FM sintetizator zvuka s ARM32 razvojnim sustavom

Sorić, Robert

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij računarstva

FM sintetizator zvuka s ARM32 razvojnim sustavom

Diplomski rad

Robert Sorić

Osijek, 2022.

SADRŽAJ (Heading 5 stil)

1. Uvod

Digitalni sintetizatori zvuka često su korišteni alati u glazbenoj produkciji i izvođenju glazbe uživo. Korisnici sintetizatora očekuju pregledno korisničko sučelje i mogućnost proizvodnje raznih vrsta zvukova koji su uporabljivi u kontekstu glazbe. Jedna od podvrsta sintetizatora su ritam mašine, elektronički glazbeni instrumenti namijenjeni za imitaciju zvukova bubnja prema korisnički zadanom ritmu [1].

Digitalna FM (engl. *Frequency Modulation*) sintetizacija zvuka popularna je i jednostavna metoda sintetizacije koja se zasniva na postupku modulacije frekvencije.

Tema ovog diplomskog rada je izrada digitalnog FM sintetizatora zvuka zasnovanog na ARM32 računalnoj arhitekturi koji će korisniku omogućiti proizvodnju zvuka vlastito definiranog ritma uz mogućnost promjene zvuka pojedinog bubnja kroz podešavanje pojedinih parametara njihove sintetizacije.

Ostatak rada sastoji se od sljedećih poglavlja: pregled stanja postojećih digitalnih FM sintetizatora, prikaz rješenja FM sintetizatora s ARM32 arhitekturom, testiranje i zaključak. U pregledu stanja navedeni su primjeri postojećih rješenja dostupnih na tržištu. U prikazu rješenja objašnjen je sam pojam sintetizacije, FM sintetizacija, korišteno sklopovlje, shema i izrađena programska podrška. U poglavlju testiranje, biti prikazan je i analiziran rezultat sintetizacije.

1.1. Zadatak diplomskog rada

Zadatak ovog diplomskog rada je izrada FM ritam mašine koja se upravlja putem ulaznog MIDI signala te omogućuje korisniku pristup promjeni parametara sintetizacije pojedinih bubnjeva. Također, bilo je potrebno pobliže prikazati samu FM sintetizaciju i njenu metodologiju.

2. Pregled stanja postojećih digitalih FM sintetizatora

FM sintetizatori dolaze u raznim oblicima uz različite značajke i mogućnosti primjene. Osnovno ih se može podijeliti na sintetizatore ostvarene kroz programsku podršku (engl. *software synthesizer*) i sintetizatore zasnovane na sklopovlju (engl. *hardware synthesizer*). Sintetizatori u programskoj podršci imaju prednost u pristupačnosti korištenja i manjoj cijeni. Sintetizatori zasnovani na sklopovlju korisniku pružaju veći stupanj kontrole nad zvukom i za njih nije potrebno trošiti procesorske resurse za proizvodnju zvuka.

U nastavku su prikazana tri popularna FM sintetizatora zasnovanih na sklopovlju.

2.1. Korg Volca FM

Volca linija proizvoda Japanskog proizvođača Korg sastoji se od niza budžetnih elektroničkih instrumenata koji su namijenjeni početnicima u domeni sintetizacije. Korg Volca FM (slika 2.1.), je instanca Volca serije koja koristi FM metodu sinteze zvuka. Podržava rad do 6 FM operatora uz 32 moguća algoritma. Može se programirati ručno ili putem osobnog računala, što znatno olakšava proces podešavanja parametara. Također, podržava jednostavan oblik sekvenciranja te ritmičnu sinkronizaciju uz MIDI takt [2]. Prodaje se po cijeni od 170 EUR.

Sl. 2.1. Korg Volca FM digitalni sintetizator zvuka.

2.2. Elektron Model:Cycles

Digitalni FM sintetizator Model:Cycles (slika 2.2.) koji proizvodi Švedska tvrtka Elektron predstavlja moderno rješenje FM sintetizatora široke primjene. Ima mogućnost proizvodnje zvukova različitih bubnjeva i melodijskih tonova koristeći jedan od 6 algoritama, svaki od kojih pruža 4 parametra koji utječu na FM sintezu, uz 6 dodatnih općih parametara. Korisniku je

omogućen unos željenog ritma ili melodije putem detaljnog sekvenciranja sa maksimalnom mogućom duljinom sekvence od 128 koraka i sinkronizacijom putem MIDI takta. LCD zaslon sa 128x64 piksela prikazuje informacije o vrijednosti parametra koji se trenutno podešava, trenutnom koraku u sekvenci, izboru FM algoritma i ostalom [3]. Prodaje se po cijeni od 330 EUR.

Sl. 2.2. Elektron Model:Cycles digitalni FM sintetizator.

2.3. Yamaha Montage 8

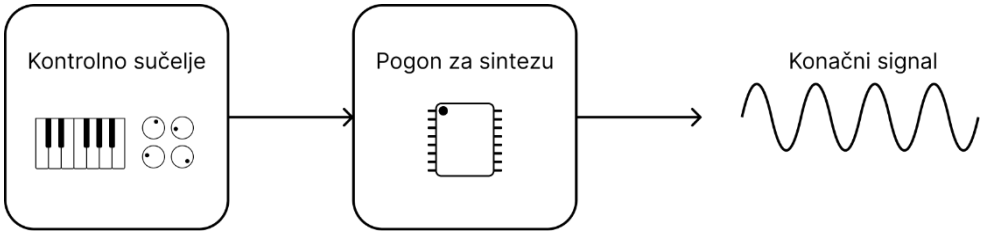
Montage 8 je FM sintetizator Japanskog proizvođača Yamaha (slika 2.3.). U konvencionalnijem je obliku, sa digitalnim klavirom, velikim brojem tipkala i dodirnim zaslonom. Za FM sintezu sadržava 8 operatora, mogućnost sviranja 128 nota odjednom te podržava 88 različitih algoritama. Namijenjen je kao cjelokupna radna stanica za sintezu zvuka [4]. Prodaje se po cijeni od 4000 EUR.

Sl. 2.3. Yamaha Montage 8 radna stanica za sintetizaciju.

3. FM Sintetizator s ARM 32 razvojnim sustavom

3.1. Sinteza zvuka

Sinteza zvuka je proces proizvodnje audio signala koristeći elektroničke komponente [1]. Sintetizatori su elektronički uređaji namijenjeni za sintezu zvuka, najčešće za potrebu produkcije i izvođenja glazbe. Sadržavaju dvije osnovne funkcionalne cjeline: kontrolno sučelje i pogon za sintezu (slika 3.1.).



Sl. 3.1. Sinteza zvuka.

U kontrolnom sučelju korisnici određuju parametre sinteze, primjerice, zadani ton, izbor zvuka i njegovih karakteristika. Kontrolno sučelje čini sklopovlje za unos podataka, primjerice, klavijatura, tipkala i potenciometri.

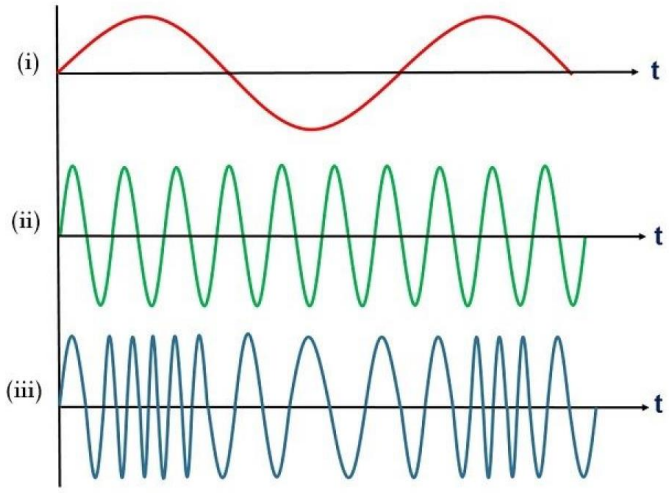
Pogon za sintezu interpretira korisnički unos i proizvodi konačni zvučni signal. Može biti zasnovan na analognoj ili digitalnoj bazi.

3.2. FM sinteza zvuka

FM sinteza je digitalna metoda sinteze zvuka zasnovana na procesu frekvencijske modulacije [6]. U njoj sudjeluju dva signala: nositelj i modulacijski signal. Modulacija signala se odvija tako da iznos trenutne amplitude modulacijskog signala utječe na trenutnu frekvenciju signala nositelja, koji onda postaje konačni signal. Postupak FM modulacije prikazan je u jednadžbi 3-1.

$$FM(t) = A \times sin(\omega_c t + \beta \times sin(\omega_m t)) \tag{3-1}$$

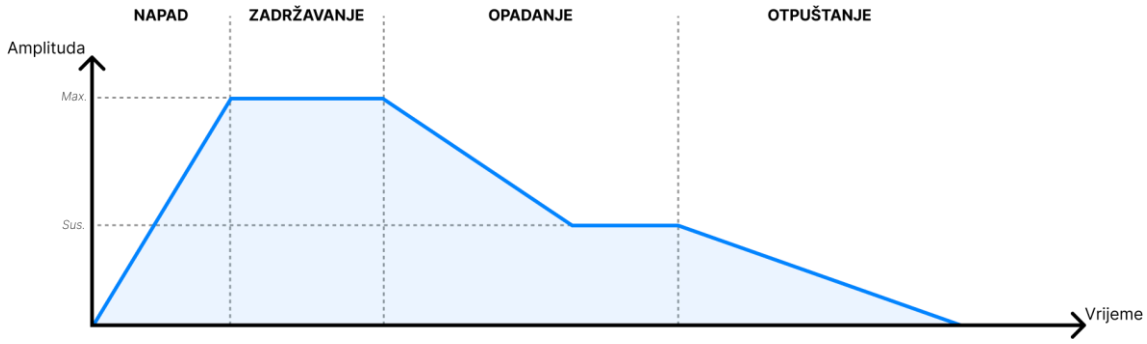
Na signal nositelj (frekvencije $\omega_c t$ u jednadžbi 3-1) djeluje modulacijski signal ($\beta \times \omega_m t$ u jednadžbi 3-1) tako što mu množi frekvenciju uz faktor modulacije β. Vizualizacija amplituda pri frekvencijskoj modulaciji vidljiva je na slici 3.2. uz prikaz modulacijskog signala (i), signala nositelja (ii) i konačnog signala (iii).



Sl. 3.2. Vizualizacija FM modulacije.

3.2.1. Zvučna omotnica

Postupno povećavanje te smanjivanje glasnoće pri sviranju note se u sintezi zvuka opisuje sa zvučnom omotnicom (slika 3.3.). Ona je korisnički definirana i opisuje promjenu amplitude signala nakon trenutka sviranja pojedine note u sintetizatoru. U standardnom obliku sadržava 4 elementa: napad (engl. *attack*), zadržavanje (engl. *hold*), opadanje (engl. *decay*) i otpuštanje (engl. *release*) [5].



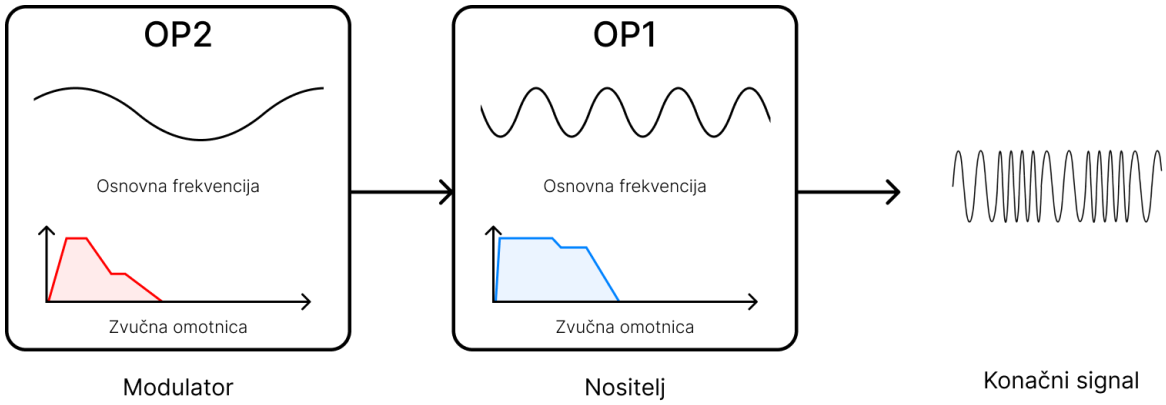
Sl. 3.3. Zvučna omotnica

Vrijeme napada opisuje koliko dugo omotnici treba da dosegne svoju maksimalnu amplitudu (na slici 3.3. označenoj sa *max*). Povećavanje vremena napada se primjenjuje kod zvukova koji polako rastu u glasnoći, primjerice, u sintezi gudačkih instrumenata ili ambijentalnih zvukova. Naglo ili trenutačno vrijeme napada se pripisuje zvukovima trzalačkih glazbala ili bubnjeva. Zadržavanje je faza u kojoj omotnica zadanu količinu vremena zadržava maksimalnu vrijednost amplitude. U nekim slučajevima poželjno je zadržati maksimalnu vrijednost cijelo vrijeme dok korisnik ne otpusti tipku, dok je u drugima poželjno maksimalnu vrijednost zadržati samo u kratkom trenutku.

Faza opadanja započinje nakon što je vrijeme zadržavanja prošlo. U njoj se amplituda omotnice stišava do razine održavanja (na slici 3.3. označenoj sa *sus*). Otpuštanje nastupa nakon što korisnik otpusti sviranu notu te predstavlja spuštanje amplitude od razine održavanja do nule. Povećanje ove vrijednosti može služiti oponašanju instrumenata sa pedalom za održavanje, kao što je klavir.

3.2.2. Operator

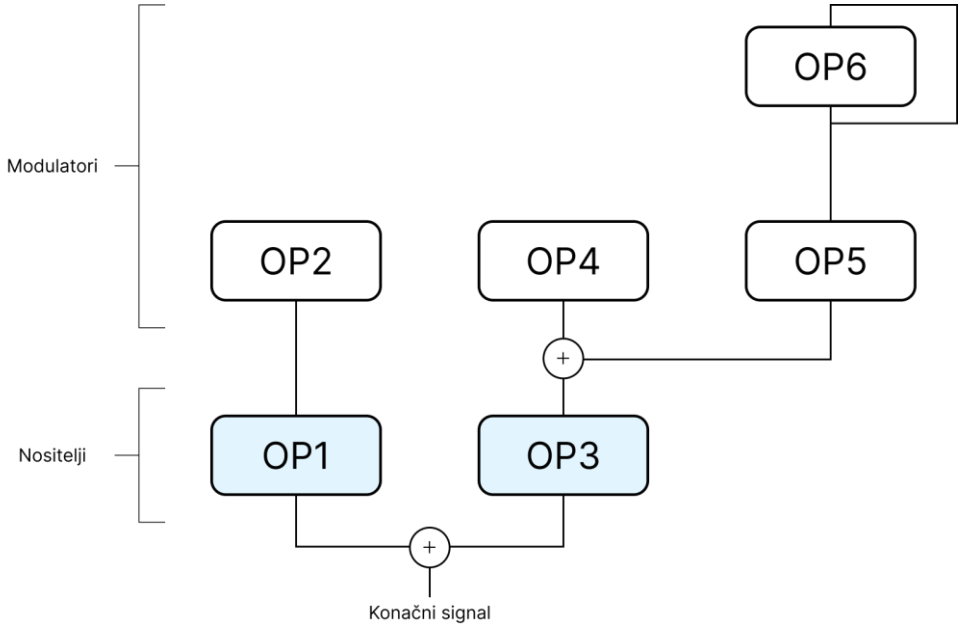
U FM sintezi zvuka, signal može biti nositelj ili modulator te se uz svoju zvučnu omotnicu kao funkcionalna cjelina naziva operator [7]. FM algoritam predstavlja cjelokupni tok signala kroz operatore i način na koji pojedini operatori vrše modulaciju. Osnovni FM algoritam (slika 3.4.) tvori se od dva operatora, jedan od kojih je nositelj (označen sa OP1 na slici 3.4.) nad kojim frekvencijsku modulaciju izvršava modulator (označen sa OP2 na slici 3.4.).



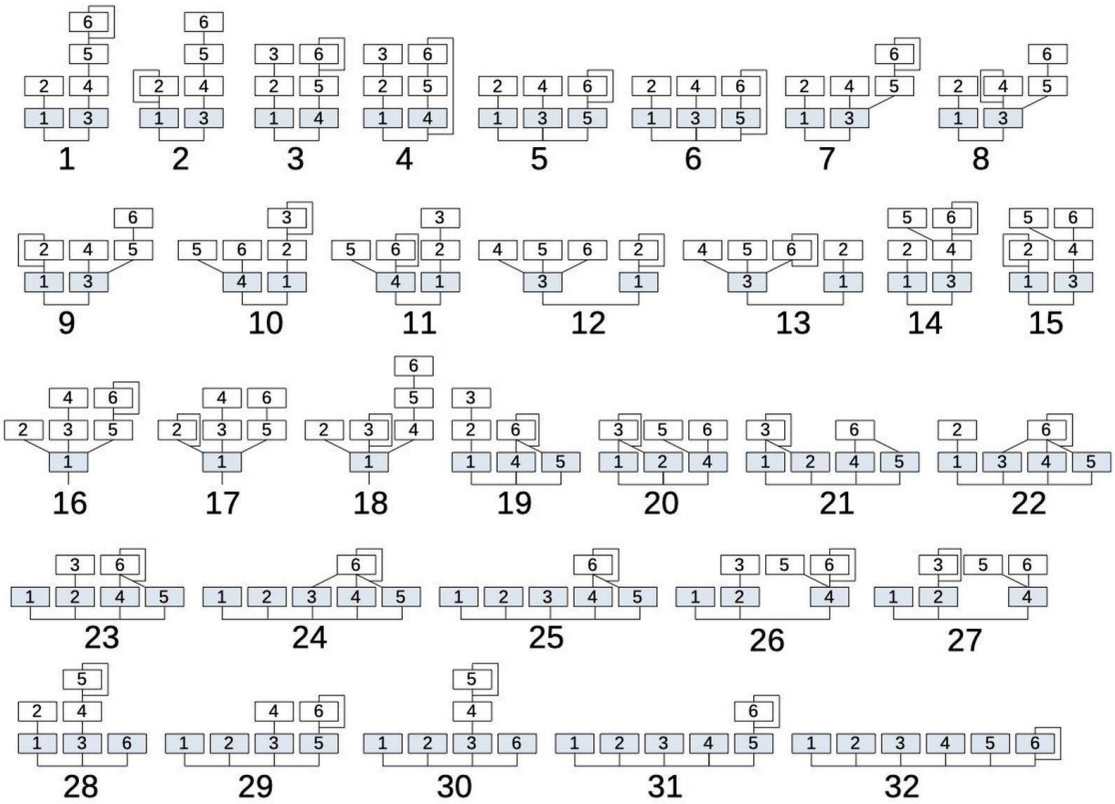
Sl. 3.4. Osnovni FM algoritam.

FM algoritmi u digitalnim sintetizatorima koriste veći broj operatora te je izbor FM algoritma jedan od osnovnih parametara koje korisnik može ugađati pri programiranju FM sintetizatora. Primjer složenijeg FM algoritma (slika 3.5.) sastoji se od 6 operatora. Operatori 1 i 3 su signali nositelji. Operator 2 vrši frekvencijsku modulaciju operatora 1. Operator 6 vrši frekvencijsku modulaciju nad samim sobom (*samo-modulacija*) te rezultat te modulacije frekvencijski modulira operator 5. Ovdje se javlja algebarsko zbrajanje signala iz izlaza operatora 4 i 5. Suma ovih signala u konačnici modulira operator 3.

U digitalnim FM sintetizatorima su ovi algoritmi pojednostavljeno prikazani radi bolje preglednosti. Zbog memorijskih ograničenja ranih FM sintetizatora izabrano je 32 algoritma (slika 3.6.) koji se mogu pronaći u većini digitalnih FM sintetizatora i danas zbog mogućnosti proizvodnje širokog spektra zvuka.



Sl. 3.5. Primjer složenog algoritma FM sinteze.



Sl. 3.6. 32 često korištena algoritma FM sinteze.

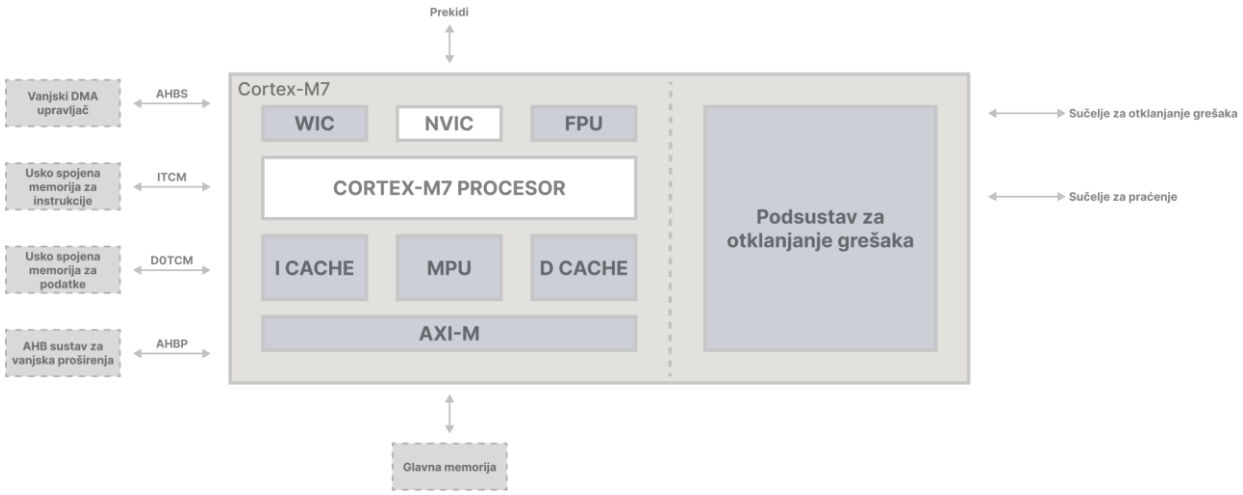
3.3. Sklopovlje

Konačno rješenje diplomskog rada sadržava Teensy 4.1 mikroupravljač zasnovan na 32 bitnoj ARM arhitekturi, Teensy Audio Adapter, LCD zaslon te 6 rotirajućih enkodera za korisnički unos.

3.3.1. ARM

ARM (engl. *Advanced RISC Machines*) je arhitektura računalnih procesora dizajnirana od strane tvrtke Acorn Computers. Karakterizira ih mala potrošnja energije i ograničen set procesorskih instrukcija RISC (engl. *Reduced Instruction Set Computer*). Najzastupljenija je procesorska arhitektura na svijetu. ARM arhitekturu primjenjuje širok raspon mikroprocesora različitih zahtjeva performansi i potrošnje energije. Pronalaze se u senzorima, digitalnim kamerama, pametnim telefonima, superračunalima i brojnim drugim elektroničkim uređajima [8].

Specifična instanca ARM arhitekture koja je korištena u izradi diplomskog rada je Cortex-M7 (slika 3.7.). U odnosu na ostale procesore u Cortex-M seriji opisuju ga najbolje DSC performanse (engl. *Digital Signal Controller*). Ukupno sadržava 26 registara, 13 od kojih su registri opće primjene. Tok izvođenja za instrukcije se odvija u 6 faza te može izvršiti do dvije instrukcije po jednom taktu [9].



Sl. 3.7. Arhitektura Cortex-M7 procesora

3.3.2. Teensy 4.1

Teensy je serija razvojnih pločica koju dizajnira i proizvodi Američka tvrtka PJRC. Teensy 4.1 (slika 3.8.) sadržava ARM Cortex-M7 procesor koji radi pri frekvenciji od 600 MHz, 55 digitalnih ulaz/izlaz nožica te 18 analognih ulaza te se programira u C++ programskom jeziku. Sadržava 7936K Flash memorije, 1024K RAM memorije i 4K EEPROM memorije. Podržava serijsku

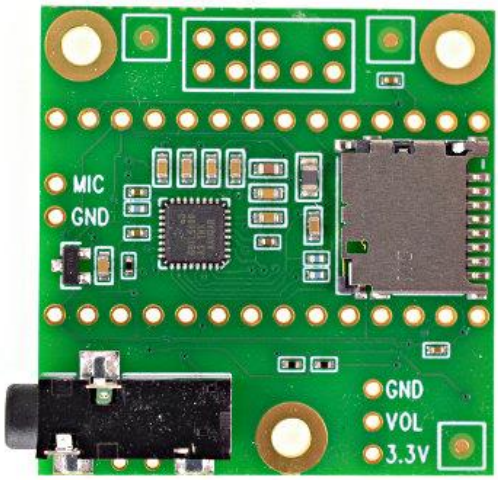
komunikaciju i programiranje putem USB sučelja uz brojne druge hardverske i softverske značajke [10].

Glavna prednost Teensy mikroupravljača je u podržavanju programske i biblioteka namijenjenih za Arduino mikroupravljače uz značajno bolje performanse i širi raspon mogućnosti.

Sl. 3.8. Teensy 4.1. mikroupravljač.

3.3.3. Teensy Audio Adapter

Teensy Audio Adapter (slika 3.9.) je sklopovlje koje je namijenjeno kao proširenje za Teensy mikroupravljače, omogućavajući im proizvodnju i izlaz audio signala.



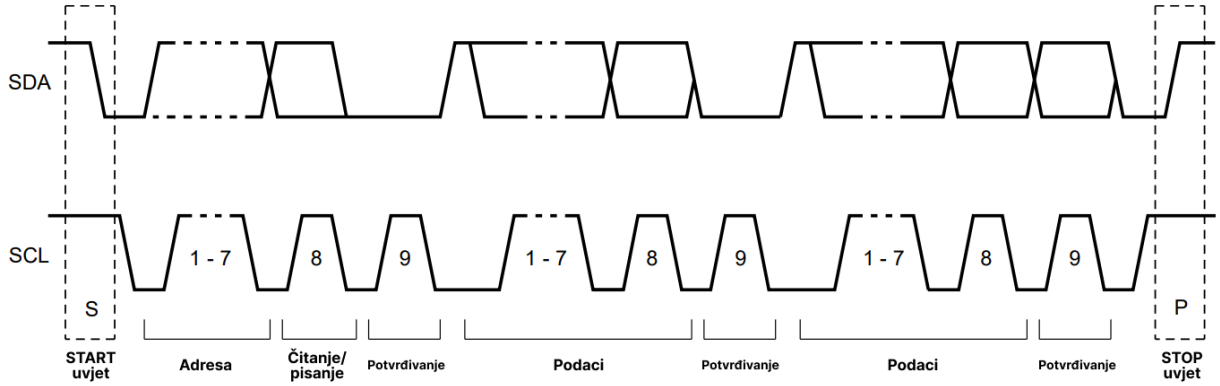
Sl. 3.9. Teensy Audio Adapter.

Omogućuje stereo audio izlaz po rezoluciji uzorkovanja od 16 bita i frekvenciji uzorkovanja od 44.1 kHz. Zasniva se na integriranom krugu SGTL5000, koji se na Teensy spaja preko 7 signala, 3 od kojih su clock signali: LRCLK po 44.1 kHz, BCLK po 1.41 MHz te MCLK po 11.28 MHz. Ostali signali služe za konfiguraciju, koja se odvija putem I2C komunikacije i prijenos željenog izlaznog audio signala, koji se šalje putem I2S komunikacije [11].

3.3.4. I2C i I2S komunikacija

I2C (engl. *Inter-Integrated Circuit*) je standard dvosmjernog serijskog prijenosa podataka između integriranih krugova koju je razvio Nizozemski proizvođač Philips. Omogućuje povezivanje do 127 uređaja koristeći dvije žice: Serial Data (SDA), koja sadržava podatke i Serial Clock (SCA), koja služi za sinkronizaciju. Standard I2C komunikacije nalaže kako je jedan od članova spojenih na I2C sabirnicu upravljač (engl. *controller*) a ostali su sljedbenici (engl. *follower*). I2C komunikacija je često primijenjena u konfiguraciji i upravljanju integriranih krugova putem čitanja i pisanja u njihove memorijske registre [12].

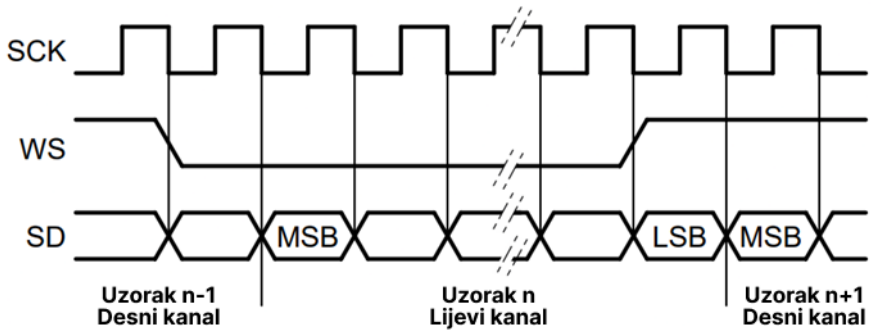
Prvih 7 bita svake instance I2C komunikacije sadržava adresu I2C sljedbenika s kojim upravljač želi imati interakciju (slika 3.10.). Ove adrese su često nepromjenjive i definiraju se pri proizvodnji mikroupravljača s kojim želimo upravljati. Sljedeći bit opisuje želi li voditelj čitati (1) ili pisati (0). Nadalje, šalje se bit za potvrđivanje (engl. *acknowledge*), indikator poslan od strane sljedbenika da je uspješno primio prijašnji slijed bitova. Greška u slanju bita za potvrđivanje se može dogoditi ako sljedbenik radi neku drugu operaciju, ne razumije primljeni podatak ili nije spojen na I2C sabirnicu. Sljedećih 8 bitova u komunikaciji sadržavaju adresu na sljedbeniku s kojom upravljač želi imati interakciju. Nakon poslane adrese, ovisno o tome piše li se ili čita u registar, voditelj ili sljedbenik će poslati željene podatke [13].



Sl. 3.10. Prijenos podataka putem I2C standarda.

I2S (engl. *Inter-Integrated Circuit Sound*) sabirnica je također stvorena u Philipsu i služi za prijenos PCM (engl. *Pulse-Code modulation*) digitalnog audio signala između integriranih krugova [14]. Podržava simultano slanje 2 digitalna audio signala za stereo zvuk (slika 3.11.).

I2S sabirnica se sastoji od 3 žice: BCLK (taktni signal), WS (engl. *Word Select*, izbor lijevog ili desnog kanala) i SD (engl. *Seiral Data*, serijski podatci). Frekvencija takta BCLK se računa ovisno o željenoj frekvenciji uzorkovanja, broju bita po kanalu i broju kanala. Zvučni signal uzorkovan po 44.1 kHz sa 16 bita preciznosti i dva kanala za stereo zvuk ima BCLK frekvenciju od 1.4112 MHz. WS je digitalni impuls iste frekvencije kao frekvencija uzorkovanja. Dok je u logičkoj nuli, I2S standard deklarira da se trenutno šalje lijevi kanal te dok je u jedinici šalje se desni. Pojedini uzorak audio signala se onda putem SD sabirnice šalje sa više značajnim bitom prvim.



Sl. 3.11. Prijenos digitalnog zvučnog signala putem I2S sučelja.

3.3.5. Rotacijski enkoderi

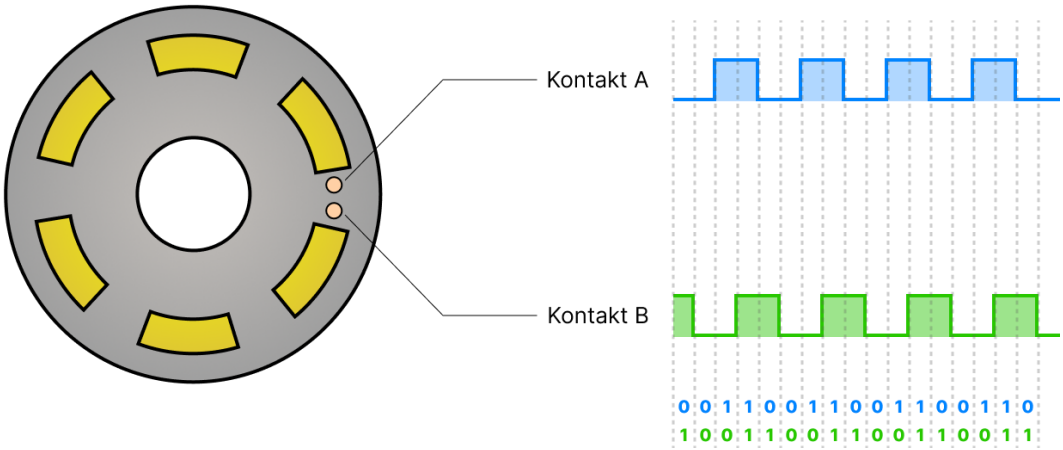
Rotacijski enkoderi (slika 3.12.) su senzori koji mogu odrediti rotacijski položaj osovine, čineći ih korisnima za korisnički unos [15]. Za razliku od potenciometra, mogu se rotirati beskonačno u bilo kojem smjeru te ne ovise o analognom mjerenju vrijednosti napona.

Dva elektronička kontakta koja se nalaze na statičnom dijelu enkodera se izmjenično kratko spajaju sa diskom koji se rotira zajedno sa osovinom s kojom korisnik ima interakciju. Ovo izmjenično kratko spajanje i odvajanje kontakta generira dva impulsa iz kojih je moguće kontinuirano očitavati u kojem smjeru i koliko se enkoder rotirao (slika 3.13.). Spajaju se na mikroupravljač putem nožica za digitalni ulaz/izlaz.

U radu, 5 rotacijskih enkodera se koristi za podešavanje jednog od parametara FM sinteze za pojedini bubanj dok se posljednji enkoder u nizu koristi kao izbornik te služi za promjenu bubnja čiji se parametri mijenjaju. Također, posljednji enkoder sadržava i funkcionalnost standardnog tipkala te dok je taj enkoder pritisnut, utjecaj pomaka enkodera za svaki pojedini parametar biti će

povećan. Ovo je namijenjeno za bolje korisničko iskustvo pri unosu jer znatno ubrzava vrijeme koje je potrebno da korisnik drastično promijeni vrijednost pojedinog parametra.

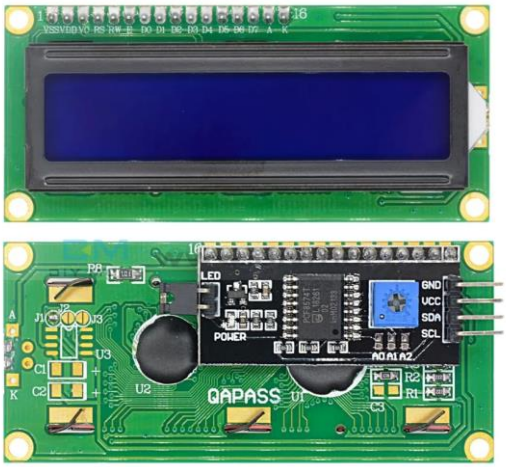
Sl. 3.12. Rotacijski enkoder.



Sl. 3.13. Rad rotacijskog enkodera.

3.3.6. LCD modul

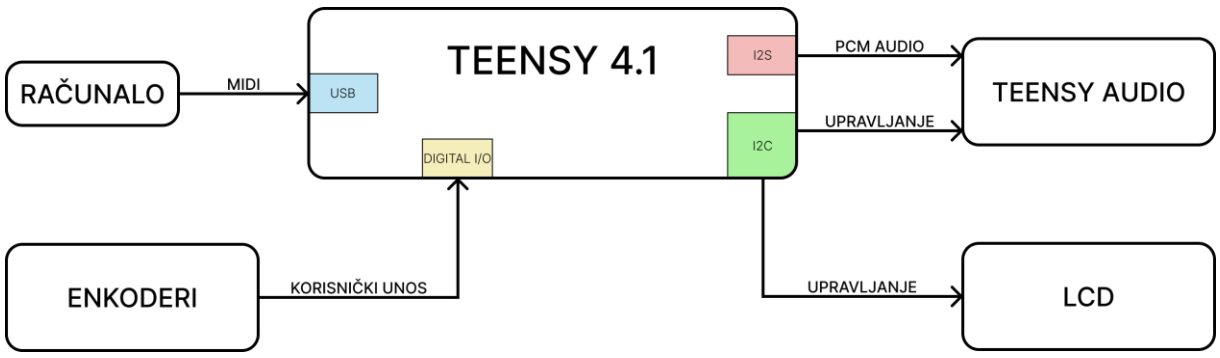
Programibilni LCD (engl. *Liquid Crystal Display*) zaslon sa mogućim prikazom 16x2 slova (slika 3.14.), korišten je za prikaz informacija korisniku o trenutno selektiranom zvuku, zadnje promijenjenom parametru i njegovoj vrijednosti. Zbog I2C modula moguće ga je programirati putem samo dvije žice koristeći I2C protokol.



Sl. 3.14. 16x2 LCD zaslon sa I2C modulom.

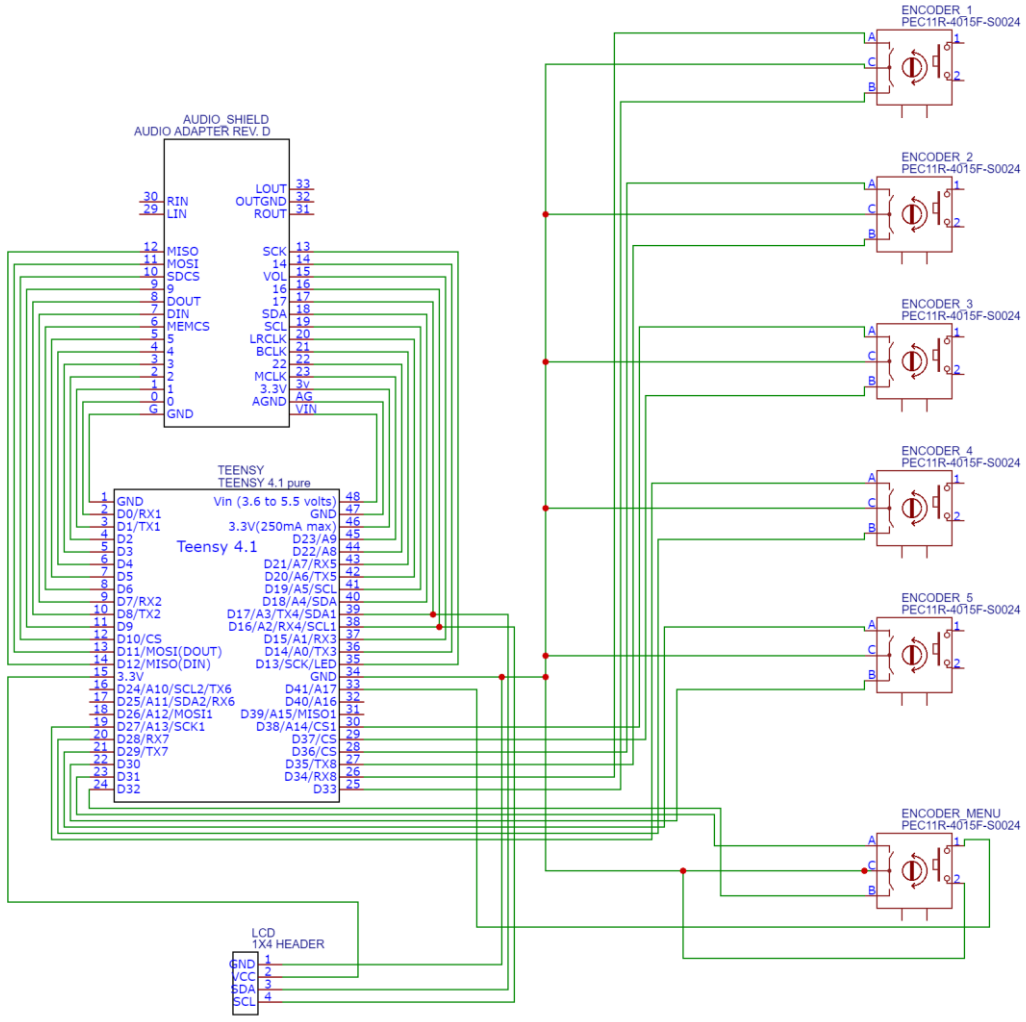
3.3.7. Konačna shema

U idejnoj shemi uređaja (slika 3.15.) sudjeluju sve prije spomenute komponente. Teensy mikroupravljač proizvodi konačni audio izlaz koji se pretvara u analogni signal koristeći Audio Adapter. Postavke Audio Adaptera koje će se mijenjati tijekom korisničkog rada se šalju putem I2C komunikacijskog protokola. LCD se upravlja koristeći I2C1, sekundarno I2C sučelje na Teensy mikroupravljaču. Enkoderi su spojeni putem nožica za digitalni ulaz/izlaz te se putem USB komunikacije Teensy pojavljuje kao MIDI kompatibilan uređaj koji može primati podatak koje note treba svirati.

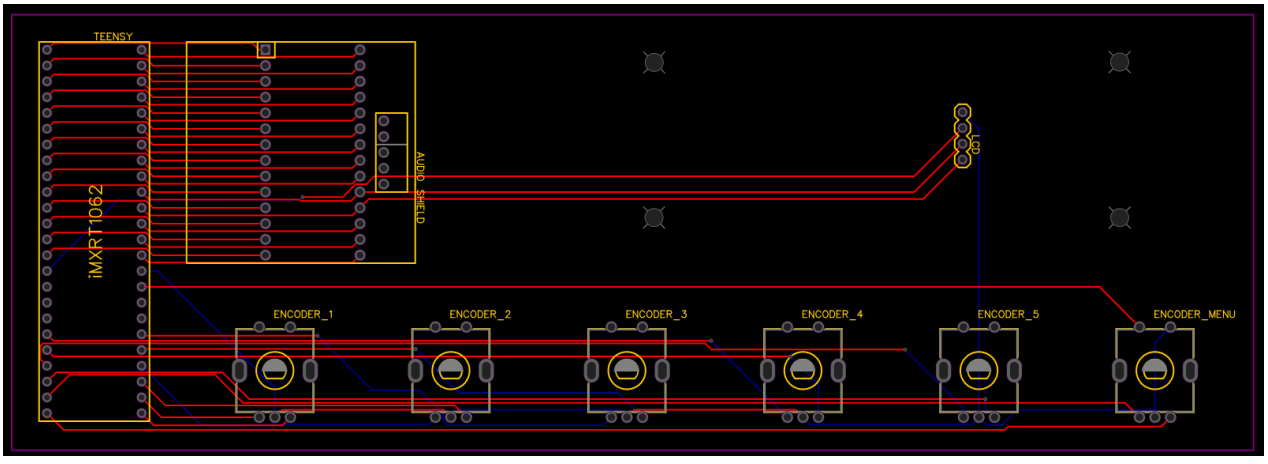


Sl. 3.15. Idejna shema rješenja ritam mašine

Detaljna shema cijelog uređaja (slika 3.16.) prikazuje spojene nožice između mikroupravljača i vanjskih komponenti te je korištena za izradu tiskane ploče (slika 3.17.) unutar programske podrške EasyEDA proizvođača JLCPB.



Sl. 3.16. Detaljna shema uređaja.



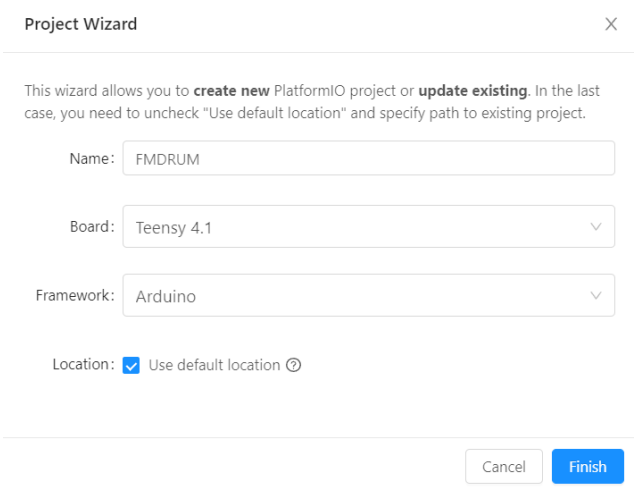
Sl. 3.17. Shema za PCB ploče.

3.4. Programska podrška

Programska podrška stvorena je u razvojnom okruženju PlatformIO, zasnovanom na programu Visual Studio Code [16]. Omogućuje napredan razvoj softvera za mikroupravljače uz sve korisne značajke i dodatke koji već postoje za Visual Studio Code.

3.4.1. Kreiranje i konfiguracija PlatformIO projekta

Pri kreiranju projekta u PlatofrmIO okruženju, potrebno je navesti naziv projekta i mikroupravljač za koji se softver razvija (slika 3.18.). Nadalje, nakon kreiranja projekta, kako bi bilo moguće upravljati sa perifernim uređajima potrebno je instalirati odgovarajuće biblioteke programske podrške. Ovom instalacijom, te se biblioteke dodaju u projekt i mogu se koristiti u daljnjem pisanju kôda. Putem PlatformIO izbornika, instalirano je 5 biblioteka (tablica 3.1.).



Sl. 3.18. Kreiranje PlatformIO projekta.

Tablica 3.1. Instalirane biblioteke.

	Naziv biblioteke	*Opis*
1	LiquidCrystal_I2C	Upravljanje sa LCD zaslonom putem I2C komunikacije
2	RotaryEncoder	Očitavanje položaja rotacijskih enkodera
3	RBD_Button	Upravljanje s tipkalom na enkoderu za izbornik
4	TeensyThreads	Omogućuje višenitno izvođenje kôda

Daljnja konfiguracija projekta (kôd 3.19.) zahtjeva uređivanje konfiguracijske datoteke (*Platformio.ini*). Potrebno je dodati zastavicu koja će pri kompilaciji koda uključiti način rada Teensy mikroupravljača kao MIDI uređaj [17], što u glavnoj datoteci (*main.cpp*) omogućuje pristup midiUSB objektu koji sadržava potrebnu funkcionalnost za rad sa MIDI porukama [18].

```
[env:teensy41]
platform = teensy
board = teensy41
framework = arduino
build_flags = -D USB_MIDI -D TEENSY_OPT_FASTEST
lib_deps =
    marcoschwartz/LiquidCrystal_I2C@^1.1.4
    mathertel/RotaryEncoder@^1.5.3
    ftrias/TeensyThreads@^1.0.2
    alextaujenis/RBD_Button@^2.2.1
```

Kôd. 3.19. Konfiguracija PlatformIO projekta.

3.4.2. Čitanje ulaznih MIDI poruka

U slučaju ritam mašine, potrebno je simultano provjeravati korisničke unose na rotacijskim enkoderima kao i čitati ulazne MIDI poruke. Ovo je ostvareno putem Teensythreads biblioteke. Ona je specifično dizajnirana za Teensy mikroupravljače te omogućuje izvođenje više procesorskih niti (engl. *thread*) simultano na istom uređaju [19].

Čitanje ulaznih MIDI signala koristeći usbMIDI objekt je prvo potrebno smjestiti u beskonačnu petlju te onda u zasebnu funkciju. Ta funkcija se onda može pokrenuti kao nit izvođenja u *setup()* funkciji (kôd 3.20.).

```
#include "TeensyThreads.h"
void thread_MIDIREAD(){
  while (1){
    usbMIDI.read();
  }
}
void setup(){
    //. . .
    threads.addThread(thread_MIDIREAD);
}
```

Kôd. 3.20. Stvaranje i inicijalizacija niti za čitanje s usbMIDI ulaza.

3.4.3. Očitavanje položaja rotacijskih enkodera

Arduino biblioteka *RotaryEncoder* omogućuje rad sa raznim vrstama rotacijskih enkodera [20]. U njenom korištenju prvo je potrebno izraditi *RotaryEncoder* objekt na pripadajućim nožicama digitalnih ulaza te odrediti parametar *LatchMode*, koji opisuje unutarnju građu enkodera kako bi se mogao pravilno dekodirati. Kreirane su i varijable koje pamte posljednje izmjerene položaje svakog enkodera. Svi objekti enkodera i njihovi posljednje izmjereni položaji smješteni su u nizove kako bi se njima moglo pristupiti putem petlje koristeći indeksiranje (kôd 3.21.). Očitavanje stanja enkodera se onda obavlja *polling* metodom pomoću vremenskog brojača, provjeravajući stanje svakih pojedinog enkodera nekoliko milisekundi unutar glavne petlje (kôd 3.22.).

```
#include <RotaryEncoder.h>
static RotaryEncoder encoder1(33, 34, RotaryEncoder::LatchMode::TWO03);
static RotaryEncoder encoder2(35, 36, RotaryEncoder::LatchMode::TWO03);
static RotaryEncoder encoder3(37, 38, RotaryEncoder::LatchMode::TWO03);
static RotaryEncoder encoder4(28, 27, RotaryEncoder::LatchMode::TWO03);
static RotaryEncoder encoder5(30, 29, RotaryEncoder::LatchMode::TWO03);
static RotaryEncoder encoderMenu(31, 32, RotaryEncoder::LatchMode::FOUR3);
static RotaryEncoder encoders[5] = {encoder1, encoder2, encoder3, encoder4, encoder5};
static int encoder1pos = 0;
static int encoder2pos = 0;
static int encoder3pos = 0;
static int encoder4pos = 0;
static int encoder5pos = 0;
static int encoderMenupos = 0;
static int encoderPositions[5] = {encoder1pos, encoder2pos, encoder3pos, encoder4pos, encoder5pos};
```

Kôd 3.21. Kreiranje objekata i varijabli za rad s rotacijskim enkoderima.

```
#include <elapsedMillis.h>
static int indexOfEncoderToPoll;
static elapsedMillis encoderPollTimer;
static unsigned int encoderPollDelay;
void setup(){
  indexOfEncoderToPoll = 0;
  encoderPollDelay = 1;
  encoderPollTimer = 0;
  //. . .
}
void loop(){
  if (encoderPollTimer > encoderPollDelay){
    encoderPollTimer = 0;
    pollEncoders(indexOfEncoderToPoll);
    pollMenuEncoder();
    indexOfEncoderToPoll++;
    if (indexOfEncoderToPoll == 5){
      indexOfEncoderToPoll = 0;
    }
    //. . .
  }
}
```

Kôd 3.22. Očitavanje stanja rotacijskih enkodera u glavnoj petlji.

3.4.4. Očitavanje stanja tipkala

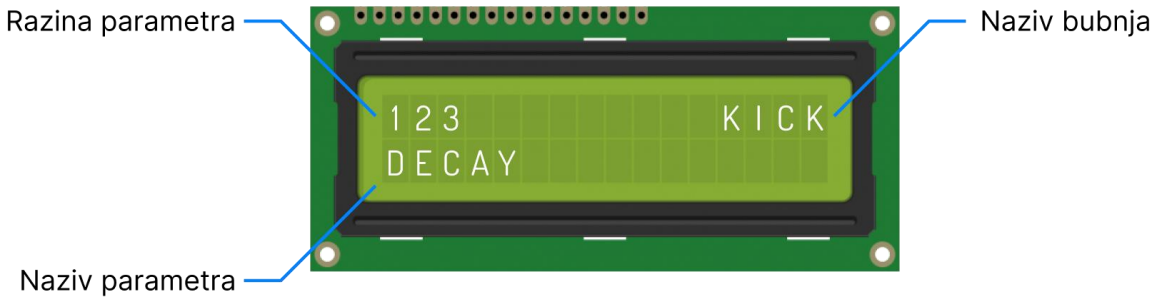
Šesti rotacijski enkoder u nizu služi kao izbornik za selekciju koji pojedinačni zvuk bubnja korisnik trenutno želi modificirati. Radi poboljšanog korisničkog iskustva pri modifikaciji parametara upotrijebiti će se tipkalo koje se nalazi na tom enkoderu koristeći Arduino biblioteku *RBD_Button* [21]. Pri korištenju te biblioteke, potrebno je kreirati objekt na nožici digitalnog ulaza Teensy mikroupravljača na koju je tipkalo spojeno. Analogno rotacijskim enkoderima, provjera stanja tipkala pamtiti će se u lokalnoj varijabli tipa te će se u glavnoj petlji *polling* metodom provjeravati trenutno stanje (kôd 3.23.).

```
1   #include <RBD_Timer.h>
2   #include <RBD_Button.h>
3   static RBD::Button menuButton(41, true);
4   static bool isMenuButtonPressed = false;
5   void loop(){
6     if (encoderPollTimer > encoderPollDelay){
7       //. . .
8       if (menuButton.onPressed()) isMenuButtonPressed = true;
9       if (menuButton.onReleased()) isMenuButtonPressed = false;
10    }
11  }
```

Kôd 3.23. Očitavanje stanja tipkala enkodera za izbornik.

3.4.5. Upravljanje LCD zaslonom

Za upravljanje LCD zaslonom korištena je biblioteka *LiquidCrystal_I2C* [22]. LCD prikazuje vrijednost parametra u troznamenkastom obliku u prva 3 slova prvog reda zaslona. Također, u zadnja 4 slova prvog reda prikazan je naziv trenutačno izabranog bubnja. U drugom redu prikazan je naziv parametra koji je zadnje promijenjen (slika 3.24.).



Sl. 3.24. Prikaz rada LCD zaslona

3.4.6. Klasa parameter

Kreirana je nova datoteka (*parameters.h*) u kojoj je stvorena klasa *parameter* koja sadržava vrijednost, nazive parametra i funkcije za promjenu njegove vrijednosti prema prethodno definiranim pravilima. Parametar se pri deklaraciji definira po svojoj minimalnoj i maksimalnoj vrijednosti. Od strane korisnika se može podesiti na jednu od 128 razina između te dvije vrijednosti. Minimalna vrijednost pripada razini 0, dok je maksimalna vrijednost razine 128. Svi parametri su pri pokretanju uređaja inicijalizirani na algebarsku srednju vrijednost između minimalne i maksimalne, razinu 64. Uvećanje parametra putem funkcije *increment()* povećati će razinu za 1 ako se nije dosegla maksimalna. Sukladno tome, umanjivanje parametra sa funkcijom *decrement()* smanjiti će razinu za 1 ako nije dosegnuta minimalna. Nakon svake promjene vrijednosti razine parametra, trenutna vrijednost se računa i sprema u varijablu unutar objekta. Dodane su i funkcije za uvećanje i smanjivanje parametra za 8 koraka: *increment8Times()* i *decrement8Times(),*za korištenje uz tipkalo na enkoderu za izbornik. Konačno, dodane su funkcije za dohvaćanje trenutne vrijednosti parametra, trenutnu razinu i imena parametra (kôd 3.25.).

Pozivanje funkcija za promjenu vrijednosti odvija se unutar funkcije *pollEncoders()*. Ako je rotacija u smjeru kazaljke na satu, poziva se *increment()*, ako je u suprotnom smjeru kazaljke na satu, poziva se *decrement()*. Također, ako je u trenutku promjene vrijednosti parametra pritisnuta tipka enkodera glavnog izbornika, umjesto standardnih, pozivaju se funkcije *increment8Times()* odnosno *decrement8Times()* (kôd 3.26.).

```
1   void pollEncoders(int i){
2     //. . .
3     if (newPos != encoderPositions[i]){
4       encoderPositions[i] = newPos;
5       if ((int)encoders[i].getDirection() == 1){
6         if (!isMenuButtonPressed){
7           params[currentDrumIndex][i]->increment();
8         }
9         else{
10          params[currentDrumIndex][i]->increment8Times();
11        }
12      }
13      else{
14        if (!isMenuButtonPressed){
15          params[currentDrumIndex][i]->decrement();
16        }
17        else{
18          params[currentDrumIndex][i]->decrement8Times();
19        }
20      }
21      //. . .
22    }
23  }
```

Kôd 3.26. Promjena vrijednosti parametara.

```
#include "Arduino.h"
class parameter{
private:
    float value;
    float stepIncrementValue;
    int currentStep;
    float min;
    float max;
    String name;
public:
    parameter(String name, float min, float max){
        this->name = name;
        this->currentStep = 64;
        this->stepIncrementValue = (max - min) / 128.0;
        this->min = min;
        this->max = max;
        this->value = (currentStep * stepIncrementValue) + this->min;
    }
    void increment(){
        if (currentStep < 128){
            this->currentStep++;
            this->value = (currentStep * stepIncrementValue) + this->min;
        }
    }
    void increment8Times(){
        if (currentStep < 120){
            this->currentStep = this->currentStep + 8;
            this->value = (currentStep * stepIncrementValue) + this->min;
        }
    }
    void decrement(){
        if (currentStep > 0){
            this->currentStep--;
            this->value = (currentStep * stepIncrementValue) + this->min;
        }
    }
    void decrement8Times(){
        if (currentStep > 8){
            this->currentStep = this->currentStep - 8;
            this->value = (currentStep * stepIncrementValue) + this->min;
        }
    }
    float getValue(){
        return this->value;
    }
    String getName(){
        return this->name;
    }
    int getCurrentStep(){
        return this->currentStep;
    }
};
```

Kôd 3.25. Objekt *parameter*.

3.4.7. Sintetizacija zvuka

Za sintetizaciju zvuka koristi se Teensy *Audio* biblioteka [11]. Ona omogućuje upravljanje Audio Adapter sklopovlja za primjenu sintetiziranja zvuka, filtriranja audio signala, reprodukciju i snimanje zvučnih zapisa uz brojne druge mogućnosti. Teensy *Audio* biblioteka se u projekt uključuje pomoću grafičkog alata *Audio System Design Tool* [23]. Alat radi na principu grafičkog spajanja elemenata kao što su oscilatori, zvučne omotnice, filteri, audio mikseri i efekti. Iz alata je onda moguće generiranje programskog kôda koji inicijalizira objekte iz Audio biblioteke prema grafičkom spoju. Generirani kod se dodaje u postojeću glavnu datoteku (*main.cpp*) prije pozivanja *setup()* funkcije.

U radu s Audio bibliotekom, prvo je potrebno stvoriti objekt za upravljanje integriranog kruga SGTL5000. U *setup()* funkciji definira se izlazna glasnoća te se pokreće funkcija *AudioMemory()*, koja dinamički zauzima određeni broj blokova memorije za sve potrebne međusobne veze između objekata Audio biblioteke (kôd 3.27.).

```
1   #include <Audio.h>
2   AudioControlSGTL5000     sgtl5000_1;
3   void setup(){
4     //. . .
5     AudioMemory(60);
6     sgtl5000_1.enable();
7     sgtl5000_1.volume(0.5);
8   }
```

Kôd 3.27. Stvaranje i inicijalizacija objekta SGTL5000.

U nastavku ovog poglavlja biti će pojašnjen postupak sinteze svakog od 8 pojedinih zvukova bubnja uz njihove definirane FM algoritme. Pojedini operator tvori 2 audio objekta: *WAVEFORM_MOD*, sinusoidni oscilator sa ulazom za signal koji ga frekvencijski modulira te *Audio Effect Envelope*, audio omotnica.

Tone je najjednostavniji stvoreni elektronički bubanj koji se sastoji od dva operatora, prvi od kojih je signal nositelj dok drugi izvodi frekvencijsku modulaciju nad njim (slika 3.28.). Korisnik može podesiti frekvenciju prvog i drugog operatora, vrijeme opadanja omotnice prvog operatora i amplitudu drugog operatora, što određuje količinu frekvencijske modulacije.

Sl. 3.28. Algoritam bubnja *tone* dizajniran u sučelju Audio System Design Tool.

Tone_op1 je oscilator prvog operatora, koji je nositelj, dok je tone_op1_env njegova zvučna omotnica. Isto vrijedi i za tone_op2 i tone_op2_env, koji je modulator.

Kreiranje objekta klase *parameter* za bubanj tone obavlja se prije pozivanja *setup()* funkcije (kôd 3.29.). Parametar 1 odnosi se na frekvenciju operatora 1, kojoj je maksimalna vrijednost postavljena na 4320Hz, dok je minimalna 20Hz. Parametar 2 predstavlja vrijeme opadanja zvučne omotnice operatora 1 te ga je moguće mijenjati od 10ms za iznimno kratak zvuk do 1200ms za produženi opadajući zvuk. Parametar 3 mijenja frekvenciju oscilacije operatora 2 te je graničnih vrijednosti 1Hz do 210Hz. Sa parametrom 4 moguće je odrediti amplitudu operatora 2 od razine 0 (potpuna tišina) do 1 (maksimalna vrijednost). Kod svih bubnjeva parametar 5 određuje ukupnu glasnoću. Svi parametri za pojedini bubanj dodani su u nizove kako bi im se moglo pristupiti putem indeksiranja unutar funkcije *pollEncoders()*.

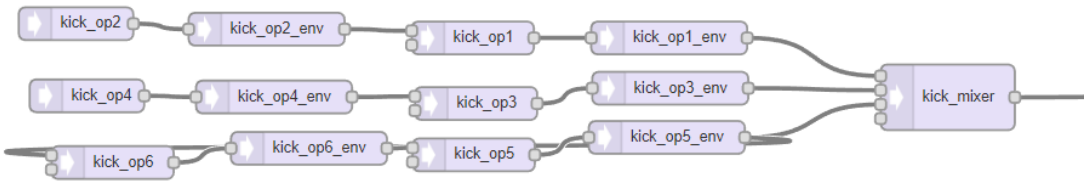
```
1   static parameter toneParam1("OP 1 FREQUENCY  ", 20, 4320);
2   static parameter toneParam2("OP 1 DECAY      ", 10, 1200);
3   static parameter toneParam3("OP 2 FREQUENCY  ", 1, 210);
4   static parameter toneParam4("OP 2 AMPLITUDE  ", 0, 1);
5   static parameter toneParam5("VOLUME          ", 0, 1);
6   static parameter *toneParams[5] = {&toneParam1, &toneParam2, &toneParam3, &toneParam4, &toneParam5};
```

Kôd 3.29. Kreiranje parametara za bubanj *tone*.

Kick je bubanj dizajniran prema zvuku bas bubnja sa konvencionalnog bubnjarskog seta. Karakteriziraju ga pretežno niske frekvencije (basovi). FM algoritam za sintetiziranje *kick* bubnja sadržava 6 operatora te se u njemu 3 signala operatora nositelja 1, 3 i 5 algebarski zbrajaju kako bi se dobio konačan zvuk. Operatori 2, 4 i 6 su modulatori te operator 6 nad sobom vrši samo-modulaciju. (slika 3.30.).

Parametar 1 bubnja *kick* određuje frekvencije operatora 1, 2, 3 i 4 u različitim omjerima. Parametar 2 mijenja brzinu opadanja zvučne omotnice operatora 1, 2 i 3, što određuje ukupno trajanje bubnja, od jako kratkog zvuka (20ms) do dugačkog (1750ms). Parametar 3 određuje amplitude operatora

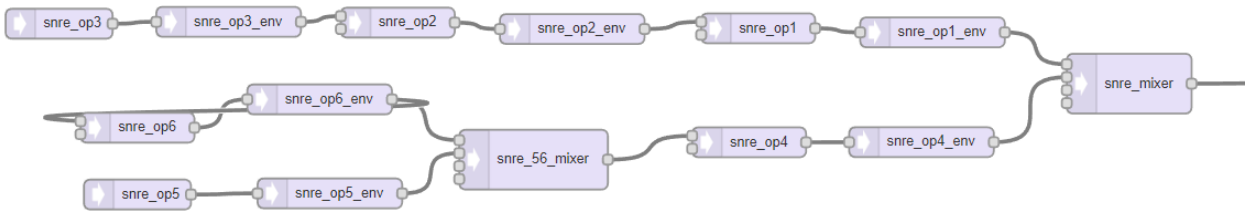
2 i 4, što će odražavati količinu frekvencijske modifikacije u cjelokupnoj sintezi. Parametar 4 mijenjati će frekvencije operatora 2 i 4.



Sl. 3.30. FM algoritam *kick*.

Bubanj *snare* je modeliran po bubnju imena „doboš“ koji se također može pronaći u konvencionalnom setu bubnjeva. Karakterizira ga ton kojim odzvanja te mogućnost spuštanja mreže na donju opnu bubnja koja stvara kratak zvuk šuma pri sviranju. FM algoritam *snare* bubnja sadržava 6 operatora (slika 3.31.).

Parametar 1 bubnja *snare* određuje frekvenciju tona kojim odzvanja, to jest, frekvenciju operatora 1, 2, 3 i 4. Parametar 2 mijenja vrijeme opadanja zvučne omotnice operatora 1, 3 i 4 dok će parametar 3 mijenjati samo vrijeme opadanja omotnice operatora 4, koji tvori šum bubnja. Parametar 4 mijenja amplitudu operatora 4.



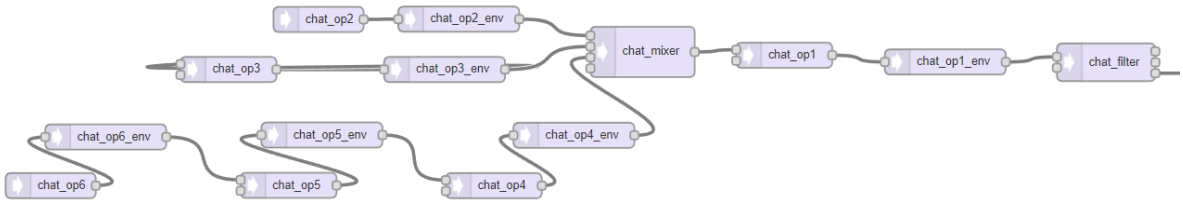
Sl. 3.31. FM algoritam *snare*.

Bubanj *cHat* predstavlja zatvorenu dvostruku činelu (engl. *closed hi-hat*, slika 3.32.). Karakterizira ga kratak zvuk koji se pretežito sastoji od šuma. Parametar 1 određuje frekvenciju operatora 1. Parametar 2 utječe na vrijeme opadanja omotnice operatora 1. Parametar 3 mijenja frekvenciju operatora 2. Parametar 4 određuje amplitudu operatora 2.

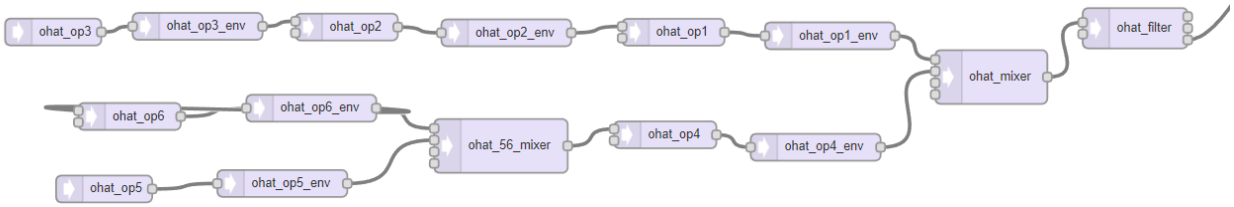
Analogno ovom bubnju, *oHat* je modeliran prema otvorenoj dvostrukoj čineli te ga karakterizira zvuk šuma koji opada kroz kraće vrijeme (slika 3.33.). Parametar 1 bubnja *oHat* mijenja

frekvenciju operatora 4 dok parametar 2 utječe na vrijeme opadanja njegove zvučne omotnice. Parametar 3 određuje će amplitudu operatora 5 a parametar 4 frekvenciju.

Na kraju algoritama za bubnjeve *cHat* i *oHat* dodani su *filter* objekti koji su konfigurirani da iz konačnih zvukova ovih bubnjeva filtriraju sve zvukove ispod 500 Hz kako bi rezultati algoritama vjernije odgovarali zvuku stvarnih činela.

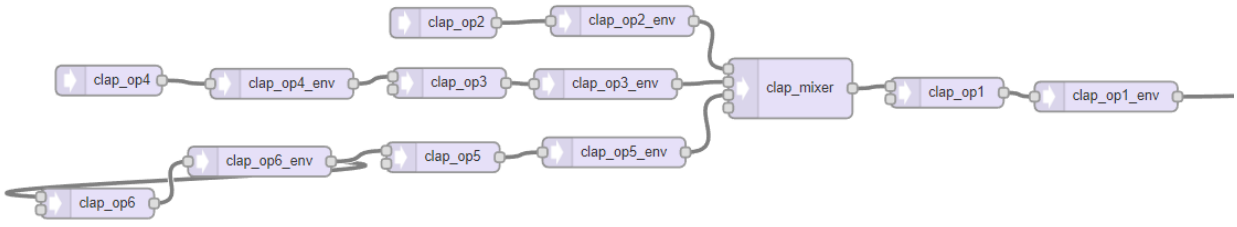


Sl. 3.32. FM algoritam *cHat*.



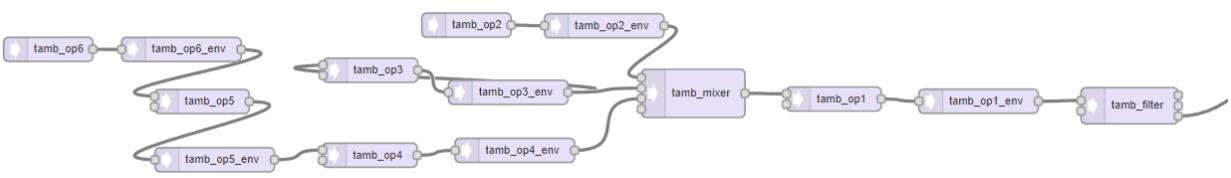
Sl. 3.33. FM algoritam *oHat*.

Bubanj *clap* kreiran je kao imitacija zvuka ljudskog pljeska. Pri njegovoj sintezi, parametar 1 mijenja frekvenciju operatora 1. Parametar 2 utječe na vrijeme opadanja njegove zvučne omotnice. Parametar 3 mijenja frekvenciju operatora 5, te parametar 4 povećava ili smanjuje amplitudu operatora 4 (slika 3.34.).



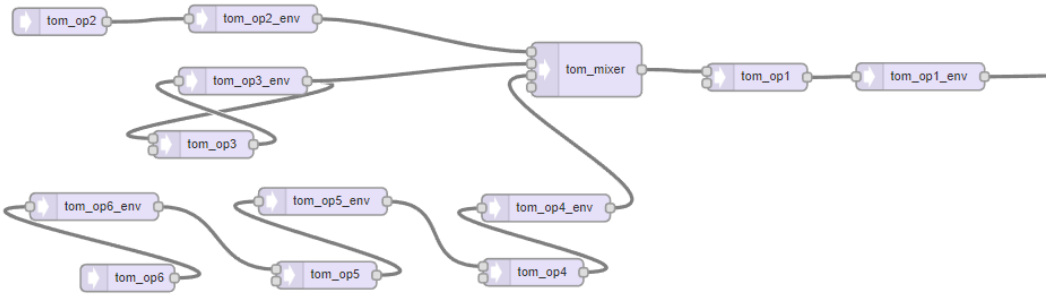
Sl. 3.34. FM algoritam *clap*.

Bubanj *tamb* stvoren je prema zvuku udaraljke tamburina. Kao i *cHat* i *oHat*, primjenjuje filter za niske frekvencije. Parametar 1 bubnja *tamb* mijenja frekvenciju operatora 1 dok parametar 2 utječe na vrijeme opadanja njegove zvučne omotnice. Parametar 3 mijenja frekvenciju operatora 2 te parametar 4 utječe na amplitudu operatora 4 (slika 3.35.).



Sl. 3.35. FM algoritam *tamb*.

Naposlijetku, dizajniran je zvuk *tom* bubnja, koji je modeliran prema istoimenom bubnju u klasičnom bubnjarskom setu. Parametar 1 određuje frekvenciju prvog operatora dok parametar 2 određuje vrijeme trajanja opadanja njegove zvučne omotnice. Parametar 3 određuje frekvenciju operatora 3. Parametar 4 određuje amplitudu operatora 6 (slika 3.36.).



Sl. 3.36. FM algoritam *tom*.

3.4.8. Sviranje nota

Kako bi se bubnjevi mogli svirati putem MIDI signala, potrebno je pripisati jednu MIDI notu za svaki bubanj. U tablici 3.2. prikazani su bubnjevi uz njihove pripisane note uz njihove MIDI vrijednosti.

Kako bi se provjeravale ulazne MIDI note, potrebno je usbMIDI objektu ukazati na *callback* funkciju *noteOnHandler()* koja će biti pokrenuta kada se putem kontinuiranog čitanja pročita nova

ulazna nota (kôd 3.37.). Ovisno o noti koja je pročitana, potrebno je započeti zvučne omotnice svih operatora koje pripadaju tom bubnju (kôd 3.38.).

Tablica 3.2. Bubnjevi i njihove pripisane note uz pripadajuće MIDI vrijednosti.

Bubanj	*Nota*	*MIDI vrijednost*
Kick	C4	0x48
Snare	C#4	0x49
cHat	D4	0x4A
oHat	D#4	0x4B
Tone	E	0x4C
Clap	F	0x4D
Tamb	F#	0x4E
Tom	G	0x4F

```
1   void setup(){
2     ///. . .
3     usbMIDI.setHandleNoteOn(noteOnHandler);
4     ///. . .
5   }
```

Kôd. 3.37. Dodavanje *callback* funkcije za čitanje ulaznih MIDI poruka.

```
1   void noteOnHandler(byte channel, byte note, byte velocity){
2     switch (note){
3     case 0X48:
4       kick_op1_env.noteOn();
5       kick_op2_env.noteOn();
6       kick_op3_env.noteOn();
7       kick_op4_env.noteOn();
8       kick_op5_env.noteOn();
9       kick_op6_env.noteOn();
10      break;
11    case 0X49:
12      snre_op1_env.noteOn();
13      snre_op2_env.noteOn();
14      snre_op3_env.noteOn();
15      snre_op4_env.noteOn();
16      snre_op5_env.noteOn();
17      snre_op6_env.noteOn();
18      break;
19      ///. . .
20    }
21  }
```

Kôd. 3.38. Pokretanje zvučnih omotnica za *kick* i *snare* bubnjeve.

4. Testiranje

Nakon programiranja uređaj omogućuje sviranje bubnjeva poslanih putem MIDI poruka dok je spojen putem USB sučelja. Za slanje MIDI poruka i određivanje sekvence moguće je koristiti bilo koju digitalnu zvučnu radnu stanicu. Za testiranje odabran je program *Ableton Live Lite*. Izlazni zvuk sa Audio Adaptera spojen je putem zvučnog kabla u audio ulaz na matičnoj ploči računala te je snimljen i analiziran u programu za uređivanje zvuka *Audacity*.

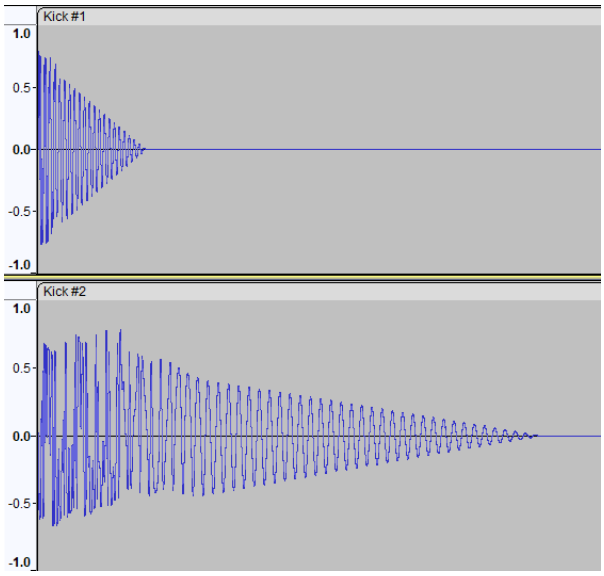
U naknadnim poglavljima biti će uspoređena 2 različita odziva 3 različita bubnja. Svaki od odziva sadržavati će različito postavljene parametre kako bi se uočilo njihovo djelovanje na proizvedeni zvuk bubnja.

4.1. Analiza zvukova bubnja

Snimljena su dva amplitudna odziva zvuka bubnja *kick* pri različitim parametrima sinteze (tablica 4.1.). Odzivi su grafički prikazani u programu *Audacity* (slika 4.1).

Tablica 4.1. Parametri za testni odziv bubnja *kick*.

Parametar	*Odziv 1*	*Odziv 2*
Pitch	90	20
Decay	20	100
Mod Amplitude	10	90
Mod Frequency	35	80
Volume	80	80



Sl. 4.1. Grafička usporedba amplitudnih odziva dva različita *kick* bubnja.

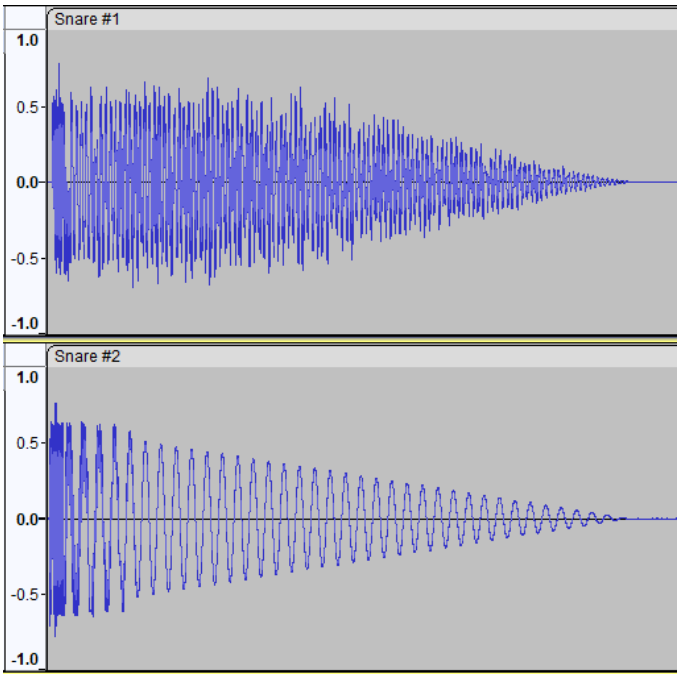
Odmah je moguće uočiti kako parametar *decay* utječe na trajanje opadanja vodeće zvučne omotnice u procesu sinteze te je zbog toga drugi odziv produžen. Također je lako uočljiv učinak prvog parametra, opća frekvencija zvuka bubnja je znatno veća u prvom odzivu nego u drugom. Povećana amplituda frekvencijske modulacije drugog odziva također čini razliku između početka udarca bubnja, koji sadržava frekvencijsku modulaciju, i kraja, koji izgleda kao čisti sinusoidni signal.

Također, snimljena su dva amplitudna odziva zvuka *snare* (tablica 4.2. i slika 4.2.)

Tablica 4.2. Parametri za testni odziv bubnja *snare*.

Parametar	*Odziv 1*	*Odziv 2*
Pitch	85	5
Decay	120	120
Noise length	120	10
Noise amplitude	120	120
Volume	80	80

U ovom slučaju je uočljivo djelovanje parametra *noise length* koji se odražava u dugačkom trajanju komponente šuma prvog odziva i iznimno kratkog šuma drugog odziva. Također, izuzetak šuma ostavlja prikazanu nisku frekvenciju drugog odziva koju odražava parametar *pitch*.

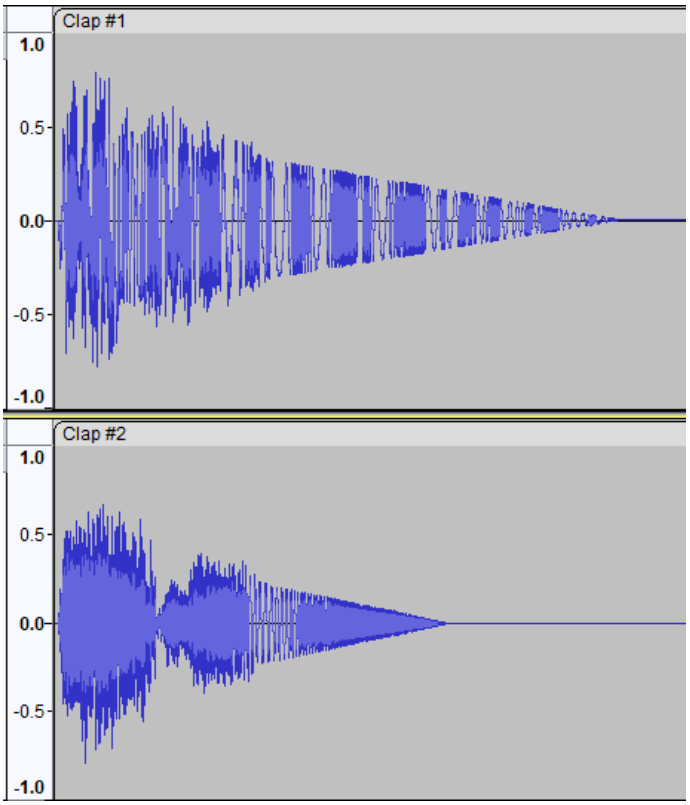


Sl. 4.2. Grafička usporedba amplitudnih odziva dva različita *snare* bubnja.

Konačno, snimljena su dva odziva bubnja *clap* (tablica 4.3. i slika 4.3.).

Tablica 4.3. Parametri za testni odziv bubnja *clap*.

Parametar	*Odziv 1*	*Odziv 2*
Pitch	10	80
Decay	50	30
Mod frequency	95	0
Mod amplitude	95	90
Volume	80	80



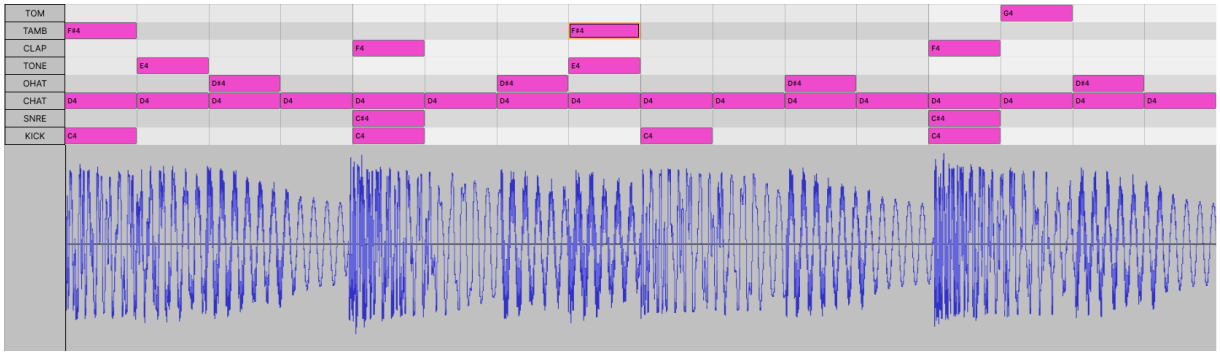
Sl. 4.3. Grafička usporedba amplitudnih odziva dva različita *clap* bubnja.

U ovom slučaju je moguće primijetiti učinak frekvencijske modulacije na prvom odzivu u brzom izmjenjivanju visokih i niskih frekvencija tijekom trajanja opadanja zvuka. U drugom odzivu je frekvencija znatno snižena i u toku cjelokupnog trajanja zvuka stigne modulirati zvuk bubnja samo jednom. Također, osnovna frekvencija koju postavlja parametar *pitch* je primjetno veća u drugom odzivu.

4.2. Izvođenje sekvenci bubnjeva

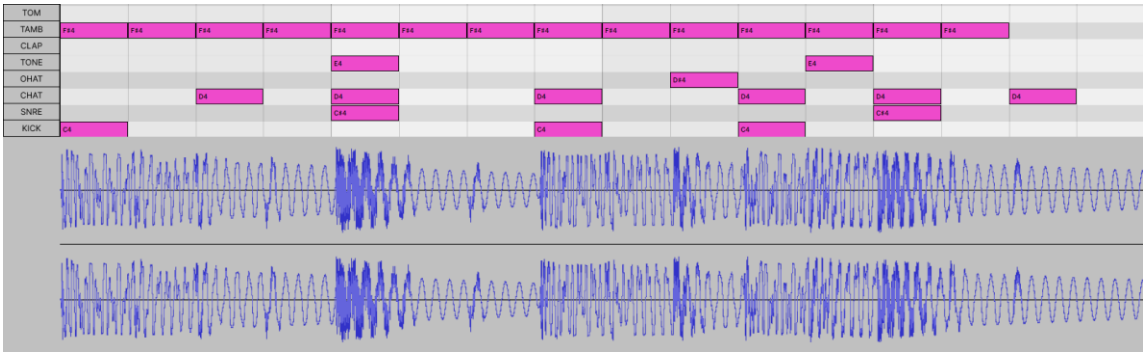
Konačno ispitivanje biti će izvođenje ritamske sekvence koja uključuje sve bubnjeve. U programu *Ableton Live Lite* izrađene su tri ritamske sekvence. Vizualizacija MIDI sekvenci unutar programa

uz njihovi snimljen amplitudni odziv nalaze se na slikama 4.4., 4.5. i 4.6. Ritam mašina u radu vidljiva je na slici 4.7.

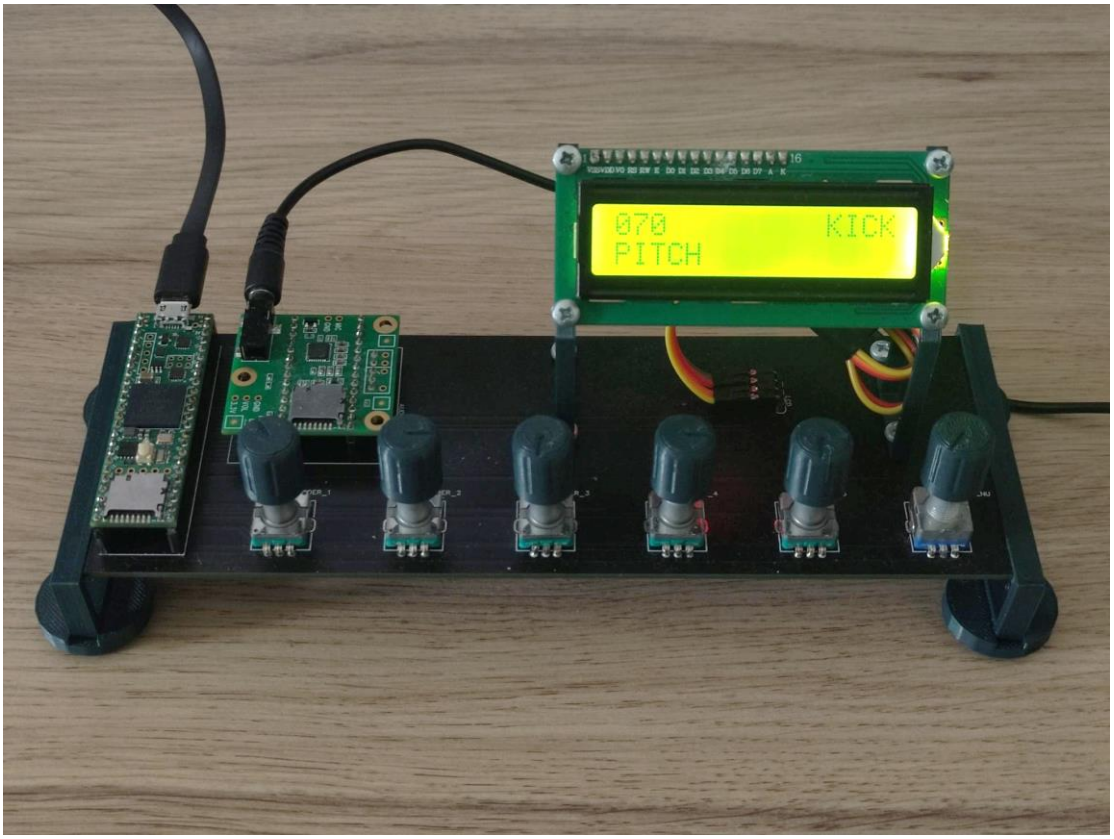


Sl. 4.4. Amplitudni odziv ritma 1.

Sl. 4.5. Amplitudni odziv ritma 2.



Sl. 4.6. Amplitudni odziv ritma 3.



Sl. 4.7. Ritam mašina u radu.

5. Zaključak

Tema ovog rada je izrada FM ritam mašine zasnovanoj na ARM32 razvojnom sustavu. Ritam mašine su uređaji koji prema korisnički zadanom ritmu sintetiziraju audio signal bubnjeva u tom ritmu, uz mogućnost promjene zvuka tih bubnjeva u stvarnom vremenu. FM sinteza zvuka ostvaruje se digitalno, putem frekvencijske modulacije signala kroz različite algoritme.

Izrađen je uređaj zasnovan na Teensy mikroupravljačkoj platformi uz Teensy Audio Adapter za audio izlaz, 6 rotacijskih enkodera za korisnički unos te LCD zaslonom. Izrađeni uređaj može proizvesti širok raspon različitih zvukova koje je moguće ugoditi za specifične potrebe po želji korisnika. Karakteristike digitalne FM sinteze čine ovu ritam mašinu pogodnu za korištenje u žanrovima elektroničke i lo-fi glazbe. Tijekom izrade bilo je potrebno detaljno upoznati se sa načelima FM sinteze te eksperimentirati sa različitim algoritmima kako bi se utvrdilo koji algoritam je prikladan za određeni zvuk. Također, bilo je potrebno samostalno utvrditi koji parametar sinteze korisnik može podesiti. Određeni parametri primjetnije utječu na sintetizirani zvuk nego ostali, te će dopuštanje modifikacije tih parametara imati veći utjecaj na konačni raspon zvuka sintetizatora. Rad sa rotacijskim enkoderima pruža efektivan način korisničkog unosa za uvećavanje ili smanjivanje parametara. LCD zaslon upotpunjava jednostavno i pregledno sučelje uređaja, omogućujući korisniku uvid u vrijednost parametra koji se trenutno mijenja.

Poboljšanje u radu bi se moglo ostvariti u mogućnosti unosa ritma na samom uređaju kroz ugrađeni sekvencer. Također, kroz bolje i naprednije FM algoritme te složenije parametre koji utječu na veći broj operatora u algoritmu.

Literatura

[1] M., Vail, *Vintage synthesizers: pioneering designers, groundbreaking instruments, collecting tips, mutants of technology*, 2nd ed. updated & Expanded. Miller Freeman Books: San Francisco, 2000.

[2] KORG USA, „Volca FM - DIGITAL FM SYNTHESIZER“ [online]. Dostupno na: https://www.korg.com/us/products/dj/volca_fm2/. [Pristupljeno: 9.8.2022.].

[3] ELEKTRON SE, „Elektron Model:Cycles“ [online]. Dostupno na: https://www.elektron.se/nl/modelcycles-explorer. [Pristupljeno: 9.8.2022.].

[4] YAMAHA UK, „MONTAGE 8 - Overview“ [online]. Dostupno na: https://uk.yamaha.com/en/products/music_production/synthesizers/montage/index.html. [Pristupljeno: 9.8.2022.].

[5] M., Russ, *Sound Synthesis and Sampling*. CRC Press, 2012.

[6] S., Cann, Hoopla digital, *Frequency modulation synthesis*. BookBaby : Made available through hoopla: United States, 2011.

[7] „Yamaha DX7 chip reverse-engineering, part 4: how algorithms are implemented“ [online]. Dostupno na: http://www.righto.com/2021/12/yamaha-dx7-chip-reverse-engineering.html. [Pristupljeno: 9.8.2022.].

[8] ARM, „ARM CPU Architecture“ [online]. Dostupno na: https://www.arm.com/. [Pristupljeno: 9.8.2022.].

[9] „Exploring the ARM® Cortex®-M7 Core: Providing Adaptability for the Internet of Tomorrow“, str. 10.

[10] „Teensy® 4.0“ [online]. Dostupno na: https://www.pjrc.com/store/teensy40.html. [Pristupljeno: 9.8.2022.].

[11] PJRC, „Teensy Audio Adaptor“ [online]. Dostupno na: https://www.pjrc.com/store/teensy3_audio.html. [Pristupljeno: 9.8.2022.].

[12] D., Paret, *The I^2C bus: from theory to practice*. Wiley: Chichester ; New York, 1997.

[13] NXP, Philips, „I2C-bus specification and user manual“, sv. 2021, str. 62, 2021.

[14] „I2S bus specification“, sv. 2022, str. 14, 2022.

[15] Dejan, „How Rotary Encoder Works and How To Use It with Arduino“ [online], 25-srp-2016. .

[16] PlatformIO, „PlatformIO is a professional collaborative platform for embedded development“ [online]. Dostupno na: https://platformio.org. [Pristupljeno: 21.8.2022.].

[17] „Teensy — PlatformIO latest documentation“ [online]. Dostupno na: https://docs.platformio.org/en/latest/platforms/teensy.html. [Pristupljeno: 21.8.2022.].

[18] „Teensyduino: Using USB MIDI with Teensy on the Arduino IDE“ [online]. Dostupno na: https://www.pjrc.com/teensy/td_midi.html. [Pristupljeno: 21.8.2022.].

[19] F., Trias, „Teensy Threading Library“. 03-kol-2022.

[20] „A Library for the Arduino environment for using a rotary encoder as an input.“ [online]. Dostupno na: http://www.mathertel.de/Arduino/RotaryEncoderLibrary.aspx. [Pristupljeno: 21.8.2022.].

[21] „RBD_Button - Arduino Reference“ [online]. Dostupno na: https://www.arduino.cc/reference/en/libraries/rbd_button/. [Pristupljeno: 23.8.2022.].

[22] „LiquidCrystal I2C - Arduino Reference“ [online]. Dostupno na: https://www.arduino.cc/reference/en/libraries/liquidcrystal-i2c/. [Pristupljeno: 23.8.2022.].

[23] „Audio System Design Tool for Teensy Audio Library“ [online]. Dostupno na: https://www.pjrc.com/teensy/gui/. [Pristupljeno: 23.8.2022.].

Sažetak

Ovaj rad pobliže prikazuje izradu hardverske digitalne FM ritam mašine. U uvodu prikazana su 3 postojeća FM sintetizatora na tržištu u tri različita cjenovna ranga. Nadalje je objašnjen sam pojam sinteze i pobliže je prikazana FM sinteza kroz pojmove frekvencijske modulacije, zvučnih omotnica, operatora i algoritama. Obavljen je pregled ARM32 arhitekture procesora te Teensy mikroupravljača i njegovog hardverskog proširenja Teensy Audio Adapter. Za korisnički unos na izrađenom uređaju korišteni su rotacijski enkoderi čiji je princip rada objašnjen u radu. Korisnici imaju vizualnu povratnu informaciju putem LCD zaslona. Izrađena ritam mašina može proizvesti 8 različitih zvukova bubnja, svaki od kojih sadržava 5 parametara koji se mogu mijenjati u stvarnom vremenu. Svaki bubanj koristi različit FM algoritam koji oblikuje njegov konačan zvuk. Ritam mašina je testirana kroz proizvodnju individualnih zvukova bubnja i proizvodnju ritmova putem MIDI USB ulaza.

Ključne riječi: Sintetizacija zvuka, FM, Teensy, MIDI, Ritam mašina

Abstract

Title: FM synthesizer developed on ARM32 development kit

This paper provides a detailed display of creating a digital FM drum machine. The introduction features a comparison between 3 existing FM synthesizers at 3 different price points. Next is a closer look at the definition of audio synthesis in general, frequency modulation, FM synthesis, audio envelopes, FM operators and FM algorithms. An overview was provided of the ARM32 processor architecture alongside the Teensy development board with its Audio Adaptor hardware expansion. Rotary encoders were used for user input and an LCD display was used for displaying visual information. The constructed drum machine can make 8 different drum sounds, each of which contains 5 parameters which can be changed in real time. Each drum uses a different FM algorithm which shapes its final sound. The drum machine was tested with synthesizing individual drum sounds and full drum loops, which were played over USB MIDI.

Keywords: Sound synthesis, FM, Teensy, MIDI, Drum machine

Životopis

Robert Sorić rođen je u Osijeku 20.7.1997. godine. Prebivalište mu je u Dardi, gdje je pohađao osnovnu školu. U Osijeku je nakon toga pohađao Elektrotehničku i Prometnu školu, smjer tehničar za računalstvo, u kojoj je imao priliku otići u Irski grad Bray na stručnu praksu i s kolegom sudjelovati u izradi nove web stranice za školu. Na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku upisuje i završava preddiplomski studij računarstva, te nastavlja na diplomski studij. Tijekom studiranja sudjelovao je na Erasmus studentskoj mobilnosti u gradovima Budimpešta i Prag, gdje je na Engleskom jeziku uspješno položio sve upisane kolegije. Trenutno radi na započinjanju karijere u razvoju programske podrške za mikroupravljače.

_____

Potpis autora

Prilozi na CD-u

Izvorni kod za Teensy 4.1 unutar PlatformIO projekta

EasyEDA shema za PCB

EasyEDA dizajn PCB ploče

Audio uzorci proizvedeni na ritam mašini

Rad u .pdf obliku

Rad u .docx obliku