Simulation and Synthesis of Mini-AES: A Verilog-based Approach

Richard Groves
Graduate Student, Department of Electrical and Computer Engineering
Boise State University
Boise, Idaho, USA
RichardGroves197@u.boisestate.edu

Abstract—**This paper explores the simulation and synthesis of Simplified Advanced Encryption Standard (Mini-AES), a streamlined version of the AES algorithm. The project involved designing and testing various encryption components using Verilog, with a focus on achieving a parameterized number of encryption rounds. Emphasis was placed on module and function design, including NibbleSub, ShiftRow, MixCollumn Key Addition and a controller to step through the encryption process. The results demonstrate Mini-AES's potential as an educational tool, offering insights into its design efficiency and encryption reliability.**

Index Terms—**Mini-AES, Data Encryption, Security, Algorithm, Cryptography**

I. Introduction

The topic of cryptography has been key in securing digital communications and data. Among its various algorithms, the Advanced Encryption Standard (AES) stands out because of its widespread adoption and robust security features. However, AES's complexities can be challenging to understand and implement, especially for projects designed for students. To address this, we explore Mini-AES, a simplified version of AES designed for educational purposes. This paper focuses on the simulation and synthesis of Mini-AES using Verilog, a hardware description language. This study aims to provide learning opportunities of the structural and operational nuances of AES, making it more accessible for academic and research pursuits. By leveraging Verilog, we simulate the intricate processes of Mini-AES, offering a detailed perspective on its cryptographic mechanisms and potential applications in teaching and understanding encryption standards.

II. Background

A. AES Overview

The Advanced Encryption Standard (AES) has been a staple in cryptographic security since its establishment by the National Institute of Standards and Technology (NIST) in 2000. AES is renowned for its robust security, supported by a block size of 128 bits and key sizes of 128, 192, or 256 bits. The intricacies of AES, with its complex structure and multiple rounds of encryption, make it a challenging subject for students. [1]

B. Mini-AES Rationale

To bridge this educational gap, Mini-AES was introduced. Mini-AES is a simplified version of AES, designed primarily for educational purposes, retaining the essential structure of AES but with significantly reduced parameters. Unlike AES, Mini-AES operates with a smaller block size and fewer rounds, making it an excellent tool for understanding the basic principles of AES in a more manageable context. This simplification aids in grasping cryptographic concepts, preparing students for the complexities of full-scale AES. The primary purpose of Mini-AES is to serve as a testbed for cryptographic education, rather than for real-world security applications. [1]

III. Mini-AES Structure

Mini-AES simplifies the complex structure of AES, adapting it for educational purposes. It operates on 16-bit blocks, compared to AES's 128-bit blocks, and uses a 16-bit key. Mini-AES maintains the fundamental structure of AES but reduces the complexity. The Mini-AES process includes transformations like NibbleSub, ShiftRows, and MixColumns, akin to AES, but tailored to the smaller block and key size. NibbleSub is a substitution step using a simplified S-Box, ShiftRows rearranges the bits in the block, and MixColumns involves polynomial multiplication. This streamlined structure of Mini-AES makes it an excellent tool for demonstrating the basic principles of AES in a more accessible format. [1]

1) **Initialization Vector Addition:** Start by XORing the plaintext and the initialization vector.
2) **Initial Key Addition:** Next, use the sum of the initialization vector and plaintext and XOR it with the initial round key (round key 0). This is sometimes referred to as the pre-round transformation.
3) **Round Transformations:** Perform the following steps for a fixed number of rounds, denoted as Nr:
    a) **NibbleSub:** Apply the NibbleSub transformation, a non-linear substitution step where each byte of the block is replaced with another according to the S-Box.
    b) **ShiftRows:** Apply the ShiftRows transformation, where each row of the block is shifted cyclically to the left by a certain number of positions.

c) **MixColumns:** (Only if the current round is not the last round.) Apply the MixColumns transformation, which is a column-wise operation that mixes the bytes within each column.

d) **Add Round Key:** XOR the block with the round key for that round.

4) **Final Round:** In the final round, perform the SubBytes and ShiftRows transformations, followed by XORing the block with the final round key, omitting the MixColumns step.
5) **Output Ciphertext:** The result after the final round key addition is the ciphertext.

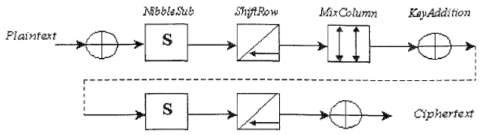
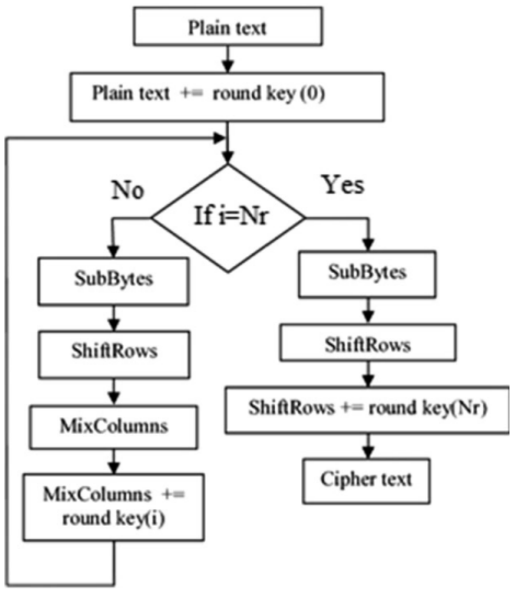


Fig. 1. Mini-AES Dataflow [1]



Plain text
Plain text += round key (0)
No
Yes
If i=Nr
SubBytes
ShiftRows
MixColumns
MixColumns += round key(i)
SubBytes
ShiftRows
ShiftRows += round key(Nr)
Cipher text

Fig. 2. Mini-AES Datapath [1]

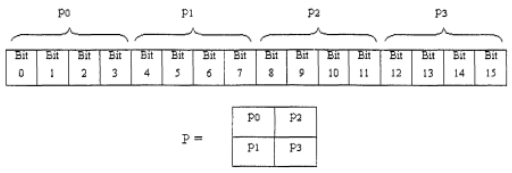


Fig. 3. 16-bit Block Represented as Nibble Matrix [1]

Input	Output	Input	Output
0000	1110	1000	0011
0001	0100	1001	1010
0010	1101	1010	0110
0011	0001	1011	1100
0100	0010	1100	0101
0101	1111	1101	1001
0110	1011	1110	0000
0111	1000	1111	0111

Fig. 4. Substitution Table (S-Box) [1]

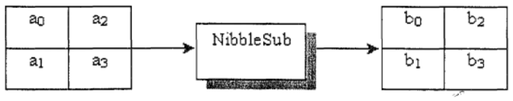


Fig. 5. Nibble Substitution [1]

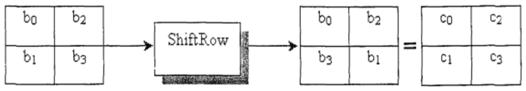
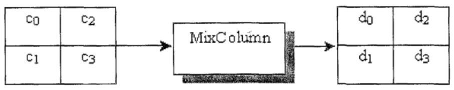


Fig. 6. Row Shift [1]

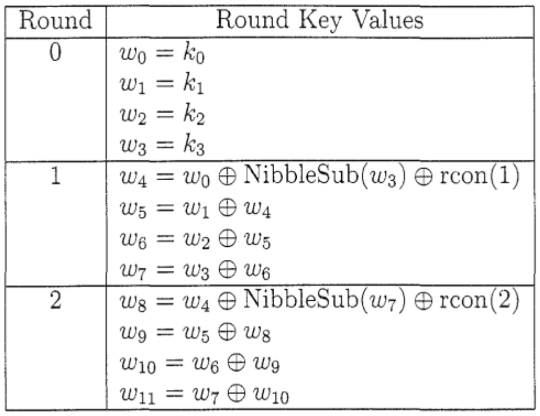


where $\begin{bmatrix} d_0 \\ d_1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}$ and $\begin{bmatrix} d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} c_2 \\ c_3 \end{bmatrix}$

Fig. 7. Column Mix [1]

$$\begin{array}{|c|c|}\hline d_0 & d_2 \\ \hline d_1 & d_3 \\ \hline \end{array} \oplus \begin{array}{|c|c|}\hline k_0 & k_2 \\ \hline k_1 & k_3 \\ \hline \end{array} = \begin{array}{|c|c|}\hline e_0 & e_2 \\ \hline e_1 & e_3 \\ \hline \end{array}$$

Fig. 8. Key Addition [1]



Round	Round Key Values
0	$w_0 = k_0$ $w_1 = k_1$ $w_2 = k_2$ $w_3 = k_3$
1	$w_4 = w_0 \oplus \mathrm{NibbleSub}(w_3) \oplus \mathrm{rcon}(1)$ $w_5 = w_1 \oplus w_4$ $w_6 = w_2 \oplus w_5$ $w_7 = w_3 \oplus w_6$
2	$w_8 = w_4 \oplus \mathrm{NibbleSub}(w_7) \oplus \mathrm{rcon}(2)$ $w_9 = w_5 \oplus w_8$ $w_{10} = w_6 \oplus w_9$ $w_{11} = w_7 \oplus w_{10}$

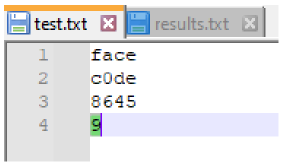
Fig. 9. Key Generation [1]

IV. Methodology

A. Verilog-based Simulation

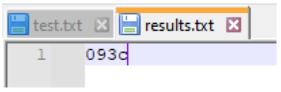
The simulation of Mini-AES was meticulously carried out using Verilog. As each module and function was completed, it was tested to ensure its correct operation before being integrated into the larger system. The testbench module, `tb_AES_Cntr`, was integral to this process, simulating the AES encryption process by feeding test vectors into the `AES_Cntr` module. This module encompassed essential components like NibbleSub and ShiftRow, implemented as functions rather than separate modules, allowing for a more integrated and efficient design. File I/O operations were employed for reading input vectors and writing the resulting ciphertext to files, offering a dynamic and easy-to-use testing environment. Continuous verification after each development step ensured the reliability and correctness of the simulation from the ground up.

The nine rounds of encryption were performed in 116 clock cycles. The majority of the cycles were used for polynomial multiplication as part of the `Mix_Column` module.



```
face
c0de
8645
9
```

Fig. 10. Simulation Inputs (plaintext, Input Key, Initialization Vector, Number of Rounds)



```
093c
```

Fig. 11. Simulation Output (Encrypted Text)

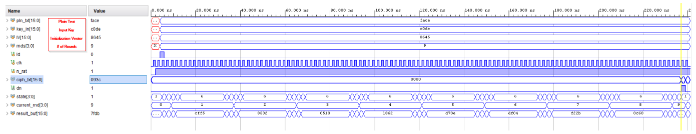
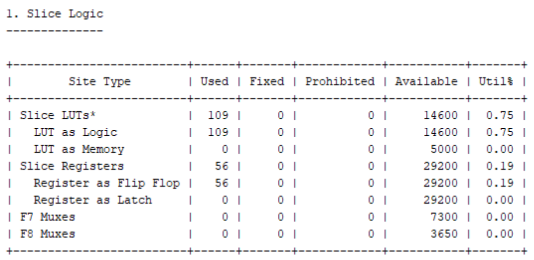


Fig. 12. Simulation Waveform

B. Synthesis and Performance Analysis

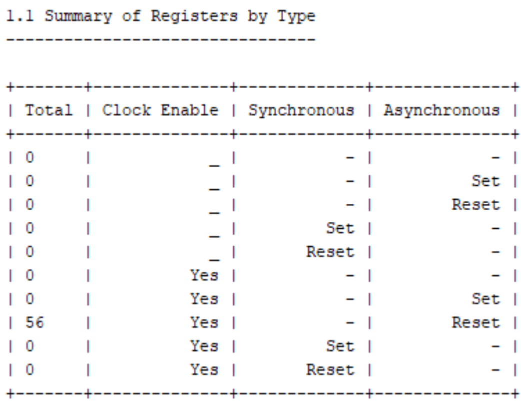
After simulation, the Verilog code was synthesized, converting the high-level design into a digital logic representation. This process was useful for assessing the design's practical feasibility and performance. Synthesis offered insights into key information like resource utilization, which is necessary for evaluating the algorithm's efficiency.



```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             |  109 |     0 |          0 |     14600 |  0.75 |
|   LUT as Logic          |  109 |     0 |          0 |     14600 |  0.75 |
|   LUT as Memory         |    0 |     0 |          0 |      5000 |  0.00 |
| Slice Registers         |   56 |     0 |          0 |     29200 |  0.19 |
|   Register as Flip Flop |   56 |     0 |          0 |     29200 |  0.19 |
|   Register as Latch     |    0 |     0 |          0 |     29200 |  0.00 |
| F7 Muxes                |    0 |     0 |          0 |      7300 |  0.00 |
| F8 Muxes                |    0 |     0 |          0 |      3650 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

Fig. 13. Utilized Slices



```
1.1 Summary of Registers by Type
--------------------------------

+-------+--------------+-------------+--------------+
| Total | Clock Enable | Synchronous | Asynchronous |
+-------+--------------+-------------+--------------+
| 0     |            _ |           - |            - |
| 0     |            _ |           - |          Set |
| 0     |            _ |           - |        Reset |
| 0     |            _ |         Set |            - |
| 0     |            _ |       Reset |            - |
| 0     |          Yes |           - |            - |
| 0     |          Yes |           - |          Set |
| 56    |          Yes |           - |        Reset |
| 0     |          Yes |         Set |            - |
| 0     |          Yes |       Reset |            - |
+-------+--------------+-------------+--------------+
```

Fig. 14. Utilized Registers

C. Functional Verification and Testing

Functional verification combined file-based input testing with output analysis. The use of functions for key cryptographic operations facilitated thorough testing and verification of the Mini-AES functionality. This approach was vital for ensuring the implementation's accuracy and reliability across various encryption scenarios.

D. Future Work

Looking forward, there are several avenues for further development and research. One potential area is the exploration of different encryption modes and their implementation in

Verilog. Additionally, there is the potential for developing a decryption counterpart for the Mini-AES, which would involve reversing the encryption process to provide a complete educational toolset. This could also lead to a deeper understanding of the decryption process and the challenges associated with it. Furthermore, extending the Mini-AES project to design a full-scale AES system could offer valuable insights. It would provide practical experience in implementing encryption systems in hardware for real-world applications

V. Conclusion

This project demonstrated the simulation and synthesis of Mini-AES using Verilog, providing an understanding of its structural and operational principles. The project highlighted the effectiveness of Mini-AES as an educational tool, simplifying the complexities of the AES algorithm for academic purposes. The testing and verification process ensured the accuracy and reliability of the simulation. Future research could explore more advanced features of AES or apply similar methodologies to other cryptographic algorithms, further enhancing the educational value of such simulations in cryptography.

References

[1] R. C.-W. Phan, ”Mini Advanced Encryption Standard (Mini-AES): A Testbed for Cryptanalysis Students,”.