



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Никола Андрић

**Аутоматизација испитивања објектно
оријентисане аудио технологије на наменској
платформи са ограниченим ресурсима**

МАСТЕР РАД

Нови Сад, 2016



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	Монографска документација
Тип записа, ТЗ :	Текстуални штампани материјал
Врста рада, ВР :	Дипломски – мастер рад
Аутор, АУ :	Никола Андрић
Ментор, МН :	Др Јелена Ковачевић
Наслов рада, НР :	Аутоматизација испитивања објектно оријентисане аудио технологије на наменској 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 :	2016
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)	7/38/0/2/21/0/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	испитивање, аутоматизација, наменски системи, ББТ (испитивање методом црне кутије)
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	У овом раду реализовано је решење за аутоматско испитивање сложене аудио технологије на наменској платформи са ограниченим ресурсима. Циљ рада је скраћивање времена потребног за испитивање, као и смањење утицаја људског фактора приликом испитивања.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник: Др Илија Башичевић
	Члан: Др Иван Мезеи
	Члан, ментор: Др Јелена Ковачевић
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual printed material
Contents code, CC :	Master Thesis
Author, AU :	Nikola Andrić
Mentor, MN :	PhD Jelena Kovačević
Title, TI :	Automation of testing object-oriented audio technology on an embedded 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 :	2016
Publisher, PB :	Author's reprint
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/38/0/2/21/0/0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, S/KW :	testing, automation, embedded systems, BBT (Black Box Testing)
UC	
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, N :	
Abstract, AB :	This paper describes the solution for automated testing of complex audio technology on embedded platform. The main goal was saving time needed for testing as well as reducing human impact on testing process.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	
Defended Board, DB :	President: PhD Ilija Bašičević
	Member: PhD Ivan Mezei
	Member, Mentor: PhD Jelena Kovačević
	Menthor's sign

Zahvalnost

Zahvaljujem se mentoru i kolegama na stručnoj pomoći tokom izrade ovog rada.
Posebno se zahvaljujem porodici na podršci koju su mi pružili dok sam studirao.

SADRŽAJ

1. Uvod.....	1
2. Teorijske osnove	3
2.1 Audio tehnologije	3
2.2 Ispitivanje softvera	5
2.3 Nivoi ispitivanja	6
2.3.1 Jedinično ispitivanje	6
2.3.2 Integraciono ispitivanje	6
2.3.3 Sistemsko ispitivanje	6
2.3.4 Ispitivanje prihvatljivosti	7
2.3.5 Regresivno ispitivanje	7
2.4 Dinamičko i statičko ispitivanje.....	7
2.5 Funkcionalno i nefunkcionalno ispitivanje	7
2.6 Metodologije crne i bele kutije.....	8
2.6.1 Ispitivanje metodom bele kutije.....	8
2.6.2 Prednosti i nedostaci ispitivanja metodom bele kutije	9
2.6.3 Ispitivanje metodom crne kutije	9
2.6.4 Prednosti i nedostaci ispitivanja metodom crne kutije	10
2.7 Ručno i automatsko ispitivanje	11
3. Koncept rešenja.....	13
3.1 Faze životnog ciklusa ispitivanja softvera	13
3.1.1 Planiranje procesa ispitivanja	13
3.1.2 Dizajn ispitnih slučajeva.....	14
3.1.3 Izvršavanje ispitnih slučajeva	14

3.1.4	Aktivnosti nakon izvršavanja	14
3.2	BBT	14
3.3	Ciljna platforma.....	16
4.	Implementacija.....	19
4.1	Planiranje procesa ispitivanja.....	19
4.2	Dizajn ispitnih slučajeva	21
4.2.1	Organizacija radnog okruženja	21
4.2.2	Skripte za generisanje ispitnih slučajeva	22
4.3	Izvršavanje ispitnih slučajeva.....	26
4.4	Aktivnosti nakon ispitivanja.....	31
5.	Rezultati	34
6.	Zaključak	36
7.	Literatura.....	37

SPISAK SLIKA

Slika 2.1 – Primer 5.1 i 7.1 konfiguracije	4
Slika 2.2 – Primer jedne putanje pokretnog objekta	5
Slika 3.1 – Primer ispitnog okruženja	15
Slika 3.2 – Blok dijagram procesora CS49844	16
Slika 3.3 – Blok šema funkcionisanja složene audio tehnologije na ciljnoj platformi	17
Slika 3.4 – Ispitno okruženje.....	18
Slika 4.1 – Blok dijagram testnog okruženja	20
Slika 4.2 – Organizacija direktorijuma radnog okruženja	21
Slika 4.3 – Organizacija direktorijuma testnog okvira.....	22
Slika 4.4 – Primer jedne datoteke sa karakteristikama ispitnog slučaja	23
Slika 4.5 – Blok dijagram procesa generisanja ispitnih slučajeva	23
Slika 4.6 – Organizacija poddirektorijuma <i>tests</i> direktorijuma	24
Slika 4.7 – Blok dijagram jednog BBT testa	25
Slika 4.8 – BBT aplikacija nakon pokretanja	26
Slika 4.9 – Konfigurisanje BBT alata	27
Slika 4.10 - Prikaz prozora u kom se može pratiti napredovanje izvršavanja ispitnog slučaja	28
Slika 4.11 – Kreiranje BBT projekta.....	29
Slika 4.12 – Pokretanje ranije kreiranih BBT projekata	30
Slika 4.13 - Prikaz ishoda izvršavanja testova.....	31
Slika 4.14 – Blok dijagram poređenja referentnih i izlaza sa ciljne platforme	32
Slika 5.1 – Grafik zavisnosti vremena trajanja ispitivanja od broja izvršenih ispitnih slučajeva kada se ispitivanje izvršava ručno ili automatski	35

SPISAK TABELA

Tabela 3-1: Raspoloživa memorija procesora CS49844.....	16
Tabela 3-2 – Prikaz potrošnje resursa ciljne platforme.....	17

SKRAĆENICE

DSP	- <i>Digital Signal Processing</i> , Digitalni signal procesor
BBT	- <i>Black Box Testing</i> , Ispitivanje metodom crne kutije
WBT	- <i>White Box Testing</i> , Ispitivanje metodom bele kutije
SDLC	- <i>Software Development Life Cycle</i> , Životni ciklus razvoja softvera
STLC	- <i>Software Testing Life Cycle</i> , Životni ciklus ispitivanja softvera
CLI	- <i>Command Line Interface</i> , Sprega pomoću komandne linije
GUI	- <i>Graphical User Interface</i> , Grafička korisnička sprega
MIPS	- <i>Million Instructions Per Second</i> , Milion instrukcija po sekundi
PCM	- <i>Pulse Code Modulation</i> , Impulsna kodna modulacija
USB	- <i>Universal Serial Bus</i> , Univerzalna serijska sprega
I2S	- <i>Integrated Interchip Sound</i> , sprega za digitalni prenos audio signala

1. Uvod

Tema ovog rada je automatizacija procesa ispitivanja objektno orijentisane audio tehnologije na namenskoj platformi sa ograničenim resursima. Cilj rada je postizanje potpune automatizacije procesa ispitivanja. Automatizacijom procesa ispitivanja se vreme potrebno za izvršavanje ispitnih slučajeva skraćuje, sam proces je pouzdaniji, a potreba za intervencijom čoveka se svodi na minimum. Rešenje podrazumeva upoznavanje sa specifikacijom ispitivane tehnologije, kreiranje ispitnog okruženja, generisanje ispitnih slučajeva, kao i realizaciju skripti i alata koji su bili neophodni da bi se izvršilo ispitivanje.

Munjevit napredak uređaja potrošače elektronike nije zaobišao ni oblast audio sistema. Nakon višegodišnje vladavine formata baziranih na kanalima, kod kojih se trodimenzionalni doživljaj bazirao na raspoređivanju zvukova po kanalima, nove generacije audio tehnologija donose određene promene. Savremene audio tehnologije su zasnovane virtuelnim izvorima zvuka tzv. audio objektima koji su pozicionirani na određenoj lokaciji u prostoru. Upotreba objekata omogućava tačnu prostornu reprodukciju zvučnog signala nezavisno od konfiguracije zvučnika u sistemu.

Dodatne mogućnosti dovode do usložnjavanja sistema. Samim tim se i proces ispitivanja i verifikacije ovih rešenja usložnjava i predstavlja veći izazov za inženjere. Imajući u vidu složenost današnjih audio tehnologija, metode ručnog ispitivanja su neefikasne u primeni, kako sa stanovišta utrošenog vremena, tako i pouzdanosti vezane za ocenu ispravnosti ispitivanog sistema. Ručno ispitivanje je zavisno od ljudske interpretacije, odnosno podložno greškama test inženjera prilikom ocene ispravnosti sistema, usled nedovoljnog poznavanja specifikacije sistema, iskustva ili psiho-fizičkog stanja test inženjera. Sa druge strane, primena metoda ručnog ispitivanja oduzima značajan deo vremena potrebnog za razvoj sistema. Usled težnje proizvođača da obezbede konkurentnost na tržištu, razvoj sistema u ovoj oblasti je u velikoj meri

uslovljen potrebom za skraćenjem vremena pojavljivanja proizvoda na tržištu. Skraćenje vremena razvoja se u praksi najčešće postiže skraćivanjem procesa ispitivanja. Sa druge strane, nedovoljno ispitan proizvod dovodi do smanjenja kvaliteta, što negativno utiče na konkurentnost. Da bi se obezbedilo kraće vreme proizvodnje i pružio bolji kvalitet, potrebno je više pažnje posvetiti analizi kvaliteta sistema i unaprediti proces ispitivanja, čime bi se poboljšao njegov kvalitet.

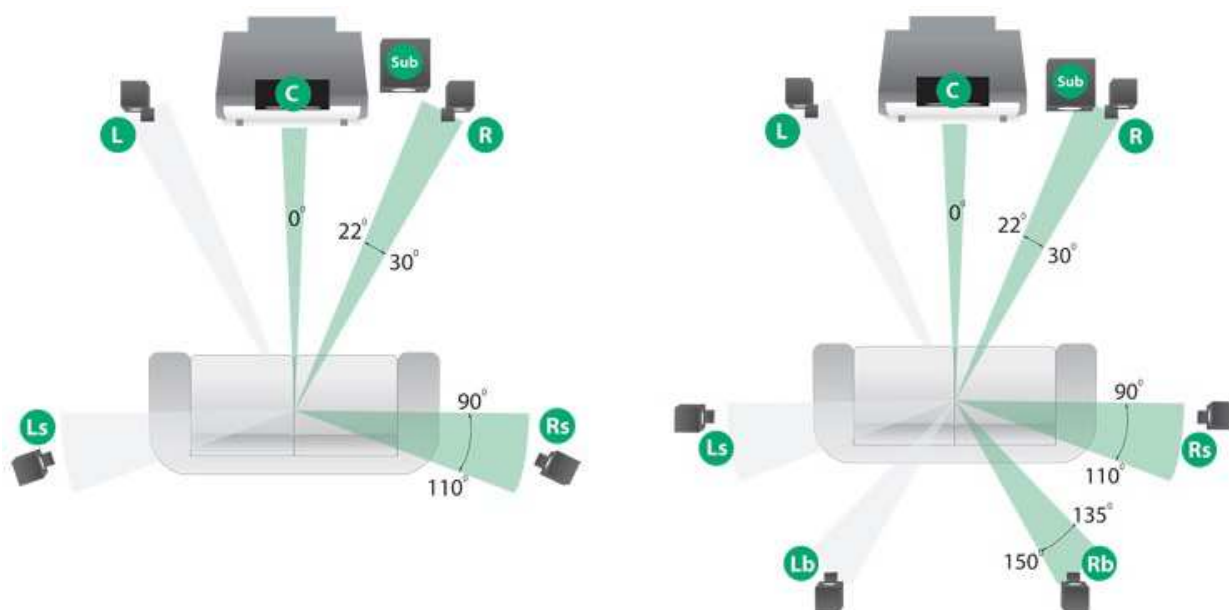
Rad se sastoji iz sedam poglavlja. U drugom poglavlju su opisani su teorijski pojmovi na kojima se rad zasniva. U trećem poglavlju opisan je koncept rešenja na kom je zasnovana implementacija ovog rešenja. Četvrto poglavlje odnosi se na detalje implementacije. U petom poglavlju su sumirani i opisani postignuti rezultati. Šesto poglavlje sadrži kratak pregled svega urađenog u okviru rada. U sedmom poglavlju dat je spisak korišćene literature.

2. Teorijske osnove

U ovom poglavlju opisani su teorijski pojmovi na kojima je rad zasnovan. U prvom delu poglavlja, predstavljene su nove audio tehnologije, zasnovane na audio objektima, kao i njihove razlike u odnosu na tradicionalne audio tehnologije, zasnovane na kanalima. U nastavku je dat odgovor na pitanje šta je ispitivanje softvera i zbog čega je značajno, zatim će biti pomenute neke od kategorija i nivoa ispitivanja softvera. Takođe, dat je opis metoda ispitivanja koje se najčešće koriste. Jedan deo poglavlja posvećen je značaju automatizacije ispitivanja softvera, i prednostima automatskog ispitivanja u odnosu na manuelno ispitivanje.

2.1 Audio tehnologije

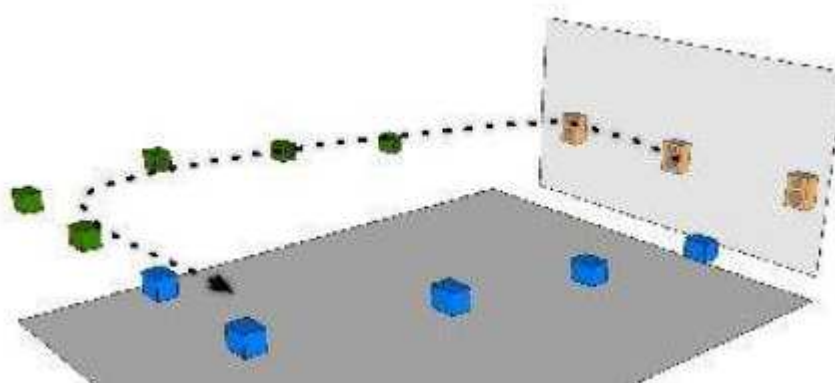
Jedan od najzastupljenijih načina reprodukcije prostornog zvuka jesu tzv. kanalni signali, gde svaki od njih predstavlja set talasnih oblika predviđenih za unapred utvrđenu konfiguraciju zvučnika. Zvuk koji publika čuje predstavlja kombinaciju velikog broja pojedinačnih zvučnih elemenata, dijaloga snimljenih na setu, muzike snimljene u studiju itd. Svi ovi elementi distribuirani su na nekoliko audio zapisa i snimljeni u studiju koristeći kompleksne sisteme za snimanje, uređivanje i reprodukciju digitalnog signala u skladu sa standardizovanom konfiguracijom zvučnika. Krajnji rezultat ovog procesa je skup kanala, gde svaki od njih predstavlja sadržaj koji će se slati na jedan od zvučnika [1]. Danas su u upotrebi različite konfiguracije, od dvokanalnog stera, višekanalnih 5.1 i 7.1 konfiguracija, do trodimenzionalnih konfiguracija sa različitim brojem zvučnika koji se nalaze iznad ili ispod horizontalne ravni u kojoj se nalazi slušalac [2]. Kanalno orijentisani zvučni materijal je dosta izučavan i dugo u upotrebi, međutim kako je stvoren za specifičnu zvučničku konfiguraciju, suštinski je ograničen, nema potrebne informacije za pružanje potpunijeg i bogatijeg zvuka ukoliko se sistem proširi.



Slika 2.1 – Primer 5.1 i 7.1 konfiguracije

Na primer, ako je filmski sadržaj namenjen za 5.1 sisteme, u njemu su snimljene informacije potrebne za reprodukciju na pet zvučnika punog opsega i jednom zvučniku namenjenom za niskofrekventne sadržaje (eng. subwoofer). U tom slučaju dodavanje više zvučnika u sistem nema smisla jer reprodukcija na drugačijoj konfiguraciji zvučnika neće dovesti poboljšanja. Pored toga, potrebno je voditi računa o poziciji zvučnika u prostoriji, jer svako odstupanje od optimalne pozicije takođe može rezultovati smanjenjem kvaliteta sveukupnog doživljaja.

Kada govorimo o audio tehnologijama novije generacije, govorimo o tehnologijama baziranim na audio objektima. Prostorni zvuk se danas može opisati i pomoću virtuelnih izvora tzv. audio objekata koji su pozicionirani na određenoj lokaciji u prostoru. Upotreba objekata omogućava tačnu prostornu reprodukciju zvučnog signala nezavisno od konfiguracije zvučnika. Osim zvučnog zapisa svaki audio objekat sadrži i dodatne informacije tzv. meta podatke. Ovi podaci definišu kada, gde i koliko glasno će se zvučni signal pojaviti. Meta podaci su ti koji omogućavaju adaptaciju zvučnog sadržaja ciljnoj konfiguraciji zvučnika, dinamičko objektno renderovanje kao i interaktivnost sa korisnikom. Ovi meta podaci sadrže informacije koje su promenljive u toku vremena, i koje kontrolišu proces renderovanja, kao što su na primer prostorne koordinate, pojačanje, širenje virtuelnog izvora. Zbog mogućnosti vremenski promenljivih pozicija objekata, moguće je i modelovati različite zvučne efekte, npr. avion koji preleće iznad slušaoca, cvrkut ptica i sl. Na Slika 2.2 dat je primer jedne trajektorije objekta, gde se njegov položaj u prostoru menja sa vremenom, od centralnog zvučnika, preko levog, levog surround zvučnika, zatim preko zadnjih zvučnika do zadnjeg desnog ugla. Tom putanjom i slušalac čuje zvuk koji dati objekat nosi.



Slika 2.2 – Primer jedne putanje pokretnog objekta

Meta podaci mogu biti asocirani svakom pojedinačnom audio elementu, ili su organizovani u grupe, tako da se odnose na više audio elemenata. Posebni meta podaci mogu definisati kontrolu interakcije sa podacima od strane korisnika, npr. dozvoljeni opseg promene pojačanja [3]. S obzirom na količinu podataka potrebnih, za realizaciju ovakvih formata neophodan je medijum velikog kapaciteta.

2.2 Ispitivanje softvera

Ispitivanje softvera predstavlja pokušaj da se pronađu greške u softveru koji je napravljen. Softver je implementiran prema korisničkim zahtevima kojima se rešava neki realni problem ili se kreira neka korisna funkcionalnost koja pruža usluge krajnjim korisnicima. Nakon implementacije softver može u manjoj ili većoj meri da se slaže sa originalnim zahtevima prema kojima je napravljen. Svako ponašanje softvera koje se ne slaže sa originalnim zahtevima predstavlja grešku koju je potrebno identifikovati i otkloniti. Proces ispitivanja predstavlja proveru da li je određeni softver u potpunosti implementiran u skladu sa originalnim korisničkim zahtevima. Ispitivanje softvera je važan deo procesa razvoja softvera. Neke procene govore da se za ispitivanje i pronalaženje grešaka utroši i do 50% ukupnog vremena potrebnog za razvoj softverskog proizvoda [4].

Proces ispitivanja softvera uključuje validaciju i verifikaciju. Validacijom se utvrđuje da li se razvija softver u skladu sa specifikacijom koja je utvrđena na početku procesa izrade softvera. Verifikacija opisuje proces utvrđivanja da li se softver razvija ispravno, i njen cilj je da pokaže da svi delovi sistema, pojedinačno i kao celina, rade ispravno.

Proces ispitivanja skoro nikada ne može biti kompletan, odnosno ne može pokazati da softver radi bez ikakvih grešaka. Odatle se nameće zaključak da je cilj ispitivanja detektovanje što većeg broja grešaka. Neke greške su kompleksnije od drugih i njihov efekat može biti ozbiljniji. Kompleksne greške su povezane sa jednostavnijim na takav način da će skup testova kojim se detektuju sve jednostavnije greške detektovati i visok procenat složenih grešaka [5].

Kako postoji više različitih tipova grešaka, postoji i više različitih tipova ispitivanja. Neke od klasifikacija ispitivanja su opisane u nastavku poglavlja.

2.3 Nivoi ispitivanja

Ispitivanje može da se sprovodi na različitim nivoima. U tom smislu ispitivanje se deli na jedinično, integraciono, sistemsko, regresivno, ispitivanje prihvatljivosti i alfa i beta ispitivanje. Ove podele su detaljnije predstavljene u [6],[7],[8]. Cilj ovih različitih nivoa ispitivanja je da se sistem sagleda iz različitih perspektiva kako bi se pronašle greške različitih tipova. Ako se ovi nivoi sagledavaju sa strane automatizacije, nivoi mogu da budu jedinično ispitivanje, ispitivanje komponenti i sistemsko ispitivanje.

2.3.1 Jedinično ispitivanje

Jedinično ispitivanje se odnosi na testove koji proveravaju funkcionalnost svakog dela koda i to obično na nivou funkcije. Osnovni cilj ovakvog ispitivanja je otkrivanje strukturnih i funkcionalnih defekata u odgovarajućoj jedinici.

Jedinične testove najčešće piše programer koji je napisao i jedinicu sistema. Greške koje se otkriju ovakvim ispitivanjem su jednostavne za lociranje i za ispravljanje.

Jediničnim ispitivanjem se ne može proveriti funkcionalnost sistema, ali se potvrđuje da pojedine komponente sistema mogu raditi zasebno.

2.3.2 Integraciono ispitivanje

Kada se osnovne jedinice sistema grupišu dobijaju se podsistemi. Komponente sistema se mogu integrisati iterativnim načinom ili sve zajedno takozvano Big-Bang.

Cilj integracionog ispitivanja je da verifikuje da sprege između komponenata rade ispravno. Takođe, integracioni testovi verifikuju i korektnost toka operacija i toka podataka između komponenti.

2.3.3 Sistemsko ispitivanje

Kada se ispitivani podsistemi sjedine u finalni sistem, moguće je izvršavati sistemske testove. Sistemskim ispitivanjem se proverava funkcionalno ponašanje sistema i nefunkcionalne osobine sistema.

Cilj sistemskog ispitivanja je provera da li se sistem ponaša ispravno kada se sve komponente koriste zajedno. Nakon ovog ispitivanja sistem je spreman za ispitivanje prihvatljivosti.

2.3.4 Ispitivanje prihvatljivosti

Ovaj oblik ispitivanja obavlja se nakon sistemskog ispitivanja. Cilj ispitivanja prihvatljivosti jeste evaluacija sistema u smislu klijentskih očekivanja i ciljeva, proverava se da li sistem zadovoljava specifikaciju. Kada se ova faza uspešno obavi softver ulazu u fazu produkcije.

Ukoliko je razvijani softver namenjen široj upotrebi i razvija se bez klijenta, često se ispitivanje prihvatljivosti i ne vrši na prethodno opisan način, već kroz alfa i beta ispitivanje.

U alfa ispitivanje, potencijalni korisnici sistema kao i članovi organizacije koja razvija sistem testiraju proizvod. Posle uklanjanja grešaka otkrivenih prilikom alfa ispitivanja, sprovodi se beta ispitivanje. Proizvod se šalje korisnicima koji ga koriste u realnom okruženju i izveštavaju o pronađenim greškama.

2.3.5 Regresivno ispitivanje

Regresivno ispitivanje ima za cilj da utvrdi da li su stare karakteristike sistema i dalje zadovoljene nakon nekih izmena u sistemu i da verifikuje da promene nisu proizvele nove defekte. Važnost regresivnog ispitivanja je veća kod sistema koji se održavaju duži vremenski period i koji se objavljuju u više verzija. Funkcionalnosti implementirane u ranijim verzijama treba da rade zajedno sa novim funkcionalnostima, a verifikacija toga može oduzeti dosta vremena. Za ovakve zadatke preporučuje se automatizacija procesa ispitivanja.

2.4 Dinamičko i statičko ispitivanje

Ova podela vrši se na osnovu toga da li se tokom ispitivanja kod izvršava ili ne. Kod statičkog ispitivanja, kod se ne izvršava. Statičko ispitivanje predstavlja analizu dokumentacije i njeno poklapanje sa definisanim standardima. Na ovaj način mogu se otkriti propusti u funkcionalnosti, performansama i sigurnosti. Ovaj vid ispitivanja obuhvata i analizu koda, poput provere sintaksne ispravnosti. Sa ovakvim ispitivanjem greške u kodu se mogu uočiti u ranim fazama implementacije.

Dinamičko ispitivanje podrazumeva izvršavanje samog koda, mogu se izvršavati samo delovi ili ceo sistem. Ovaj vid ispitivanja dodatno se deli na funkcionalno i nefunkcionalno ispitivanje.

2.5 Funkcionalno i nefunkcionalno ispitivanje

Funkcionalnim ispitivanje se ispituje da li program pruža funkcionalnost koja je opisana zahtevima. Ovaj vid ispitivanje se fokusira na unošenje ulaznih parametara u sistem i verifikaciju izlaznih podataka. Na taj način može se proveriti da li se ceo sistem, ili deo sistema, ponaša onako kao je to definisano u specifikaciji.

Nefunkcionalno ispitivanje se odnosi na kvalitativne aspekte softvera kao što su brzina rada, skalabilnost, sigurnost itd. Neki od primeri nefunkcionalnog ispitivanje su:

- Ispitivanje performansi, čiji je cilj provera ispravnosti rada sistema u odnosu na zahteve vezane za brzinu izvršavanja određenih zahteva pod različitim opterećenjem.
- Ispitivanje bezbednosti, koje se fokusira na bezbednost podataka u sistemu.
- Ispitivanje pouzdanosti
- Ispitivanje u stresnim uslovima, je jedna vrsta ispitivanja pouzdanosti, gde si ispituje ponašanje sistema pri velikom opterećenju.
- Ispitivanje zagušenja, odnosno provera da li sistem može da zadovolji višestruke zahteve različitih aktera za istim resursom.

2.6 Metodologije crne i bele kutije

Prilikom ispitivanja, sistem koji se ispituje može se posmatrati kao crna kutija ili kao bela kutija.

Kada se koristi metodologija bele kutije, unutrašnja struktura sistema je poznata. Cilj ovakvog ispitivanja je verifikacija korektnog ponašanja unutrašnjih strukturnih elemenata. Ovaj vid ispitivanja je vremenski veoma zahtevan i obično se primenjuje na manje delove sistema.

Kod ispitivanja zasnovanog na metodi crne kutije ne zna se ništa o unutrašnjoj strukturi sistema. Cilj ovakvog ispitivanja je provera da li sistem za dovedene ulazne parametre daje izlaze u skladu sa specifikacijom.

2.6.1 Ispitivanje metodom bele kutije

Ispitivanje metodom bele kutije podrazumeva analizu izvornog koda, i zahteva dobro poznavanje programiranja, u programskom jeziku u kom se vrši implementacija, kao i dizajn konkretnog softverskog proizvoda. Plan ispitivanja se formira na osnovu strukture programa. Ispitni slučajevi se mogu kreirati tako da izvršavaju sve naredbe u programu, pozivaju sve funkcije i prolaze kroz sve kontrole u okviru programa. Na ovaj način može se proveriti skoro celokupan kod, i detektovati greške u dizajnu, logičke i greške u toku podataka. Specifičnim testovima može se proveriti postojanje beskonačnih petlji ili koda koji se nikada ne izvršava [10].

Strategija bele kutije obuhvata ispitivanje kontrole toka i ispitivanje putanja.

Grafovi kontrole toka predstavljaju vizuelnu reprezentaciju strukture koda nekog programa. Algoritam za ispitivanje pomoću grafova kontrole toka prvo pretvara programski kod u graf, a zatim analizira moguće puteve kroz graf.

Pokrivenost testovima je pokazatelj koji ukazuje na to koji procenat izvornog koda je pokriven bar jednim testom, i može se koristiti kako indikator efikasnosti kompletnog procesa ispitivanja.

Neki od kriterijuma pokrivenosti koda su:

Pokrivenost iskaza (eng. Statement Coverage), se postiže tako što se kreiraju ispitni slučajevi koji izvršavaju svaku liniju koda u programu. Međutim, ispravno izvršavanje iskaza za jednu ulaznu vrednost, ne znači da će se iskaz ispravno izvršiti i za neku drugu ulaznu vrednost. Ovo je osnovni nedostatak ove tehnike. Neefikasnost ove tehnike se naročito primećuje kako iskazi postaju složeniji. Tada raste i broj ispitnih slučajeva kojima se obezbeđuje izvršavanje svake linije u kodu.

Pokrivenost odluka (eng. Decision Coverage), je tehnika koja podrazumeva kreiranje ispitnih slučajeva tako da se svaka od različitih grana uslovnog iskaza izvrši bar jednom. Pokrivanje odluka garantuje pokrivanje iskaza, pa je samim tim efikasniji metod.

Pokrivenost uslova (eng. Condition Coverage) podrazumeva ispitne slučajeve kod kojih svi uslovi (i negacije uslova) moraju biti zadovoljeni barem jednom. Ova tehnika ne garantuje pokrivenost svih odluka, pa samim tim ni pokrivanje svih iskaza u programu.

2.6.2 Prednosti i nedostaci ispitivanja metodom bele kutije

Osnovne prednosti ispitivanja metodom bele kutije su:

Uvid u programski kod omogućuje kreiranje efikasnijih ispitnih slučajeva.

Ispitivanje metodom bele kutije pomaže u optimizaciji koda, tako što omogućuje pronalaženje skrivenih grešaka i njihovo otklanjanje. Ovaj vid ispitivanja omogućuje uklanjanje suvišnih linija koda koje mogu dovesti do nepredviđenog ponašanja programa.

Osnovni nedostaci ispitivanja metodom bele kutije su:

Ispitivanje metodom bele kutije unosi kompleksnost u proces ispitivanja jer zahteva da osoba koja kreira ispitne slučajeve poseduje visok nivo znanja o sistemu koji se ispituje, kao i odlično poznavanje programiranja.

U nekim slučajevima nije moguće ispitati svaki mogući uslov (ili iskaz) u programu, tako da neki uslovi ostaju neispitani.

Ovim vidom ispitivanja neke nedostajuće funkcionalnosti u programu mogu ostati neotkrivene.

Pošto ovaj vid ispitivanja zahteva dobro obučene i stručne inženjere, obično je skuplje od ostalih vrsta ispitivanja.

2.6.3 Ispitivanje metodom crne kutije

Ispitivanje metodom crne kutije, je strategija koja zahteva da se program posmatra kao zatvoreni sistem kod kojeg se analizira odziv na određenu pobudu. Ovaj vid ispitivanja ne zahteva poznavanje unutrašnjeg dizajna programskog koda niti analizu koda, već samo osobine definisane specifikacijom softvera koji se ispituje. Zbog toga se ova metodologija često naziva i funkcionalno ispitivanje jer uzima u obzir samo ponašanje, odnosno funkcionalnost softvera.

Specifikacija softvera mora biti tačna, razumljiva i potpuna. Čest uzrok postojanja grešaka u programskom kodu jeste nepotpuna i neodređena specifikacija softvera [11].

Ulazni podaci za ispitivanje softvera se definišu sa ciljem da povećaju verovatnoću pronalaženja greške i da smanje broj ispitnih slučajeva, a da je pri tom obuhvaćeno ispitivanje kompletne funkcionalnosti ispitivanog sistema.

Tehnike koje se koriste kod ispitivanja metodom crne kutije su:

- Deljenje na klase ekvivalencije
- Analiza graničnih vrednosti

Tehnika podele na klase ekvivalencije se temelji na ideji da se ulazni podaci mogu razvrstati u klase, tako da se za sve pripadnike jedne klase program ponaša na isti ili sličan način. Te klase se nazivaju klasama ekvivalencije. Klase ekvivalencije bi trebalo da budu međusobno disjunktne i da pokrivaju ceo skup ulaza. Jedna klasa ekvivalencije može biti jedan numerički podatak ili grupa podataka. Formiranje klasa ekvivalencije se izvodi na osnovu svih uslova iz specifikacije koji se odnose na ulaze programa. Za svaki uslov se na osnovu zadovoljenosti tog uslova posmatraju dve grupe klasa:

- validne (legalne) koje obuhvataju dozvoljene situacije
- nevalidne (nelegalne) koje obuhvataju sve ostale situacije.

Nakon formiranja klasa ekvivalencije, formiraju se ispitni slučajevi za sve validne i sve nevalidne klase. Cilj je da se kod validnih klasa jedan ispitni slučaj primeni na što više validnih klasa. Sa druge strane, za svaku nevalidnu klasu mora se napisati poseban ispitni slučaj kako bi se izbeglo da jedan neregularan ulazni podatak maskira neki drugi koji je takođe nevalidan.

U radu sa klasama ekvivalencije često se pojavljuju greške zbog neadekvatnog definisanja granica klasa. Zbog toga je uvedena analiza graničnih vrednosti koja se primenjuje pri generisanju ispitnih slučajeva u cilju potpune podele opsega ulaznih podataka. Granična vrednost se definiše kao minimalna, odnosno maksimalna, vrednost iz skupa ulaza za određeni sistem.

Ideja je da se ispitni slučajevi biraju na granicama klasa ekvivalencije. Pri tome se vodi računa ne samo u ulaznim uslovima već i o izlaznim podacima iz programa.

U analizi graničnih vrednosti koriste se sledeći tipovi ispitnih slučajeva:

- dva validna (legalna) ispitna slučaja koja pokrivaju krajnje granice
- dva nevalidna (nelegalna) ispitna slučaja neposredno preko granica opsega.

2.6.4 Prednosti i nedostaci ispitivanja metodom crne kutije

Osnovne prednosti ispitivanja metodom crne kutije su:

Ispitivanje metodom crne kutije je veoma efikasno za ispitivanje velikih delova programskog koda nego ispitivanje metodom bele kutije.

Osoba koja formira ispitne slučajeve ne mora imati nikakvo znanje o načinu na koji je softver implementiran.

Ispitni slučajevi se mogu formirati odmah nakon završetka funkcionalnih specifikacija.

Osnovni nedostaci ispitivanje metodom crne kutije su:

Ispitivanje metodom crne kutije može biti neefikasno zbog činjenice da osoba koja kreira ispitne slučajeve ima ograničeno znanje o softveru koji se ispituje. Zbog nepoznavanja unutrašnje strukture koda može se propustiti ispitivanje nekog dela.

Ova vrsta ispitivanja se obično vrši za mali skup ulaznih vrednosti, jer bi ispitivanje ponašanja sistema za veliki broj ulaznih vrednosti trajalo jako dugo.

2.7 Ručno i automatsko ispitivanje

Prema načinu izvršavanja ispitnih slučajeva proces ispitivanja se deli na: ručno i automatsko ispitivanje.

Osnove ručnog, odnosno manuelnog, ispitivanja počivaju na tome da su njegovi nosioci ljudi. Ručno ispitivanje sadrži listu ispitnih slučajeva koje izvršava čovek i čiji rezultati se unose ručno. Prilikom manuelnog ispitivanja, osoba koja izvršava ispitne slučajeve koristi softver kao što bi to radio krajnji korisnik, i odlučuje da li se softver ponaša na očekivani način.

Manuelno ispitivanje je pogodnije za manje projekte nego za ispitivanje velikih sistema.

Ručno izvršavanje je neefikasno sa stanovišta utrošenog vremena i pouzdanosti. Pošto je zavisno od ljudske interpretacije podložno je greškama test inženjera prilikom procene ispravnosti sistema, usled npr. nedovoljnog poznavanja specifikacije sistema, iskustva ili stanja test inženjera.

Automatsko ispitivanje podrazumeva korišćenje specijalnog softvera, nezavisnog od softvera koji je predmet ispitivanja, za kontrolu izvršavanja ispitnih slučajeva, poređenje dobijenih i očekivanih rezultata, dovođenja sistema u stanja neophodna za izvršavanje ispitnih slučajeva i druge kontrolne i aktivnosti vezane za izveštavanje [9].

Automatizacijom procesa ispitivanja prevazilaze se skoro svi problemi koji se javljaju kod manuelnog ispitivanja. Automatsko ispitivanje je brže, efikasnije, tačnije i preciznije.

Automatizaciji procesa ispitivanja se najčešće sreće kod softvera koji imaju dug period održavanja, jer čak i najmanje promene u novoj verziji koda mogu izazvati kvar aplikacije, iako je u prethodnim verzijama radila na očekivan način.

Automatizacija ispitivanja softvera može se sprovesti na različite načine primenom neke od postojećih metodologija. Svaka od njih ima neke prednosti i nedostatke tako da se za rešavanje konkretnih problema često kombinuje nekoliko različitih metodologija.

Dva pristupa koja se najčešće koriste su:

- Grafička korisnička sprega (GUI) , testni okvir koji generiše događaje kao što su pritisci tastera miša ili tastature, i prati promene rezultata u korisničkom interfejsu, kako bi potvrdio da je primećeno ponašanje programa ispravno.
- API driven testing, testni okvir koji koristi programsku spregu ka aplikaciji kako bi proverio njeno ponašanje tokom ispitivanja. Takođe, moguće je testirati javnu spregu ka klasama: Moduli ili biblioteke se ispituju različitim ulaznim argumentima kako bi se potvrdilo da su vraćeni rezultati ispravni.

Automatsko ispitivanje je isplativo na duži rok, naročito ako se koristi u regresivnom ispitivanju. Dva ključna pitanja koja se izdvajaju kod automatizacije ispitivanja su šta i kada automatizovati. Ove odluke donosi ispitni ili razvojni tim. Uspeh automatizacije u velikoj meri zavisi od izbora karakteristike proizvoda za automatizaciju. Automatizaciju nestabilne funkcionalnosti ili funkcionalnosti koja prolazi kroz promene treba izbegavati [12].

3. Koncept rešenja

Ispitivanje je proces koji obuhvata veliki broj aktivnosti. Kao što postoji životni ciklus razvoja softvera (eng. Software Development Life Cycle - SDLC), tako postoji i životni ciklus ispitivanja softvera (eng. Software Testing Life Cycle - STLC).

Ispitivanje softvera je proces koji se mora isplanirati, isprojektovati, implementirati i dokumentovati. Ispitivanje ima dve svrhe, jer se osim detektovanja grešaka u softveru proverava i kvalitet softvera. Čitav proces se izvodi kroz niz koraka, odnosno faza. Svaki korak se karakteriše zadacima koji se u okviru njega izvršavaju i ciljevima koje je potrebno postići. Faze STLC se mogu razlikovati od projekta do projekta, međutim osnova je uvek ista.

3.1 Faze životnog ciklusa ispitivanja softvera

Životni ciklus ispitivanja softvera (STLC) se najčešće sastoji od sledećih faza:

- Planiranje procesa ispitivanja
- Dizajn ispitnih slučajeva
- Izvršavanje ispitnih slučajeva
- Aktivnosti nakon izvršavanja, pregled i zaključci

U nastavku poglavlja su ukratko predstavljene karakteristike pomenutih faza.

3.1.1 Planiranje procesa ispitivanja

Planiranje predstavlja prvi korak u svakom procesu rada. Planiranje procesa ispitivanja predstavlja pripremu za ceo proces. U ovoj fazi potrebno je sagledati šta je sve potrebno uraditi i na koji način. Rezultat planiranja predstavlja najopštiji pogled na sistem koji se ispituje, aktivnosti koje će biti sprovedene tokom ispitivanja, kao i strategije i alate koji će biti korišćeni. Pored toga bitno je isplanirati proces komunikacije kojim se definiše ko će kome prijavljivati

probleme, na koji način će se dodeljivati pronađeni problemi projektnom timu, na koji način će se vraćati test timu radi provere i slično.

3.1.2 Dizajn ispitnih slučajeva

Kada se definiše šta je potrebno uraditi, pristupa se detaljnoj specifikaciji načina na koji će se aktivnosti predviđene planom izvršiti, i formulišu se konkretna uputstva kako će se vršiti ispitivanje sistema. Tokom ove faze se analizira sistem koji će biti ispitivan, identifikuje se šta je potrebno testirati i na koji način. Kao rezultat ovih aktivnosti kreira se skup ispitnih slučajeva i procedura koje će biti korišćene tokom ispitivanja sistema. Takođe, u ovoj fazi se vrši priprema svih podataka koji su neophodni za ispitivanje. Paralelno sa ovim aktivnostima često se vrši uspostavljanje ispitnog okruženja. Ispitno okruženje se priprema sa ciljem da simulira okruženje koje imaju krajnji korisnici.

3.1.3 Izvršavanje ispitnih slučajeva

Izvršavanje ispitnog slučaja podrazumeva izvršavanje svih koraka navedenih u ispitnom slučaju. Nakon izvršenja, porede se dobijeni i očekivani rezultati. Na osnovu dobijenih rezultata određuje se status testa. Ukoliko se svi koraci u ispitnom slučaju uspešno izvrše, i dobijeni rezultati se slažu sa očekivanim, ispitni slučaj se karakteriše kao uspešan, odnosno, kaže se da je prošao (eng. PASS). Ukoliko je izvršavanje ispitnog slučaja neuspešno, odnosno, ako test padne (eng. FAIL), razlog se prijavljuje programerskom timu na dalju analizu.

3.1.4 Aktivnosti nakon izvršavanja

Aktivnosti nakon izvršavanja kojima se zaokružuje proces ispitivanja su:

- Praćenje statusa problema
- Izveštavanje

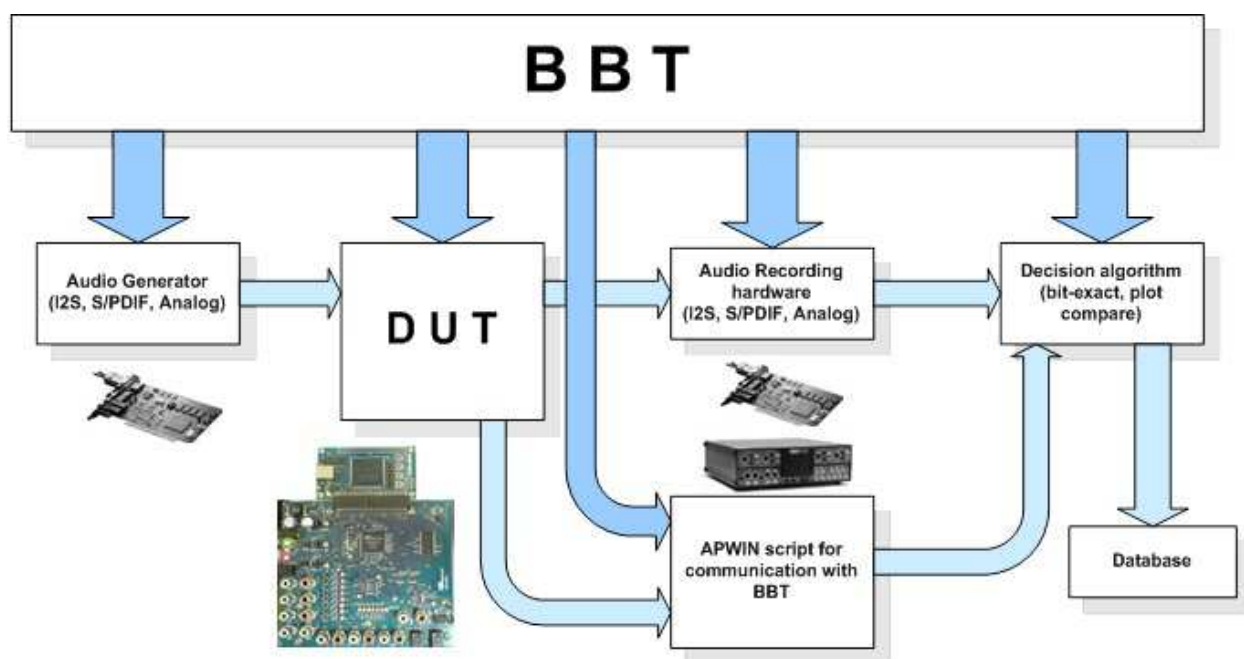
Praćenje statusa problema se često izvršava i paralelno sa fazom izvršavanja ispitnih slučajeva i obuhvata praćenje prijavljenih problema što uključuje rešavanje problema od strane programerskog tima kao i potvrđivanje da je problem rešen od strane test tima. Završna aktivnost u procesu ispitivanja je kreiranje izveštaja kojim se opisuje šta je ispitivano i potvrđuje da je softver spreman za korišćenje u skladu sa kriterijumima kvaliteta koji su definisani u test planu.

3.2 BBT

U svrhe BBT ispitivanja koristi se poseban alat RT-Executor [13]. Ovaj alat je razvijen na katedri za računarsku tehniku i računarske komunikacije Fakulteta tehničkih nauka u Novom Sadu. RT-Executor je sofisticiran alat namenjen za kontrolu, razvoj i izvršavanje BBT automatizovanog test rešenja. Organizovan je tako da se sastoji od jezgra aplikacije i posebnih

modula u kojima su implementirane sve funkcionalnosti. Dodavanje novih funkcionalnosti postiže se dodavanjem nove dinamičke biblioteke (eng. *Dynamic link library - dll*) sa definicijom novog modula u instalacioni direktorijum. Prilikom pokretanja aplikacije BBT sistem dinamički uvlači sve uređaje smeštene u instalacioni direktorijum. Na ovaj način se dobija na modularnosti, jer dodavanje novih funkcionalnosti ne zahteva izmene u jezgru aplikacije. Aplikacija omogućuje integraciju sa mernim uređajima, signal generatorima, uređajima za prikupljanje podataka u priključu i koristi maniru (eng. *Plug & Play*). Alat sadrži skup biblioteka za analizu video i audio podataka, koristi baze podataka, kako bi se kontrolisala prava pristupa samoj aplikaciji, kao i u izveštavanju.

Prednosti BBT alata su brojne, od relativno jednostavne sintakse za pisanje ispitnih slučajeva, do lakog korišćenja, prenosivosti, mnoštva ugrađenih alata koji pojednostavljaju proces ispitivanja, sve do preglednih izveštaja o statusu procesa ispitivanja. Ovaj alat omogućuje brz razvoj ispitnih slučajeva, a ugrađeni alati omogućuju izvršavanje širokog spektra operacija. Standardna procedura za ispitivanje izlaza sa ciljne platforme (konfiguracija, snimanje izlaza, vremensko poravnanje i bit-identično poređenje), korišćenjem BBT alata, prilično je jednostavna, a jednom razvijeni ispitni slučajevi mogu se ponoviti neograničen broj puta, kao i na proizvoljnom računaru. Na sledećoj slici dat je primer ispitnog okruženja.



Slika 3.1 – Primer ispitnog okruženja

U okviru rada korišćenjem BBT alata izvršava se konfiguracija ciljne platforme, reprodukcija i snimanje izlaza i vremensko poravnanje snimljenih izlaza. Upoređivanje snimljenih izlaza sa referentnim testnim vektorima urađeno je posebno.

3.3 Ciljna platforma

Kao ciljni procesor koristi se CS48944 [15], digitalni signal procesor firme Cirrus Logic. Digitalni signal procesori iz familije CS49xxx kompanije Cirrus Logic namenjeni su obradi audio signala u uređajima potrošačke elektronike. Procesori iz ove familije zasnovani su na Crystal arhitekturi DSP jezgra, a razlike između pojedinih modela se odnose na broj jezgara, radni takt i količinu interne memorije. CS48944 je četvorोजezgarni procesor sa radnim taktom od 300MHz, što ga čini jednim od najmoćnijih procesora iz Crystal familije. Zasnovan je na unapređenoj Harvard arhitekturi koja podrazumeva odvojenu memoriju za podatke i memoriju za instrukcije. Dodatno, memorija za smeštanje podataka je podeljena u dva segmenta. Svako jezgro predstavlja tridesetdvobitni programabilan DSP koji radi sa aritmetikom u fiksnoj zarezu. Na sledećoj slici prikazan je funkcionalni blok dijagram CS49834 procesora, a u tabeli su dati podaci o memoriji.

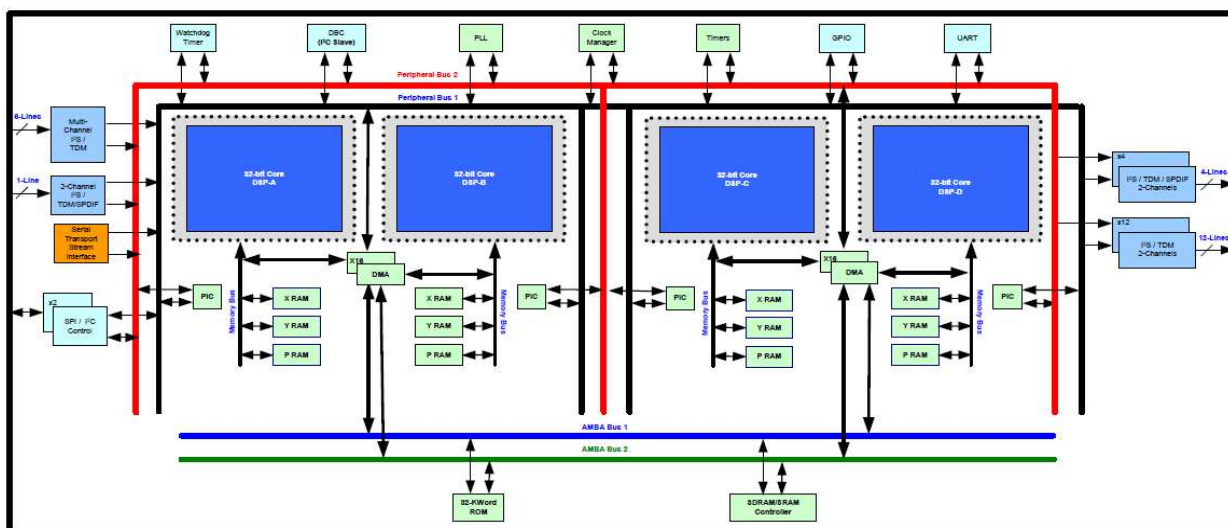


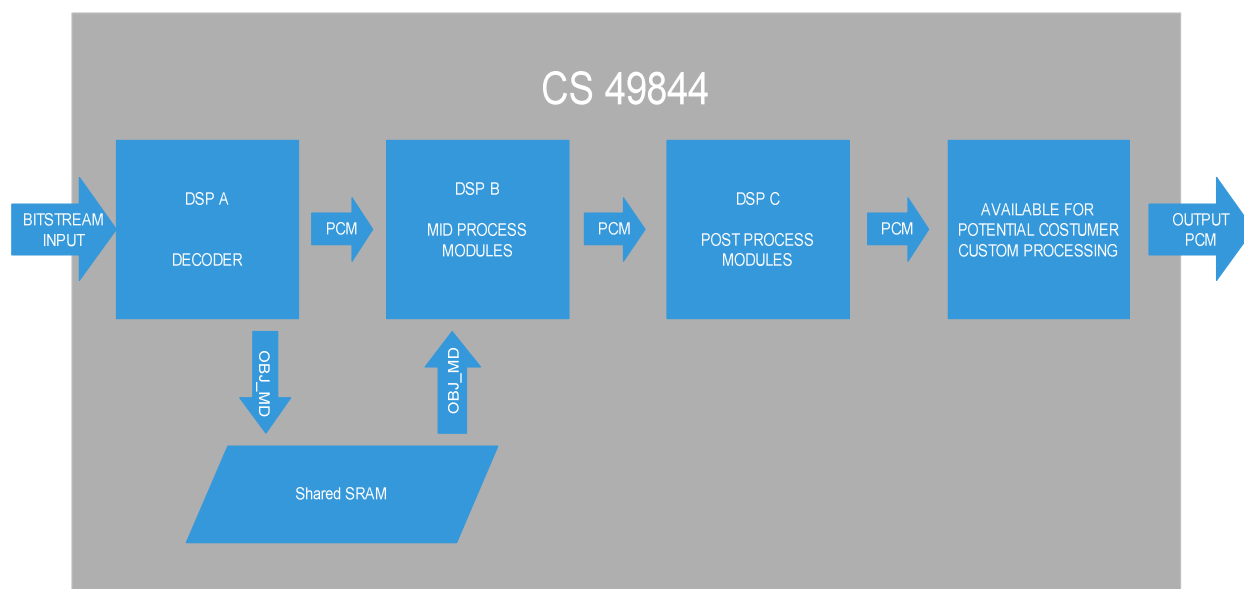
Figure 4-2. CS49844 Top-Level Block Diagram

Slika 3.2 – Blok dijagram procesora CS49844

Tip memorije	Jezgro A	Jezgro B	Jezgro C	Jezgro D
X, Y, P	60kWord SRAM	60kWord SRAM	60kWord SRAM	60kWord SRAM

Tabela 3-1: Raspoloživa memorija procesora CS49844

Kako je u poglavlju 2.1 pomenuto da su nove audio tehnologije veoma zahtevne u pogledu memorije i MIPS-a. Na sledećoj slici je data pojednostavljena blok šema organizacije modula jedne audio tehnologije na ciljnoj platformi, a u nastavku kratak opis funkcionisanja.



Slika 3.3 – Blok šema funkcionisnja složene audio tehnologije na ciljnoj platformi

Na prvo procesorsko jezgro, jezgro A, „spušta“ se firmware jednog od dekodera. U zavisnosti od ulaznog bitskog toka bira se koji dekodер će biti korišćen. Ukoliko se u bitskom toku nalaze i audio objekti, meta podaci iz audio objekata se parsiraju, a zatim smeštaju u deljenu memoriju, odakle će ih kasnije koristiti modul na jezgru B. Sa druge strane, ukoliko se radi sa kanalima, PCM-ovi se prosleđuju na jezgro B.

Na jezgru B se izvršavaju se moduli među obrade (*Mid-Process Module - MPM*). Kada je reč o tehnologijama zasnovanim na kanalima, ovi moduli najčešće menjaju broj izlaznih kanala, tako što ga povećavaju (*upmix*), odnosno smanjuju (*downmix*). Kod audio tehnologija zasnovanim na objektima na jezgru B se izvršavaju moduli zaduženi za renderovanje na izlaznu konfiguraciju zvučnika. Renderovanje se obavlja koristeći i PCM-ove i meta podatke sačuvane u deljenoj memoriji. Na jezgru C izvršavaju se moduli završne obrade, ovi moduli vrše završnu obradu nad podacima koji su rezultat prethodnih nivoa obrade. U sledećoj tabeli dati su podaci o potrošnje resursa ciljne platforme.

	Jezgro A	Jezgro B	Jezgro C	Jezgro D
X [reči]	51216	57332	40548	
Y [reči]	50654	37000	42868	
P [reči]	50504	17222	29132	
MIPS	~220	~200	~180	

Tabela 3-2 – Prikaz potrošnje resursa ciljne platforme

Podaci dati u tabeli potvrđuju da su savremene audio tehnologije izuzetno zahtevne. Čak i kada se implementiraju u asemblerskom jeziku, a pritom i maksimalno optimizuju.

Za reprodukciju i snimanje signala korišćena je zvučna kartica Echo Audio [13].

Na slici Slika 3.4 prikazano je ispitno okruženje. Na levoj strani slike je razvojna ploča CDB49X, u čijem centralnom delu je pozicioniran digitalni signal procesor CS48944, a sa desne strane se nalazi Echo Audio zvučna kartica. Komunikacija između računara i razvojne ploče ostvaruje se pomoću USB sprege, dok se Echo Audio kartica sa računarom povezuje pomoću sprege IEEE 1394, takozvane FireWire. Razvojna ploča i zvučna kartica povezane su pomoću I2S serijske sprege, koja se koristi za digitalan prenos audio signala.



Slika 3.4 – Ispitno okruženje

4. Implementacija

U ovom poglavlju izneti su detalji implementacije datog rešenja. Rešenje se zasniva na konceptu koji je predstavljen u prethodnom poglavlju. U ovom poglavlju su detaljno su opisani postupci primenjeni u svakoj od faza u procesu ispitivanja.

4.1 Planiranje procesa ispitivanja

Kao što je pomenuto u odeljku 3.1.1, u fazi planiranja se definišu aktivnosti koje će biti sprovedene tokom procesa ispitivanja, kao i strategije i alati koji će biti korišćeni.

U okviru ovog rada urađeno je funkcionalno ispitivanje metodom crne kutije (eng. Black Box Testing). U svrhe BBT ispitivanja, koristi se RT-Executor kontrolna aplikacija. Ovaj alat omogućuje relativno brz razvoj ispitnih slučajeva, sadrži mnoštvo ugrađenih alata koji omogućuju izvršavanje širokog spektra operacija, što ga čini vrlo pogodnim za ispitivanje audio tehnologija.

Od proizvođača tehnologije koja je bila predmet ispitivanja dobijena je dokumentacija koji predstavlja ispitnu proceduru. Ova dokumentacija sadrži specifikacije ispitnih slučajeva, ulazne podatke i ostale dokumente potrebne za verifikaciju implementacije ispitivanje audio tehnologije. Ova dokumentacija je dostupan samo softverskim kompanijama koje žele da verifikuju svoju implementaciju i da dobiju sertifikat od kompanije koja je licencirala tu tehnologiju.

Za uspešnu verifikaciju implementiranog rešenja neophodno je da se svi ispitni slučajevi uspešno izvrše, i da implementirano rešenje odgovori na zahteve predviđene specifikacijom. Od ukupno 604 testna slučaja, bit-identična ispitivanja se vrše za 594, dok se preostalih 10 tretiraju kao slušni testovi. Ispitni slučajevi kod kojih se proverava bit-identičnost organizovani su u 17

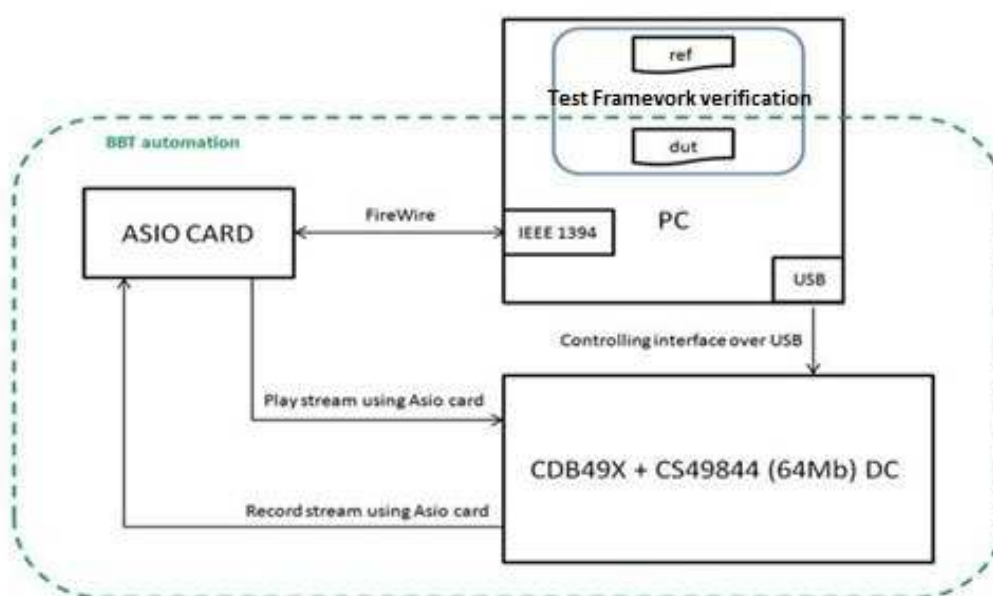
grupa, gde su ispitni slučajevi iz jedne grupe namenjeni za ispitivanje jedne funkcionalnosti ispitivane audio tehnologije.

Da bi se omogućilo potpuno automatizovano ispitivanje prihvatljivosti implementiranog rešenja, proizvođač tehnologije je obezbedio i posebnu softversku komponentu, odnosno testni okvir, koja omogućuje spregu pomoću komandne linije (eng. Command-Line Interface - CLI) za izvršavanje ispitivanja. Osim za ispitivanje prihvatljivosti, testni okvir se koristi i za generisanje referentnih signala, koji će biti upoređivani sa snimljenim signalima. Testni okvir je potpuno automatizovan i ne zahteva dalju interakciju sa korisnikom nakon pokretanja. Testni okvir obuhvata skripte za automatizaciju, specifikacije ispitnih slučajeva, kao i neophodne ulazne signale.

Osnovni delovi ovog okvira su:

- Skup namenskih alata koji se koriste za evaluaciju,
- Skup namenskih .xml datoteka koje sadrže specifikacije ispitnih slučajeva,
- Skup namenskih test signala,
- Okvir zasnovan na Python skript jeziku, čija uloga je da pomoću pomenutih alata, na osnovu specifikacije, generiše referentne izlaze ili vrši upoređivanje referentnih izlaza sa izlazima snimljenim na ciljnoj platformi.

Ceo postupak ispitivanja podeljen je u dva koraka. Prvi korak obuhvata snimanje izlaza sa ciljne platforme, i izvodi se upotrebom BBT alata. Kada se snime izlazni testni vektori, za željeni ispitni slučaj, prelazi se na analizu snimljenog izlaza. Ova analiza obavlja se upotrebom testnog okvira. Na sledećoj slici prikazan je blok dijagram testnog okruženja.



Slika 4.1 – Blok dijagram testnog okruženja

4.2 Dizajn ispitnih slučajeva

Faza STLC u koja obuhvata razvoj ispitnih slučajeva zahtevala je najviše truda, i bila je najsloženija za realizaciju.

4.2.1 Organizacija radnog okruženja

Početni korak u fazi dizajna, odnosno razvoja, ispitnih slučajeva podrazumevao je organizaciju direktorijuma radnog okruženja. Pošto i testni okvir i BBT alat imaju određene specifičnosti, i unose određene zahteve, radno okruženje moralo je biti organizovano tako da odgovori sve zahteve. Organizacija foldera radnog okruženja data je na sledećoj slici.



Name	↑ Ext	Size
[-.]		<DIR>
[docs]		<DIR>
[install]		<DIR>
[projects]		<DIR>
[test_templates]		<DIR>
[testing]		<DIR>
[tests]		<DIR>
[tools]		<DIR>
dap_globals	ini	1.000
test_env	ini	6.550
dap_ReadMe	txt	2.782

Slika 4.2 – Organizacija direktorijuma radnog okruženja

U direktorijumu **docs** su sačuvane tekstualne datoteke koje sadrže instrukcije i uputstva za ceo proces ispitivanja, počev od generisanja referentnih izlaza, generisanja ispitnih slučajeva, uputstava za pokretanje izvršavanja ispitnih slučajeva, kao i uputstava za poređenje referentnih i izlaza snimljenih na ciljnoj platformi.

Direktorijum **install**, sadrži instalacione datoteke svih aplikacija i alata koji su neophodni za izvršavanje procesa ispitivanja, ko što su na primer Python, na kom je zasnovan testni okvir, zatim BBT alat, alati za upoređivanje snimljenih wav datoteka itd.

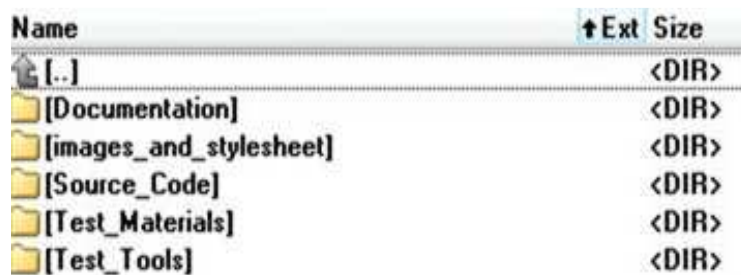
U okviru foldera **projects**, sačuvani su BBT projekti o kojima će biti više reči u narednim poglavljima.

Direktorijum **test_templates** sadrži datoteke pomoću kojih se generišu sami ispitni slučajevi. O ovim datotekama će biti više reči u nastavku poglavlja.

Direktorijum **tests** sadrži ispitne slučajeve organizovane u grupe. Svaki ispitni slučaj sačuvan je u posebnom folderu (testni folder), koji ima naziv kao i ispitni slučaj. Pre izvršavanja ispitnog slučaja u testnom folderu postoji samo datoteka sa ekstenzijom .tst, dok se po izvršenju ispitnog slučaja tu generišu i log datoteke.

U okviru **tools** direktorijuma čuvaju se svi alati, skripte i konfiguracione datoteke koje će se pozivati tokom izvršavanja ispitnih slučajeva.

Testni okvir ima predefinisanu strukturu direktorijuma, koja je prikazana na sledećoj slici.



Name	Ext	Size
[..]		<DIR>
[Documentation]		<DIR>
[images_and_stylesheet]		<DIR>
[Source_Code]		<DIR>
[Test_Materials]		<DIR>
[Test_Tools]		<DIR>

Slika 4.3 – Organizacija direktorijuma testnog okvira

Svi dokumenti neophodni za upoznavanje sa ispitivanom tehnologijom, sa samim testnim okvirom, i druga propratna dokumentacija nalaze se u direktorijumu **Documentation**. Direktorijum **Test_Materials**, sadrži signale koji se koriste kao ulazne datoteke koje se šalju u sistem, i specifikacije ispitnih slučajeva u obliku .xml datoteka. U okviru **Test_Tools** foldera grupisani su svi alati koje testni okvir koristi. Prilikom prvog poziva glavne skripte iz testnog okvira, u okviru **Test_Tools** direktorijuma, generišu se **Test_Report** i **Work** direktorijumi. U direktorijumu **Test_Report** će se čuvati svi izveštaji vezani za rezultate ispitivanja, dok **Work** direktorijum sadrži poddirektorijume nazvane po ispitnom slučaju, u okviru kojih se čuvaju referentni i izlazi snimljeni na ciljnoj platformi, kao i ostale datoteke vezane za ispitne slučajeve.

4.2.2 Skripte za generisanje ispitnih slučajeva

Kao što je ranije pomenuto, direktorijum **test_templates** sadrži skripte i druge datoteke pomoću kojih se generišu BBT testovi i druge datoteke neophodne za pravilno izvršavanje BBT testova pomoću BBT alata.

U poddirektorijumima **Work** direktorijuma sačuvane su tekstualne datoteke u kojima se nalaze sve informacije po kojima je odrđeni ispitni slučaj karakterističan. U ovim datotekama navedeni su ulazni testni vektori, vrednosti parametara za konfigurisanje firmware-a i drugo. Kako bi sve ove informacije, za sve testne slučajeve, nalazile sumirane na jednom mestu, napisana je Python skripta `test_specification_parser.py`, koja parsira prethodno pomenute datoteke i kreiraju Excel tabelu. U ovoj tabeli, u prvoj koloni izlistani su svi ispitni slučajevi, dok su u narednim kolonama upisane vrednosti svih parametara neophodnih za odgovarajući testni slučaj. Na sledećoj slici prikazan je sadržaj jedne od ovih datoteka.

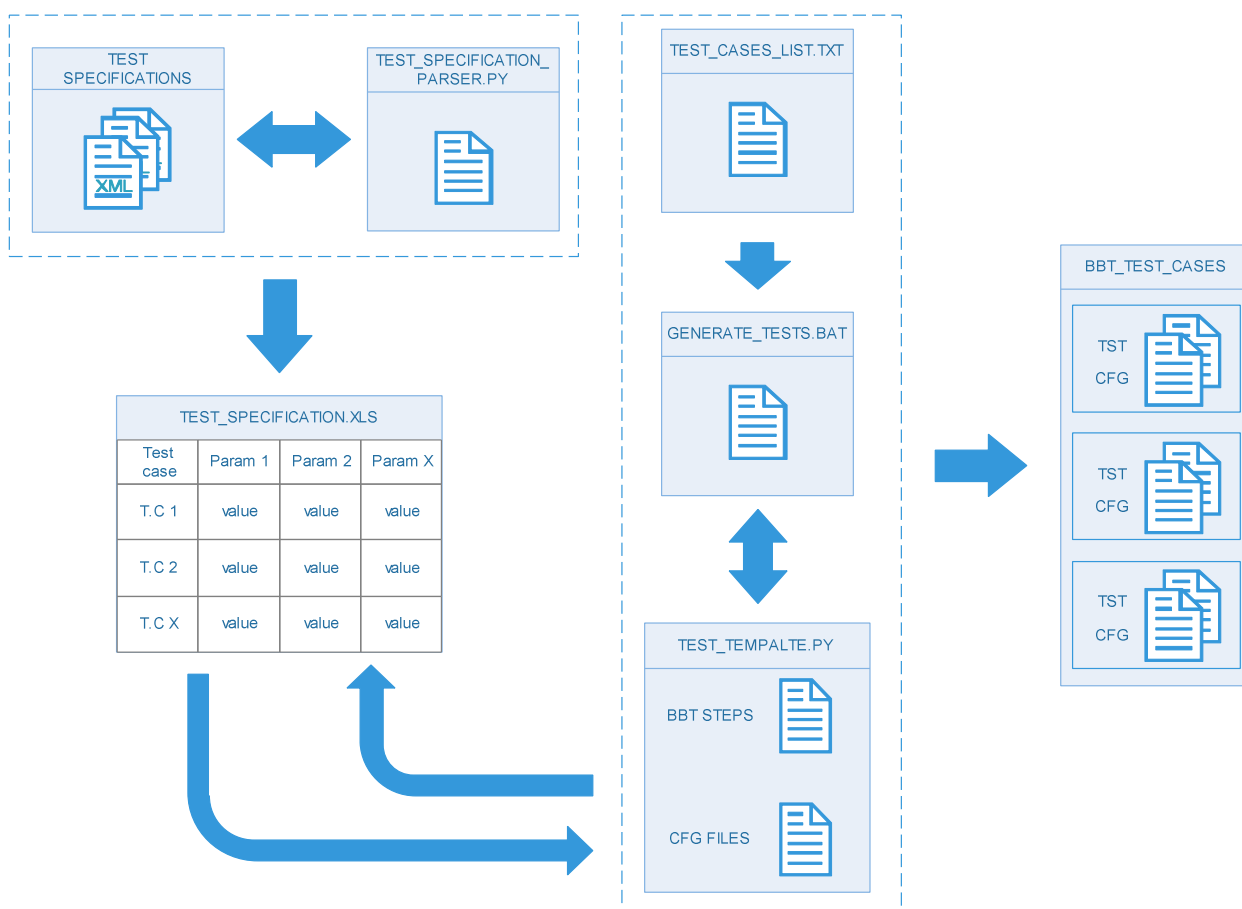
```

</testspec>
<testspec id="sys_alg_volume_and_ieq_silencerreset_io2ch_io32k">
  <description>dap system test</description>
  <expected_results>The envelope of the output wav should match the reference as the resets should occur at same time.</expected_results>
  <step type="dut">
    <processor>DAP_HT_RCL</processor>
    <file direction="in" type="static">signals/volume_and_ieq_silence_reset_2ch_32k.wav</file>
    <file direction="out" type="dynamic">out-1.wav</file>
    <options>
      <setparam name="regulator-enable">0</setparam>
      <setparam name="surround-boost">0</setparam>
      <setparam name="output-mode">1</setparam>
      <setparam name="volume-levsig-enable">1</setparam>
      <setparam name="volume-levsig-amount">7</setparam>
      <setparam name="volume-mode-lev-enable">0</setparam>
      <setparam name="volume-max-boost">144</setparam>
      <setparam name="surround-decoder-enable">0</setparam>
    </options>
  </step>
  <step type="eval">
    <processor>FREQCMP</processor>
    <file direction="in" type="static">reference/volume_and_ieq/sys_alg_volume_and_ieq_silencerreset_io2ch_io32k.wav</file>
    <file direction="in" type="dynamic">out-1.wav</file>
    <options>
      <peak_limit>0.240000</peak_limit>
      <error_threshold>-96</error_threshold>
      <median_window_length>6</median_window_length>
      <median_peak_limit>0.1</median_peak_limit>
    </options>
  </step>
  <meta>alg</meta>
  <meta>default</meta>
  <meta>volume</meta><meta>and</meta><meta>ieq</meta>
  <meta>shuffle</meta>
  <meta>i2ch</meta>
  <meta>i32k</meta>
  <meta>io2ch</meta>
  <meta>io32k</meta>
  <meta>o2ch</meta>
  <meta>o32k</meta>
  <meta>silencerreset</meta>
  <meta>sys</meta>
</testspec>

```

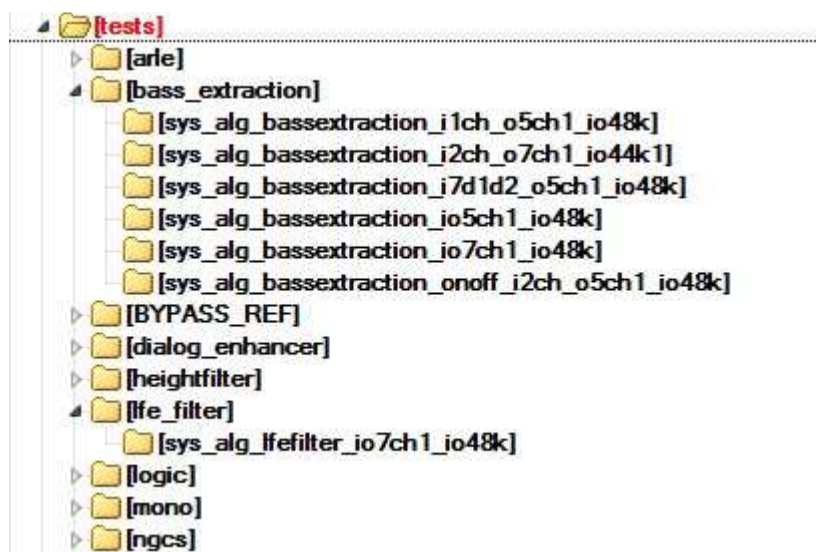
Slika 4.4 – Primer jedne datoteke sa karakteristikama ispitnog slučaja

Na sledećoj slici prikazan je blok dijagram, koji ilustruje ceo proces generisanja BBT ispitnih slučajeva.



Slika 4.5 – Blok dijagram procesa generisanja ispitnih slučajeva

Za generisanje BBT ispitnih slučajeva koristi se Batch skripta `generate_tests.bat`. Ovoj skripti se kao parametar mora proslediti tekstualna datoteka koja sadrži listu ispitnih slučajeva koje treba generisati. Skripta `generate_tests.bat`, poziva `test_template.py` skriptu u kojoj je implementirana logika pomoću koje se generišu BBT testovi. Skripti `test_template.py` se kao parametar prosleđuje prethodno pomenuta Excel tabela. Iz liste ispitnih slučajeva prosleđenih skripti `generate_tests.bat` uzima se jedna po jedna linija, odnosno naziv jednog ispitnog slučaja u jednoj iteraciji. Skripta `test_template.py`, nalazi u Excel tabeli naziv željeni testni slučaj, sa svim njegovim parametrima. Uloga skripte `test_template` je dovstruka. Pored toga što će generisati ispitni slučaj za BBT alat (.tst datoteka), ona će generisati i konfiguracione fajlove (.cfg datoteke). Kao krajnji rezultat izvršavanje skripte `generate_tests.bat` dobijaju se BBT ispitni slučajevi i konfiguracione datoteke za sve ispitne slučajeve navedene u listi testova. Ispitni slučajevi se čuvaju u direktorijumu `tests`, u korenskom direktorijumu ispitnog okruženja, organizovani u podgrupe, i poddirektorijume nazvane po nazivu ispitnog slučaja. Organizacija ispitnih slučajeva u okviru test foldera data je na sledećoj slici.



Slika 4.6 – Organizacija poddirektorijuma *tests* direktorijuma

Na sledećoj slici je prikazan blok dijagram, koji ilustruje korake jednog BBT ispitnog slučaja. Svaki ispitni slučaj ima određene specifičnosti, zato su na dijagramu navedeni najosnovniji koraci koji su uključeni u svaki BBT test.



Slika 4.7 – Blok dijagram jednog BBT testa

Aktivnosti koje se izvrše kroz korak **LOAD**, tiču se konfiguracije ciljne platforme, odnosno „spuštanje“ svih izvršnih datoteka koje su potrebne (firmware svih komponenti koje se paralelno izvršavaju) na procesor. U ovom koraku se takođe vrši i pozivanje konfiguracionih datoteka, odnosno komandi koje su u njima navedene, a pomoću kojih se uključuju odnosno isključuju pojedine funkcionalnosti, šalju vrednosti pojedinih parametara, kao što su razna pojačanja, vrednosti graničnih frekvencija i tako dalje.

Korak **PREPARE_INPUT_STREAM** podrazumeva sve aktivnosti vezane za pripremu ulaznog testnog vektora. U nekim slučajevima, ulazni testni vektor se nalazi u formatu koji se ne može reprodukovati a da se prethodno ne dekoduje. Ponekad se dešava da je neophodno povećati frekvenciju odabiranja kod ulaznog testnog vektora. U nekim slučajevima se zahteva da se dva testna vektora spoje i da se kreira novi ulazni testni vektor. Sve ove i slične aktivnosti, izvršavaju se kroz korake predviđene za blok **PREPARE_INPUT_STREAM**.

U okviru bloka **PLAY_AND_RECORD** navode se koraci pomoću kojih se vrše podešavanja zvučne kartice koja se koristi za reprodukciju i snimanje. U okviru BBT alata, ova kartica je podržana kao jedan od mnogih ugrađenih alata. Neki od parametara koje je neophodno postaviti su putanja i naziv ulaznog testnog vektora, putanja i naziv izlaznog testnog vektora, frekvencija na kojoj se pušta, broj kanala ulaznog i izlaznog testnog vektora. Zvučna kartica koja je korišćena u ovom radu ograničena je da može reprodukovati testne vektore koji sadrže maksimalno osam kanala. Ukoliko ulazni testni vektor sadrži više od osam kanala, reprodukcija i snimanje se mora izvoditi u nekoliko iteracija. U tom slučaju izvršavanje ispitnog slučaja traje

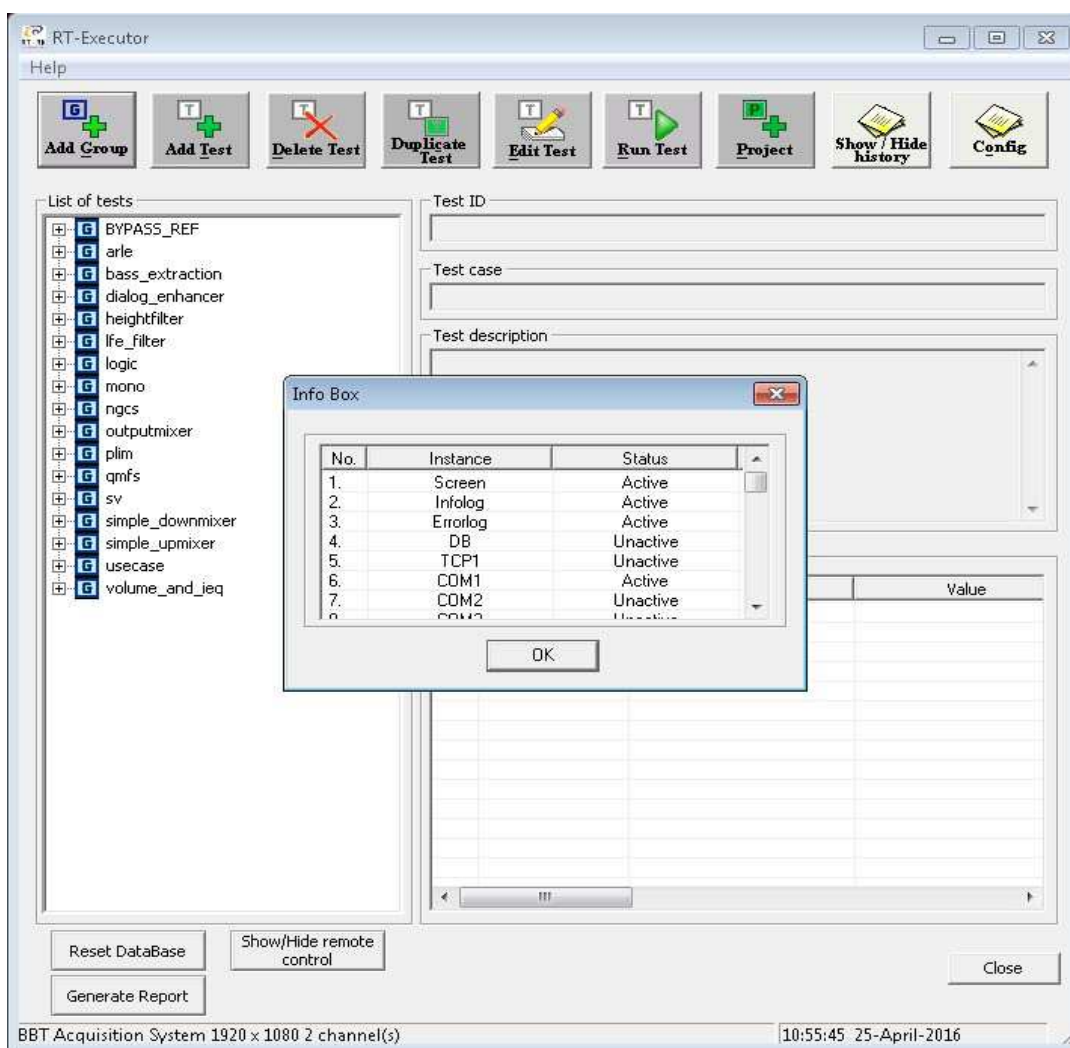
duže, pošto je neophodno pre druge iteracije snimanja ponovo konfigurisati ciljnu platformu na odgovarajući način, ali i izvršiti neke dodatne korake nakon snimanja izlaznog testnog vektora.

Koraci u okviru bloka **PREPARE_OUTPUT_STREAM** odnose se na aktivnosti nakon snimanja izlaznog testnog vektora. Najčešće su to vremensko poravnanje snimljenog izlaza, kombinovanje testnih vektora snimljenih kroz nekoliko iteracija u jedan izlazni testni vektor i slično.

U okviru bloka **CLEAN_UP** vrši se brisanje svih privremenih datoteka koje su generisane tokom izvršavanja testnog slučaja. U suštini, ove aktivnosti nije neophodno izvršiti, međutim pokazalo se da je prisustvo ovog koraka u testovima dobra praksa. Konkretno u ovom radu tokom izvršavanja testova generiše se nekoliko privremenih wav datoteka, koje mogu biti veličine nekoliko stotina mega bajta. Kako ceo skup sadrži 594 testa, memorija koju zauzmu nepotrebne, privremene, datoteke dostiže i nekoliko giga bajta.

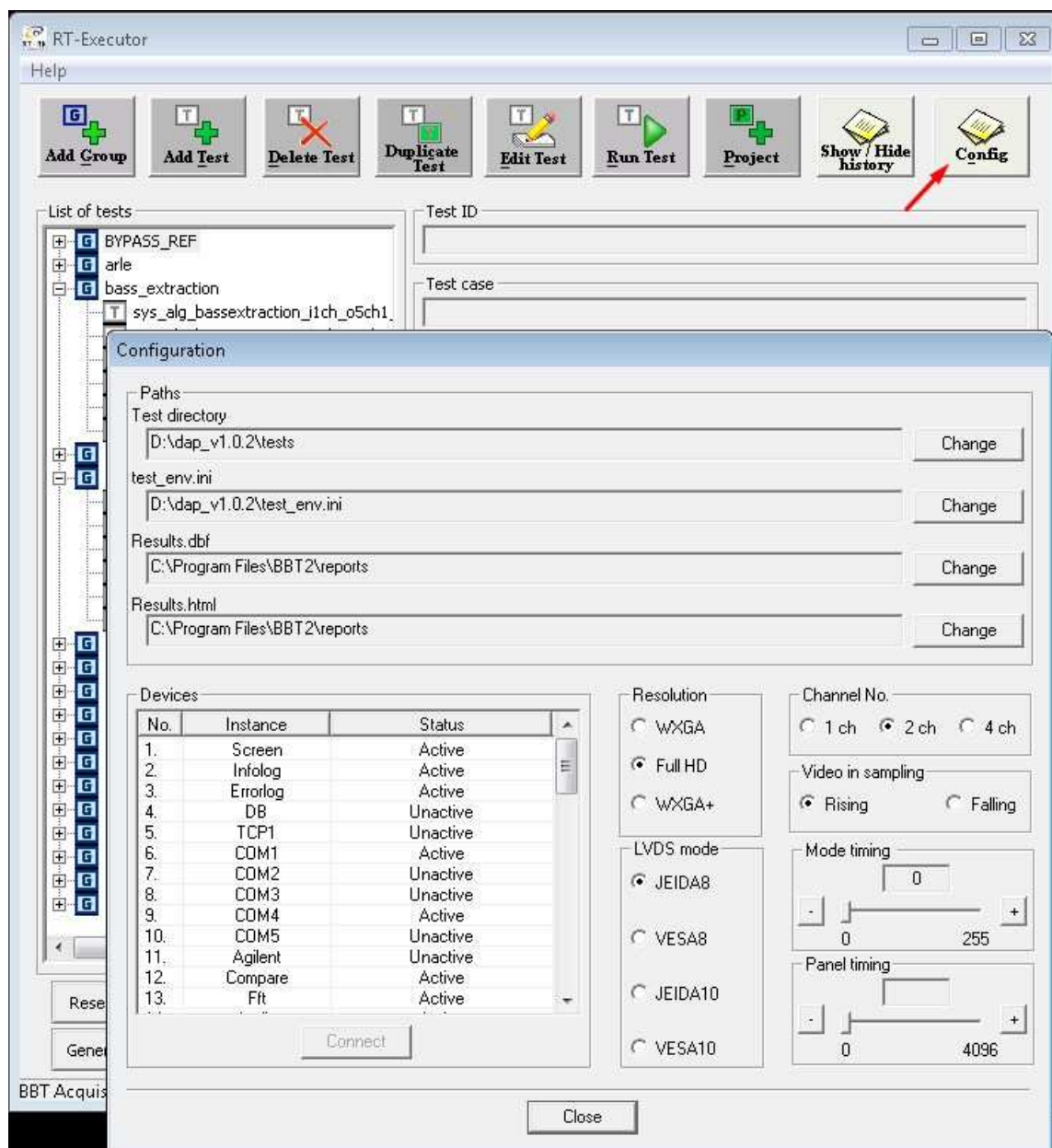
4.3 Izvršavanje ispitnih slučajeva

Za izvršavanje ispitnih slučajeva koristi se BBT aplikacija. Na sledećoj slici prikazan je izgled BBT aplikacije nakon njenog pokretanja.



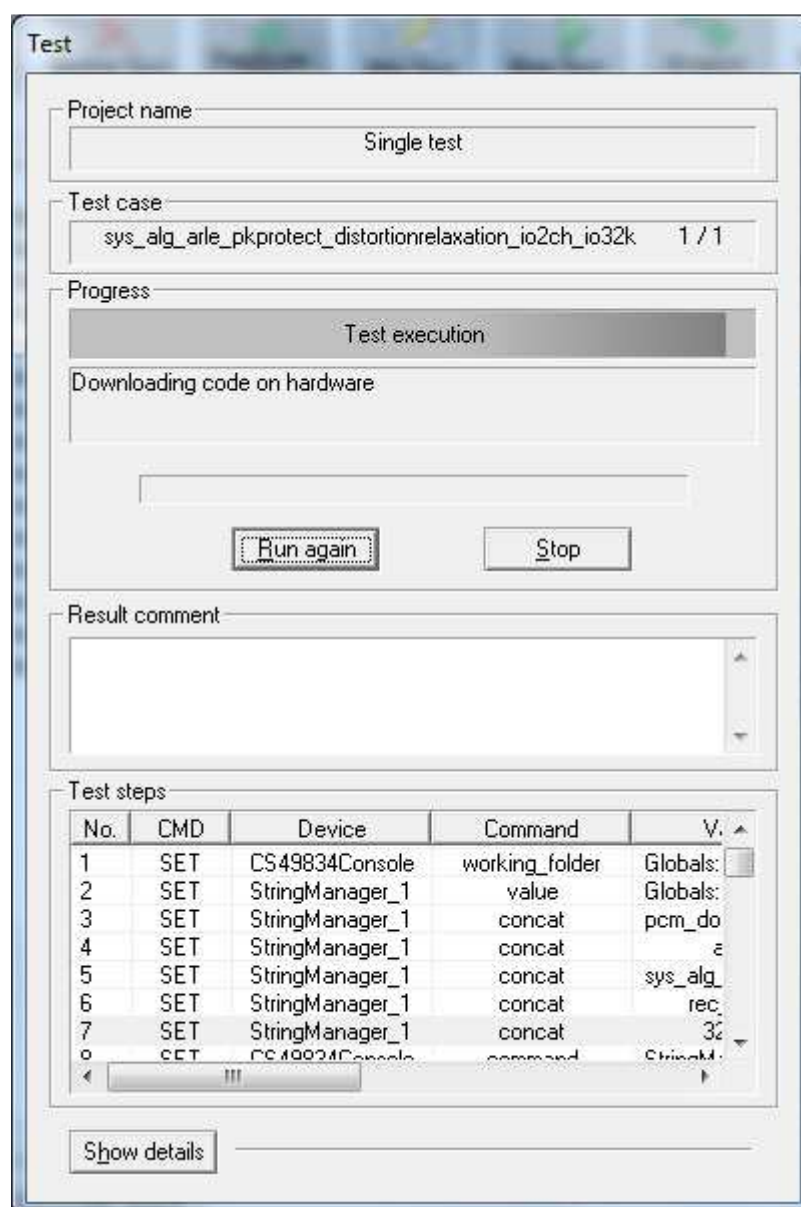
Slika 4.8 – BBT aplikacija nakon pokretanja

Pre izvršavanja testova, neophodno je izvršiti proveru konfiguracije BBT alata. Klikom na *Config* dugme, otvara se *Configuration* dijalog u kom je potrebno postaviti putanju do direktorijuma u kom su sačuvani testovi kao i putanje do direktorijuma u kom će se čuvati rezultati izvršavanja ispitnih slučajeva. Takođe, potrebno je postaviti putanju do *test_env.ini* datoteke u kojoj su navedeni ugrađeni BBT alati koji će biti korišćeni tokom ispitivanja.



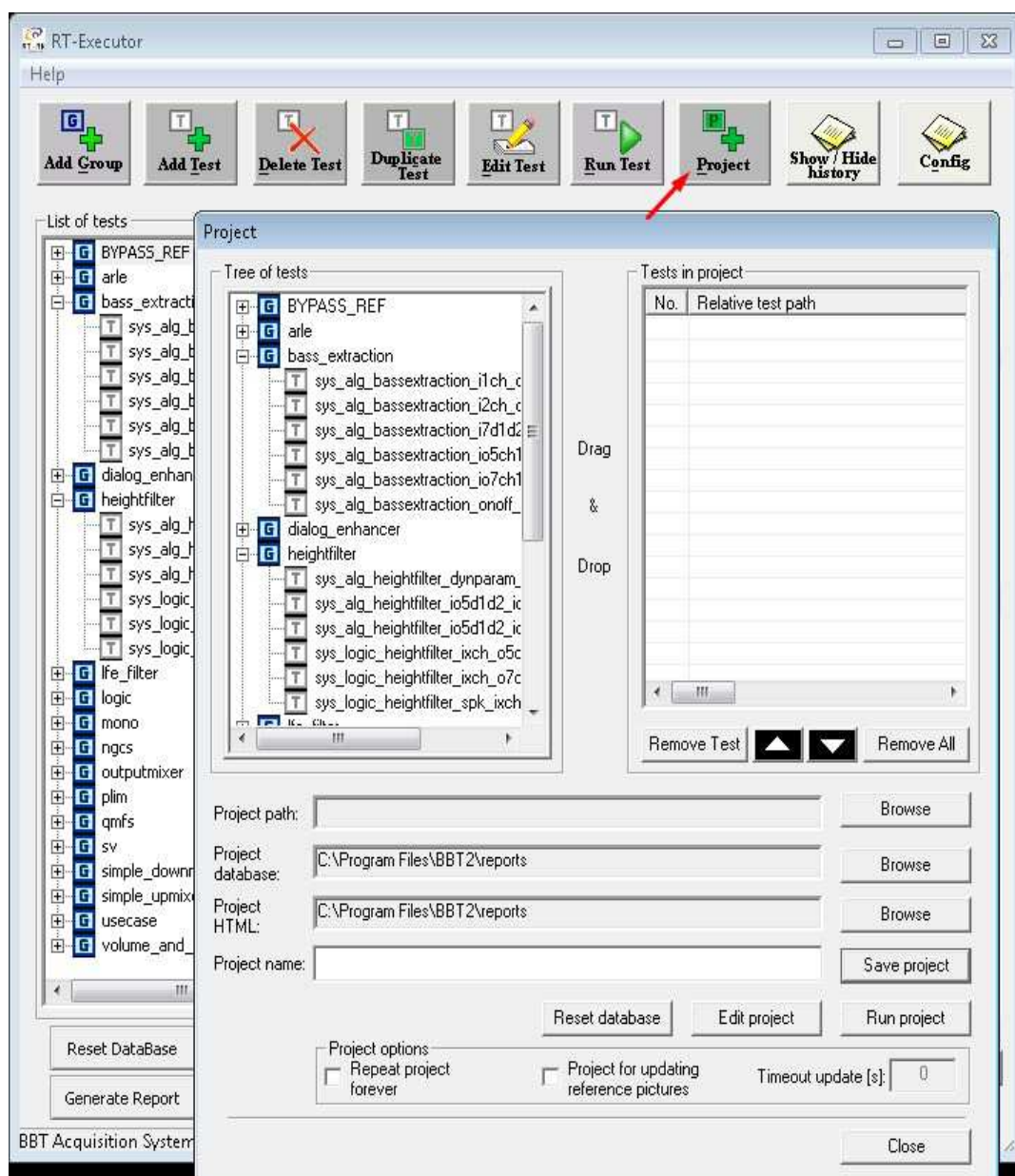
Slika 4.9 – Konfigurisanje BBT alata

BBT alat podržava dva načina ispitivanja, izvršavanjem pojedinačnih ispitnih slučajeva ili izvršavanjem skupa ispitnih slučajeva. Za izvršavanje određenog ispitnog slučaja, dovoljno je izabrati ispitni slučaj iz panela *List of tests* i kliknuti na dugme *Run Test*. Kada se ispitni slučaj pokrene, prikazuje se dodatni prozor kojim se prati napredovanje izvršavanja datog ispitnog slučaja (sledeća slika).



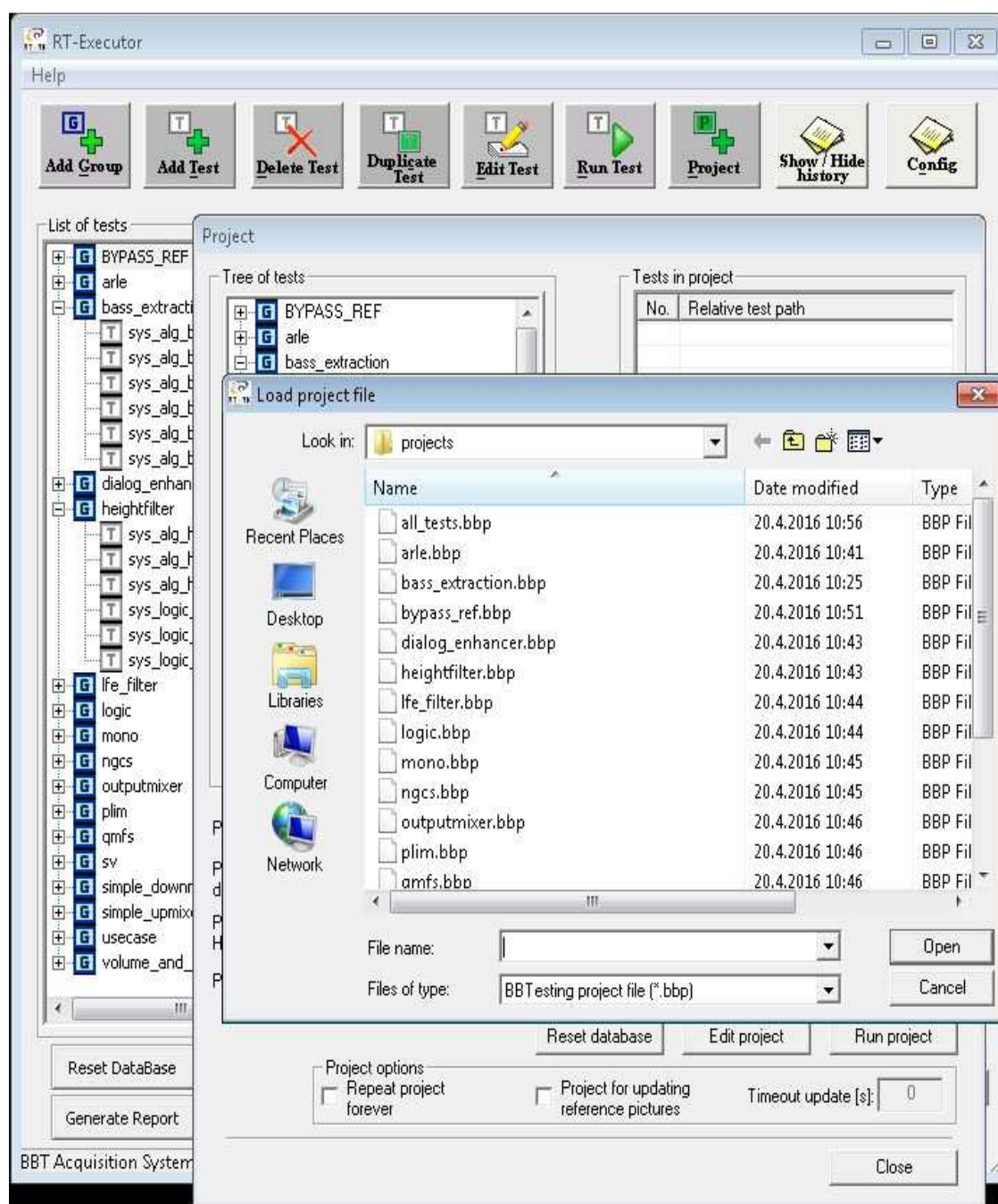
Slika 4.10 - Prikaz prozora u kom se može pratiti napredovanje izvršavanja ispitnog slučaja

Za izvršavanje skupa testova kreiraju se BBT projekti, klikom na dugme *Project* u glavnom prozoru aplikacije. Kreiranje BBT projekta je jednostavno, dovoljno je iz *Tree of tests* panela prevući željene testove u *Tests in project* panel, a zatim izabrati ime projekta i putanju do direktorijuma gde će biti sačuvan, a potom kliknuti na *Save project*.



Slika 4.11 – Kreiranje BBT projekta

Sa druge strane, pritiskom na dugme *Run project* pojavljuje se dijalog *Load project file* iz kog se može pokrenuti neki od postojećih BBT projekata. Pomenute situacije su prikazane na slici. BBT projekti se najčešće kreiraju pomoću BBT alata, ali mogu se kreirati, odnosno uređivati, i pomoću nekog od programa za obradu teksta. Svaki BBT projekat se sastoji od zaglavlja i liste ispitnih slučajeva. U zaglavlju su date informacije o projektu, poput naziva projekta, putanje do lokacije gde će biti sačuvan izveštaj o statusu izvršavanja projekta i slično. Lista ispitnih slučajeva obuhvata listu pojedinačnih ispitnih slučajeva koji će biti izvršeni prilikom izvršavanja izabranog projekta. Na ovaj način se ispitni slučajevi lako organizuju u grupe.



Slika 4.12 – Pokretanje ranije kreiranih BBT projekata

U ovom radu su ispitni slučajevi za bit-identična ispitivanja podeljeni u 16 grupa. Dodatnu grupu čine i slušni testovi. Zbog toga je na početku faze izvršavanja ispitnih slučajeva, napravljeno 17 BBT projekata, tako da jedan projekat sadrži ispitne slučajeve koji pripadaju jednoj grupi. Pored ovih projekata tokom rada javila se potreba za još nekim specifičnim BBT projektima. Na primer projekat koji obuhvata sve testove, koji se koristio prilikom regresivnog ispitivanja, ili projekata koji su objedinili ispitne slučajeve kod kojih izlazna wav datoteka sadrži 2 kanala, 6 kanala, itd.

Nakon izvršavanja, moguće je proveriti ishod pritiskom na dugme *Show/Hide history*. Mogući ishodi izvršavanja BBT testova su *PASS*, *FAIL* i *INCOLCLUSIVE*.

Ukoliko se svi koraci predviđeni testom uspešno izvrše, ishod izvršavanja ispitnog slučaja biće *PASS*, sa druge strane, ukoliko se desi da prilikom izvršavanja nekog od koraka predviđenih testom dođe do greške ishod testa biće *FAIL*. Ukoliko se, pak, desi da ishod izvršavanja testa bude *INCOLCLUSIVE* to znači da je izvršavanje testa prekinuto pre nego što su izvršeni svi predviđeni koraci.

Na sledećoj slici dat je primer pregleda ishoda izvršavanja nekoliko ispitnih slučajeva.

No.	Test Name	Result	Time
1	sys_alg_arle_pkprotect_distortionrel...	PASS	15.08.2016, 10:29:32
2	sys_alg_arle_pkprotect_distortionrel...	PASS	15.08.2016, 10:30:53
3	sys_alg_arle_pkprotect_timbrepese...	PASS	15.08.2016, 10:32:03
4	sys_alg_arle_pkprotect_timbrepese...	PASS	15.08.2016, 10:33:35
5	sys_alg_arle_pkprotect_io7ch1_io48k	PASS	15.08.2016, 10:34:45
6	sys_alg_arle_pkprotect_io2ch_io32k	FAIL	15.08.2016, 10:36:38
7	sys_alg_arle_pkprotect_io2ch_io44k1	FAIL	15.08.2016, 10:37:17
8	sys_alg_arle_pkprotect_io2ch_io48k	PASS	15.08.2016, 10:37:55
9	sys_alg_arle_pkprotect_io5ch1_io32k	FAIL	15.08.2016, 10:38:32
10	sys_alg_arle_pkprotect_io5ch1_io44k1	FAIL	15.08.2016, 10:39:12
11	sys_alg_arle_pkprotect_io7ch1_io32k	INCONCLUSIVE	15.08.2016, 10:39:42
12	sys_alg_arle_pkprotect_timbrepese...	INCONCLUSIVE	15.08.2016, 10:40:00

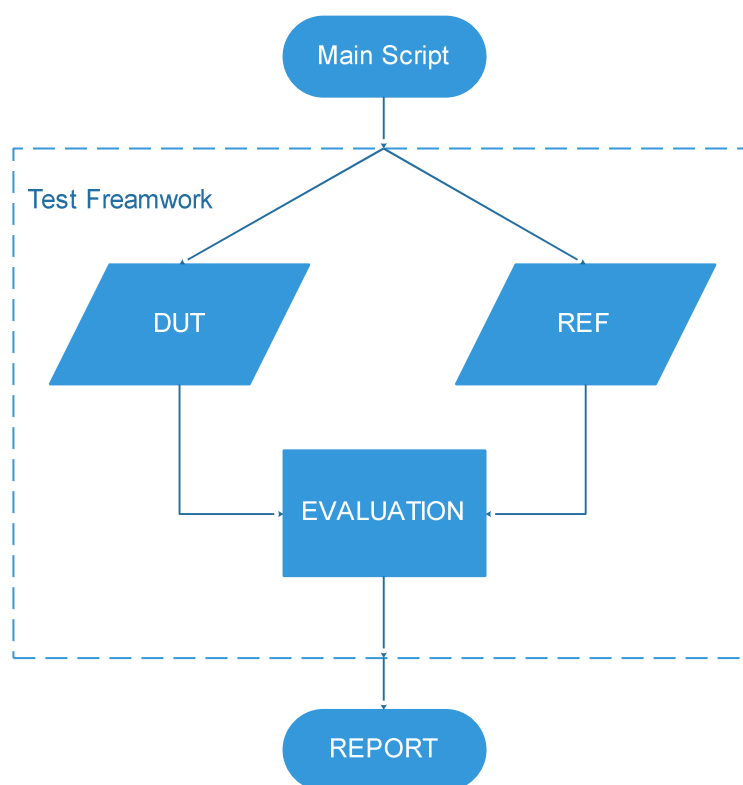
Slika 4.13 - Prikaz ishoda izvršavanja testova

Nakon izvršavanja testa, u testnom folderu se generišu log datoteke. U ovim datotekama sačuvane su poruke koje se dobijaju od ugrađenih BBT alata. Tako se nakon izvršavanja ispitnog slučaja može pratiti ponašanje, na primer, *Cmd_Line* alata kroz *commandline_output.log* datoteku, ili pak ponašanje *Asio* alata kroz *AsioDev.log* datoteku. Ove datoteke su naročito značajne jer se njihovom analizom vrlo brzo mogu detektovati eventualne greške u samim ispitnim slučajevima.

4.4 Aktivnosti nakon ispitivanja

Kao što je pomenuto u konceptu, akcije koje se preduzimaju u ovoj fazi procesa ispitivanja, i kojima se sam proces zaokružuje su: praćenje statusa problema i izveštavanje.

Analizom izveštaja koje generiše BBT alat, nakon izvršavanja ispitnih slučajeva, dobijaju se informacije da li je ciljna platforma konfigurisana na željeni način i da li su izlazne datoteke uspešno snimljene. Međutim, proverava da li se snimljeni izlazi slažu sa referentnim vrši se pomoću testnog okvira. Na sledećoj slici prikazan je blok dijagram koji objašnjava proces poređenja referentnih i snimljenih izlaza.



Slika 4.14 – Blok dijagram poređenja referentnih i izlaza sa ciljne platforme

Glavna skripta testnog okvira se poziva iz komande linije i na osnovu argumenata koji su joj prosleđeni, poziva ostale komponente testnog okvira. Najčešće korišćeni parametri u ovom radu su bili *-RUN_MODE (-r)* i *-FILTER (-f)*.

U zavisnosti od vrednosti parametra *RUN_MODE*, vršiće se generisanje referentnih signala, ili pak evaluacija, odnosno poređenje referentnih signala sa signalima snimljenim na ciljnoj platformi.

Pomoću parametra *FILTER* može se vršiti filtriranje ispitnih slučajeva. Filtriranje ispitnih slučajeva vrši se na bazi ključnih reči. Ključna reč može biti naziv testnog slučaja, naziv podgrupe kojoj testni slučaj pripada, ili neka meta oznaka. Meta oznake su dodatne informacije o ispitnom slučaju koje se mogu koristiti prilikom filtriranja testova. Na primer: meta oznaka za sve testove koji imaju dvo kanalne ulaze i izlaze bila bi *io2ch*, ili meta oznaka *io48kHz* kojom će se filtrirati svi testovi kod kojih je frekvencija odabiranja signala 48 kHz.

Pomoću logičkih operacija i, ili i ne moguće je napraviti složenije kombinacije za filtriranje ispitnih slučajeva.

Svaki put prilikom pozivanja glavne skripte, na unapred definisanoj lokaciji, generiše se izveštaj u obliku HTML datoteke. Izveštaj ima formu tabele, a najbitnija polja su:

- Test case – jedinstveni identifikator testnog slučaja.

- Status – označava opšte stanje nakon pokretanja glavne skripte. Mogući statusi su PASSED, FAILED i UNRESOLVED. Ukoliko je status testa FAILED ili UNRESOLVED, potrebno je preduzeti određene korake, i ispitati razlog zbog kog test ne prolazi.
- Output files – Tekstualne datoteke koje se generišu od strane testnog okvira. U njima su navedene opšte informacije o ispitnom slučaju, kao i poruke o greškama.
- Evaluation method – definiše metodu evaluacije koja je upotrebljena u konkretnom ispitnom slučaju.
- Summary – Kratak pregled opšteg stanja nakon pozivanja glavne skripte.

Nakon svakog ciklusa ispitivanja, izveštaji dobijeni od testnog okvira se postavljaju na softver za praćenje stanja projekta, i na taj način dele sa ostatkom tima.

Prilikom izrade ovog rada za praćenje stanja projekta korišćen je Redmine. Redmine je besplatan, softver otvorenog koda (eng. Open Source Software) namenjen za upravljanje projektima i praćenje problema. Ukoliko se nakon izvršavanja ciklusa ispitivanja primete greške, odnosno neslaganja sa zahtevima predviđenim specifikacijom, test inženjer prijavljuje problem timu koji je zadužen za razvoj softvera koji se ispituje. Na osnovu ispitnih slučajeva koji se neuspešno izvrše, test inženjer može zaključiti koja funkcionalnost softvera nije implementirana na očekivan način. Prijavljivanje problema, podrazumeva i otvaranje novog zadatka, odnosno otvaranje Bug-a, za tim koji razvija softver. Test inženjer je dužan da detaljno opiše problem, kod kog ispitnog slučaja se javlja, odnosno koju funkcionalnost pogađa. Što više informacija tim koji razvija softver poseduje, lakše će locirati grešku. Nakon popravljanja greške, vrši se novi ciklus ispitivanja, kako bi se proverilo da je greška zaista otklonjena. Ukoliko se pokaže da nova verzija softvera radi bez grešaka, prethodno otvoreni zadatak (Bug) se zatvara.

Upotrebom softvera za praćenje projekta, komunikacija između tima koji razvija softver i tima koji ispituje, se podiže na viši nivo, sve akcije koje preduzimaju i jedan i drug tim su dokumentovane i mogu se analizirati, što olakšava posao i menadžmentu.

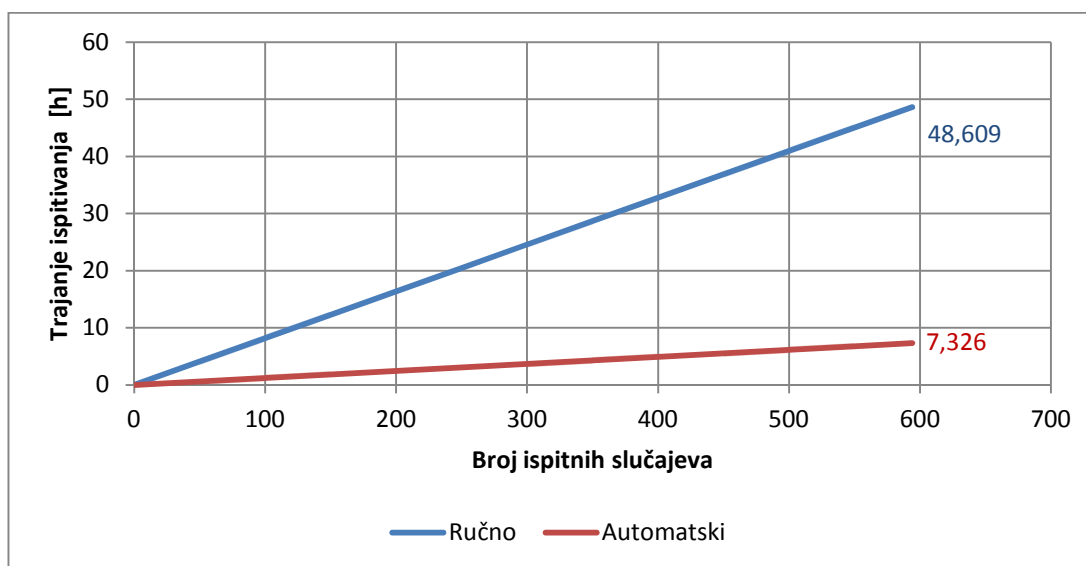
5. Rezultati

U ovom poglavlju izneti su rezultati koji su postignuti u radu.

Kompletan proces ispitivanja je podeljen u dva dela. Kao proizvod prvog dela dobijamo izlazne datoteke snimljene na ciljnoj platformi. Međutim za generisanje tih datoteka potrebno je izvršiti niz aktivnosti. Konfiguracija ciljne platforme, priprema ulazne datoteke, reprodukcija i snimanje izlaza, vremensko poravnanje snimljenih izlaza automatizovani su upotrebom BBT alata. Drugi deo kojim se proces ispitivanja kompletira podrazumeva poređenje referentnih izlaza sa izlazima generisanim na ciljnoj platformi. Postupak poređenja se izvodi pomoću testnog okvira dobijenog od proizvođača tehnologije i taj proces je, takođe, potpuno je automatizovan. Kao proizvod drugog dela dobijamo izveštaj o tome da li je ispitni slučaj uspešno izvršen, odnosno da li je zadovoljio uslove predviđene specifikacijom.

Za potvrdu ispravnosti implementacije tehnologije koja je bila predmet ispitivanja, specifikacijom je predviđeno 594 ispitna slučaja za bit identično poređenje i 10 slušnih testova. Ručno izvršavanje celog skupa ispitnih slučajeva zahtevalo bi previše vremena. Iz tog razloga, ručno je izvršen samo manji broj ispitnih slučajeva, a zatim se izračunala prosečna vrednost trajanja izvršavanja jednog testa.

U nastavku je dat dijagram na kom je prikazana zavisnost vremena trajanja ispitivanja od broja izvršenih ispitnih slučajeva kada se ispitivanje izvršava ručno ili automatski. Za automatsko izvršavanje svih ispitnih slučajeva predviđenih specifikacijom potrebno nešto više od 7 sati, dok bi za ručno izvršavanje bilo potrebno oko 48 sati.



Slika 5.1 – Grafik zavisnosti vremena trajanja ispitivanja od broja izvršenih ispitnih slučajeva kada se ispitivanje izvršava ručno ili automatski

Na osnovu rezultata prikazanih na dijagramu, zaključuje se da je automatizacijom proces ispitivanja ubrzan za oko 6,6 puta. U ovom primeru u obzir nije uzeta činjenica da čovekova koncentracija i efikasnost opadaju tokom vremena, a samim tim se povećava verovatnoća da će pri manuelnom ispitivanju doći do grešaka. Zaključuje se da bi u realnim uslovima ručno izvršavanje kompletnog skupa ispitnih slučajeva trajalo daleko duže. Ova ušteda vremena je naročito značajna ako se uzme u obzir da se ispitivanje ponavlja nakon izmena u kodu aplikacije koja se testira.

Rezultati sa dijagrama mogu se analizirati i sa stanovišta potrošnje radnih dana, pri tom se pretpostavi da radno vreme (radni dan) traje osam sati. Za automatsko izvršavanje skupa ispitnih slučajeva potreban je jedan radni dan. Sa druge strane ručno izvršavanje trajalo bi nešto više od 6 radnih dana. Pošto kod automatskog ispitivanja gotovo da nema potrebe za intervencijom čoveka, testovi se mogu izvršavati i van radnog vremena. Često se ciklus ispitivanja pokretalo pred kraj radnog vremena, kako bi se testovi izvršili tokom noći, a novi rezultati bili dostupni narednog dana. Tako se radno vreme moglo iskoristiti za eventualno popravljjanje samih ispitnih slučajeva, ili alata koji se koriste pri ispitivanju.

6. Zaključak

Tema ovog rada je automatizacija procesa ispitivanja jedne složene audio tehnologije. U okviru rada vršeno je funkcionalno ispitivanje implementirane tehnologije koristeći metodu crne kutije.

Jedan od prvih zadataka, bio je upoznavanje sa razvojnim okruženjem tehnologije koja je bila predmet ispitivanja. Nakon detaljnog proučavanja specifikacije, i kreiranja plana ispitivanja prešlo se na kreiranje ispitnih slučajeva. Generisanje ispitnih slučajeva je automatizovano pomoću nekoliko skripti napisanih u Python skript jeziku. Ispitni slučajevi su kreirani tako da budu zadovoljeni svi uslovi predviđeni specifikacijom. Kako je u svrhe BBT ispitivanja korišćen poseban alat, ispitni slučajevi morali su biti kreirani tako da ispoštuju pravila koja alat RT-Executor donosi.

Pošto se svi BBT testovi uspešno izvrše, na predefinisanoj lokaciji čuvaju se izlazni testni vektori, odnosno wav datoteke snimljene na ciljnoj platformi. Upoređivanju referentnih izlaza sa izlazima snimljenih na ciljnoj platformi vrši se pomoću testnog okvira. Nakon što izvrši proveru za izabrane ispitne slučajeve, testni okvir generiše izveštaj u kom su dati rezultati provere.

U ovom radu izvršena je i automatizacija generisanja ispitnih slučajeva, kao i automatizacija izvršavanja ispitnih slučajeva. Dugački ispitni slučajevi, sa puno različitih ulaznih parametara, kada se izvode manuelno zahtevaju mnogo vremena. Kao i svaki manuelni rad, podložni su greškama koje spadaju u takozvani ljudski faktor. Vreme koje je uloženo u automatizaciju procesa ispitivanja, vraća se kroz vreme potrebno za izvršavanje ispitnih slučajeva. Automatizacijom izvršavanja ispitnih slučajeva, proces ispitivanja ubrzan je šest puta. Na kraju, testni okvir, namenjen za upoređivanje snimljenih i referentnih izlaza, takođe je automatizovan tako da je potreba za intervencijom čoveka u celom procesu ispitivanja svedena na minimum.

7. Literatura

- [1] *Multichannel sound technology in home and broadcasting applications*, International Telecommunication Union, Report ITU-R BS.2159-4 (05/2012)
- [2] Guillaume Potard, *3D-audio object oriented coding*, University of Wollongong, 2006.
- [3] S. Füg, *Design, Coding and Processing of Metadata for Object-Based Interactive Audio*, AES 137th Convention, Los Angeles, Clifornia, October 9-12, 2014
- [4] G.Myers, *Art of software testing*, ISBN: 0471043281, 1979
- [5] DeMillo, R.A. and R.J. Lipton, *A Probabilistic Remark on Algebraic Program Testing*. Defense Technical Information Center, 1978.
- [6] Burnstein, I., *Practical Software Testing: A Process-Oriented Approach*, Springer-Verlag New York, Inc, 2003.
- [7] Dustin, E., *Automated Software Testing*. Boston: Addison Wesley, 1999.
- [8] Craig, R.D. and S.P. Jaskiel, *Systematic Software Testing*. Artech House Publishers, 2002
- [9] Kolawa, A. and D. Huizinga, *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press, September 2007.
- [10] Ron Patton, *Software Testing*, Que/Sams, November, 2000, ISBN 9780768657258
- [11] Boris Beizer, *Black-Box Testing : Techniques for Functional Testing of Software and Systems*, John Wiley & Sons, Inc. 1995, ISBN 0471120944
- [12] Gerald D. Everett, Raymond McLeod, Jr., *Software Testing Testing Across the Entire Software Development Life Cycle*, by John Wiley & Sons, Inc, 2007.
- [13] Dušica Marijan, Vladimir Zlokolica, Nikola Teslić, Vukota Peković, Tarkan Teckan,

-
- “Automatic Functional TV Set Failure Detection System”, IEEE Transactions on Consumer Electronics, Vol. 56, No. 1, February 2010.
- [14] „Owner’s Manual Version 2.2 for Windows“, Echo Digital Audio Corporation, September, 2009
- [15] „CS498xx_Datasheet“, Cirrus Logic, Inc., March, 2013