



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
У НОВОМ САДУ



Димитрије Гузијан

**Развој софтверске подршке у систему за
препознавање инструмената**

ДИПЛОМСКИ РАД

-Основне академске студије-

Нови Сад, 2026.



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

ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

Кандидат

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ЗАВРШНИ РАД	Датум:

(Податке уноси предметни наставник - ментор)

Студијски програм:	Енергетика, електроника и телекомуникације		
Студент:	Димитрије Гузијан	Број индекса:	ЕЕ 120/2020
Степен и врста студија:	Основне академске студије		
Област:	Електротехника и рачунарство		
Ментор:	др Предраг Теодоровић, ванредни професор		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;			

НАСЛОВ ЗАВРШНОГ РАДА:

Развој софтверске подршке у систему за препознавање инструмената
--

ТЕКСТ ЗАДАТКА:

Развити софтверску подршку система за препознавање музичких инструмената, имплементиран као хардверски акцелератор на FPGA платформи. Описати модел система на системском нивоу, поступак екстракције MFCC коефицијената, имплементацију и верификацију IP језгра. Реализовати платформски карактер драјвер у оперативном систему Linux који преко AXI4-Lite интерфејса управља радом IP језгра и излаже засебне датотеке уређаја за упис тест вектора, управљање и читање резултата. Развити корисничку апликацију за демонстрацију рада система и извршити проверу функционалности на развојној плочи Zybo Z7-10.
--

Руководилац студијског програма:	Ментор рада:
др Милан Сечујски, ванредни професор	др Предраг Теодоровић

Примерак за: ☐ - Студента; ☐ - Ментора
--

Изјава о академској честитости

Студент: **Димитрије Гузијан**

Број индекса: **ЕЕ120-2020**

Студент **основних академских студија**

Аутор рада под називом: **Развој софтверске подршке у систему за препознавање инструмената**

Потписивањем изјављујем:

- да је рад искључиво резултат мог сопственог истраживачког рада;
- да сам рад и мишљења других аутора које сам користио у овом раду назначио или цитирао и наведени у списку литературе/референци који су саставни део овог рада;
- да сам добио све дозволе за коришћење ауторског дела који се у потпуности/целости уносе у предати рад и да сам то јасно навео;
- да сам свестан да је плагијат коришћење туђих радова у било ком облику (као цитата, парафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, веб сајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима, Службени гласник Републике Србије, бр. 104/2009, 99/2011, 119/2012), као и других закона и одговарајућих аката Универзитета у Новом Саду;
- да сам свестан да плагијат укључује и представљање, употребу и дистрибуирање рада предавача или других студената као сопствених;
- да сам свестан последица које код доказаног плагијата могу проузроковати на предати рад и мој статус;
- да је електронска верзија рада идентична штампаном примерку и пристајем на његово објављивање под условима прописаним актима Универзитета.

Нови Сад, 1.9.2026.

Потпис студента



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 РАД		
Аутор, АУ :	Димитрије Гузијан		
Ментор, МН :	др Предраг Теодоровић, ванредни професор		
Наслов рада, НР :	Развој софтверске подршке у систему за препознавање инструмената		
Језик публикације, ЈП :	СРПСКИ		
Језик извода, ЈИ :	СРПСКИ		
Земља публикавања, ЗП :	СРБИЈА		
Уже географско подручје, УГП :	ВОЈВОДИНА		
Година, ГО :	2026.		
Издавач, ИЗ :	АУТОРСКИ РЕПРИНТ		
Место и адреса, МА :	НОВИ САД; ТРГ ДОСИТЕЈА ОБРАДОВИЋА 6а		
Физички опис рада, ФО : (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/41/12/25/15/0/1		
Научна област, НО :	ЕЛЕКТРОТЕХНИКА И РАЧУНАРСТВО		
Научна дисциплина, НД :	Електроника		
Предметна одредница/Кључне речи, ПО :			
УДК			
Чува се, ЧУ :	БИБЛИОТЕКА ФТН, НОВИ САД ТРГ ДОСИТЕЈА ОБРАДОВИЋА 6а		
Важна напомена, ВН :	НЕМА		
Извод, ИЗ :	У завршном раду описано је пројектовање и имплементација IP језгра које убрзава извршавање критичне функције за препознавање музичких инструмената. Описан развој и имплементација драјвера и корисничке апликације, као и резултати добијени тестирањем на Зубо развојној плочи.		
Датум прихватања теме, ДП :	14.7.2026.		
Датум одбране, ДО :	1.9.2026.		
Чланови комисије, КО :	Председник:	др Иван Мезеи, редован професор	
	Члан:	др Вук Врањковић, ванредни професор	Потпис ментора
	Члан, ментор:	др Предраг Теодоровић, ванредни професор	



KEY WORDS DOCUMENTATION

Accession number, ANO :			
Identification number, INO :			
Document type, DT :	MONOGRAPHIC PUBLICATION		
Type of record, TR :	TEXTUAL MATERIALS, PRINTED		
Contents code, CC :	BACHELOR THESIS		
Author, AU :	Dimitrije Guzijan		
Mentor, MN :	Predrag Teodorović, PhD, associate professor		
Title, TI :	Development of Software Support for an Instrument Recognition System		
Language of text, LT :	SERBIAN		
Language of abstract, LA :	SERBIAN		
Country of publication, CP :	SERBIA		
Locality of publication, LP :	VOJVODINA		
Publication year, PY :	2026.		
Publisher, PB :	AUTHOR'S REPRINT		
Publication place, PP :	FTN, NOVI SAD, TRG DOSITEJA OBRADOVIĆA 6a		
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/41/0/25/15/0/1		
Scientific field, SF :	Electrical Engineering and Computer Science		
Scientific discipline, SD :	Electronics		
Subject/Key words, S/KW :			
UC			
Holding data, HD :	LIBRARY OF THE FACULTY OF TECHNICAL SCIENCES, NOVI SAD, TRG DOSITEJA OBRADOVIĆA 6a		
Note, N :	NONE		
Abstract, AB :	The thesis describes the design and implementation of an IP core that accelerates the execution of the critical function for musical instrument recognition. It also describes the development and implementation of driver and application, and shows the final results of testing on Zynq development board		
Accepted by the Scientific Board on, ASB :			
Defended on, DE :			
Defended Board, DB :	President:	Ivan Mezei, PhD, full professor	
	Member:	Vuk Vranjković, PhD, associate professor	Mentor's sign
	Member, Mentor:	Predrag Teodorović, PhD, associate professor	

Садржај

1.	Увод	7
2.	Пројектовање апликације на системском нивоу	8
2.1	Екстракција MFCC коефицијената	8
3.	Имплементација IP језгра система за препознавање инструмената	10
4.	Верификација IP језгра	12
5.	Драјвери и Linux	14
5.1	Увод у оперативне системе	14
5.2	Увод у оперативни систем Linux	16
5.3	Драјвери и уређаји	17
5.4	Управљачки бројеви	18
5.4.1	file_operations структура	19
5.4.2	cdev структура	19
5.5	Механизам рада платформских уређаја и драјвера	20
5.6	Покретање Linux ОС на Zybo развојној плочи	21
5.7	Драјвер система за препознавање инструмената	23
5.7.1	ir_init функција	23
5.7.2	ir_probe функција	24
5.7.3	ir_exit и ir_remove функција	24
5.7.4	tv_write функција	24
5.7.5	ctrl_write и ctrl_read функција	25
5.7.6	result_read функција	25
5.7.7	Протокол комуникације и синхронизација	25
6.	Апликација за тестирање драјвера	26
7.	Закључак	30
8.	Литература	31

1. Увод

У савременим информационим системима, аудио сигнал представља важан извор података који се све чешће анализира и обрађује. Развој дигиталне обраде сигнала и раст доступности рачунарских ресурса омогућили су примену напредних алгоритама за анализу звука, чиме се отвара простор за решавање сложених задатака као што је аутоматско препознавање музичких инструмената. Препознавање инструмената има широку примену у системима за препоруку музике, дигиталној продукцији, као и у едукативним алатима. Овај задатак је изазован због сложености аудио сигнала, присуства шума као и међусобне сличности карактеристика различитих инструмената.

У циљу решавања овог проблема, користе се различите методе обраде сигнала и машинског учења. Један од најчешће коришћених приступа подразумева екстракцију карактеристичних параметара из аудио сигнала, као што су Мел-фреквенцијски цепстрални коефицијенти (MFCC), који представљају ефикасан начин описивања спектралних особина звука.

У оквиру овог рада развијена је софтверска подршка за комуникацију са хардверским модулом за препознавање инструмената, која обухвата имплементацију драјвера у Linux окружењу као и једноставну корисничку апликацију. Посебан фокус стављен је на реализацију механизма за приступ регистрима хардверског акцелератора, управљање током извршавања и обезбеђивање коректне размене података између корисничког и кернел простора.

Рад је организован у више целина. У другом поглављу приказан је модел система за препознавање инструмената. Треће поглавље обухвата имплементацију IP језгра. У четвртом поглављу описана је верификација језгра. Пето поглавље се односи на развој драјвера, док је у шестом поглављу приказана корисничка апликација. Последње поглавље садржи закључак рада.

2.Пројектовање апликације на системском нивоу

Дизајн на електронском системском нивоу или скраћено „ESL“ је методологија електронског дизајна која се фокусира на аспекте вишег нивоа апстракције. ESL се гледа као скуп различитих активности за пројектовање система различитих врста. Ти системи су електронски, али укључују хардвер и софтвер [1].

Модел система за препознавање инструмената је имплементиран у програмском језику SystemC [1] и организован је као виртуелна платформа која се састоји из три модула: Софтвер, IP језгро и блок за међусобно повезивање (енг.*interconnect*).

Комуникација се одвија преко TLM-а (енг.*Transaction Level Modeling*). Блок за међусобно повезивање прима све захтеве од софтвера и усмерава их ка IP језгру. IP језгро чува интерно тренинг податке у осам ПЕ (енг. *Processing Element*) кеш структура, чиме се елеминише потреба за физичким BRAM ресурсима. Ток извршавања почиње тако што софтвер учита 680 тренинг вектора из фајла и распоређује их у 8 ПЕ јединица. Када се учитају тренинг вектори, корисник преко терминала уноси две путање за wav фајлове (тест вектор А и тест вектор Б). Након тога софтвер позива *Python* скрипту која израчунава MFCC коефицијенте (поступак је детаљније описан у поглављу 2.1) и враћа их као низ од 13 вредности по wav фајлу. Оба тест вектора се пакују у 32-битне речи. Горњих 16 бита нам представљају коефицијент узорка Б, а доњих 16 бита нам износи узорак А.

У следећем кораку софтвер проверава регистар „ready“, а затим уписује вредност 1 у „start“ регистар. IP језгро прелази у стање „busy“ и започиње рачунање. Сваки вектор, од оних 680, се одузима прво од тест вектора А па од тест вектора Б, када се израчунају све вредности, гледа се која вредност је најмања тачније који тренинг вектор има најмању дистанцу са тест вектором. Та вредност се узима и преко „labels“ фајла видимо који је то инструмент. На пример ако би имали вредност 50 та вредност упада у опсег инструмента бенцо јер нам тај инструмент представља опсег од 0 до 74.

Офсет	Назив	Приступ	Опис
0x00	Start	W	Кад се упише 1 покреће IP
0x04	Ready	R	1=спреман 0=заузет
0x08	result_index_a	R	Индекс најближег вектора за А
0x0C	result_index_b	R	Индекс најближег вектора за Б

Табела 1 Регистарска мапа

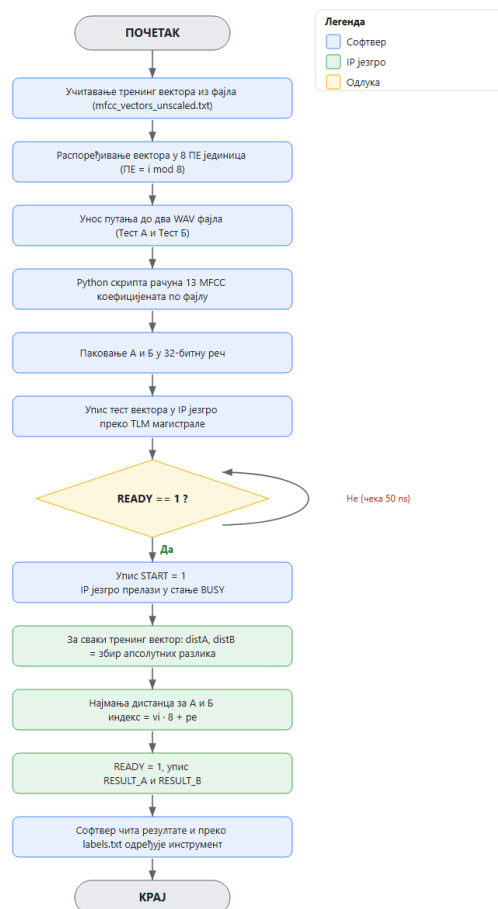
2.1 Екстракција MFCC коефицијената

Улазни података у систему за препознавање инструмената није сирови аудио сигнал, већ скуп обележја које сажето описују спектралне особине звука. Као обележја користе се Мел-фреквенцијски цепстрални коефицијенти.

Поступак издвајања MFCC из једног аудио снимка обухвата следеће кораке. Први корак је учитавање и поново узорковање (енг. *resample*) аудио сигнала на јединствену фреквенцију одабирања. Након тога се извршава уоквиравање (енг. *framing*) сигнала на кратке међусобно преклапајуће временске прозоре и примена прозорске функције, чиме се сигнал дели на квази-стационарне сегменте. Онда следи брза фуријеова трансформација (FFT) сваког оквира и рачунање спектра снаге. Следећи корак је примена банке Мел филтара, којом се енергија спектра пресликава на Мел скалу, затим се извршава логаритмовање енергија по Мел опсезима, чиме се опонаша логаритамска осетљивост слуха на јачину звука. Потом се ради дискретна косинусна трансформација (DCT) логаритмованих енергија, при чему се задржава првих 13 коефицијената. Коефицијенти свих оквира се усредњавају, чиме се сваки снимак своди на један вектор од 13 вредности.

Екстракција је реализована у програмском језику Python, коришћењем библиотеке *librosa*, функцијом *librosa.feature.mfcc* [11] рачуна се 13 коефицијената по оквиру, а потом се функцијом усредњавања по временској оси доија коначни вектор од 13 вредности који представља цео снимак. Пошто хардверски акцелератор ради са целобројном аритметиком, добијене реалне вредности се заокружују на целе бројеве. На исти начин је формиран и скуп тренинг вектора, а они су смештени у фајл који се учитава у меморију система.

Дијаграм тока извршавања система



Слика 1. дијаграм тока извршавања система заснованог на TLM комуникацији

3. Имплементација IP језгра система за препознавање инструмената

RT (енг. *Register Transfer*) методологија представља поступак формализованог пројектовања дигиталног електронског система који имплементира изабрани алгоритам. Овај поступак има неколико корака. Први корак је елиминација наредби понављања (као што су петље *for*, *while* итд...) и дефинисање интерфејса (енг. *interface*). Следећи корак је пројектовање „*controlpath*“ и „*datapath*“ модула, а на крају се пише HDL (енг. *Hardware Description Language*) модел где користимо VHDL или Верилог. Мој пројекат је писан у VHDL-у [2].

Током развијања IP језгра (eng. *Intellectual Property core*) за препознавање музичких инструмената, „*datapath*“ и „*controlpath*“ су описани унутар истог ентитета. Секвенцијална логика развојена је од комбинационе логике, што нам омогућава јасније праћење тока података кроз ASMD дијаграм и једноставније одржавање кода у току итеративног развоја. IP језгро заснива се на алгоритму који гледа сличност MFCC коефицијената улазних вектора са векторима који представљају неке инструменте. Као меру сличности користимо суму апсолутних разлика (*Manhattan distance*).

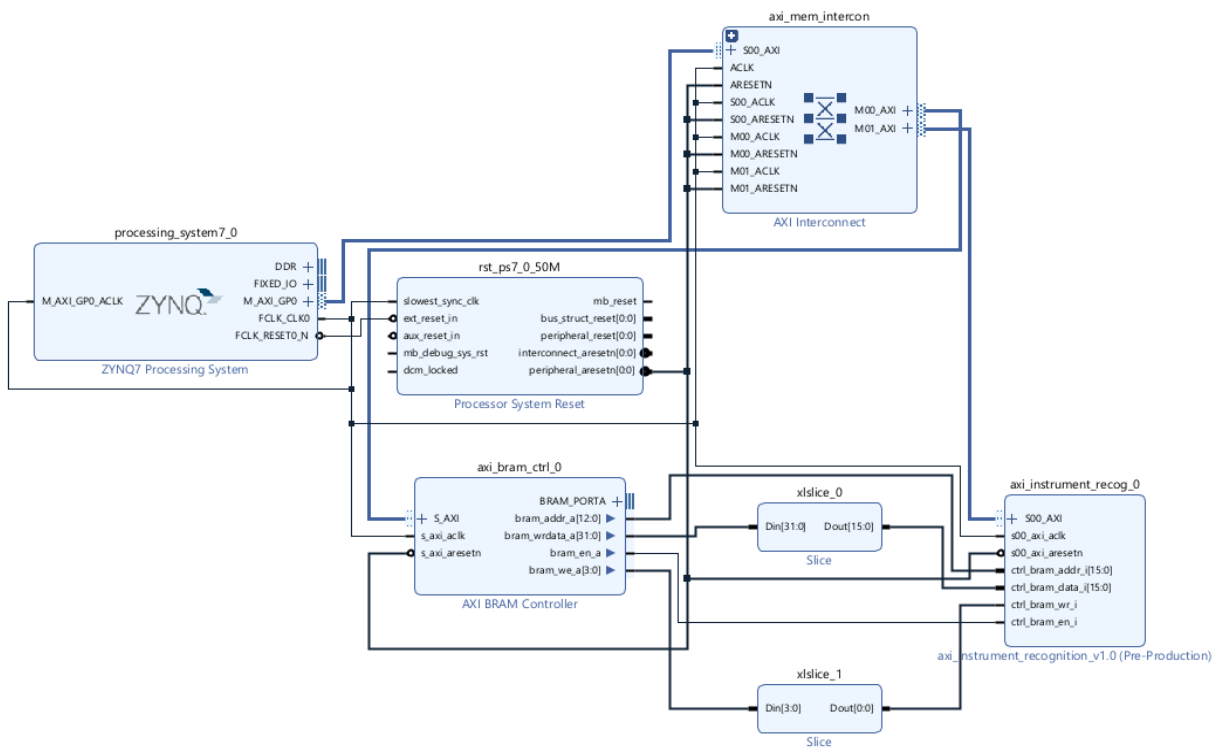
IP се састоји од следећих компоненти:

- AXI4 Lite контролер – обезбеђује комуникацију између процесорског система и хардверског акцелератора преко 18 мапираних регистара. Регистри *slv_reg2* до *slv_reg14* се конкатенирају у један 416-битни улазни вектор. То су два улазна вектора од 13 MFCC коефицијената, где је сваки коефицијент величине 16 бита. Регистар *slv_reg0* користимо као start сигнал, а регистри *slv_reg15*, *slv_reg16* и *slv_reg17* служе за представљање резултата и статусни регистар.

- Меморијски подсистем је реализован од 8 двопрístupних БРАМ блокова где се у сваком БРАМ-у чува 85 тренинг вектора, чиме добијамо равномерну расподелу тренинг вектора на 8 процесних јединица ради паралелног претраживања. Подсистем поседује два независна приступа по меморији: први приступ користи процесор за читање коефицијената тренинг вектора током извршавања алгоритма, а други приступ служи за иницијално пуњење БРАМ блокова преко AXI интерфејса. Адресни декодер на основу три задња бита бира циљни БРАМ, а осталих 11 битова представљају локалну адресу унутар меморије.

- Процесор нам представља хардверску имплементацију алгоритма добијену применом РТ методологије. Имамо коначни аутомат са стањима (конкретно у пројекту су нпр. *getVec*, *calcDiff*, *waitMem* итд...) који управља проласком кроз све тренинг векторе, рачунањем дистанци у осам процесних јединица, проналажењем локалног минимума и проналажењем глобалног минимума. Резултате представљају два 32-битна индекса који одговарају најближим тренинг векторима за тест векторе А и В.

На слици 3 имамо приказ система у Вивадо IP интегратору. У дизајн је убачена компонента *Zynq processing_system7_0* која представља процесор. Поред тога имамо *axi_mem_intercon* која повезује ПС са више периферија (у овом случају БРАМ контролер и IP). БРАМ контролер нам омогућава процесору да директно пише и чита из блок РАМ-а. *Xlslice0* врши конверзију ширине података. Пошто *AXI BRAM Controller* ради са 32-битним речима (*bram_wrdata_a[31:0]*), а IP користи 16-битне улазе (*ctrl_bram_data_i[15:0]*), овај модул издваја доњих 16 бита (LSB). *Xlslice1* издваја *Write Enable* сигнал (*bram_we_a*) како би IP тачно знао када се валидни подаци уписују у меморију.



Слика 3 приказ система у Вивадо ИП интегратору

4. Верификација IP језгра

Верификација хардвера представља процес где се утврђује да ли пројектовани дигитални систем даје исте одзиве као и спецификација. Најраширенија методологија за функционалну верификацију је UVM (енг. *Universal Verification Methodology*).

У пројекту је верификован модул *axi_instrument_recognition_v1_0* – AXI4-Lite периферија. Комплетно верификационо окружење изграђено је у складу са UVM методологијом. Структура пројекта се дели на:

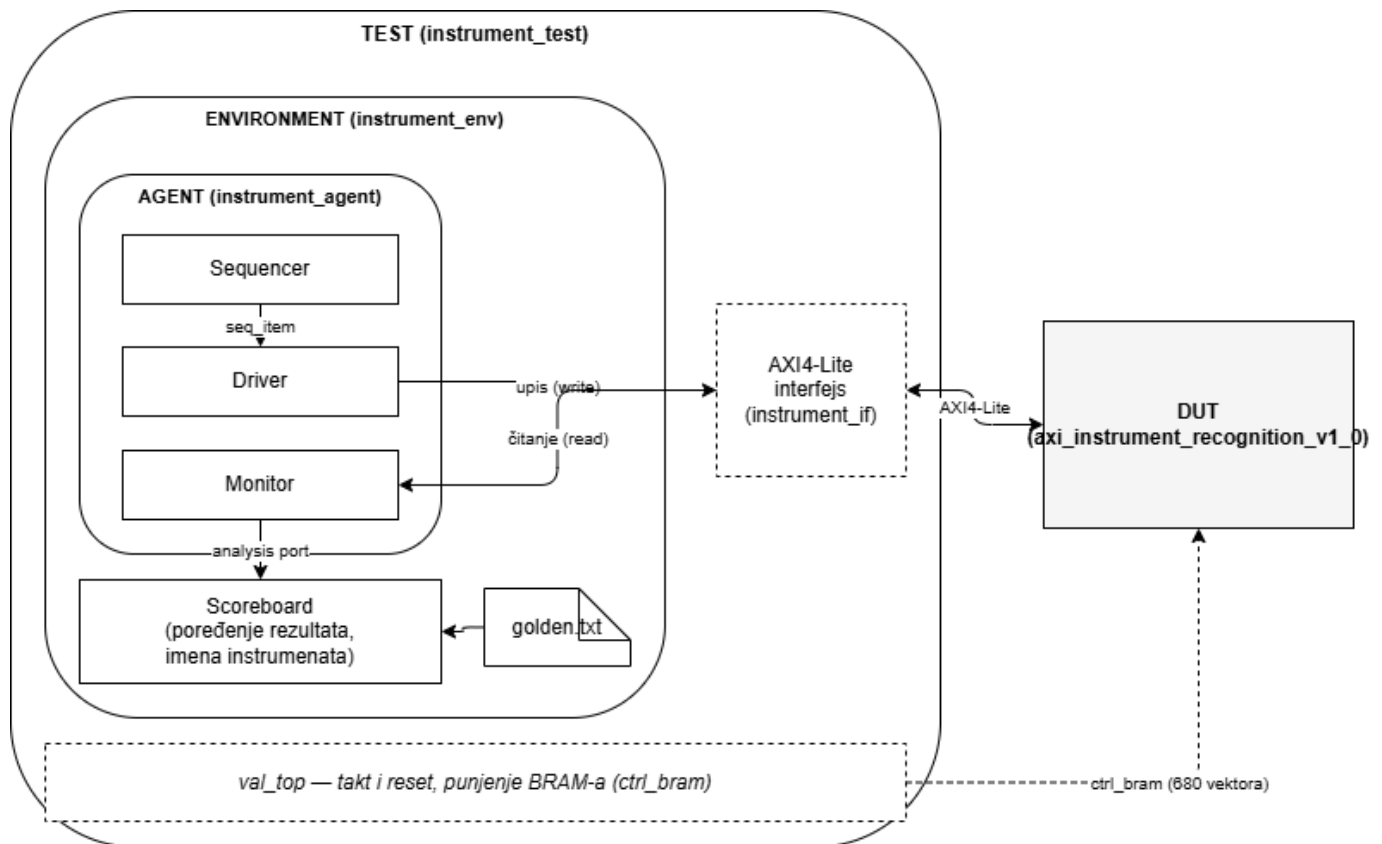
- *Item* садржи низ од 26 MFCC коефицијената, заставицу *is_read*, као и поља *result_a* и *result_b* у која монитор уписује прочитане резултате.
- Секвенца чита тест векторе из фајла са вредностима MFCC и за сваки вектор генерише две трансакције. Једна је слање коефицијената, а друга је читање резултата.
- Драјвер имплементира AXI4-Lite *write* и *read* протокол. Након уписа MFCC коефицијената у регистре, драјвер поставља *start* сигнал, затим испитивањем регистра на адреси 0x44 чека да хардвер пређе у стање обраде, па у стање *done* и тек тада чита резултате.
- Монитор прати AXI магистралу и прати одговоре дизајна, а затим се то шаље у *scoreboard* преко *analysis* порта.
- Агент представља контејнерску UVM компоненту која обједињује драјвер, секвенцер и монитор у јединствену логичку целину. У оквиру *build_phase* фазе, увек се инстанцира монитор, док се драјвер и секвенцер креирају искључиво у случају када је агент конфигурисан у активном режиму рада (*UVM_ACTIVE*), што се проверава методом *get_is_active()*. На овај начин омогућена је флексибилна употреба истог агента и у активном и у пасивном режиму верификације.
- У *connect_phase* фази врши се повезивање компоненти унутар агента, при чему се *seq_item_port* драјвера повезује са *seq_item_export* секвенцера. Тиме се успоставља комуникациони канал који омогућава прослеђивање генерисаних трансакција из секвенцера ка драјверу, где се оне затим преводе у одговарајуће сигнале на интерфејсу
- *Scoreboard* је најважнија компонента провере тачности, ту се добијене вредности упоређују са референтним вредностима учитаним из фајла. Ова компонента броји поклапања и неподударања са референтним вредностима и врши мапирање излазних резултата према унапред дефинисаним опсезима. Расподела инструмената према опсегу је дата у табели 4.1.
- Топ модул правимо да инстанцирамо DUT, генеришемо такт и ресет сигнал, повезује све сигнале на виртуелни интерфејс и пре покретања UVM теста пуни интерни БРАМ са 680 тренинг вектора.

Верификација је извршена над скупом од сто тест вектора који су коришћени за проверу исправности и имплементације алгорита. Добијени резултати су упоређени са очекиваним вредностима, при чему је утврђено потпуно поклапање, што потврђује функционалну исправност пројектованог система.

Верификационо окружење засновано је на методологији UVM и приказано је на слици 4.1. За разлику од система код којих је потребно више агената, овде окружење садржи један активни агент који комуницира са пројектованим IP језгром преко *AXI4-Lite* интерфејса, и *scoreboard*. Прекидни агент није потребан јер језгро не поседује прекидну линију, а тренинг подаци се у меморију језгра уписују директно из *testbench*-а

Назив инструмента	Опсег индекса
Бенџо	0-74
Бас-кларинет	75-177
Чело	178-281
Дабл-бас	282-372
Гитара	373-478
Саксофон	479-572
Виолина	573-680

Табела 4.1 пресликавање опсега индекса у назив инструмента

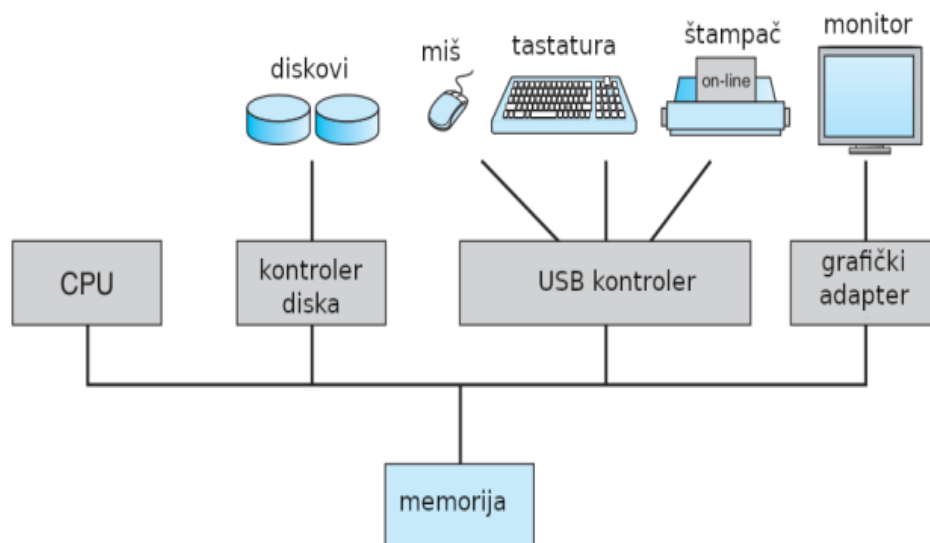


Слика 4.1 Верификационо окружење

5. Драјвери и Linux

5.1 Увод у оперативне системе

Рачунарски систем се може поделити на хардвер, оперативни систем, апликативне и системске програмске подршке и кориснике. Оперативни систем је слој софтвера који управља ресурсима рачунара за своје кориснике и њихове апликације. Оперативни системи раде у широком распону рачунарских система [3].



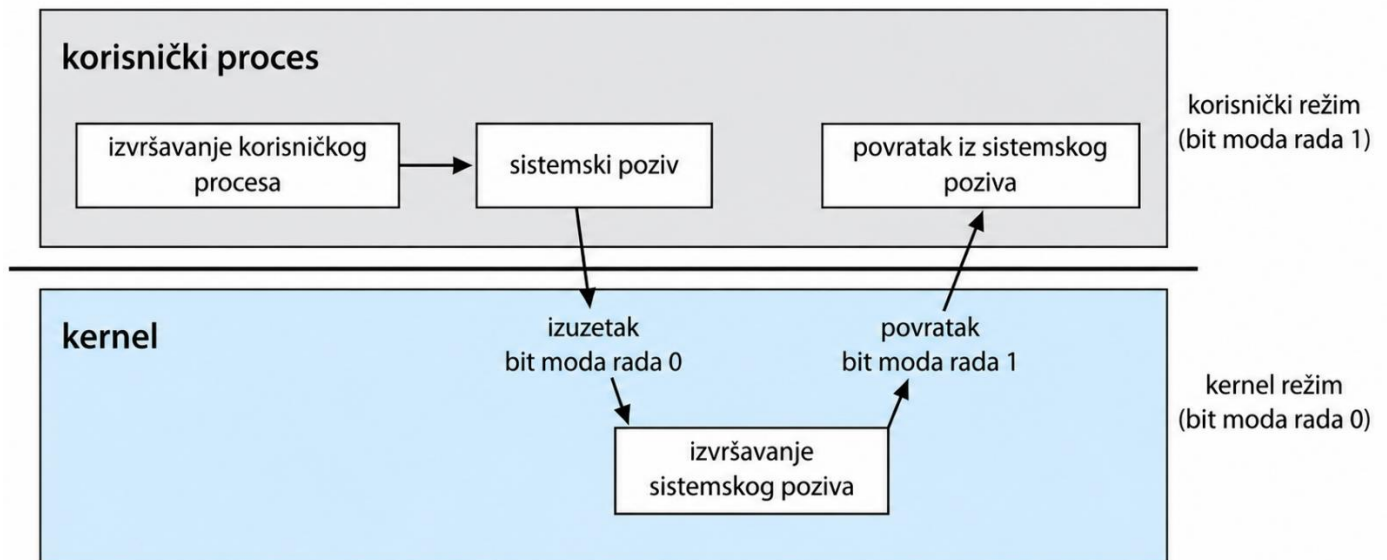
Слика 5 – структура рачунарског система [3]

Основне улоге оперативног система су:

- Улога судије – управља заједничким ресурсима између различитих апликација које раде на истој физичкој машини.
- Улога илузионисте – Пружа апстракцију физичког хардвера како би поједноставили дизајн апликација.
- Улога лепка – пружа скуп различитих сервиса који се деле између апликација. Заједнички сервиси постоје како би нам олакшали комуникацију и дељење података између апликација.

Савремени оперативни системи су базирани на прекидима. Пошто корисници и оперативни систем деле ресурсе, неопходна је заштита како грешка у једном програму не би угрозила друге процесе или сам систем. Као решење овог проблема имамо два одвојена режима рада, то су кориснички и кернел режим. Разликовање се постиже хардверским битом мода рада (0 је кернел, а 1 је кориснички). При покретању система хардвер стартује у кернел режиму. При сваком прекиду хардвер аутоматски прелази у кернел режим. Пре предаје контроле корисничком програму, систем поставља бит на 1.

Привилеговане функције су функције које могу нашкодити оперативном систему на било који начин. Ове функције се извршавају само у кернел режиму и то нам хардвер гарантује. Покушај да се користе у корисничком режиму нам ствара изузетак [3].



Слика 5.1 прелазак између корисничког и кернел мода [3]

5.2 Увод у оперативни систем **Linux**

Linux оперативни систем је настао од стране младог студента Linus Torvaldsa на хелсиншком универзитету у Финској. Почео је са радом 1991. године када је настала верзија 0.02, а 1994. године излази верзија 1.0 *Linux* кернела. Кернел је развијен и објављен под GNU General Public Licence лиценцом. То значи да је изворни код кернела доступан свима.

Осим тога што је *Linux* бесплатан оперативни систем, његова функционалност, адаптивност и робусност су га учинили алтернативним оперативним системом *Unix*-у и свим *Microsoft*-овим оперативним системима. *Linux* има могућност имплементације на микрочиповима што омогућава настајање такозваних „ембедед“ система [3].

Linux кернел представља централни део оперативног система који управља хардверским ресурсима и омогућава комуникацију између софтвера и физичких компоненти рачунара. Он се понаша као посредник између корисничких апликација и хардвера, обезбеђујући апстракцију хардвера и контролу над ресурсима. Кернел обезбеђује више ствари, а то су управљање процесима, управљање меморијом и управљање улазно излазним операцијама.

Захваљујући својој модуларности и стабилности, *Linux* кернел се користи у широком спектру уређаја почевши од персоналних рачунара и сервера до мобилних телефона и ембедед система. Значајан је веома у ембедед системима због своје могућности оптимизације за специфичан хардвер. Кернел се може проширити динамички док је систем подигнут, додавањем модула, ово представља једну од најважнијих особина *Linux* оперативног система. Постоје различити типови модула међу којима су и драјвери, модули се додају у кернел помоћу *insmod* програма, при чијем позиву кернел тражи макро *module_init*, док се модули уклањају преко *rmmod* програма где кернел тражи макро *module_exit* [3].

5.3 Драјвери и уређаји

У оперативном систему *Linux* драјвери омогућавају комуникацију између софтвера и хардверских компоненти. Драјвери се често понашају као „црне кутије“, јер у потпуности скривају детаље имплементације хардвера. Кориснички програми и остали делови система приступају путем стандардних системских позива, док је задатак драјвера да те опште захтеве преведе у конкретне операције над физичким уређајем [4].

Једна од основних карактеристика *Linux* драјвера је њихова модуларност. Драјвери се могу развијати независно од остатка система и динамички укључивати у рад када је то потребно. Важан концепт у дизајну драјвера јесте одвајање механизма и политике. Драјвер обезбеђује механизам, тачније основну функционалност за рад са хардвером, док се начин коришћења те функционалности препушта вишим нивоима система или апликацијама. Ово нам омогућава већу свестраност и прилагодљивост система различитим потребама корисника. Драјвер се треба дизајнирати тако да максимално искористи могућности хардвера, али без наметања ограничења у начину коришћења [4].

У зависности од начина комуникације и типа података које обрађују, хардверски уређаји у *Linux* систему се класификују у више категорија. Постоје три класе уређаја: секвенцијални (карактерни) уређаји, блоковски и мрежни уређаји.

Карактер уређаји су уређаји којима приступамо као низу бајтова, слично као раду са фајловима. Управљање овим уређајима се врши путем карактер драјвера, који имплементирају основне системске позиве као што су *open*, *close*, *read* и *write*. Карактер уређајима се приступа преко фајл система, најчешће кроз */dev* директоријум (на пример, */dev/tty1*).

Блок уређаји су уређаји којима се такође може приступати преко */dev* директоријума. То су уређаји на којима се може налазити фајл систем, као што су хард дискови или SSD-ови. За разлику од карактерних уређаја, блок уређаји организују податке у блокове фиксне величине и омогућавају ефикаснији приступ подацима.

Мрежни уређаји представљају интерфејсе који омогућавају трансфер података између различитих рачунара у мрежи. Они могу бити хардверски или софтверски и за разлику од блок и карактер уређаја нису директно мапирани у фајл систем, већ се идентификују путем јединствених имена [4].

5.4 Управљачки бројеви

Секвенцијалним уређајима се приступа преко имена у фајл систему. То називамо специјалне датотеке или датотеке уређаја. Они су смештени у `/dev` директоријуму. Уколико се испише садржај `/dev` директоријума користећи `ls -l` команду добијамо резултате налик овима:

```
brw-rw---- 1 root disk 8, 0    Nov 13 08:19 sda
brw-rw---- 1 root disk 8, 1    Nov 13 08:19 sda1
brw-rw---- 1 root disk 8, 2    Nov 13 08:19 sda2
crw-rw---- 1 root disk 21, 0   Nov 13 08:19 sg0
crw----- 1 root root 10, 231 Nov 13 08:19 snapshot
crw-rw-rw- 1 root tty 5, 0    Nov 13 11:30 tty
crw--w---- 1 root tty 4, 10   Nov 13 08:19 tty10
crw--w---- 1 root tty 4, 11   Nov 13 08:19 tty11
crw--w---- 1 root tty 4, 12   Nov 13 08:19 tty12
crw--w---- 1 root tty 4, 13   Nov 13 08:19 tty13
```

Слика 5.4 садржај `/dev` директоријума

Знамо да прво слово представља тип драјвера, ако је у питању слово „с” то је ознака за *char*, а слово „b” је ознака за блок драјвере. Два броја која су одвојена зарезом називамо управљачки бројеви уређаја. Први број се зове главни (енг. *major*), а други се зове споредни (енг. *minor*). Сваки драјвер има свој уникатни главни број, а у случају да више уређаја користи исти драјвер, онда ће сваки од уређаја имати различити споредни број, тако да се може приступити сваком од њих. *Dev_t* тип је специјалан тип који се користи за представљање уређаја како главних тако и споредних. Он је дефинисан у `<linux/types.h>` и он је 32-битни, 12 бита служе главне бројеве, а 20 бита се користи за споредне. У случају да имамо главни и споредни број неког уређаја, које треба претворити у *dev_t* тип, користи се макро `MKDEV(int major, int minor)`.

Након дефинисања управљачких бројева, потребно је да се обезбеди алокација у оквиру кернела. Ту имамо два начина доделе, статички и динамички. Код статичког алоцирања се подразумева да програмер унапред задаје главни и споредни број уређаја. То се реализује функцијом `register_chrdev_region` и са њом се резервише опсег управљачких бројева. Предност је једноставност, али недостатак је што захтева познавање слободних бројева у систему. Динамичко алоцирање управљачких бројева је флексибилније и више се користи од статичког.

Код динамичког алоцирања управљачких бројева кернел сам додели слободне управљачке бројеве помоћу функције `alloc_chrdev_region`, чиме се избегава могућност конфликта. Добијени бројеви се враћају кроз променљиву типа *dev_t*. При динамичкој алокацији, креирање *device* фајла у `/dev` директоријуму се обично врши аутоматски, коришћењем функција `class_create` и `device_create`. Ове функције омогућавају да се уређај региструје у систему без ручног позива `mknod` команде. При уклањању модула, ресурси се ослобађају функцијама `device_destroy` и `class_destroy`.

5.4.1 file_operations структура

Комуникација са драјверима за секвенцијалне уређаје извршава се као са текстуалним фајловима, потребно је дефинисати функционалности које се требају извршити када се позове нека од функција за рад са фајловима. Томе служи структура *file_operations* чија су поља показивачи на функције

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (int, struct file *, int);
    ...
};
```

5.4.2 cdev структура

Последњи корак у дефинисању драјвера је регистровање *c_dev* структуре. Њена декларација се налази у `<linux/cdev.h>`. Потребно је прво алоцирати меморију за *c_dev* структуру, а затим поставити њена поља *ops* и *owner*. Поље *ops* се поставља на претходну дефинисану структуру *file_operations*.

```
struct cdev *my_cdev = cdev_alloc();
my_cdev->ops = &my_fops;
my_cdev->owner = THIS_MODULE;
```

Уклањање *cdev* структуре се врши помоћу следеће функције:

```
void cdev_del(struct cdev *dev);
```

5.5 Механизам рада платформских уређаја и драјвера

За разлику од уређаја повезаних на магистрале који подржавају аутоматску детекцију, као што је *USB* или *PCI*, постоје уређаји који ту функционалност немају. Те уређаје називамо платформски уређаји и типично су присутни код система заснованих на *ARM* архитектури и ту се користе магистрале попут *AXI*.

Као што сам већ рекао овде не постоји механизам за аутоматско детектовање, тако да оперативни систем мора на други начин да добије информације о хардверу. Ранији приступ подразумевао је дефинисање уређаја у изворном коду кернела, кроз такозване „*board-file*“. Овај приступ није флексибилан, зато што свака измена хардвера изискује измену и понову компилацију кернела. Данашњи приступ заснива се на употреби стабла уређаја (енг. *Device Tree*), које представља структуру података независну од кернела. Приликом покретања система, *bootloader* учитава бинарну верзију стабла уређаја у меморију, чиме кернел добија приступ потребним информацијама без потребе за изменом сопственог кода.

Стабло уређаја се користи у више облика, као:

- изворни текстуални фајл (.dts)
- бинарни фајл (.dtb)
- део виртуелног фајл система доступног преко путање */proc/device-tree*

Кернел током рада користи ове податке за повезивање уређаја са одговарајућим драјверима.

Платформски драјвери користе посебну структуру под именом *platform_driver*, која представља основу за њихову имплементацију. Ова структура се састоји од структуре *device_driver*. и различитих показивача на функције које можемо дефинисати и користити. Структура *device_driver* има три поља: *name*, *owner*, *of_device_id*. *Name* је стринг који представља име драјвера, *owner* представља власника драјвера, скоро увек се користи макро *THIS_MODULE*. Низ структура *of_device_id* имају поље *compatible* где се наводи назив свих уређаја које покрива неки драјвер.

Најзначајније функције су *probe* и *remove* и оне као једини параметар примају структуру *platform_device* која у себи садржи поље *resource*. Функција *probe* се позива када кернел пронађе уређај који одговара драјверу и служи за иницијализацију и заузимање ресурса, док се функција *remove* позива приликом уклањања и служи за ослобађање ресурса.

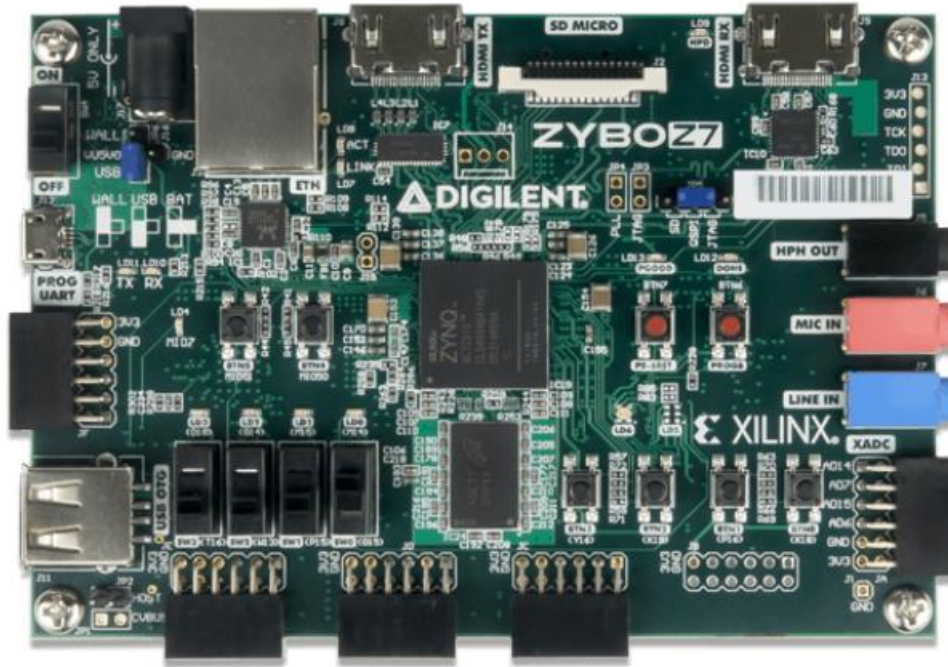
Када би се направила структура *platform_device*, она би се проследила кернелу помоћу позива функције *int platform_device_register (struct platform_device *pdev)*.

Повезивање драјвера и уређаја врши се на основу информација из стабла уређаја, где се упоређују вредности поља *compatible*. Када дође до проналаска одређеног уређаја кернел одмах позива функцију *probe*. У *_exit* функцији је неопходно позвати *platform_device_unregister* након чега кернел позива *remove* функцију драјвера.

Овакав приступ нам омогућава већу флексибилност система, јер се промене у хардверу могу реализовати без потребе за изменама и поновном компилацијом кернела.

5.6 Покретање Linux ОС на Zybo развојној плочи

Да би могли да развијемо и тестирамо драјвер за дизајнирани IP потребно је имати плочу за развијање драјвера. У овом пројекту користимо *Zybo Z7: Zynq-7010 ARM/FPGA SoC* и ова плоча је приказана на слици испод.



Слика 5.6.1 Zybo Z7 [7]

Да би *Linux* систем био функционалан на *Zybo* плочи, *PetaLinux* мора да генерише четири кључне компоненте: иницијални програм који покреће систем (*bootloader*), кернел, стабло уређаја и скуп фајлова и директоријума неопходних за рад корисничког окружења.

Процес покретања система почиње експортовањем хардверског описа из *Vivado* пројекта у облику *.hdf* или *.xsa* фајла, уз обавезно укључивање *bitstream* фајла за конфигурацију *FPGA* при сваком паљењу. *PetaLinux* пројекат се креира коришћењем *zynq* шаблона, а затим се врши иницијална конфигурација на основу хардверског описа наредбом `petalinux-config --get-hw-description`. Да би се систем покретао са *SD* картице, неопходно је у менију поставити *root filesystem type* на *EXT* и *image storage media* на *primary sd* [8].

Специфичне исправке хардверских параметара, попут дефинисања *compatible* поља за упаривање уређаја са драјверима, уносе се искључиво у фајл *system-user.dtsi*. На овај начин се осигурава да ручне измене не буду пребрисане приликом аутоматског генерисања стабла уређаја [8].

За покретање је потребна micro SD картица са две партиције:

- *BOOT* (FAT32): на коју се копирају *BOOT.BIN*, *system.dtb* и *image.ub*.
- *ROOT_FS* (ext4): на коју се инсталира *root* фајл-систем.

Иако *PetaLinux* генерише свој систем, за развој драјвера на самој плочи погоднија је *Debian (Linaro)* дистрибуција због подршке за *apt-get* пакет менаџер.

Приступ систему након подизања остварује се преко серијске везе (брзином 115200 *baud*) или путем SSH протокола. Да би се омогућило компајлирање модула директно на плочи, потребно је инсталирати кернел заглавља (*headers*) и основне развојне алате као што су *git-core*, *bison* и *flex*. Функционалност система се коначно тестира учитавањем драјвера за контролу LED диода путем */dev/* интерфејса [8].

5.7 Драјвер система за препознавање инструмената

Драјвер добија слободне управљачке бројеве од оперативног система, а три датотеке уређаја се креирају у оквиру фајл система на основу података из стабла уређаја, при чему свака покрива једну логичку целину комуникације са IP језгром:

- `/dev/ir_tv` – врши се упис тест вектора тачније MFCC коефицијенти инструмената.
- `/dev/ir_ctrl` – покретање IP језгра и провера његовог статуса.
- `/dev/ir_result` – читање резултата класификације.

Раздвајање функционалности на три засебна секвенцијална уређаја нам омогућава јаснију и читљивију апликацију јер свака операција (упис вектора, покретање, читање) има сопствену датотеку уређаја, као и прецизнију контролу права приступа. Комуникација са IP језгром се остварује преко *AXI4-Lite* интерфејса, а сви регистри су меморијски мапирани почевши од базе адресе језгра. Распоред регистара дат је у табели 5.7.1.

Офсет	Назив	Опис
0x00	CTRL	Контролни регистар (0 старт, 1 софт ресет)
0x08–0x1C	TV_B	MFCC коефицијенти вектора Б
0x20	TV_A0B12	Гранични регистар, горњих 16 бита су вектор А доњих 16 бита су вектор Б
0x24–0x38	TV_A	MFCC коефицијент вектора А
0x3C	RESULT_A	Резултат за најближи индекс вектора А
0x40	RESULT_B	Резултат за најближи индекс вектора Б
0x44	STATUS	Статусни регистар (1 је <i>ready</i>)

Табела 5.7.1 Меморијска мапа регистара IP језгра

IP језгро система за препознавање инструмената не поседује прекидну линију ка процесору. Из тог разлога драјвер за детекцију завршетка обраде користи механизам испитивања статусног регистра уместо прекидне рутине.

Подаци који се преносе су веома малени и фиксне су дужине (26 коефицијената, 2 резултата), није коришћено меморијско мапирање, већ стандардни *read/write* системски позиви, што чини овај драјвер једноставнијим.

5.7.1 `ir_init` функција

`Init` функција се позива при учитавању модула у кернел макроом `module_init` и у њој се креира класа уређаја позивом функције `class_create`, а затим се региструје платформски драјвер позивом функције `platform_driver_register`. Након успешне регистрације, кернел у стаблу уређаја проналази уређај чији се назив поклапа са вредношћу у *compatible* пољу и позива `ir_probe` функцију ради иницијализације.

5.7.2 *ir_probe* функција

Функција *ir_probe* се позива аутоматски када кернел повеже драјвер са одговарајућим уређајем из стабла уређаја. У њој се врши комплетна иницијализација.

Алоцира се структура стања уређаја *ir_dev* помоћу *devm_kzalloc*. Коришћење *devm* (енг. *device managed*) варијанте алокације обезбеђује да кернел аутоматски ослободи меморију при уклањању уређаја, чиме се поједностављује руковање ресурсима.

Позивом *platform_get_resource* са макроом *IORESOURCE_MEM* добијају се подаци о физичком адресном опсегу IP језгра, а функција *devm_ioremap_resource* мапира тај опсег у виртуелни адресни простор кернела, како би се могле користити функције *ioread32* и *iowrite32*.

Врши се иницијализација семафора *sema_init* која обезбеђује међусобно искључивање при истовременом приступу уређају из више процеса.

Позивом *alloc_chrdev_region* динамички се резервише опсег од три управљачка броја (један главни и три споредна).

За сваки од три уређаја, у петљи, иницијализује се и додаје *cdev* структура (*cdev_init*, *cdev_add*), а затим се позивом *device_create* креира одговарајући чвор у */dev* директоријуму (*ir_tv*, *ir_ctrl*, *ir_result*)

5.7.3 *ir_exit* и *ir_remove* функција

Функција *ir_exit* позива се при уклањању модула из кернела (*rmmod*) и у њој се позива *platform_driver_unregister*, након чега кернел позива *ir_remove*. У *ir_remove* функцији се ослобађају заузети ресурси за сваки од три уређаја позива се *device_destroy* и *cdev_del*, а на крају се позивом *unregister_chrdev_region* ослобађају управљачки бројеви. Меморија и мапирани адресни опсег ослобађају се аутоматски захваљујући коришћењу *devm* функција. У *ir_exit* се додатно уништава и класа уређаја позивом *class_destroy*.

5.7.4 *tv_write* функција

Упис тест вектора врши се *write* системским позивом над датотеком */dev/ir_tv*. Апликација прослеђује бинарни блок од 26 шеснаестобитних вредности (52 бајта), односно 13 коефицијаната вектора Б и 13 вектора А, чија је дужина најпре проверена, а потом је садржај преузет из корисничког простора функцијом *copy_from_user*. Након заузимања семафора, помоћна функција пакује по два коефицијента у једну 32-битну реч и уписује их у одговарајуће регистре функцијом *iowrite32*, водећи рачуна о граничном регистру (0x20) који дели последњи коефицијент вектора Б и први коефицијент вектора А.

5.7.5 `ctrl_write` и `ctrl_read` функција

Датотека `/dev/ir_ctrl` користи се и за упис и за читање. При `write` позиву са вредношћу различитом од нуле, драјвер покреће обраду. При `read` позиву функција `ctrl_read` чита статусни регистар и враћа кориснику вредност 1 ако је IP језгро завршило обраду (`ready=1`), односно 0 ако је обрада у току.

При покретању драјвер прво на кратко активира бит за софтверски ресет (бит 1 контролног регистра), чиме се цео IP (управљачки аутомат и меморијски подсистем) доводи у познато почетно стање, идентично стању непосредно након укључивања система. Тек након тога активира се `start` бит (бит 0), који остаје активан све док се резултат не прочита.

5.7.6 `result_read` функција

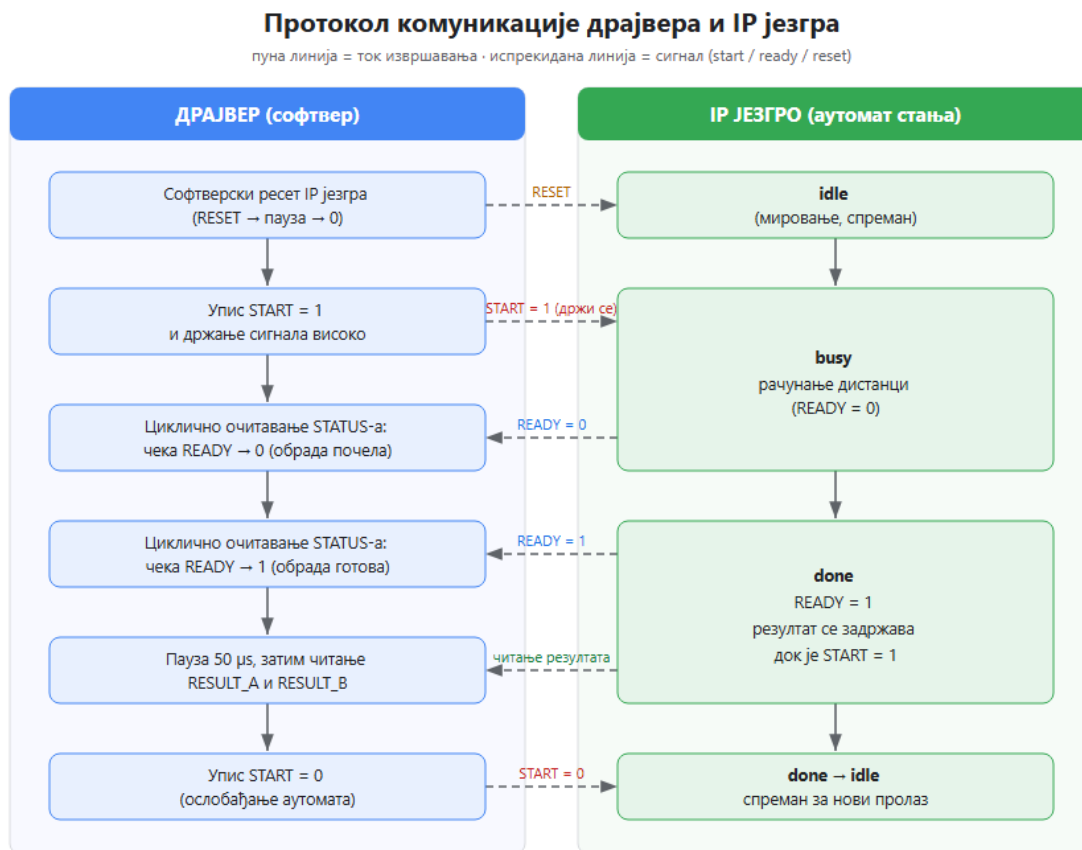
Читање резултата врши се `read` системским позивом над датотеком `/dev/ir_result`, који враћа два 32-битна податка – индекс најближег тренинг вектора за узорак А и тако исто за узорак Б.

Функција најпре испитује статусни регистар док не региструје да је обрада почела (`ready` прелази у 0), а затим док не региструје да је обрада завршена (`ready` се враћа на 1). Тек тада, након кратке паузе како би се стабилизовали излази, читају се регистри резултата и вредности се прослеђују корисничком простору функцијом `copy_to_user`. На крају се старт сигнал спушта на нулу, чиме се аутомат ослобађа и припрема за следећи пролаз.

5.7.7 Протокол комуникације и синхронизација

Машина стања по завршетку обраде улази у стање `done`, у коме поставља `ready` сигнал на 1 и задржава важећи резултат све док је `start` сигнал активан. То значи да је за поуздано читавање потребно да драјвер задржи `start` сигнал активним током целог трајања обраде и све док не прочита резултат, па га тек онда спусти на нулу (чиме аутомат прелази из стања `done` у стање `idle` и постаје спреман за нови пролаз).

Пулсни `start` сигнал (краткотрајно постављање на 1 па враћање одмах на 0) би довело до тога да аутомат напусти стање `done` пре него што драјвер стигне да прочита резултат, што за последицу има задржавање резултата претходне класификације. Како би сваки пролаз почињао из потпуно чистог стања, пре сваког покретања извршава се софтверски ресет IP језгра. Приступ меморији уређаја заштићен је семафором, чиме се спречавају неисправна понашања при истовременом приступу из више процеса.



Слика 5.7.1 Протокол комуникације између драјвера и IP језгра

6. Апликација за тестирање драјвера

Важно је нагласити разлику између кернел модула и корисничке апликације. Модул се извршава у кернел простору и сме да користи искључиво функције дефинисане у кернел простору, а грешка у његовом коду може довести до пада целог система.

Апликација се са друге стране, извршава у корисничком простору, користи стандардне библиотеке и флексибилнија је по питању управљања ресурсима. Апликација је развијена као интерактивни програм, са менијем који кориснику омогућава да унесе произвољан тест вектор, покрене класификацију и прочита резултат, чиме се демонстрира исправан рад целог система.

Комуникација између корисничке апликације и IP језгра реализована је преко кернел драјвера који излаже три датотеке уређаја: `/dev/ir_tv`, `/dev/ir_ctrl` и `/dev/ir_result`. Апликација уписује тест вектор у `/dev/ir_tv`, иницира обраду уписом контролне вредности у `/dev/ir_ctrl`, док се резултати читају из `/dev/ir_result`. Драјвер управља овом комуникацијом и врши приступ меморијски мапираним регистрима IP језгра.

Мени у апликацији нуди следеће опције:

- Приказ тренутног тест вектора
- Унос тест вектора преко тастатуре
- Упис тест вектора и покретање IP језгра
- Провера статуса (да видимо да ли је обрада готова)
- Читање резултата
- Обједињена операција (упис, провера, читање резултата)
- Излаз из апликације

Тест вектор се чува као низ од 26 шеснаестобитних целобројних вредности, при чему првих 13 вредности представља MFCC коефицијенте једног узорка, а преосталих 13 другог узорка. Упис података реализован је коришћењем стандардних системских позива за рад са датотекама.

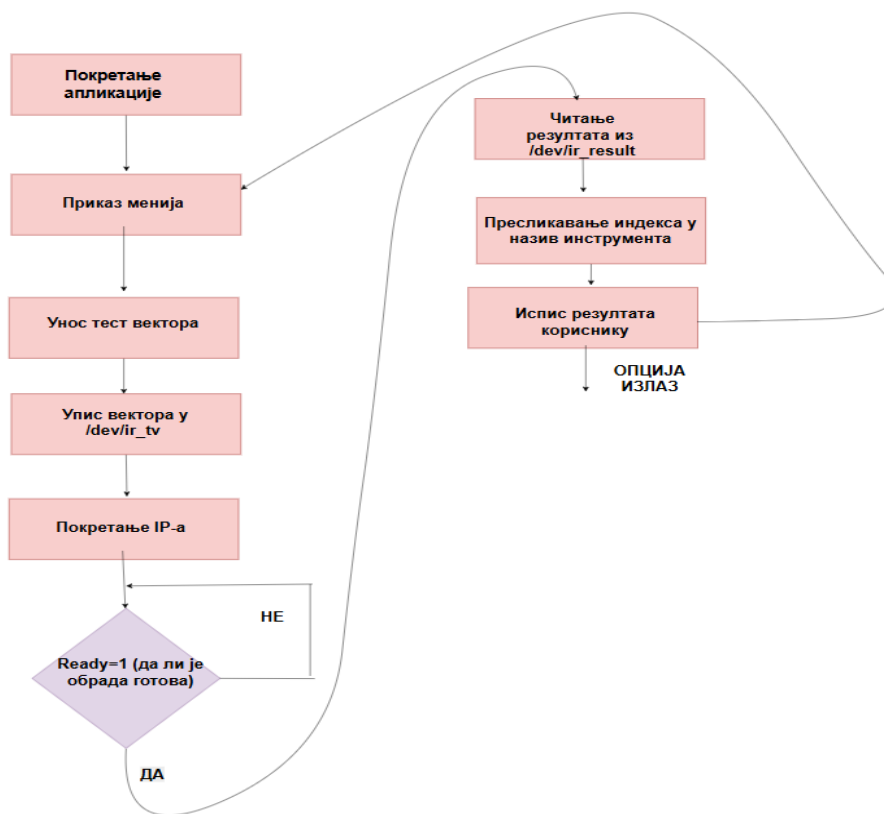
У пракси ћемо најчешће користити опцију обједињених операција, која редом позива операције уписа вектора, покретање обраде и читање резултата. Пошто је чекање на завршетак обраде у потпуности реализовано у самом драјверу, тако што испитујемо статусни регистар, апликација једноставно позове операцију читања над датотеком резултата, која се блокира све док резултат не постане доступан.

Резултат класификације IP језгра се враћа као индекс најближег тренинг вектора. Да би испис био разумљив кориснику, апликација тај индекс пресликава у назив инструмента. Скуп тренинг вектора је организован тако да узастопни опсежи индекса одговарају појединачним инструментима. у складу са истом расподелом коришћеном у верификационом окружењу (табела 4.1).

Апликација за сваки од два узорка исписује број индекса победничког вектора као и име препознатог инструмента. Исправност система потврђена је слањем већег броја тест вектора и поређењем добијених резултата са очекиваним вредностима из верификационог окружења.

За све тестиране улазе на FPGA плочи, резултати су се поклопили са референтим, чиме је потврђено да драјвер и апликација исправно управљају IP језгром и да систем поуздано ради и при узастопном слању више тест вектора.

На слици 6.1 приказан је дијаграм тока рада апликације. Процес почиње уносом тест вектора, након чега следи његов упис у `/dev/ir_tv` и покретање обраде у IP језгру. Након тога се проверава да ли је обрада готова, ако јесте врши се читање резултата из `/dev/ir_result`, а онда се врши пресликавање индекса у назив инструмента и исписује се резултат кориснику. После приказивања резултата добијамо почетни мени.



Слика 6.1- Приказ тока рада апликација

На сликама 6.2, 6.3, 6.4 и 6.5 имамо приказан сам рад апликације.

```
===== Instrument Recognition =====
1 - Prikazi trenutni test vektor
2 - Unesi novi test vektor sa tastature
3 - Upisi test vektor i startuj IP
4 - Proveri status (da li je IP gotov)
5 - Procitaj rezultate
6 - Upisi + startuj + procitaj (sve u jednom)
7 - Izlaz
Izbor: █
```

Слика 6.2 приказ менија у апликацији за препознавање инструмената

```
===== Instrument Recognition =====
1 - Prikazi trenutni test vektor
2 - Unesi novi test vektor sa tastature
3 - Upisi test vektor i startuj IP
4 - Proveri status (da li je IP gotov)
5 - Procitaj rezultate
6 - Upisi + startuj + procitaj (sve u jednom)
7 - Izlaz
Izbor: 1

Trenutni test vektor:
Uzorak B (b[0..12]): -416 129 -28 -2 14 11 -21 -4 18 6 -3 2 8
Uzorak A (a[0..12]): -509 42 -39 -33 -12 -14 -9 13 31 35 6 -21 -22
```

Слика 6.3 приказ опције број 1.

```
===== Instrument Recognition =====
1 - Prikazi trenutni test vektor
2 - Unesi novi test vektor sa tastature
3 - Upisi test vektor i startuj IP
4 - Proveri status (da li je IP gotov)
5 - Procitaj rezultate
6 - Upisi + startuj + procitaj (sve u jednom)
7 - Izlaz
Izbor: 3
Test vektor uspjesno upisan u /dev/ir_tv.
IP startovan (start=1 upisan u /dev/ir_ctrl).

===== Instrument Recognition =====
1 - Prikazi trenutni test vektor
2 - Unesi novi test vektor sa tastature
3 - Upisi test vektor i startuj IP
4 - Proveri status (da li je IP gotov)
5 - Procitaj rezultate
6 - Upisi + startuj + procitaj (sve u jednom)
7 - Izlaz
Izbor: 4
Status: GOTOVO (ready=1)

===== Instrument Recognition =====
1 - Prikazi trenutni test vektor
2 - Unesi novi test vektor sa tastature
3 - Upisi test vektor i startuj IP
4 - Proveri status (da li je IP gotov)
5 - Procitaj rezultate
6 - Upisi + startuj + procitaj (sve u jednom)
7 - Izlaz
Izbor: 5

--- Rezultati klasifikacije ---
Uzorak A: 176 -> Bass-clarinet
Uzorak B: 92 -> Bass-clarinet
-----
```

Слика 6.4 приказ опција 3, 4 и 5

```
Izbor: 2

Unesi 13 MFCC koeficijenata za uzorak B (b[0..12]):
b[0] = -416
b[1] = 129
b[2] = -28
b[3] = -2
b[4] = 14
b[5] = 11
b[6] = -21
b[7] = -4
b[8] = 18
b[9] = 6
b[10] = -3
b[11] = 2
b[12] = 8

Unesi 13 MFCC koeficijenata za uzorak A (a[0..12]):
a[0] = -509
a[1] = 42
a[2] = -39
a[3] = -33
a[4] = -12
a[5] = -14
a[6] = -9
a[7] = 13
a[8] = 31
a[9] = 35
a[10] = 6
a[11] = -21
a[12] = -22

Test vektor unesen.
```

Слика 6.5 приказ опције 2 (унос тест вектора)

7. Закључак

У оквиру овог рада приказан је цео ток развоја IP језгра за препознавање музичких инструмената, од писања модела, дизајнирања IP језгра, његове верификације до развоја драјвера и апликације за његову демонстрацију.

Главни циљ је био писање драјвера система за препознавање инструмената. Драјвер је реализован као платформски карактер драјвер са три засебне датотеке уређаја (за упис тест вектора, управљање и читање резултата), а комуникација са језгром остварена је преко AXI4-Lite интерфејса. Пошто IP језгро не поседује прекидну линију, завршетак обраде се извршава испитивањем статусног регистра.

Систем је тестиран на Zybo Z7-10 плочи слањем више тест вектора и поређењем добијених резултата са референтим вредностима из верификационог окружења. За све тестиране улазе резултати добијени на плочи поклопили су се са очекиваним.

У неким будућим надоградњама пројекта било би занимљиво проширити систем у више праваца. Прекидни механизам би се могао додати уколико би се језгро надоградило прекидном линијом, чиме би се процесор ослободио током обраде дужих или сложенијих улаза. Тренинг скуп би се могао проширити већим бројем инструмената и узорака, чиме би се повећала тачност и општост класификатора. Даље, систем би се могао проширити екстракцијом MFCC коефицијената у реалном времену из аудио улаза, уместо да се коефицијенти задају ручно, чиме би се добила потпуна апликација за препознавање инструмената у реалним условима.

8. Литература

- [1] Предмет – Пројектовање електронских уређаја на системском нивоу проф. Вук Врањковић <http://www.elektronika.ftn.uns.ac.rs/projektovanje-elektronskih-uredjaja-na-sistemskom-nivou/wp-content/uploads/sites/117/2018/03/model.pdf> [посећено 14. мај 2026]
- [2] Предмет пројектовање сложених дигиталних система. проф. Растислав Струхарик <https://www.elektronika.ftn.uns.ac.rs/projektovanje-slozenih-digitalnih-sistema/wp-content/uploads/sites/120/2018/03/Vezba-3-5-RT-Modeling.pdf> [посећено 15. мај 2026]
- [3] Предмет- ембедед оперативни системи проф. Предраг Теодоровић <https://www.elektronika.ftn.uns.ac.rs/mikroracunarski-sistemi-za-rad-u-realnom-vremenu/wp-content/uploads/sites/99/2018/03/OS.pdf> [посећено 23. мај 2026]
- [4] Предраг Теодоровић, Вељко Малбаша, Микрорачунарски системи за рад у реалном времену, [Microsoft Word - Glava 1 - ARM kontroleri](#) [посећено 23. мај 2026]
- [5] Предмет- ембедед оперативни системи проф. Предраг Теодоровић https://www.elektronika.ftn.uns.ac.rs/mikroracunarski-sistemi-za-rad-u-realnom-vremenu/wp-content/uploads/sites/99/2018/03/Vezba5_Drajveri_za_sekvencijalne_ure%C4%91aje-1.pdf [посећено 5. јун 2026]
- [6] Предмет- ембедед оперативни системи проф. Предраг Теодоровић https://www.elektronika.ftn.uns.ac.rs/mikroracunarski-sistemi-za-rad-u-realnom-vremenu/wp-content/uploads/sites/99/2018/03/Vezba9_Drajveri_za_platformske_uredjaje.pdf [посећено 5. јун 2026]
- [7] <https://digilent.com/shop/zybo-z7-zynq-7000-arm-fpga-soc-development-board/> [посећено 5. јун 2026]
- [8] Предмет- ембедед оперативни системи проф. Предраг Теодоровић http://www.elektronika.ftn.uns.ac.rs/mikroracunarski-sistemi-za-rad-u-realnom-vremenu/wp-content/uploads/sites/99/2018/03/Vezba11_Instaliranje_Linux-a_na_Zybo_ploci_na_primeru_BRAM_VGA-1.pdf [посећено 8. јун 2026]
- [9] Third Edition of Linux device drivers, by Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman <https://lwn.net/Kernel/LDD3/> [посећено 20. јун 2026]
- [10] <https://www.kernel.org/doc/html/latest/> [посећено 20. јун 2026]
- [11] <https://librosa.org/doc/latest/generated/librosa.feature.mfcc.html> [посећено 1. јул 2026]
- [12] Muda, Lindasalwa, Mumtaj Begam, and Irraivan Elamvazuthi. "Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques." *arXiv preprint arXiv:1003.4083*(2010). <https://arxiv.org/pdf/1003.4083> [посећено 1. јул 2026]

Додатак – Табеле са описом функција

Табела 1. *Опис MAJOR, MINOR и MKDEV*

<i>Макро</i>	<i>Опис</i>
MAJOR (dev_t dev)	Издаја главни број из параметра dev, tipa dev_t
MINOR (dev_t dev)	Издаја споредни број из параметра dev, tipa dev_t
MKDEV (int major, int minor)	Претвара и главни и споредни број уређаја у dev_t tip

Табела 2. *Опис alloc_chrdev_region функција*

<i>Функција</i>	<i>Опис</i>
<i>alloc_chrdev_region</i>	Креира низ управљачких бројева за секвеницијалне уређаје
<i>Параметри</i>	<i>Опис</i>
dev_t * dev	Функција мења dev у први додељени управљачки број
unsigned baseminor	Први од захтеваних споредних бројева
unsigned count	Број захтеваних споредних бројева
const char * name	Име драјвера

Табела 3. *Опис class_create функције*

<i>Функција</i>	<i>Опис</i>
<i>class_create</i>	Ствара struct class показивач који ће бити коришћен у функцији <i>device_create</i>
<i>Параметри</i>	<i>Опис</i>
struct module* owner	Показивач на модул који ће бити власник класе
const char* name	Показивач на стринг који ће садржати име ове класе

Табела 4. *Onuc device_create* функције

Функција	Опис
<i>device_create</i>	Ствара чвор у <i>/dev</i> повезан са одговарајућим управљачким бројевима
Параметри	Опис
struct class * class	Показивач на повратну вредност из <i>device_create</i> функције
struct device* parent	Показивач на <i>parent</i> структуру уређаја, може се поставити NULL
dev_t dev_t	Управљачки број секвенцијалног уређаја
void* drvdata	Подаци који ће бити додати уређају за накнадне позиве (енг. <i>callback</i>), може се поставити NULL
const char* name	Име драјвера

Табела 5. *Onuc device_destroy* функције

Функција	Опис
<i>device_destroy</i>	Уништава и ослобађа уређај креиран позивом <i>device_create</i>
Параметри	Опис
struct class* class	Показивач на класу насталу позивом функције <i>device_create</i>
dev_t devt	Претходно регистровани управљачки бројеви уређаја

Табела 6. *Onuc class_destroy* функције

Функција	Опис
<i>class_destroy</i>	Уништава и ослобађа уређај креиран позивом <i>class_create</i>
Параметри	Опис
struct class* class	Показивач на класу која ће бити уништена

Табела 7. *Опис unregister_chrdev_region функције*

<i>Функција</i>	<i>Опис</i>
<code>unregister_chrdev_region</code>	Ослобађа заузете управљачке бројеве
<i>Параметри</i>	<i>Опис</i>
<code>dev_t first</code>	Први управљачки број који треба ослободити
<code>unsigned int count</code>	Број управљачких бројева које треба ослободити

Табела 8. *Опис cdev_add функције*

<i>Функција</i>	<i>Опис</i>
<code>cdev_add</code>	Додаје уређај dev у систем
<i>Parametri</i>	<i>Opis</i>
<code>struct cdev *dev</code>	c_dev структура дрјавера којег користимо
<code>dev_t num</code>	Први управљачки број драјвера
<code>unsigned int count</code>	Број споредних бројева које ће драјвер имати

Табела 9. *Опис cdev_del функције*

<i>Функција</i>	<i>Опис</i>
<code>cdev_del</code>	Уклања уређај dev из система
<i>Параметри</i>	<i>Опис</i>
<code>struct cdev *dev</code>	c_dev структура одговарајућег драјвера

Табела 10. *Опис platform_get_resource функције*

Функција	Опис
<i>platform_get_resource</i>	Враћа потребне ресурсе драјверу на основу редног броја
Параметри	Опис
struct platform_device* pdev	Иста структура која је прослеђена <i>probe</i> функцији од стране кернала
unsigned int type	Тип ресурса прослеђује се један од макроя дефинисаних у <i><linux/ioport.h></i> заглављу
unsigned int i	Редни број захтеваног ресурса

Табела 11. *Опис ioread32 функције*

Функција	Опис
<i>ioread32</i>	Чита 32-битну вредност из меморијски мапираног регистра
Параметри	Опис
const void __iomem *addr	Виртуелна адреса регистра са којег чита

Табела 12. *Опис udelay функције*

Функција	Опис
<i>udelay</i>	Активно чекање за одређен број микросекунди
Параметри	Опис
unsigned long usecs	Број микросекунди колико треба чекати

Табела 12. *Опис iowrite32 функције*

<i>Функција</i>	<i>Опис</i>
<i>iowrite32</i>	Уписује 32-битну вредност у меморијски мапиран регистар
<i>Параметри</i>	<i>Опис</i>
u32 value	Вредност која се уписује у регистар
void __iomem *addr	Виртуелна адреса регистра у који се уписује

Табела 13. *Опис sema_init функције*

<i>Функција</i>	<i>Опис</i>
<i>sema_init</i>	Активно чекање за одређен број микросекунди
<i>Параметри</i>	<i>Опис</i>
struct semaphore *sem	Показивач на семафор који се иницијализује
int val	Почетна вредност семафора

Табела 14. *Опис down_interruptible функције*

<i>Функција</i>	<i>Опис</i>
<i>down_interruptible</i>	Заузима семафор, ако је заузет, процес се успављује
<i>Параметри</i>	<i>Опис</i>
struct semaphore *sem	Показивач на семафор који се заузима

Табела 15. *Опис ир функције*

<i>Функција</i>	<i>Опис</i>
<i>ир</i>	Ослобађа претходно заузети семафор
<i>Параметри</i>	<i>Опис</i>
struct semaphore *sem	Показивач на семафор који се ослобађа

Табела 16. *Опис devm_kzalloc функције*

<i>Функција</i>	<i>Опис</i>
<i>devm_kzalloc</i>	Алоцира и нулира меморију везану за уређај
<i>Параметри</i>	<i>Опис</i>
struct device *dev	Уређај за који се везује алоцирана меморија
size_t size	Величина меморије коју треба алоцирати
gfp_t flags	Тип алокације

Табела 17. *Опис devm_ioremap_resource функције*

<i>Функција</i>	<i>Опис</i>
<i>devm_ioremap_resource</i>	Резервише меморијски опсег и мапира физичке адресе у виртуелене
<i>Параметри</i>	<i>Опис</i>
struct device *dev	Уређај за који се везује мапирање
const struct resource *res	Показивач на ресурс добијен функцијом <i>platform_get_resource</i>

Табела 18. *Onuc cdev_init* функције

<i>Функција</i>	<i>Опис</i>
<code>cdev_init</code>	Иницијализује <code>cdev</code> структуру
<i>Параметри</i>	<i>Опис</i>
<code>struct cdev *cdev</code>	Структура која се иницијализује
<code>const struct file_operations *fops</code>	Показивач на структуру са операцијама над фајлом

Табела 19. *Onuc platform_driver_register* функције

<i>Функција</i>	<i>Опис</i>
<code>platform_driver_register</code>	Региструје платформски драјвер у кернел.
<i>Параметри</i>	<i>Опис</i>
<code>struct platform_driver *drv</code>	Показивач на структуру платформског драјвера

Табела 20. *Onuc platform_driver_unregister* функције

<i>Функција</i>	<i>Опис</i>
<code>platform_driver_unregister</code>	Одјављује платформски драјвер из кернела
<i>Параметри</i>	<i>Опис</i>
<code>s struct platform_driver *drv</code>	Показивач на структуру платформског драјвера

Табела 21. *Опис функције copy_to_user*

<i>Функција</i>	<i>Опис</i>
<i>copy_to_user</i>	Копира податке у кориснички простор
<i>Параметри</i>	<i>Опис</i>
void __user* to	Адреса у корисничком простору на коју се копира
const void* from	Адреса у кернел простору са које се копира
unsigned long n	Број бајтова који треба копирати

Табела 22. *Опис функције copy_from_user*

<i>Функција</i>	<i>Опис</i>
<i>copy_from_user</i>	Копира податке из корисничког простора
<i>Параметри</i>	<i>Опис</i>
void* to	Адреса у кернел простору на коју се копира
const void __user* from	Адреса у корисничком простору са које се копира
unsigned long n	Број бајтова који треба копирати