

УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: Јасмина Савић

Број индекса: РА40-2017

Тема рада: Динамичка детекција врсте улазног аудио сигнала у реалном времену на платформи са ограниченим ресурсима

Ментор рада: доц. др Јелена Ковачевић

Нови Сад, јул, 2021



КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР:		
Идентификациони број, ИБР:		
Тип документације, ТД:	Монографска документација	
Тип записа, ТЗ:	Текстуални штампани материјал	
Врста рада, ВР:	Завршни (Bachelor) рад	
Аутор, АУ:	Јасмина Савић	
Ментор, МН:	доц. др Јелена Ковачевић	
Наслов рада, НР:	Динамичка детекција врсте улазног аудио сигнала у реалном времену на платформи са ограниченим ресурсима	
Језик публикације, ЈП:	Српски / латиница	
Језик извода, ЈИ:	Српски	
Земља публиковања, ЗП:	Република Србија	
Уже географско подручје, УГП:	Војводина	
Година, ГО:	2021	
Издавач, ИЗ:	Ауторски репринт	
Место и адреса, МА:	Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, ФО: (поглавља/страна/цитата/табела/слика/графика/прилога)	8/39/14/8/25/0/0	
Научна област, НО:	Електротехника и рачунарство	
Научна дисциплина, НД:	Рачунарска техника	
Предметна одредница/Кључне речи, ПО:	Дигитална обрада сигнала, анализа аудио садржаја, аутоматска сегментација аудио садржаја	
УДК		
Чува се, ЧУ:	У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, ВН:		
Извод, ИЗ:	У оквиру рада имплементирано је једно решење алгоритма за детекцију врсте улазног аудио сигнала на процесору за дигиталну обраду сигнала фирме <i>Cirrus Logic</i>	
Датум приhvатања теме, ДП:		
Датум одбране, ДО:		
Чланови комисије, КО:	Председник:	
	Члан:	Потпис ментора
	Члан, ментор:	



KEY WORDS DOCUMENTATION

Accession number, ANO:		
Identification number, INO:		
Document type, DT:	Monographic publication	
Type of record, TR:	Textual printed material	
Contents code, CC:	Bachelor Thesis	
Author, AU:	Jasmina Savić	
Mentor, MN:	Jelena Kovačević PhD	
Title, TI:	Dynamic segmentation and classification of audio signals on a resource-constrained real-time platform	
Language of text, LT:	Serbian	
Language of abstract, LA:	Serbian	
Country of publication, CP:	Republic of Serbia	
Locality of publication, LP:	Vojvodina	
Publication year, PY:	2021	
Publisher, PB:	Author's reprint	
Publication place, PP:	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD: (chapters/pages/ref./tables/pictures/graphs/appendices)	8/39/14/8/25/0/0	
Scientific field, SF:	Electrical Engineering	
Scientific discipline, SD:	Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, S/KW:	Digital signal processing, Audio content analysis, Automatic audio segmentation	
UC		
Holding data, HD:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, N:		
Abstract, AB:	This paper presents an implementation of an audio segmentation and classification algorithm of the input audio signal on a <i>Cirrus Logic</i> digital signal processor	
Accepted by the Scientific Board on, ASB:		
Defended on, DE:		
Defended Board, DB:	President:	
	Member:	
	Member, Mentor:	
		Menthor's sign

Zahvalnost

Zahvaljujem se Institutu *RT-RK* iz Novog Sada koji mi je omogućio realizaciju ovog rada, svojoj mentorki, doc. dr Jeleni Kovačević i tehničkim mentorima Azri Samac i Goranu Babiću na stručnoj pomoći, savetima, podršci i razumevanju tokom izrade završnog rada.

Takođe, zahvaljujem se svojoj porodici, prijateljima i kolegama na neizmernoj podršci koju mi pružaju u svakom trenutku.



KEY WORDS DOCUMENTATION

SADRŽAJ

1.	Uvod	1
2.	Segmentacija i klasifikacija audio signala.....	3
2.1	Primena algoritama za segmentaciju i klasifikaciju audio signala	3
2.2	Pregled algoritama za segmentaciju i klasifikaciju audio signala.....	5
2.3	Osnova za odabrani algoritam segmentacije i klasifikacije audio signala.....	7
3.	Ciljna hardverska arhitektura za rad u realnom vremenu.....	9
3.1	Predlog ciljne hardverske arhitekture	9
3.2	Karakteristike procesora za digitalnu obradu signala	10
3.3	Arhitektura procesora CS49844.....	12
3.4	Sistemska programska podrška procesora CS49844	15
4.	Razvoj algoritma za segmentaciju i klasifikaciju audio signala.....	16
4.1	Tok razvoja algoritma za segmentaciju i klasifikaciju audio signala	16
4.1.1	Diskriminacija između muzike, govora i šuma	16
4.1.2	Prilagodavanje algoritma izvršavanju na ciljnoj platformi	18
4.1.3	Klasifikacija i konačna odluka na osnovu konteksta.....	19
4.2	Konačna struktura algoritma.....	20
4.3	Završna obrada signala	22
4.4	Programska implementacija algoritma na PC platformi.....	22
4.4.1	Programska implementacija u C programskom jeziku.....	22
4.4.1.1	Funkcija main.....	23
4.4.1.2	Funkcija detection_init.....	23
4.4.1.3	Funkcija module_main.....	23
4.4.1.4	Funkcija update_mler.....	24
4.4.1.5	Funkcija segment_level_detection	24



KEY WORDS DOCUMENTATION

4.4.1.6	Funkcija out_processing.....	24
4.4.1.7	Funkcija shelving_lp	24
4.4.1.8	Funkcija first_order_IIR.....	24
5.	Programska implementacija u asemblerском језику цилне платформе	25
5.1	Razvojno okruženje	25
5.2	Implementacija modula za samostalno izvršavanje	26
5.3	Povezivanje sa radnim okruženjem	27
6.	Rezultati.....	30
6.1	Mere performanse algoritma.....	30
6.1.1	Određeni parametri algoritma i njegove performanse.....	31
6.2	Ispitivanje ispravnosti programa.....	32
6.3	Utrošak resursa.....	34
7.	Zaključak	36
8.	Literatura	38

SPISAK SLIKA

Slika 2.1 Primer sistema za anotaciju audio datoteke [5]	4
Slika 2.2 Automatska segmentacija i detekcija vrste audio signala sa usmeravanjem završne obrade [6]	4
Slika 2.3 Tipičan primer algoritma za segmentaciju i klasifikaciju audio signala [6].....	6
Slika 3.1 Poređenje arhitektura procesora opšte namene (Fon Nojman) i digitalnih signal procesora (Harvard).....	11
Slika 3.2 Blok dijagram procesora CS49844	12
Slika 3.3 <i>Cirrus Logic</i> 32-bitna arhitektura	13
Slika 3.4 Prikazi brojeva procesora CS49844	14
Slika 3.5 Tok podataka i akumulatorska jedinica jezgra	14
Slika 3.6 Blok dijagram sprežnog sistema modula i OS.....	15
Slika 4.1 Histogram MLER za $\delta = 0.1$	17
Slika 4.2 Histogram MLER za $\delta = 0.5$	17
Slika 4.3 Dijagram toka odluke o vrsti signala	17
Slika 4.4 Primer klizećeg prozora.....	18
Slika 4.5 Dijagram toka inicijalnog dela algoritma	21
Slika 4.6 Dijagram toka glavnog dela algoritma	21
Slika 4.7 Kontrolna C struktura stanja algoritma	22
Slika 4.8 Hijerarhija funkcija u C modulu	23
Slika 5.1 Primer kontrolisanog izvršavanja u CLIDE razvojnem okruženju	26

Slika 5.2 Kontrolna struktura stanja algoritma u asemblerском језику	26
Slika 5.3 Primer paralelnog izvršavanja instrukcija kod implementacije IIR filtra.....	27
Slika 5.4 Pokazivači na ulazno-izlaznu sprežnu memoriju	28
Slika 5.5 Dijagram toka inicijalne faze prilagođenog algoritma	29
Slika 5.6 Dijagram toka glavnog dela prilagođenog algoritma	29
Slika 6.1 <i>RT-AG</i> zvučna kartica povezana sa razvojnom pločom CDB49x	33
Slika 6.2 Poređenje spektra muzičkog segmenta na ulazu i izlazu u programu <i>Audacity</i>	34

SPISAK TABELA

Tabela 4.1 Opis parametara modela	20
Tabela 6.1 Matrica konfuzije za klasu A	31
Tabela 6.2 Vrednost parametara dobijenog modela	31
Tabela 6.3 Matrica konfuzije za dobijeni model za testni skup tokova audio podataka	32
Tabela 6.4 Preciznosti i odzivi za pojedinačne klase.....	32
Tabela 6.5 Makro vrednosti preciznosti i odziva klasifikatora i F_1 mera	32
Tabela 6.6 Utrošak memorije modula.....	34
Tabela 6.7 Utrošak procesorskog vremena u najgorem slučaju	35

SKRAĆENICE

DSP	- <i>Digital Signal Processing</i> , Digitalna obrada signala
ASR	- <i>Automatic Speech Recognition</i> , Automatsko prepoznavanje govora
EQ	- <i>Equalizer</i> , Ekvilajzer
ZCR	- <i>Zero Crossing Rate</i> , Stopa promene znaka signala
STE	- <i>Short-term Energy</i> , Kratkotrajna energija signala
MFCC	- <i>Mel-Frequency Cepstral Coefficients</i> , Spektralni koeficijenti Melove frekvencijeske skale
LSTER	- <i>Low Short-Time Energy Ratio</i> , Procenat okvira signala s niskom kratkotrajnom energijom
MLER	- <i>Modified Low Energy Ratio</i> , Modifikovani procenat okvira signala s niskom kratkotrajnom energijom
AVR	- <i>Audio/Video Receiver</i> , Audio/video prijemnik
FPGA	- <i>Field-Programmable Gate Array</i>
ASIC	- <i>Application-Specific Integrated Circuit</i> , Integrисано kolo specifično za primenu
MAC	- <i>Multiply-Accumulate</i> , Pomnoži i saberi
SIMD	- <i>Single Instruction, Multiple Data</i> , Jednostruka instrukcija, višestruki podaci
AGU	- <i>Address Generation Unit</i> , Jedinica za generisanje adresa, adresni generator
SRS	- <i>Shifter/Rounder/Saturator</i> , Jedinica za pomeranje, zaokruživanje i zasićenje
ALU	- <i>Arithmetic Logic Unit</i> , Aritmetičko-logička jedinica

OS	- <i>Operating System</i> , Operativni sistem
MIF	- <i>Module Interface</i> , Sprežni podsistem modula
MCT	- <i>Module Call Table</i> , Tabela rutina modula za spregu sa OS
MCV	- <i>Module Control Vector</i> , Tabela konfiguracionih parametara modula
IIR	- <i>Infinite Impulse Response</i> , Beskonačan impulsni odziv
CLIDE	- <i>Cirrus Logic IDE</i> , Integrisano razvojno okruženje <i>Cirrus Logic</i>
PCM	- <i>Pulse-code modulation</i> , Impulsna kodna modulacija
MIPS	- <i>Million Instructions Per Second</i> , Milion instrukcija po sekundi

1. Uvod

Algoritmi za segmentaciju i klasifikaciju audio signala javljaju se uglavnom kao podsistemi kod sistema za automatsko prepoznavanje govora [1], sistema za analizu sadržaja radio i TV programa u cilju poboljšanja doživljaja audio sadržaja od strane pratilaca programa [2], sistema za indeksiranje audio-vizuelnih podataka na osnovu sadržaja [3][4], i mnogih drugih sistema za segmentaciju audio sadržaja. Ovakav podistem se može koristiti kao komponenta za početnu, ulaznu obradu signala, gde je, u zavisnosti od detektovane vrste datog audio signala, moguće izvršiti obradu odgovarajuću toj vrsti audio signala kao pripremu za dalje faze rada, što doprinosi boljem radu celokupnog sistema. Takođe, može se koristiti na izlaznoj strani sistema, gde će se na osnovu detekcije vrste audio signala doneti odluka o završnoj obradi signala koji se reprodukuje.

Zadatak rada je razvoj algoritma za segmentaciju ulaznog audio sadržaja i klasifikaciju audio signala u jednu od 3 grupe – govor, muzika ili šum. Nakon segmentacije i klasifikacije, signal je potrebno propustiti kroz odgovarajuću završnu obradu u zavisnosti od detektovane vrste signala. Predloženi algoritam je potrebno implementirati na DSP platformi i optimizovati u skladu sa zahtevima rada u realnom vremenu.

Cilj rada je prolazak kroz celokupan proces razvoja audio aplikacija na digitalnom signal procesoru: od razvoja algoritma, preko upoznavanja sa postupkom pisanja programske podrške u realnom vremenu na platformama sa ograničenim resursima, opštim postupcima optimizacije koda, kontrolisanog izvršavanja na simulatoru kao i na hardverskoj platformi, ispitivanja, verifikacije i analize rada audio DSP aplikacija, i na kraju, izradom odgovarajuće tehničke dokumentacije.

Izrada ovog rada je tekla u dve faze. Prva faza obuhvata upoznavanje sa problemom zadatka i ciljnom platformom, odabir i prilagođavanje algoritma za DSP platforme, određivanje parametara algoritma segmentacije i klasifikacije u programskom jeziku *Python* i realizacija referentnog koda algoritma detekcije u *C* programskom jeziku. Druga faza obuhvata prolazak kroz proces implementacije algoritma u vidu DSP aplikacije u asemblerskom jeziku ciljne platforme, kao i optimizaciju asemblerskog koda radi smanjenja utroška hardverskih resursa. Rešenje je kroz proces izrade testirano na više načina, a DSP aplikacija koja ga implementira je profilisana radi analize utroška resursa. Pri izradi rada, kao ciljna platforma korišćena je DSP platforma firme *Cirrus Logic*.

Rad se sastoji iz osam poglavlja. U drugom poglavlju opisani su algoritmi za segmentaciju i klasifikaciju audio signala i njihova primena, kao i osnova za razvijeni algoritam u ovom radu. Treće poglavlje opisuje odabranu ciljnu platformu za implementaciju algoritma. U četvrtom poglavlju, opisan je tok izrade algoritma i njegovo prilagođenje za izvršavanje na ciljnoj platformi, kao i programsku implementaciju na PC platformi. Peto poglavlje opisuje implementaciju algoritma u asemblerskom jeziku ciljne platforme. U šestom poglavlju su prikazane dobijene performanse algoritma, kao i rezultati programskog rešenja, u vidu utroška memorije, procesorskog vremena i testova na nivou bita. Sedmo poglavlje daje zaključak rada, dok osmo poglavlje nabraja literaturu korišćenu pri izradi rada.

2. Segmentacija i klasifikacija audio signala

U ovom poglavlju, ukratko je opisana primena i značaj sistema za segmentaciju i klasifikaciju audio signala, urađen je pregled literature i radova koji opisuju različite algoritme segmentacije i klasifikacije audio signala i opisana je osnova za razvijeni algoritam za platforme sa ograničenim resursima i za rad u realnom vremenu.

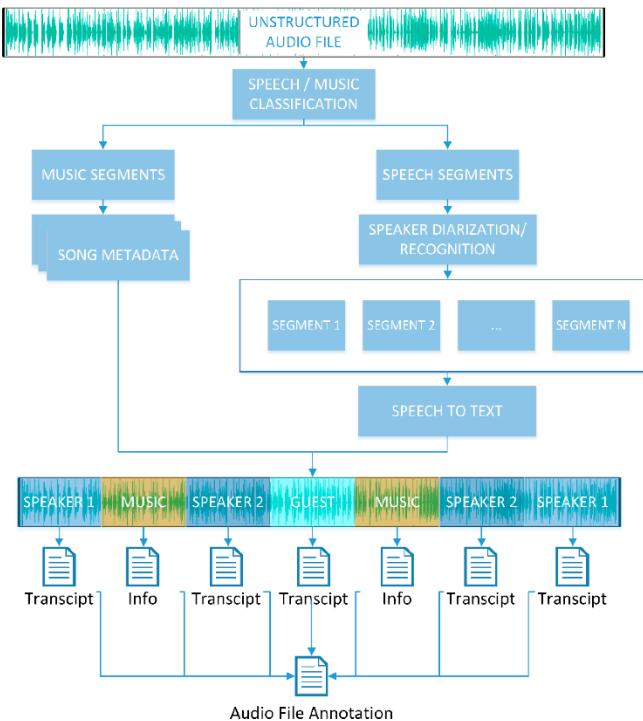
2.1 Primena algoritama za segmentaciju i klasifikaciju audio signala

Algoritmi za segmentaciju i klasifikaciju vrste audio signala se obično koriste kao podsistem nekog većeg sistema. Kako bi se razvio odgovarajući algoritam, potrebno je upoznati se sa primenom algoritma unutar većeg sistema, ciljnom platformom, kao i postavljenim ograničenjima – vremenskim i u vidu računarskih resursa.

Algoritmi za segmentaciju i klasifikaciju audio signala prema vrsti sadržaja značajno doprinose radu većih sistema kojima pripadaju, gde služe kao korak za pripremu podataka ili signala za dalju obradu.

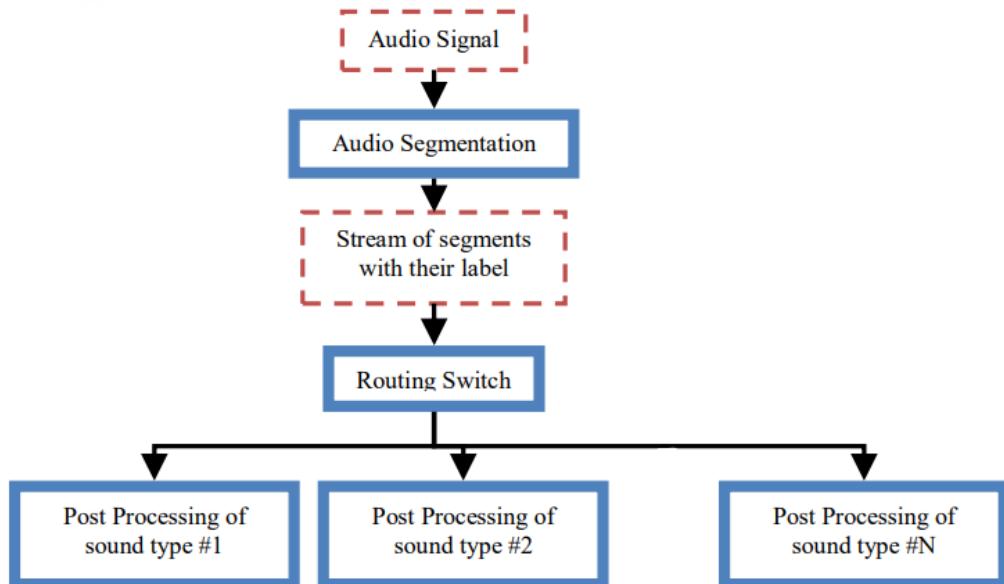
Jedan primer ovakve primene algoritama za segmentaciju i klasifikaciju audio signala je kod sistema za automatsko prepoznavanje govora (ASR). Kod ovakvih sistema, iz prikupljenog audio sadržaja se izdvajaju segmenti audio signala koji sadrže govor, a segmenti koji sadrže šum, muziku ili neku drugu pozadinsku buku se odbacuju, ili se preusmeravaju ka sistemu za prepoznavanje scenarija radi pružanja konteksta ASR sistemu. Ovakvi sistemi mogu da rade i u realnom vremenu, ali i bez vremenskih ograničenja.

Drugi primer su sistemi za anotaciju audio datoteka, gde bi podsistem za segmentaciju audio signala pripremio podatke drugim podsistemima – ASR i podsistemu za identifikaciju muzičkog sadržaja, za odgovarajuću obradu u okviru ovih podsistema – prepoznavanje govornika, transkripcija govora, pronalaženje izvornih podataka muzičkog sadržaja itd. [5]



Slika 2.1 Primer sistema za anotaciju audio datoteke [5]

Jedna od primena gde se algoritam za segmentaciju i detekciju vrste audio signala koristi za odluku o završnoj obradi signala je kod sistema za analizu emitovanog i reprodukovanih audio/video sadržaja u okviru ugrađenih sistema za reprodukciju zvuka (npr. radio) ili multimedijalnog sadržaja.



Slika 2.2 Automatska segmentacija i detekcija vrste audio signala sa usmeravanjem završne obrade [6]

2.2 Pregled algoritama za segmentaciju i klasifikaciju audio signala

Kao što je već spomenuto, pri odabiru algoritma za segmentaciju i klasifikaciju audio signala, posebnu pažnju treba obratiti na primenu sistema u okviru kog se koristi i na ciljnu platformu na kojoj se taj algoritam treba izvršavati. U slučaju platformi za rad u realnom vremenu sa ograničenim resursima, potrebno je razviti algoritam u okviru ograničenja računarskih resursa ciljne platforme, ili algoritam koji se može lako prilagoditi ciljnoj platformi radi maksimalnog iskorišćenja njenih proširenja za specifičnu primenu. Kako primena i ciljna platforma znatno utiču na algoritam, potrebno je analizirati njegovu opštu strukturu i modifikacije koje se u nju mogu uneti, kako bi se prilagodio specifičnoj primeni.

Analizom radova i literature iz oblasti segmentacije i klasifikacije audio signala [6], može se uopštiti struktura sistema za segmentaciju i klasifikaciju vrste audio signala, razdvajanjem obrade audio signala u nekoliko koraka:

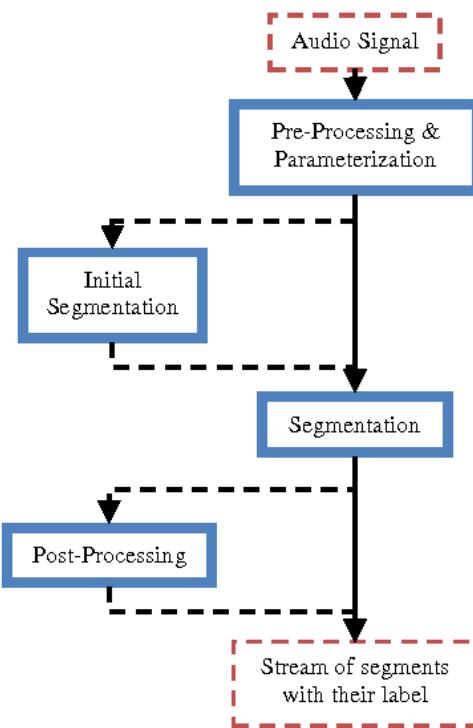
1. izdvajanje obeležja signala,
2. preliminarna faza detekcije (opciono),
3. glavna faza detekcije i
4. donošenje odluke na osnovu konteksta ili završna obrada rezultata (opciono).

U prvom koraku, signal se obično razdvaja na okvire određenog broja odbiraka i sračunavaju se obeležja signala potrebna za odabrani algoritam detekcije.

U drugom koraku, okviri audio signala se grupišu u segmente fiksne dužine, gde se na nivou tih segmenata preliminarno vrši odluka o vrsti audio signala. Ovaj korak nije obavezan, ukoliko odabrani algoritam detekcije ne zahteva proračune na nivou ovako grupisanih okvira.

Treći korak podrazumeva glavnu fazu detekcije, gde se vrši konačna odluka o segmentaciji audio signala i određivanju vrste signala u okviru tog segmenta. Ovaj korak je mesto gde se uglavnom algoritmi razlikuju. Detaljan proces detekcije i segmentacije zavisi od prvobitno odabranih obeležja signala i sistema za odlučivanje o vrsti audio signala i segmentaciju.

Četvrti korak nije obavezan. Ukoliko izlaz iz trećeg koraka ne daje zadovoljavajuće rezultate, tj. uočavaju se nagle i veoma brze promene vrste audio signala manjeg trajanja od empirijski utvrđenog kao prihvatljivog, potrebno je izvršiti završnu obradu rezultata trećeg koraka, kako bi se ovakve anomalije uklonile.



Slika 2.3 Tipičan primer algoritma za segmentaciju i klasifikaciju audio signala [6]

Obeležja signala kao i odgovarajući mehanizmi odluke o vrsti audio signala i segmentaciji koje algoritmi mogu da koriste su veoma opširni i različiti. Koriste se vremenske karakteristike signala, kao i spektralne, za koje je utvrđeno različitim analizama da imaju moć razlikovanja vrste audio signala. Algoritmi mogu da koriste različit broj obeležja – od samo jednog, pa do preko dvadeset obeležja [7]. Neka od obeležja audio signala koje se najčešće koriste su:

- stopa promene znaka amplitude signala (ZCR), kratkotrajna energija signala (STE) – karakteristike u vremenskom domenu;
- spektralni koeficijenti Melove frekvencijske skale (MFCC), spektralna energija, spektralni centroid, širina frekventnog opsega – karakteristike u spektralnom domenu.

Što se tiče mehanizama odluke o vrsti audio signala i segmentaciji, koriste se razni mehanizmi odluke i klasifikacije, od jednostavnijih, kao što su metrike na osnovu udaljenosti (npr. Euklidova), jednostavni Bajesovi klasifikatori, stabla odlučivanja, statistički testovi, do složenijih, kao što različiti algoritmi mašinskog učenja i hibridni algoritmi koji kombinuju neke od prethodnih klasifikatora.

Platforme za rad u realnom vremenu često su ograničene u vidu memorijskih resursa i procesne moći. Dodatno, obrada u realnom vremenu i analiza toka audio podataka na ulazu kod ovakvih platformi znatno razlikuje od obrade koja nema vremenska ograničenja i koja omogućava analizu bilo kog dela ulazne audio datoteke. Obrada u realnom vremenu pruža vremenski mnogo kraći kontekst i ograničeno vreme za odluku o vrsti signala, za razliku od

mogućnosti rada nad celom audio datotekom u drugim slučajevima. Ovi faktori su odlučujući pri odabiru algoritma za segmentaciju i klasifikaciju za rad na takvим platformama.

2.3 Osnova za odabrani algoritam segmentacije i klasifikacije audio signala

Pri odabiru algoritma detekcije i segmentacije uzeta su u obzir ograničenja ciljne platforme – digitalnog signal procesora, kao i ograničenja koje nameću realno vreme i rad sa tokom audio podataka, koji mora biti reprodukovani u realnom vremenu na izlazu DSP platforme.

Algoritmi koji zahtevaju bilo kakav nasumičan pristup odbircima audio signala, odnosno podrazumevaju rad sa već spremnim datotekama sa audio sadržajem su odbačeni kao nepovoljan izbor. To su algoritmi koji zahtevaju odluke na osnovu dužeg vremenskog konteksta, algoritmi kod kojih odluke o vrsti audio signala u određenom vremenskom trenutku zahtevaju informaciju o obeležjima signala u vremenskom trenutku nakon odabranog trenutka, algoritmi koji odluku donose nakon analize celokupne audio datoteke itd. Ovakvi algoritmi nisu pogodni jer njihova primena ne podrazumeva trenutne odluke zarad obrade signala koji će biti pušten na izlazu.

Kao pogodno rešenje, odabran je algoritam za segmentaciju i klasifikaciju koji koristi činjenicu da procenat okvira sa niskom kratkotrajnom energijom (STE) u okviru većeg segmenta audio signala (LSTER – engl. *Low Short Time Energy Ratio*) ima moć da razlikuje muziku i govor. Konkretno, korišćena je modifikacija ovog algoritma [9], koja koristi parametar δ kako bi prilagodio granicu (*lowthres*) ispod koje se STE signala smatra niskom, tzv. MLER (engl. *Modified Low Energy Ratio*).

STE, MLER i *lowthres* se računaju prema formulama:

$$\begin{aligned} STE &= \sum_{m=0}^{M-1} x(m)^2 \\ MLER &= \frac{1}{2N} \sum_{n=0}^{N-1} [sgn(lowthres - E(n)) + 1] \\ lowthres &= \delta \cdot \frac{\sum_{n=0}^{N-1} E(n)}{N} \end{aligned}$$

gde je M ukupan broj odbiraka u jednom okviru signala, a N broj okvira u jednom segmentu.

Prateći prethodno opisanu strukturu algoritama za segmentaciju i klasifikaciju audio signala, prototip rezultujućeg algoritma je sledeći:

1. računanje STE, *lowthres* i MLER na nivou okvira, radi preliminarne odluke između govora i muzike,
2. odluka o vrsti signala koristeći neki od klasifikatora i
3. donošenje konačne odluke o vrsti signala na osnovu konteksta.

Kako je ovakav algoritam razvijen samo za razlikovanje muzike i govora, potrebno je naći način kako uvesti i diskriminaciju od šuma. Takođe, potrebno je odabrat odgovarajući klasifikator na osnovu tog mehanizma diskriminacije. O ovim problemima će više reći biti u četvrtom poglavlju – Razvoj algoritma za klasifikaciju.

3. Ciljna hardverska arhitektura za rad u realnom vremenu

U ovom poglavlju obrazložena je potreba za digitalnim signal procesorom kao ciljnom platformom, opisane su specifičnosti digitalnih signal procesora i dat je opis konkretnе ciljne platforme korišćene u ovom radu – *Cirrus Logic DSP CS49844*.

3.1 Predlog ciljne hardverske arhitekture

Kao što je već spomenuto, algoritmi za segmentaciju i klasifikaciju audio signala kod ugrađenih audio sistema, mogu se koristiti radi usmeravanja audio signala ka odgovarajućoj završnoj obradi. Primer upotrebe ovog algoritama bi se mogao naći kod audio/video prijemnika (AVR – engl. *Audio/Video Receiver*), TV uređaja, i set-top box uređaja. Kod ovakvih uređaja, u zavisnosti od vrste signala – govor, muzika, pozadinska buka, šum – može se primeniti završna obrada koja bi poboljšala doživljaj emitovanog sadržaja kod korisnika. Obrada može podrazumevati pojačavanje jačine reprodukovanih zvuka u slučaju govora, utišavanje u slučaju pozadinske buke, šuma ili reklama i ekvalizacija (*EQ*) u slučaju muzike.

Algoritmi za digitalnu obradu signala najčešće koriste veliki broj složenih matematičkih operacija, koje se izvršavaju nad blokom podataka određene dužine. U ovu svrhu mogu se koristiti mikrokontroleri i mikroprocesori opšte namene. Međutim, njihove performanse prilikom izvršavanja ovakvih algoritama mogu biti veoma loše, čak i kod algoritama niske složenosti, jer nisu prilagođeni za izvršavanje tih algoritama u realnom vremenu. Prilikom izvršavanja DSP algoritama u realnom vremenu, gde je blok podataka potrebno prihvati, dekodovati, obraditi i pustiti na izlaz u unapred određenom, precizno definisanom vremenskom intervalu, potrebni su procesori koji sadrže proširenja i arhitekturu koji to i omogućuju. Takođe,

pogotovo u uređajima potrošačke elektronike, potrebno je uzeti u obzir i cenu i potrošnju energije.

FPGA (engl. *Field-programmable Gate Array*) platforme, uopšteno daju veliku fleksibilnost pri projektovanju integrisanog kola u određenu svrhu, što ih čini pogodnim i za implementiranje DSP algoritama. Međutim, proizvodnja FPGA platformi, kao i njihova potrošnja energije su veće u poređenju sa drugim procesorima.

ASIC (engl. *Application-Specific Integrated Circuit*) platforme takođe mogu biti upotrebljene, jer omogućavaju implementaciju specifičnu za upotrebu. Međutim, jednom projektovano integrisano kolo se ne može menjati, što onemogućava uvođenje bilo kakvih izmena u implementaciji u slučaju promene delova algoritma.

Digitalni signal procesori se ovde pokazuju kao adekvatno rešenje, ne samo zbog svoje arhitekture koja je prilagođena izvršavanju DSP algoritama u realnom vremenu, već i zato što imaju nisku cenu i potrošnju energije. U odnosu na FPGA, iako manje fleksibilni, digitalni signal procesori znatno su nižih cena nego i troše manje energije. U odnosu na ASIC, digitalni signal procesori imaju mogućnost izmene programa koji se izvršava na njima i omogućavaju njihova redovna ažuriranja.

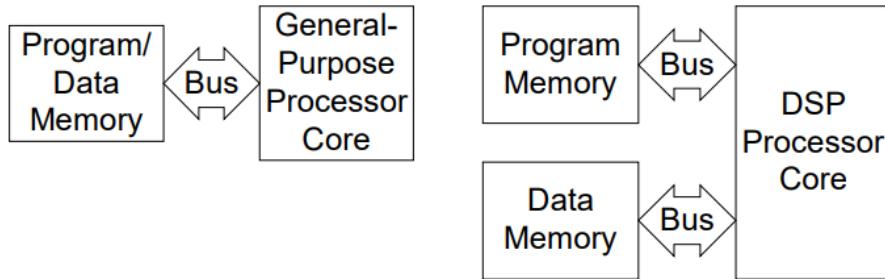
3.2 Karakteristike procesora za digitalnu obradu signala

Još od njihovog nastanka, arhitektura i skup instrukcija digitalnih signal procesora su oblikovani pod uticajem algoritama DSP obrade i oni direktno utiču na njihov razvoj. Elementi arhitekture digitalnih signal procesora u kojima se ogleda njihovo prilagođenje algoritmima DSP obrade su:

- efikasan pristup memoriji,
- množači,
- višestrukost izvršnih jedinica,
- hardverska podrška petljama,
- adresni generatori i
- specijalizovan skup instrukcija.

Radi efikasnog pristupa memoriji, digitalni signal procesori imaju višestruke magistrale za program i podatke, odnosno prate Harvard arhitekturu. Ovo im omogućava istovremeni pristup instrukcijama i podacima iz memorije, što znatno ubrzava njihov rad. S druge strane, mikrokontroleri i mikroprocesori opšte namene često prate Fon Nojmanovu arhitekturu (engl. *Von Neumann*), odnosno imaju jedinstvenu magistralu za program i podatke. To znači da je, za izvršenje instrukcije, potrebno prvo pročitati je iz memorije i dekodirati je, a tek zatim dobaviti operande iz memorije. Digitalni signal procesori često imaju tri magistrale – dve za podatke i

jednu programsку, što omogućava istovremeno čitanje instrukcije i dobavljanje dva operanda iz memorije.



Slika 3.1 Poređenje arhitektura procesora opšte namene (Fon Nojman) i digitalnih signal procesora (Harvard)

Jedna od glavnih operacija koja se pojavljuje u DSP algoritmima je tzv. MAC (engl. *Multiply-Accumulate*), odnosno pomnoži i saberi operacija. Kod drugih procesora, za izvršenje jedne operacije bio bi potreban veliki broj ciklusa, jer je potrebno dobaviti dva operanda, pomnožiti ih i na kraju ih tek sabrati sa akumulatorskim registrom. Operacija sabiranja kod ovih procesora traje nekoliko ciklusa, dok se operacija množenja izvršava kroz niz sabiranja i aritmetičkih pomeraja, što rezultuje velikom utrošku procesorskog vremena. Nasuprot tome, digitalni signal procesori su prilagođeni izvršenju ove operacije i u svom skupu instrukcija obično imaju MAC instrukciju, koja omogućava izvršenje jedne MAC operacije u jednom procesorskom ciklusu.

Kako bi se dalje poboljšale performanse digitalnih signal procesora, oni često imaju više izvršnih jedinica – aritmetičko-logičkih, pomeračkih, MAC. Ovo omogućuje efikasno izvršavanje algoritama koji koriste SIMD (engl. *Single Instruction Multiple Data*) paralelizam, odnosno izvršavanje jednog tipa instrukcije nad višestrukim podacima paralelno.

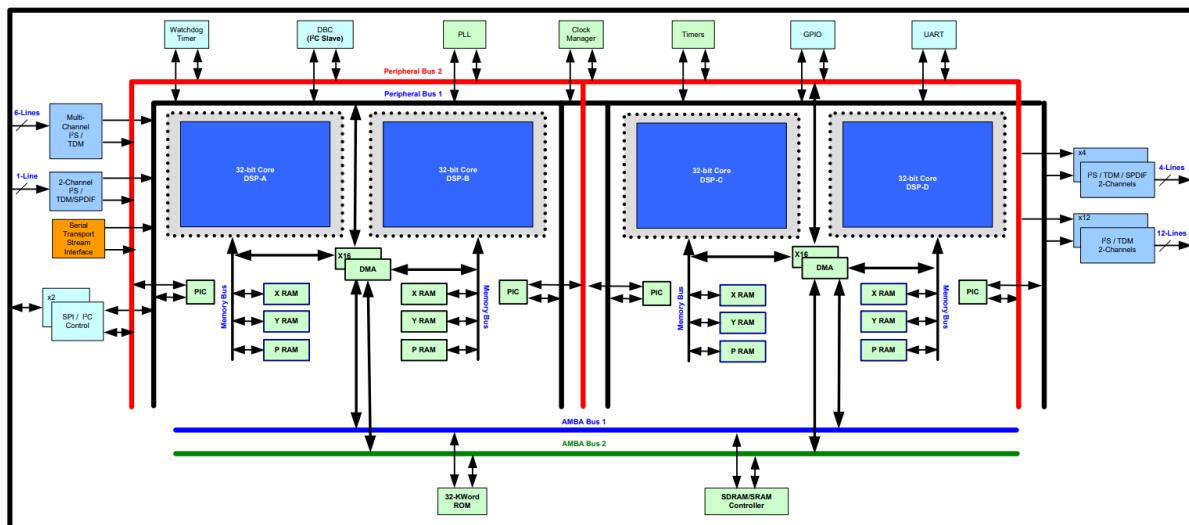
U DSP algoritmima, često je potrebno obavljati neku instrukciju više puta uzastopno, odnosno, instrukcije se izvršavaju u petljama sa unapred poznatim brojem ponavljanja. Digitalni signal procesori sadrže proširenja koja omogućavaju prolazak kroz takve petlje, bez trošenja vremena na uvećavanje brojača petlje ili proveru uslova petlje.

DSP algoritmi obično rade nad velikim brojem podataka, koji se nalaze skladišteni u nizovima. Računanje adrese na kojoj se nalazi podatak često traje veći broj ciklusa, nego što je potrebno obraditi podatak. Radi efikasnijeg pristupa podacima, digitalni signal procesori imaju specijalizovane jedinice za računanje adrese na kojoj se nalazi podatak – AGU, odnosno adresne generatore. Adresni generatori često pružaju i dodatne mogućnosti, kao što su različiti modovi adresiranja – adresiranje po modulu i bit-inverzno adresiranje.

Digitalni signal procesori su razvijani s ciljem maksimalne moguće iskorišćenosti dostupnih hardverskih resursa i smanjenjem količine potrebne memorije za skladištenje DSP programa. Da bi se postiglo prvo, digitalni signal procesori poseduju specifičan skup instrukcija. On nudi instrukcije koje se mogu izvršavati paralelno, na višestrukim izvršnim jedinicama (npr. prihvatanje/skladištenje dva ili više podatka, izvršenje dve ili više aritmetičke operacije i inkrement pokazivača na podatke). S drugim ciljem na umu, ove instrukcije se održavaju kratkim, ograničavanjem skupa i kombinacija registara koji se mogu koristiti pri ovim paralelnim izvršenjima.

3.3 Arhitektura procesora CS49844

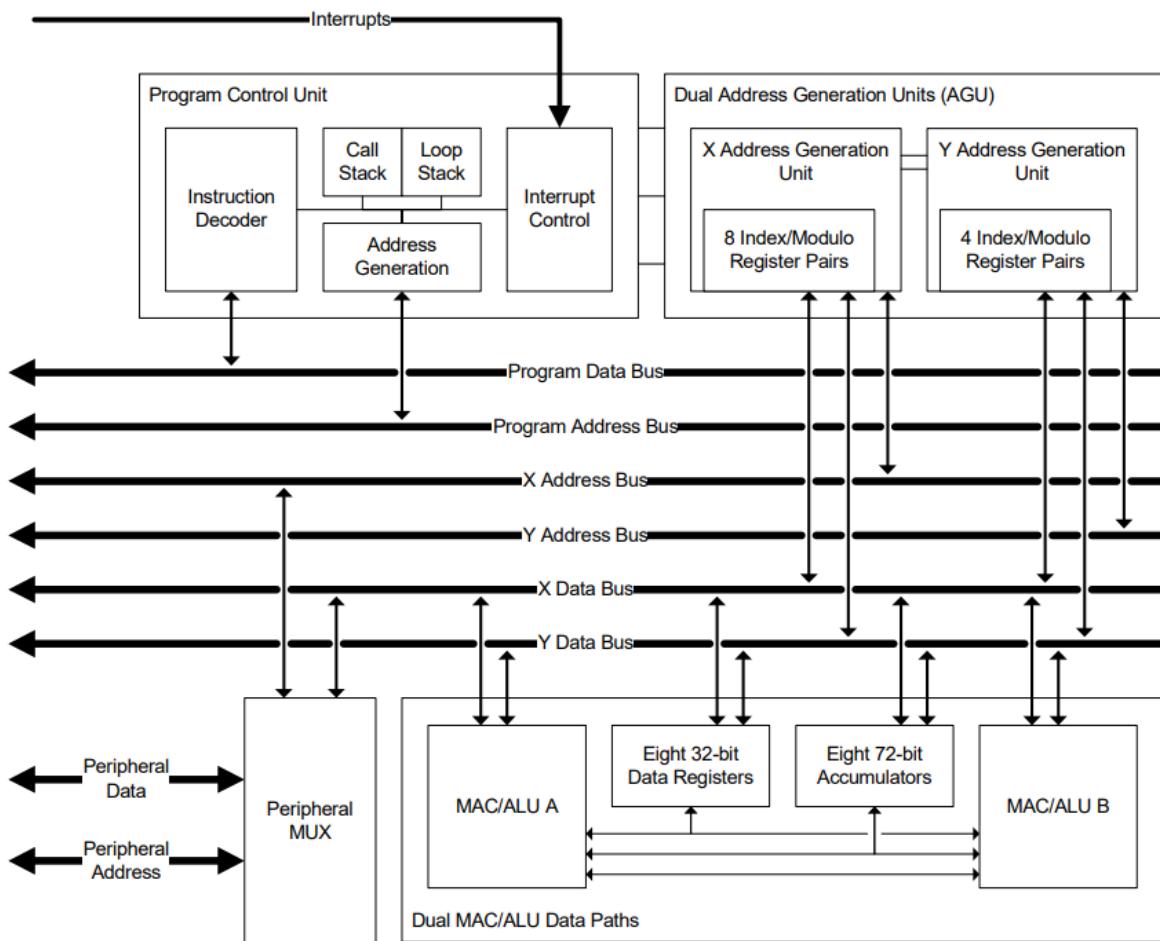
U ovom radu, kao ciljna platforma korišćen je digitalni signal procesor CS49844, koji se nalazi na razvojnoj ploči firme *Cirrus Logic* sa oznakom CDB49x. Ovaj procesor sadrži četiri 32-bitna DSP jezgra koji rade u nepokretnom zarezu, od kojih je svako u mogućnosti da izvrši do čak 6 operacija u jednom ciklusu (dve MAC operacije, dva skladištenja u memoriju ili čitanja iz memorije i dva inkrementa/dekrementa indeksnih registara).



Slika 3.2 Blok dijagram procesora CS49844

Jezege su tzv. *Cirrus Logic* 32-bitne arhitekture (unapređena Harvard arhitektura) i imaju zasebno po dve memorije za podatke – X i Y, kao i jednu programsku memoriju – P. Takođe, logički postoji i L segment memorije, koji je namenjen podacima kod kojih je pogodno deo podataka čuvati u X, a deo u Y memoriji i pristupiti im istovremeno. Svaki od memorijskih prostora X, Y i P na svakom od jezgara je veličine 60 000 reči.

U svakom jezgru postoje dva paralelna toka podataka, odgovarajući adresni generatori i programska kontrolna jedinica. Jedinice za generisanje adresa sadrže po dvanaest indeksnih registara ($i0, i1, \dots, i11$) sa odgovarajućim modulo registrima ($nm0, nm1, \dots, nm11$), koji nude različite adresne režime – linearni, modulo, bit-inverzni.



Slika 3.3 Cirrus Logic 32-bitna arhitektura

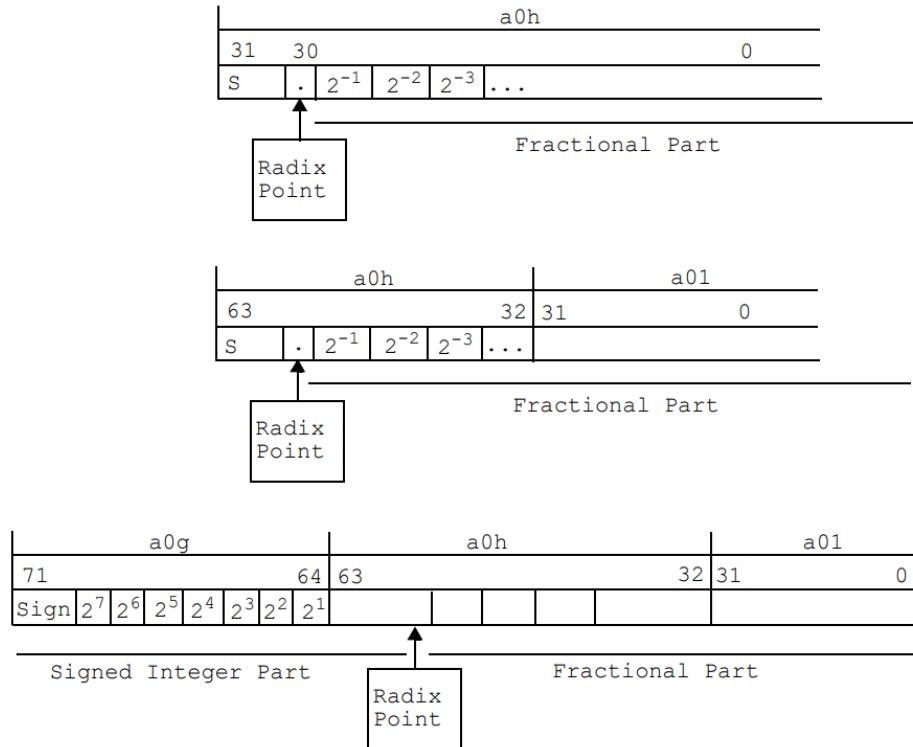
Svaki od tokova podataka ima osam 32-bitnih registara za podatke, četiri X (x_0, x_1, x_2, x_3) i četiri Y (y_0, y_1, y_2, y_3), kao i osam 72-bitnih akumulatorskih registara ($a_0, a_1, a_2, a_3, b_0, b_1, b_2, b_3$). Akumulatori služe za skladištenje međurezultata i sastoje se iz 32-bitnog *low*, 32-bitnog *high* i 8-bitnog *guard* dela, kako bi rezultati aritmetičkih operacija 32-bitnih brojeva stali u taj registar, gde *guard* deo služi ukoliko rezultat izađe van opsega. Svakom delu akumulatorskog registra može se pristupiti zasebno.

Tu se nalaze i MAC, SRS (jedinica za pomeranje, zaokruživanje i zasićenje) i aritmetičko-logičke jedinice (ALU). SRS jedinica služi kao sprega ka magistralama podataka, gde se prolazom kroz nad 72-bitnim rezultatom vrši odgovarajuća obrada (pomeranje, zaokruživanje, pa zasićenje, tim redom), ukoliko je to potrebno, pre slanja na 32-bitnu magistralu podataka. Rad SRS jedinice se podešava pomoću posebnog mod registra (*mr*). ALU obavlja logičke operacije nad akumulatorskim registrima.

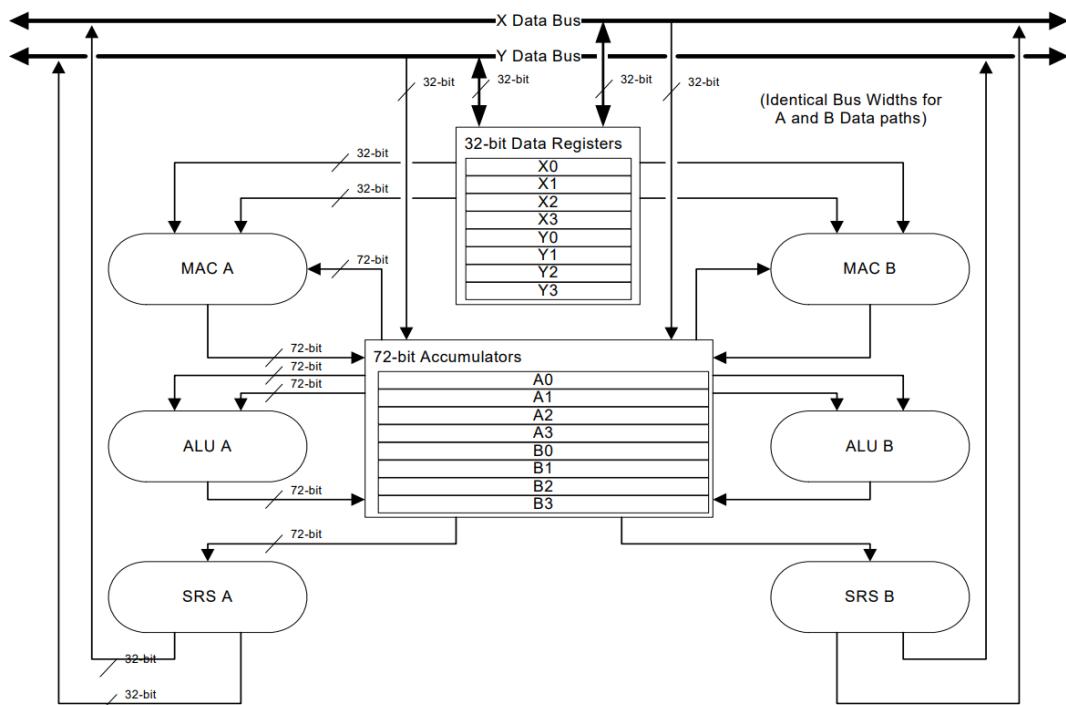
Prikazi brojeva su u nepokretnom zarezu, u drugom komplementu. Podaci mogu sledećim prikazima:

- 32-bitni prikaz <1.31>: opseg brojeva od -1.0 (0x80000000) do $1 \cdot 2^{-31}$ (0x7fffffff),

- 64-bitni prikaz <1.63>: opseg brojeva od -1.0 (0x80000000 00000000) do $1 \cdot 2^{-63}$ (0x7fffffff ffffffff) i
- 72-bitni prikaz <9.63>: opseg brojeva od -256.0 (0x80 00000000 00000000) do $256 \cdot 2^{-63}$ (0x7f ffffffff ffffffff).



Slika 3.4 Prikazi brojeva procesora CS49844



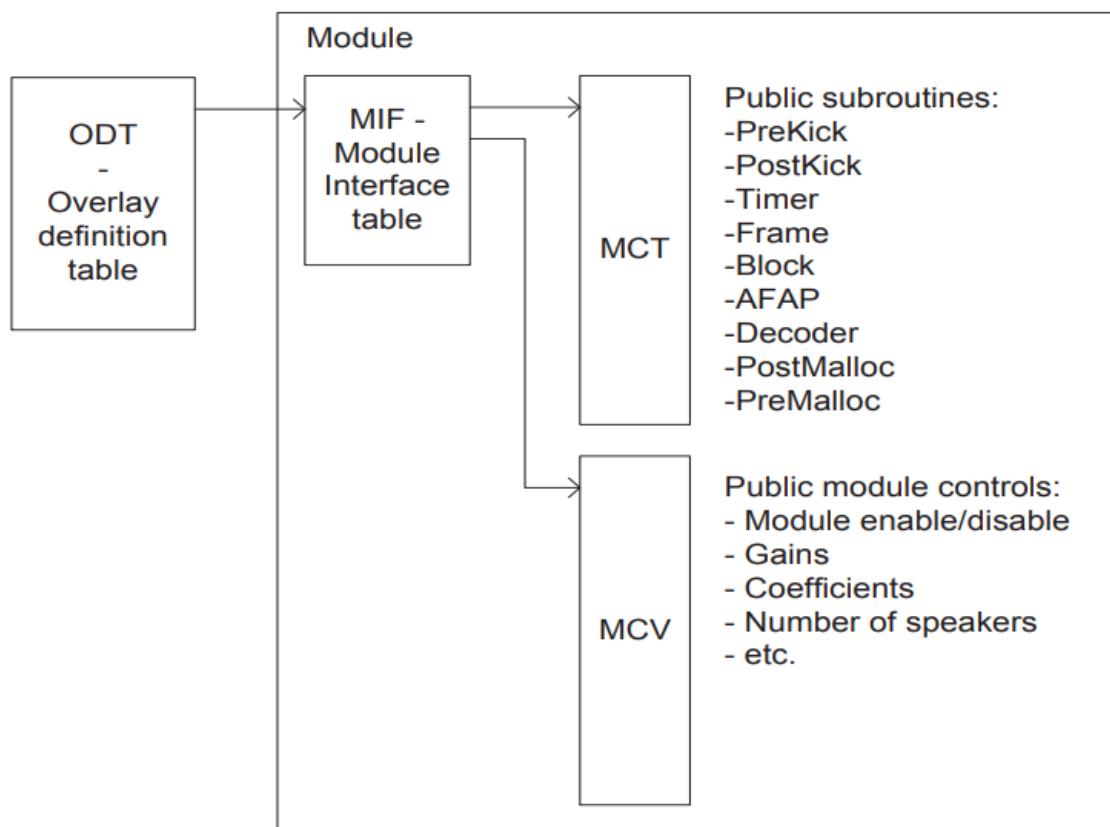
Slika 3.5 Tok podataka i akumulatorska jedinica jezgra

3.4 Sistemska programska podrška procesora CS49844

Cirrus Logic radni okvir (engl. *framework*) predstavlja sistemsku programsku podršku procesora. Jezgro ovog radnog okvira je jednostavan operativni sistem (OS), čija je glavna uloga raspoređivanje procesnih entiteta. On radi tako što zapravo predstavlja monitorsku petlju koja proziva rutine učitanih modula po određenom redosledu.

Osnovne komponente sistemske programske podrške su moduli. Svaki modul se sastoji od rutina i podataka i ima svoj jedinstven sprežni podsistem (MIF – engl. *Module Interface*) kojim se povezuje sa OS. Njega čini MIF tabela koja sadrži pokazivače na tabele sa ostalim sprežnim informacijama, među kojima su i MCT (engl. *Module Call Table*) i MCV (engl. *Module Control Vector*) tabele. OS, s druge strane, spregu prema modulima ostvaruje kroz ODT (engl. *Overlay Definition Table*) tabelu, koja sadrži pokazivače na MIF tabele svih učitanih modula. Datoteke modula koje sadrže mašinske instrukcije i mogu se izvršavati imaju ekstenziju *.uld*.

MCT tabela sadrži pokazivače na osnovne javne rutine modula, po unapred definisanom redosledu. Ove rutine OS proziva kao odgovor na pojavu odgovarajućih događaja u sistemu. MCV tabela sadrži javno dostupne konfiguracione parametre datog modula. Strukturu i sadržaj ove tabele definiše sam programer modula.



Slika 3.6 Blok dijagram sprežnog sistema modula i OS

4. Razvoj algoritma za segmentaciju i klasifikaciju audio signala

U ovom poglavlju, opisan je tok formiranja strukture algoritma na osnovu opisanog prototipa i određivanje parametara algoritma za segmentaciju i klasifikaciju audio signala uzimajući u obzir ograničenja ciljne platforme i dat je opis njegove konačne strukture. Takođe, opisan je tok izrade referentnog koda na osnovu takve strukture algoritma.

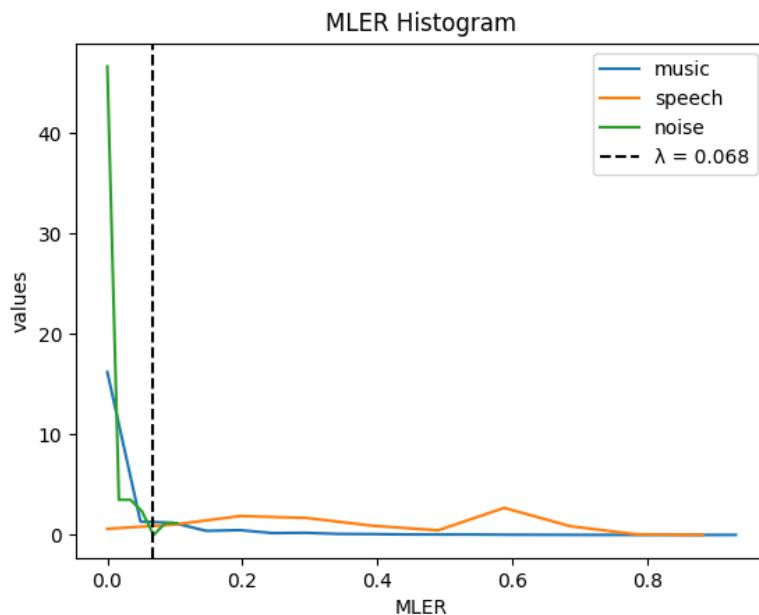
4.1 Tok razvoja algoritma za segmentaciju i klasifikaciju audio signala

Nakon prethodno određenog prototipa algoritma koji može da razlikuje samo između signala govora i muzike, potrebno je taj prototip proširiti mogućnošću da razlikuje i šum od govora i muzike. Takođe, potrebno je i prilagoditi određene delove algoritma ograničenjima ciljne platforme i izvršavanju u realnom vremenu.

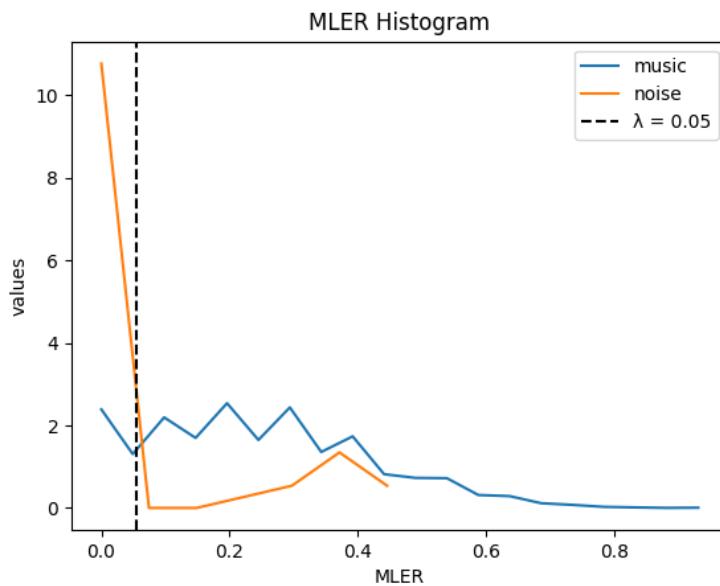
4.1.1 Diskriminacija između muzike, govora i šuma

Analiza MLER za svaki od tipova signala urađena je koristeći programski jezik *Python* i razvojno okruženje *PyCharm*. Računjem MLER za različite parametre δ i analizom rezultata, utvrđeno je da je moguće uraditi i diskriminaciju šuma od muzike i govora.

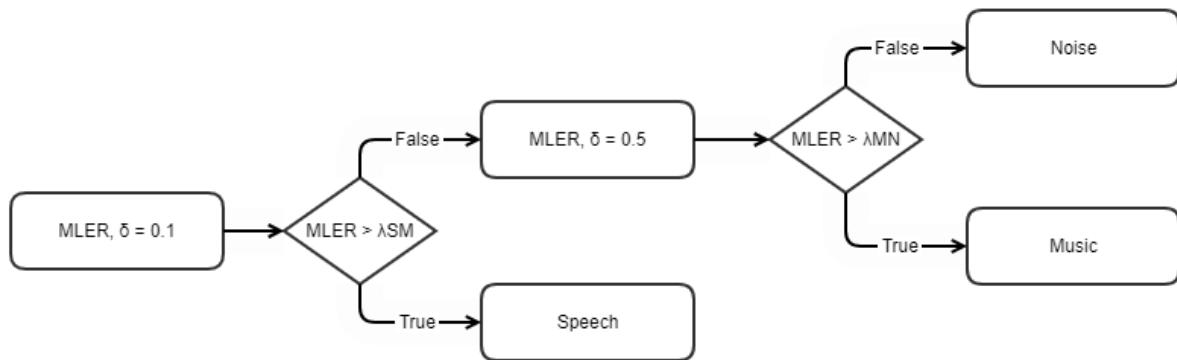
Za vrednost parametra $\delta = 0.1$, ukoliko je vrednost MLER iznad empirijski utvrđene granice $\lambda_{SM} = 0.068$, vrsta signala se označava kao govor. Ukoliko je vrednost ispod te granice, potrebno je odrediti vrednost MLER za $\delta = 0.5$. Za tako određen MLER, ukoliko je njegova vrednost iznad empirijski utvrđene granice $\lambda_{MN} = 0.05$, vrsta signala se označava kao muzika, odnosno šum ukoliko je ispod te vrednosti. Ovim se dobijaju dva diskriminatora – diskriminator govora od muzike/šuma i diskriminator muzike od šuma.



Slika 4.1 Histogram MLER za $\delta = 0.1$



Slika 4.2 Histogram MLER za $\delta = 0.5$

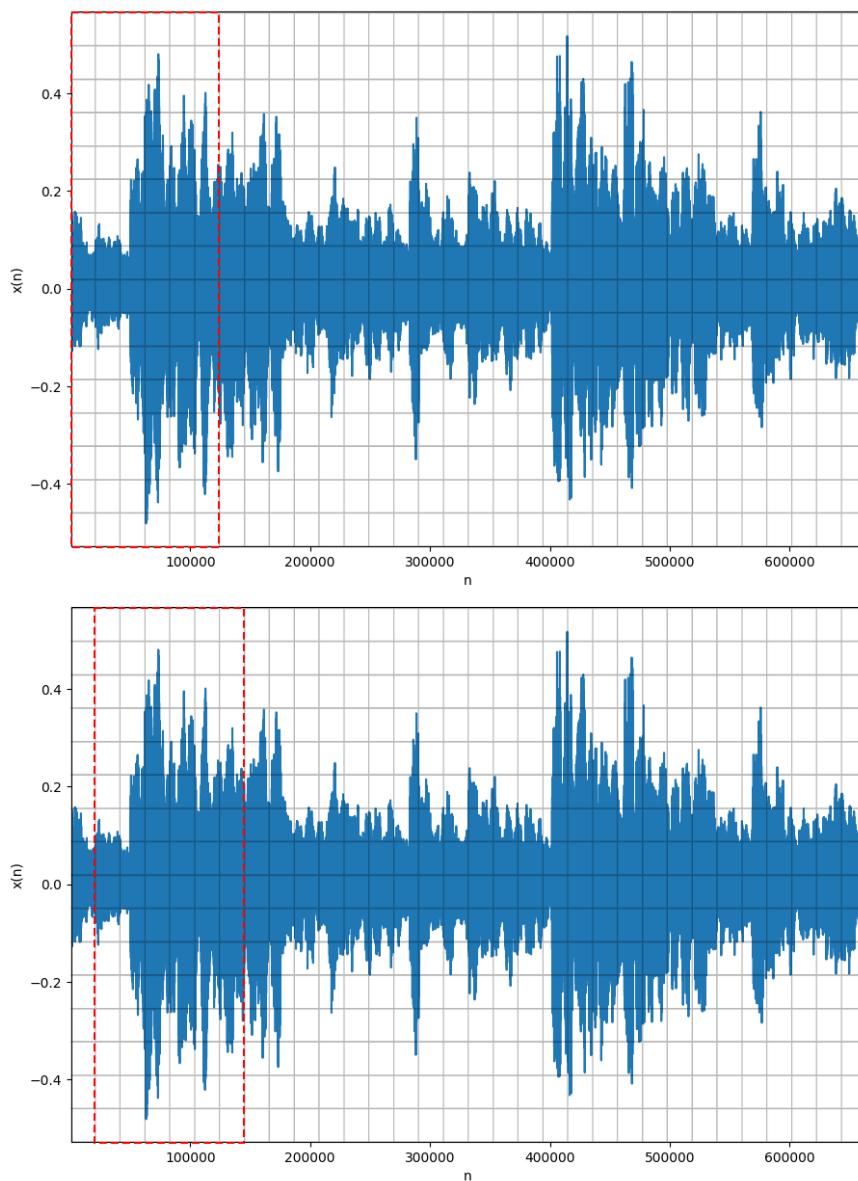


Slika 4.3 Dijagram toka odluke o vrsti signala

4.1.2 Prilagođavanje algoritma izvršavanju na ciljnoj platformi

Prvobitni algoritam je zamišljen da radi tako što prikuplja jedan ceo segment podataka dužine 1-1.5s, gde se računa MLER za svaki od okvira (obično dužine od 12ms do 25ms), a zatim donosi odluku o vrsti signala na tom delu segmenta. U zavisnosti od frekvencije odabiranja signala, ovo znači da bi trebalo obrađivati blok od po nekoliko desetina hiljada pa do nekoliko stotina hiljada odbiraka, odjednom. Kako bi se ovaj problem rešio, algoritam je izmenjen tako da radi principom klizećeg prozora.

Slika 4.4 prikazuje princip rada klizećeg prozora. U ovom primeru, klizeći prozor je dužine šest okvira i pravi korak dužine jednog okvira.



Slika 4.4 Primer klizećeg prozora

Umesto obrađivanja susednih segmenata zasebno, segment se pomera za korak dužine jednog okvira. Ova modifikacija omogućava da se parametar $lowthres$ za trenutni segment k računa tako što se oduzme sabirak koji uključuje STE najstarijeg okvira i doda sabirak koji uključuje STE najnovijeg okvira. Ovim se broj sabiranja potrebnih za računanje parametra $lowthres$ smanjuje sa L (broj okvira u jednom segmentu) na samo dva. Takođe, ovakva modifikacija uvodi rezoluciju odluke o vrsti signala na nivou okvira, iako se zasniva na informaciji o celom segmentu.

$$lowthres(k) = lowthres(k - 1) - \delta \cdot \frac{E(k - 1)}{N} + \delta \cdot \frac{E(k)}{N}$$

Vrednosti STE koje se čuvaju za računanje MLER na nivou segmenta su uvek vrednosti STE za poslednjih N okvira, gde je N broj okvira koji čini jedan segment. Princip klizećeg prozora sa korakom veličine jednog okvira omogućava da se skladištenje STE za računanje $lowthres$ implementira pomoću kružnog bafera, gde bi se uvek najnovija vrednost STE prepisivala preko najstarije u baferu.

4.1.3 Klasifikacija i konačna odluka na osnovu konteksta

Konačna odluka o vrsti signala uvedena je kao poslednji korak algoritma kako bi se poboljšao njegov učinak. Istraživanja iz ove oblasti ukazuju na to da je verovatnoća smene između vrsta audio signala veoma mala u kontinualnom toku audio podataka, odnosno, audio signal jedne vrste traje obično nekoliko sekundi, pre nego što se pojavi audio signal druge vrste. Dakle, potrebno je ukloniti kratkotrajne prelaze između vrste signala, koji su često rezultat greške prethodnog koraka algoritma – kako na nivou segmenta, tako i na nivou više segmenata.

Prvo, potrebno je ukloniti nagle tranzicije na nivou trajanja jednog segmenta. Ovo je urađeno uvođenjem brojača vrste okvira. Svakim pomerajem za jedan korak veličine jednog okvira, određuje se vrsta signala na osnovu izračunatog MLER, po prethodno opisanom mehanizmu odluke. Potom, uvećava se brojač za vrstu signala koja je određena na osnovu MLER, dok se za ostale vrste brojač smanjuje. Odluka se dalje donosi pravilom većine – na nivou tog segmenta, vrstom signala se proglašava ona čiji je brojač okvira maksimalan. Maksimalna vrednost brojača se može odrediti eksperimentalno, obično tako da maksimalna vrednost brojača okvira rezultuje trajanjem od 0.3-0.5s.

Kako bi se rešio problem smene vrste signala uzimajući u obzir kontekst, odnosno, kako bi se uklonile česte tranzicije signala između više susednih segmenata, uveden je krajnji korak odluke na osnovu konteksta. Ovo je urađeno tako što je uvedena memorija stanja, koja se ažurira na osnovu odluke prethodnog koraka, na sledeći način: ukoliko je vrsta signala za trenutni okvir jednaka vrsti signala za prethodni okvir, brojač stanja se uvećava, a kao finalna odluka ostaje trenutna vrsta signala. U suprotnom, ukoliko je došlo do promene vrste signala

okvira, brojač stanja se smanjuje i trenutno stanje se proglašava konačnom odlukom, sve dok brojač ne padne na nulu. Kada brojač dođe do nule, smatra se da je zaista došlo do tranzicije i novim konačnim stanjem se proglašava vrsta signala koja je određena kao izlaz iz prethodnog koraka. Ovaj brojač stanja je ograničen na takvu vrednost, da se dozvoli tranzicija vrste signala nakon nekoliko sekundi (obično 3-5s).

Ovaj mehanizam unosi maksimalno kašnjenje odluke o vrsti signala, u trajanju od maksimalne vrednosti brojača okvira uvećane za veličinu memorije stanja.

4.2 Konačna struktura algoritma

U konačnoj strukturi algoritma uočavaju se dve faze: početna – inicijalizacija i glavni deo algoritma – segmentacija i klasifikacija.

Da bi se opisala struktura algoritma, potrebno je definisati parametre modela. Parametre modela opisuje Tabela 4.1:

Parametar	Opis
Fs	Frekvencija odabiranja audio signala.
FRAME_SIZE	Veličina okvira; broj odbiraka koji čini jedan okvir.
NO_FRAMES	Veličina segmenta; broj okvira koji čine jedan segment
COUNTER_LIMIT	Maksimalna vrednost brojača vrste okvira.
STATE_LIMIT	Maksimalna vrednost brojača stanja.
λ_{SM}	Granica za razlikovanje govora od muzike/šuma.
λ_{MN}	Granica za razlikovanje muzike od šuma.

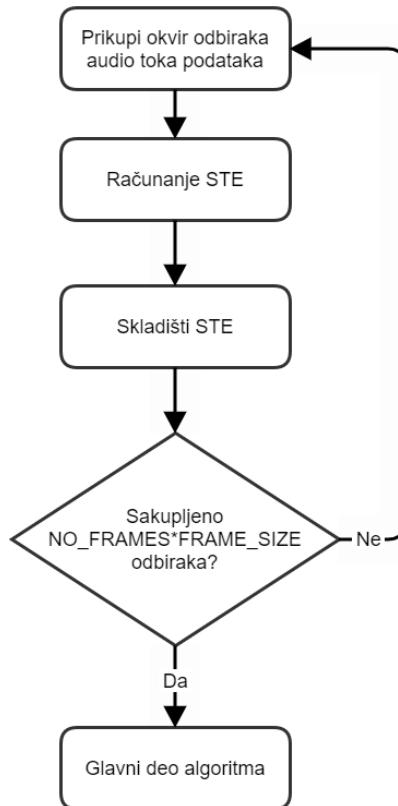
Tabela 4.1 Opis parametara modela

Zbog rada u realnom vremenu, potrebno je uraditi inicijalizacioni korak pre prelaska na glavni deo algoritma. Prvi segment podataka ostaće neobrađen, tj. za njega će biti samo izračunata obeležja (STE, MLER, *lowthres*), koja će biti potrebna radi odluke o vrsti signala na sledećem segmentu, i biće samo propušten na izlaz. Tek nakon što budu izračunata obeležja za prvih NO_FRAMES*FRAME_SIZE.

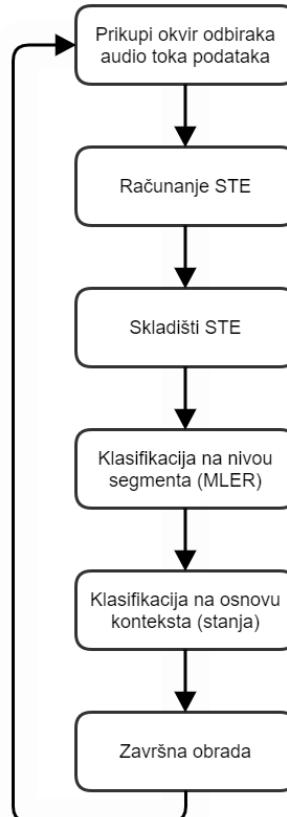
Dalje, glavni deo algoritma teče isto kao inicijalizacioni deo, osim toga što se ne proverava da li je sakupljeno dovoljno odbiraka za jedan ceo segment.

Na osnovu prethodno skladištenih i novoizračunatog STE, potrebno je ažurirati vrednost *lowthres* za oba diskriminatora, za diskriminator govora od muzike/šuma i za diskriminator muzike od šuma. Sa tako ažuriranim vrednostima *lowthres*, računaju se vrednosti MLER za odgovarajuće vrednosti δ . Sledi odluka o vrsti signala poređenjem vrednosti MLER sa odgovarajućim granicama λ_{SM} i λ_{MN} , po prethodno opisanom mehanizmu odluke, uz ažuriranje odgovarajućih brojača okvira vrste signala.

Konačan korak je klasifikacija na osnovu konteksta. Ovde se vrši ažuriranje brojača stanja i donošenje konačne odluke o vrsti signala.



Slika 4.5 Dijagram toka inicijalnog dela algoritma



Slika 4.6 Dijagram toka glavnog dela algoritma

4.3 Završna obrada signala

Nakon određivanja vrste audio signala, odbirci signala se prosleđuju na odgovarajuću završnu obradu. U ovom radu, na odbirke obeležene kao signal govora primenjeno je pojačanje od oko 6dB (linearno pojačanje 2 puta), na odbirke označene kao šum primenjeno je utišanje u vrednosti oko -6dB (linearno utišanje 2 puta), dok je kod odbiraka označenih kao muzika održena kontrola basa pomoću niskopropusnog *shelving* filtra.

4.4 Programska implementacija algoritma na PC platformi

Kako bi se odredili parametri algoritma kao što su veličina okvira i segmenta, veličine memorije stanja i maksimalna vrednost brojača okvira po vrsti signala, kao i performanse algoritma, algoritam je prvo implementiran za izvršavanje na PC platformi. Referentni kod algoritma segmentacije i klasifikacije za izvršavanje na PC platformi implementiran je prvo u programskom jeziku *Python* radi određivanja parametara i performansi algoritma, a zatim i u *C* programskom jeziku radi lakšeg prelaska na asemblerски jezik.

4.4.1 Programska implementacija u C programskom jeziku

Programsko okruženje korišćeno za razvoj *C* koda je *Microsoft Visual Studio 2019*. Ovo programsko okruženje, osim uređivača teksta, pruža i mogućnost kontrolisanog izvršavanja programa (engl. *debug*).

Glavne promenljive algoritma (brojači okvira vrste signala, stanje, MLER i *lowthres* za oba diskriminatora) se čuvaju u *C* strukturi tipa *detect_state_t*. U istoj strukturi se čuvaju i brojači odbiraka (po modulu broja odbiraka unutar jednog okvira – *FRAME_SIZE*), kako bi algoritam znao da li je učitano dovoljno blokova odbiraka za ceo okvir. Takođe, postoji i jedan brojač okvira. Kada ovaj brojač dosegne vrednost *NO_FRAMES*, algoritam prelazi s inicijalnog koraka na glavni deo.

```
typedef struct
{
    int sample_rate;
    int frame_size; // in samples
    int segment_size; // in samples

    double mler_sm; // mler for sound/music discrimination
    double mler_mn; // mler for music/noise discrimination
    double th_sm; // low threshold for sound/music discrimination
    double th_mn; // low threshold for music/noise discrimination
    int ptr_mste; // current index in mste buffer

    int init_frame_count; // frames to be processed for stats before making decision for the first time

    // number of samples in the past segment for each type
    int speech_samples;
    int music_samples;
    int noise_samples;

    // type of signal for segment-level context and inter-segment level context
    SIGNAL_TYPE signal_type_prelim;
    SIGNAL_TYPE signal_type_final;

    int state_counter; // keep track of state changes during context level decision
} detect_state_t;
```

Slika 4.7 Kontrolna *C* struktura stanja algoritma

Vrednosti STE se čuvaju u kružnom baferu dužine broja okvira u jednom segmentu. U inicijalnom koraku algoritma, u bafer će se upisivati izračunate vrednosti STE. Na kraju inicijalnog koraka, bafer će biti popunjeno. Dalje, u glavnom delu algoritma, najstarija vrednost se iščitava iz bafera radi računanja *lowthres*, a najnovije izračunata vrednost STE će biti upisana na njeno mesto.

Slika 4.8 prikazuje hijerarhiju funkcija u C modulu. Funkcije unutar modula će biti opisane u nastavku.

- main
 - detection_init
 - module_main
 - update_mler
 - segment_level_detection
 - our_processing
 - shelving_lp
 - first_order_IIR

Slika 4.8 Hijerarhija funkcija u C modulu

4.4.1.1 Funkcija main

- int main(int argc, char* argv[])

U glavnoj funkciji programa prvo se poziva funkcija za inicijalizaciju strukture `detect_state_t`. U glavnoj petlji se učitavaju odbirci signala u ulazno-izlazni bafer, u blokovima veličine `FRAME_SIZE` iz tokova audio podataka `.wav` (engl. *Waveform Audio File Format*) formata. Nakon svakog učitanog bloka, poziva se funkcija koja implementira algoritam segmentacije i klasifikacije (`module_main`). Nakon povratka iz te funkcije, poziva se funkcija za završnu obradu signala (`out_processing`) i tako obrađeni odbirci signala se upisuju u izlaznu `.wav` datoteku.

4.4.1.2 Funkcija detection_init

- void detection_init()

U ovoj funkciji vrši se inicijalizacija polja strukture `detect_state_t`.

4.4.1.3 Funkcija module_main

- void module_main(double* p_in_buffer, double* p_mste, int ch)

Funkcija koja implementira algoritam prvo poziva funkciju `update_mler` u kojoj se računaju STE i ažuriraju vrednosti *lowthres* i MLER. Ova funkcija takođe prati broj obrađenih odbiraka. Ukoliko je on manji od `NO_FRAMES*FRAME_SIZE`, to znači da je algoritam u

inicijalnoj fazi i ova funkcija se završava. Ukoliko nije, to znači da je algoritam u glavnoj fazi i da može da se pozove funkcija za klasifikaciju na nivou segmenta (segment_level_detection). Po povratku iz nje, brojači vrste signala su ažurirani. Dalje, vrši se klasifikacija na nivou konteksta po opisanom pravilu i proglašava se konačna odluka o vrsti signala.

4.4.1.4 Funkcija update_mler

- `void update_mler(double* p_in_buffer, double* p_mste, detect_state_t* state)`

U ovoj funkciji prvo se vrši inkrementalno računanje STE, a zatim se proverava brojač odbiraka. On se prvo povećava za veličinu bloka učitanih odbiraka (16). Ukoliko nije dosegao FRAME_SIZE, funkcija se završava. Ukoliko jeste, to znači da je računanje STE za jedan okvir završeno, brojač se obara na nulu i sledi ažuriranje vrednosti *lowthres*. Nakon toga, računaju se odgovarajuće vrednosti MLER i sledi povratak iz funkcije.

4.4.1.5 Funkcija segment_level_detection

- `SIGNAL_TYPE segment_level_detection(detect_state_t* state)`

U funkciji segment_level_detection, implementiran je klasifikator na osnovu MLER, λ_{SM} i λ_{MN} i ažuriraju se brojači okvira vrste signala, prema prethodno opisanom pravilu u zavisnosti od vrste signala koja je određena ovim klasifikatorom.

4.4.1.6 Funkcija out_processing

- `void out_processing(double* p_in_buffer, detect_state_t* state)`

Funkcija koja služi za završnu obradu signala. U zavisnosti od vrste signala ili će samo primeniti odgovarajuće pojačanje nad odbircima u ulazno-izlaznom baferu (u slučaju govora i šuma), ili će izvršiti filtriranje odbiraka *shelving* filtrom, u slučaju muzike.

4.4.1.7 Funkcija shelving_lp

- `double shelving_lp(double input, double k)`

Funkcija shelving_lp je pomoćna funkcija koja implementira niskopropusni *shelving* filter, pomoću funkcije first_order_iir, koji omogućava kontrolu basa.

4.4.1.8 Funkcija first_order_IIR

- `double first_order_IIR(double input, double* coefficients, double* x_history, double* y_history)`

Ova funkcija služi kao pomoćna funkcija shelving_lp i implementira filter sa beskonačnim impulsnim odzivom (IIR) prvog reda.

5. Programska implementacija u asemblerskom jeziku ciljne platforme

U ovom poglavlju opisan je proces programske implementacije u asemblerskom jeziku za *Cirrus Logic* DSP. Ovaj proces je obuhvatao dve faze:

1. implementacija modula za samostalno izvršavanje u simulatorskom okruženju i
2. prilagođavanje dobijenog asemblerskog koda radu operativnog sistema i povezivanje sa radnim okruženjem.

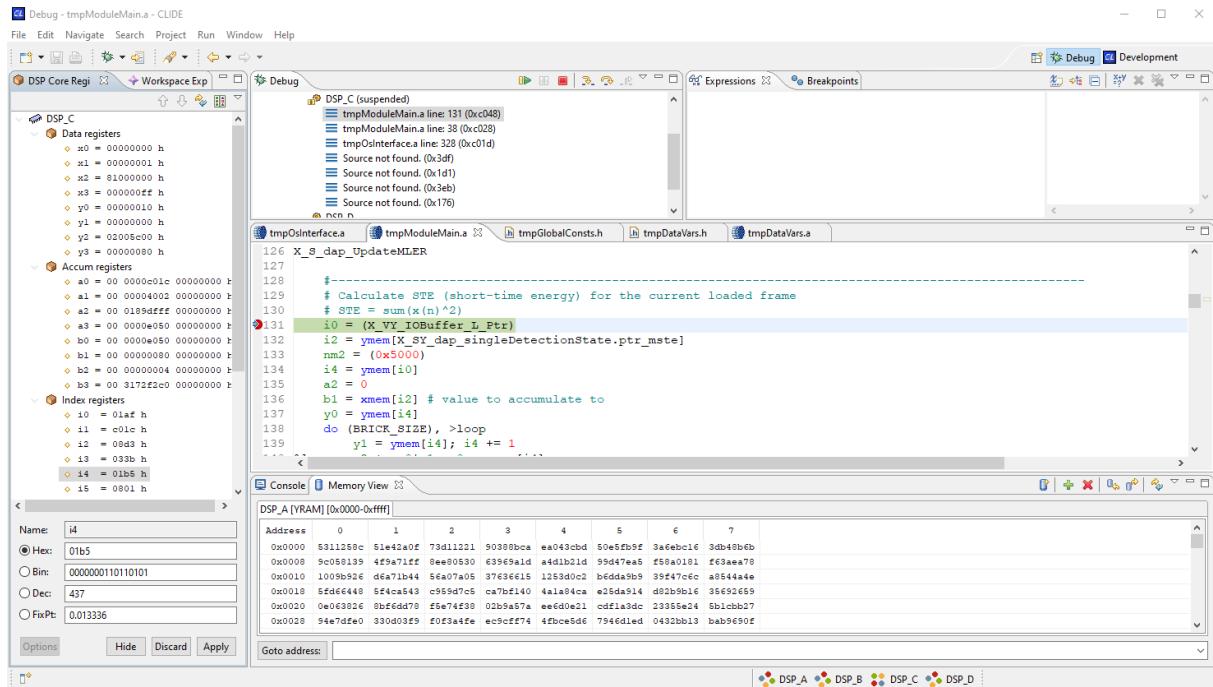
U prvoj fazi, referentni *C* kod je implementiran u asemblerskom jeziku, uzimajući u obzir ograničenja platforme kao što su opseg vrednosti, predstava brojeva i ograničenost dostupnih operacija. U ovoj fazi je i urađena optimizacija koda.

U drugoj fazi, kod dobijen kao rezultat prve faze je prilagođen radu operativnog sistema i implementirane funkcije su raspoređene na odgovarajuće rutine operativnog sistema.

5.1 Razvojno okruženje

Za razvoj asemblerskog koda korišćen je CLIDE (*Cirrus Logic IDE*), integrisano razvojno okruženje zasnovano na *Eclipse* platformi, za razvoj programske podrške za digitalne signal procesore firme *Cirrus Logic*.

CLIDE obezbeđuje uređivač teksta za *C* i asemblerski kod, mogućnost prevođenja aplikacije i podršku za izvršavanje DSP aplikacija u simulatorskim uslovima, kao i na ciljnoj platformi. Takođe, CLIDE sadrži alate za pregled registara i pregled memorije, alat za profilisanje DSP aplikacije i podršku za kontrolisano izvršavanje programa.



Slika 5.1 Primer kontrolisanog izvršavanja u CLIDE razvojnom okruženju

5.2 Implementacija modula za samostalno izvršavanje

Asemblerski kod za samostalno izvršavanje na simulatoru je implementiran s ciljem da se dobije asemblerski kod funkcionalno ekvivalentan referentnom C kodu, kako bi se moglo preći na sledeći korak uvezivanja sa radnim okruženjem i operativnim sistemom radi izvršavanja na ciljnoj platformi. Rešenje je realizovano za jednokanalni audio sadržaj.

Prvobitno, deklarisana je struktura `detection_state_struct`, sa poljima ekvivalentnim C strukturi `detect_state_t` iz referentnog koda, kao i ostali ekvivalentni baferi i definisane su ekvivalentne konstante.

Slika 5.2 Kontrolna struktura stanja algoritma u asemblerskom jeziku

Usledila je implementacija asemblerskih funkcija ekvivalentnih C funkcijama iz referentnog koda. U ovoj fazi je bio akcenat na rešavanju problema koji dolaze uz specifičnosti DSP platforme u odnosu na opštenamenski procesor.

Zbog prikaza brojeva na platformi, određene konstante je bilo potrebno skalirati na opseg [-1.0, 1-2⁻³¹]. Takođe, kako je maksimalna vrednost STE jednaka dužini okvira, i ovu vrednost je bilo potrebno skladištiti skaliranu nakon računanja. Vrednost *lowthres* za diskriminaciju govora od šuma takođe može izaći van tog opsega, tako da je i ona skladištena skalirana na odgovarajuću vrednost.

U ovom koraku izvršena je i optimizacija asemblerskog koda. Gde god je to bilo moguće, korišćena je mogućnost paralelnog izvršavanja instrukcija. Takođe, sve petlje iz programskog jezika C implementirane su kao hardverske petlje u asemblerskom jeziku.

```
# Sample is in y0
AnyReg(y1, y0) # Save original input
b0 = 0 # Output accumulator
i0 = (X_BX_ShelvingCoeffs)
i5 = (X_BY_HistoryX_C)
x0 = xmemp[i0]; i0 += 1 # x0 -> coeff[0]
b0 += x0*y0; y0 = ymem[i5]; i5 += 1; x0 = xmemp[i0]; i0 += 1 # y0 <= histx, x0 <= coeff[1], b0 = input * coeff[0]
# i5 -> histy, i0 -> coeff[2]
b0 += x0*y0; y0 = ymem[i5]; i5 -= 1; x0 = xmemp[i0]; # y0 <= histy, x0 <= coeff[2], b0 = histx * coeff[1]
# i5 -> histx
b0 -= x0*y0; ymem[i5] = y1; i5 += 1 # b0 = histy * coeff[2], histx <= input
# i5 -> histy
ymem[i5] = b0 # histy <= accum
```

Slika 5.3 Primer paralelnog izvršavanja instrukcija kod implementacije IIR filtra

5.3 Povezivanje sa radnim okruženjem

Nakon što je potvrđeno da je asemblerski kod za samostalno izvršavanje u simulatorskom okruženju funkcionalno ispravan i daje izlaze kao referentni C kod, usledio je korak uvezivanja sa radnim okruženjem i operativnim sistemom.

U glavnoj asemblerskoj datoteci, implementirane su potrebne funkcije za spregu s operativnim sistemom, dok su rutine algoritma implementirane u posebnoj datoteci.

Ostvarenje sprege je urađeno, pre svega, popunjavanjem MCT tabele pokazivačima na rutine operativnog sistema. Korišćene su *Pre-Kickstart* rutina, *Block* rutina i *Background* rutina. Zatim, definisana je MCV tabela sa dva polja – polje za omogućavanje rada modula i polje za kontrolu jačine pojačanja basa.

Pre-Kickstart rutinu poziva OS nakon prijema inicijalizacione poruke (*reset*) sistema. U njoj su inicijalizovana polja MCV tabele, kontrolna struktura algoritma i baferi korišćeni pri radu algoritma.

Background rutina se periodično izvršava u pozadinskoj niti i u njoj je odrađeno ažuriranje lokalne kopije MCV tabele.

Block rutina je iskorišćena za izvršavanje algoritma i obradu. Njeno izvršavanje je upravljano ulaznim tokom podataka. *Block* rutina se izvršava po prijemu 16 PCM odbiraka u ulazno-izlaznom nizu. U ovoj rutini redom su pozvane glavna rutina algoritma, `X_S_tmpModuleMain`, a zatim i rutina za završnu obradu, `X_S_dap_OutProcessing`.

Takođe, umesto ulazno-izlaznih portova, korišćeni su pokazivači na ulazno-izlaznu sprežnu memoriju.

Rastojanje od prve lokacije	Ime pokazivača	Opis
0	<code>X_BY_IOBuffer_Ptrs</code> <code>X_VY_IOBuffer_0_Ptr</code>	Left Channel IO Buffer
1	<code>X_VY_IOBuffer_1_Ptr</code>	Center Channel IO Buffer
2	<code>X_VY_IOBuffer_2_Ptr</code>	Right Channel IO Buffer
3	<code>X_VY_IOBuffer_3_Ptr</code>	Left Surround IO Buffer
4	<code>X_VY_IOBuffer_4_Ptr</code>	Right Surround IO Buffer
5	<code>X_VY_IOBuffer_5_Ptr</code>	Surround Back Left IO Buffer
6	<code>X_VY_IOBuffer_6_Ptr</code>	Surround Back Right IO Buffer
7	<code>X_VY_IOBuffer_7_Ptr</code>	LFE0 IO Buffer
8	<code>X_VY_IOBuffer_8_Ptr</code>	Left High IO Buffer
9	<code>X_VY_IOBuffer_9_Ptr</code>	Right High IO Buffer
10	<code>X_VY_IOBuffer_10_Ptr</code>	Left Wide IO Buffer
11	<code>X_VY_IOBuffer_11_Ptr</code>	Right Wide IO Buffer
12	<code>X_VY_IOBuffer_12_Ptr</code>	Left DualZone IO Buffer
13	<code>X_VY_IOBuffer_13_Ptr</code>	Right DualZone IO Buffer
14	<code>X_VY_IOBuffer_14_Ptr</code>	Left Auxiliary IO Buffer
15	<code>X_VY_IOBuffer_15_Ptr</code>	Right Auxiliary IO Buffer

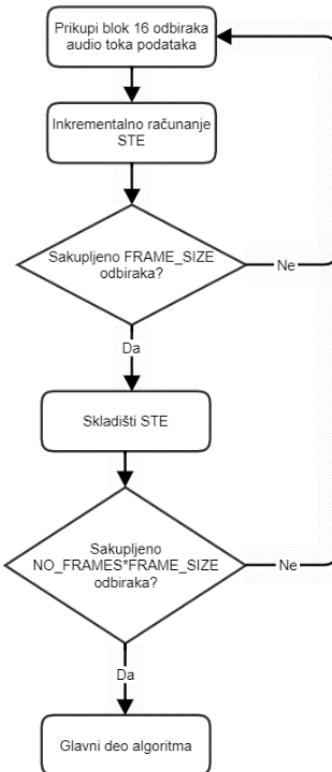
Slika 5.4 Pokazivači na ulazno-izlaznu sprežnu memoriju

Algoritam je bilo potrebno prilagoditi *Block* rutini. Kako se ne bi skladištilo 16 po 16 odbiraka dok se ne dosegne broj odbiraka veličine jednog okvira radi računanja STE, algoritam je izmenjen tako da uzima u obzir blokove od 16 odbiraka. STE za k -ti okvir će biti inkrementalno računat, doprinosom energije od po 16 odbiraka, dok se ne izračuna za ceo okvir. Nakon toga sledi računanje *lowthres* i *MLER* za taj okvir, a zatim sledi odluka o vrsti signala na nivou segmenta.

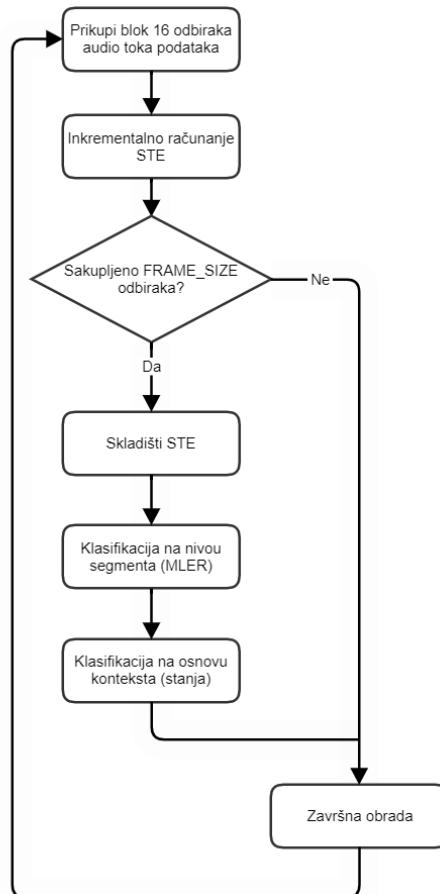
U toku rada aplikacije, prikupljenih 16 odbiraka će uvek biti obrađeno po trenutno odlučenoj vrsti signala (ili samo propušteno, u inicijalnom delu).

Ovako izmenjen kod uvezan sa radnim okruženjem je razvijan pod simulatorskim okruženjem, dok nije potvrđeno da daje ispravne rezultate. Nakon toga, aplikaciju je bilo moguće učitati na razvojnu ploču i potvrditi njen ispravan rad i pri izvršavanju na ciljnoj platformi.

Slika 5.5 i Slika 5.6 prikazuju izgled prilagođenog algoritma, koji je funkcionalno identičan polaznom.



Slika 5.5 Dijagram toka inicialne faze prilagođenog algoritma



Slika 5.6 Dijagram toka glavnog dela prilagođenog algoritma

6. Rezultati

Ovo poglavlje sadrži opis načina merenja performansi algoritma, testiranja i potvrde ispravnosti implementiranog koda, kao i prikaz utrošaka resursa digitalnog signal procesora.

6.1 Mere performanse algoritma

Performanse algoritma određene su pomoću F_1 mera. F_1 mera predstavlja meru performanse algoritma klasifikacije, uzimajući u obzir dve mere – preciznost (engl. *precision*) i odziv (engl. *recall*). Konkretno, bilo ih je potrebno izračunati na makro nivou (na nivou celog klasifikatora), računanjem ovih mera za svaku klasu zasebno.

Preciznost predstavlja odnos tačnih predviđanja pripadnosti klasi i ukupnog broja predviđanja pripadnosti klasi. Jednostavnije, od svih predmeta klasifikacije koji su označeni da pripadaju klasi, koji procenat njih zaista pripada toj klasi.

Odziv predstavlja odnos tačnih predviđanja pripadnosti klasi i ukupnog broja predmeta klasifikacije koji pripadaju toj klasi. Jednostavnije, od svih predmeta klasifikacije koji pripadaju klasi, koji procenat njih je zaista označeno da pripada toj klasi.

F_1 mera je uvedena u slučaju skupa podataka kod kojih pripadnost klasama nije izbalansirana, odnosno, neke klase se u skupu podataka pojavljuju znatno više ili znatno manje puta, što može da rezultuje npr. veoma visokom preciznošću dok bi odziv bio veoma nizak, ili obrnuto. F_1 mera zapravo predstavlja kombinovanu meru performanse algoritma, koja se dobija kao harmonijska sredina preciznosti i odziva.

Preciznost i odziv za klasu k se računaju na sledeći način:

$$\text{precision}_k = \frac{tp_k}{tp_k + fp_k}$$

$$\text{recall}_k = \frac{tp_k}{tp_k + fn_k}$$

gde su tp , fp i fn definisani pomoću matrice konfuzije za višeklasni problem. Matrica konfuzije kod klasifikacionih problema sa više klase, u opštem slučaju opisuje odnos između broja predviđenih i pravih klasa za predmet klasifikacije. Primer matrice konfuzije kod višeklasnog problema prikazuje Tabela 6.1.

Preciznost i odziv na makro nivou, kao i F_1 računaju se pomoću formula:

$$\text{MacroAveragePrecision} = \frac{\sum_{k=1}^K \text{precision}_k}{K}$$

$$\text{MacroAverageRecall} = \frac{\sum_{k=1}^K \text{recall}_k}{K}$$

$$F_1 = 2 \cdot \frac{\text{MacroAveragePrecision} \cdot \text{MacroAverageRecall}}{\text{MacroAveragePrecision} + \text{MacroAverageRecall}}$$

gde je K ukupan broj klasa.

Pripadnost klasi		Stvarna klasa		
		A	B	C
Predviđena klasa	A	tp – tačno predviđena pripadnost	fp – netačno predviđena pripadnost	fp – netačno predviđena pripadnost
	B	fn – netačno predviđena nepripadnost	tn – tačno predviđena nepripadnost	tn – tačno predviđena nepripadnost
	C	fn – netačno predviđena nepripadnost	tn – tačno predviđena nepripadnost	tn – tačno predviđena nepripadnost

Tabela 6.1 Matrica konfuzije za klasu A

6.1.1 Određeni parametri algoritma i njegove performanse

Tabela 6.3 prikazuje matricu konfuzije za klase govor, muzika i šum, dobijenu nad testnim skupom audio podataka. Tabela 6.4 prikazuje preciznost i odziv za pojedinačne klase, pod vrednostima parametara modela koje prikazuje Tabela 6.2.

Tabela 6.5 prikazuje vrednosti makro vrednosti preciznosti i odziva, kao i vrednost F_1 mere.

Parametar	fs	FRAME_SIZE	NO_FRAMES	COUNTER_LIMIT	STATE_LIMIT	λ_{SM}	λ_{MN}
Vrednost	22050	512	64	32	255	0.068	0.05

Tabela 6.2 Vrednost parametara dobijenog modela

Pričinost klasi		Stvarna klasa		
		Govor	Muzika	Šum
Predviđena klasa	Govor	101657	36060	89
	Muzika	743	172962	432
	Šum	611	1158	1732

Tabela 6.3 Matrica konfuzije za dobijeni model za testni skup tokova audio podataka

Mera		Vrednost (%)
Preciznost	Govor	73.78
	Muzika	99.33
	Šum	49.47
Odziv	Govor	98.69
	Muzika	82.29
	Šum	76.87

Tabela 6.4 Preciznosti i odzivi za pojedinačne klase

Mera	Vrednost (%)
Preciznost	74.19
Odziv	85.93
F ₁	79.63

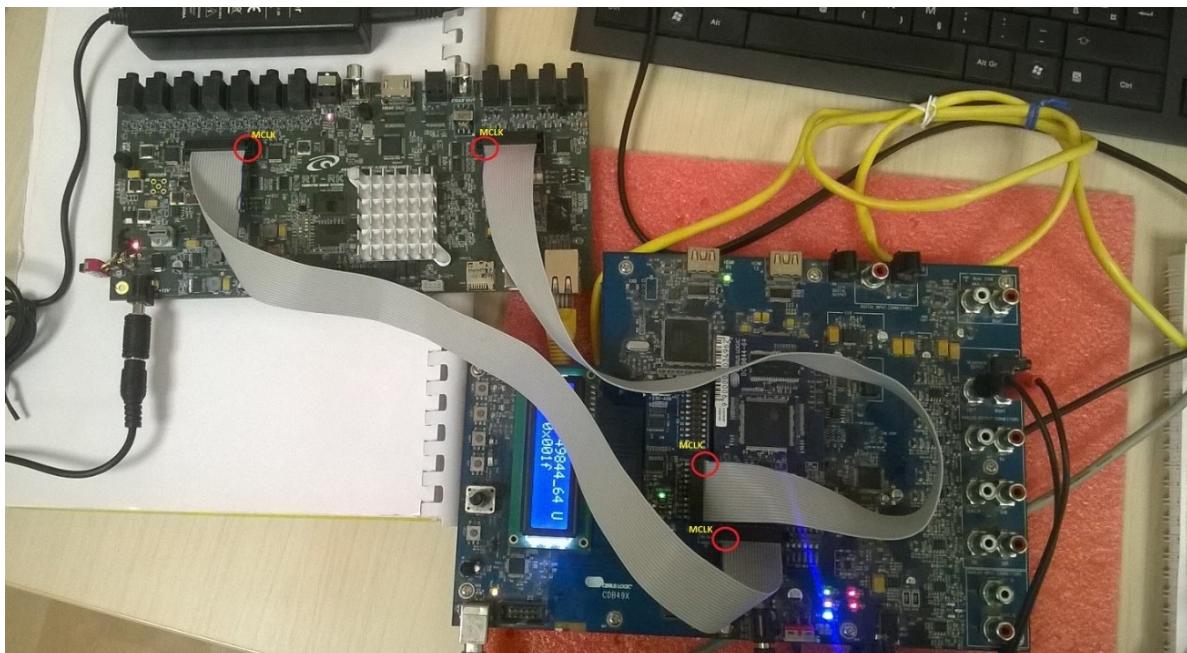
Tabela 6.5 Makro vrednosti preciznosti i odziva klasifikatora i F₁ mera

6.2 Ispitivanje ispravnosti programa

U fazama prelaska s referentnog *C* koda na asemblerski kod za samostalno izvršavanje, kao i prilagođavanju algoritma radnom okruženju procesora, za potvrdu ispravnosti koda korišćeno je testiranje izlaznih datoteka iz ovih programa na nivou bita. Za ovo je korišćen alat *PCMCompare*. Kao ulaz su korišćene mono audio .wav datoteke sa frekvencijom odabiranja 22050 Hz i 44100 Hz.

Testiranje je prvo vršeno tako što su te audio datoteke korišćene kao ulazne za referentni C kod i za asemblererski kod za samostalno izvršavanje u simulatoru. Dobijeni izlazi su korišćeni kao ulazne datoteke za alat *PCMCompare*. Utvrđeno je da, u slučaju segmenata koji su obeleženi kao govor i šum nema razlike na nivou bita, dok se kod segmenata koji su označeni kao muzika uočava razlika od maksimalno jednog bita, usled različite preciznosti podržane od strane PC platforme i ciljne platforme.

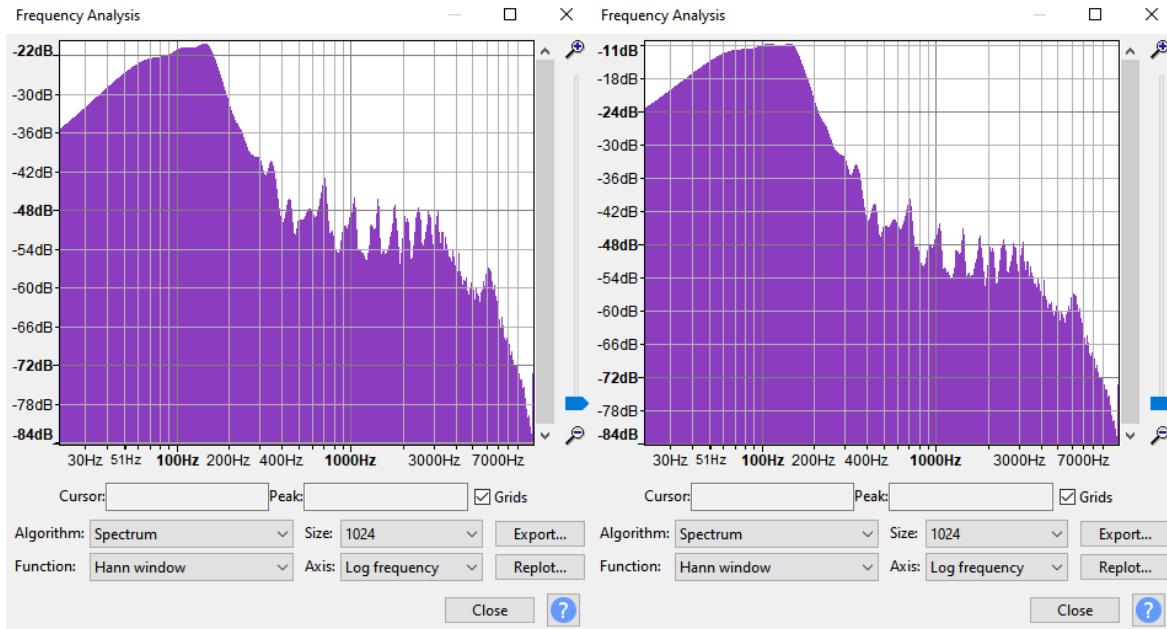
Kod asemblererskog koda uvezanog s radnim okruženjem, testiranje je vršeno poređenjem sa izlazima iz asemblererskog koda za samostalno izvršavanje. Nakon što je aplikacija spuštena na ploču, pomoću eksterne zvučne kartice *RT-AG* koja je korištena za puštanje toka audio podataka na ulaz digitalnog signal procesora, takođe je snimljen tok podataka na njegovom izlazu. Izlazne datoteke iz ove dve aplikacije su poređene na nivou bita i utvrđeno je da nema razlike.



Slika 6.1 *RT-AG* zvučna kartica povezana sa razvojnom pločom CDB49x

Ispravnost rutina za završnu obradu proverena je pomoću alata za spektralnu analizu u programu *Audacity*.

Slika 6.2 pokazuje poređenje spektra muzičkog segmenta kod ulazne i izlazne datoteke DSP aplikacije. Kako se segmenti obeleženi kao muzika obrađuju niskopropusnim *shelving* filtrom koji se koristi za kontrolu jačine basa (niskofrekventnog sadržaja signala), u odnosu na spektr muzičkog segmenta kod ulazne datoteke (levo), na spektru tog istog segmenta na izlazu (desno) uočava se pojačanje niskofrekventnog sadržaja.



Slika 6.2 Poređenje spektra muzičkog segmenta na ulazu i izlazu u programu *Audacity*

6.3 Utrošak resursa

Utrošak memorije prikazuje Tabela 6.6. Prikazana je količina utrošene memorije algoritma za segmentaciju i klasifikaciju, završnu obradu i spregu s radnim okruženjem po memorijskim zonama, u 32-bitnim rečima. Podaci o utrošenoj memoriji se dobijaju iz datoteke sa ekstenzijom *.MAP* dobijenom prilikom generisanja *.uld* datoteke.

Memorijska zona	Utrošena memorija (u 32-bitnim rečima)
X	72
Y	21
L	0
P	256

Tabela 6.6 Utrošak memorije modula

Utrošak procesorskog vremena se meri u milionima instrukcija u sekundi (MIPS – engl. *Million Instructions Per Second*). Utrošen broj MIPS-a se računa pomoću formule:

$$MIPS = \frac{\text{broj_ciklusa} \cdot \frac{Fs}{\text{BLOCK_SIZE}}}{1000000}$$

gde Fs predstavlja frekvenciju odabiranja signala, a $BLOCK_SIZE$ veličinu bloka obrade u odbircima.

Kako aplikacija u jednom delu samo inkrementalno računa STE, sve dok ne izračuna za jedan ceo okvir kada se nastavlja na sam algoritam klasifikacije, najveći utrošak procesorskog

vremena aplikacije će biti u tom trenutku. Takođe, pri završnoj obradi, najveći utrošak vremena je kod okvira koji su obeleženi kao muzika, jer je obrada kod tih okvira zahtevnija od obrade okvira govora ili šuma. Tabela 6.7 prikazuje utrošak procesorskog vremena za obradu jednog bloka dužine 16 odbiraka u najgorem slučaju, za jedan kanal: prikupljeno je odbiraka dužine okvira i osim inkrementalnog računanja STE, izvršava se i algoritam segmentacije i klasifikacije, gde je okvir klasifikovan kao muzika.

Broj utrošenih ciklusa je određen pomoću kontrolisanog izvršavanja programa u CLIDE okruženju, profilisanjem aplikacije izvršene u simulatoru.

Fs (Hz)	44100	
	Broj ciklusa	MIPS
X S tmpModuleMain	661	1.822
X S dap_OutProcessing	488	1.345
Ukupno	1149	3.167

Tabela 6.7 Utrošak procesorskog vremena u najgorem slučaju

7. Zaključak

U ovom radu razvijen je algoritam za segmentaciju i klasifikaciju za obradu audio signala i izvršena je njegova implementacija i optimizacija u asemblerском jeziku u obliku modula za digitalni signal procesor CS49844 firme *Cirrus Logic*. Pre samog razvoja algoritma i njegove implementacije, bilo je potrebno upoznati se sa algoritmima segmentacije i klasifikacije audio signala, kao i sa ciljnom platformom na kojoj će se algoritam izvršavati.

Izrada je tekla po fazama. U prvoj fazi, nakon pregleda literature i upoznavanja sa različitim algoritmima segmentacije i klasifikacije audio signala, kao i sa ograničenjima ciljne platforme, konstruisan je odgovarajući algoritam pomoću programskog jezika *Python* i razvojnog okruženja *PyCharm*. Ovde su određeni parametri algoritma i njegova efikasnost pomoću F_1 mere.

U drugoj fazi, referentni kod algoritma i odgovarajuće završne obrade implementirani su u programskom jeziku *C*, koristeći razvojno okruženje *Visual Studio*, radi lakšeg prelaska na pisanje koda u asemblerском jeziku ciljne platforme.

Dalje, koristeći razvojno okruženje CLIDE, modul je na osnovu referentnog koda prvo implementiran u asemblerском jeziku ciljne platforme kao modul za samostalno izvršavanje u simulatoru. Nakon optimizacija i potvrde ispravnosti asemblerског кода pomoću testova na nivou bita, usledilo je uvezivanje sa radnim okruženjem i operativnim sistemom ciljne platforme, gde je algoritam morao biti prilagođen prirodi rutina za spregu s operativnim sistemom i radnim okruženjem. Nakon toga je utvrđena ispravnost ovako izmenjenog i povezanog modula s operativnim sistemom u simulatorskim uslovima testovima na nivou bita. Izračunati su utrošci resursa i potvrđeno je da su oni u granicama ciljne platforme. Generisana je *.uld* datoteka za dobijeni modul i on je zajedno s radnim okruženjem učitan na ciljnu platformu. Pomoću *RT-AG* zvučne kartice su snimljeni izlazi iz algoritma radi provere ispravnosti rada aplikacije na ciljnoj platformi. Uočeno je da nema razlike na nivou bita u

odnosu na kod izrađen za izvršavanje u simulatoru, čime je zaključen ispravan rad aplikacije i na ciljnoj platformi.

Realizovano rešenje moguće je unaprediti na nekoliko načina. Prvo, u algoritmu za segmentaciju i klasifikaciju moguće je uneti izmene koje bi u delu odlučivanja o vrsti signala na osnovu konteksta mogle koristiti neki drugi princip, umesto memorije stanja, što bi moglo rezultovati boljim performansama algoritma i manjim kašnjenjem odluke. Takođe, moguće je uvesti dinamičko određivanje nekih od parametara algoritma u odnosu na frekvenciju odabiranja algoritma radi poboljšanja performansi algoritma na različitim frekvencijama odabiranja. Zatim, asemblerско rešenje bi se moglo unaprediti uvođenjem podrške za rad sa više kanala, kako je realizovano rešenje implementirano za rad samo sa jednim kanalom, pri čemu bi najviše trebalo voditi računa o utrošku procesorskog vremena.

8. Literatura

- [1] E. Scheirer, M. Slaney, „*Construction and evaluation of a robust multifeature speech/music discriminator*“, 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing, 1997.
- [2] J. Saunders, „*Real-time discrimination of broadcast speech/music*“, 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings, 1996.
- [3] T. Zhang, C. Kuo, „*Audio content analysis for online audiovisual data segmentation and classification*“, IEEE Transactions on Speech and Audio Processing, 2001.
- [4] Y. Patsis, W. Verhelst „*A Speech/Music/Silence/Garbage/ Classifier for Searching and Indexing Broadcast News Material*“, 2008 19th International Conference on Database and Expert Systems Applications, 2008.
- [5] N. Vryzas, N. Tsipas, and C. Dimoulas, „*Web Radio Automation for Audio Stream Management in the Era of Big Data*“, MDPI, 11.04.2020. [Na mreži]. Dostupno: <https://www.mdpi.com/2078-2489/11/4/205/htm>. [Pristupljeno: 01.06.2021]
- [6] T. Theodorou, I. Mporas, N. Fakotakis, „*An Overview of Automatic Audio Segmentation*“, International Journal of Information Technology and Computer Science, 2014.
- [7] Y. Lavner, D. Ruinskiy, „*A Decision-Tree-Based Algorithm for Speech/Music Classification and Segmentation*“, EURASIP Journal on Audio, Speech, and Music Processing, vol. 2009, pp. 1–14, 2009.
- [8] J. Eyre, J. Bier, „*The evolution of DSP processors*“, IEEE Signal Processing Magazine, vol. 17, no. 2, pp. 43-51, 2000.

- [9] W. Q. Wang, W. Gao, D. W. Ying, „*A fast and robust speech/music discrimination approach*“, Fourth International Conference on Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint, 2004.
- [10] Cirrus Logic, Inc, „*CS498xx_Datasheet*“, 2015.
- [11] Cirrus Logic Inc, „*Cirrus Logic 32-bit DSP Assembly Programmer's Guide*“, 2014
- [12] V. Kovačević, M. Popović, M. Temerinac, N. Teslić, „*Arhitekture i algoritmi digitalnih signal procesora 1*“, FTN izdavaštvo, Novi Sad, 2005.
- [13] J. Kovačević, D. Bokan, „*Arhitekture i algoritmi digitalnih signal procesora: zbirka zadataka i laboratorijski priručnik*“, FTN izdavaštvo, Novi Sad, 2016.
- [14] M. Grandini, E. Bagli, G. Visani, „*Metrics for Multi-Class Classification: An Overview*“, arXiv preprint arXiv:2008.05756, 2020.