objekt. Najnižji in najvišji toni so animirani v obliki zvezdice, medtem ko so nižji in višji srednji toni animirani v obliki nasprotno usmerjenih puščic. V animacijo so ob straneh dodani tudi trikotniki, ki potujejo v obratne smeri in dajejo občutek dinamičnosti. Za kreiranje vseh teh objektov je bil uporabljen pametni matrični objekt (*SmartPanelObject*).

Glasbene udarce pa tudi v tem primeru animira RGB osvetlitev predprostora enako kot pri stolpčnem vizualizatorju zvoka.

Delovanje vizualizatorja je vidno na sliki 22.

Slika 22: Vizualizator z več raznovrstnimi animiranimi objekti.

4.5.4 Izbiralec aplikacij

Ta vizualizacijska aplikacija ni namenjena vizualiziranju. Ima posebno nalogo. Ogrodje jo namreč pokliče takrat ko se aktivna aplikacija zapre. Takrat imamo možnost, da izberemo naslednjo aplikacijo.

Aplikacija na matriko izpiše ime vizualizacijske aplikacije, ki je na voljo. Uporabnik s tipko, ki ima funkcijo potrditve izbere aplikacijo ali pa uporabi tipki levo in desno da se sprehodi med ostalimi vizualizacijskimi aplikacijami, ki so na voljo. Ob potrditvi se instancira izbrana aplikacija, vendar se le ta požene šele, ko se aplikacija izbiralec aplikacij zaključi.

Poglavje 5

5 Sklepne ugotovitve

Diplomsko delo nas je popeljalo skozi postopek izdelave nizkotonskega ohišja s svetlobnimi efekti. Pričeli smo z izdelavo samega ohišja in takoj nadaljevali s pripravo vsake od posameznih svetilnih komponent in vgradnje le-teh v samo ohišje. Spoznali smo različne načine za vgradnjo različnih diod LED. Za krmiljenje sedaj že vgrajenih komponent smo potrebovali dodatno vezje, ki smo ga od začetka do konca načrtali in izdelali tudi fizično ter se tako seznanili s celotnim postopkom. Ob tem pa smo spoznali tudi nekaj naprednejših načinov krmiljenja diod LED, na primer multipleksiranje. Vezje smo nato povezali z razvojno ploščico STM32F4-Discovery. Preden smo se lotili pisanja kode, smo dodobra premislili, kako zastaviti arhitekturo programa. Program smo spisali po plasteh od spodaj navzgor. Podrobno so bili opisani vsi problemi, na katere smo naleteli med programiranjem. Med zanimivejšimi se lahko spomnimo razreševanja krmiljenja dodatnega vezja ali pa implementacije vzorčenja in obdelave zvoka. Ta del, praktično uporabo Fouriereve transformacije za potrebe analize spektrov avdio signala, smo tudi teoretično najbolj podprli in tudi najbolj na široko opisali. Na koncu smo zaključili s programiranjem ogrodja in napisali še nekaj animacijskih aplikacij in s tem pokazali hitro razširljivost, ki jo omogoča zastavljena arhitektura.

Namen izdelka, ki je končni rezultat tega uspešno izpeljanega projekta, je zabava poslušalcev in ustvarjanje sproščene atmosfere. Po njihovem mnenju sta bila cilja dosežen. Razširljivost sistema, ki omogoča hiter razvoj novih animacij, pa zagotavlja, da zanimanje za izdelek ne upade.

Čeprav je bil glavni cilj razširljivosti dosežen, pa vseeno obstaja nekaj odprtih vprašanj v zvezi z arhitekturo programa. Za programiranje je bil uporabljen objektni programski jezik C++ z izrazito plastovito strukturo. Uporaba objektno orientiranega programiranja ima za posledico nekoliko večjo porabo pomnilnika, saj je potrebno v pomnilniku hraniti še nekaj dodatnih podatkov, kot so definicije konstruktorjev in destruktorjev. Tudi samo izvajanje je nekoliko počasnejše, saj nekatere naprednejše funkcionalnosti, kot je polimorfizem ali pa dedovanje, od več razredov zahtevajo več re-alokacij pomnilnika v času izvajanja [28]. Osebno

pa vidim prednost pri uporabi objektno orientiranega programiranja v hitrejšem programiranju in lažjem vzdrževanju programa. Kljub temu bi bila uporaba C++ in objektno orientiranega programiranja nespametna odločitev za najnižji rang vgrajenih sistemov z zelo malo pomnilnika. Uporabljena razvojna ploščica STM32F4-Discovery pa premore 192kB pomnilnika. Med programiranjem in testiranjem sem večkrat naletel na zasičenje pomnilnika, kar je povzročilo, da se je izvajanje programa končalo, vendar mi je vsakič uspelo program dovolj optimizirati, da sem zasičenje preprečil.

Alternativno pa bi lahko program realiziral kot skupek opravil, niti in semaforjev v realno-časovnem operacijskem sistemu FreeRTOS [29]. FreeRTOS je operacijski sistem, ki je na voljo tudi za uporabljeno razvojno ploščico STM32F4-Discovery. Deluje kot časovni razvrščevalnik, ki s preklapljanjem omogoča hkratno izvajanje večih opravil v realnem času (odvisno od nastavljenih prioritet niti). Z uporabo FreeRTOS bi se do neke mere znebil potrebe po plastovitem sistemu, saj bi za porazdelitev nalog skrbel FreeRTOS. S tem bi sistem tudi malce pohitril. V kolikor bi za programiranje opravil in niti uporabil programski jezik C, bi sistem še malo pohitril in zmanjšal porabo pomnilnika. Verjetno pa bi nekaj izgubil na preglednosti programske kode.

Literatura

[1] (2013) STM32F4 Discovery. Dostopno na: *http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/PF252419*

[2] (2013) RE SXX Subwoofer Series. Dostopno na: *http://reaudio.com/products/woofers_sxx.php*

[3] (2013) Ground Zero GZTA1.1200DX. Dostopno na: *http://www.ground-zero-audio.com/en/products/amplifiers_/gzta-11200dx.html*

[4] (2013) Medium density fibreboard. Dostopno na: *http://en.wikipedia.org/wiki/Medium-density_fibreboard*

[5] (2013) Light Emitting Diode. Dostopno na: *http://en.wikipedia.org/wiki/Light-emitting_diode*

[6] (2013) STM32F407VG. Dostopno na: *http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN11/PF252140*

[7] (2013) ST-LINK/V2. Dostopno na: *http://www.st.com/web/catalog/tools/FM146/CL1984/SC724/SS1677/PF251168*

[8] (2013) STSW-STM32068. Dostopno na: *http://www.st.com/web/en/catalog/tools/PF257904*

[9] (2013) Multiplexer. Dostopno na: *http://en.wikipedia.org/wiki/Multiplexer*

[10] (2013) Circuitlab. Dostopno na: *https://www.circuitlab.com*

[11] (2013) ROHM BD7851FP-E2. Dostopno na: *http://rohmfs.rohm.com/en/products/databook/datasheet/ic/logic_switch/serial_parallel/bu2050f-e.pdf*

[12] (2013) MOSFET. Dostopno na: *http://en.wikipedia.org/wiki/MOSFET*

[13] (2013) Tranzistor. Dostopno na: *http://sl.wikipedia.org/wiki/Tranzistor*

[14] (2013) I.R. IRFML8244TRPbF. Dostopno na: *http://www.irf.com/product-info/datasheets/data/irfml8244pbf.pdf*

[15] (2013) NXP PSMN9R0-25MLC. Dostopno na: *http://www.nxp.com/documents/data_sheet/PSMN9R0-25MLC.pdf*

[16] (2013) Altium Designer. Dostopno na: *http://www.altium.com/en/products/altium-designer*

[17] (2013) FR-2. Dostopno na: *http://en.wikipedia.org/wiki/FR-2*

[18] (2013) LED Calculator. Dostopno na: *http://led.linear1.org/1led.wiz*

[19] (2013) CPT DC-DC Converter. Dostopno na: *http://www.current-logic.com/dcdc/CLL.pdf*

[20] (2013) Sampling. Dostopno na: *http://en.wikipedia.org/wiki/Sampling_(signal_processing)*

[21] (2013) Nyquist-Shannon. Dostopno na: *http://en.wikipedia.org/wiki/Nyquist–Shannon_sampling_theorem*

[22] (2013) Hearing range. Dostopno na: *http://en.wikipedia.org/wiki/Hearing_range*

[23] (2013) Sampling rate. Dostopno na: *http://en.wikipedia.org/wiki/Sampling_rate*

[24] (2013) DMA. Dostopno na:
http://en.wikipedia.org/wiki/Direct_Memory_Access

[25] (2013) FFT. Dostopno na:
http://en.wikipedia.org/wiki/Fast_Fourier_transform

[26] (2013) Window function. Dostopno na:
http://en.wikipedia.org/wiki/Window_function

[27] (2013) Ooura's Mathematical Software Packages. Kyoto University, Dostopno na:
http://www.kurims.kyoto-u.ac.jp/~ooura/

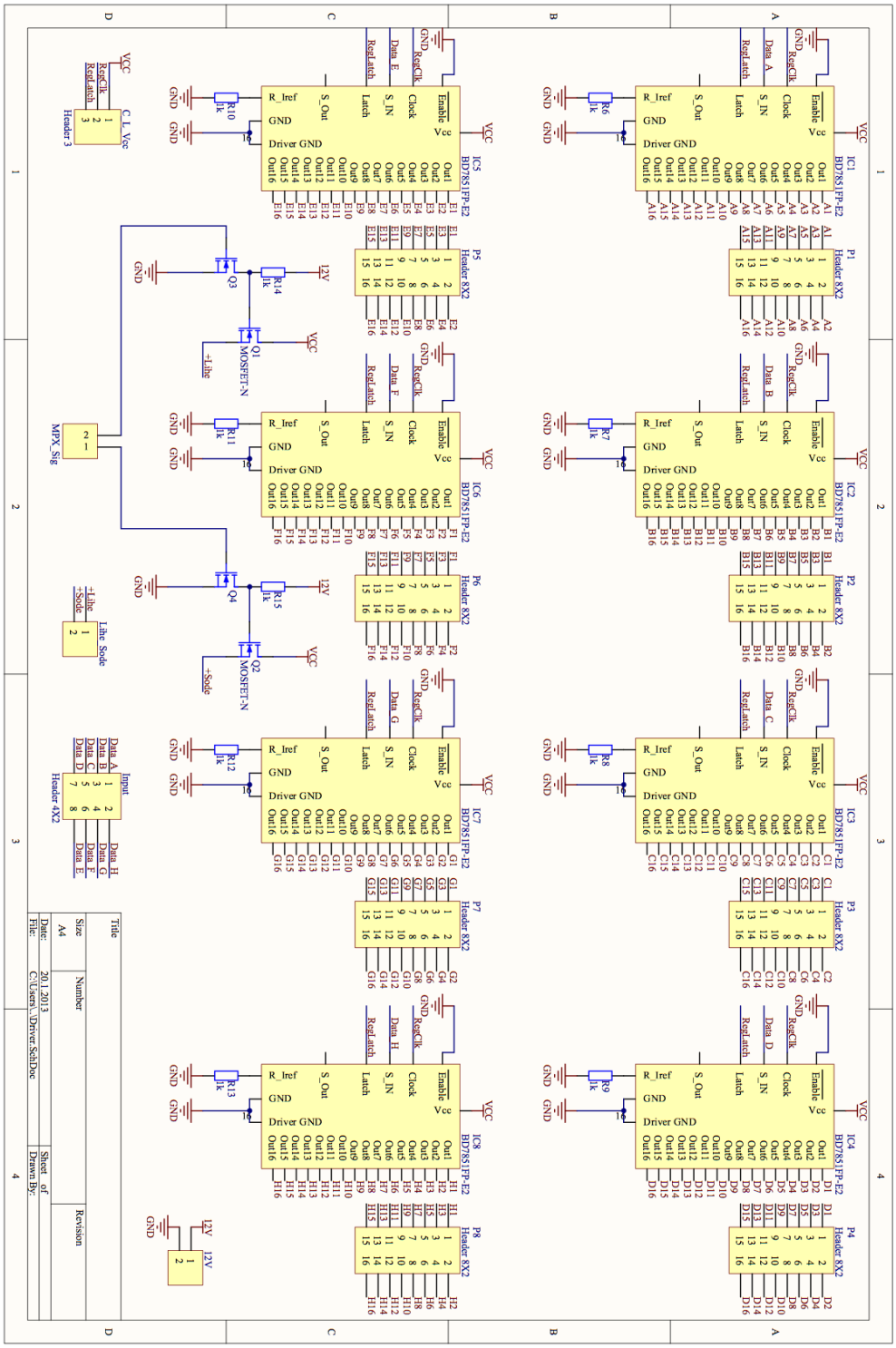
[28] (2013) Technical Report on C++ Performance. Dostopno na:
http://www.open-std.org/jtc1/sc22/wg21/docs/TR18015.pdf

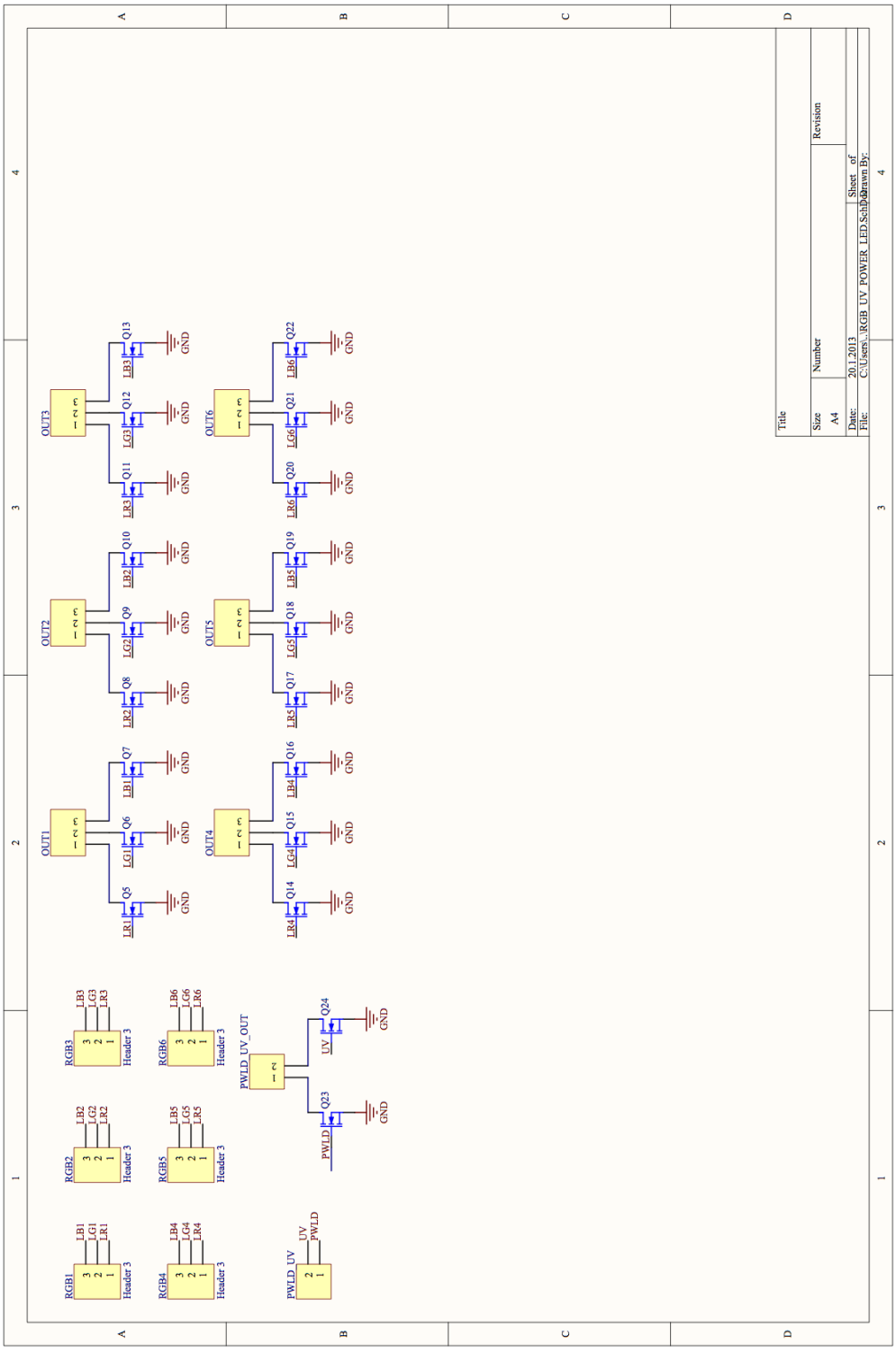
[29] (2013) FreeRTOS. Dostopno na:
http://www.freertos.org

[30] (2013) Total harmonic distortion. Dostopno na:
http://en.wikipedia.org/wiki/Total_harmonic_distortion

Dodatek A: Sheme zunanjega vezja

Sheme zunanjega vezja generirane s programom *Altium Designer* [16].




Title
Size
A4
Number
Revision
Date:
20.1.2013
Sheet of
File:
C:\Users\..\RGB_UV_POWER_LED.SchDoc
Drawn By:
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q8
Q9
Q10
Q11
Q12
Q13
Q14
Q15
Q16
Q17
Q18
Q19
Q20
Q21
Q22
GND
LR1
LG1
LB1
LR2
LG2
LB2
LR3
LG3
LB3
LR4
LG4
LB4
LR5
LG5
LB5
LR6
LG6
LB6
RGB1
RGB2
RGB3
RGB4
RGB5
RGB6
Header 3
PWLD_UV
PWLD_UV_OUT
UV
PWLD
Q23
Q24