International Invention of Scientific Journal
Online ISSN: 2457-0958
Available Online at http://www.iisj.in **Volume 8, Issue 02(April-May-June)|2024**|Page: 66-83

Original Research Paper-Electrical Engineering

Comprehensive Analysis and Exploration of Design Space for Hardware Implementation of Advanced Encryption Standard (AES) Algorithm on FPGA Platform

Authors:
Navid Vaziri[1*], Mirza Kouchaki[1].
[1]*Department of Electrical Engineering, Iran University of Science and Technology (IUST), Tehran, Iran*
Corresponding Author:
Navid Vaziri
Department of Electrical Engineering, Iran University of Science and Technology (IUST), Tehran, Iran, Email: Navid.Vaziri1360@gmail.com

Article Received: 18-April-2024 Revised: 08-May-2024 Accepted: 28-May-2024

ABSTRACT:
This research delves into the analysis and exploration of various design implementations of the AES encryption algorithm using Vivado software. Twenty-eight executions encompassing different implementation designs, employing resource sharing techniques and pipelining, are presented. Through synthesis and implementation, metrics such as static and dynamic power consumption, maximum circuit operating frequency, resource utilization, the number of slices per module, operational circuit efficiency, etc., are reported. To evaluate the results, four merit coefficients are considered, namely the ratio of operational efficiency to the number of consumed slices and dynamic power consumption, and the ratio of the maximum circuit operating frequency to the number of slices and dynamic power consumption. The highest operational efficiency and top two merit coefficients are obtained for various scenarios using pipelining and not utilizing resource sharing techniques. The highest third and fourth merit coefficients, along with the highest operating frequency, are achieved when employing both pipelining and resource sharing simultaneously. The optimal design is one that, considering trade-offs between metrics, can achieve the best results in terms of dynamic power consumption, resource utilization, operating frequency, and operational efficiency.

Keywords: Encryption, Synthesis, Maximum Operating Frequency, Circuit Operational Efficiency, Merit Coefficient

INTRODUCTION:

In contemporary times, the growth of lightweight, robust, and effective encryption algorithms is essential for ensuring network security in various information technology applications. Consequently, the Advanced Encryption Standard (AES) has been developed. This advanced encryption standard is an efficient secure mechanism that performs better than other symmetric key encryption algorithms by maintaining the confidentiality of messages. Identity verification, confidentiality, and integrity are considered as crucial objectives for encryption protocols. AES fulfills fundamental security objectives such as availability, confidentiality, and integrity throughout communication in insecure transmission media. Due to its resistance against brute-force attacks, AES has been widely implemented on various hardware platforms, including Graphics Processing Units (GPUs), Embedded Processors, Application-Specific Integrated Circuits (ASIC), and Field-Programmable Gate Arrays (FPGA) [1]. The AES encryption involves a sequence of operations, including AddRoundKey, SubBytes, ShiftRows, and MixColumns. The decryption process comprises a similar sequence of operations and functions, with the only difference being that decryption involves a process of calculating inverses [2].

Recently, numerous articles have focused on the efficient implementation of AES on FPGA. For instance, authors in [3] present an efficient implementation leading to high operational throughput, suitable for applications requiring speed and high performance. Furthermore, authors in [4] and [5] investigate pipeline techniques, sub-pipelining, and loop unrolling to enhance the frequency and operational throughput of AES execution on FPGA. Farashahi and colleagues provide a high-speed hardware implementation of the AES algorithm on Xilinx Virtex-5, achieving an operational throughput of 86 gigabits per second and a theoretical maximum frequency of 671.5241 megahertz [6]. Two different methods based on search tables, namely Substitution-Permutation Network (SPN) and T-box, are employed for effective FPGA design. SPN-based encryption and decryption are not only memory-intensive but also asymmetric. Consequently, separate designs for encryption and decryption processes are implemented, occupying a significant amount of Block RAM (BRAM) in SPN-based AES. However, achieving clock speed and operational throughput in AES design is challenging due to its complexity and dynamic nature [7].

Some common architectures are outlined as follows. Sheikhpour and Mahani [8] developed a 32-bit AES encryption/decryption for the Internet of Things (IoT) and resource-constrained applications. A cost-effective and error-resistant structure for the data path was devised. Subsequently, an expanded key processing unit for encryption/decryption processes was designed, minimizing the area used by sharing resources among encryption and decryption operations.

Zodpe and Sapkal [9] presented a Random Sequence Number Generator (PNSG) to generate S-box and initial keys for encryption/decryption. Linear Feedback Shift Register (LFSR) with an initial state and feedback specified by a polynomial generator was used for PNSG design. The encryption strength increased, but the non-pipelined design requires substantial hardware resources. In [10], an AES_q encryption was introduced for securing the TCP/IP protocol. To enhance the conventional AES, a Boolean Expression (BE) of column mixing using gate substitution and resource-sharing structure was employed. An optimized AES architecture was obtained to minimize power consumption. However, time complexity needs to be minimized to avoid delays during communication using the TCP/IP protocol. Kumar et al. [11] designed the Lightweight AES algorithm (LAES) on Artix-7 and Kintex-7 FPGAs for securing audio data. The necessary operations for column mixing in the LAES algorithm were reduced compared to the regular AES algorithm. This reduction led to lower delays, utilized fewer logical operations, and reduced the complexity of the LAES algorithm during audio data encryption. However, the reduction in MixColumn resulted in increased use of multiplexers.

Wagner and colleagues [12] implemented a substitution box using the Rotational Symmetry function. It was designed with internal multiplexers and registers, requiring no Block RAM (BRAM). Boolean coverage in decryption was applied to enhance resistance against attacks. A masked AES design was utilized to optimize the execution of the search table. However, the replication of linear operators and their independent functioning increased the overall area of AES. Kumar [13] developed the MPPRM architecture for designing the substitution byte transformation and its inverse. As a result, hardware resources such as AND and XOR gates were reduced using MPPRM. A 128-bit key was generated by the key expansion structure and applied to the sub-pipelined data structure. High-speed encryption/decryption in AES was achieved by using a delay module at the output of the AND gate. However, due to key repetition in the encryption process, the area usage increased.

The existing methods are analyzed and explained as follows: For improved AES execution, hardware resources (such as slices and search tables) are less required during the encryption/decryption process. However, when the MixColumn operation is minimized in LAES, the demands for multiplexers increase [11]. The repetition of linear operations leads to an increase in the required area for AES in encryption/decryption processes [12]. Additionally, hardware resources for AES improve when designed in a non-pipelined manner. The delay in the MixColumn operation in regular AES is higher [14].

The AES algorithm can be implemented using pipeline, semi-pipeline, and non-pipeline techniques. Non-pipeline implementations lead to area optimization at the expense of reduced speed. The execution of pipeline and sub-pipeline techniques results in increased operational throughput and increased area usage since the pipeline technique allows processing multiple blocks simultaneously [15]. In this study, the exploration of the design space for AES-128 encryption has been conducted. Twenty-eight different implementation designs are executed, and aspects such as power consumption, circuit operating frequency, efficiency, and resource utilization are reported. By examining the results, the best implementations, achieving good results while maintaining a trade-off between the mentioned criteria, are introduced. The aim of this research is a comprehensive analysis and exploration of the design space for the hardware implementation of the Advanced Encryption Standard (AES) algorithm on the FPGA platform. In this regard, various combinations of pipeline implementations and hardware reuse techniques are considered. Subsequently, the results are compared based on resource utilization, operating frequency, output efficiency, and power consumption.

METHODS:

The AES cipher employs a block algorithm with a fixed block size of 128 bits and a variable key length of 128/192/256 bits, requiring 10/12/14 rounds for a complete operation. The architecture of AES includes encryption and decryption processes. The encryption and decryption rounds for an AES-128 cipher are illustrated in Fig. 1. The complete encryption of a plaintext requires ten rounds. Each round of the encryption algorithm (except the tenth round) comprises four logical operations. The MixColumns operation is not performed in the last round to ensure reversibility during decryption. Similarly, the decryption process consists of ten rounds, serving as the inverse of the encryption for generating the decrypted plaintext.

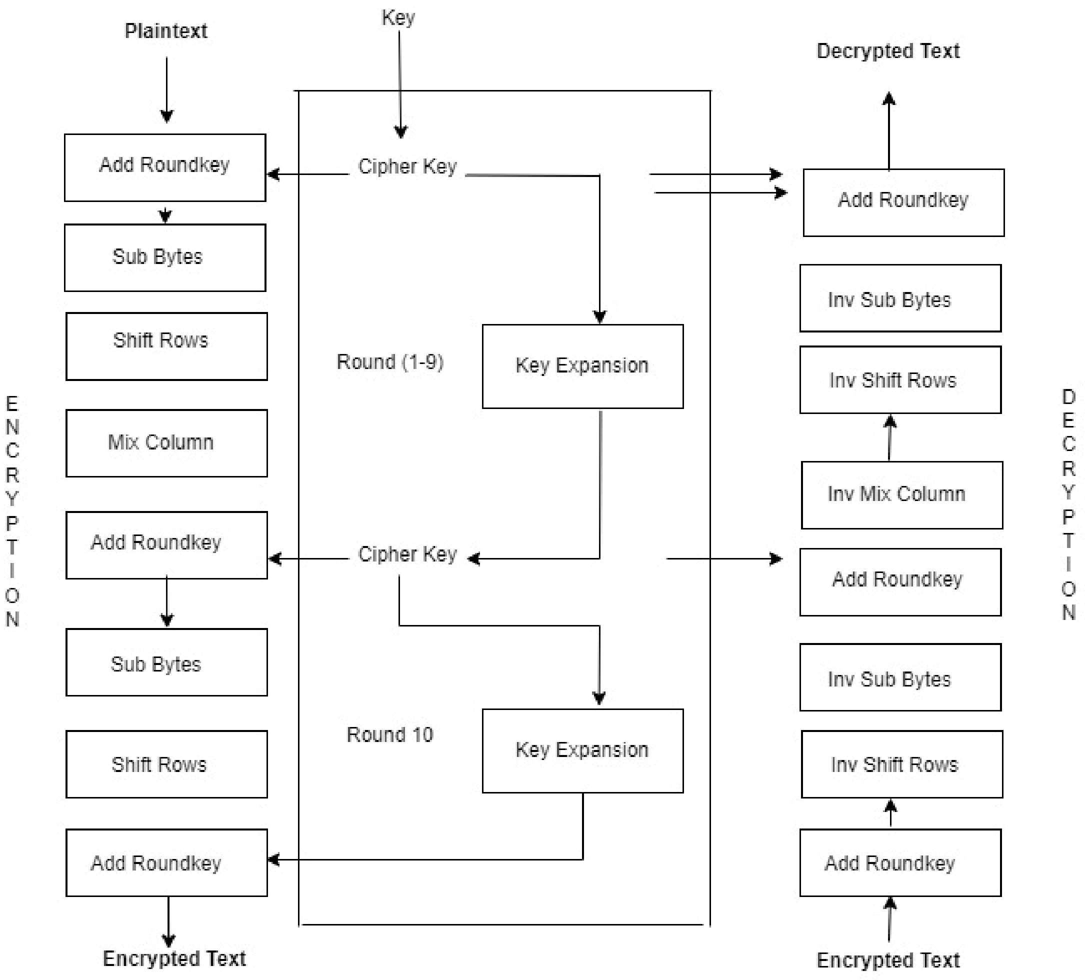


Fig. 1 AES Encryption and Decryption Flowchart

The encryption process begins with the addition of the key for the initial round. The plaintext, XORed with an initial round key, undergoes encryption. Processed data then undergoes multiple rounds of operations to generate the ciphertext. The result after each round is recognized as a state. The state is an array of bytes, forming a 4x4 byte matrix since the block size is 128 bits (16 bytes).

Mathematical Implementation of AES Decryption:

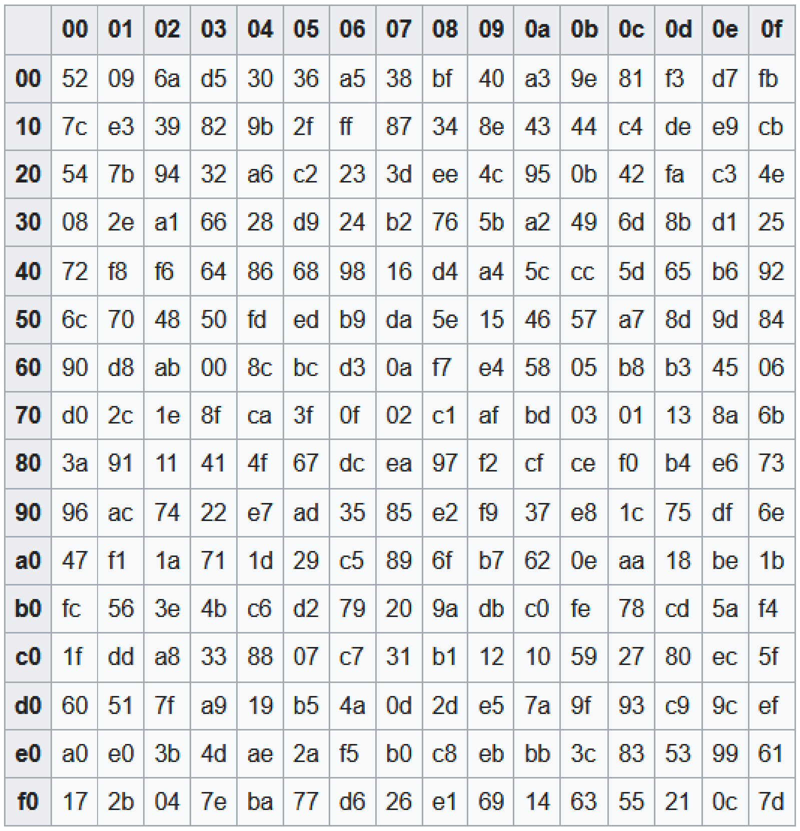
The inverse operations of row permutation, inverse substitution, key addition, and inverse column mixing are performed in order.

Inverse Row Permutation:

Depending on the row number, each row of the matrix is cyclically shifted to the right. The first row remains unchanged, and the second, third, and fourth rows are shifted by 1, 2, and 3 positions to the right, respectively.

$$\begin{bmatrix} a0,0 & a0,1 & a0,2 & a0,3 \\ a1,0 & a1,1 & a1,2 & a1,3 \\ a2,0 & a2,1 & a2,2 & a2,3 \\ a3,0 & a3,1 & a3,2 & a3,3 \end{bmatrix} ==> \begin{bmatrix} a0,0 & a0,1 & a0,2 & a0,3 \\ a1,3 & a1,0 & a1,1 & a1,2 \\ a2,2 & a2,3 & a2,0 & a2,1 \\ a3,1 & a3,2 & a3,3 & a3,0 \end{bmatrix}$$

Inverse Byte Substitution:
This is a non-linear transformation where a byte is replaced with a value from the inverse substitution box (S-box). The inverse substitution box is used to substitute data. In 8-bit data, the first 4 bits represent the row, and the last 4 bits represent the column. The byte substitution method for a block is to replace the 8-bit data with the specified row and column indices. The decryption substitution box is illustrated in hexadecimal format in Fig. 2.



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

Fig. 2 Inverse S-box in Hexadecimal Format

Adding Round Key:
Adding the round key is the same for both encryption and decryption. Here, 16 bytes (128 bits) are considered, and they are XORed with the round key, as illustrated in Fig. 3.

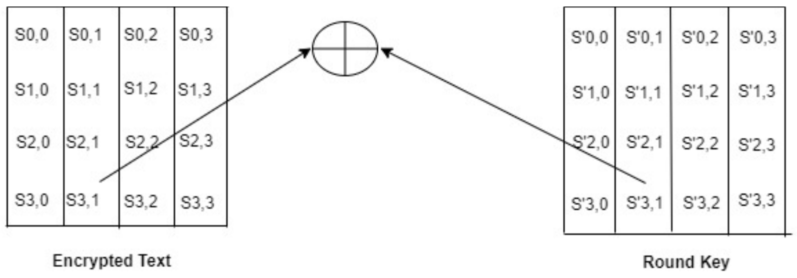


Fig. 3 Adding Round Key

Inverse Mix Column:
Consider the inverse state matrix as follows:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0F & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

Here, each row is multiplied in the column to obtain the inverse mixed column (S') as shown in the equation.

$$\begin{bmatrix} S0c \\ S1c \\ S2c \\ S3c \end{bmatrix} \cdot \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0F & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} = \begin{bmatrix} S'0c \\ S'1c \\ S'2c \\ S'3c \end{bmatrix}$$

Therefore, matrix multiplication with a predefined matrix for decryption, for example, the output of the previous transformation function in the decryption process, yields the new state matrix in the decryption process as indicated in the equation.

New State Matrix == State Matrix Predefined Matrix

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0F & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \cdot \begin{bmatrix} S0,0 & S0,1 & S0,2 & S0,3 \\ S0,1 & S1,1 & S1,2 & S1,3 \\ S0,2 & S1,2 & S2,2 & S2,3 \\ S0,3 & S1,3 & S2,3 & S3,3 \end{bmatrix} = \begin{bmatrix} S'0,0 & S'0,1 & S'0,2 & S'0,3 \\ S'0,1 & S'1,1 & S'1,2 & S'1,3 \\ S'0,2 & S'1,2 & S'2,2 & S'2,3 \\ S'0,3 & S'1,3 & S'2,3 & S'3,3 \end{bmatrix}$$

Pipeline Architecture Considered:

The architecture, depicted in Fig. 4, includes a data register between each round for sharing processed data. Each round key is generated and stored in a separate register. By implementing the pipeline structure in the following architecture, the processing cycle time is reduced, and the operational capability of the system is increased.

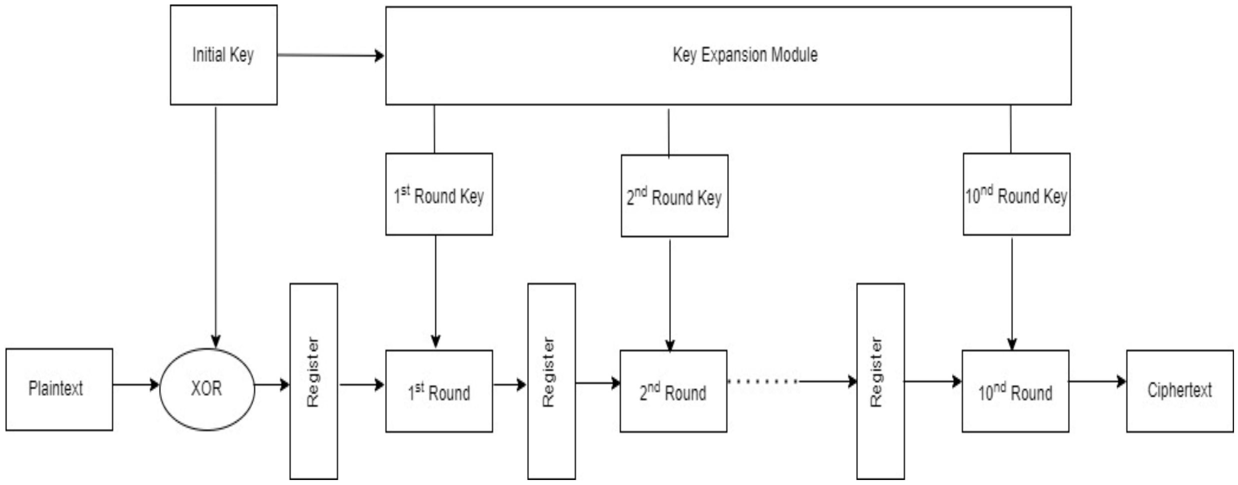


Fig. 4 Pipeline Structure for AES Algorithm

Resource Sharing Technique:

This technique, also known as hardware reuse and resource sharing, efficiently shares hardware resources between the encryption and decryption processes. The goal of using this technique is to minimize the utilized area.

A summary of the implementation of pipelined AES on FPGA includes the following:

1. Input and Output: The FPGA design takes plaintext inputs and keys and produces the corresponding ciphertext as output.
2. Pipeline Stages: The AES algorithm is divided into several stages, where each stage performs a specific operation of the AES encryption or decryption process. These stages are implemented as modules or separate blocks in the FPGA design.
3. State Registers: Internal registers (state) are used to store intermediate values at each stage of the pipeline.
4. Round Keys: Round keys are derived from the initial encryption key and are used at each stage of the pipeline. The FPGA design generates and updates round keys as needed for each round of the encryption algorithm.
5. Parallel Processing: FPGA utilizes its parallel processing capabilities to perform multiple encryption operations simultaneously.
6. Clock Synchronization: Pipeline stages are synchronized with a common clock signal to ensure proper sequencing and coordination of encryption operations.
7. Resource Utilization: Resources such as lookup tables and flip-flops are effectively used for executing pipeline stages and managing data flow.

Simulation Descriptions:

In this project, the exploration of the design space of the AES encryption algorithm and the presentation of various implementation methods have been addressed.

To achieve this, techniques such as pipelining and resource sharing have been employed. The codes were executed in the Vivado software using VHDL language and on the Virtex-7 FPGA chip.
The file naming convention is in the form of AES_PR(X)_P{Y)_HR(Z).
In this naming convention, PR is used to specify the presence or absence of pipeline registers between the four different stages of one round out of the 10 rounds of AES encryption, and the number X represents the count of these registers.
If X = 0, it means no registers have been used within each round of the algorithm. If X = 1, it means one register between ShiftRow and MixColumn has been used. If X = 2, it means one register between ShiftRow and MixColumn and one register between MixColumn and AddRoundKey have been used.
If X = 3, it means that a register has been used between all stages (four stages) of each round.
P indicates the presence or absence of a register between different rounds. If Y = 0, it means there is no register, and if it is 1, it means a register has been placed between consecutive rounds.
HR represents the number of hardware resources (modules used). If Z = 10, it means there is no resource sharing, and for 10 rounds, 10 separate modules have been used.
If Z = 2, it means that for 10 rounds, two modules have been used. Therefore, each of these two modules must be used 5 times, which is equivalent to using 10 modules.
If Z = 5, it means that for 10 rounds, 5 modules have been used. Therefore, each of these 5 modules must be used 2 times. If Z = 1, it means that for 10 rounds, 10 modules have been used. In the case of Z = 1, the condition Y = 1 is no longer meaningful because only one module is present. As a result, we will have 28 different implementation scenarios. By implementing and coding these 28 scenarios, a search was conducted in the design space, and the obtained results were examined and analyzed. The green color in Table 1 indicates cases where both resource sharing techniques and internal and intermediate pipeline registers have been used in the design. For example, in PR0_P0_HR1, the reuse of resources has occurred 10 times, and the design lacks internal and intermediate pipeline registers. In Table 1, a brief description of 28 different implementation scenarios is provided.

Row	**Descriptions**	**PR(X)_P(Y)_HR(Z)**
1	No intermediate and internal round registers	0_0_1
2	No intermediate and internal round registers	0_0_2
3	No intermediate and internal round registers	0_0_5
4	No intermediate and internal round registers, no resource sharing	0_0_10
5	No internal round registers.	0_1_2
6	No internal round registers.	0_1_5
7	No internal round registers, no resource sharing	0_1_10
8	No intermediate round registers	1_0_1
9	No intermediate round registers	1_0_2
10	No intermediate round registers	1_0_5
11	No intermediate round registers, no resource sharing	1_0_10
12		1_1_2
13		1_1_5
14	No resource sharing	1_1_10
15	No intermediate round registers	2_0_1
16	No intermediate round registers	2_0_2
17	No intermediate round registers	2_0_5
18	No register between rounds and no sharing of resources	2_0_10
19		2_1_2
20		2_1_5
21	No resource sharing	2_1_10
22	No intermediate round registers	3_0_1
23	No intermediate round registers	3_0_2
24	No intermediate round registers	3_0_5
25	No intermediate round registers, no resource sharing	3_0_10
26		3_1_2
27		3_1_5
28	No resource sharing	3_1_10

Code Review:

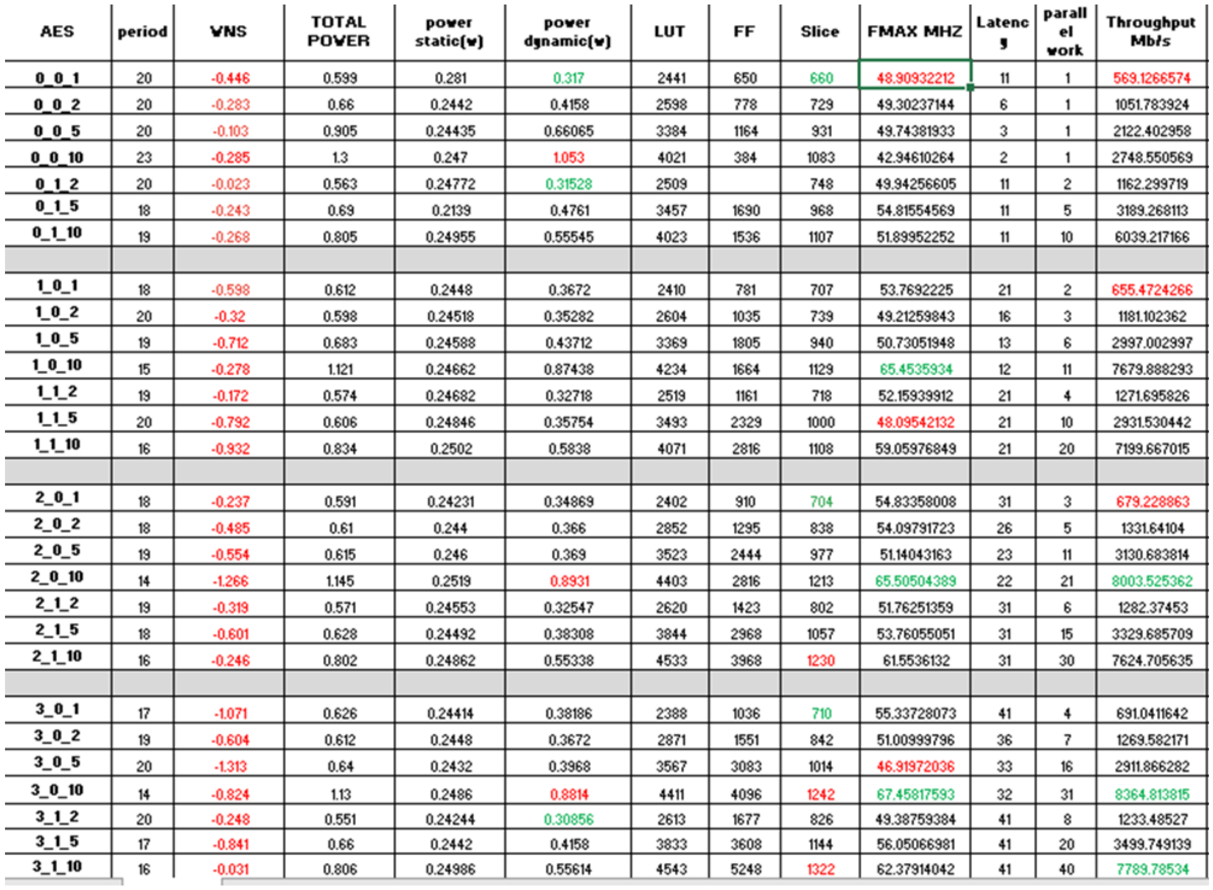
For each of the 28 mentioned scenarios, a folder containing VHDL codes has been provided. For each scenario, there are codes named AES_Sbox, Key_Expansion, round, top, and top_test. In the byte substitution stage, it is possible to use a pre-existing RAM and implement the Sbox function by addressing it and reading the output value. However, this memory requires a large volume. Therefore, instead of this approach, calculations for the substitution box have been coded in the AES_Sbox code.

RESULTS:

In this project, a search in the design space of the AES encryption algorithm has been conducted, and various implementations have been presented using pipeline techniques and resource sharing. The execution time of the code is considered to be 1000 nanoseconds (one microsecond). The clock cycle period is set to 10000 picoseconds (10 nanoseconds). The encryption is AES_128, so the plaintext, key, and ciphertext are all 128 bits, with bit numbers ranging from 0 to 127. The input data and encryption key are assumed to be the same for all 28 implementations, resulting in the same ciphertext. Table 2 displays the input data, encryption key, and ciphertext.

Table 2. Display of Input Data, Encryption Key, and Ciphertext.

Input Data	3243f6a8885a308d313198a2e0370734
Encryption Key	2b7e151628aed2a6abf7158809cf4f3c
Ciphertext	3902dc1925dc116a409850b1dfb9732



AES	period	WNS	TOTAL POWER	power static(w)	power dynamic(w)	LUT	FF	Slice	FMAX MHZ	Latency	parallel work	Throughput Mb/s
0_0_1	20	-0.446	0.599	0.281	0.317	2441	650	660	48.90932212	11	1	569.1266574
0_0_2	20	-0.283	0.66	0.2442	0.4158	2598	778	729	49.30237144	6	1	1051.783924
0_0_5	20	-0.103	0.905	0.24435	0.66065	3384	1164	931	49.74381933	3	1	2122.402958
0_0_10	23	-0.285	1.3	0.247	1.053	4021	384	1083	42.94610264	2	1	2748.550569
0_1_2	20	-0.023	0.563	0.24772	0.31528	2509		748	49.94256605	11	2	1162.299719
0_1_5	18	-0.243	0.69	0.2139	0.4761	3457	1690	968	54.81554569	11	5	3189.268113
0_1_10	19	-0.268	0.805	0.24955	0.55545	4023	1536	1107	51.89952252	11	10	6039.217166
1_0_1	18	-0.598	0.612	0.2448	0.3672	2410	781	707	53.7692225	21	2	655.4724266
1_0_2	20	-0.32	0.598	0.24518	0.35282	2604	1035	739	49.21259843	16	3	1181.102362
1_0_5	19	-0.712	0.683	0.24588	0.43712	3369	1805	940	50.73051948	13	6	2997.002997
1_0_10	15	-0.278	1.121	0.24662	0.87438	4234	1664	1129	65.4535934	12	11	7679.888293
1_1_2	19	-0.172	0.574	0.24682	0.32718	2519	1161	718	52.15939912	21	4	1271.695826
1_1_5	20	-0.792	0.606	0.24846	0.35754	3493	2329	1000	48.09542132	21	10	2931.530442
1_1_10	16	-0.932	0.834	0.2502	0.5838	4071	2816	1108	59.05976849	21	20	7199.667015
2_0_1	18	-0.237	0.591	0.24231	0.34869	2402	910	704	54.83358008	31	3	679.228863
2_0_2	18	-0.485	0.61	0.244	0.366	2852	1295	838	54.09791723	26	5	1331.64104
2_0_5	19	-0.554	0.615	0.246	0.369	3523	2444	977	51.14043163	23	11	3130.683814
2_0_10	14	-1.266	1.145	0.2519	0.8931	4403	2816	1213	65.50504389	22	21	8003.525362
2_1_2	19	-0.319	0.571	0.24553	0.32547	2620	1423	802	51.76251359	31	6	1282.37453
2_1_5	18	-0.601	0.628	0.24492	0.38308	3844	2968	1057	53.76055051	31	15	3329.685709
2_1_10	16	-0.246	0.802	0.24862	0.55338	4533	3968	1230	61.5536132	31	30	7624.705635
3_0_1	17	-1.071	0.626	0.24414	0.38186	2388	1036	710	55.33728073	41	4	691.0411642
3_0_2	19	-0.604	0.612	0.2448	0.3672	2871	1551	842	51.00999796	36	7	1269.582171
3_0_5	20	-1.313	0.64	0.2432	0.3968	3567	3083	1014	46.91972036	33	16	2911.866282
3_0_10	14	-0.824	1.13	0.2486	0.8814	4411	4096	1242	67.45817593	32	31	8364.813815
3_1_2	20	-0.248	0.551	0.24244	0.30856	2613	1677	826	49.38759384	41	8	1233.48527
3_1_5	17	-0.841	0.66	0.2442	0.4158	3833	3608	1144	56.05066981	41	20	3499.749139
3_1_10	16	-0.031	0.806	0.24986	0.55614	4543	5248	1322	62.37914042	41	40	7789.78534

Fig. 5 Simulation Results for 28 Different Implementations of Pipelining and Resource Sharing in AES Encryption

In Fig. 5, for each of the 28 states PR(X)_P(Y)_HR(Z), parameters such as WNS, total power consumption, static power, dynamic power, operational efficiency, maximum circuit operating frequency, latency, number of flip-flops, number of lookup tables, and number of slices are reported.

WNS is used to indicate the correct timing for data output. The period is the time period in which WNS becomes negative. According to the Vivado guide, for more accurate parameter recording, WNS should be negative.

The maximum circuit operating frequency is in MHz and is obtained from the following relationship:

F_{max} = 1000/ (period – WNS)

Static power is specific to the chip being used and is relatively constant across different implementation states. Therefore, in the 28 executions, static power consumption remains relatively constant.

Dynamic power is the instantaneous power of the circuit and depends on the voltage level and logic and variable resources.

Troput, or operational efficiency, is essentially the encoding rate and is reported in megabits per second. In AES-128, it is obtained from the following relationship.

Throughput = (F_{max}*128*parallel work) / latency

Parallel work refers to the concurrent execution of tasks, which occurs as a result of pipelining.

The lookup table (LUT) in the search table is used to determine the logic of the output and represents the combinational logic of the circuit. The number of LUTs affects the number of consumed slices. The number of slices in the utilized region and the final circuit size are influential. Flip-flops are single-bit memory cells used to store the circuit state. The number of flip-flops and search tables provide an indication of the consumed resources, but for the purpose of comparing the 28 implementations, the required number of slices for each is also presented in the "slice" column. Power consumption parameters, resource utilization, and lower latency are desirable, and higher operating frequency and operational efficiency result in better performance. However, these factors are not independent, and reliance on a single criterion is not advisable. An ideal implementation should have both operational efficiency and power consumption within acceptable ranges.

In Table 3, the values for total power consumption, static power, and dynamic power are presented. Static power is related to the board and is relatively independent of different implementation scenarios. Therefore, in the 28 executions, static power consumption remains relatively constant at around 0.25 watts.

Table 3. Total Power Consumption, Static Power, and Dynamic Power.

Dynamic Power Consumption	**Static Power Consumption**	**Total Power Consumption**	
0.317	0.281	0.599	0_0_1
0.4158	0.2442	0.66	0_0_2
0.66065	0.24435	0.905	0_0_5
1.053	0.247	1.3	0_0_10
0.31528	0.24772	0.563	0_1_2
0.4761	0.2139	0.69	0_1_5
0.55545	0.24955	0.805	0_1_10
0.3672	0.2448	0.612	1_0_1
0.35282	0.24518	0.598	1_0_2
0.43712	0.24588	0.683	1_0_5
0.87438	0.24662	1.121	1_0_10
0.32718	0.24682	0.574	1_1_2
0.35754	0.24846	0.606	1_1_5
0.5838	0.2502	0.834	1_1_10
0.34869	0.24231	0.591	2_0_1
0.366	0.244	0.61	2_0_2
0.369	0.246	0.615	2_0_5
0.8931	0.2519	1.145	2_0_10
0.32547	0.24553	0.571	2_1_2
0.38308	0.24492	0.628	2_1_5
0.55338	0.24862	0.802	2_1_10
0.38186	0.24414	0.626	3_0_1
0.3672	0.2448	0.612	3_0_2
0.3968	0.2432	0.64	3_0_5
0.8814	0.2486	1.13	3_0_10

0.30856	0.24244	0.551	3_1_2
0.4158	0.2442	0.66	3_1_5
0.55614	0.24986	0.806	3_1_10

According to Table 4, designs 0_0_10, 1_0_10, 2_0_10, and 3_0_10 have the highest dynamic power consumption, while designs 0_0_1, 0_1_2, 1_1_2, 2_1_2, and 3_1_2 exhibit the lowest dynamic power consumption.

Table 4. Comparison of Dynamic Power Consumption in 28 Implementations.

0_0_1	0.317
0_0_2	0.4158
0_0_5	0.66065
0_0_10	1.053
0_1_2	0.31528
0_1_5	0.4761
0_1_10	0.55545
1_0_1	0.3672
1_0_2	0.35282
1_0_5	0.43712
1_0_10	0.87438
1_1_2	0.32718
1_1_5	0.35754
1_1_10	0.5838
2_0_1	0.34869
2_0_2	0.366
2_0_5	0.369
2_0_10	0.8931
2_1_2	0.32547
2_1_5	0.38308
2_1_10	0.55338
3_0_1	0.38186
3_0_2	0.3672
3_0_5	0.3968
3_0_10	0.8814
3_1_2	0.30856
3_1_5	0.4158
3_1_10	0.55614

Resource consumption, including the number of lookup tables, flip-flops, and the number of slices, is reported in Table 5.

Table 5. Comparison of Resource Utilization.

	LUT	FF	Slice
0_0_1	2441	650	660
0_0_2	2598	778	729
0_0_5	3384	1164	931
0_0_10	4021	384	1083
0_1_2	2509	904	748
0_1_5	3457	1690	968
0_1_10	4023	1536	1107
1_0_1	2410	781	707
1_0_2	2604	1035	739

1_0_5	3369	1805	940
1_0_10	4234	1664	1129
1_1_2	2519	1161	718
1_1_5	3493	2329	1000
1_1_10	4071	2816	1108
2_0_1	2402	910	704
2_0_2	2852	1295	838
2_0_5	3523	2444	977
2_0_10	4403	2816	1213
2_1_2	2620	1423	802
2_1_5	3844	2968	1057
2_1_10	4533	3968	1230
3_0_1	2388	1036	710
3_0_2	2871	1551	842
3_0_5	3567	3083	1014
3_0_10	4411	4096	1242
3_1_2	2613	1677	826
3_1_5	3833	3608	1144
3_1_10	4543	5248	1322

According to Table 6, it can be observed that for a specific X and Y, an increase in Z leads to an increase in the number of consumed slices. Additionally, an increase in X, with fixed Y and Z, as well as an increase in Y with fixed X and Z, results in an increase in the number of consumed slices. In total, configurations 2_0_10, 2_1_10, 3_0_10, and 3_1_10 have the highest number of slices, while configurations 0_0_1, 1_0_1, 2_0_1, and 3_0_1 have the lowest number of slices.

Table 6. Comparison of the Number of Slices in 28 Implementation Methods.

0_0_1	660
0_0_2	729
0_0_5	931
0_0_10	1083
0_1_2	748
0_1_5	968
0_1_10	1107
1_0_1	707
1_0_2	739
1_0_5	940
1_0_10	1129
1_1_2	718
1_1_5	1000
1_1_10	1108
2_0_1	704
2_0_2	838
2_0_5	977
2_0_10	1213
2_1_2	802
2_1_5	1057
2_1_10	1230
3_0_1	710

3_0_2	842
3_0_5	1014
3_0_10	1242
3_1_2	826
3_1_5	1144
3_1_10	1322

According to Table 7, the circuit operating frequency in configurations 3_0_10, 2_0_10, and 1_0_10 is higher than the other 28 implementations. However, configurations such as 0_0_10, 1_1_5, 2_0_5, and 3_0_5 have the lowest operating frequencies.

Table 7. Comparison of Maximum Operating Frequencies in 28 Implementations.

0_0_1	48.90932212
0_0_2	49.30237144
0_0_5	49.74381933
0_0_10	42.94610264
0_1_2	49.94256605
0_1_5	54.81554569
0_1_10	51.89952252
1_0_1	53.7692225
1_0_2	49.21259843
1_0_5	50.73051948
1_0_10	65.4535934
1_1_2	52.15939912
1_1_5	48.09542132
1_1_10	59.05976849
2_0_1	54.83358008
2_0_2	54.09791723
2_0_5	51.14043163
2_0_10	65.50504389
2_1_2	51.76251359
2_1_5	53.76055051
2_1_10	61.5536132
3_0_1	55.33728073
3_0_2	51.00999796
3_0_5	46.91972036
3_0_10	67.45817593
3_1_2	49.38759384
3_1_5	56.05066981
3_1_10	62.37914042

The minimum delay, according to Table 8, is achieved in the implementation with X, Y = 0. Therefore, pipelining introduces delays in execution.

Table 8. Comparison of Delays in 28 Implementations.

0_0_1	11
0_0_2	6
0_0_5	3
0_0_10	2
0_1_2	11
0_1_5	11

0_1_10	11
1_0_1	21
1_0_2	16
1_0_5	13
1_0_10	12
1_1_2	21
1_1_5	21
1_1_10	21
2_0_1	31
2_0_2	26
2_0_5	23
2_0_10	22
2_1_2	31
2_1_5	31
2_1_10	31
3_0_1	41
3_0_2	36
3_0_5	33
3_0_10	32
3_1_2	41
3_1_5	41
3_1_10	41

According to Table 9, the configurations 2_0_10, 3_0_10, and 3_1_10 exhibit the highest operational efficiency. In executions 0_0_1, 1_0_1, 2_0_1, 3_0_1, we observe the lowest operational efficiency. Therefore, an increase in resource sharing leads to a reduction in operational efficiency.

Table 9. Comparison of Operational Efficiency in 28 Implementations.

0_0_1	569.1266574
0_0_2	1051.783924
0_0_5	2122.402958
0_0_10	2748.550569
0_1_2	1162.299719
0_1_5	3189.268113
0_1_10	6039.217166
1_0_1	655.4724266
1_0_2	1181.102362
1_0_5	2997.002997
1_0_10	7679.888293
1_1_2	1271.695826
1_1_5	2931.530442
1_1_10	7199.667015
2_0_1	679.228863
2_0_2	1331.64104
2_0_5	3130.683814
2_0_10	8003.525362
2_1_2	1282.37453
2_1_5	3329.685709
2_1_10	7624.705635

3_0_1	691.0411642
3_0_2	1269.582171
3_0_5	2911.866282
3_0_10	8364.813815
3_1_2	1233.48527
3_1_5	3499.749139
3_1_10	7789.78534

In the continuation, the four merit coefficients are presented, respectively, as the ratio of operational efficiency to the number of slices and dynamic power, and the ratio of the maximum circuit frequency to the number of slices and dynamic power. These coefficients, denoted as C.F4, C.F3, C.F2, C.F1, are shown in Table 10.
According to Table 10, the ratio of operational efficiency to the number of slices has the lowest value in configurations 0_0_1, 1_0_1, 2_0_1, 3_0_1, while the highest values are observed in configurations 0_1_10, 1_1_10, 2_1_10, 3_1_10, 1_0_10, 2_0_10, 3_0_10.

Table 10. Ratio of Operational Efficiency to the Number of Slices.

0_0_1	0.862313117
0_0_2	1.442776302
0_0_5	2.279702425
0_0_10	2.537904496
0_1_2	1.55387663
0_1_5	3.294698464
0_1_10	5.455480729
1_0_1	0.927118001
1_0_2	1.598244063
1_0_5	3.188301061
1_0_10	6.802381127
1_1_2	1.771164103
1_1_5	2.931530442
1_1_10	6.497894418
2_0_1	0.964813726
2_0_2	1.589070453
2_0_5	3.204384661
2_0_10	6.598124783
2_1_2	1.598970736
2_1_5	3.150128391
2_1_10	6.198947671
3_0_1	0.973297414
3_0_2	1.507817306
3_0_5	2.871663
3_0_10	6.734954763
3_1_2	1.493323572
3_1_5	3.059221275
3_1_10	5.892424614

According to Table 11, regarding the ratio of operational efficiency to dynamic power, it is observed that configurations 0_0_1, 1_0_1, 2_0_1, and 3_0_1 have the lowest C.F2 values. On the other hand, configurations 0_1_10, 1_1_10, 2_1_10, and 3_1_10 exhibit the highest C.F2 values.

Table 11. Ratio of Operational Efficiency to Dynamic Power.

0_0_1	1795.352231
0_0_2	2529.542867
0_0_5	3212.598135
0_0_10	2610.209467
0_1_2	3686.563432
0_1_5	6698.735797
0_1_10	10872.6567
1_0_1	1785.055628
1_0_2	3347.606038
1_0_5	6856.247705
1_0_10	8783.238743
1_1_2	3886.838518
1_1_5	8199.167763
1_1_10	12332.42038
2_0_1	1947.944773
2_0_2	3638.363496
2_0_5	8484.237979
2_0_10	8961.510875
2_1_2	3940.069838
2_1_5	8691.880831
2_1_10	13778.42646
3_0_1	1809.671514
3_0_2	3457.467787
3_0_5	7338.372686
3_0_10	9490.371926
3_1_2	3997.554027
3_1_5	8416.905096
3_1_10	14006.87838

According to Table 12, concerning the ratio of the maximum circuit frequency to the number of slices, it is observed that configurations 0_0_10, 0_1_10, and 3_0_5 have the lowest C.F3 values. On the other hand, configurations 0_0_1, 1_0_1, 2_0_1, and 3_0_1 exhibit the highest C.F3 values.

Table 12. Ratio of Maximum Circuit Frequency to Number of Slices.

0_0_1	0.074105034
0_0_2	0.067630139
0_0_5	0.053430526
0_0_10	0.039654758
0_1_2	0.066768136
0_1_5	0.05662763
0_1_10	0.046883038
1_0_1	0.076052649
1_0_2	0.066593503
1_0_5	0.053968638
1_0_10	0.057974839
1_1_2	0.072645403
1_1_5	0.048095421
1_1_10	0.05330304

2_0_1	0.077888608
2_0_2	0.064555987
2_0_5	0.052344352
2_0_10	0.054002509
2_1_2	0.064541788
2_1_5	0.050861448
2_1_10	0.050043588
3_0_1	0.077939832
3_0_2	0.060581945
3_0_5	0.046271914
3_0_10	0.054314151
3_1_2	0.059791276
3_1_5	0.047185431
3_1_10	0.048995341

According to Table 13, concerning the ratio of the maximum circuit frequency to dynamic power consumption, it is observed that configurations 0_0_10, 1_0_10, 2_0_10, and 3_0_10 have the lowest C.F4 values. On the other hand, configurations 0_1_2, 1_1_2, 2_1_2, and 3_1_2 exhibit the highest C.F4 values.

Table 13. Ratio of Maximum Circuit Frequency to Dynamic Power Consumption.

0_0_1	154.2880824
0_0_2	118.5723219
0_0_5	75.2952688
0_0_10	40.78452293
0_1_2	158.4070225
0_1_5	115.1345215
0_1_10	93.43689355
1_0_1	146.4303445
1_0_2	139.4835849
1_0_5	116.0562763
1_0_10	74.85714838
1_1_2	159.4211111
1_1_5	134.5175961
1_1_10	101.1643859
2_0_1	157.2559583
2_0_2	147.808517
2_0_5	138.5919556
2_0_10	73.34569912
2_1_2	159.0392773
2_1_5	140.3376593
2_1_10	111.2320886
3_0_1	144.9151017
3_0_2	138.9161164
3_0_5	118.245263
3_0_10	76.53525747
3_1_2	160.0583155
3_1_5	134.8019957
3_1_10	112.1644557

From the analysis of the tables, it can be observed that increasing Z leads to a reduction in resource consumption, decreased latency (in the case of Y=0), and decreased operational efficiency. Decreasing Z results in a reduction in dynamic power consumption. The only negative effect of increasing pipelining is the increase in circuit delay. Therefore, overall, increasing the number of pipeline registers contributes to efficiency, power consumption, and resource savings, with only a slight increase in delay (Table 14). On the other hand, using the resource sharing technique mainly increases operational efficiency, with minimal impact on other aspects.

Table 14. Results Analysis.

	Best Performance	**Worst Performance**	**Analysis**

Merit Factor 1	X arbitrary
	
	Y arbitrary
	
	Z=10
		X arbitrary
		
		Y=0
		Z=1
			Achieves the best performance without using resource sharing.

Merit Factor 2	X arbitrary
	
	Y=1
	Z=10
		X arbitrary
		
		Y=0
		Z=1
			Achieves the best performance without resource sharing and using pipeline registers between cycles

Merit Factor 3	X arbitrary
	
	Y=0
	Z=10
		X=0 Z=10,
		
		X=3, Z=5
			Achieves the best performance without resource sharing and without using pipeline registers between cycles

Merit Factor 4	X arbitrary
	
	Y=1
	Z=2
		X=0
		Y=0
		Z=10
			Achieves the best performance with resource sharing and using pipeline registers between cycles

CONCLUSION:

The results indicate that the first merit coefficient, or the ratio of operational efficiency to the number of consumed slices, has the highest value for Z=10 and the lowest for Z=1. Therefore, using the resource sharing technique more will result in a greater reduction in the ratio of operational efficiency to the number of slices.

The second merit coefficient, or the ratio of operational efficiency to dynamic power consumption, has the highest value for Z=10, Y=1, and the lowest for Z=1, Y=0. Thus, employing more of the resource-sharing technique and not using registers between stages will lead to a greater reduction in this ratio. The third merit coefficient, or the ratio of the maximum circuit frequency to the number of slices, has the highest value for Z=1, Y=0, and the lowest for Z=10, X=0. Consequently, using less of the resource-sharing technique and not employing registers between stages will result in a greater reduction in the maximum circuit frequency to the number of slices ratio. The fourth merit coefficient, or the ratio of the maximum circuit frequency to dynamic power consumption, has the highest value for Z=2, Y=1, and the lowest for Z=10, Y=0. Thus, using less of the resource-sharing technique and not employing registers between stages will lead to a greater reduction in the ratio of the maximum circuit frequency to dynamic power consumption.
Considering these results, there is a need to focus on having a robust design with better performance in terms of resource consumption, operational efficiency, circuit frequency, and power consumption. In the future, investigating the impact of additional features of Spartan devices may be interesting. Additionally, integrating some dynamic reconfiguration mechanisms to optimize the execution of the AES algorithm could be worthwhile.

Funding: None.

Conflict of interest: The authors declare that they have no conflict of interest.

REFERENCES:

1. Kumar TM, Balmuri KR, Marchewka A, Bidare Divakarachari P, Konda S (2021) Implementation of Speed-Efficient Key-Scheduling Process of AES for Secure Storage and Transmission of Data. Sensors 21(24): 8347. https://doi.org/10.3390/s21248347

2. Lin S-H, Lee J-Y, Chuang C-C, Lee N-Y, Chen P–Y, Chin W-L (2023) Hardware Implementation of High-Throughput S-Box in AES for Information Security. IEEE Access 11: 59049-59058. https://doi.org/10.1109/ACCESS.2023.3284142

3. Wang SS, Ni WS (2004) An efficient FPGA implementation of advanced encryption standard algorithm. In: 2004 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, Vol. 2, pp. 2-597

4. Soliman MI, Abozaid GY (2011) FPGA implementation and performance evaluation of a high throughput crypto coprocessor. Journal of Parallel and Distributed Computing 71(8): 1075-1084

5. Gielata A, Russek P, Wiatr K (2008) September. AES hardware implementation in FPGA for algorithm acceleration purpose. In: 2008 International Conference on Signals and Electronic Systems, IEEE, pp. 137-140

6. Farashahi RR, Rashidi B, Sayedi SM (2014) FPGA based fast and high-throughput 2-slow retiming 128-bit AES encryption algorithm. Microelectronics Journal 45(8): 1014-1025

7. Kundi DS, Aziz A, Ikram N (2016) A high performance ST-Box based unified AES encryption/decryption architecture on FPGA. Microprocessors and Microsystems 41: 37-46

8. Sheikhpour S, Ko SB, Mahani A (2021) A lowcost fault-attack resilient AES for IoT applications. Microelectronics Reliability 123: 114202

9. Zodpe H, Sapkal A (2020) An efficient AES implementation using FPGA with enhanced security features. Journal of King Saud University-Engineering Sciences 32(2): 115-122

10. Madhavapandian S, MaruthuPandi P (2020) FPGA implementation of highly scalable AES algorithm using modified mix column with gate replacement technique for security application in TCP/IP. Microprocessors and Microsystems 73: 102972

11. Kumar K, Ramkumar KR, Kaur A (2022) A lightweight AES algorithm implementation for encrypting voice messages using field programmable gate arrays. Journal of King Saud University-Computer and Information Sciences 34(6): 3878-3885

12. Wegener F, De Meyer L, Moradi A (2020) Spin me right round rotational symmetry for FPGA-specific AES: Extended version. Journal of Cryptology 33: 1114-1155

13. Kumar TM, Reddy KS, Rinaldi S, Parameshachari BD, Arunachalam K (2021) A low area high speed FPGA implementation of AES architecture for cryptography application. Electronics 10(16): 2023

14. Shahbazi K, Ko SB (2020) High throughput and area-efficient FPGA implementation of AES for high-traffic applications. IET Computers & Digital Techniques, 14(6): 344-352

15. Zodpe H, Sapkal A (2020) An efficient AES implementation using FPGA with enhanced security features. Journal of King Saud University-Engineering Sciences, 32(2): 115-122