IMPLEMENTACIJA AES ALGORITMA OPTIMIZIRANA ZA FPGA

Gašperov, Duje

Graduate thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Split / Sveučilište u Splitu**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:228:292232

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-01-30**

Repository / Repozitorij:

Repository of University Department of Professional Studies

zir.nsk.hr

UNIVERSITY OF SPLIT

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Elektrotehnika

DUJE GAŠPEROV

Z A V R Š N I R A D

IMPLEMENTACIJA AES ALGORITMA OPTIMIZIRANA ZA FPGA

Split, rujan 2024.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Elektrotehnika

Predmet: Kriptografija s primjenom

Z A V R Š N I R A D

Kandidat: Duje Gašperov

Naslov rada: Implementacija AES algoritma optimizirana za FPGA

Mentor: Tonko Kovačević

Split, rujan 2024.

SADRŽAJ

Implementacija AES algoritma optimizirana za FPGA

Sažetak:

Projektni zadatak bio je dizajnirati digitalni logički sklop s ciljem implementacije na FPGA platformi. Rješenje ću izraditi u Vivado razvojnom okruženju koristeći Verilog HDL za Xilinx Zynq Ultrascale+ MPSoC. Testiranje i implementaciju ću odraditi na Xilinx ZCU102 razvojnoj ploči.

Ključne riječi: kriptografija, FPGA, AES, GCM, Verilog, Zynq.

FPGA-optimized AES algorithm implementation

Abstract:

The project task was to design a digital logic circuit with the aim of implementation on an FPGA platform. I will build the solution in Vivado development environment using Verilog HDL for Xilinx Zynq Ultrascale+ MPSoC. I will do the testing and implementation on a Xilinx ZCU102 development board.

Keywords: cryptography, FPGA, AES, GCM, Verilog, Zynq.

1. UVOD

Kriptografija je proces skrivanja odnosno kodiranja informacije na način da je može pročitati isključivo osoba kojoj je poruka poslana. Ova grana znanosti, poznata kao kriptologija, koristi različite discipline poput računalnih znanosti, inženjerstva, elektronike i digitalne obrade signala za stvaranje složenih kodova koji skrivaju pravo značenje poruke.

Prva poznata upotreba kriptografije bila je od strane Julija Cezara (100 g. pr. Kr. do 44 g. pr. Kr.) koji nije vjerovao svojim izaslanicima u prijenosu poruka. Zbog toga je stvorio sustav u kojemu je svaki znak u njegovim porukama zamijenjen znakom tri mjesta ispred njega u latinskom pismu.

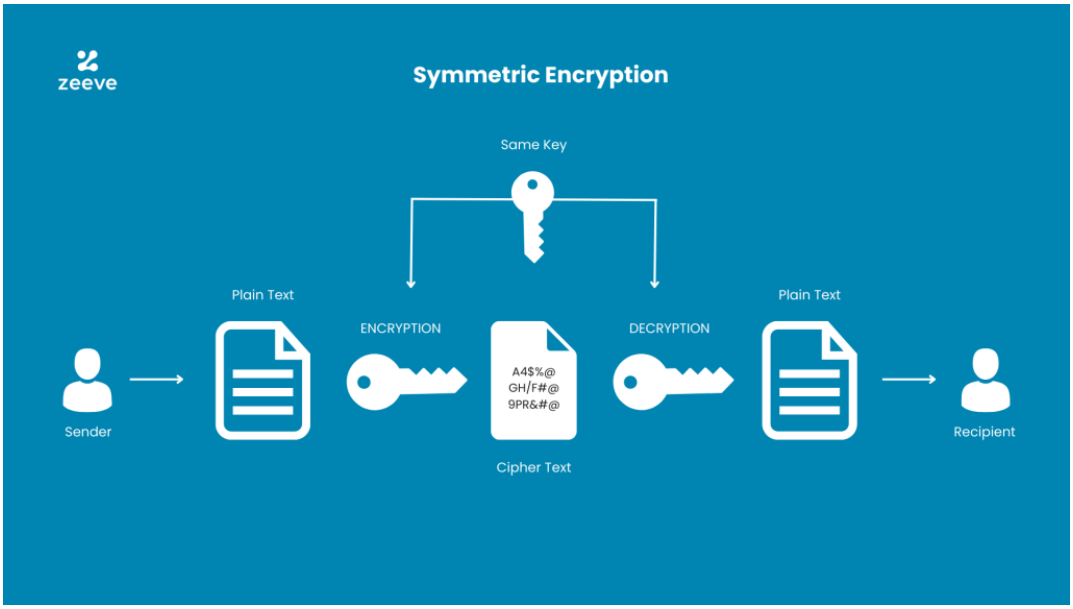
U novije vrijeme kriptografija je postala neizbježan element u pohrani i razmjeni informacija s obzirom da se većina razmjene i pohrane informacija događa preko interneta počevši od običnih aplikacija za razmjenu poruka do ogromnih podatkovnih centara, mobilnog i internet bankarstva te nacionalne sigurnosti. Budući da vlade ne žele da određeni subjekti u i izvan njihovih zemalja imaju pristup načinima primanja i slanja skrivenih informacija koje bi mogle biti prijetnja nacionalnim interesima, primjenjena su razna ograničenja u mnogim zemljama u rasponu od upotrebe do zabrane izvoza softvera za javno širenje matematičkih koncepata koji bi se mogli koristiti za razvoj kriptosustava.

2. KRIPTOGRAFIJA

Poruka koju pošiljatelj želi poslati u kriptografiji se naziva otvoreni tekst (eng. plaintext) koju će postupkom šifriranja (eng. encryption) pomoću dogovorenog ključa (eng. key) dobiti šifriranu poruku, odnosno šifrat ili kriptogram (eng. ciphertext). Šifrat se zatim šalje primatelju koji zna ključ pa može dešifrirati šifrat i na taj način doći do originalne poruke odnosno otvorenog teksta. Sve navedeno čini sustav koji funkcionira po strogo određenim pravilima definiranima kriptografskim algoritmom. Kriptografski algoritam je matematička funkcija koja se koristi za šifriranje i dešifriranje. Te funkcije preslikavaju elemente otvorenog teksta u šifrat i obrnuto. Osnovna podjela kriptosustava je na simetrične kriptosustave i asimetrične kriptosustave.

2.1. Simetrični kriptosustav

Simetrična kriptografija je poznata i kao kriptografija privatnog ključa. Zove se simetrična iz razloga što koristi isti ključ za šifriranje otvorenog teksta i dešifriranje šifrata. To jest, jedan ključ poznat kao tajni ključ dijeli se između pošiljatelja i primatelja. U usporedbi sa asimetričnom kriptografijom, simetrična kriptografija je brža zbog jednog tajnog ključa i jednostavnosti algoritma. Međutim, postoji veliki rizik ako napadač presretne tajni ključ pa može lako dešifrirati ili promijeniti sve sigurnosne poruke.



Slika 2.1. Simetrična kriptografija

Neki primjeri simetričnih sustava su Cezarova šifra i Vigenerova šifra koje su se najviše koristile u vojne i diplomatske svrhe. Razvojem računala kriptografija postaje zanimljiva većem broju korisnika te se pojavljuje potreba za uvođenjem standarda u kriptografiji. Tako je 1976. nastao DES (Dana Encryption Standard) koji šifrira otvoreni tekst (blokove) duljine 64 bita koristeći ključ K duljine 56 bitova. Na taj način dobiva se šifrat duljine 64 bita. Razbijanje DES – a dogodilo se 1998. godine zbog male duljine ključa te ga 2001. godine zamjenjuje AES (Advanced Encryption Standard) o kojem ćemo kasnije u radu detaljnije govoriti koji je danas najpoznatiji i najsigurniji standard za simetrično šifriranje. Ima blokove duljine 128 bita te ključeve veličine 128, 192 ili 256 bitova.

Simetrični kriptosustavi se danas koriste kao dio SSL odnosno TLS protokola koji omogućuju aplikacijama sigurnu komunikaciju preko internet mreže (autentifikacija poslužitelja, udaljeni pristup resursima, osiguravanje elektroničke pošte i slično).

2.1.1. Cezarov šifrat

Cezarov šifrat jedna je od najjednostavnijih i najpoznatijih tehnika šifriranja. To je vrsta supstitucijske šifre u kojoj se svako slovo otvorenog teksta zamijeni slovom pomaknutog 3 mjesta ispred u abecedi. Npr. riječ FAKULTET bit će zamijenjena s riječi IDNXOWHW.

Tekst	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Šifrat	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Tablica 2.1. Primjer Cezarovog šifrata sa pomakom 3 mjesta unaprijed

Ovo šifriranje možemo prikazati koristeći modularnu aritmetiku tako što ćemo prvo uvesti prirodnu korespondenciju između slova abecede (A - Z) i brojeva (0 - 25). Skup {0, 1, 2, …, 25} označavati ćemo s $\mathbb{Z}_{26}$ i predstavljati će nam skup ostataka pri dijeljenju s 26. Pretpostaviti ćemo da su na njemu definirane operacije zbrajanja, oduzimanja i množenja na način da računamo kao u skupu cijelih brojeva, ali tako da na kraju rezultat zamijenimo njegovim ostatkom pri dijeljenju s 26.

Definicija 1. *Neka je* $\alpha = \sigma = \kappa = \mathbb{Z}_{26}$. Za $0 \leq K \leq 25$ *definiramo* $e_K(x) = (x + K)\ mod\ 26$ i $d_K(y) = (y - K)\ mod\ 26$.

Šifra je definirana nad skupom $\mathbb{Z}_{26}$ jer promatramo skup od 26 slova abecede, a K je ključ i određujemo koliko slova udesno ćemo pomicati slova.

2.1.2. Vigenereov šifrat

Zbog svoje jednostavnosti Cezarova šifra je vrlo brzo razbijena. 1576. godine francuski kriptograf Blaise de Vigenere osmislio je šifru gdje se svaki znak u tekstu može preslikati u jedan od *n* mogućih znakova. Za šifriranje se koristi tablica abecede nazvana Vigenerov kvadrat ili Vigenereova tablica prikazanu na slici 2.2.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Slika 2.2. Vigenerova tablica

Iz slike 2.2. vidimo da Vigenerova tablica ima abecedu ispisanu u 26 redova na način da je u svakom retku abeceda ciklički pomaknuta ulijevo za jedno mjesto što odgovara 26 mogućih Cezarovih šifri.

Definicija 2. *Neka je n fiksan prirodan broj i* $\alpha = \sigma = \kappa = (\mathbb{Z}_{26})^n$. *Za ključ*

$K = (k_1, k_2, \dots, k_n)$ *definiramo:*

$e_K(x_1, x_2, \dots, x_n) = ((x_1 + k_1) mod\ 26, (x_2 + k_2) mod\ 26, \dots, (x_n + k_n) mod\ 26),$

$d_K(y_1, y_2, \dots, y_n) = ((y_1 - k_1) mod\ 26, (y_2 - k_2) mod\ 26, \dots, (y_n - k_n) mod\ 26).$

Slova otvorenog teksta pomičemo za k_1, k_2 ili k_n mjesta u ovisnosti o tome na kojem se mjestu nalaze u otvorenom tekstu.

Primjer 1. Odrediti ćemo šifrat poruke FAKULTET koristeći ključ UNIST.

Korak 1. Prikazati ćemo poruku i ključ ekvivalentnoj brojčanoj vrijednosti koristeći Vigenerovu tablicu,

P = {5, 0, 10, 20, 11, 19, 4, 19},

K = {20, 13, 8, 18, 19, 20, 13, 8}.

Korak 2. Prema formuli iz Definicije 2, računamo pomak šifrata

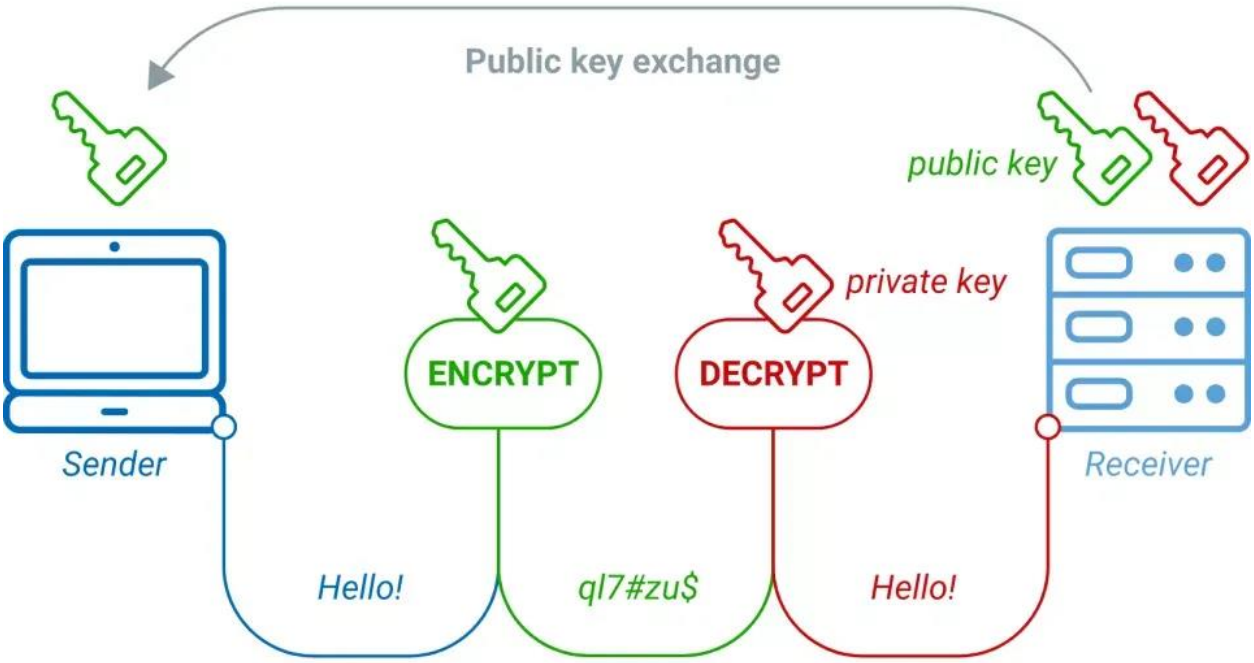
C = {25, 13, 18, 12, 4, 13, 17, 1}.

Korak 3. Koristeći Vigenerovu tablicu prikazati ćemo šifrat odgovarajućim slovima

C = ZNSMENRB.

2.2. Asimetrični kriptosustav

Za razliku od simetričnog kriptosustava, asimetrični kriptosustav koristi različite ključeve za šifriranje i dešifriranje koji se zovu javni i tajni ključ. Javni ključ kako mu samo ime govori je dostupan svima, a tajni ključ smije znati isključivo vlasnik ključa. Šifrat koji je šifriran javnim ključem može se jedino dešifrirati tajnim ključem i obrnuto, ako je šifrat šifriran tajnim ključem može se jedino dešifrirati javnim ključem.



Slika 2.3. Dijagram asimetrične kriptografije

Najveći nedostatak simetrične kriptografije je taj što primatelj i pošiljatelj moraju razmijeniti ključ preko nezaštićenog komunikacijskog kanala. 1976. godine Whitfield Diffie i Martin Hellman razvili su algoritam za razmjenu simetričnog ključa preko nezaštićenog komunikacijskog kanala koristeći asimetričnu kriptografiju. Njihov algoritam nazvan je „Diffie – Hellman key exchange“, a koraci su sljedeći:

1. Dva sudionika se na bilo koji način dogovore o dva velika broja *n* i *g* koji ne moraju biti tajni gdje je broj g relativno prost u odnosu na n, a najveći zajednički djelitelj im je broj 1. Najpraktičnije je za *n* odabrati veliki prosti broj *p*.
2. Nakon što su odabrali brojeve *p* i *g,* jedan sudionik odabere privatni nasumični broj *x*, a drugi y.
3. Prvi sudionik šalje drugom rezultat operacije
$$X = g^x (mod\ p)$$
4. Drugo sudionik šalje prvom rezultat operacije
$$Y = g^y (mod\ p)$$
5. Prvi sudionik tada izračunava ključ K:
$$K = Y^x (mod\ p) = (g^y)^x (mod\ p) = g^{xy} (mod\ p)$$

6. Analogno prvom, drugi sudionik izračunava:

$$K = X^y (mod\ p) = (g^x)^y (mod\ p) = g^{xy} (mod\ p)$$

7. Oba sudionika sada imaju isti simetrični ključ za šifriranje.

3. AES ENKRIPCIJSKI ALGORITAM

AES (eng. Advanced Encryption Standard) je enkripcijski algoritam prihvaćen od strane NIST – a (eng. National Institute of Standards and Technology) 2001. godine kao zamjena za DES koji je tada bio u upotrebi.

AES je simetrični algoritam te samim time koristi isti ključ za šifriranje i dešifriranje te radi sa blokovima podataka fiksne veličine od 128 bitova i podržava veličine ključeva od 128, 192 i 256 bitova. Sukladno tome danas postoje tri primarne vrste AES šifriranja:

- *AES – 128*: koristi ključ veličine 128 bitova i izvodi 10 krugova obrade tijekom šifriranja i dešifriranja. Ovaj algoritam je kompromis između sigurnosti i performansi, ali nije dovoljan za naprednije sigurnosne sustave
- *AES – 192*: koristi ključ veličine 192 bita i izvodi 12 krugova obrade tijekom šifriranja i dešifriranja. Nudi višu razinu sigurnosti u usporedbi sa AES – 128, s malim kompromisom u performansama
- *AES* – 256: koristi ključ veličine 256 bitova i izvodi 14 krugova obrade tijekom šifriranja i dešifriranja. Pruža najvišu razinu sigurnosti, ali s lošijim performansama među sve tri vrste, to jest, treba mu puno više vremena za šifriranje i dešifriranje istog bloka podataka nego prethodna dva.

3.1. Opis algoritma

Ulaz i izlaz iz AES algoritma je niz od 128 bitova. U daljnjem tekstu, ovaj niz zvati će se blok, a broj bitova njegova duljina. Ključ za šifriranje je niz duljine 128, 192 ili 256 bitova. Bitovi su numerirani na način da počinju od nule, a završavaju s duljinom niza umanjenim za jedan ($0 \leq i < 128$, $0 \leq i < 192$ ili $0 \leq i < 256$).

3.2. Okteti

Osnovna jedinica za obradu u AES algoritmu je oktet, niz od osam bitova koji se promatraju kao cjelina. Ulaz, izlaz i ključ obrađuju se kao niz okteta koje se formira dijeljenjem ovih blokova u grupe od osam bitova da bi formirali polje okteta. Za ulaz, izlaz ili ključ označen sa *a*, okteti u

rezultirajućem polju bit će označeni kao jedan od dva oblika, a_n ili a[n] gdje će n biti u sljedećem rasponu:

$$\text{duljina bloka} = 128, \quad 0 \leq n < 16;$$

$$\text{duljina ključa} = 128, \quad 0 \leq n < 16;$$

$$\text{duljina ključa} = 192, \quad 0 \leq n < 24;$$

$$\text{duljina ključa} = 256, \quad 0 \leq n < 32.$$

Sve vrijednosti okteta bit će predstavljene kao slaganje pojedinačnih vrijednosti bitova (0 ili 1) sljedećim redoslijedom:

$$\{b_7, \quad b_6, \quad b_5, \quad b_4, \quad b_3, \quad b_2, \quad b_1, \quad b_0\}$$

Ovi okteti su elementi konačnog polja i možemo ih prikazati kako polinom:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0 = \sum_{i=0}^{7} b_i x^i$$

Na primjer, {10010110} definira element konačnog polja $x^7 + x^4 + x^2 + x$.

Zbog velike duljine blokova, u većini slučajeva blokove prikazujemo u heksadecimalnom obliku kao što je prikazano u tablici 3.2.

Niz bitova	**Znak**	**Niz bitova**	**Znak**	**Niz bitova**	**Znak**	**Niz bitova**	**Znak**
0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

Tablica 3.2. Heksadecimalni zapis niza bitova

3.3. Matrica stanja

Sve operacije AES algoritma interno se provode na dvodimenzionalnom nizu okteta, odnosno matrici koju zovemo stanje. Matrica stanja sastoji se od četiri retka od kojih svaki sadrži Nb broj okteta gdje Nb označava duljinu bloka podijeljenu sa 32.

ULAZ

ul_0	ul_4	ul_8	ul_{12}
ul_1	ul_5	ul_9	ul_{13}
ul_2	ul_6	ul_{10}	ul_{14}
ul_3	ul_7	ul_{11}	ul_{15}

→

MATRICA STANJA

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

→

IZLAZ

izl_0	izl_4	izl_8	izl_{12}
izl_1	izl_5	izl_9	izl_{13}
izl_2	izl_6	izl_{10}	izl_{14}
izl_3	izl_7	izl_{11}	izl_{15}

Tablica 3.3. Ulaz, izlaz i matrica stanja

U matrici stanja svaki pojedini oktet ima dva indeksa, broj retka r ($0 \leq r < 4$) i broj stupca c ($0 \leq c < Nb$).

Ulazni blok podataka kopira se u matricu pomoću sljedeće formule:

$$s[r, c] = ul[r + 4c] \quad \text{za } 0 \leq r < 4 \text{ i } 0 \leq c < Nb.$$

Analogno ulaznom bloku, izlazni blok podataka kopira se u izlaznu matricu pomoću formule:

$$izl[r + 4c] = s[r, c] \quad \text{za } 0 \leq r < 4 \text{ i } 0 \leq c < Nb.$$

Ako pogledamo Tablicu 3.3, možemo primijetiti da svaki stupac matrice stanja sadržava četiri okteta od kojih možemo formirati 32 – bitnu riječ gdje broj retka r predstavlja indeks okteta u 32 – bitnoj riječi. Na taj način matricu stanja možemo promatrati kao jednodimenzionalno polje od 32 – bitnih riječi, $w_0, \dots, w_3$, gdje broj stupca c predstavlja indeks ovoga polja.

$$w_0 = s_{0,0}s_{1,0}s_{2,0}s_{3,0}$$

$$w_1 = s_{0,1}s_{1,1}s_{2,1}s_{3,1}$$

$$w_2 = s_{0,2}s_{1,2}s_{2,2}s_{3,2}$$

$$w_3 = s_{0,3}s_{1,3}s_{2,3}s_{3,3}$$

3.4. Matematički aparat

Svi okteti u AES algoritmu promatraju se kao elementi konačnog polja. Oni se mogu zbrajati i množiti ali ne kao što se to radi u algebarskoj matematici.

3.4.1. Zbrajanje

Operacija zbrajanja u konačnom polju postiže se tako da se zbroje koeficijenti elemenata odgovarajućih potencija u polinomima za ta dva elementa. Zbrajanje se izvodi XOR operacijom označenom sa $\oplus$ simbolom što je zapravo algebarsko zbrajanje na čiji rezultat primijenimo operaciju modulo 2. Prema tome, oduzimanje polinoma identično je zbrajanju.

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 0 = 1$$

$$1 \oplus 1 = 0$$

Analogno ovome, zbrajanje elemenata konačnog polja postiže se izvođenjem XOR operacije odgovarajućih bitova u oktetu.

Primjer:

$$(x^5 + x^4 + x^2 + x + 1) + (x^6 + x + 1) = x^6 + x^5 + x^4 + x^2$$

$$\{00110111\} \oplus \{01000011\} = \{01110100\}$$

$$\{0x37\} + \{0x43\} = \{0x74\}$$

3.4.2. Množenje

Množenje u Galoisovom polju GF(2^8) odgovara množenju polinoma (označeno s ·) te primjenom modulo operacije sa nedjeljivim polinomom osmog stupnja.

Definicija 3: *Polinom je nedjeljiv onda i samo onda ako je djeljiv s jedinicom i samim sobom*

Za AES algoritam ovaj nedjeljivi polinom je:

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Modularna redukcija za m(x) garantira nam da će rezultat biti binarni polinom stupnja manjeg od osam i stoga se može prikazati oktetom. Ovako definirano množenje je asocijativno, a element $\{01\}$ je multiplikativni identitet. Također se za svaki polinom b(x) stupnja manjeg od osam i različitog od nul – polinoma može izračunati multiplikativni inverz (označen s $b^{-1}(x)$) korištenjem proširenog Euklidovog algoritma za računanje polinoma a(x) i c(x) tako da je:

$$b(x)a(x) + m(x)c(x) = 1.$$

Stoga, $a(x) \cdot b(x)\ mod\ m(x) = 1$, što znači

$$b^{-1}(x) = a(x)\ mod\ m(x).$$

3.5. Algoritam

Šifriranje se izvodi tako da se ulaz kopira u matricu stanja kako je ranije opisano te se zatim dodaje podključ. Nakon toga, matrica stanja se transformira 10, 12 ili 14 puta ovisno o duljini ključa gdje se zadnji korak razlikuje od prethodno provedenih. Na kraju, izlaz se kopira u matricu stanja na način koji je opisan ranije.

Algoritam se sastoji od četiri individualne transformacije:

- zamjena okteta
- pomak redaka
- miješanje stupaca
- dodavanje podključa

Algoritam se može opisati sljedećim pseudokodom:

```
AES_Šifriranje(uint8_t ulaz, uint8_t izlaz, uint32_t w[Nb * (Nr + 1)])
{
    uint8_t matrica_stanja[4, Nb]
    matrica_stanja = ulaz
```

```
    DodajPodKljuc(matrica stanja, w[0, Nb - 1])
    za korak = 1; korak < Nr; korak++ {
        ZamjenaOkteta(matrica_stanja)
        PomakRedaka(matrica_stanja)
        MiješanjeStupaca(matrica_stanja)
        DodajPodKljuč(matrica stanja, w[korak*Nb, (korak +1)*Nb -1])
    }
    ZamjenaOkteta(matrica_stanja)
    PomakRedaka(matrica_stanja)
    DodajPodKljuč(matrica stanja, w[Nr*Nb, (Nr +1)*Nb -1])
    izlaz = matrica_stanja
}
```

3.5.1. Zamjena okteta

Zamjena okteta je nelinearna operacija koja se provodi neovisno za svaki oktet iz matrice stanja koristeći supstitucijsku tablicu. Supstitucijska tablica je invertibilna, sastavljena od dvije transformacije:

1. Uzimamo multiplikativni inverz u konačnom polju GF(2^8) gdje se element {00} preslikava sam na sebe.
2. Primjenjujemo sljedeću afinu transformaciju preko GF(2^8) polja:

   $$b_i = b_i \oplus b_{(i+4)\ mod\ 8} \oplus b_{(i+5)\ mod\ 8} \oplus b_{(i+6)\ mod\ 8} \oplus b_{(i+7)\ mod\ 8} \oplus c_i$$

   za $0 \leq i < 8$, gdje je b_i i – ti bit okteta, a c_i je i – ti bit okteta c vrijednosti {63} ili {01100011}.

U matričnom obliku afinu transformaciju možemo prikazati kao:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

x/y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Tablica 3.4. Supstitucijska tablica u heksadecimalnom obliku(xy)

3.5.2. Pomak redaka

Pomak redaka se provodi na način da se zadnja tri retka rotiraju ulijevo kako slijedi:

$$s'_{r,c} = s_{r,(c+pomak(r,Nb))\ mod\ Nb} \qquad \text{za } 0 < r < 4 \quad \text{i} \quad 0 \le c < Nb,$$

gdje vrijednost pomaka(r, Nb) ovisi o broju retka, r:

pomak(1, 4) = 1,

pomak(2, 4) = 2,

pomak(3, 4) = 3.

s			
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

pomak(1, 4)

pomak(2, 4)

pomak(3, 4)

s'			
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

3.5.3. Miješanje redaka

Miješanje redaka provodi transformacije nad svakim pojedinim stupcem u matrici stanja. Stupci se promatraju kao polinomi četvrtog reda koji se množe sa konstantnim polinomom a(x) te se rezultat reducira operacijom modulo $x^4 + 1$.

$$a(x) = 0x03\ x^3 + 0x01\ x^2 + 0x01\ x + 0x02$$

Matrični prikaz:

$$s'(x) = a(x) \oplus s(x)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Rezultat množenja dviju matrica:

$$s'_{0,c} = (0x02 \cdot s_{0,c}) \oplus (0x03 \cdot s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (0x02 \cdot s_{1,c}) \oplus (0x03 \cdot s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (0x02 \cdot s_{2,c}) \oplus (0x03 \cdot s_{3,c})$$

$$s'_{3,c} = (0x03 \cdot s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (0x02 \cdot s_{3,c})$$

3.5.4. Dodavanje podključa

Dodavanje podključa provodi se tako što jednostavno primijenimo XOR funkciju nad svim bitovima matrice stanja i podključa. Svaki podključ sastoji se od Nb 32 – bitnih riječi dobivenih proširenjem ključa. Dodavanje se provodi na sljedeći način:

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus [w_{korak*Nb+c}] \qquad \text{za } 0 \leq c < Nb$$

gdje su w_i riječi dobivene iz proširenja ključa, a $korak$ je korak algoritma i može imati vrijednost $0 \leq korak \leq Nr$. Prilikom šifriranja, inicijalno se dodaje podključ u prvom koraku (korak = 0) i zatim se poziva funkcija *dodaj_podkljuc()* za idućih Nr koraka algoritma.

3.5.4.1. Proširenje ključa

Za šifriranje AES algoritam koristi ključ K duljine 128, 192 ili 256 bitova te vrši njegovo proširenje da bi se dobili podključevi. Podključevi su duljine 32 bita i ukupno ih je Nb(Nr + 1). Rezultat proširenja ključa je linearni niz 32 – bitnih riječi označenih sa w[i] gdje je

$$0 \leq i < Nb(Nr + 1).$$

gdje, kako smo ranije već rekli, Nb duljina bloka podataka u 32 – bitnim riječima, a Nr broj koraka algoritma.

Pseudokod proširenja ključa:

*Proširenje ključa(uint8_t key[4*Nk], uint32_t w[Nb * (Nr + 1)], Nk) {*

uint32_t temp

```
    i = 0
    dok i < Nk {
        w[i] = (uint32_t)ključ[4*i] or (uint32_t)ključ[4*i + 1] or (uint32_t)ključ[4*i + 2]
        or (uint32_t)ključ[4*i + 3]
        i++
    }
    i = Nk
    dok i < Nb * (Nr + 1) {
        temp = w[i - 1]
        if( i mod Nk = 0)
            temp = zamijeni_riječ(rotiraj_riječ(temp)) xor R_konst[i/Nk]
        else if(Nk > 6 and i mod Nk = 4)
            temp = zamijeni_riječ(temp)
        w[i] = w[i - Nk] xor temp
        i++
    }
}
```

Funkcija *zamijeni_riječ* kao argument uzima 32 – bitnu riječ i koristi supstitucijsku tablicu za generiranje 32 – bitnog izlaza.

Funkcija *rotiraj_riječ* za argument uzima riječ oblika $a_0\ a_1\ a_2\ a_3$ i kao rezultat rotacije vraća riječ $a_1\ a_2\ a_3\ a_0$.

R_{konst} je konstantna vrijednost za svaki korak i ima vrijednost $\{02^{i-1}, 00, 00, 00\}$ gdje indeks i ima početnu vrijednost 1.

Iz ovog postupka vidimo da se Nk podključeva generira iz glavnog ključa za šifriranje. Svaka sljedeća riječ, w[i] dobije se izvođenjem XOR operacija prethodne riječi w[i – 1] i riječi za Nk pozicija prije w[i - Nk]. Za riječi na pozicijama koje su višekratnik Nk prvo se provodi transformacija na w[i – 1] prije izvođenja XOR operacije. Transformacija se sastoji od rotacije riječi koju slijedi zamjena riječi koristeći supstitucijsku tablicu te nakon toga XOR operacija sa $R_{konst}[i]$.

Moramo napomenuti da je proširenje ključa za 256-bitne ključeve (Nk = 8) malo drugačije nego za 128- i 192-bitne ključeve. Ako je Nk = 8 i i-4 je višekratnik Nk, tada se zamjena riječi primjenjuje na w[i-1] prije XOR operacije.

4. AES ENKRIPCIJSKI ALGORITAM U GCM NAČINU RADA

GCM(Galois Counter Mode) osigurava povjerljivost podatka korištenjem CTR (counter mode) način rada te nam garantira autentičnost i integritet podataka pomoću hash funkcije koja je definirana preko Galoisovog polja. GCM može otkriti:

- slučajne izmjene podataka
- namjerne i neovlaštene izmjene

Dvije funkcije GCM načina rada su autentificirano šifriranje i autentificirano dešifriranje.

Obje funkcije su vrlo efikasne i mogu se paralelizirati, kako u softveru tako i u hardveru.

Autentificirano šifriranje provodi šifriranje povjerljivih podataka i izračunava oznaku autentičnosti na svim povjerljivim i nepovjerljivim dodatnim podacima.

Autentificirano dešifriranje provodi dešifriranje povjerljivih podataka i provjeru oznake autentičnosti kako bismo bili sigurni da je očuvana autentičnost i integritet podataka.

4.1. Autentificirano šifriranje

4.1.1. Ulazni podaci

Autentificirano šifriranje koristi tri ulazna niza podataka:

- čisti tekst, označen sa P,
- dodatne autentificirane podatke, označene sa AAD,
- inicijalizacijski vektor, označen sa IV.

Ovaj način rada osigurava povjerljivost i autentifikaciju čistog teksta dok dodatni autentifikacijski podaci ostaju nezaštićeni ali autentificirani.

Inicijalizacijski vektor je jedinstvena vrijednost za svaki niz bloka podataka koja se koristi za šifriranje povjerljivih podataka što ćemo vidjeti kasnije u tekstu.

Duljine ulaznih podataka moraju zadovoljiti sljedeće uvjete:

- duljina(P) $\leq 2^{39} - 256$
- duljina(AAD) $\leq 2^{64} - 1$
- $1 \leq$ duljina(IV) $\leq 2^{64} - 1$

Implementacija može dodatno ograničiti duljine ovih podataka u skladu sa gore navedenim zahtjevima. Na primjer, preporučuje se da duljina inicijalizacijskog vektora bude 96 bita da bi se osigurala interoperabilnost, efikasnost i jednostavnost dizajna.

4.1.2. Izlazni podaci

Izlazni podaci iz funkcije autentificiranog šifriranja su:

- šifrat, označen sa C, čija je duljina jednaka duljini čistog teksta,
- autentifikacijska oznaka, označena sa T.

Duljina autentifikacijske oznake, označena sa *t*, je povjerljivi parametar i može poprimiti sljedeće vrijednosti: 128, 120, 112, 104 ili 96 bita.

4.2. Autentificirano dešifriranje

Ulazni podaci u funkciju autentificiranog šifriranja su:

- inicijalizacijski vektor(IV),
- dodatni autentificirani podaci(AAD),
- šifrat(C),
- autentifikacijska oznaka(T).

Izlazni podaci mogu biti čisti tekst ili greška u slučaju da je autentifikacijska oznaka neispravna. Duljine inicijalizacijskog vektora, šifrata i dodatnih autentificiranih podataka moraju biti jednake kao i kod funkcije šifriranja.

4.3. Matematičke operacije potrebne za GCM

4.3.1. Funkcija inkrementiranja

Za pozitivni cijeli broj *s* i niz bitova X duljine veće od *s*, definiramo funkciju inkrementiranja:

$$ink_s(X) = MSB_{duljina(X)-s}(X) \mid [int(LSB_X(X)) + 1 \; mod \; 2^s]_s$$

Ova funkcija inkrementira odnosno uvećava *s* krajnje desnih (najmanje značajnih) bitova modulo 2^s, dok krajnje lijevi (najviše značajni) bitovi ostaju nepromijenjeni.

4.3.2. Množenje niza podataka

Neka je R niz bitova vrijednosti *11100001* | 0^{120}, neka su X i Y ulazni nizovi podataka, X·Y izlazni niz podatka, definiramo funkciju množenja:

1. Neka su $x_0x_1 \cdots x_{127}$ bitovi niza X
2. Neka je $Z_0 = 0$ duljine 128 bitova
3. Neka je $V_0 = Y$
4. Za *i = 0 do 127* računamo Z_{i+1} i V_{i+1}:

$$Z_{i+1} = \begin{cases} Z_i & za \; x_i = 0; \\ Z_i \oplus V_i & za \; x_i = 1. \end{cases}$$

$$V_{i+1} = \begin{cases} V_i \gg 1 & za \; LSB_1(V_i) = 0; \\ (V_i \gg 1) \oplus R & za \; LSB_1(V_i) = 1. \end{cases}$$

5. Rezultat je Z_{128}.

4.3.3. GHASH algoritam

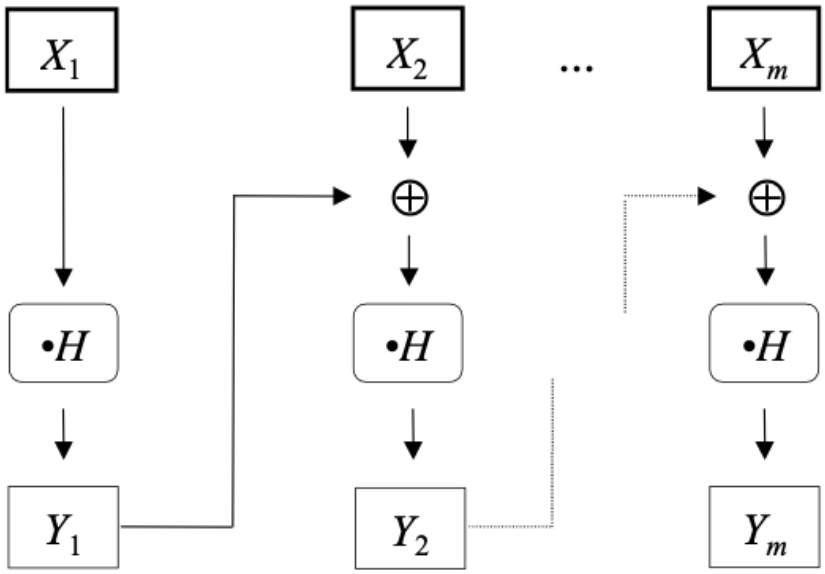
Preduvjet za GHASH algoritam je podključ H koji dobijemo šifriranjem niza duljine 128 bitova vrijednosti 0.

Ulazni podatak u funkciju je niz bitova X duljine jednake 128*m za neki pozitivan cijeli broj m, a izlaz iz funkcije je niz bitova označen sa Y.

Koraci:

1. Neka $X_1, X_2, \cdots, X_{m-1}, X_m$ definira niz blokova takav da je $X = X_1||X_2|| \cdots ||X_{m-1}||X_m$

2. Neka je Y_0 blok duljine 128 bitova vrijednosti 0.
3. Neka je $Y_i = (Y_{i-1} \oplus X_i) \cdot H$ za $i = 1, \dots, m$
4. Rezultat je Y_m.



Slika 4.4. GHASH algoritam

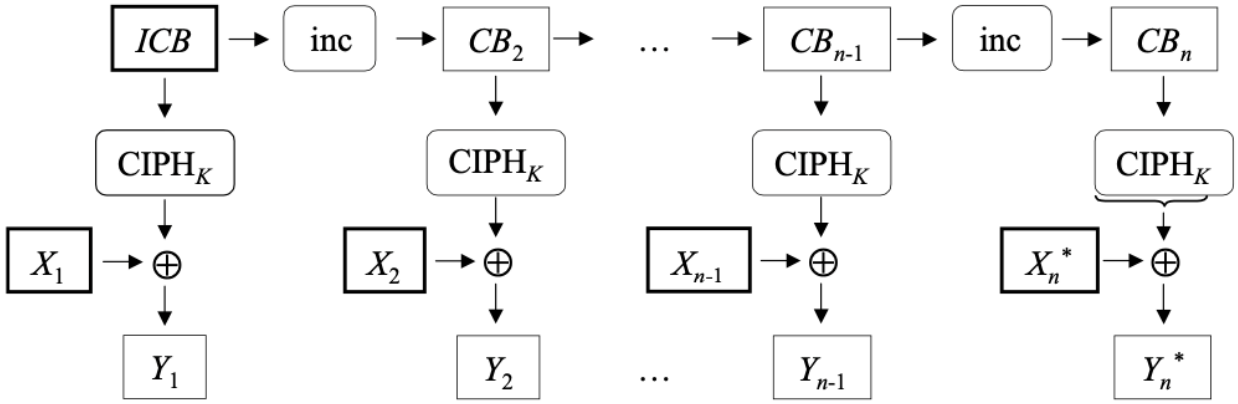
4.3.4. GCTR algoritam

Ulazni podaci za GCTR algoritam su inicijalni blok ICB, otvoreni tekst X i ključ K, a izlaz je šifrirani tekst duljine jednake kao i otvoreni tekst.

Koraci:

1. Neka je $n = duljina(X)/128$
2. Neka $X_1, X_2, \cdots, X_{n-1}, X_n^*$ definira niz blokova takav da je $X = X_1||X_2|| \cdots ||X_{n-1}||X_n^*$ $X_1, X_2, \cdots, X_{n-1}$ su blokovi duljine 128 bita, a X_n^* može biti i duljine manje od 128 bita
3. Neka je $CB_1 = ICB$
4. Neka je $CB_i = ink_{32}(CB_{i-1})$ za $2 \leq i \leq n$
5. Neka je $Y_n^* = X_n^* \oplus MSB_{duljina(X_n^*)}(CIPH_K(CB_n))$
6. Rezultat je $Y = Y_1||Y_2|| \dots ||Y_n^*$

U prvom i drugom koraku, ulazni niz proizvoljne duljine podijeljen je u najvećoj mogućoj mjeri na blokove potpune duljine, odnosno duljine 128 bita, tako da samo krajnji desni blok u nizu može biti duljine manje od 128 bita. U trećem i četvrtom koraku, inkrementiramo početni blok tako da dobijemo potreban niz blokova koje u petom i šestom koraku šifriramo AES algoritmom te primjenjujemo XOR funkciju s odgovarajućim blokovima otvorenog teksta i na kraju spajamo u izlaz Y.



Slika 4.5. AES algoritam u „counter“ načinu rada

4.3.5. Algoritam za autentificirano šifriranje

U ovaj algoritam ulazni podaci su:

- ključ za šifriranje K,
- otvoreni tekst P,
- inicijalizacijski vektor IV,
- dodatni autentificirani podaci AAD,

a izlazni podaci su:

- šifrat C,
- autentifikacijska oznaka T.

Postupak šifriranja:

1. Neka je $H = CIPH_K(0^{128})$.
2. Neka je $J_0 = IV||0^{31}||1$ ako je duljina(IV) = 96,

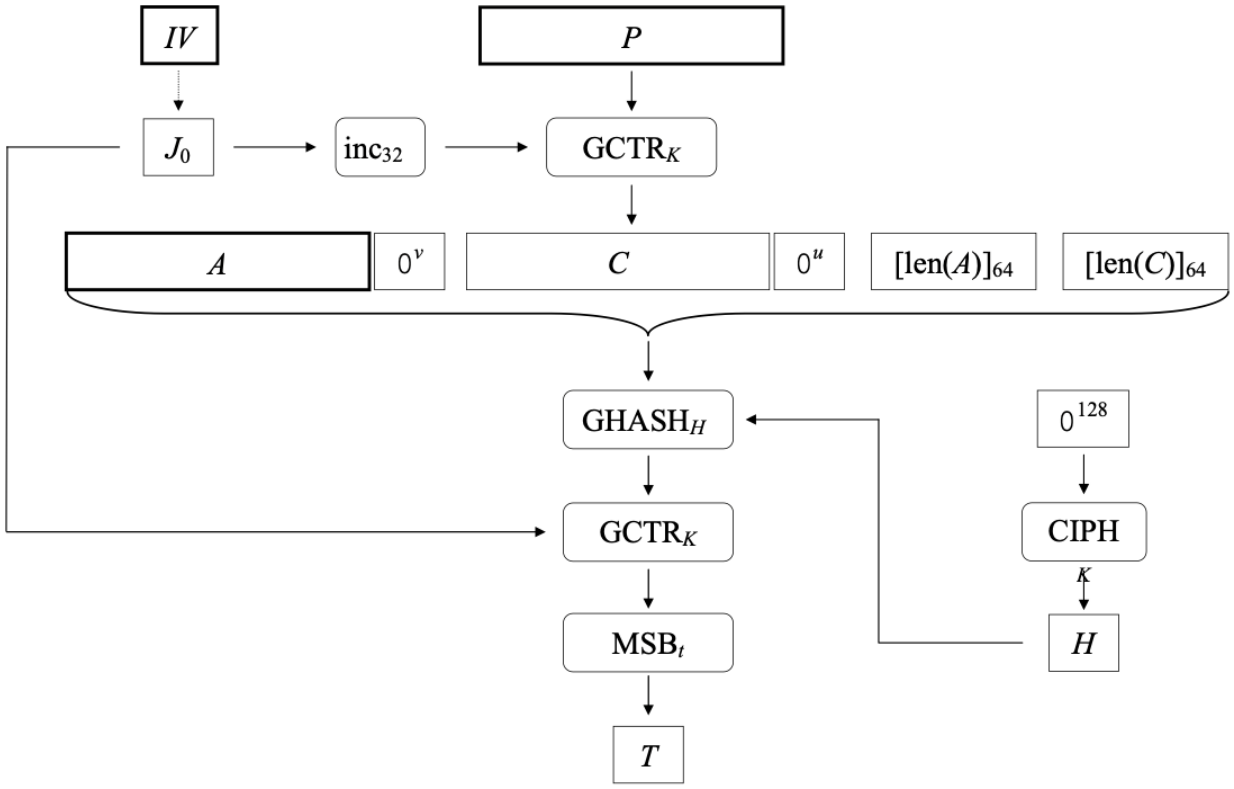
   $J_0 = \begin{cases} s = duljina\left(\frac{IV}{128}\right) - duljina(IV) \\ GHASH_H(IV|0^{s+64}|[len(IV)]_{64}) \end{cases}$ ako je duljina(IV) ≠ 96.
3. Neka je $C = GCTR_K(ink_{32}(J_0),\ P)$.
4. Neka je $u = 128 \cdot \left\lceil \frac{duljina(C)}{128} \right\rceil - duljina(C)$ i $v = 128 \cdot \left\lceil \frac{duljina(A)}{128} \right\rceil - duljina(A)$.
5. Definiramo blok S:

   $S = GHASH_H(A \,||\, 0^v \,||\, C \,||\, 0^u \,||\, [duljina(A)]_{64} \,||\, [duljina(C)]_{64})$.
6. Neka je $T = MSB_t(GCTR_K(J_0, S))$.
7. Rezultat je (C,T).

U prvom koraku računamo hash podključ tako da šifriramo niz duljine 128 bitova čija je vrijednost nula AES algoritmom nakon čega u drugom koraku generiramo inicijalni blok J_0. Konkretno, ako je duljina inicijalizacijskog vektora jednaka 96 tada inicijalizacijski vektor nadopunjujemo nizom bitova vrijednosti $0^{31}||1$ da bi dobili blok duljine 128 bitova. U suprotnom, inicijalizacijski vektor nadopunjujemo minimalnim brojem nula, po mogućnosti s niti jednom nulom, tako da je rezultirajući blok duljine 128 bita. Ovom bloku se dodaju još 64 bita vrijednosti 0 te 64 – bitna vrijednost duljine inicijalizacijskog vektora nakon čega se primjenjuje GHASH funkcija da bi dobili inicijalni blok J_0.

U trećem koraku primjenjujemo funkciju inkrementiranja inicijalnog bloka J_0 da bi generirali inicijalni blok koji će se koristiti u šifriranju otvorenog teksta CTR (counter mode) načinom rada (GCTR funkcija). Rezultat GCTR funkcije je šifrat.

U koraku 4 i 5, šifrat nadopunjujemo minimalnim brojem nula, po mogućnosti niti s jednom, na način da je rezultirajući niz višekratnik duljine bloka (128 bita). Nakon ulančavanja ovih nizova, dodatno ga nadopunjujemo 64 – bitnim vrijednostima duljine dodatnih autentificiranih podataka te primjenjujemo funkciju GHASH kako bi dobili jedan izlazni blok. Ovaj izlazni blok je šifriran GCTR funkcijom sa inicijalnim blokom generiranim u koraku 2 te se rezultat skraćuje na definiranu duljinu autentifikacijske oznake da bi za rezultat dobili autentifikacijsku oznaku.



Slika 4.6. AES algoritam u GCM načinu rada za autentifikacijsko šifriranje

4.3.6. Algoritam za autentifikacijsko dešifriranje

Ulazni podaci:

- ključ za dešifriranje K,
- šifrat C,
- inicijalizacijski vektor IV,
- dodatni autentificirani podaci AAD,
- autentifikacijska oznaka T.

Izlazni podaci:

- otvoreni tekst P ili greška u slučaju nepodudaranja autentifikacijske oznake

Postupak:

1. Neka je $H = CIPH_K(0^{128})$.
2. Neka je $J_0 = IV||0^{31}||1$ ako je duljina(IV) = 96,

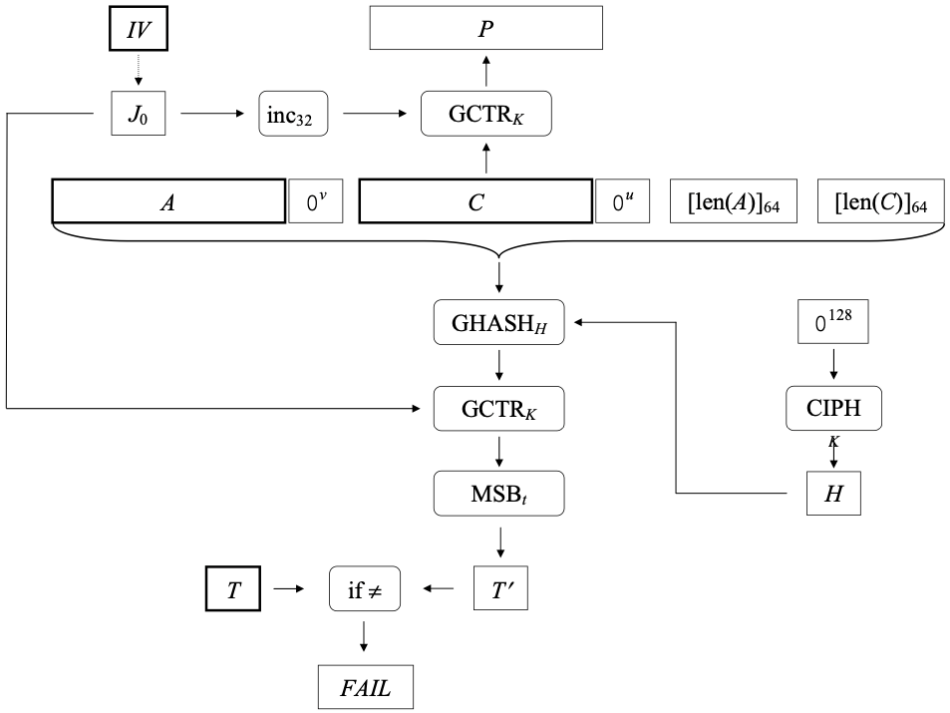
$$J_0 = \begin{cases} s = duljina\left(\frac{IV}{128}\right) - duljina(IV) \\ GHASH_H(IV|0^{s+64}|[len(IV)]_{64}) \end{cases}$$ ako je duljina(IV) ≠ 96.

3. Neka je $P = GCTR_K(ink_{32}(J_0),\ C)$.
4. Neka su: $u = 128 \cdot \left\lceil \frac{duljina(C)}{128} \right\rceil - duljina(C)$

   $v = 128 \cdot \left\lceil \frac{duljina(A)}{128} \right\rceil - duljina(A)$.
5. Definiramo blok S:

   $S = GHASH_H(A \,||\, 0^v \,||\, C \,||\, 0^u \,||\, [duljina(A)]_{64} \,||\, [duljina(C)]_{64})$.
6. Neka je $T' = MSB_t(GCTR_K(J_0, S))$.
7. Ako je $T = T'$ rezultat je otvoreni tekst P, u suprotnom je rezultat greška.

Koraci 1 i 2 identični su kao i kod autentifikacijskog šifriranja. U trećem koraku primjenjujemo funkciju inkrementiranja inicijalnog bloka J_0 da bi generirali inicijalni blok koji će se koristiti u dešifriranju šifrata CTR (counter mode) načinom rada (GCTR funkcija). Rezultat GCTR funkcije je otvoreni tekst. Korake 5 i 6 primjenjujemo kao i kod autentifikacijskog šifriranja. U koraku 7 uspoređujemo izračunatu autentifikacijsku oznaku sa onom iz ulaznih podataka i ako su jednake kao rezultat vraćamo otvoreni tekst, u suprotnom je rezultat greška.

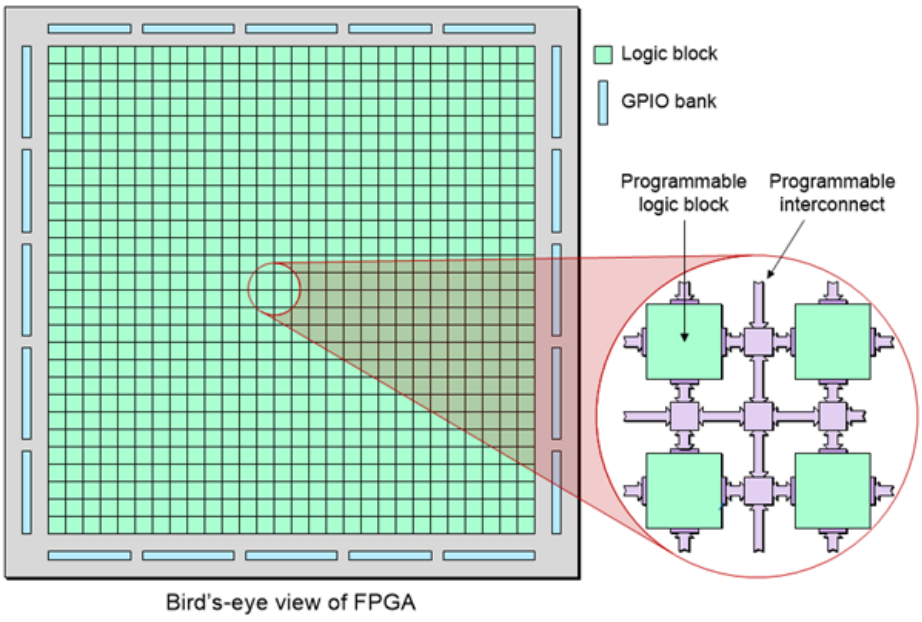


Slika 4.7. AES algoritam u GCM načinu rada za autentifikacijsko dešifriranje

5. FPGA (Field Programmable Gate Array)

FPGA je digitalni sklop koji je, za razliku od logičkih sklopova kao što je ASIC, dizajniran da se može programirati i po potrebi reprogramirati kako bi odgovarao različitim namjenama, posebno brzim izradama prototipa i izradama visoko – efikasnim aplikacijama.

Prvi FPGA proizvela je tvrtka Xilinx 1985. godine. Izraz „field programmable“ odnosi se na sposobnost mijenjanja dizajna nakon što je čip proizveden što nam daje mogućnost „programiranja hardvera“. FPGA se može prilagoditi tako da odgovara raznovrsnim slučajevima upotrebe, uključujući nove ili eksperimentalne svrhe, bez potrebe za fizičkim modificiranjem ili mijenjanjem njihovog hardvera. Ova mogućnost rekonfiguriranja postiže se nizom fiksnih programabilnih logičkih blokova i fleksibilnih interkonekcija koje se mogu konfigurirati za izvođenje složenih operacija ili da služe kao jednostavna logička vrata. FPGA također uključuju memorijske elemente, u rasponu od jednobitnih flip – flopova do vrlo gustih memorijskih nizova, koji služe za pohranu podataka unutar uređaja.



Slika 5.8. Struktura FPGA čipa

FPGA su visoko cijenjeni zbog svoje kombinacije visokih performansi i ekstremne svestranosti. Posebno su korisni u aplikacijama koje zahtijevaju visoke performanse, nisku latenciju i fleksibilnost u stvarnom vremenu. Zbog toga se često koriste u telekomunikacijskoj, automobilskoj i zrakoplovnoj industriji.

5.1. Princip rada FPGA

Specifične funkcije temeljene na FPGA, kao i međusobne veze između tih funkcija, "opisane su" u jeziku za opis hardvera (HDL - Hardware Description Language). Opis je sastavljen za izradu FPGA konfiguracijske datoteke (eng. bitstream). Koristeći jezik za opis hardvera, moguće je koristiti ugrađene FPGA resurse kao na primjer memorijske blokove ili razna sučelja kao i stvoriti prilagođene logičke sklopove (zbrajače, multipleksere i druge funkcije specifične za aplikaciju) iz primitivnijih FPGA elemenata.

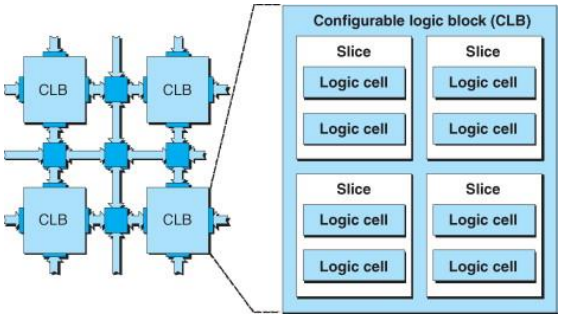
U procesu poznatom kao sinteza, HDL kod se prevodi u popis linija(vodiča), učinkovit opis logičkih vrata i interkonekcija potrebnih za implementaciju HDL koda. Popis linija se zatim preslikava na programabilne logičke blokove i interkonekcije koji fizički tvore jedinstveni krug.

Moderni FPGA nude širok raspon digitalnih i analognih značajki, uključujući iznimno veliku gustoću logičkih blokova, flash memoriju, ugrađene procesore i blokove za digitalnu obradu signala (DSP). FPGA se mogu konfigurirati i rekonfigurirati modificiranjem električnih ulaza i izlaza (GPIO), odabirom internih resursa koji se koriste i određivanjem načina na koji su ti resursi povezani kroz konfigurabilne resurse usmjeravanja. Krajnji rezultat je namjensko hardversko rješenje za rješavanje određenog problema.

Kao i kod softvera, razvoj složenih FPGA dizajna mogao bi se pojednostaviti korištenjem unaprijed dizajniranih biblioteka raznih funkcija i digitalnih sklopova, koje se nazivaju jezgre intelektualnog vlasništva (IP). Ove su jezgre dostupne za kupnju ili iznajmljivanje od FPGA dobavljača i dobavljača trećih strana koji su često specijalizirani za razvoj različitih funkcija.

5.2. Podesivi logički blokovi

Podesivi logički blokovi (CLB) su primarna komponenta FPGA. Podesivi logički blokovi općenito sadrže nekoliko primitivnih logičkih elemenata kao što su odsječci (eng. slices), tablice za pretraživanje (eng. lookup tables), flip – flopovi te multiplekseri.



Slika 5.9. FPGA CLBs

5.2.1. Tablica za pretraživanje

Tablica za pretraživanje (eng. lookup table) služi za ubrzavanje obrade podataka, tako da postavi izlaz za dati ulaz bez potrebe za stalnim računanjem na ulaznim podacima. To se postiže spremanjem vrijednosti izlaza na određenu memorijsku lokaciju koju ćemo odrediti ulaznim podacima.

5.2.2. Multiplekser

Multipleksiranje unutar jednog odsječka (eng. slice) obrađuje se na način da umjesto velikog broja namjenskih multipleksera s fiksnim ulazima koristi fleksibilnost LUT – ova. Svaki LUT može se implementirati kao 4:1 multiplekser koristeći dva od šest ulaza kao odabire za ostala četiri. Na ovaj način mogu se stvoriti puno veći multiplekseri.

5.2.3. Flip – flop

Flip – flop je osnovni sklop za pohranu podataka koji može pohraniti jedan bit informacije.

Flip – flopovi se dijele na sinkrone i asinkrone. Asinkroni flip – flop ima svojstvo da mijenja izlazni signal čim se promijeni ulazni signal što može dovesti do pogrešnog rezultata ako ulazni signali nisu sinkronizirani. Sinkroni flip – flop je verzija asinkronog s dodatnim signalom koji zovemo takt (eng. clock) te se izlazni signali mijenjaju samo na promjenu takta.

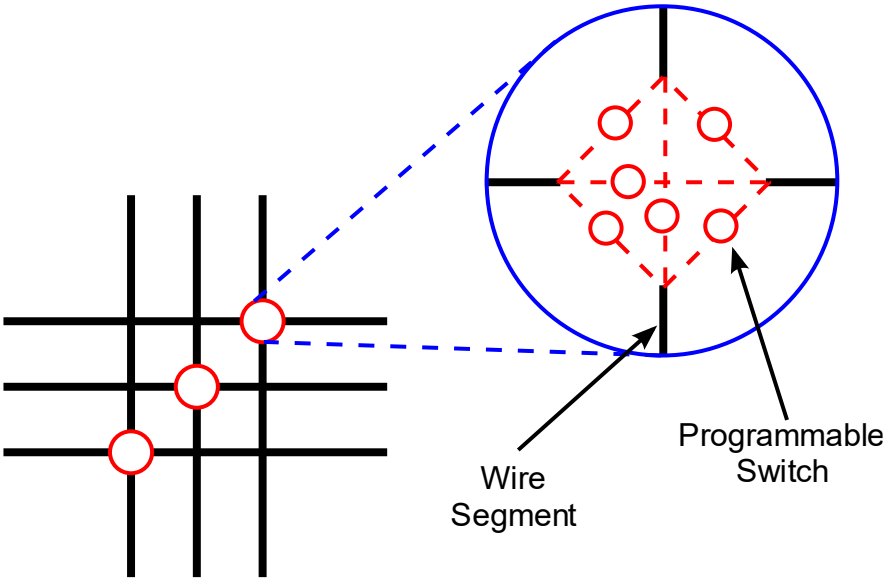
5.2.4. FPGA odsječak

FPGA odsječak (eng. slice) čini osnovnu jedinicu CLB – a. Svaki odsječak sastoji se od 4 LUT – a, 4 flip – flopa, multipleksera te prijenosnog lanca. Neki odsječci također sadrže distribuirani RAM te 32 – bitne pomačne registre.

U razvojnom alatu za Xilinx FPGA, položaj presjeka je predstavljen sa X_mY_n gdje je m horiznontalna koordinata odsječka, a n okomita koordinata odsječka.

5.3. Programabilna interkonekcija

Veze konstruirane od žičanih segmenata spojenih električnim programabilnim sklopkama osiguravaju putanje usmjeravanja između logičkog bloka FPGA. Usmjerivačke matrice koje sadrže višestruke osnovne poluvodičke sklopke obično se koriste za uspostavu programibilne interkonekcije FPGA. Ove interkonekcije omogućuju da se izlazi određene jedinice ili ulazne pločice povežu s bilo kojom drugom ćelijom ili pločicom unutar kruga.



Slika 5.10. FPGA programabilna interkonekcija

5.4. Programabilno usmjeravanje

Ugrađeni žičani segmenti spojeni su zajedno (ili ostavljeni nepovezani) programabilnim interkonekcijama kako bi se dobila potpuno programabilna infrastruktura usmjeravanja unutar FPGA. Resursi za usmjeravanje hijerarhijski su po prirodi, s kombinacijom dugih, srednjih i kratkih žica koje obuhvaćaju različite "duljine" unutar FPGA. Ova programabilna infrastruktura usmjeravanja, omogućena programabilnim interkonekcijama, omogućuje korisnicima da konfiguriraju FPGA resurse za implementaciju određenog zadatka ili aplikacije.

5.5. Programabilni ulazno – izlazni blokovi

Sučelje između FPGA i drugih vanjskih uređaja omogućeno je ulazno - izlaznim (I/O) blokovima (IOB). IOB-ovi su programibilni ulazni i izlazni resursi koji su konfigurirani da odgovaraju protokolima bilo kojeg vanjskog uređaja na koji se povezuje FPGA. Svi signali koji ulaze ili izlaze iz FPGA čine to preko pinova uređaja i povezanih IOB-ova.

5.6. Memorija na čipu

Najraniji FPGA uređaji koristili su samo flip-flopove (FF) za integriranje memorije u FPGA logičke blokove. Međutim, kako su se mogućnosti FPGA povećavale, sve složeniji dizajni zahtijevali su namjensku memoriju na čipu za međuspremnik podataka i ponovnu upotrebu. Moderni FPGA koriste velike SRAM memorijske nizove, manje tablice pretraživanja (LUT) i tradicionalne flip-flop elemente kako bi osigurali potrebnu pohranu za određenu aplikaciju.

5.7. Blokovi za digitalnu obradu signala (DSP)

U ranim FPGA-ima, jedini dostupni aritmetički resursi bili su jednostavni zbrajači, sve složenije bilo je izgrađeno od primitivnijih logičkih elemenata. Međutim, kako je tehnologija silicija napredovala, složeniji aritmetički resursi su ugrađeni u FPGA, što je kulminiralo modernom FPGA DSP bloku. DSP blokovi pružaju visoko optimizirane resurse kao na primjer množitelje za implementaciju aritmetičkih funkcija visokih performansi. Njihova upotreba eliminira potrebu za implementacijom ovih funkcija u CLB-ove opće namjene, čime se CLB-ovi oslobađaju za druge svrhe. Operacije kao što su digitalno filtriranje, konvolucija, Fourierove transformacije, trigonometrijske operacije i mnoge druge mogu iskoristiti ove resurse za postizanje performansi u stvarnom vremenu u aplikacijama koje se kreću od radarske obrade, oblikovanja snopa, prepoznavanja uzoraka i mnogih drugih.

5.8. Jezik za opis hardvera (HDL)

Dizajn jezika za opis hardvera (HDL) temelji se na stvaranju i korištenju tekstualnih opisa digitalnog logičkog sklopa ili sustava. Korištenjem određenog HDL-a (dva IEEE standarda u uobičajenoj uporabi u industriji i akademskoj zajednici su Verilog i VHDL), opis sklopa može se stvoriti na različitim razinama apstrakcije od osnovnih logičkih vrata opis prema jezičnoj sintaksi (gramatički raspored riječi i simbola koji se koriste u jeziku) i semantici (značenje riječi i simbola koji se koriste u jeziku).

Hardverski sklopovi ili dizajn sustava stvoreni pomoću HDL-a generiraju se na različitim razinama apstrakcije. Počevši od najviše razine, ideja ili koncept sustava početni je opis dizajna na visokoj razini koji daje specifikaciju dizajna. Algoritam opisuje ponašanje dizajna u matematičkim

terminima. Niti ideja sustava niti algoritam ne opisuju kako se ponašanje dizajna treba implementirati. Struktura algoritma u hardveru opisana je arhitekturom, koja određuje funkcionalne blokove visoke razine za korištenje i način na koji su funkcije povezane. Razine algoritma i arhitekture opisuju ponašanje dizajna koje se kasnije provjeri u simulaciji.

Niža razina od arhitekture je razina prijenosa registara (RTL), koja opisuje pohranu (u registrima) i protok podataka u dizajnu, zajedno s logičkim operacijama koje se izvode na podacima. Ovu razinu obično koriste alati za sintezu koji opisuju strukturu dizajna (popis mreža dizajna u smislu logičkih vrata i međusobnog ožičenja između logičkih vrata). Sama logička vrata su implementirana pomoću tranzistora.

5.8.1.1. Verilog HDL

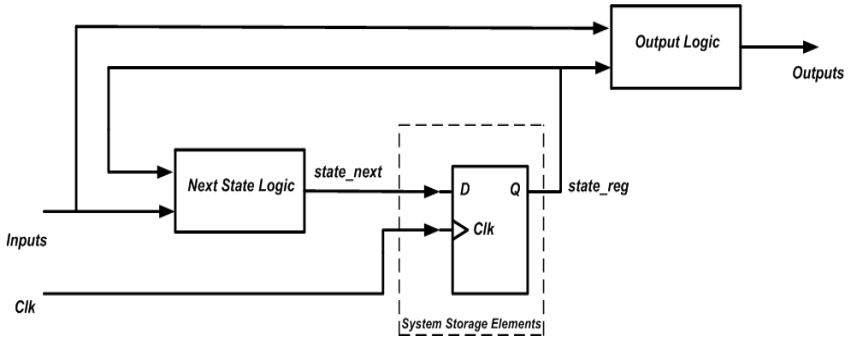
Verilog je jezik za opis hardvera (HDL) koji se koristi za opisivanje digitalnih sustava i sklopova u obliku koda. Razvio ga je Gateway Design Automation sredinom 1980-ih, a kasnije ga je preuzeo Cadence Design Systems.

Verilog se naširoko koristi za dizajn i provjeru digitalnih i mješovitih logičkih sklopova, uključujući integrirane sklopove specifične za primjenu (ASIC-ove) i FPGA. Podržava niz razina apstrakcije, a koristi se i za dizajn temeljen na simulaciji i dizajn temeljen na sintezi.

Jezik se koristi za hijerarhijski opisati digitalni sklop, počevši od najosnovnijih elemenata kao što su logička vrata i flip-flopovi do složenijih funkcionalnih blokova i sustava. Također podržava niz tehnika modeliranja, uključujući modeliranje na razini vrata, RTL razini i modeliranje na razini funkcionalnosti.

Korištenjem Verilog HDL – a digitalne logičke sklopove možemo opisati sekvencijalnom logikom i kombinatornom logikom.

5.8.1.2. Sekvencijalna logika



Sekvencijalni logički sklop znači da izlaz kruga ne ovisi samo o trenutnom ulazu, već i o prethodnim ulazima koje možemo spremiti u memoriju pomoću registara ili flip – flopova.

Slika 5.11. Digitalna sekvencijalna logika

Primjer:

```
module Brojac (
    input clk,
    input reset,
    input smjer,
    output [7:0] rezultat
)
reg [7:0] brojac = 8'd0, sljedeci_brojac;
assign rezultat = brojac;
always @* begin
    if(smjer)       sljedeci_brojac = brojac + 1;
    else            sljedeci_brojac = brojac - 1;
end
always @(posedge clk) begin
    if(reset)       brojac <= 8'd0;
    else            brojac <= sljedeci_brojac;
end
endmodule
```

5.8.1.3. Kombinatorna logika

Kombinacijski logički sklop znači da izlaz sklopa ovisi samo o trenutnom ulazu. Prethodni ulaz ne može promijeniti trenutni izlaz što znači da u ovom krugu nema memorijske jedinice.

Primjer:

```
module xor_vrata (
    input a,
    input b,
    output rezultat
)
assign rezultat = a ^ b;
endmodule
```

5.8.1.4. Blokirajuća i ne blokirajuća logika

Glavni razlog korištenja ova dva načina je za opisati sekvencijalnu ili kombinatornu logiku. U softveru se linije koda izvršavaju jedna nakon druge, odnosno iduća linija koda će se izvršiti tek nakon što prethodna završi. U HDL – u linije koda mogu se izvršavati paralelno korištenjem operatora <=, ili kao u softveru jedna nakon druge korištenjem operatora =.

Primjer:

```
always @(posedge clk) begin
    test1 <= 1'b1;
    test2 <= test1;
    test3 <= test2;
end
```

U gornjem primjeru koristili smo neblokirajući operator što znači da nam trebaju 3 takta da bi se vrijednost 1'b1 propagirala do registra test3. Važno je napomenuti da se sve tri linije koda poviše, kao što smo rekli ranije, izvršavaju potpuno paralelno što je jedna od najvećih prednosti FPGA koja nam osigurava visoku propusnost i nisku latenciju.

U niže navedenom primjeru koristimo blokirajući operator što znači da se vrijednost 1'b1 odmah propagira do registra test3. Blokirajći operator odmah pridružuje vrijednost s desne strane lijevoj strani.

Primjer:

```
always @* begin
    test1 = 1'b1;
    test2 = test1;
    test3 = test2;
end
```

Praktično pravilo je da se nikad ne miješaju ova dva operatora u istom always bloku te da neblokirajući operator koristimo u sekvencijalnoj logici, a blokirajući u kombinatornoj logici.

5.9. FPGA ILI CPU

FPGA i CPU imaju različite karakteristike koje ih čine prikladnima za različite primjene. Što se tiče performansi, FPGA se ističu u zadacima koji zahtijevaju masivnu paralelnu obradu, omogućujući značajno smanjenje latencije i povećanje propusnosti, što je ključno za aplikacije kao što su obrada signala, znanstveno računarstvo, kriptografiju i slično. CPU su, s druge strane, dizajnirani za sekvencijalno izvršavanje instrukcija i optimizirani su za širok raspon zadataka opće namjene, nudeći veliku svestranost.

Kada je u pitanju fleksibilnost, FPGA nadmašuju CPU zahvaljujući svojoj sposobnosti reprogramiranja, omogućujući inženjerima rekonfiguraciju hardvera prema specifičnim potrebama svake aplikacije. Ova prilagodljivost je osobito korisna u okruženjima u kojima se zahtjevi mogu brzo mijenjati. CPU – ovi, imaju koristi od velike softverske kompatibilnosti i jednostavnosti programiranja, što ih čini pristupačnijim programerima.

Što se tiče potrošnje energije, FPGA mogu biti učinkovitiji za specifične zadatke, budući da omogućuju prilagodbu arhitekture koja optimizira korištenje resursa. CPU – ovi, unatoč njihovom

napretku u smislu energetske učinkovitosti, često su manje učinkoviti u slučajevima gdje je potrebna napredna optimizacija hardvera.

Naposljetku, u pogledu troškova, FPGA mogu uključivati veća početna ulaganja zbog svoje složenosti dizajna i programiranja, ali mogu ponuditi dugoročne uštede eliminirajući potrebu za višestrukim hardverskim komponentama i omogućujući nadogradnju hardvera jednostavnim reprogramiranjem. CPU – ovi, s općenito nižim troškovima proizvodnje i postavljanja, ostaju isplativo rješenje za velike aplikacije opće namjene.

5.9.1.1. Prednosti FPGA u odnosu na CPU

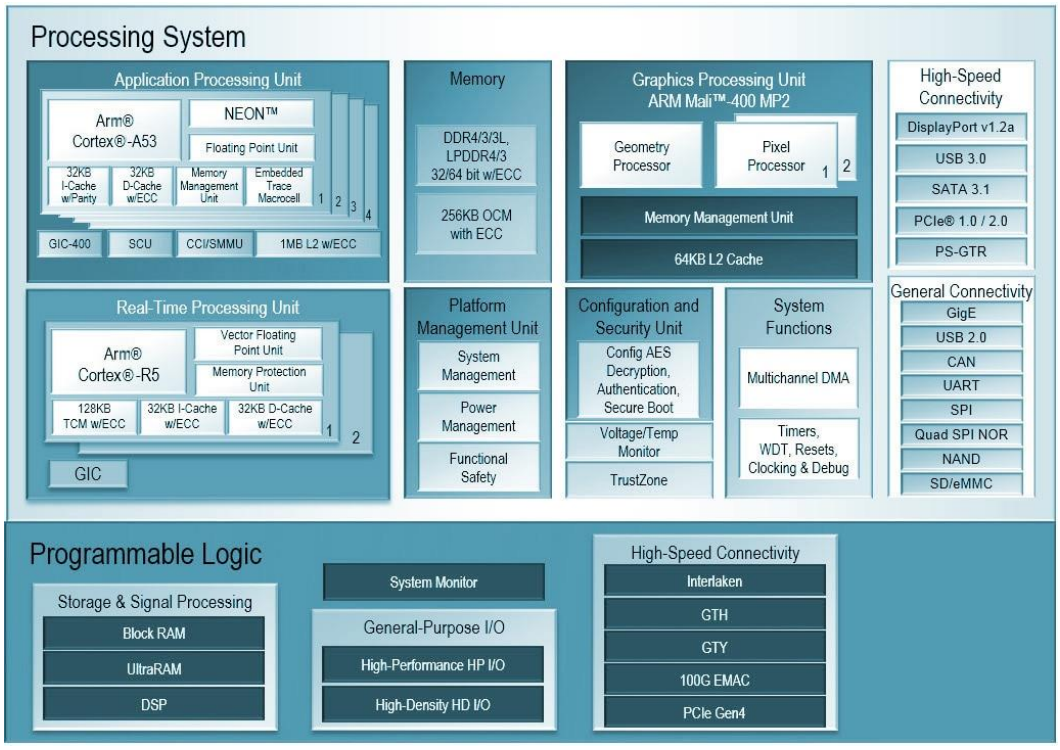
FPGA nudi nekoliko prednosti u odnosu na CPU, osobito u pogledu performansi i fleksibilnosti. Prvo, FPGA se mogu optimizirati za obavljanje specifičnih zadataka s daleko većom učinkovitošću, zahvaljujući njihovoj sposobnosti da iskoriste hardverski paralelizam. To se prevodi u značajno smanjenje latencije i povećanje propusnosti za kritične aplikacije, kao što je obrada signala ili kriptografija. Drugo, fleksibilnost FPGA znači da se hardver može rekonfigurirati kako bi zadovoljio promjenjive potrebe, nudeći nenadmašnu skalabilnost. Treće, FPGA-ovi imaju tendenciju da budu energetski učinkovitiji za određene specijalizirane zadatke, troše manje energije od CPU-a, a daju visoke performanse. Naposljetku, u određenim scenarijima, FPGA mogu smanjiti dugoročne troškove izbjegavanjem potrebe za više namjenskih hardverskih komponenti, zahvaljujući njihovoj sposobnosti da integriraju više funkcija na jednom reprogramabilnom čipu.

6. FPGA IMPLEMENTACIJA AES ALGORITMA NA ZYNQ ULTRASCALE+ MPSoC PLATFORMI

AMD Zynq Ultrascale+ MPSoC je višeprocesorski SoC (eng. System on Chip) sa 64 – bitnim i 32 – bitnim podsustavima za obradu, namjenskim procesorima za obradu grafike i videa u stvarnom vremenu, naprednim perifernim uređajima velike brzine i programabilnom logikom.

Da bismo postigli visoke performanse i nisku latenciju, napredni multimedijski uređaji moraju imati odgovarajući hardver te mogućnost dodavanja prilagođene logike. Zynq Ultrascale+ ima sve navedeno u jednom čipu. Mogućnost dodavanja prilagođene logike i povezivanje bilo kojeg sa bilo kojim na istom uređaju rezultira značajnom uštedom energije.

Zynq UltraScale+ MPSoC kombinira PS (Processing system) i korisnički PL (Programmable logic) u isti uređaj. PS ima Arm Cortex – A53 64-bitni četverojezgreni ili dvojezgreni procesor (APU), Cortex-R5F dvojezgreni procesor stvarnog vremena (RPU) i grafičku procesorsku jedinicu (GPU).



Slika 6.12. Blok shema Zynq Ultrascale+ MPSoC EG uređaja

Istaknute značajke Zynq UltraScale+ MPSoC:

- Jedinica za obradu aplikacija radi sa četverojezgrenim ili dvojezgrenim Arm Cortex A53 procesorima
    - Arm v8 arhitektura koja podržava 64 – bitni ili 32 – bitni način rada
    - Idealni za Linux i bare – metal aplikacijske sustave
- Dvojezgrena procesorska jedinica za rad u stvarnom vremenu Arm Cortex R5F
- Grafička procesorska jedinica (GPU) Arm Mali 400MP2
- Integrirana periferija velike brzine
    - PCIe Gen2
    - USB 3.0/2.0
    - Gigabitni ethernet
    - SATA 3.1 host
- 6 – portni DDR kontroler sa ECC značajkom

Implementacija je izvršena na Xilinx ZCU102 razvojnoj ploči koja sadrži Zynq UltraScale+ XCZU9EG-2FFVB1156 MPSoC čip.

Logičke čelije	**RAM memorija**	**DSP blokovi**	**16.3 Gb/s transiver**	**Broj pinova**
600K	32.1	2520	24	328

Tablica 6.5. Specifikacija Zynq UltraScale+ XCZU9EG-2FFVB1156

Bitne značajke ZCU102 razvojne ploče:

- PS 4GB DDR4 64 – bit SODIMM w/ ECC
- PL 512MB DDR4 1200MHz/2400Mbps
- 8KB IIC EEPROM
- Dual 64MB Quad SPI flash
- USB 2.0/3.0
- 2x FMC-HPC konektori
- 1 Gbps ethernet sučelje
- 4x SFP+ sučelja
- SATA
- Programabilni izvori takta

6.1. AES GCM VERILOG MODUL

Kako bi pouzdano upravljali tokom podataka, kako ulaznih tako i izlaznih podataka iz jezgre koristio sam AXI4 STREAM protokol. Sučelja koje AXI4 STREAM protokol koristi prikazani su u sljedećoj tablici:

Naziv	Širina podataka (bit)	Obavezan	Opis
TDATA	proizvoljna	DA	Primarno sučelje za prijenos podataka
TKEEP	Širina TDATA / 8	NE	Svaki bit ovoga sučelja označava valjanost pripadnog okteta iz TDATA sučelja
TLAST	1	NE	Označava kraj paketa
TVALID	1	DA	Pošiljatelj označava jesu li podaci ispravni
TREADY	1	NE	Primatelj označava jeli spreman primiti nove podatke
TUSER	proizvoljna	NE	Korisnički dodatni podaci po izboru
TSTRB	Širina TDATA / 8	NE	Svaki bit ovoga sučelja označava jeli pripadni oktet iz TDATA sučelja predstavlja podatke ili poziciju
TDEST	proizvoljna	NE	Predstavlja informacije o usmjeravanju toka podatatka
TID	proizvoljna	NE	Identifikator toka podataka

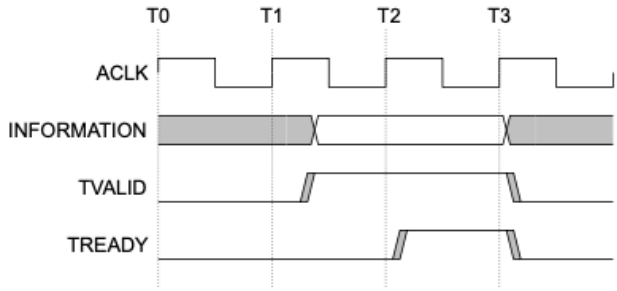
Tablica 6.6. AXI4 STREAM sučelje

TVALID i TREADY rukovanje(eng. handshake) određuje kada se informacije prosljeđuju preko sučelja. Mehanizam dvosmjerne kontrole protoka omogućuje i pošiljatelju i primatelju kontrolu brzine kojom se podaci i upravljačke informacije prenose preko sučelja.

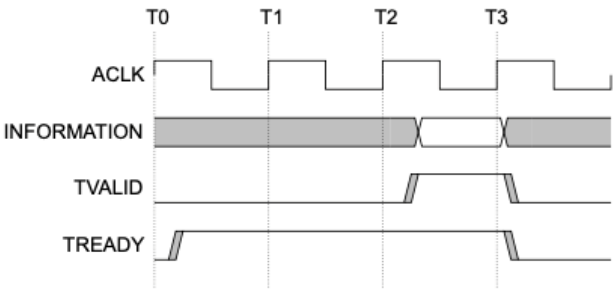
Da bi došlo do prijenosa podataka, moraju biti potvrđeni i TVALID i TREADY signali. TVALID ili TREADY mogu se prvi potvrditi ili se oba mogu potvrditi na istom bridu takta.

Pošiljatelju nije dopušteno čekati valjani TREADY signal da bi potvrdio TVALID. Nakon što je TVALID potvrđen, mora ostati potvrđen dok se ne dogodi rukovanje.

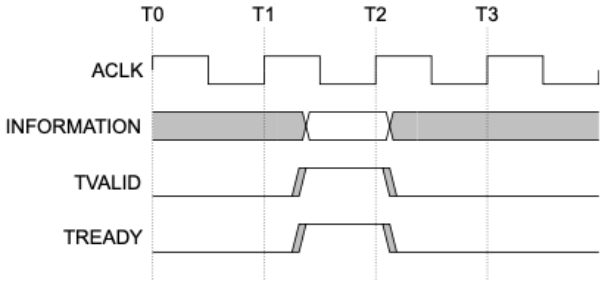
Primatelju je dopušteno čekati da se TVALID potvrdi prije nego što potvrdi TREADY te je također dopušteno da primatelj potvrdi i opozove TREADY bez potvrđivanja TVALID-a.



Slika 6.13. TVALID potvrđen prije TREADY

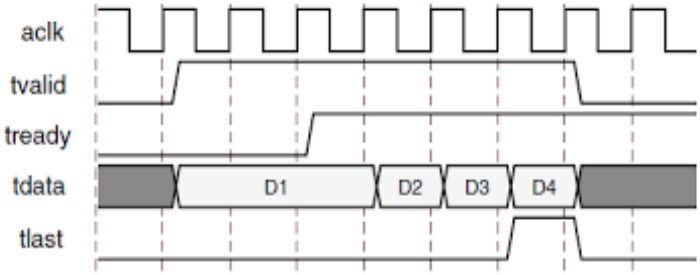


Slika 6.14. TREADY potvrđen prije TVALID



Slika 6.15. TVALID I TREADY potvrđeni istovremeno

Ako se koristi paketni prijenos podataka, kraj paketa, odnosno zadnji blok podatka označava se tako da se potvrdi TLAST. TLAST postavlja pošiljatelj zajedno sa zadnjim blokom podataka i TVALID signalom.



Slika 6.16. Korištenje TLAST signala

```
module axi_aes_gcm_enc #
(
    parameter AXI_WIDTH = 32,
    parameter AAD_WIDTH = 160
)
(
    (* X_INTERFACE_INFO = "xilinx.com:signal:clock:1.0 s_axi_aclk CLK" *)
    (* X_INTERFACE_PARAMETER = "ASSOCIATED_BUSIF S_AXI_IN : M_AXI_OUT, ASSOCIATED_RESET s_axi_aresetn" *)
    input                       s_axi_aclk,
    (* X_INTERFACE_INFO = "xilinx.com:signal:reset:1.0 s_axi_aresetn RST" *)
    input                       s_axi_aresetn,

    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 S_AXI_IN TDATA" *)
    input [AXI_WIDTH - 1 : 0]   s_axi_in_tdata,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 S_AXI_IN TKEEP" *)
    input [AXI_WIDTH/8 - 1 : 0] s_axi_in_tkeep,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 S_AXI_IN TVALID" *)
    input                       s_axi_in_tvalid,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 S_AXI_IN TLAST" *)
    input                       s_axi_in_tlast,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 S_AXI_IN TREADY" *)
    output                      s_axi_in_tready,

    input [255 : 0]             key,
    input [95 : 0]              iv,
    input [AAD_WIDTH - 1 : 0]   aad,

    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 M_AXI_OUT TDATA" *)
    output [AXI_WIDTH- 1 : 0]  s_axi_out_tdata,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 M_AXI_OUT TKEEP" *)
    output [AXI_WIDTH/8 - 1 :0] s_axi_out_tkeep,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 M_AXI_OUT TVALID" *)
    output                      s_axi_out_tvalid,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 M_AXI_OUT TLAST" *)
    output                      s_axi_out_tlast,
    (* X_INTERFACE_INFO = "xilinx.com:interface:axis:1.0 M_AXI_OUT TREADY" *)
    input                       s_axi_out_tready
);
```

Slika 6.17. Ulazni i izlazni podaci iz aes gcm jezgre

Ulazni podaci u AES GCM jezgru su:

- čisti tekst za koji koristimo AXI4 STREAM protokol (S_AXI_IN)
- inicijalizacijski vektor (iv)
- ključ (key)
- dodatni autentificirani podaci (aad)

Izlazni podaci iz AES GCM jezgre:

- šifrat zajedno sa autentifikacijskom oznakom za koji također koristimo AXI4 STREAM protokol (M_AXI_OUT)

Ulazni i izlazni tok podatka je blok širine 32 bita. Kako smo ranije napisali, AES algoritam zahtijeva blokove širine 128 bita. Kako bi zadovoljili ovo svojstvo, implementirao sam konverter širine podataka proizvoljne duljine koji koristi AXI4 STREAM protokol. Ulazna i izlazna duljina podataka zadaju se pomoću parametara modula. Duljina mora zadovoljiti uvjet da je potencija broja 2 (AXI_WIDTH_IN = 2^n, AXI_WIDTH_OUT = 2^n). Tako od četiri ulazna bloka podataka širine 32 bita formiramo blok duljine 128 bita, te analogno ovome, četiri izlazna bloka širine 32 bita formiramo od jednog bloka širine 128 bita.

```
axi_width_converter #
(
    .AXI_WIDTH_IN(AXI_WIDTH),
    .AXI_WIDTH_OUT(128)
)
awc128
(
    .s_axi_aclk(s_axi_aclk),
    .s_axi_aresetn(s_axi_aresetn),

    .s_axi_in_tdata(s_axi_in_tdata),
    .s_axi_in_tkeep(s_axi_in_tkeep),
    .s_axi_in_tvalid(s_axi_in_tvalid),
    .s_axi_in_tlast(s_axi_in_tlast),
    .s_axi_in_tready(s_axi_in_tready),

    .s_axi_out_tdata(s_axi_awc128_out_tdata),
    .s_axi_out_tkeep(s_axi_awc128_out_tkeep),
    .s_axi_out_tvalid(s_axi_awc128_out_tvalid),
    .s_axi_out_tlast(s_axi_awc128_out_tlast),
    .s_axi_out_tready(s_axi_awc128_out_tready)
);
```

Slika 6.18. Kreiranje instance konvertera duljine podataka u aes gcm jezgri

Nakon što smo kreirali blokove duljine 128 bita, sada može početi šifriranje podataka. Znamo da AES u GCM načinu rada kombinira AES u „counter“ načinu rada te GHASH funkciju za računanje autentifikacijske oznake tako da dobiveni blokovi duljine 128 bita ulaze u „aes_counter“ verilog modul kako je prikazano na slici ispod.

```
aes_ctr #
(
    .AXI_WIDTH(128)
)
data_enc
(
    .s_axi_aclk(s_axi_aclk),
    .s_axi_aresetn(s_axi_aresetn),

    .s_axi_in_tdata(aes_ctr_in_tdata),
    .s_axi_in_tkeep(aes_ctr_in_tkeep),
    .s_axi_in_tvalid(aes_ctr_in_tvalid),
    .s_axi_in_tlast(aes_ctr_in_tlast),
    .s_axi_in_tuser(aes_ctr_in_tuser),
    .s_axi_in_tready(aes_ctr_in_tready),

    .key(aes_ctr_key),
    .iv(aes_ctr_iv),

    .s_axi_out_tdata(aes_ctr_out_tdata),
    .s_axi_out_tkeep(aes_ctr_out_tkeep),
    .s_axi_out_tvalid(aes_ctr_out_tvalid),
    .s_axi_out_tlast(aes_ctr_out_tlast),
    .s_axi_out_tuser(aes_ctr_out_tuser),
    .s_axi_out_tready(aes_ctr_out_tready)
);
```

Slika 6.19. Kreiranje instance aes_ctr verilog modula

Kako bismo osigurali visoku propusnost podataka koristio sam ulančavanje (eng. pipeline). U slučaju da primatelj nije spreman primiti podatke iz AES GCM jezgre (TREADY signal nije postavljen) nema smisla blokirati cijeli lanac.

Kako već znamo da AES – 256 protokol ima 14 koraka, a primatelj nije spreman primiti nove podatke, svaki korak algoritma (modul „aes_round“) može primiti nove podatke dok mu god izlaz nije valjan ili je idući korak spreman primiti nove podatke što se vidi iz slike 6.20.

```
assign s_axi_in_tready = (!s_axi_out_tvalid) ? 1'b1 : s_axi_out_tready;


subBytes sb(s_axi_in_tdata, subBytes_out);
shiftRows sr(subBytes_out, shiftRows_out);

generate
    if(LAST_ROUND == 0) begin
        mixColumns mc(shiftRows_out, mixColumns_out);
    end
endgenerate

always @(posedge s_axi_aclk) begin

    if(!s_axi_out_tvalid || s_axi_out_tready) begin
        tdata_out <= (LAST_ROUND == 0) ? mixColumns_out ^ ske_key_out[127 -: 128] : shiftRows_out ^ ske_key_out[127 -: 128];
        tkeep_out <= s_axi_in_tkeep;
        tvalid_out <= s_axi_in_tvalid;
        tlast_out <= s_axi_in_tlast;
        tuser_out <= s_axi_in_tuser;
        plainText <= plainTextIn;
        key_reg <= ske_key_out;
    end

end
```

Slika 6.20. Dio koda AES koraka

```
module mul2
(
    input [7:0] dataIn,
    output [7:0] dataOut
);

assign dataOut = dataIn[7] ? (dataIn << 1) ^ 8'h1B : (dataIn << 1);

endmodule
```

```
module mul3
(
    input [7:0] dataIn,
    output [7:0] dataOut
);

wire[7:0] mul2Out;

mul2 m2_1(dataIn, mul2Out);

assign dataOut = mul2Out ^ dataIn;
endmodule
```

```
module mix_column_single
(
    input [31:0] dataIn,
    output [31:0] dataOut
);

wire [7:0] c1, c2, c3, c4;
wire [7:0] out1, out2, out3, out4;
wire [7:0] mul2_out1, mul2_out2, mul2_out3, mul2_out4;
wire [7:0] mul3_out1, mul3_out2, mul3_out3, mul3_out4;

assign c1 = dataIn[31:24];
assign c2 = dataIn[23:16];
assign c3 = dataIn[15:8];
assign c4 = dataIn[7:0];

begin
    mul2 m2_1(c1, mul2_out1);
    mul2 m2_2(c2, mul2_out2);
    mul2 m2_3(c3, mul2_out3);
    mul2 m2_4(c4, mul2_out4);

    mul3 m3_1(c1, mul3_out1);
    mul3 m3_2(c2, mul3_out2);
    mul3 m3_3(c3, mul3_out3);
    mul3 m3_4(c4, mul3_out4);
end

assign out1 = mul2_out1 ^ mul3_out2 ^ c3 ^ c4;
assign out2 = c1 ^ mul2_out2 ^ mul3_out3 ^ c4;
assign out3 = c1 ^ c2 ^ mul2_out3 ^ mul3_out4;
assign out4 = mul3_out1 ^ c2 ^ c3 ^ mul2_out4;

assign dataOut = {out1, out2, out3, out4};

endmodule
```

```
module mixColumns
(
    input [127:0] dataIn,
    output [127:0] dataOut
);

wire [31:0] c1, c2, c3, c4;
wire [31:0] out1, out2, out3, out4;

assign c1 = dataIn[127 -: 32];
assign c2 = dataIn[95 -: 32];
assign c3 = dataIn[63 -: 32];
assign c4 = dataIn[31 -: 32];

mix_column_single mix1(c1, out1);
mix_column_single mix2(c2, out2);
mix_column_single mix3(c3, out3);
mix_column_single mix4(c4, out4);

assign dataOut = {out1, out2, out3, out4};

endmodule
```

Slika 6.21. Modul za miješanje stupaca u Verilog HDL – u

```
always @* begin

    iv_next = iv_reg;

    if(s_axi_in_tready) begin
        aes_tdata_next = aes_tdata;
        aes_tkeep_next = aes_tkeep;
        aes_tvalid_next = aes_tvalid;
        aes_tlast_next = aes_tlast;
        aes_tuser_next = aes_tuser;
    end
    else begin
        aes_tdata_next = aes_tdata_reg;
        aes_tkeep_next = aes_tkeep_reg;
        aes_tvalid_next = aes_tvalid_reg;
        aes_tlast_next = aes_tlast_reg;
        aes_tuser_next = aes_tuser_reg;
    end

    key_next = key_reg;

    case(state_reg)
        STATE_START: begin
            state_next = STATE_START;

            if(aes_tvalid && s_axi_in_tready) begin

                iv_next = iv + 1;
                key_next = key;
                state_next = STATE_PROCESS;
            end
        end

        STATE_PROCESS: begin
            state_next = STATE_PROCESS;

            if(aes_tvalid && s_axi_in_tready) begin

                iv_next = iv_reg + 1;
                state_next = (aes_tlast) ? STATE_START : STATE_PROCESS;
            end
        end
    endcase

end

always @(posedge s_axi_aclk) begin
    state_reg <= state_next;
    iv_reg <= iv_next;
    key_reg <= key_next;

    aes_tdata_reg <= aes_tdata_next;
    aes_tkeep_reg <= aes_tkeep_next;
    aes_tvalid_reg <= aes_tvalid_next;
    aes_tlast_reg <= aes_tlast_next;
    aes_tuser_reg <= aes_tuser_next;

end
endmodule
```

Slika 6.22. Implementacija aes_ctr verilog modula

Na slici 6.22. je prikazana implementacija aes_ctr verilog modula. Kako radimo u paketnom prijenosu podataka, početno stanje FSM – a (eng. Finite State Mashine) je STATE_START gdje čekamo novi paket, odnosno prvi blok duljine 128 bita novog paketa. Kada zadovoljimo uvjet da smo spremni primiti nove podatke i pošiljatelj nam isporuči ispravne podatke, uzimamo ključ i inicijalizacijski vektor koje spremamo u registar te ih koristimo za vrijeme trajanja trenutnog paketa. Nakon toga prelazimo u stanje STATE_PROCESS gdje uzimamo iduće blokove trenutnog paketa te inkrementiramo inicijalizacijski vektor sve dok TLAST signal ne bude postavljen nakon čega se vraćamo u početno stanje.

6.2. REZULTATI

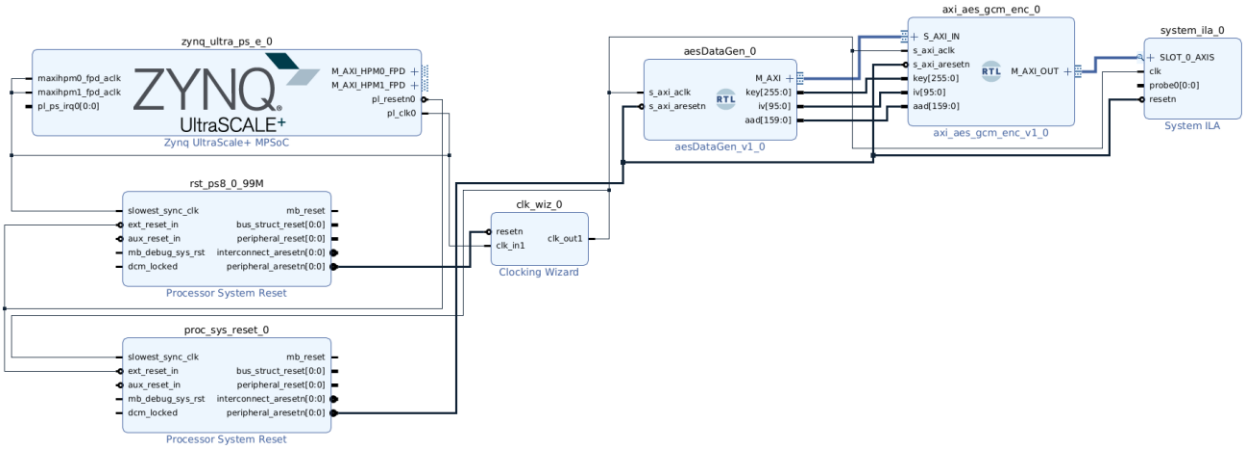
Za potrebe testiranja napravio sam jednostavni generator podataka (modul „aesDataGen) u kojem sam koristio NIST – ove test vektore:

KLJUČ: feffe9928665731c6d6a8f9467308308feffe9928665731c6d6a8f9467308308

ČISTI TEKST: d9313225f88406e5a55909c5aff5269a86a7a9531534f7da2e4c303d8a318a72
1c3c0c95956809532fcf0e2449a6b525b16aedf5aa0de657ba637b39

DODATNI AUTENTIFICIRANI PODACI: feedfacedeadbeeffeedfacedeadbeef
abaddad2

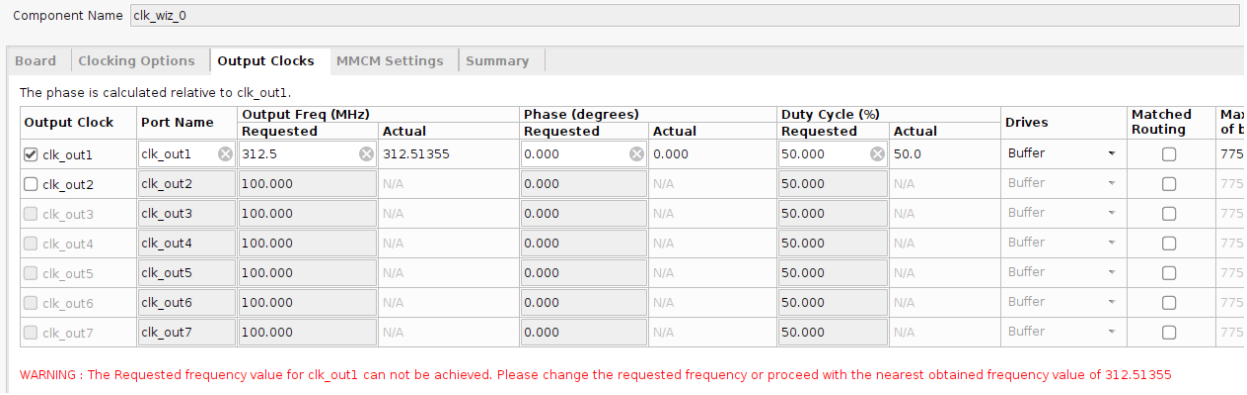
INICIJALIZACIJSKI VEKTOR: cafebabefacedbaddecaf888



Slika 6.23. Blok dijagram finalnog dizajna

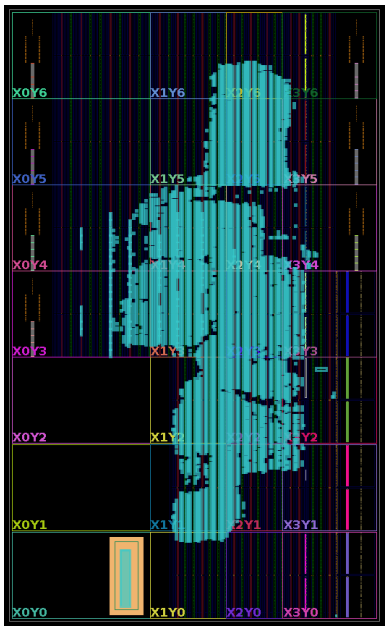
Kako vidimo iz blok dijagrama dizajna, takt za programabilnu logiku generira PS (modul „zynq_ultra_ps_e_0“) frekvencije 100Mhz nakon čega koristeći Xilinx jezgru „Clocking wizard“

generiramo takt frekvencije cca 312.5 MHz što se vidi iz slike ispod te trenutno stanje svih signala možemo pratiti u stvarnom vremenu zahvaljujući jezgri System ILA (Integrated Logic Analyzer).



Slika 6.24. Generiranje takta pomoću „Clocking Wizard“ jezgre

Zadnji korak koji preostaje je odraditi proces sinteze i implementacije te nakon uspješne implementacije treba provjeriti jeli nam logički dizajn zadovoljava vremena propagacije signala (vrijeme propagacije signala mora biti manje od perioda takta) što se vidi na sljedećim slikama.



Slika 6.25. Implementirani dizajn na FPGA čipu

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.103 ns	Worst Hold Slack (WHS):	0.002 ns	Worst Pulse Width Slack (WPWS):	1.058 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	51183	Total Number of Endpoints:	51167	Total Number of Endpoints:	30637

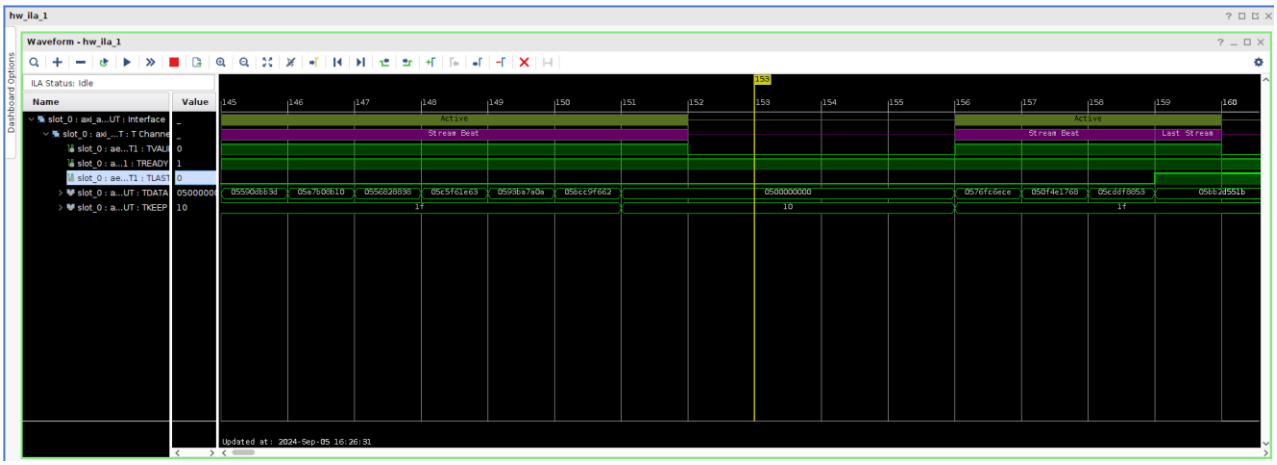
All user specified timing constraints are met.

Slika 6.26. Provjera vremena propagacije signala

Name	CLB LUTs (274080)	CLB Registers (548160)	CARRY8 (34260)	F7 Muxes (137040)	F8 Muxes (68520)	CLB (34260)	LUT as Logic (274080)	LUT as Memory (144000)	Block RAM Tile (912)	GLOBAL CLOCK BUFFERs (404)	MMCM (4)	PS8 (1)	BSCANE2 (4)
design_1_wrapper	22.38%	5.54%	0.17%	9.55%	7.37%	33.27%	22.30%	0.16%	9.87%	0.99%	25.00%	100.0	25.00%
dbg_hub (dbg_hub)	0.16%	0.14%	0.02%	0.00%	0.00%	0.42%	0.15%	0.01%	0.00%	0.25%	0.00%	0.00%	25.00%
design_1_i (design_1)	22.22%	5.40%	0.15%	9.55%	7.37%	32.89%	22.14%	0.14%	9.87%	0.74%	25.00%	100.0	0.00%
aesDataGen_0 (desi	<0.01%	<0.01%	0.00%	0.00%	0.00%	0.06%	<0.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
axi_aes_gcm_enc_0 (	21.78%	5.10%	0.09%	9.48%	7.31%	32.16%	21.75%	0.05%	0.00%	0.25%	0.00%	0.00%	0.00%
clk_wiz_0 (design_1_c	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.25%	25.00%	0.00%	0.00%
proc_sys_reset_0 (de	<0.01%	<0.01%	0.00%	0.00%	0.00%	0.02%	<0.01%	<0.01%	0.00%	0.00%	0.00%	0.00%	0.00%

Slika 6.27. Iskorišteni resursi FPGA čipa

Rezultat uspješne implementacije je bitstream koji učitamo na FPGA te u ovom slučaju provjeravamo ispravnost logike u stvarnom vremenu na FPGA čipu koristeći ILA jezgru.



Slika 6.28. Praćenje rada logike u stvarnom vremenu

7. ZAKLJUČAK

U ovom radu osim povijesti i osnova kriptografije detaljno je opisan AES enkripcijski algoritam u GCM načinu rada te njegova implementacija na FPGA platformi. Kako je u današnje vrijeme sve "online", od komunikacije u stvarnom vremenu, pohrane podataka u oblak, internet bankarstva i slično, nezamislivo je da se prijenos podatka odvija nezaštićen jer bi to dovelo do nezamislivih posljedica.

Brzina prijenosa podataka je sve veća i veća te je sve zahtjevnije šifrirati podatke u stvarnom vremenu. Za tu svrhu se koristi hardverski ubrzano šifriranje za što se koristi FPGA čije su najbitnije značajke visoka propusnost i niska latencija.

LITERATURA

1. AES: How the Most Advanced Encryption Actually Works (https://medium.com/codex/aes-how-the-most-advanced-encryption-actually-works-b6341c44edb9)
2. Principles of digital communications, Chapter 7: "Introduction to finite fields" (https://web.stanford.edu/~marykw/classes/CS250_W19/readings/Forney_Introduction_to_Finite_Fields.pdf)
3. Federal Information Processing Standards Publication 197: ADVANCED ENCRYPTION STANDARD (AES), 2001
4. NIST Special Publication 800-38D; Morris Dworkin: COMPUTER SECURITY, 2007
5. Sarah L. Harris, David Money Harris: Digital Design and Computer Architecture, 2016
6. David A. McGrew, John Viega: The Galois/Counter Mode of Operation (GCM)
7. Josipa Barić, Ivana Grgić, Iva Jurić: Osnove kriptografije
8. AES algoritam: CCERT-PUBDOC (https://www.cis.hr/www.edicija/LinkedDocuments/CCERT-PUBDOC-2003-08-37.pdf)

POPIS KRATICA

FPGA	Field Programmable Gate Array
MPSoC	MultiProcessor System on Chip
HDL	Hardware Description Language
DES	Data Encryption Standard
SSL	Secure Sockets Layer
TLS	Transport Layer Security
NIST	National Institute of Standards and Technology
GF	Galois Field
AES	Advanced Encryption Standard
GCM	Galois/Counter Mode
CTR	Counter mode
ASIC	Application Specific Integrated Circuit
DSP	Digital Signal Processor
GPIO	General Purpose Input/Output
IP	Intellectual Property
CLB	Configurable Logic Block
LUT	LookUp Table
RAM	Random Access Memory
SRAM	Static Random Access Memory
IOB	Input/Output Block
FF	Flip – Flop
RTL	Register Transfer Level
CPU	Central Processing Unit
GPU	Graphics Processing Unit
RPU	Real – time Processing Unit
PS	Processing System
PL	Programmable Logic
ILA	Integrated Logic Analyzer
FSM	Finite State Machine

POPIS SLIKA

POPIS TABLICA