2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)
Karnataka, India. Jun 28-29, 2024

2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS) | 979-8-3503-8269-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICITEICS61368.2024.10625542

Design and Implementation of Efficient Block Cipher

Korkonda Srivaishnavi
BTech Student
Dept.of. ECE
Gokaraju Rangaraju Institute of Engineering and Technology
Hyderbad,India.
vaishnukorkonda03@gmail.com

Gorigapudi Sowmya
BTech Student
Dept.of. ECE
Gokaraju Rangaraju Institute of Engineering and Technology
Hyderbad,India.
sowmyea.2002@gmail.com

P Laxmikanth
BTech Student
Dept.of. ECE
Gokaraju Rangaraju Institute of Engineering and Technology
Hyderbad,India.
laxmikanthcharry@gmail.com

Manchalla O V P Kumar
Dept of ECE
Gokaraju Rangaraju Institute of Engineering and Technology.
Hyderabad,India
pavanomkar@gmail.com

Kiran Mannem
Dept of ECE
Gokaraju Rangaraju Institute of Engineering and Technology.
Hyderabad,India
kiranmannem14@gmail.com

***Abstract*—This paper introduces a novel block cipher architecture incorporating S-box and P-box structures. The cipher is implemented using the Verilog hardware description language and synthesized on the Xilinx Vivado FPGA platform. S-box and P-box components are crucial for ensuring robust encryption and are meticulously designed to enhance security and efficiency. The design methodology, implementation specifics, and performance metrics of the proposed block cipher are detailed. Experimental results on the Xilinx Vivado platform showcase the cipher's efficacy in terms of security and resource utilization, underscoring its potential for secure data encryption across diverse applications.**

***Index Terms*—Block cipher, S-box, P-box, Verilog, Xilinx Vivado, FPGA, Encryption**

I. Introduction

Block ciphers are fundamental cryptographic primitives widely used for securing data in various applications. The design and implementation of efficient block ciphers have become increasingly important, especially for resource-constrained environments such as embedded systems and wireless networks. In recent years, there has been a growing demand for lightweight ciphers that can provide strong security while minimizing resource consumption. This paper addresses this need by proposing a novel block cipher architecture that leverages S-box and P-box structures to achieve a balance between security and efficiency. The cipher is implemented using the Verilog hardware description language and targeted for the Xilinx Vivado FPGA platform, making it suitable for deployment in hardware-constrained environments.

The use of S-box and P-box structures in block ciphers is well-established, with these components playing a crucial role in the encryption and decryption processes. S-boxes are nonlinear substitution functions that introduce confusion in the cipher, making it resistant to cryptanalysis techniques such as differential and linear cryptanalysis. P-boxes, on the other hand, are used for permutation of bits to further enhance the diffusion properties of the cipher. By carefully designing these components and integrating them into the cipher architecture, we aim to achieve a high level of security while minimizing the hardware resources required for implementation. In situations with limited resources, such as embedded systems and the Internet of Things, cryptography is essential to maintaining data integrity, secrecy, and user privacy. In these kinds of situations, lightweight ciphers are essential because they provide a careful balance between security and computational efficiency.

PRESENT, KATAN/KTANTAN, LED, Piccolo, and RECTANGLE are popular lightweight ciphers that are carefully designed to provide strong security and minimize resources. These ciphers are critical to the security of applications in pervasive computing, wireless network communication, cyber-physical systems, and IoT/CPS frameworks.

II. LITERATURE REVIEW

The usage of lightweight block ciphers in resource-constrained embedded devices like RFIDs, sensor nodes, and smart cards, as well as in wireless networks and low-cost cryptosystems has led to a surge in demand for them in recent years (1). One such cipher is RECTANGLE, which supports the bit-slice approach and provides an effective lightweight architecture appropriate for numerous platforms and hardware-constrained situations (3). Performance of the synthesized and implemented RECTANGLE architecture on a Xilinx Virtex-5 FPGA is equivalent to that of previous architectures (1). Furthermore, 2362 gate equivalents (GE) are used by an ASIC

implementation of RECTANGLE using SCL 180 nm CMOS technology (1). When comparing block ciphers, one finds that AES is no longer appropriate for some networks, including those that include RFID, sensors, and Internet of Things devices (5). In order to counter this, the PRESENT cipher provides an extremely light block figure that is important for hardware planning as well as security in these kinds of situations (5).

Another popular block cipher is AES, for which an effective VLSI design has been presented for its cryptography in memory implementation (2). This tactic provides increased cryptographic strength without changing the algorithm's optimal compatibility with the distributed standard or its frequency of work (2). By exploiting NVM's intrinsic logic to jumble memory regions only when necessary, the AES implementation in memory (AIM) eliminates the need for extra processing steps (2). The PRESENT block cipher design holds great significance for lightweight cryptography in embedded devices and the Internet of Things (4). PRESENT is an incredibly thin block cipher that strikes a compromise between security and hardware needs in order to solve the security issues that IoT devices confront (4). Furthermore, a hardware-light architecture for PRESENT has been suggested to address the growing need for security in computer domains such as Internet of Things (4).

Lightweight cryptographic algorithms that can enable secure communication between devices with constrained resources, like Internet of Things devices, are the focus of S-box implementations (6). In some cases, parallel S-box implementation can result in better performance, which emphasizes how important it is to choose the right security primitives for the right applications (6).

III. IMPLEMENTATION

First, construct the S-box and P-box circuits as lookup tables (LUTs) in Xilinx Vivado to implement a block cipher in Verilog. Next, develop the logic for round key generation, which produces round keys using the key schedule algorithm of the encryption. Subsequently, carry out the primary encryption/decryption procedures, utilizing the round keys for several rounds of substitution, permutation, and XOR operations. Create a top-level module that manages control signals and input/output data and instantiates these parts. Lastly, use Vivado's simulation tools to confirm functionality. S-box lookup, P-box permutation, and XOR with the round key are possible fundamental rounds of the encryption, with Verilog logic representing each operation.

A. Software Setup

Installing the Vivado Design Suite from the Xilinx website is the first step in configuring the software environment for building a block cipher in Verilog using Xilinx Vivado. Make that your computer satisfies all of Vivado's system requirements, including having enough RAM and disk space. After installation, launch Vivado and start a new project, naming it and indicating the target device (such as an FPGA or SoC) and location.

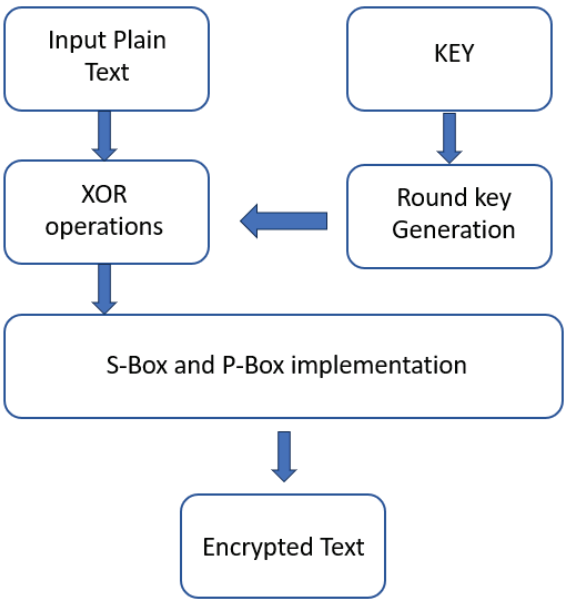

Fig. 1. Workflow of the project

Next, update the project containing the Verilog source file for your block cipher implementation. S-box, P-box, round key generation, and the primary encryption/decryption logic should all have Verilog codes written for them. For coding, use an external editor or the built-in text editor in Vivado.Pin assignments for input/output signals and time limitations for the design can be specified by adding constraints using the Xilinx limitations Editor (XDC). In order to implement the design and create the netlist, place and route the logic onto the target device. This will create the bitstream needed for device configuration. Lastly, synthesize the design.

B. Input Parameters

To create a block cipher using an 80-bit cryptographic key and 64-bit plaintext, it is necessary to modify pre-existing algorithms like DES or 3DES. These algorithms require 56-bit or 168-bit keys, respectively, and are intended for 64-bit blocks. Changes to the key scheduling procedure are needed to produce the round keys needed to support the 80-bit key. It may be necessary to add padding to the plaintext handling in order to accommodate the 64-bit block size.

To summarize, DES or 3DES adaptation for an 80-bit key and 64-bit plaintext entails altering the key scheduling procedure and controlling the plaintext to match the block size. Using these algorithms in situations where certain input size criteria must be met is made possible by the modifications that guarantee security and compatibility throughout the encryption/decryption process.

C. Round key Generation

The key schedule technique of the DES (Data Encryption Standard) cipher can be used to produce round keys from an 80-bit cryptographic key for a block cipher implementation. Using the PC-1 permutation table, this approach first converts the 80-bit key into a 56-bit key. Next, the 56-bit key is split into two 28-bit parts.

Depending on the round number, the algorithm rotates each half of a round key separately by one or two bits. Following rotation, the two parts are joined, compressed, and permuted using the PC-2 permutation table to create a 48-bit round key. A set of round keys for the encryption and decryption procedures are created by repeating this method for each round of the block cipher.

D. Round Operations

Each round of a block cipher for an 80-bit cryptographic key and 64-bit plaintext usually consists of three primary operations: P-box permutation, S-box substitution, and XOR.

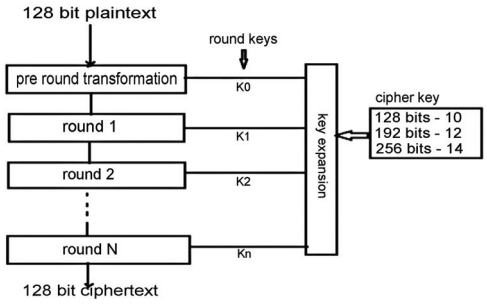

Fig. 2. Round Operations

- **XOR Operation**: A round key generated from the cryptographic key is used to XOR the plaintext. By combining important information with the plaintext, this process creates misunderstanding.The round key and plaintext are combined to form the ciphertext in a block cipher, and here is where the XOR process comes into play. The 64-bit plaintext is first split up into smaller chunks, usually 32 bits each, and these chunks go through a number of changes per round. Every round of encryption starts with the round key being XORed with the plaintext block. In order to create confusion and make the encryption process non-linear and more safe, this bitwise XOR technique makes sure that each bit of the plaintext is joined with the appropriate bit of the round key.S-box substitution and P-box permutation are applied to the resultant output in order to further conceal the data. The ciphertext is XORed with the round key to retrieve the original plaintext in the decryption process, which also makes use of the XOR function. All things considered, the XOR operation is a key component of block ciphers and greatly enhances both their security and efficacy when it comes to data encryption.

- **S-box Substitution**: After the XOR operation, the result is split up into smaller pieces, usually consisting of six bits each. A value is substituted for each part of the result by selecting it from an S-box. Non-linearity is added to the encryption process using S-boxes.An essential part of a block cipher that gives the encryption process non-linearity is the S-box (Substitution Box). Based on a predetermined substitution table, the S-box typically replaces a specified number of bits from the input data with a new set of bits. Each S-box of the DES (Data Encryption Standard), for instance, receives six bits of input and outputs four bits. This substitution is predicated on the precise values found in the table of the S-box, which are intended to guarantee that slight modifications in the input bits yield notable modifications in the output bits.The S-box creates confusion and makes it difficult to determine the relationship between the plaintext and the ciphertext without knowing the secret key, thwarting cryptographic attacks like differential and linear cryptanalysis.

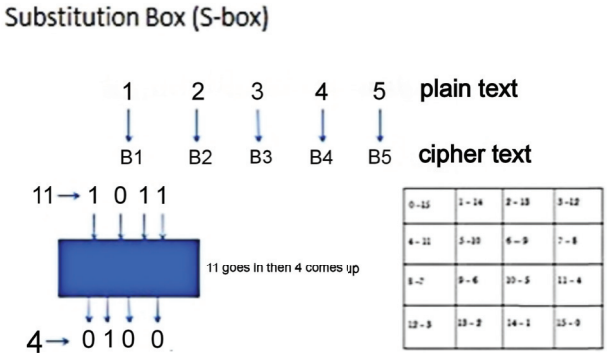

Fig. 3. Substitution Box

- **P-box Permutation**: Lastly, the bits are rearranged using a P-box permutation. Prior to proceeding to the following round, the data is further jumbled using this permutation step.The P-box, or permutation box, is a part of a block cipher that arranges the data bits in a particular way. After the S-box substitution in every round, it is typically applied. To achieve confusion and diffusion—two necessary qualities for strong encryption—the P-box rearranges the data bits. The P-box, for instance, is a predetermined permutation used in the Data Encryption Standard (DES) that rearranges the 32 bits of data that are output from the S-boxes. The new locations of each bit in this permutation are determined by a particular table.In order to increase the complexity of the encryption process and make it more difficult for attackers to decipher or predict the encryption algorithm, the P-box makes sure that every bit of the output from the S-boxes affects multiple bits in the subsequent round.

RESULTS

The Xilinx Vivado block cipher project uses an 80-bit cryptographic key and 64-bit plaintext as inputs to create a

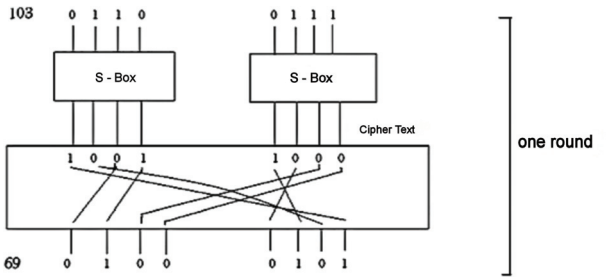

Fig. 4. P-box layer Operation

ciphertext. There are several rounds in the encryption process, and each one consists of three primary operations: P-box permutation, S-box substitution, and XOR.Every round, a round key generation technique is used to generate a round key from the cryptographic key, which is then XORed with the plaintext. An S-box lookup table is then used to substitute the result, mapping each 6-bit input to a 4-bit output. A P-box

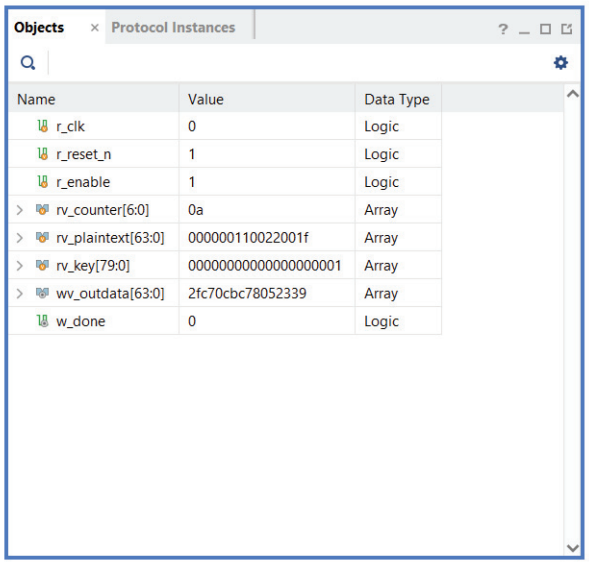

Fig. 5. Objects

permutation is then used to rearrange the bits in the S-box output in accordance with a predetermined pattern. The result of all rounds is the ciphertext, which is the plaintext modified using the cryptographic key.Simulations were performed after the Verilog code for the block cipher project was executed on Xilinx Vivado in order to confirm the design's functionality and performance.In the simulations, several plaintext inputs and cryptographic keys were supplied to the design, and the associated ciphertext outputs were observed.The XOR, S-box, and P-box operations were correctly implemented by the design at each round of the block cipher, as the simulations verified. The simulations also showed that each round's expected round keys were generated using the round key generation algorithm. All things considered, the simulations confirmed that the block cipher architecture functioned as planned, yielding the anticipated ciphertexts for various plaintext inputs and cryptographic keys.

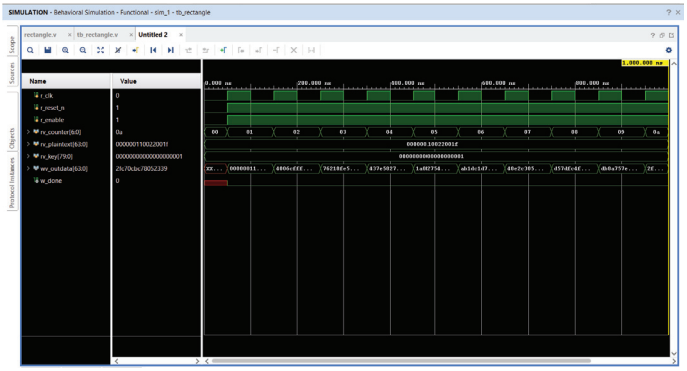

Fig. 6. Simulation

The encryption design's power consumption profile was made visible by the power analysis, which also identified high-power usage regions that might be modified for greater energy efficiency. The area analysis showed the actual footprint of

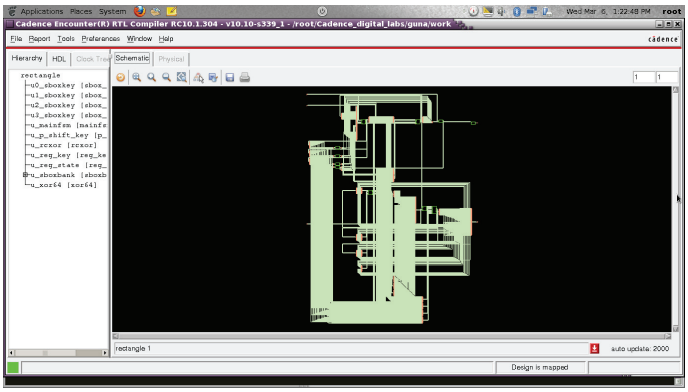

Fig. 7. Schematic(Cadence)

the cipher implementation and offered insights into how the FPGA's resources were used. The time delay analysis also examined the cipher's latency, indicating how long it took to finish the encryption operation. Understanding the trade-offs between power consumption, space usage, and processing speed in the context of the block cipher project is made easier with the help of this analysis report, which also provides insights for future optimizations and design enhancements.

library	Total Power(nW)	Total Area	Delay (ps)	Arrival (ps)
1	679508.786	16459.027	+732	3490 R
2	87136.140	4146.106	+490	2313 R
3	240674.472	16459.027	+2608	10548 R

Fig. 8. Analysis Report

CONCLUSION AND FUTURE SCOPE

In conclusion, the Xilinx Vivado block cipher project effectively illustrated how round operations like XOR, S-box, and P-box are used in the encryption process. The research succeeded in generating a ciphertext by executing several rounds of these processes using plaintext and a cryptographic key as inputs. Data encryption that is both safe and effective is made possible by the Verilog code, which was created and simulated to guarantee the cipher's proper operation and performance. The project's future scope might be increased to incorporate new features like support for various encryption algorithms, block sizes, and decryption capabilities. Enhancing the speed and effectiveness of the cipher implementation may require additional tuning. The project might also be expanded to include hardware implementation for real-time encryption applications on FPGA devices. All things considered, the block cipher project has a great deal of room to grow and improve in order to satisfy the changing requirements of secure data encryption.

REFERENCES

[1] J. G. Pandey, A. Laddha and S. D. Samaddar, "A Lightweight VLSI Architecture for RECTANGLE Cipher and its Implementation on an FPGA," 2020 24th International Symposium on VLSI Design and Test (VDAT), Bhubaneswar, India, 2020, pp. 1-6, doi: 10.1109/VDAT50263.2020.9190623. keywords: Ciphers;Registers;Schedules;Clocks;Computer architecture;Field programmable gate arrays;Encryption;Lightweight cryptography;Block ciphers;RECTANGLE;VLSI architecture;FPGA

[2] Bijjam.Swathi, Manchalla.O.V.P.Kumar, G.Marlin Sheeba, M.Kiran, Y.Sudarsana Reddy, "An Efficient VLSI Design of AES Cryptography in Memory Implementation",International Journal of Recent Technology and Engineering (IJRTE),November 2019. keywords:AES Algorithm, Verilog HDL, FPGA, MEMORY UNIT.

[3] Zhang, Wentao, et al. "RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms." Cryptology ePrint Archive (2014).

[4] G. Sravya, Manchalla. O.V.P. Kumar, G. Merlin Sheeba, K. Jamal, Kiran Mannem,Hardware lightweight design of PRESENT block cipher, Materials Today: Proceedings,Volume 33, Part 7,2020,Keywords: PRESENT cipher; Ultra-light weight; Less area; Cryptography; Iot

[5] G. Sravya, M. O. V. P. Kumar, Y. Sudarsana Reddy, K. Jamal and K. Mannem, "The Ideal Block Ciphers - Correlation of AES and PRESENT in Cryptography," 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), Thoothukudi, India, 2020, pp. 1107-1113, doi: 10.1109/ICISS49785.2020.9315883. keywords: Ciphers;Cryptography;Registers;Encryption;Sensors;Hardware;Conferences; PRESENT block cipher;AES cipher;Ultra-light weight;Area;Timing;Cryptography;IoT

[6] Changle, A. S., S. P. Metkar, and R. K. Patole. "Implementation of S-box for lightweight block cipher." 2023 3rd International Conference on Intelligent Technologies (CONIT). IEEE, 2023.

[7] Christy, M. Anitha, et al. "Design and implementation of low power advanced encryption standard S-Box using pass transistor XOR-AND logic." 2014 International Conference on Electronics and Communication Systems (ICECS). IEEE, 2014.

[8] Zhang, Runtong, and Like Chen. "A block cipher using key-dependent S-box and P-boxes." 2008 IEEE International Symposium on Industrial Electronics. IEEE, 2008.

[9] Prathiba, A., and VS Kanchana Bhaaskaran. "Lightweight S-box architecture for secure internet of things." Information 9.1 (2018): 13.

[10] 1Saravanan, P., et al. "An Efficient ASIC Implementation of CLEFIA Encryption/Decryption Algorithm With Novel S-Box Architectures." 2019 IEEE 1st International Conference on Energy, Systems and Information Processing (ICESIP). IEEE, 2019.

[11] Gangadari, Bhoopal Rao, and Shaik Rafi Ahamed. "Design of cryptographically secure AES like S-Box using second-order reversible cellular automata for wireless body area network applications." Healthcare technology letters 3.3 (2016): 177-183.

[12] Kazlauskas, Kazys, and Jaunius Kazlauskas. "Key-dependent S-box generation in AES block cipher system." Informatica 20.1 (2009): 23-34.

[13] Kazlauskas, Kazys, Gytis Vaicekauskas, and Robertas Smaliukas. "An algorithm for key-dependent S-box generation in block cipher system." Informatica 26.1 (2015): 51-65.