



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Дина Војновић

**Функционална верификација АЕС-128 алгоритма за
енкрипцију**

ДИПЛОМСКИ РАД

- Основне академске студије -

Нови Сад, 2026

	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:	Дина Војновић	Број индекса:	ЕЕ132-2014
Степен и врста студија:	Основне академске студије		
Област:	Функционална верификација хардвера		
Ментор:	др. Предраг Теодоровић		
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:

Функционална верификација АЕС-128 алгоритма за енкрипцију

ТЕКСТ ЗАДАТКА:

Применом <i>UVM</i> методологије, пројектовати верификационо окружење које ће проверити функционалност АЕС алгоритма за енкрипцију.

Руководилац студијског програма:	Ментор рада:

Примерак за: ' - Студента; ' - Ментора
--



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:	Дина Војновић
Ментор, МН:	др. Предраг Теодоровић, ванредни професор
Наслов рада, НР:	Функционална верификација хардверског акцелератора АЕС алгоритма
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:	21000 Нови Сад, Трг Доситеја Обрадовића 6
Физички опис рада, ФО: (поглавља/страна/ цитата/табела/слика/графика/прилога)	5/54/0/1/17/0/0
Научна област, НО:	Електротехника и рачунарство
Научна дисциплина, НД:	Електроника
Предметна одредница/Кључне речи, ПО:	Хардверски акцелератор, <i>FPGA</i> , <i>UVM</i>
УДК	
Чува се, ЧУ:	Библиотека Факултета техничких наука 21000 Нови Сад, Трг Доситеја Обрадовића 6
Важна напомена, ВН:	Нема
Извод, ИЗ:	У раду је детаљно представљен ток развоја верификационог окружења намијењеног за испитивање хардверског <i>IP</i> језгра <i>AEC</i> акцелератора. Развијено верификационо окружење базирано је на <i>UVM</i> методологији.
Датум прихватања теме, ДП:	
Датум одбране, ДО:	29.6.2026
Чланови комисије, КО:	Председник:
	Члан:
	Члан:
	Члан, ментор:
	др. Предраг Теодоровић, Ванредни професор
	Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual material
Contents code, CC :	BSc thesis
Author, AU :	Dina Vojnović
Mentor, MN :	Predrag Teodorović, PhD, Assistant professor
Title, TI :	Functional verification of a hardware accelerator for the <i>AES128</i> algorithm
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 :	Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	5/54/0/1/17/0/0
Scientific field, SF :	Electrical engineering and computer science
Scientific discipline, SD :	Electronics
Subject/Key words, S/KW :	Hardware accelerator, <i>FPGA</i> , <i>UVM</i>
UDC	
Holding data, HD :	Library of the Faculty of technical sciences, Trg Dositeja Obradovića 6, Novi Sad
Note, N :	None
Abstract, AB :	This thesis presents in detail the development process of a verification environment intended for testing the hardware IP core of the AES128 accelerator. Developed verification environment is based on the UVM methodology.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	29.6.2026
Defended Board, DB :	President:
	Member:
	Member:
	Member, Mentor: Predrag Teodorović, PhD, Associate professor
	Menthor's sign

Изјава о академској честитости

Студент : Дина Војновић

Број индекса : ЕЕ132-2014

Студент основних академских студија

Аутор рада под називом : **Функционална верификација АЕС алгоритма за енкрипцију**

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, 20.5.2026.

Потпис студента



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

Слике

Слика 1: Шема шифровања у AES алгоритму	13
Слика 2: S-box lookup табела.....	14
Слика 3: ShiftRows операција.....	15
Слика 4: Дефинисана матрица.....	15
Слика 5: Пример MixColumns операције	16
Слика 6: Пример XOR операције над матрицом стања и матрицом кључа	16
Слика 7: Ротација и SubBytes	17
Слика 8: Процес добијања прве колоне трансформисаног кључа.....	17
Слика 9: Генерисање друге колоне	18
Слика 10: Генерисање треће и четврте колоне	18
Слика 11: Трансформисани кључеви	19
Слика 12: AXI Lite interface.....	19
Слика 13 : Интерфејси AES128-битног алгоритма.....	21
Слика 14: ASMD дијаграм round algoritma	23
Слика 15: архитектура UVM tesbencha	28
Слика 16 : Testbench architecture.....	33
Слика 17: Резултат укупне покривености.....	45

Табеле

Табела 1: сигнали AXI4 Lite интерфејса	20
--	----

Кодни листинзи

Листинг 1: AXI4-Lite agent имплементација.....	34
Листинг 2: Sequencer имплементација	35
Листинг 3: AXI4-Lite driver run_phase имплементација	35
Листинг 4: AXI4-Lite driver do write имплементација	36
Листинг 5: AXI4-Lite driver do read имплементација	36
Листинг 6: AXI4-Lite monitor имплементација count_delay i run_phase taskova	37
Листинг 7: AXI4-Lite monitor имплементација collect_item taska	37
Листинг 8: AES environment имплементација.....	38
Листинг 9: Део write() функције за уписе у регистре	39
Листинг 10 : Део write() функције који скупља вредности регистра из DUTa.....	40
Листинг 11: Део write() функције који обухвата проверу исправности једног дела регистра.....	40
Листинг 12: Део write() функције који обухвата проверу исправности другог дела регистра	40
Листинг 13: Имплементација једног дела aes_reg_value_cg.....	42
Листинг 14: Имплементација другог дела aes_reg_value_cg.....	42
Листинг 15: Имплементација aes_reset_cg.....	43
Листинг 16: Имплементација aes_invalid_start_during_busy_cg	43
Листинг 17: Имплементација једног дела axi_cg.....	44
Листинг 18: Имплементација другог дела axi_cg	44
Листинг 19: Ограничења приликом randomizације aes_random_seq	46
Листинг 20: Имплементација body() методе aes_random_seq	47
Листинг 21: Имплементација body() дела aes_reg_access_seq	48
Листинг 22: Имплементација невалидних уписа унутар body() дела aes_reg_access_seq	48
Листинг 23: Имплементација body() taska aes_reset_seq	49
Листинг 24: Имплементација body() taska aes_reset_seq после ресета.....	50
Листинг 25: Имплементација body() дела aes_invalid_access_seq	51
Листинг 26: Имплементација body() дела aes_invalid_access_seq са невалидним стартом	51

Садржај

1.	Увод	11
2.	AES Rijndael алгоритам	12
2.1	Принцип рада AES 128-битне енкрипције	12
2.2	Замена бајтова (<i>SubBytes</i>).....	13
2.3	Померање редова (<i>ShiftRows</i>)	14
2.4	Мешање колона (<i>MixColumns</i>)	15
2.5	Додавање кључа рунде (<i>AddRoundKey</i>)	16
2.6	Генерисање новог кључа	17
2.7	Хардверска имплементација AES алгоритма	19
2.7.1	Имплементација AXI4-Lite интерфејса	19
2.7.2	Интерфејси имплементираних дизајна алгоритма AES 128-битне енкрипције	21
2.7.3	ASMD дијаграм и datapath round algoritma	22
3.	Функционална верификација хардвера	25
3.1	Увод	25
3.2	Верификациони циклус	25
3.3	UVM – Универзална верификациона методологија	27
3.4	Архитектура и компоненте верификационог окружења	28
3.4.1	UVM Test	29
3.4.2	UVM Environment.....	29
3.4.3	UVM Agent	29
3.4.4	UVM Scoreboard	29
3.4.5	UVM Sequencer	30
3.4.6	UVM Driver	30
3.4.7	UVM Monitor.....	30
3.4.8	UVM Sequence	31
3.4.9	UVM Sequence item	31
3.5	UVM фазе	31
3.6	Архитектура верификационог окружења IP AES-128	33
3.6.1	AXI-4 Lite UVC.....	34
3.6.2	AES Environment	38
3.6.3	AES Scoreboard	38
3.6.4	AES Coverage model.....	41
3.6.5	Тестови и секвенце.....	45
3.7	Пронађени проблеми дизајна и њихово решење	52
4.	Закључак	53
5.	Литература.....	54

1. Увод

Криптографија је наука која се бави методама очувања тајности информација и података. Развој информационих технологија и све већа размена и дељење података, који не би требало да буду доступни свима је довео до све веће потребе за сигурним преносом и складиштењем података. Са порастом количине информација које се размењују путем Интернета јављају се бројне претње које могу да угрозе приватност и безбедност корисника. Из тог разлога, криптографија има веома важну улогу у заштити података.

AES алгоритам (енг. *Advanced Encryption Standard*) представља алгоритам са енкрипцију података који се користи у великом броју савремених система. Усвојен је 2001. године од стране Националног института за стандарде и технологију (NIST) као наследник DES алгоритма, чији ниво сигурности више није био довољан за савремене потребе. Због своје поузданости, брзине и отпорности на криптографске нападе, AES је постао један од најчешће коришћених алгоритама за заштиту података.

Алгоритам ради над блоковима података величине 128,192 или 256 бита, као и кључа који је такође величине 128,192 или 256 бита (на овом пројекту је дизајниран IP који ради са величином података као и ширином кључа од 128 бита). Процес енкрипције реализује се кроз више узастопних рунди које извршавају различите трансформације над подацима. Основне операције су замена бајтова, померање редова, мешање колона и додавање кључа рунде. Комбинацијом ових операција се обезбеђује висок степен сигурности и заштита од различитих врста напада. Поред сигурности AES алгоритам карактеришу и добре перформансе и због тога је погодан за имплементацију у хардверским и ембедед системима.

Функционална верификација се бави провером исправности рада имплементираног дизајна. Она је веома важан корак у току развоја дигиталног система. У процесу верификације потребно је да се утврди да ли систем испуњава све захтеве описане у спецификацији. Најчешће се користи UVM (енг. *Universal Verification Methodology*) методологија, која омогућава систематичан приступ верификовања дигиталних система.

Овај дипломски рад се бави функционалном верификацијом AES алгоритма применом UVM методологије. У наредним поглављима ће бити описан принцип рада AES алгоритма, архитектура система као и процес развоја верификационог окружења. Акценат ће бити стављен на проверу исправности рада алгоритма и анализу добијених резултата.

2. AES Rijndael алгоритам

AES, познат и као Rijndael изабран је као веома робустан алгоритам од стране агенције NIST 2001[1]. године након трогодишњег периода тестирања унутар криптографске заједнице. Међу 15 кандидата који су се такмичили за најбољи алгоритам изабрано је 5 финалиста: MARS (IBM), RC6 (RSA Laboratories), Rijndael (Džon Dejmen и Vincent Rijmen), Serpent (Ros Anderson и други) и TwoFish (Brus Šnajer и други). Сви кандидати су били изузетно добри али је Rijndael однео сигурну победу. AES алгоритам припада групи симетричних шифарских блокова, што значи да користи исти кључ и за енкрипцију и дешифрирање података. Такође, AES је блоковска шифра, што значи да податке не обрађује бит по бит већ их дели у блокове фиксне величине од 128,192 или 256 бита и над сваким блоком извршава процес енкрипције.

2.1 Принцип рада AES 128-битне енкрипције

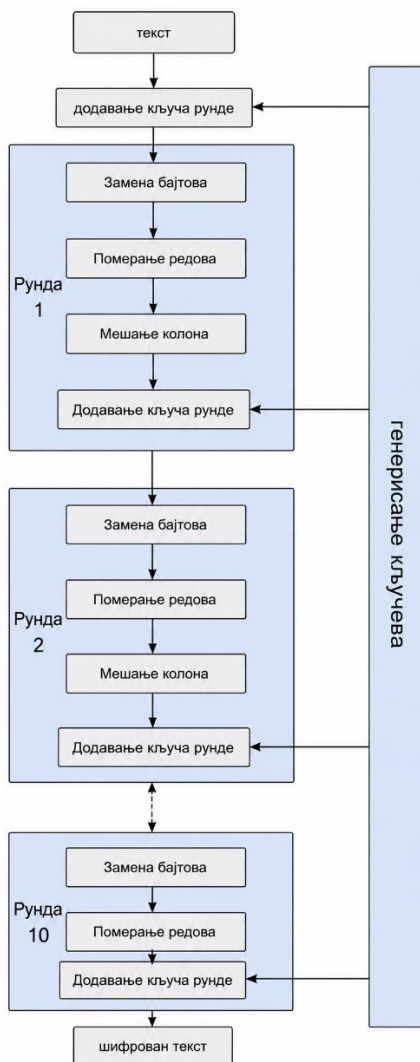
Криптовање података помоћу AES алгоритма извршава се кроз више узастопних итерација. Улазни подаци се прво деле на 128-битне блокове, при чему се сваки блок представља као дводимензионална матрица са 4 реда и 4 колоне. Сваки елемент те матрице представља један бајт податка. Оваква структура се назива матрица стања (*state matrix*). Све операције AES алгоритма се изводе над матрицом стања и биће детаљније описане у наставку рада.

Структура AES алгоритма заснива се на извршавању више узастопних рунди над блоком података фиксне величине[2]. Основни кораци алгоритма могу се представити на следећи начин:

1. Формирање матрице стања – улазни подаци се организују у матрицу стања величине 4x4 над којом се извршавају све операције AES алгоритма
2. Почетно додавање кључа рунде (AddRoundKey) – на почетку процеса енкрипције улазни подаци комбинују се са почетним кључем
3. Извршавање главних рунди алгоритма – AES алгоритам извршава више рунди чији број зависи од дужине кључа, односно у овом случају извршава се 10 рунди.
4. Операције унутар сваке рунде :
 - Замена бајтова (SubBytes),
 - Померање редова (ShiftRows),
 - Мешање колона (MixColumns),
 - Додавање кључа рунде (AddRoundKey)
5. Завршна рунда алгоритма – последња рунда се извршава без операције мешања колона

6. Проширење кључа (Key Expansion) – из почетног кључа се генеришу посебни кључеви рунди који се користе током процеса енкрипције.

На Слици 1 приказана је шема шифровања у AES-128



Слика 1: Шема шифровања у AES алгоритму

2.2 Замена бајтова (*SubBytes*)

SubBytes је операција супституције која се користи у AES алгоритму енкрипције. Ова операција се извршава над сваким бајтом матрице стања тако што се сваки елемент мења са одговарајућом вредности из помоћне *lookup* табеле познате као *s-box*. *S-box* табела је унапред дефинисана за AES енкрипцију и служи за повећање сигурности алгоритма.

Поступак замене врши се тако што се сваки бајт прво представи у хексадецималном облику. Прва цифра представља ред, док друга цифра представља колону у *s-box* табели. На пресеку одговарајућег реда и колоне налази се нова вредност која мења оригинални бајт.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Слика 2: S-box lookup табела

На пример , за бајт A4, цифра A одређује ред, а цифра 4 одређује колону. Вредност пронађена у табели је 49. Дакле, бајт A4 после супституције прелази у 49. Ова трансформација се примењује над свим бајтовима матрице стања током процеса AES енкрипције.

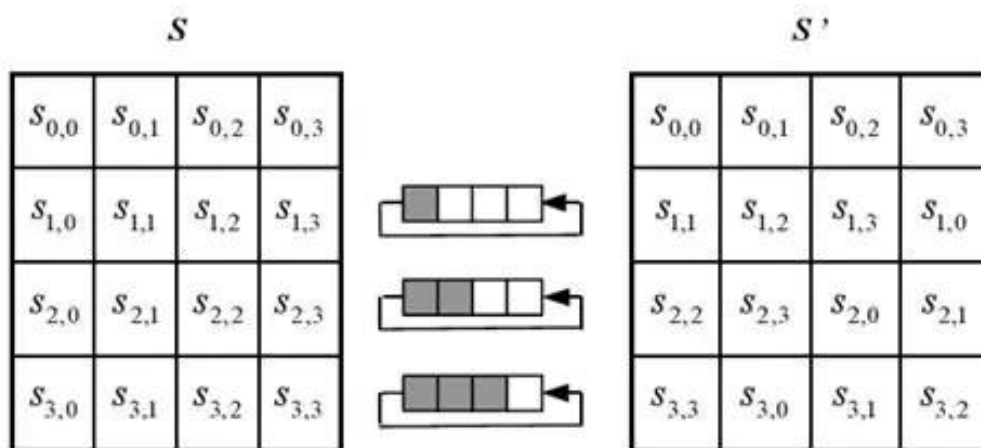
2.3 Померање редова (ShiftRows)

ShiftRows је операција у AES алгоритму служи за кружно померање елемената (бајтова) унутар редова матрице стања. Циљ ове операције је додатно мешање података како би се повећала сигурност енкрипције.

Током операције померања редова:

- први ред остаје непромењен,
- други ред се помера улево за једно место,
- трећи ред се помера улево за два места,
- четврти ред се помера улево за три места

Померање се врши кружно, што значи да елементи који изађу са леве стране поново улазе са десне стране реда.



Слика 3: ShiftRows операција

2.4 Мешање колона (*MixColumns*)

MixColumns је операција у AES алгоритму која служи за додатно мешање података унутар матрице стања. Ова операција се раснива на множењу матрица у оквиру Galoisovog поља $GF(2^8)$.

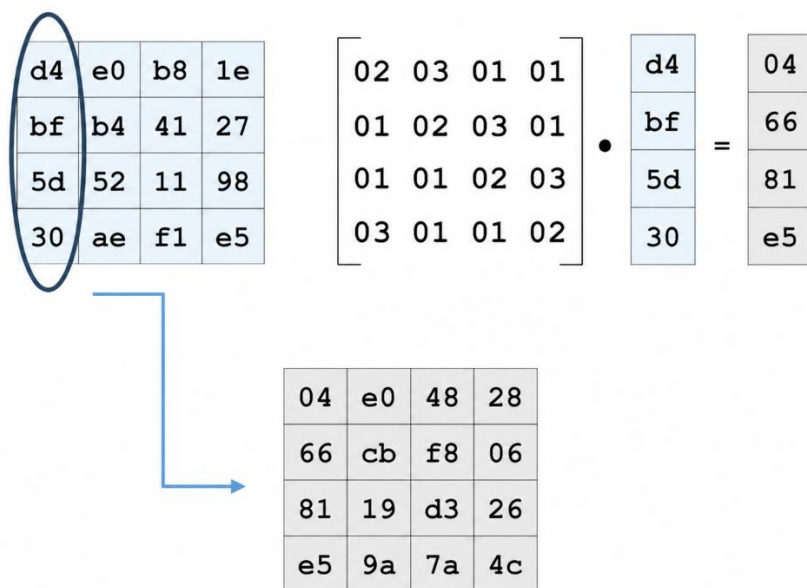
Током *MixColumns* операције свака колона матрице стања посматра се засебно и множи се унапред дефинисаном матрицом:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Слика 4: Дефинисана матрица

Резултат множења представља нову колону која мења оригиналну колону у матрици стања. Исти поступак се примењује над свим колонама матрице, чиме се вредности додатно комбинују и распоређују кроз целу матрицу.

Пример је приказан на слици 5:



Слика 5: Пример MixColumns операције

2.5 Додавање кључа рунде (*AddRoundKey*)

AddRoundKey је операција у AES алгоритму која се заснива на примени XOR операције између матрице стања и матрице кључа. Ова фаза представља повезивање података са кључем и један је од најважнијих корака AES енкрипције.

На почетку процеса енкрипције формира се матрица кључа димензије 4x4 на основу задатог AES кључа. Затим се сваки елемент матрице стања комбинује са одговарајућим елементом матрице кључа коришћењем XOR операције.

На Слици 6 се налази пример XOR операције.

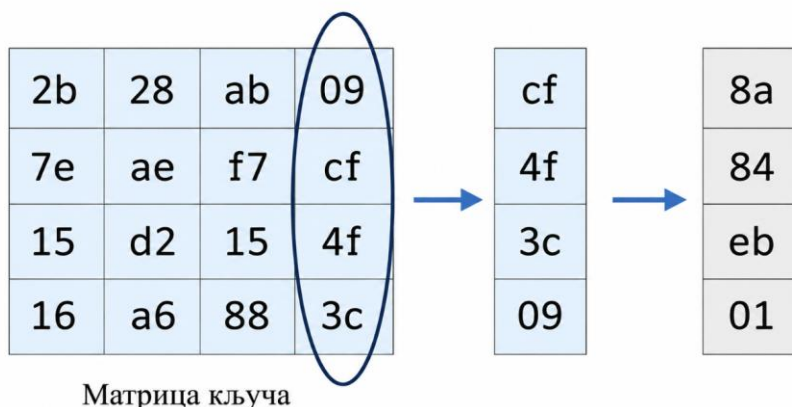


Слика 6: Пример XOR операције над матрицом стања и матрицом кључа

2.6 Генерисање новог кључа

У AES алгоритму свака рунда енкрипције користи посебан кључ. Због тога је потребно да се изврши трансформација почетног кључа како би се нови кључеви генерисали за наредне рунде. Процес генерисања новог кључа састоји се из више корака.

1. Ротација и *SubBytes* операција – Најпре се узима четврта колона матрице кључа и врши се кружно померање елемената за једно место нагоре. Након тога се над добијеном колоном примењује *SubBytes* операција, при чему се сваки бајт мења одговарајућом вредношћу из *S-box* табеле.

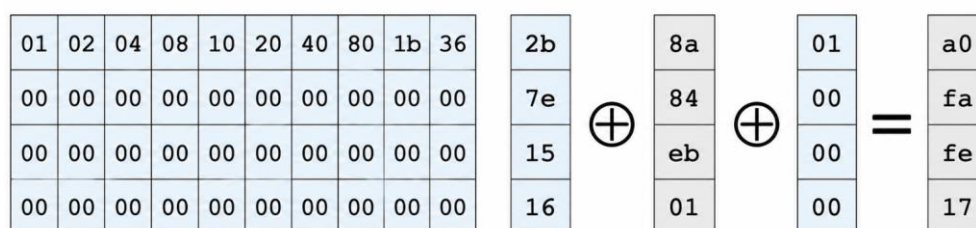


Слика 7: Ротација и *SubBytes*

2. XOR операција са првом колоном и *Rcon* табелом – Затим се врши XOR операција између:

- прве колоне почетног кључа,
- колоне добијене у претходном кораку,
- и одговарајуће колоне из *Rcon* табеле

Rcon табела представља унапред дефинисану табелу константи које се користе током генерисања кључева за рунде. Свака колона *Rcon* табеле одговара једној рунди генерисања кључа. Код AES-128 алгоритма генерише се 10 рундних кључева, па табела садржи 10 различитих *Rcon* константи. Приликом генерисања прве колоне новог рундног кључа користи се одговарајућа *Rcon* константа за текућу рунду (у првој рунди користи се прва колона, у другој друга колона, итд). На тај начин се добија се трансформисани кључ.



Rcon таблица

Слика 8: Процес добијања прве колоне трансформисаног кључа

3. Генерисање друге колоне - Друга колона новог кључа добија се XOR операцијом између друге колоне почетног кључа и прве колоне трансформисаног кључа.

28
ae
d2
a6

 $\oplus$

a0
fa
fe
17

 $=$

88
54
2c
b1

Слика 9: Генерисање друге колоне

4. Генерисање треће и четврте колоне – Трећа и четврта колоне се добијају на исти начин:

- трећа колоне настаје XOR операцијом треће колоне почетног кључа и друге колоне трансформисаног кључа,
- четврта колоне настаје XOR операцијом четврте колоне почетног кључа и треће колоне трансформисаног кључа

2b	28	ab	09		a0	88	23	2a
7e	ae	f7	cf		fa	54	a3	6c
15	d2	15	4f		fe	2c	39	76
16	a6	88	3c		17	b1	39	05

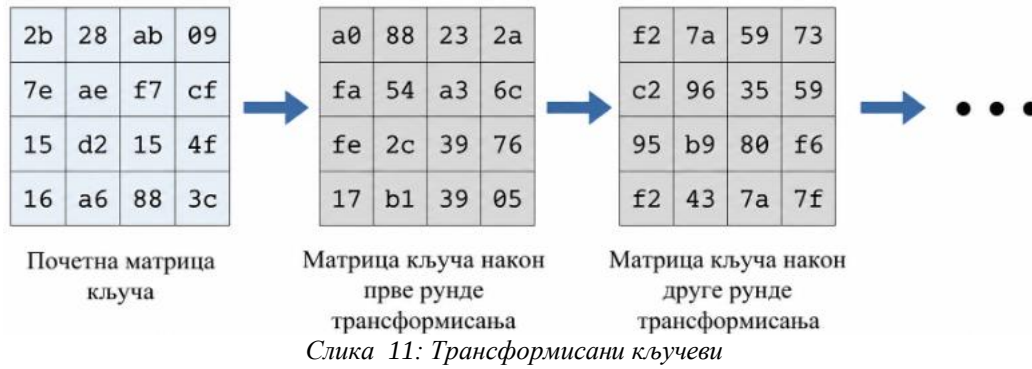
матрица кључа

трансформисани кључ

Слика 10: Генерисање треће и четврте колоне

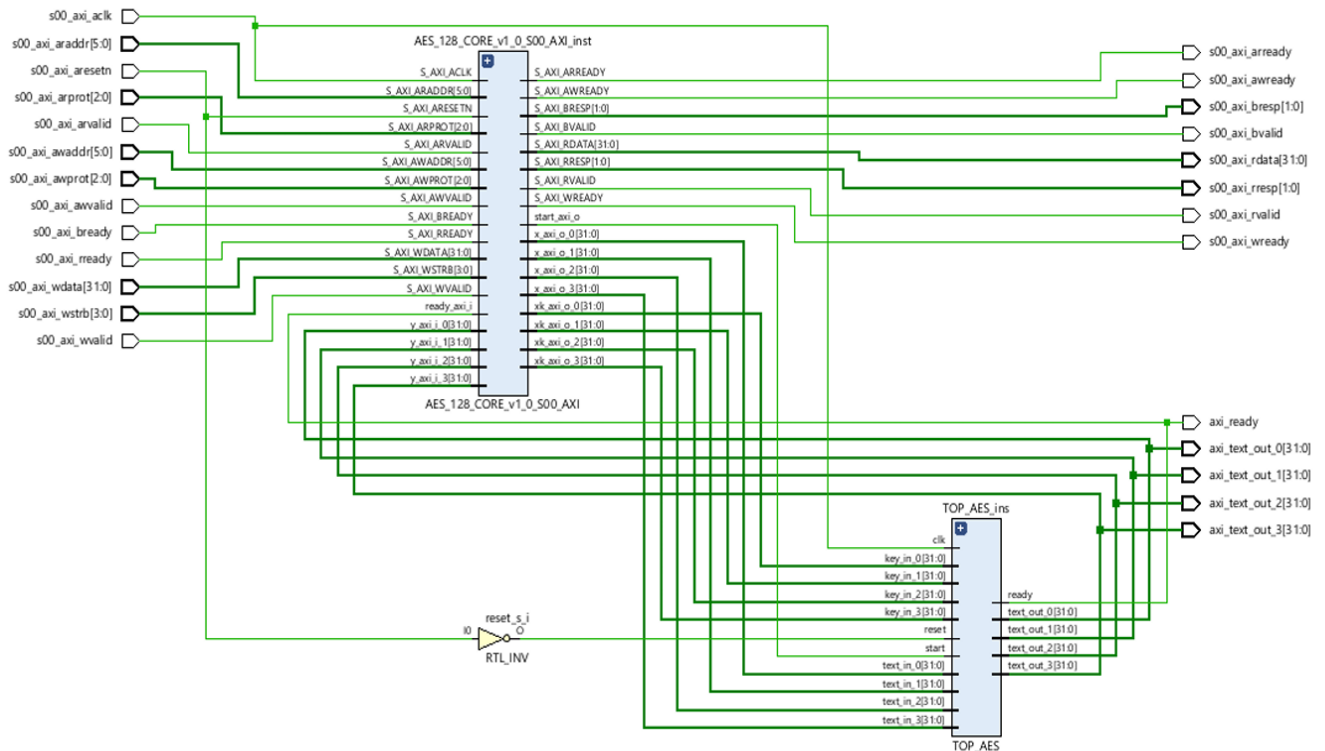
Након свих корака добија се нови трансформисани кључ. Трансформисани кључ постаје почетни кључ за наредну рунду генерисања. Исти поступак се понавља више пута како би се добили сви потребни кључеви за AES енкрипцију.

Овај процес омогућава да свака рунда користи другачији кључ, чиме се додатно повећава сигурност алгоритма и отежава покушај дешифровања података без познавања оригиналног кључа.



2.7 Хардверска имплементација AES алгоритма

2.7.1 Имплементација AXI4-Lite интерфејса



Имплементирани дизајн представља *AES IP* повезан на систем путем *AXI4-Lite* интерфејса. Основна улога интерфејса је омогућавање комуникације између *master* компоненте и *AES IP* језгра кроз приступ интерним регистрима, контролу рада модула и размену података неопходних за процес енкрипције.

У оквиру дизајна реализован је *memory-mapped* приступ регистрима. *Master* компонента уписује *plaintext* и *key* вредности у одговарајуће регистре, након чега активира *START* регистар којим се покреће процес енкрипције. По завршетку обраде *AES* модул поставља *READY* сигнал, и у излазном *ciphertext* регистру се налази резултујући шифрат.

За комуникацију је коришћен AMBA AXI4-Lite протокол због једноставне структуре, мале хардверске сложености и погодности за управљање регистрима IP језгара. AXI4-Lite не подржава *burst* трансакције, већ се сваки приступ реализује као појединачна *read* или *write* трансакција. Одређени сигнали унутар самог протокола нису коришћени.

Интерфејс дизајна подељен је на више логичких канала:

- канал за упис адресе
- канал за упис података
- канал за *write response*
- канал за читање адресе
- канал за читање података

Комуникација између *master* и *slave* компоненти реализована је употребом *VALID/READY handshake* механизма. Пренос података се сматра успешним у тренутку када су *VALID* и *READY* сигнали истовремено активни.

У оквиру *write* трансакције *master* поставља адресу регистра и податке који се уписују. Након успешно реализованог уписа *slave* компонента враћа *write response* сигнал којим потврђује статус трансакције.

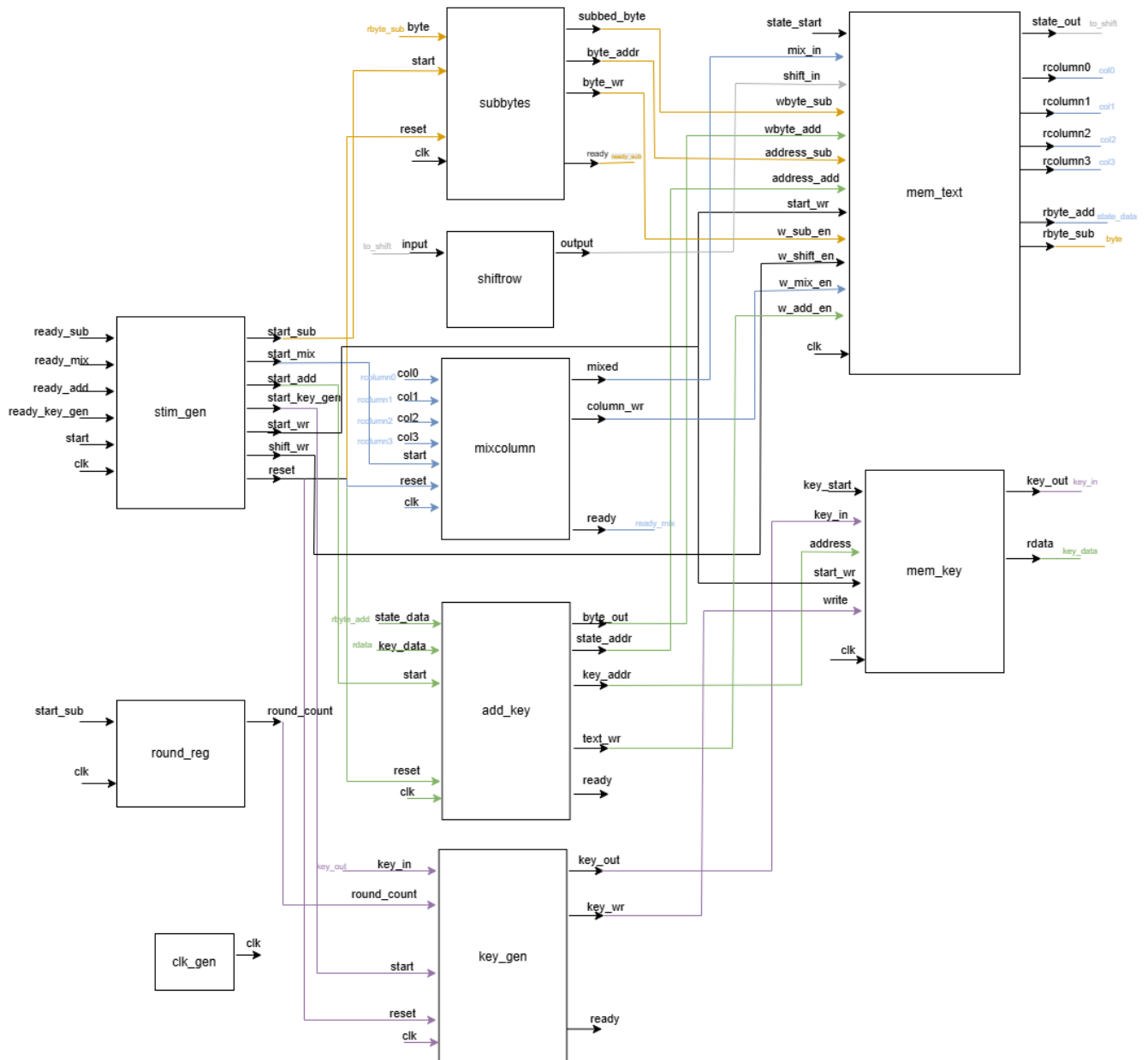
Read трансакција започиње постављањем адресе регистра који се чита. Након прихватања адресе, AES модул поставља *read* податке на магистралу и сигнализира валидност података помоћу *rvalid* сигнала.

Табела 1 приказује најважније сигнале реализованог AXI4-Lite интерфејса.

Назив сигнала	Опис сигнала
s00_axi_awaddr	Адреса регистра коме се приступа током <i>write</i> трансакције
s00_axi_awvalid	Индикација валидности <i>write</i> адресе
s00_axi_awready	Индикација да је AES модул спреман да прихвати адресу
s00_axi_wdata	Податак који се уписује у регистар
s00_axi_wstrb	Сигнал валидности појединачних бајтова податка
s00_axi_wvalid	Индикација валидности <i>write</i> података
s00_axi_wready	Индикација да је модул спреман за прихватање података
s00_axi_bresp	Статус <i>write</i> трансакције – није коришћен
s00_axi_bvalid	Индикација валидности <i>write response</i> сигнала
s00_axi_bready	Потврда <i>master</i> компоненте да је <i>response</i> прихваћен
s00_axi_araddr	Адреса регистра која се чита
s00_axi_arvalid	Индикација валидности <i>read</i> адресе
s00_axi_arready	Индикација да је модул прихватио <i>read</i> адресу
s00_axi_rdata	Прочитани податак
s00_axi_rresp	Статус <i>read</i> трансакције – није коришћен
s00_axi_rvalid	Индикација валидности <i>read</i> података
s00_axi_rready	Потврда <i>master</i> компоненте да је прочитани податак прихваћен

Табела 1: сигнали AXI4 Lite интерфејса

2.7.2 Интерфејси имплементираниог дизајна алгоритма AES 128-битне енкрипције



Слика 13 : Интерфејси AES128-битног алгоритма

Интерфејс података *subBytes* компоненте састоји се из следећих портова:

- *byte* - улазна магистрала *std_logic_vector* типа ширине 8 бита
- *Subbed_byt* - излазна магистрала *std_logic_vector* типа ширине 8 бита
- *Byte_addr* - адресна магистрала *std_logic_vector* типа ширине 4 бита
- *Byte_wr* - контролна магистрала ширине 1 бит
- *Start* - командни једнобитни порт
- *Ready* - статусни једнобитни порт

Интерфејс података *addKey* компоненте састоји се из следећих портова:

- *State_data* - улазна магистрала *std_logic_vector* типа ширине 8 бита
- *key_data* - улазна магистрала *std_logic_vector* типа ширине 8 бита

- *byte_out* - излазна магистрала *std_logic_vector* типа ширине 8 бита
- *state_addr* - адресна магистрала *std_logic_vector* типа ширине 4 бита
- *key_addr* - адресна магистрала *std_logic_vector* типа ширине 4 бита
- *text_wr* - контролна магистрала ширине 1 бит
- *Start* - командни једнобитни порт
- *Ready* - статусни једнобитни порт

Интерфејс података *mixColumn* компоненте састоји се из следећих портова:

- *Col0* - улазна магистрала *std_logic_vector* типа и ширине 32 бита
- *Col1* - улазна магистрала *std_logic_vector* типа и ширине 32 бита
- *Col2* - улазна магистрала *std_logic_vector* типа и ширине 32 бита
- *Col3* - улазна магистрала *std_logic_vector* типа и ширине 32 бита
- *mixed* - излазна магистрала *std_logic_vector* типа и ширине 128 бита
- *column_wr* - контролна магистрала ширине 1 бит
- *Start* - командни једнобитни порт
- *Ready* - статусни једнобитни порт

Интерфејс података *key expansion* компоненте састоји се из следећих портова:

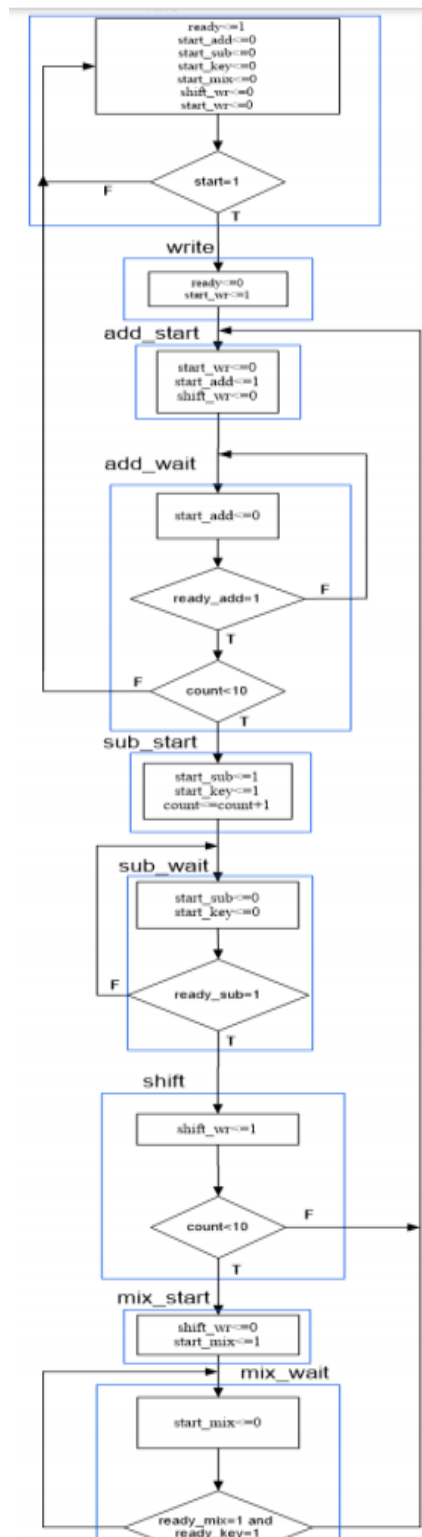
- *Key_in* - улазна магистрала *std_logic_vector* типа ширине 128 бита
- *Round_count* - улазна магистрала *integer* типа
- *Key_out* - излазна магистрала *std_logic_vector* типа ширине 128 бита
- *key_wr* - контролна магистрала ширине 1 бит
- *Start* - командни једнобитни порт
- *Ready* - статусни једнобитни порт

Интерфејс података *shiftRows* компоненте састоји се из следећих портова:

- *Input* - улазна магистрала *std_logic_vector* типа ширине 128 бита
- *Output* - излазна магистрала *std_logic_vector* типа ширине 128 бита

2.7.3 ASMD дијаграм и datapath round algoritma

На Слици 13 налази се ASMD дијаграм главног контролера процеса AES енкрипције. Дијаграм описује управљање комплетним током енкрипције кроз координацију свих компоненти система: *addKey*, *subBytes*, *shiftRows*, *mixColumn* и *keyGen*[4].



Слика 14: ASMD дијаграм round алгоритма

У почетном стању *idle* постављају се почетне вредности контролних сигнала, онемогућава се упис у меморију и активира сигнал *ready*, а систем чека пријем сигнала *start* којим се започиње процес енкрипције. Након активирања *start* сигнала прелази се у стање *write*, где се иницијализују потребни регистри и припремају подаци за почетак рада алгоритма. Затим се улази у стање *add_start*, у коме се покреће компонента *AddKey* ради извршавања почетне XOR операције између улазних

података и почетног кључа. У стању *add_wait* контролер чека завршетак рада *AddKey* компоненте преко сигнала *ready_add*. Након завршене операције проверава се број извршених рунди. После тога прелази се у стање *sub_start*, где се паралелно покрећу компоненте *subBytes* и *keyGen*. На овај начин омогућава се истовремено извршавање замене бајтова и генерисања новог кључа за рунду, чиме се повећава ефикасност и убрзава процес енкрипције. У стању *sub_wait* контролер чека завршетак *subBytes* компоненте, односно пријем сигнала *ready_sub*. Након тога прелази се на операцију *ShiftRows*, где се врши циклично померање редова матрице стања према правилима алгоритма. После *ShiftRows* операције, током првих 9 рунди извршава се *mixColumn*, која трансформише колоне матрице коришћењем операција у *Galoisovom* пољу. У последњој рунди ова операција се прескаче. Након завршетка свих операција тренутне рунде поново се активира компонента *AddKey*, где се користи новогенерисани кључ добијен из *KeyGen* компоненте. Процес се извршава док се не заврше све рунде енкрипције[3].

3. Функционална верификација хардвера

3.1 Увод

Функционална верификација хардвера је веома важан део развоја дигиталних система. Циљ верификације је да провери рад имплементираног хардвера у односу на спецификацију. Потребно је проверити да ли систем правилно извршава операције и да ли су излазни резултати тачни за различите комбинације улазних података. Неопходно је испитати исправност рада сваке компоненте, као и међусобну комуникацију између модула унутар система. Системи постају све сложенији и самим тим све су веће могућности грешака приликом имплементације, тако да је верификација неопходан корак приликом развоја система.

UVM методологија представља један од најчешће коришћених приступа функционалној верификацији. UVM омогућава систематичан и модуларан приступ верификацији, као и висок степен поновне употребљивости верификационих компоненти. Због тога, развој и одржавање верификационог окружења постају једноставнији, док се такође повећава ефикасност и поузданост верификације.

У оквиру овог рада извршена је функционална верификација коришћењем UVM методологије AES 128-битног алгоритма. Верификација је обухватила проверу рада основних компоненти: *subBytes*, *shiftRows*, *mixColumns*, *addRoundKey* и *keyGen* модуле. Наведени модули су интегрисани унутар *round_algorithm* модула, а комплетан систем је интегрисан у оквиру *AXI Wrapper-a*. Верификација AES 128-битног алгоритма обухвата проверу тачности обраде података током енкрипције, управљањем регистрима, *AXI4-Lite* интерфејса, ресет механизма, правилном генерисању кључева и синхронизацији свих компоненти унутар система, правилном понашању система у граничним случајевима.

3.2 Верификациони циклус

Верификациони циклус представља низ корака који се спроводе са циљем провере функционалне исправности хардверског дизајна. Основна сврха овог процеса јесте утврђивање да ли имплементирани систем у потпуности одговара захтевима спецификације.

Процес функционалне верификације чини неколико међусобно повезаних корака:

- анализа спецификације система,
- дефинисање верификационог плана,
- развој верификационог окружења,
- генерисање тест стимулуса,
- покретање симулација,
- анализа резултата,
- анализа функционалне покривености

Анализа спецификације

Почетна фаза верификационог циклуса подразумева детаљну анализу функционалних захтева система. На основу спецификације дефинишу се очекиване функционалности, ограничења система и критични сценарији који морају бити тестирани.

Дефинисање верификационог плана

Након анализе спецификације формира се верификациони план. У оквиру плана дефинишу се:

- типови и сценарији тестова,
- граничне вредности,
- *constrained-random* сценарији,
- функционалности које је потребно верификовати,
- механизми провере резултата (*checkers*),
- *coverage* циљеви, односно када се верификација сматра завршеном

Добро дефинисан верификациони план омогућава систематичан приступ тестирању и лакше праћење покривености функционалности.

Развој верификационог окружења

UVM верификационо окружење које се састоји од следећих компоненти:

- UVC-ева који су инстанцирани у *environment*-у (унутар UVC се налазе *sequencer*, *driver*, *monitor*, *agent* компоненте),
- *scoreboard*,
- *environment*

Покретање симулације

Током симулације генеришу се различити сценарији рада система, укључујући:

- случајне вредности улазних података,
- различите временске размаке између трансакција,
- *reset* сценарије,
- невалидне приступе регистрима,
- додатни специфични тестови у зависности од карактеристика дизајна који се верификује

Оваквим приступом омогућено је тестирање система у великом броју различитих услова рада.

Анализа резултата

Током симулације врши се анализа добијених резултата и њихово поређење са очекиваним понашањем система. На овај начин омогућено је рано откривање функционалних проблема у систему. Поређење резултата може се реализовати применом референтног модела или анализом протокола комуникације. Уколико се

уочи одступање између стварног и очекиваног понашања геренише се одговарајућа грешка. Анализа резултата омогућава:

- откривање функционалних грешака,
- проверу исправности комуникационих протокола,
- анализу временских зависности,
- проверу граничних случајева

Анализа покривености

Последња фаза верификационог циклуса подразумева анализу *coverage* извештаја. *Functional coverage* омогућава процену да ли су сви релевантни сценарији тестирани. *Code coverage* показује покривеност како би се утврдило у којој мери су секвенце успешно активирале делове RTLа, док се функционална покривеност користи за процену покривености самог верификационог окружења и дефинисаних верификационих циљева.

На основу *coverage* резултата могуће је:

- идентификовати нетестиране сценарије,
- проширити постојеће тестове,
- повећати функционалну покривеност система

Процес верификације се понавља све док се не постигне задовољавајући степен покривености и не отклоне све уочене грешке.

3.3 UVM – Универзална верификациона методологија

UVM (енгл. *Universal Verification Methodology*) представља стандардизовану методологију за функционалну верификацију дигиталних система засновану на *SystemVerilog* језику. Данас UVM представља најшире прихваћену методологију у области функционалне верификације сложених хардверских система и интегрисаних кола.

Развој UVM методологије настао је као последица потребе за стандардизацијом и унапређењем постојећих верификационих техника. Пре настанка UVM-а коришћене су различите власничке методологије као што су OVM (Open Verification Methodology) и VMM (Verification Methodology Manual). Како су ове методологије имале сличне концепте и циљеве, јавила се потреба за њиховим уједињењем и стандардизацијом.

Фирма Acclera развила је UVM методологију, која је касније стандардизована од стране IEEE организације. Данас се UVM користи у развоју великог броја савремених дигиталних система, укључујући процесоре, комуникационе системе, SoC платформе и криптографске акцелераторе.

Основна идеја UVM методологије заснива се на моделовању комуникације помоћу трансакција. Уместо директног управљања сигнаlima ниског нивоа, комуникација се описује помоћу апстрактних трансакција које представљају логичке операције

система. Оваквим приступом омогућава се једноставније моделирање сложених система и већа поновна употреба кода.

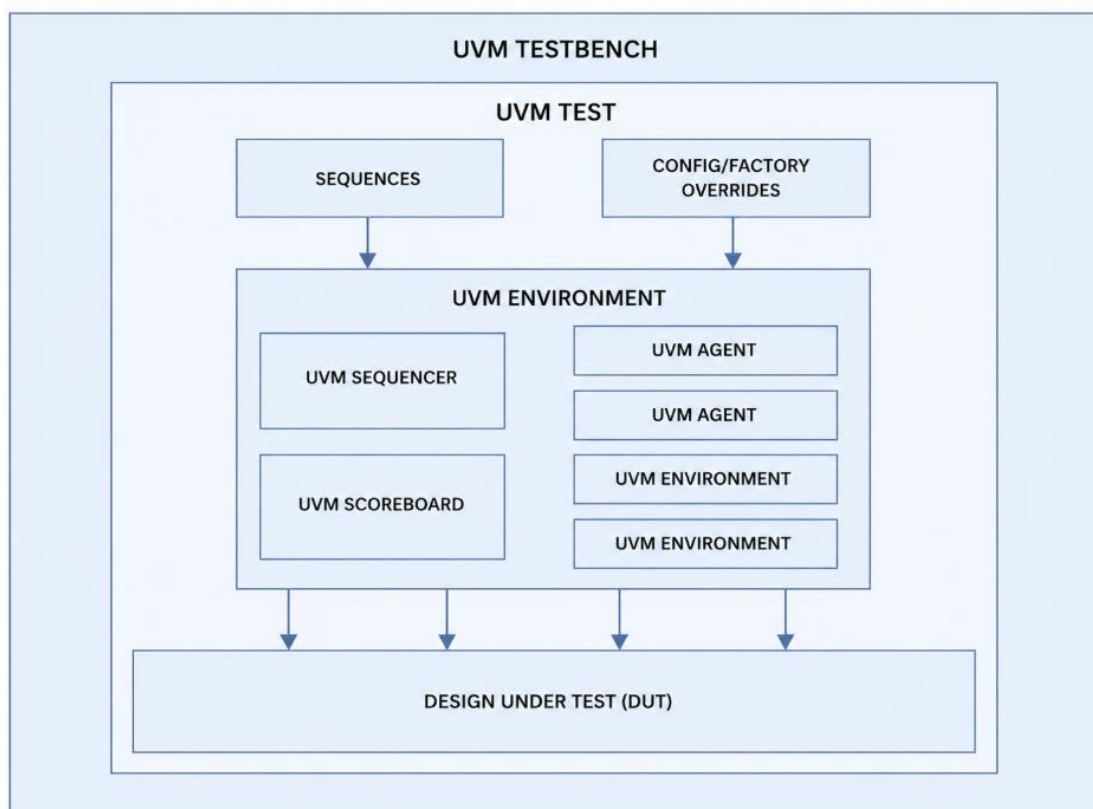
Једна од најзначајнијих предности UVM методологије јесте модуларна организација верификационог окружења. Свака компонента има јасно дефинисану улогу и комуникационе механизме, што омогућава лакше одржавање и проширење система. Поред тога, UVM обезбеђује висок степен поновне употребе компоненти у различитим пројектима.

Главне предности UVM методологије огледају се у:

- стандардизованој структури верификационог окружења,
- модуларности и лакшем одржавању кода,
- могућности поновне употребе компоненти,
- подршци за *constrained-random* верификацију,
- једноставнијем генерисању комплексних тест сценарија,
- подршци за *functional coverage*,
- лакшој аутоматизацији процеса верификације,
- једноставнијем откривању и анализи грешака.

3.4 Архитектура и компоненте верификационог окружења

Архитектура UVM верификационог окружења организована је хијерархијски, при чему се систем посматра као скуп међусобно повезаних компоненти различитог нивоа апстракције. Оваква организација омогућава јасну поделу одговорности, једноставније одржавање система и лакше проширење верификационог окружења.



Слика 15: архитектура UVM testbench

3.4.1 UVM Test

Test представља највиши ниво UVM хијерархије и дефинише конкретан сценарио симулације. Његова улога је конфигурација верификационог окружења и покретање одговарајућих секвенци.

Основне функције *test* компоненте су:

- конфигурација и инстанцирање верификационог окружења,
- избор тест сценарија,
- иницирање секвенци

3.4.2 UVM Environment

Environment представља централну компоненту верификационог окружења и налази се испод *test* компоненте. Његова улога је повезивање и организација свих осталих компоненти система.

Основне функције *environment* компоненте су:

- повезивање агената,
- повезивање scoreboard компоненти,
- организација комуникације између компоненти,
- конфигурација окружења,
- управљање TLM интерфејсима

3.4.3 UVM Agent

Agent представља логичку целину која садржи компоненте неопходне за комуникацију са DUT интерфејсом.

Agent обухвата:

- *sequencer*,
- *driver*,
- *monitor*

У зависности од начина рада, *agent* може бити:

- активан,
- пасиван

Активан *agent* генерише и шаље стимулусе DUT компоненти, док пасиван *agent* само прати активности на интерфејсу.

3.4.4 UVM Scoreboard

Scoreboard је компонента задужена за проверу исправности рада DUT компоненте. Његова основна улога је анализа резултата симулације и поређење добијених вредности са очекиваним резултатима.

Основне функције *scoreboard* компоненте су:

- поређење очекиваних и стварних резултата,
- коришћење референтног модела,
- детекција функционалних грешака,
- прикупљање coverage информација

У случају одступања од очекиваног понашања *scoreboard* генерише грешку.

3.4.5 UVM Sequencer

Sequencer представља компоненту која управља током трансакција у систему. Његова улога је синхронизација комуникације између *sequence* и *driver* компоненти.

Основне функције *sequencer*-а су:

- пријем трансакција,
- управљање редоследом извршавања,
- синхронизација комуникације са *driver* компонентом,
- прослеђивање трансакције *driver*-у,
- координација више *sequence* компоненти

3.4.6 UVM Driver

Driver представља активну компоненту која трансакције преводи у сигнале интерфејса DUT компоненте.

Основне функције *driver* компоненте су:

- генерисање сигнала интерфејса,
- управљање *handshake* механизмима,
- иницирање *read* и *write* операција,
- временска синхронизација сигнала,
- директна комуникација са DUT компонентом

Driver представља везу између трансакционог и сигналног нивоа апстракције.

3.4.7 UVM Monitor

Monitor је пасивна компонента чија је улога праћење сигнала интерфејса током симулације.

Основне функције *monitor* компоненте су:

- праћење активности на интерфејсу,
- реконструкција трансакција,
- анализа комуникације,
- прикупљање података током симулације,
- прослеђивање информација *scoreboard* компоненти

3.4.8 UVM Sequence

Улога *Sequence* је за генерисање трансакција и дефинисање редоследа њиховог извршавања.

Основне улоге *sequence* компоненте су:

- генерисање тест сценарија,
- *random* и *constrained-random* генерисање података,
- дефинисање редоследа трансакција,
- моделирање сложених сценарија рада система

Применом *random* и *constrained-random* приступа омогућено је тестирање великог броја различитих комбинација улазних података.

3.4.9 UVM Sequence item

Sequence item представља најнижи ниво трансакционе апстракције и описује једну трансакцију система.

Sequence item може да садржи:

- адресе,
- податке,
- контролне сигнале,
- тип трансакције

3.5 UVM фазе

UVM методологија дефинише скуп фаза извршавања које омогућавају синхронизован и организован рад свих компоненти верификационог окружења. Фазе су унапред дефинисане и извршавају се истим редоследом током сваке симулације. На тај начин обезбеђена је правилна иницијализација, повезивање, покретање и завршетак рада компоненти.

Најзначајније UVM фазе су:

- *build_phase*
- *connect_phase*
- *end_of_elaboration_phase*
- *start_of_simulation_phase*
- *run_phase*
- *extract_phase*
- *check_phase*
- *report_phase*
- *final_phase*

build_phase - у оквиру *build_phase* фазе врши се креирање свих UVM компоненти. Током ове фазе инстанцирају се *environment*, *agent*, *driver*, *monitor*, *sequencer* и

scoreboard компоненте, као и конфигурациони објекти неопходни за рад окружења.

connect_phase - служи за повезивање TLM портова, *analysis* интерфејса и виртуелних интерфејса. У овој фази *monitor* компонента се повезује са *scoreboard* компонентом, а *sequencer* са *driver* компонентом.

end_of_elaboration_phase - током *end_of_elaboration_phase* фазе завршава се конфигурисање окружења и проверава исправност хијерархије компоненти. Често се користи за приказ UVM топологије ради лакшег дебаговања.

start_of_simulation_phase - Ова фаза представља прелаз између конфигурисања окружења и почетка симулације. У њој се могу исписати параметри симулације, конфигурације и почетне информације о тесту.

run_phase - *run_phase* представља главну фазу симулације. У овој фази *sequencer* генерише трансакције, *driver* управља сигнаlima интерфејса, *monitor* прикупља сигнале са DUT компоненте, док *scoreboard* врши поређење добијених и очекиваних резултата. Највећи део функционалне верификације реализује се управо у оквиру ове фазе.

extract_phase - У *extract_phase* фази врши се прикупљање резултата симулације и статистичких података након завршетка *run_phase* фазе.

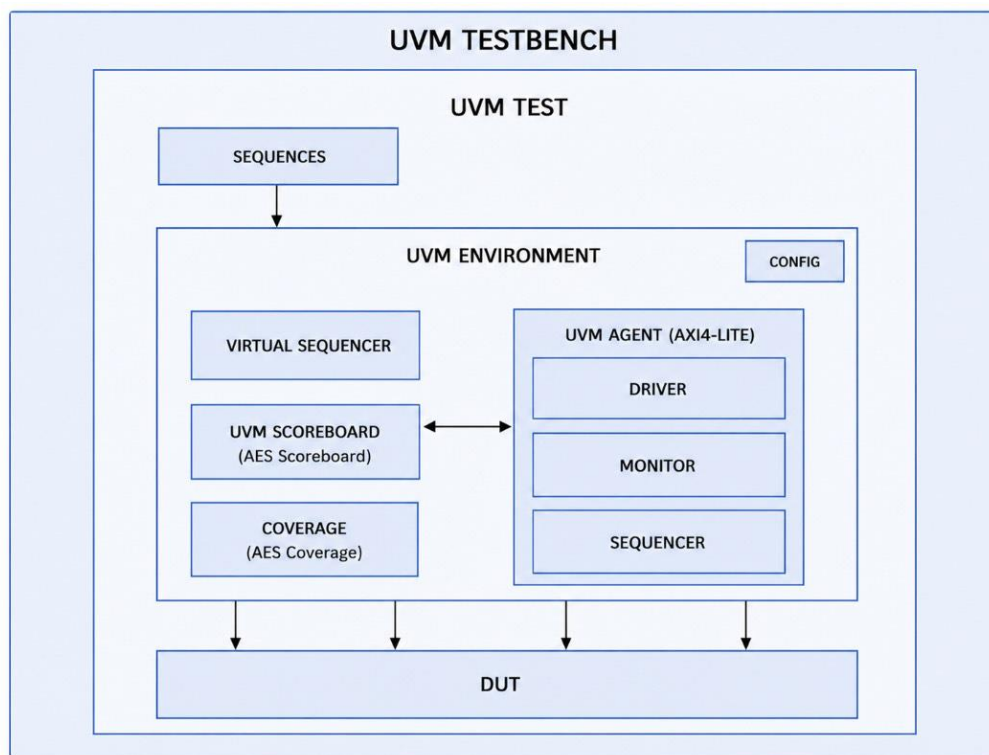
check_phase - служи за проверу исправности резултата симулације. У овој фази проверава се да ли су све трансакције успешно обрађене и да ли постоје неочекиване грешке.

report_phase - У *report_phase* фази врши се приказ коначних резултата симулације, броја грешака, упозорења и coverage статистике.

final_phase - представља последњу фазу симулације. У њој се врши ослобађање ресурса и завршне активности пре гашења симулације.

3.6 Архитектура верификационог окружења IP AES-128

У оквиру овог рада развијено је UVM (*Universal Verification Methodology*) верификационо окружење за проверу функционалности AES IP. Верификационо окружење реализовано је модуларно, тако да омогућава једноставно проширење, поновну употребу компоненти и подршку за различите тест сценарије.



Слика 16 : Testbench architecture

Комуникација са AES модулом реализована је путем AXI4-Lite интерфејса, док је читава верификациона архитектура организована у складу са UVM методологијом.

Архитектура окружења састоји се из следећих целина:

- *test* компонента
- *environment* компонента
- AXI4-Lite agent (*sequencer, driver, monitor*)
- *scoreboard*
- *sequence* и *virtual sequence* механизми
- *coverage* компоненте

У овом поглављу ће бити детаљно описана сама архитектура *testbench*а као и опис компоненти које се налазе у њему.

3.6.1 AXI-4 Lite UVC

У оквиру верификационог окружења реализован је AXI4-Lite UVC (*Universal Verification Component*). Компонента која омогућава генерисање, управљање и праћење AXI4-Lite трансакција. AXI UVC је организован хијерархијски и састоји се од:

- *agent*,
- *sequencer*,
- *driver*,
- *monitor*

AXI4-Lite agent

AXI агент представља централну компоненту UVC блока. У оквиру агента инстанцирани су *driver*, *monitor* и *sequencer* модули, као и *interface* који омогућава повезивање са DUT компонентом. Током *build_phase* фазе креирају се све компоненте агента и преузима се *virtual interface* из UVM *config* базе[5].

У *connect_phase* фази врши се повезивање *sequencer* и *driver*а преко *seq_item_port* механизма.

На Листингу 1 приказан је код реализованог AXI *agent*а.

```
class axi_agent extends uvm_component;

    `uvm_component_utils(axi_agent)
    virtual axi_if#(6,32) vif;
    axi_driver drv;
    axi_monitor mon;
    axi_sequencer seqr;

    function new(string name = "axi_agent", uvm_component parent = null);
        super.new(name, parent);
    endfunction

    function void build_phase(uvm_phase phase);
        if (!uvm_config_db#(virtual axi_if#(6,32))::get(this, "", "vif", vif))
            `uvm_fatal("NO_VIF", "Virtual interface not set")

        drv = axi_driver::type_id::create("drv", this);
        mon = axi_monitor::type_id::create("mon", this);
        seqr = axi_sequencer::type_id::create("seqr", this);

        drv.vif = vif;
        mon.vif = vif;
    endfunction

    function void connect_phase(uvm_phase phase);
        drv.seq_item_port.connect(seqr.seq_item_export);
    endfunction

endclass
```

Листинг 1: AXI4-Lite agent имплементација

AXI4-Lite sequencer

Sequencer је задужен за управљање редоследом трансакција и прослеђивање *sequence item* објеката *driver* компоненти. Ова компонента представља везу између *sequence* и *driver* модула и омогућава генерисање различитих сценарија тестирања[6].

У оквиру реализованог окружења *sequencer* наслеђује *uvm_sequencer* класу и користи AXI

транзакциони објекат *axi_seq_item*. На овај начин омогућено је генерисање *write* и *read* транзакција са различитим адресама, подацима и временским кашњењима.

На Листингу 2 приказана је имплементација *AXI sequencer* компоненте.

```
class axi_sequencer extends uvm_sequencer #(axi_seq_item);

    `uvm_component_utils(axi_sequencer)

    function new(string name, uvm_component parent);
        super.new(name, parent);
    endfunction

endclass
```

Листинг 2: *Sequencer* имплементација

AXI4-Lite driver

Driver компонента преузима транзакције са *sequencer*-а и преводи их у сигнале *AXI4-Lite* интерфејса. У оквиру *run_phase* фазе *driver* чека пристизање *sequence item* објеката, а затим у зависности од типа транзакције позива одговарајућу процедуру за *write* или *read* приступ.

Приликом *write* транзакције *driver* поставља адресу регистра на *awaddr* магистралу и податке на *wdata* магистралу. Након тога активира *awvalid* и *wvalid* сигнале и чека *handshake* механизам, односно истовремену активацију *VALID* и *READY* сигнала. По завршетку преноса *driver* чека *write response* сигнал и завршава транзакцију.

Код *read* транзакције *driver* поставља адресу регистра на *araddr* магистралу и активира *arvalid* сигнал. Након прихватања адресе чека се појава *rvalid* сигнала и преузимање података са *rdata* магистрале. Прочитани податак се затим враћа назад ка *sequencer* компоненти у облику *response* транзакције.

На Листинзима 3,4,5 приказан је код *AXI driver* компоненте.

```
task run_phase(uvm_phase phase);
    axi_seq_item tr;

    vif.awvalid <= 0;
    vif.wvalid  <= 0;
    vif.bready  <= 0;
    vif.arvalid <= 0;
    vif.rready  <= 0;
    vif.awaddr  <= 0;
    vif.awprot  <= 0;

    wait (vif.aresetn == 1);

    forever begin
        seq_item_port.get_next_item(tr);
        if (tr.rw) begin
            do_write(tr);
        end else
            do_read(tr);
        seq_item_port.item_done();
    end
endtask
```

Листинг 3: *AXI4-Lite driver run_phase* имплементација

```

task do_write(axi_seq_item tr);
    repeat(tr.trans_delay) begin
        @(posedge vif.aclk);
    end
    //set aw/v
    vif.awaddr <= tr.addr;
    vif.wdata <= tr.data;
    vif.wstrb <= tr.wstrb;

    vif.awvalid <= 1;
    vif.wvalid <= 1;
    vif.bready <= 1;

    @(posedge vif.aclk iff (vif.awready && vif.wready));
    vif.awvalid <= 0;
    vif.wvalid <= 0;

    wait(vif.bvalid && vif.bready);

endtask

```

Листинг 4: AXI4-Lite driver do write имплементација

```

task do_read(axi_seq_item tr);
    axi_seq_item rsp;
    //set araddr
    vif.araddr <= tr.addr;
    vif.arvalid <= 1;
    wait(vif.arready);

    @(posedge vif.aclk);
    vif.arvalid <= 0;
    vif.rready <= 1;

    //wait response
    wait(vif.rvalid);

    rsp = axi_seq_item::type_id::create("rsp");
    rsp.set_id_info(tr);
    rsp.rdata = vif.rdata;

    @(posedge vif.aclk);
    vif.rready <= 0;

    seq_item_port.put_response(rsp);

endtask

```

Листинг 5: AXI4-Lite driver do read имплементација

Оваква структура омогућава модуларност и једноставно проширење окружења.

AXI4-Lite monitor

Monitor компонента пасивно прати сигнале AXI4-Lite интерфејса без утицаја на рад DUT компоненте. Основна улога *monitors* је реконструкција трансакција на основу активности сигнала интерфејса и прослеђивање прикупљених информација *scoreboard* компоненти.

Monitor детектује *write* и *read* трансакције праћењем *VALID/READY handshake* механизма. Код *write* приступа прикупља адресу, податке и *wstrb* сигнале, као и информацију о кашњењу у односу на претходну трансакцију, док код *read* приступа прати адресу регистра и вредност прочитаних података и такође и информацију о кашњењу у односу на претходну трансакцију. Скупљене трансакције се затим прослеђују преко *analysis_port*.

Поред прикупљања података о трансакцијама, монитор врши мерење кашњења између *valid* && *awready* једне трансакције и следећег *valid* сигнала (иницирања наредне трансакције).

На Листинзима 6 и 7 приказана је имплементација AXI monitor компоненте.

```

task count_delay();
    forever begin
        cnt_delay = 0;
        @(posedge vif.aclk iff ((vif.awvalid && vif.awready) || (vif.arvalid && vif.arready)));
        @(negedge vif.aclk);
        if(!vif.awvalid && !vif.arvalid) cnt_delay++;

        fork
            wait(vif.awvalid || vif.arvalid);

            forever begin
                @(negedge vif.aclk iff(!vif.awvalid || !vif.arvalid));
                cnt_delay++;

            end
        join_any
        disable fork;
        cnt_delay_prev = cnt_delay;
    end
endtask

task run_phase(uvm_phase phase);
    fork
        collect_item();
        count_delay();
    join
endtask

```

Листинг 6: AXI4-Lite monitor имплементација count_delay i run_phase taskova

```

task collect_item;
    first_tx = 1;
    forever begin
        @(posedge vif.aclk iff ((vif.awvalid && vif.awready) || (vif.arvalid && vif.arready)));
        // WRITE
        if(vif.awvalid && vif.awready) begin
            tr.addr = vif.awaddr;
            tr.rw   = AXI_WRITE;

            wait(vif.wvalid && vif.wready);
            tr.data = vif.wdata;
            tr.wstrb = vif.wstrb;

        end
        // READ
        else begin
            tr.addr = vif.araddr;
            tr.rw   = AXI_READ;

            wait(vif.rvalid && vif.rready);

            tr.rdata = vif.rdata;

        end
        tr.trans_delay = cnt_delay_prev;
        ap.write(tr);
        first_tx = 0;

        tr.print();
    end
endtask

```

Листинг 7: AXI4-Lite monitor имплементација collect_item taska

3.6.2 AES Environment

Environment представља централну компоненту верификационог окружења и задужен је за повезивање свих осталих UVM компоненти. У оквиру *environmenta* инстанцирани су *AXI4-Lite agent*, *scoreboard*, *virtual sequencer*, *coverage* компоненте. Унутар *build_phase* фазе креирају се *agent*, *scoreboard* и *virtual_sequencer*. Док су у склопу *connect_phase* фазе међусобно повезани.

На Листингу 8 приказана је имплементација *environment* компоненте.

```
class aes_env extends uvm_env;
  `uvm_component_utils(aes_env)

  aes_scoreboard m_scbd;
  axi_agent      m_axi;

  aes_virtual_sequencer m_virtual_sequencer;

  function new(string name="aes_env", uvm_component parent = null);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    // create axi agent
    m_axi = axi_agent::type_id::create("m_axi", this);
    // create scoreboard
    m_scbd = aes_scoreboard::type_id::create("m_scbd", this);
    // create virtual sequencer
    m_virtual_sequencer = aes_virtual_sequencer::type_id::create("m_virtual_sequencer", this);
  endfunction

  function void connect_phase(uvm_phase phase);
    m_axi.mon.ap.connect(m_scbd.axi_imp);
    // connect sequencers in virtual sequencer
    m_virtual_sequencer.m_axi_seqr = m_axi.seqr;
  endfunction
endclass
```

Листинг 8: AES environment имплементација

3.6.3 AES Scoreboard

Scoreboard компонента представља централни елемент верификационог окружења задужен за проверу функционалне исправности AES језгра. Њена основна улога јесте да прати *AXI4-Lite* трансакције које долазе из монитора, формира референтни модел регистара и упоређује очекиване вредности са вредностима које враћа DUT.

Имплементација *scoreboard* компоненте користи функцију *write(axi_seq_item tr)*, која се активира при свакој примљеној трансакцији са монитора. У оквиру функције прво се врши идентификација типа трансакције, односно одређивање да ли је у питању операција уписа или читања.

Код операција уписа *scoreboard* ажурира референтне регистре коришћењем адресе трансакције и пристиглих података. На основу вредности *wstrb* сигнала имплементирана је функција *apply_wstrb()*, која омогућава парцијални упис бајтова у

На Листинзима 10, 11, 12 приказана је имплементација дела *write()* функције, која обухвата читања из регистра и проверу исправности рада дизајна.

```

else begin
  case (tr.addr)

    CIPHER0_ADDR: ciphertext_dut[127:96] = tr.rdata;
    CIPHER1_ADDR: ciphertext_dut[95:64]  = tr.rdata;
    CIPHER2_ADDR: ciphertext_dut[63:32]  = tr.rdata;
    CIPHER3_ADDR: ciphertext_dut[31:0]   = tr.rdata;

    PLAIN0_ADDR: plaintext_dut[127:96] = tr.rdata;
    PLAIN1_ADDR: plaintext_dut[95:64]  = tr.rdata;
    PLAIN2_ADDR: plaintext_dut[63:32]  = tr.rdata;
    PLAIN3_ADDR: plaintext_dut[31:0]   = tr.rdata;

    KEY0_ADDR: key_dut[127:96] = tr.rdata;
    KEY1_ADDR: key_dut[95:64]  = tr.rdata;
    KEY2_ADDR: key_dut[63:32]  = tr.rdata;
    KEY3_ADDR: key_dut[31:0]   = tr.rdata;

  endcase
  cov.addr = tr.addr;
  cov.rw = tr.rw;
  cov.axi_cg.sample();

```

Листинг 10 : Део *write()* функције који скупља вредности регистра из DUTa

```

if (tr.addr == START_ADDR) begin
  if (tr.rdata[0] != start_ref) begin
    `uvm_error("AES_SCBD", $sformatf(" START BIT MISMATCH!\nDUT = %h\nREF = %h", tr.rdata[0], start_ref))
  end
  else begin
    `uvm_info("AES_SCBD", $sformatf("START BIT MATCH OK: %h", start_dut), UVM_LOW)
  end
end

if (tr.addr == CIPHER3_ADDR) begin
  if (ciphertext_dut != ciphertext_ref) begin
    `uvm_error("AES_SCBD", $sformatf("CIPHERTEXT MISMATCH!\nDUT = %h\nREF = %h", ciphertext_dut, ciphertext_ref))
  end
  else begin
    `uvm_info("AES_SCBD", $sformatf("CIPHERTEXT MATCH OK: %h", ciphertext_dut), UVM_LOW)
  end
end

if (tr.addr == PLAIN3_ADDR) begin
  if (plaintext_dut != plaintext_ref) begin
    `uvm_error("AES_SCBD", $sformatf("PLAINTEXT MISMATCH!\nDUT = %h\nREF = %h", plaintext_dut, plaintext_ref))
  end
  else begin
    `uvm_info("AES_SCBD", $sformatf("PLAINTEXT MATCH OK: %h", plaintext_dut), UVM_LOW)
  end
end

if (tr.addr == KEY3_ADDR) begin
  if (key_dut != key_ref) begin
    `uvm_error("AES_SCBD", $sformatf("KEY MISMATCH!\nDUT = %h\nREF = %h", key_dut, key_ref))
  end
  else begin
    `uvm_info("AES_SCBD", $sformatf("KEY MATCH OK: %h", key_dut), UVM_LOW)
  end
end
end

```

Листинг 11: Део *write()* функције који обухвата проверу исправности једног дела регистра

```

  if (tr.addr == READY_ADDR) begin
    if (tr.rdata != ready_ref) begin
      `uvm_error("AES_SCBD", $sformatf("READY FLAG MISMATCH!\nDUT = %h\nREF = %h", tr.rdata, ready_ref))
    end
    else begin
      `uvm_info("AES_SCBD", $sformatf("READY FLAG OK: %h", ready_ref), UVM_LOW)
    end
  end
end

endfunction

```

Листинг 12: Део *write()* функције који обухвата проверу исправности другог дела регистра

Поред функционалне провере, *scoreboard* садржи и интеграцију са *coverage* компонентом. Током извршавања трансакција врши се узорковање различитих *covergroup* структура. На тај начин *scoreboard* истовремено обавља функционалну проверу и доприноси прикупљању *coverage* података.

3.6.4 AES Coverage model

У оквиру верификационог окружења имплементирани су функционалне *covergroup* структуре са циљем праћења активности *IP* и процене нивоа покривености током симулације. Покривеност је реализована употребом *SystemVerilog covergroup* механизма и обухвата проверу вредности улазних регистара, AXI комуникације, ресет сценарија и недозвољених стања током рада система[8].

За проверу вредности *plaintext* и *key* регистра имплементирана је *covergroup* структура *aes_reg_value_cg*. У оквиру ове групе дефинисани су *coverpoint* елементи за сва четири 32-битна сегмента *plaintext* и *key* вредности. За сваки сегмент дефинисани су следећи *bins* опсези:

- вредност нула,
- вредност са свим јединицама,
- ниски опсег вредности,
- средњи опсег вредности,
- високи опсег вредности

Покривеност се узоркује позивом функције *sample_regs*, у оквиру које се *plaintext* и *key* вредности раздвајају на четири 32-битна сегмента и прослеђују *covergroup* структурама ради прикупљања статистике покривености.

Поред појединачних *coverpoint* елемената, дефинисани су и посебни *coverpoint* елементи *cp_all_zero* и *cp_all_one*, који прате случајеве када су сви улазни подаци постављени на нулу, односно када су сви битови постављени на логичку јединицу. Ови случајеви представљају значајне граничне услове током функционалне верификације AES алгорита.

На Листинзима 13 и 14 приказана је имплементација *aes_reg_value_cg*.

```

// covergroup
covergroup aes_reg_value_cg;

    cp_plain_0 : coverpoint plain_0 {
        bins zero = {32'h00000000};
        bins ones = {32'hFFFFFFFF};
        bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
        bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
        bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
    }

    cp_plain_1 : coverpoint plain_1 {
        bins zero = {32'h00000000};
        bins ones = {32'hFFFFFFFF};
        bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
        bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
        bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
    }

    cp_plain_2 : coverpoint plain_2 {
        bins zero = {32'h00000000};
        bins ones = {32'hFFFFFFFF};
        bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
        bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
        bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
    }

    cp_plain_3 : coverpoint plain_3 {
        bins zero = {32'h00000000};
        bins ones = {32'hFFFFFFFF};
        bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
        bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
        bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
    }

```

Листинг 13: Имплементација једног дела `aes_reg_value_cg`

```

// KEY
cp_key_0 : coverpoint key_0 {
    bins zero = {32'h00000000};
    bins ones = {32'hFFFFFFFF};
    bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
    bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
    bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
}

cp_key_1 : coverpoint key_1 {
    bins zero = {32'h00000000};
    bins ones = {32'hFFFFFFFF};
    bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
    bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
    bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
}

cp_key_2 : coverpoint key_2 {
    bins zero = {32'h00000000};
    bins ones = {32'hFFFFFFFF};
    bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
    bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
    bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
}

cp_key_3 : coverpoint key_3 {
    bins zero = {32'h00000000};
    bins ones = {32'hFFFFFFFF};
    bins low  = {[32'h00000001 : 32'h3FFFFFFF]};
    bins mid  = {[32'h40000000 : 32'hBFFFFFFF]};
    bins high = {[32'hC0000000 : 32'hFFFFFFFE]};
}

```

Листинг 14: Имплементација другог дела `aes_reg_value_cg`

За проверу исправности рада система током ресетовања имплементирана је *covergroup* структура *aes_reset_cg*. У оквиру ове групе прати се појава ресета током активног рада, односно током стања заузетости система.

На Листингу 15 приказана је имплементација *aes_reset_cg*.

```
covergroup aes_reset_cg;

    cp_reset_busy : coverpoint reset_during_busy {
        bins happened = {1};
    }

endgroup
```

Листинг 15: Имплементација *aes_reset_cg*

Поред тога, реализована је и *covergroup* структура *aes_invalid_start_during_busy_cg*, чији је циљ праћење недозвољених *start* захтева током извршавања енкрипције. У оквиру ове групе покривена су различита интерна стања AES алгоритма, као што су:

- *ADD_WAIT*,
- *SUB_START*,
- *SUB_WAIT*,
- *SHIFT*,
- *MIX_START*,
- *MIX_WAIT*,
- *ADD_START*

На овај начин омогућена је провера реакције система на невалидне *start* захтеве током различитих фаза рада алгоритма.

На Листингу 16 приказана је имплементација *aes_invalid_start_during_busy_cg*.

```
covergroup aes_invalid_start_during_busy_cg;

    cp_state : coverpoint invalid_start_state {

        bins add_wait  = {ADD_WAIT};
        bins sub_start = {SUB_START};
        bins sub_wait  = {SUB_WAIT};
        bins shift     = {SHIFT};
        bins mix_start = {MIX_START};
        bins mix_wait  = {MIX_WAIT};
        bins add_start = {ADD_START};

    }

endgroup
```

Листинг 16: Имплементација *aes_invalid_start_during_busy_cg*

За AXI интерфејс имплементирана је *covergroup* структура *axi_cg*. У оквиру ове групе праћене су вредности *wstrb* сигнала, адресе приступа регистрима, као и размак између трансакција *trans_delay*. Дефинисани су *bins* елементи за:

- *plaintext* регистре,
- *key* регистре,
- *start* регистар,
- *ciphertext* регистре,
- *status* регистар
- *trans_delay*

Такође је реализован *cross coverage* између адресе приступа и типа операције (*read* и *write*) чиме је омогућена провера да ли су све комбинације читања и уписа над регистрима успешно покривене током симулације.

На Листинзима 17 и 18 приказана је имплементација *axi_cg*.

```
covergroup axi_cg;
  cp_wstrb_bit0 : coverpoint wstrb[0] {
    bins zero = {0};
  }

  cp_wstrb_bit1 : coverpoint wstrb[1] {
    bins zero = {0};
  }

  cp_wstrb_bit2 : coverpoint wstrb[2] {
    bins zero = {0};
  }

  cp_wstrb_bit3 : coverpoint wstrb[3] {
    bins zero = {0};
  }

  cp_wstrb_allzero : coverpoint wstrb {
    bins zero = {4'h0};
  }

  cp_addr: coverpoint addr {

    bins plaintext[] = {
      32'h00,32'h04,32'h08,32'h0C
    };

    bins key[] = {
      32'h10,32'h14,32'h18,32'h1C
    };

    bins start = {32'h20};
    bins ciphertext[] = { 32'h24,32'h28,32'h2C,32'h30};

    bins status = {32'h34};
  }
}
```

Листинг 17: Имплементација једног дела *axi_cg*

```
cp_rw: coverpoint rw {
  bins read = {0};
  bins write = {1};
}

trans_delay_rw: coverpoint trans_delay {
  bins b2b = {0};
  bins gap1_5 = {[1:5]};
}

cross_addr_rw : cross cp_addr, cp_rw;
endgroup
```

Листинг 18: Имплементација другог дела *axi_cg*

На Сlici 17 приказан је резултат укупне покривености.

Total Groups Coverage Summary

Score	Inst Score
100	100

Слика 17: Резултат укупне покривености

3.6.5 Тестови и секвенце

У оквиру UVM верификационог окружења креиран је скуп секвенци намењених провери функционалности AES IP и AXI интерфејса. Имплементиране су следеће секвенце:

1. *aes_random_seq*,
2. *aes_reset_seq*,
3. *aes_invalid_access_seq*,
4. *aes_reg_access_seq*

3.6.5.1 *aes_random_seq*

У оквиру секвенце дефинисани су *random* генерисани параметри:

- *plaintext* - 128-битни улазни текст,
- *key* - 128-битни AES кључ,
- *mode* - режим рада секвенце,
- *wstrb* - AXI *write strobe* сигнал.

Ради правилне *randomizacije* дефинисан је *solve order* механизам, којим је обезбеђено да се параметар *mode* генерише пре вредности *plaintext*, *key* и *wstrb* сигнала. На овај начин омогућено је условно формирање улазних вредности у зависности од изабраног режима рада.

У оквиру секвенце имплементирана су ограничења (*constraint*) за *wstrb* сигнал, чиме су дефинисане различите комбинације активних бајтова током AXI *write* трансакција. Највећу вероватноћу има вредност 4'b1111, док су остале комбинације распоређене са мањом вероватноћом појављивања. Овим приступом омогућено је тестирање различитих сценарија делимичног уписа података.

Поред тога, дефинисана су ограничења за *plaintext* и *key* вредности. У случају када је *mode* = 0, *plaintext* и *key* постављају се на нулту вредност, док се за *mode* = 1 генеришу максималне вредности. На овај начин реализовани су значајни гранични случајеви AES алгоритма.

AES IP језгро прихвата улаз искључиво преко 128-битног регистра формираног преко 128-битног регистра формираног од четири 32-битна регистра. У случају да се не

упишу сви регистр, неуписани делови остају на вредности 0, тако да језгро и даље обрађује 128-битни податак. Због тога сценарио улазног текста краћег од 128 бита није могуће директно генерисати на нивоу AES језгра.

На Листингу 19 приказана су имплементирана ограничења приликом *randomizacije*.

```
constraint solve_order_c {
    solve mode before plaintext;
    solve mode before wstrb;
    solve mode before key;
}

constraint c_wstrb {
    wstrb dist {
        4'b1111 := 40,
        4'b0001 := 10,
        4'b0010 := 10,
        4'b0100 := 10,
        4'b1000 := 10,
        4'b0011 := 5,
        4'b1100 := 5,
        4'b0110 := 5,
        4'b0101 := 2,
        4'b1010 := 2,
        4'b0000 := 1
    };
}

constraint c_plaintext {
    (mode == 0) -> plaintext == 128'h0 && wstrb == 4'hF;
    (mode == 1) -> plaintext == 128'hFFFFFFFFFFFFFFFFFFFFFFFF && wstrb == 4'hF;
}

constraint c_key {
    (mode == 0) -> key == 128'h0 && wstrb == 4'hF;
    (mode == 1) -> key == 128'hFFFFFFFFFFFFFFFFFFFFFFFF && wstrb == 4'hF;
}
```

Листинг 19: Ограничења приликом *randomizacije aes_random_seq*

Тело секвенце реализовано је у оквиру *body()* методе. Током извршавања секвенце:

- врши се *randomizacija* улазних параметара,
- *plaintext* вредност се уписује у AES регистре,
- *key* вредност се уписује у AES key регистре,
- генерише се *start* сигнал за покретање енкрипције,
- врши се *polling status* регистра до завршетка енкрипције,
- читају се резултати шифрата из излазних регистара

Упис *plaintext* и *key* вредности реализован је кроз више AXI *write* трансакција, при чему се 128-битне вредности деле на четири 32-битна сегмента. За адресирање регистара коришћени су параметри дефинисани у посебном *package* модулу, чиме је избегнута употреба фиксних *hexadecimal* вредности у коду, као и поједностављена употреба уколико дође до ширења имплементације или измена унутар дизајна.

Након покретања енкрипције, секвенца периодично чита *status* регистар све док *DUT* не сигнализира завршетак операције постављањем *ready* бита. По завршетку енкрипције читају се *ciphertext* регистри и преузима резултат енкрипције.

На Листингу 20 приказана је имплементација *body()* методе.

```
task body();
    axi_seq_item req;
    bit [31:0] rdata;
    super.body();
    for(int num_of_iter = 0; num_of_iter < 50; num_of_iter++ ) begin
        this.randomize();

        // PLAINTEXT
        axi_write(PLAIN0_ADDR, plaintext[127:96],wstrb);
        if(mode!=0 && mode!= 1) randomize(wstrb);
        axi_write(PLAIN1_ADDR, plaintext[95:64], wstrb);
        if(mode!=0 && mode!= 1) randomize(wstrb);
        axi_write(PLAIN2_ADDR, plaintext[63:32], wstrb);
        if(mode!=0 && mode!= 1) randomize(wstrb);
        axi_write(PLAIN3_ADDR, plaintext[31:0], wstrb);

        // KEY
        axi_write(KEY0_ADDR, key[127:96], wstrb);
        axi_write(KEY1_ADDR, key[95:64], wstrb);
        axi_write(KEY2_ADDR, key[63:32], wstrb);
        axi_write(KEY3_ADDR, key[31:0], wstrb);

        // START
        axi_write(START_ADDR, 32'h00000001, 4'hF);
        axi_write(START_ADDR, 32'h00000000, 4'hF);

        // POLLING
        do begin
            axi_read(READY_ADDR, rdata);
        end while (rdata[0] == 0);

        // READ RESULT
        axi_read(CIPHER0_ADDR, rdata);
        axi_read(CIPHER1_ADDR, rdata);
        axi_read(CIPHER2_ADDR, rdata);
        axi_read(CIPHER3_ADDR, rdata);
    end
```

Листинг 20: Имплементација *body()* методе *aes_random_seq*

Оваква организација секвенце омогућава:

- *random* тестирање AES IP,
- проверу граничних случајева,
- проверу AXI write strobe комбинација,
- генерисање различитих временских сценарија,
- проверу стабилности DUT модула током великог броја трансакција.

3.6.5.2 *aes_reg_access_seq*

Секвенца *aes_reg_access_seq* реализована је са циљем провере исправног приступа свим регистрима AES IP језгра путем AXI интерфејса. У оквиру секвенце врши се упис и читање вредности из регистра. На почетку секвенце генеришу се *random* вредности *plaintext* и *key* података, након чега се оне уписују у одговарајуће регистре. Након уписа, врши се читање истих адреса како би се проверила исправност *read/write* операција.

Даље, секвенца активира AES енкрипцију постављањем start бита, након чега се врши *polling status* регистра све док DUT не сигнализира завршетак енкрипције. По завршетку рада, читају се *ciphertext* регистри.

У оквиру секвенце имплементиран је и негативан сценарио у коме се покушава упис у излазни регистар *ciphertext* и *ready* регистре који су предвиђени само за читање. На овај начин проверава се исправно понашање DUT-а при недозвољеном приступу регистрима.

На Листинзима 21 и 22 налази се имплементација *body()* дела *aes_reg_access_seq*.

```
for(int num_of_iter = 0; num_of_iter < 5; num_of_iter++ ) begin
    this.randomize();

    // PLAINTEXT
    axi_write(PLAIN0_ADDR, plaintext[127:96], 4'hF);
    axi_write(PLAIN1_ADDR, plaintext[95:64], 4'hF);
    axi_write(PLAIN2_ADDR, plaintext[63:32], 4'hF);
    axi_write(PLAIN3_ADDR, plaintext[31:0], 4'hF);

    axi_read(PLAIN0_ADDR, rdata);
    axi_read(PLAIN1_ADDR, rdata);
    axi_read(PLAIN2_ADDR, rdata);
    axi_read(PLAIN3_ADDR, rdata);

    // KEY
    axi_write(KEY0_ADDR, key[127:96], 4'hF);
    axi_write(KEY1_ADDR, key[95:64], 4'hF);
    axi_write(KEY2_ADDR, key[63:32], 4'hF);
    axi_write(KEY3_ADDR, key[31:0], 4'hF);

    axi_read(KEY0_ADDR, rdata);
    axi_read(KEY1_ADDR, rdata);
    axi_read(KEY2_ADDR, rdata);
    axi_read(KEY3_ADDR, rdata);

    // START
    axi_write(START_ADDR, 32'h00000001, 4'hF);
    axi_read(START_ADDR, rdata);
    axi_write(START_ADDR, 32'h00000000, 4'hF);
    axi_read(START_ADDR, rdata);
end
```

Листинг 21: Имплементација *body()* дела *aes_reg_access_seq*

```
// POLLING
axi_write(READY_ADDR, 32'h00000001, 4'hF);
axi_read(READY_ADDR, rdata);
do begin
    axi_read(READY_ADDR, rdata);
end while (rdata[0] == 0);

// try to corrupt result
axi_write(CIPHER0_ADDR, ciphertext, 4'hF);
axi_write(CIPHER1_ADDR, ciphertext, 4'hF);
axi_write(CIPHER2_ADDR, ciphertext, 4'hF);
axi_write(CIPHER3_ADDR, ciphertext, 4'hF);

axi_read(CIPHER0_ADDR, rdata);
axi_read(CIPHER1_ADDR, rdata);
axi_read(CIPHER2_ADDR, rdata);
axi_read(CIPHER3_ADDR, rdata);
end
ndtask
```

Листинг 22: Имплементација невалидних уписа унутар *body()* дела *aes_reg_access_seq*

3.6.5.3 *aes_reset_seq*

Секвенца *aes_reset_seq* реализована је са циљем провере понашања *AES IP* у условима ресетовања током рада система. На почетку секвенце врши се упис *plaintext* и *key* вредности у одговарајуће регистре, након чега се активира процес енкрипције постављањем *start* бита.

Након случајно изабраног временског интервала, генерисаног помоћу функције *\$urandom_range*, активира се *reset* сигнал. *Reset* остаје активан током неколико тактова, а затим се деактивира и наставља се рад система.

По завршетку *reset* секвенце врши се читање *status* регистра, као и *plaintext*, *key* и *ciphertext* регистара, како би се проверило да ли су регистри исправно ресетовани и да ли је *DUT* наставио са коректним радом након *reset* догађаја.

Оваквим приступом омогућена је провера стабилности и исправног опоравка *AES IP* након *reset*. Након тога поново се покреће процес енкрипције са новим улазним подацима, чиме се проверава да ли је *DUT* успешно наставио нормалан рад након *reset* догађаја и да ли *AES IP* исправно функционише после опоравка система.

На Листингу 23 налази се имплементација *body()* дела *aes_reset_seq*.

```
task body();
    bit [31:0] rdata;
    super.body();

    for (int i=0; i<2; i++) begin
        this.randomize();

        // PLAINTEXT
        axi_write(PLAIN0_ADDR, plaintext[127:96], 4'hF);
        axi_write(PLAIN1_ADDR, plaintext[95:64], 4'hF);
        axi_write(PLAIN2_ADDR, plaintext[63:32], 4'hF);
        axi_write(PLAIN3_ADDR, plaintext[31:0], 4'hF);

        // KEY
        axi_write(KEY0_ADDR, key[127:96], 4'hF);
        axi_write(KEY1_ADDR, key[95:64], 4'hF);
        axi_write(KEY2_ADDR, key[63:32], 4'hF);
        axi_write(KEY3_ADDR, key[31:0], 4'hF);

        // START
        axi_write(START_ADDR, 32'h00000001, 4'hF);
        axi_write(START_ADDR, 32'h00000000, 4'hF);

        if(!i) begin
            wait_reset = $urandom_range(100,1000);
            repeat(wait_reset)
                @(posedge vif.aclk);

            vif.aresetn <= 0;
            repeat(5)
                @(posedge vif.aclk);

            vif.aresetn <= 1;

            `uvm_info("RESET", "Reset released", UVM_LOW)
        end
    end
endtask
```

Листинг 23: Имплементација *body()* taska *aes_reset_seq*

```

// READ REGISTERS AFTER RESET
do begin
    axi_read(READY_ADDR, rdata);
    end while (rdata[0] == 0);

    axi_read(PLAIN0_ADDR, rdata);
    axi_read(PLAIN1_ADDR, rdata);
    axi_read(PLAIN2_ADDR, rdata);
    axi_read(PLAIN3_ADDR, rdata);

    axi_read(KEY0_ADDR, rdata);
    axi_read(KEY1_ADDR, rdata);
    axi_read(KEY2_ADDR, rdata);
    axi_read(KEY3_ADDR, rdata);
end else begin
    do begin
        axi_read(READY_ADDR, rdata);
        end while (rdata[0] == 0);
    end
    axi_read(CIPHER0_ADDR, rdata);
    axi_read(CIPHER1_ADDR, rdata);
    axi_read(CIPHER2_ADDR, rdata);
    axi_read(CIPHER3_ADDR, rdata);
end
endtask
endclass

```

Листинг 24: Имплементација *body()* taska *aes_reset_seq* после ресета

3.6.5.4 *aes_invalid_access_seq*

Секвенца *aes_invalid_access_seq* реализована је са циљем провере понашања AES IP у случају недозвољених приступа током рада енкрипције. На почетку секвенце врши се упис *plaintext* и *key* вредности, након чега се активира процес енкрипције.

Током активне енкрипције секвенца покушава да пошаље нове *write* трансакције ка *plaintext* и *key* регистрима, чиме се симулира приступ регистрима у недозвољеном стању рада DUT-а. Поред тога, имплементирани су различити временски размаци између трансакција како би се проверило понашање система у различитим стањима FSM-а.

Након завршетка енкрипције врши се читање *status* и *ciphertext* регистара како би се проверило да ли је DUT задржао исправно понашање и игнорисао недозвољене приступе током рада.

На Листинзима 25 и 26 налази се имплементација *body()* дела *aes_invalid_access_seq*.

```

for(int num_of_iter = 0; num_of_iter < 10; num_of_iter++ ) begin
    this.randomize();

    // PLAINTEXT
    axi_write(PLAIN0_ADDR, plaintext[127:96], 4'hF);
    axi_write(PLAIN1_ADDR, plaintext[95:64], 4'hF);
    axi_write(PLAIN2_ADDR, plaintext[63:32], 4'hF);
    axi_write(PLAIN3_ADDR, plaintext[31:0], 4'hF);

    // KEY
    axi_write(KEY0_ADDR, key[127:96], 4'hF);
    axi_write(KEY1_ADDR, key[95:64], 4'hF);
    axi_write(KEY2_ADDR, key[63:32], 4'hF);
    axi_write(KEY3_ADDR, key[31:0], 4'hF);

    // START
    axi_write(START_ADDR, 32'h00000001, 4'hF);
    axi_write(START_ADDR, 32'h00000000, 4'hF);

    // send new transaction while encryption is ongoing
    if(num_of_iter) begin
        this.randomize();
        // PLAINTEXT-invalid
        axi_write(PLAIN0_ADDR, plaintext[127:96], 4'hF);
        axi_write(PLAIN1_ADDR, plaintext[95:64], 4'hF);
        axi_write(PLAIN2_ADDR, plaintext[63:32], 4'hF);
        axi_write(PLAIN3_ADDR, plaintext[31:0], 4'hF);

        // KEY-invalid
        axi_write(KEY0_ADDR, key[127:96], 4'hF);
        axi_write(KEY1_ADDR, key[95:64], 4'hF);
        axi_write(KEY2_ADDR, key[63:32], 4'hF);
        axi_write(KEY3_ADDR, key[31:0], 4'hF);
    end
end

```

Листинг 25: Имплементација body() дела aes_invalid_access_seq

```

case(num_of_iter)
    0: delay_cycles = 0;
    1: delay_cycles = $urandom_range(1,20); //add
    2: delay_cycles = 23; //startsub
    3: delay_cycles = $urandom_range(24,70); //subwait
    4: delay_cycles = 73; //shift
    5: delay_cycles = 74; // mix_start
    6: delay_cycles = $urandom_range(75,92); //mixCol
    7: delay_cycles = 93; //add_start
default:
    delay_cycles = $urandom_range(0, 90);
endcase

repeat(delay_cycles) begin
    @(posedge vif.clk);
end

// START-invalid
axi_write(START_ADDR, 32'h00000001, 4'hF);
axi_write(START_ADDR, 32'h00000000, 4'hF);

// POLLING
do begin
    axi_read(READY_ADDR, rdata);
end while (rdata[0] == 0);

// READ RESULT
axi_read(CIPHER0_ADDR, rdata);
axi_read(CIPHER1_ADDR, rdata);
axi_read(CIPHER2_ADDR, rdata);
axi_read(CIPHER3_ADDR, rdata);
end

```

task

Листинг 26: Имплементација body() дела aes_invalid_access_seq са невалидним стартом

3.7 Пронађени проблеми дизајна и њихово решење

1. Током верификације функционалности ресет механизма уочен је проблем у дизајну који се јавља приликом активирања ресета. Извршавањем одговарајуће секвенце утврђено је да након ресетовања система регистри намењени чувању резултата енкрипције (*ciphertext reg*) нису били ресетовани. Као последица тога, након завршетка ресет-а и накнадног читања садржаја регистра, у *ciphertext reg* су остајале вредности генерисане пре ресета. Проблем је отклоњен модификацијом дизајна тако што је ресет доведен до *mem_text_mapping* модула чиме су и регистри који садрже шифрат вредности обухваћени механизмом ресетовања. Поновљеним извршавањем тестова потврђено је да се након ресетовања сви *ciphertext reg* исправно чисте и да више не долази до појаве заосталих вредности из претходних операција енкрипције.
2. Такође уочена је и неусаглашеност са AXI4-Lite протоколом – Током верификације уочено је да AES IP подржава упис података само у случају када сигнали *write* адресе (*AW* канал) и *write* података (*W* канал) постану валидни истовремено. Међутим према AXI4-Lite спецификацији, адресни и канал за податке су међусобно независни и могу остварити *handshake* у различитим тактовима[7]. Имплементација не подржава сценарио у којем адреса и подаци стижу у истим тренуцима што одступа од пуне AXI4-Lite спецификације. Ово ограничење није утицало на функционалност у оквиру реализованих тестова, али га је потребно узети у обзир приликом интеграције IP језгра у сложеније системе.
3. Систем такође има ограничену подршку AXI4-Lite интерфејса. Анализом интерфејса уочено је да поједини AXI4-Lite сигнали нису коришћени у обради трансакција. Конкретно сигнали за одговор на упис и читање (*BRESP* и *RRESP*) не преносе информације о статусу извршене трансакције већ имају фиксну вредност. На тај начин нису подржани механизми сигнализације грешака предвиђене AXI4-Lite спецификацијом. Сигнали заштите приступа *AWPROT* и *ARPROT* се игноришу и немају утицај на понашање модула, без обзира на њихове вредности. Иако ово не утиче на основну функционалност AES IP језгра представља функционално ограничење интерфејса које би требало документовати ради правилне интеграције и употребе модула.

4. Закључак

У оквиру овог рада реализовано је верификационо окружење за AES-128 IP језгро са AXI4-Lite интерфејсом применом UVM методологије. Развијено окружење омогућило је систематичну проверу функционалности модула, генерисање различитих сценарија тестирања и анализу понашања система у условима који се могу јавити током реалног рада.

Током процеса верификације потврђена је исправност основне функционалности AES алгорита, укључујући упис улазних података и кључа, покретање енкрипције и читање добијеног шифрованог текста. Употребом референтног модела и функционалне покривености обезбеђена је провера исправности резултата за различите комбинације улазних вредности и сценарије рада.

Развијено верификационо окружење омогућило је откривање више проблема у имплементацији дизајна. Уочено је да регистри који чувају резултат енкрипције нису били обухваћени механизмом ресетовања, што је доводило до задржавања вредности из претходних операција. Након модификације дизајна и проширења логике ресетовања, проблем је успешно отклоњен. Поред тога, идентификована су и одступања од AXI4-Lite спецификације, пре свега ограничење да су адресни и канал за упис података морали бити валидни у истом такту како би трансакција била прихваћена. Такође је утврђено да поједини механизми предвиђени AXI4-Lite протоколом, као што су сигнали одговора и сигнали заштите трансакција, нису у потпуности искоришћени. Иако наведена ограничења нису утицала на основну функционалност AES језгра, представљају важне информације које је потребно документовати приликом интеграције модула у сложеније системе.

Посебан значај овог рада огледа се у потврди важности функционалне верификације као саставног дела развоја дигиталних система. Иако је AES IP језгро успешно реализовано са становишта функционалности, тек је кроз развијено UVM окружење било могуће открити граничне случајеве и имплементациона ограничења која нису била очигледна током самог развоја дизајна.

Резултати рада показују да је развијено верификационо окружење успешно испунило постављене циљеве и обезбедило висок степен поверења у исправност рада AES-128 IP језгра. Као могуће унапређење у будућем раду може се издвојити проширење подршке за комплетну AXI4-Lite спецификацију.

5. Литература

- [1] Daemen, J., & Rijmen, V. (2002). *The design of Rijndael* (Vol. 2). New York: Springer-verlag.
- [2] Bertaccini, M. (2022). *Cryptography Algorithms* (Vol. 358). Birmingham, UK: Packt Publishing.
- [3] Chu, P. P. (2006). *RTL hardware design using VHDL: coding for efficiency, portability, and scalability*. John Wiley & Sons.
- [4] Struharik, R. (n.d.). *Projektovanje složenih digitalnih sistema*. Fakultet tehničkih nauka. <https://www.elektronika.ftn.uns.ac.rs/projektovanje-slozenih-digitalnih-sistema/specifikacija/specifikacija-predmeta/>, posećeno 23.05.2026.
- [5] Grupa saradnika. (n.d.). *Funkcionalna verifikacija hardvera*. Fakultet tehničkih nauka. <https://www.elektronika.ftn.uns.ac.rs/funkcionalna-verifikacija-hardvera/specifikacija/specifikacija-predmeta/>, posećeno 25.05.2026.
- [6] Accellera Systems Initiative. (2015). *UVM 1.2 User's Guide*. https://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.2.pdf, posećeno 29.04.2026.
- [7] ARM Ltd. (2025). *AMBA Axi Protocol Specification (IHI 0022)*. <https://developer.arm.com/documentation/ih0022/>, posećeno 23.04.2026.
- [8] Bergeron, J. (2007). *Writing testbenches using SystemVerilog*. Springer Science & Business Media.