DIGITAL SYSTEM DESIGN FINAL PROJECT REPORT
DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITAS INDONESIA

CRYPTOGRAPHIC MODULE

GROUP 2

Nicholas Chriscia	**2406369015**
Mochammad Rafly Fatih Rabbani	**2406369021**
Yohana Indah Nathania Br Sihotang	**2406368946**
Rafael Raditya Setyono	**2406369040**

PREFACE

This project is formed in fulfilling the requirement of the Digital System Design Course Lab final project. This report documents the in depth implementations of each step when creating this project including its theory, process and results of the chosen topic that is a part of the Cryptographic Module called Advanced Encryption Standard (AES). This project implements the AES-128 encryption algorithm using VHDL. Each component of the AES transformations which includes the SubBytes, ShiftRows, MixColumns, AddRoundKey and the GF(2^8) multiplications are created between separated VHDL files where each component is then combined into the top level code along with the development of the testbench to simulate its process.

Depok, 7 December 2025

Group 2

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND

Cryptography is a category that aims at securing sensitive information such as passwords and identifications from unauthorized access that might use it for bad intentions. It is able to secure information with the help of data transformations that involve state matrixes and math computations. Cryptography is able to promote several key features which includes confidentiality, non-repudiation, integrity, adaptability, interoperability, and authentication.

One of the subcategories that contributes towards this category is AES. AES stands for Advanced Encryption Standard which has been an algorithm function that has widely been used for cryptography. The AES supports various lengths of inputs which are either 128, 192 or 256 bits.

1.2 PROJECT DESCRIPTION

This VHDL project implements a complete AES encryption core suitable for FPGA synthesis and RTL simulation. The datapath is fully structural: key schedule, SubBytes (S-box), ShiftRows, MixColumns, and AddRoundKey are separate modules. A simple controller sequences rounds, exposes a clean streaming/bus interface, and a testbench verifies correctness with known answer vectors.

1.3 OBJECTIVES

The objective of this project is to design and implement the AES-128 encryption algorithm using VHDL to demonstrate understanding of hardware-based security algorithms.

1.4 ROLES AND RESPONSIBILITIES

The roles and responsibilities assigned to the group members are as follows:

Roles	Responsibilities	Person
aes_top.vhd	top-level AES datapath + controller	Nico
sbox_comb.vhd	SubBytes (256-entry constant lookup)	Rafael
shiftrows.vhd	ShiftRows combinational	Rafael
mixcolumns.vhd	MixColumns (uses gf mul functions)	Fatih
addroundkey.vhd	XOR round key onto state	Yohana
key_expansion.vhd	Key schedule (generates round keys)	Yohana
aes_tb.vhd	AES testbench	Nico
utils.vhd	Calculations of GF(2) of multiplications 2 and 3.	Fatih

Table 1. Roles and Responsibilities

CHAPTER 2
IMPLEMENTATION

2.1 EQUIPMENT

The tools that are going to be used in this project are as follows:

- Vivado
- Github, Git
- Google Docs
- Canva

2.2 IMPLEMENTATION

2.2.1 Substitute Bytes

		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

Fig 2.2.1.1. Standard S-Box Diagram

Substitute Bytes is the first transformation function that occurs within the start of each encryption round within AES. Substitute Bytes uses an S-box which aims to improve confusion algorithms and already has the size of 256 entries to choose from during the substituting process. How it works is that after receiving the input of the 4x4 state matrix, it would then look-up through the S-box to replace bytes from the given input accordingly. Finally, the output of this transformation will form a 4x4 state matrix. It can be seen from the code where there is a list of constants that references the S-Box diagram with the size of 256 entries following the Figure 2.2.1.1. This is implemented within the sbox_comb.vhd code file.

2.2.2 Shift Rows

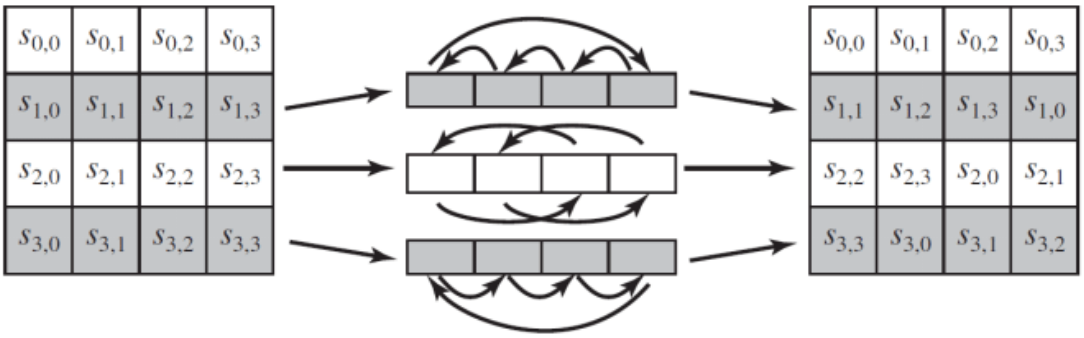


Fig 2.2.2.1. Shift Row Transformation

The Shift Rows is the second transformation function that occurs within each encryption round within AES and operates on a 4x4 matrix. As its name suggests, it will perform shifts within each individual row where it shifts to the left to provide improved diffusion by spreading individual input bytes across columns. Shift Rows perform patterns according to each individual row. Let's say that the state matrix is denoted as $S_{r,c}$ where each transformation gives the result of $S`_{r,c} = S_{r,(c + r) \bmod 4}$ thus this gives out the transformation mathematically to each row where.

$$\text{Row [0]} : S`_{0,c} = S_{0,(c + 0) \bmod 4} \; ; \; \text{Row [1]} : S`_{1,c} = S_{1,(c + 1) \bmod 4}$$

$$\text{Row [2]} : S`_{2,c} = S_{2,(c + 2) \bmod 4} \; ; \; \text{Row [3]} : S`_{3,c} = S_{3,(c + 3) \bmod 4}$$

This shows that the first row does not require any shifting, the second row is shifted by 1 position, the third row is shifted by 2 positions, and the fourth row would be shifted by 3 positions. This mathematical form also follows the figure 2.2.2.1 that has been shown above. This is implemented within the shiftrows.vhd code file.

2.2.3 Galois Field

Within AES, Galois Field specifically GF(2^8) is one of the key aspects of cryptography which is used for mix columns. When implementing the galois field for AES, it uses the rijndael irreducible polynomial which is $f(t) = t^8 + t^4 + t^3 + t + 1$. In this project, the galois field will treat each byte within the state matrix as a polynomial. Let's say a byte b consists of 8 bits where its coefficient can be written as b = (b_7 , b_6 , b_5 , b_4 , b_3 , b_2 , b_1 , b_0) and it is represented on each byte within the state matrix as :

$$b_7t^7 + b_6t^6 + b_5t^5 + b_4t^4 + b_3t^3 + b_2t^2 + b_1t + b_0$$

Each bit follows the principle of GF(2) arithmetic where addition uses XOR and multiplication uses AND. During multiplication by 2 which can be represented as t, GF(2) applies modular reduction according to the most significant bit which is b_7 :

$$t \bullet b(t) = b_7t^8 + b_6t^7 + b_5t^6 + b_4t^5 + b_3t^4 + b_2t^3 + b_1t^2 + b_0t$$

If $b_7 = 0$, then the result will only perform the left shift operation since the polynomial degree remains below 8.

$$\begin{aligned} t \bullet b(t) &= (0)\ t^8 + b_6t^7 + b_5t^6 + b_4t^5 + b_3t^4 + b_2t^3 + b_1t^2 + b_0t \\ &= b_6t^7 + b_5t^6 + b_4t^5 + b_3t^4 + b_2t^3 + b_1t^2 + b_0t \end{aligned}$$

Whereas if $b_7 = 1$, then modular reduction is applied using the field equivalence derived from $f(t) = 0$ which is $t^8 = t^4 + t^3 + t + 1$ in GF(2^8).

$$\begin{aligned} t \bullet b(t) &= (1)\ t^8 + b_6t^7 + b_5t^6 + b_4t^5 + b_3t^4 + b_2t^3 + b_1t^2 + b_0t \\ &= (1)\ t^8 + b_6t^7 + b_5t^6 + b_4t^5 + b_3t^4 + b_2t^3 + b_1t^2 + b_0t \\ &= (t^4 + t^3 + t + 1) + b_6t^7 + b_5t^6 + b_4t^5 + b_3t^4 + b_2t^3 + b_1t^2 + b_0t \\ &= b_6t^7 + b_5t^6 + b_4t^5 + (b_3 \oplus 1)t^4 + (b_2 \oplus 1)t^3 + b_1t^2 + (b_0 \oplus 1)t + 1 \\ &= b_6t^7 + b_5t^6 + b_4t^5 + (b_3 \oplus b_7)t^4 + (b_2 \oplus b_7)t^3 + b_1t^2 + (b_0 \oplus b_7)t + b_7 \end{aligned}$$

During multiplication 3 in GF(2^8) , we can take the polynomial representation which is (t + 1) where :

$$3 \bullet b(t) = (t + 1) \bullet b(t) = t \bullet b(t) \oplus b(t)$$

Both multiplications are implemented within the utility.vhd and the polynomial coefficient input of b is written as x as the input signal within the code file.

2.2.4 MixColumns

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

Fig 2.2.4.1 MixColumns Transformation Matrix

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Fig 2.2.4.2 Inverse MixColumns Transformation Matrix

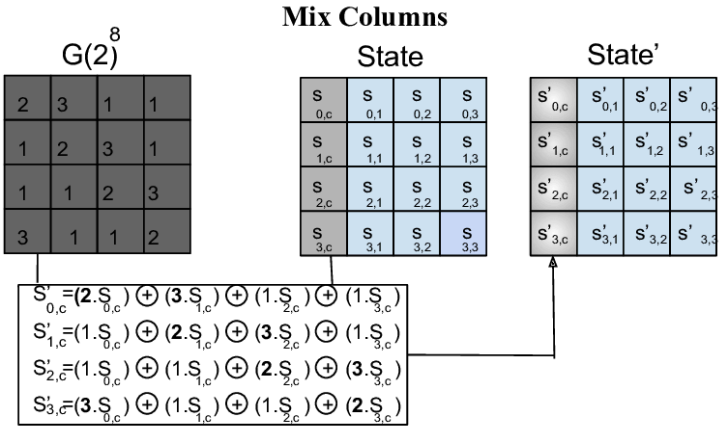


Fig 2.2.4.3 MixColumns Transformation Matrix With Formula

The Mix Column is the third transformation function that occurs within each encryption round within AES and operates on a 4x4 state matrix. The mix column uses a constant transformation matrix using the finite field arithmetic from Galois Field ($GF(2^8)$) where it multiplies each column of the state matrix. It can be seen that the matrix requires different kinds of Galois Field multiplication where the encryption is shown on Fig 2.2.4.1 requires multiplication of constants 02 and 03 and the inverse matrix is shown on Fig 2.2.4.2 which is to decrypt requires multiplication of constants 09, 0B, 0D and 0E. The mathematical formulation from Fig 2.2.4.3 shows how the matrix multiplication operates for each column independently since this project focuses on developing encryption and it is implemented within the mixcolumn.vhd file.

2.2.5 Round Key (AddRoundKey)

2.2.5.1 Theory

AddRoundKey is the last operation performed during each AES (Advanced Encryption Standard) encryption round. A bitwise XOR operation is completed with the State Matrix (128-bit data block) and the applicable Round Key (128-bit derived key) for that round. Since XOR's operation is inverted, applying the same key will restore the initial data. In mathematical terms, if:

- S is the 4×4 State Matrix or 128 Bits
- K is the 4×4 Round Key or 128 Bits,
- Then: $S' = S \oplus K$, where $\oplus$ represents the bitwise XOR operation.

The primary aim of AddRoundKey is to introduce some key randomness into the encryption process, thus improving encryption security through confusion principles. Here is the block diagram of AddRoundKey:

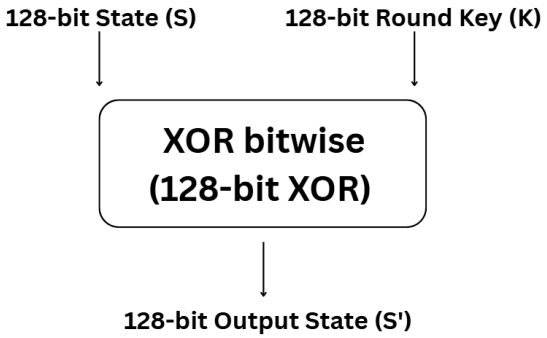


Figure 2.2.5.1: Block diagram of the AddRoundKey operation. Inputs are the State Matrix and Round Key; output is the bitwise XOR result.

2.2.5.2 Implementation

The unit provided (addroundkey.vhd) shows the simplest form of the Operation Combining Input and Key Material using an XOR function. The design of this module uses a process block which will trigger on a change of either the state's input data (state_in) or the data in the round key (round_key), and will produce a combined output from both inputs at that time (without requiring a clock to maintain stability between inputs). By using a combinational design, it removes delays associated with the required timing signals necessary for synchronous design techniques, and thus simplifies using this module within an AES Pipeline system architecture. The combination of keying material with intermediate ciphertext occurs as a logical operation performed individually on every bit of the combined 128-bit

signal(s). In doing this, it provides means to create the diffusion and confusion portions of the encryption process within AES. Here is the code:

```
architecture Behavioral of addroundkey is
begin
    process(state_in, round_key)
    begin
        state_out <= state_in xor round_key;
    end process;
end Behavioral;
```

Figure 2.2.5.2 Code Implementation

2.2.6 Key Expansion

2.2.6.1 Theory

The AES-128 algorithm generates 11 unique round keys based on a single defining 128-bit key through a process called Key Expansion. All 11 Keys are used once during each of the 11 rounds (0-10) in an operation called Add Round Key. The Key Expansion function has the following format:

- Round Key 0 = Key
- For i = 1 to 10, the calculation of each of the subsequent Round Keys is done as follows:
    - Calculate each 32-bit Word
    - Calculate the Word from the previous Round Key;
    - Shift the last byte of the Word to the left (RotWord);
    - Substitue each byte of the Word using SubWord;
    - XOR the substituted Word with the Round Constant (Rcon) so there is no pattern.

Here is the process illustration for round i:

$$w[4i] = w[4(i-1)] \oplus g(w[4i-1])$$

$$w[4i+1] = w[4i] \oplus w[4i-3]$$

$$w[4i+1] = w[4i] \oplus w[4i-2]$$

$$w[4i+3] = w[4i+2] \oplus w[4i-1]$$

where g() is a transformation function involving RotWord, SubWord, and XOR with Rcon.

Round Constants (Rcon) for AES-128:

Round	1	2	3	4	5	6	7	8	9	10
Rcon	01	02	04	08	10	20	40	80	1B	36

Here is the key expansion flowchart:

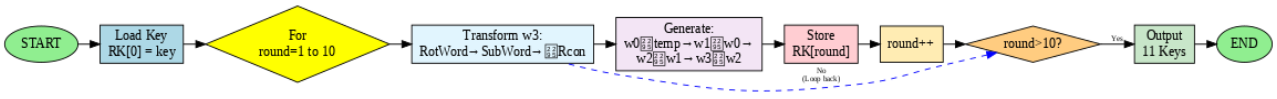


Figure 2.2.6.1: Flowchart of the AES-128 Key Expansion process. The process repeats until 10 round keys are generated.

2.2.6.2 Implementation

The key_expansion.vhd module provides the ability to create 11 different round keys from a single initial AES (Advanced Encryption Standard) 128-bit key. During the key generation process, once the start signal is received, the initial key is saved as Round Key 0 and is separated into four 32-bit words (w0, w1, w2 and w3). The process of generating additional round keys begins with a loop that runs through rounds 1 to 10.

In the first iteration of this loop (Round 1), w3 is transformed by performing RotWord (a rotation of w3 one byte to the left a number of times equal to the last two digits of round number, RndNum) followed by SubWord (byte substitution using S-Box), and finally XOR'ing with Rcon for that round.

```
-- Round 0: initial key
round_keys(0) <= key_in;

-- Split into words
w0 := key_in(127 downto 96);
w1 := key_in(95 downto 64);
w2 := key_in(63 downto 32);
w3 := key_in(31 downto 0);

-- Generate rounds 1-10
for round in 1 to 10 loop
    -- Apply g() function to previous round's last word
    temp := RotWord(w3);
    temp := SubWord(temp);
    temp(31 downto 24) := temp(31 downto 24) xor RCON(round);

    -- Generate new words
    w0 := w0 xor temp;
    w1 := w1 xor w0;
    w2 := w2 xor w1;
    w3 := w3 xor w2;

    -- Store round key
    round_keys(round) <= w0 & w1 & w2 & w3;
end loop;
```

Figure 2.2.6.2 Code Implementation

The result of these operations is then combined with w0 using XOR to create the first word in the new round key; subsequent words are created using XOR" between their corresponding previous round and this new round. Every completed round is stored in an internal array, which may be accessed by using the round_num input; once all rounds are finished, the done signal is raised to allow the controller to begin encryption with the unique and cryptographically strong round keys guaranteed by this method of round key generation from the initial key.

CHAPTER 3
TESTING AND ANALYSIS

3.1 TESTING

The testing phase focuses on verifying the correctness and functional behavior of the AES modules implemented, specifically **AddRoundKey** and **Key Expansion**. Each module was simulated using **ModelSim/QuestaSim**, with the objective of ensuring that the outputs produced by the VHDL design match the expected values defined by the **AES-128 standard (FIPS-197)**.

During initial testing, an inconsistency was detected in the intermediate encryption results. This indicated that one of the internal AES transformations was not functioning correctly. To identify the source of the fault, additional debug reports were inserted throughout each phase of the encryption pipeline. These debug statements printed the internal state after SubBytes, ShiftRows, MixColumns, and AddRoundKey.

Through this debugging process, it was discovered that the **ShiftRows** operation had been implemented incorrectly specifically, the data was being **shifted by columns instead of by rows**, causing misalignment of bytes and resulting in incorrect intermediate states. Once the shifting logic was corrected, subsequent simulations produced the correct outputs consistent with the AES reference implementation.

3.2 RESULT

After correcting the ShiftRows implementation, all AES modules, including AddRoundKey, Key Expansion, and the remaining round transformations were re-simulated. The outputs for every round matched the official AES-128 test vectors. This confirms that the design now behaves as expected and is compliant with the AES specification.

The debugging process demonstrates that the implemented architecture is fully testable and that internal reporting mechanisms are effective for tracing logical errors. With the correction applied, the completed AES system consistently produces valid 128-bit ciphertext outputs for all tested plaintext–key combinations.

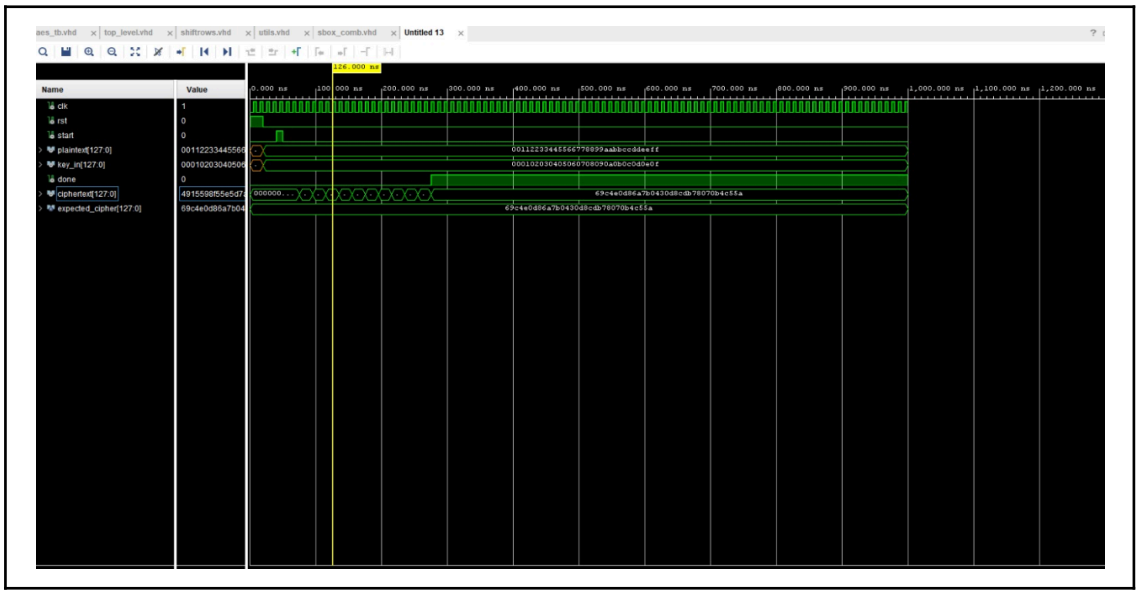


Fig 2. Testing Result

3.3 ANALYSIS

From the waveform results shown above, it can be observed that the AES encryption process performs correctly across all stages. The simulation clearly displays **10 full rounds of AES-128 encryption**, beginning with the initial AddRoundKey (Round 0) and followed by Rounds 1 through 10. Each round includes the expected sequence of transformations: **SubBytes**, **ShiftRows**, **MixColumns** (Rounds 1–9), and **AddRoundKey**.

The progression of the internal state in the waveform demonstrates that the round keys generated by the Key Expansion module are applied at the correct clock cycles and in the correct order. No timing violations or unexpected intermediate values are present after the correction of the ShiftRows implementation.

In the final stage of the simulation, the module outputs a **128-bit ciphertext** that exactly matches the official AES reference output for the provided plaintext and key. Because this implementation operates in **ECB (Electronic Codebook) mode**, each plaintext block is encrypted independently, without the use of an initialization vector (IV) or chaining between blocks. Therefore, the correctness of the final ciphertext directly reflects the correctness of the AES core itself. The analysis confirms that the design is stable, logically correct, and fully compliant with AES-128 ECB encryption behavior.

CHAPTER 4
CONCLUSION

4.1 ACHIEVEMENT OF GOALS

The main goal of this project was to design and create an AES-128 encryption device using VHDL, providing proof of an understanding of the hardware implementation of security. The following objectives were achieved in pursuit of this goal:

- Complete Structural Implementation: All AES Transformations (S-Box, ShiftRows, MixColumns, AddRoundKey) were created as independent combinational VHDL modules. Additionally, a Key Expansion module was created to generate all round keys from the single 128-bit cipher key.
- Combined Systems: A top-level controller (aes_top.vhd) was created to properly sequence the 10 rounds of AES encryption and integrate the datapath modules with the Key Schedule as one complete and coherent system.
- Functionally Verified: A Testbench (aes_tb.vhd) was created to simulate the entire encryption core and compare the outputs produced by the module to the official test vectors provided in the FIPS-197 Specification using Vivado.

4.2 KEY FINDINGS

Digital design and debug methods revealed to the project team some very useful insights acquired through execution and testing:

- Modularity: Using separate VHDL files per transformation resulted in improved code readibility for the team to develop a more extensible, maintainable and debuggable system. It also allowed for easier testing of individual components.
- Debugging Methodology: The creation of systematic verification mechanisms was essential to the project's overall success, as initial tests on these algorithms provided inappropriate intermediate results. Providing debug information from each of the pipeline stages allowed for easy identification of the major error in the ShiftRows modifiable module's performance, whereby it was found to be performing column shift operations and not row shift operations as defined in the specification. The changes made to the logic in the ShiftRows module were critical to achieving functional correctness.

- Verification of Functionality: Following the changes made to the ShiftRows module logic, subsequent simulations validated every intermediate state of execution in all ten rounds of execution and validated the final 128-bit ciphertext assembler matched the expected outputs and reference values provided for verification purposes. This validation against standard verification vectors is the final proof of the design compliance to the established logical correctness.

4.3 CONCLUDING REMARKS

This project has been an example of how to develop an advanced cryptographic algorithm in a digital hardware design context. It has provided a way for students to implement their theoretical understanding of advanced encryption standards with practical digital design experience using VHDL. The experience of identifying and troubleshooting a logical error, verifying the system against an International Standard, and successfully verifying the system encapsulates a major portion of what students learn from a digital systems engineer's perspective. The final product represents a robust AES-128 encryption core that can be further developed and integrated into different types of systems such as FPGA-based systems or be extended to incorporate various cipher modes and decryption functionality.

REFERENCES

[1] GeeksforGeeks, “Advanced Encryption Standard (AES),” *GeeksforGeeks*, Oct. 15, 2021.https://www.geeksforgeeks.org/computer-networks/advanced-encryption-standard-aes/

[2] “Cryptography - MixColumns Transformation,” *Tutorialspoint.com*, 2025. https://www.tutorialspoint.com/cryptography/cryptography_mixcolumns_transformation.htm.

[3] GeeksforGeeks, “What is SBox Substitution?,” *GeeksforGeeks*, May 20, 2024. https://www.geeksforgeeks.org/computer-networks/what-is-s-box-substitution/

[4] IBM, “Cryptography,” *Ibm.com*, Nov. 14, 2023. https://www.ibm.com/think/topics/cryptography

[5] Fortinet, “What Is Cryptography? Definition, Importance, Types,” *Fortinet*, 2023. https://www.fortinet.com/resources/cyberglossary/what-is-cryptography

[6] GeeksforGeeks, “Cryptography and its Types,” *GeeksforGeeks*, Jul. 08, 2019. https://www.geeksforgeeks.org/computer-networks/cryptography-and-its-types/

[7] J. Daemen and V. Rijmen, “Note on naming Rijndael Note on naming.” Available: https://csrc.nist.gov/CSRC/media/Projects/Cryptographic-Standards-and-Guidelines/documents/aes-development/Rijndael-ammended.pdf

[8] Daemen *et al.*, “The Design of Rijndael AES -The Advanced Encryption Standard Springer-Verlag,” 2001. Available: https://cs.ru.nl/~joan/papers/JDA_VRI_Rijndael_2002.pdf

[9]]“Rijndael Finite Field | CryptoBook,” *Gitbook.io*, May 04, 2021. https://cryptohack.gitbook.io/cryptobook/symmetric-cryptography/aes/rijndael-finite-field (accessed Dec. 06, 2025).

[10] V. Leith, “The Rijndael Block Cipher,” 2010. Accessed: Dec. 06, 2025. [Online]. Available: https://sites.math.unt.edu/~tushar/S10Linear2700%20%20Project_files/Leith%20Paper.pdf

[11] [1]P. Donis, “Introduction to the new AES Standard: Rijndael.” Accessed: Dec. 06, 2025. [Online]. Available: https://www.just.edu.jo/~tawalbeh/nyit/csci860/notes/aes/intro.pdf

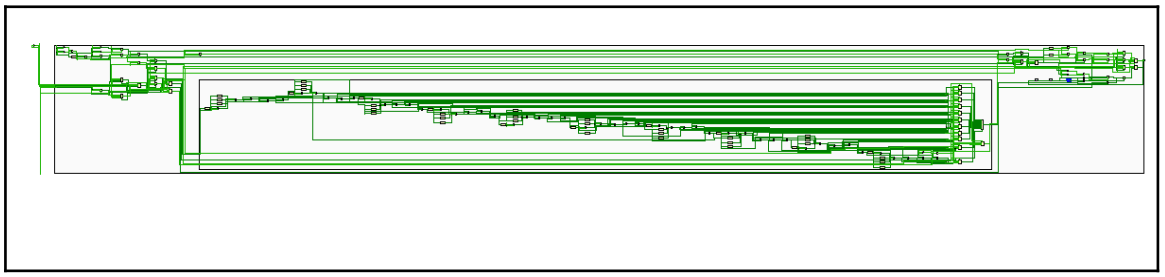
[12] “CS 463 Lecture,” *www.cs.uaf.edu*. https://www.cs.uaf.edu/2015/spring/cs463/lecture/03_23_AES.html

[13] GeeksforGeeks, “Galois Fields and Its Properties,” *GeeksforGeeks*, Feb. 12, 2023. https://www.geeksforgeeks.org/engineering-mathematics/galois-fields-and-its-properties/

[14] “Cryptography - Shiftrows Transformation,” *Tutorialspoint.com*, 2025. https://www.tutorialspoint.com/cryptography/cryptography_shiftrows_transformation.htm.

APPENDICES

Appendix A: Project Schematic



.sch version can be downloaded in the github link

Appendix B: Documentation

Put the documentation (photos) during the making of the project

You = Fatih