Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 11, No. 4, November : 2023

HARDWARE IMPLEMENTATION USING ADVANCE ENCRYPTION STANDARD (AES) FOR SECURITY SOLUTIONS OF CLOUD COMPUTING

T. Ruth Supriya, Research Scholar, Department of Computer Science, Shyam University, Dausa
Dr Gireesh Kumar Dixit Associate Prof, Shyam University
Satyanarayana Department of computer science, Principal, CMR institute of technology, Telangana

Abstract
With little administrative labour or service provider involvement, cloud computing is an architecture that enables ubiquitous, accessible, on-request network access to a common pool of programmable registering assets that can be promptly supplied and delivered. Like any other technology, this one has advantages, but it also has drawbacks. Information security, sincerity, and insurance are a few of its major risks. This study aims to create a reliable and safe cloud data security solution. The bulk of existing techniques and features for cloud security are software implementations because of the "big data" nature of the cloud. One of the most reliable encryption computations using hardware, the High-level Encryption Standard (AES), is used in this paper. This configuration provides one of the safest and fastest methods for obtaining, exchanging, and storing data. Our results shows that we can implement this method 51.5 times quicker than previous hardware implementations and 16.15 times faster than previous software implementations.

Keywords:
cloud computing, FPGA, hardware design, data security, and sophisticated encryption standard

I. Introduction

Among the most secure encryption techniques is the Advanced Encryption Standard (AES). This algorithm is chosen to be a FIPS1 by NIST2[6], and it was created by Joan Deamen and Vincent Rijmen [7]. The technique is typically employed for processing and storing top-secret material. It is implemented in conventional architectures at varying sizes with popular high-level programming languages. [2]. The block diagram of an implementation of 128-bit AES is shown in Figure 1. It takes in 128 bits of data and a key as input, and outputs encrypted or decrypted data along with a matching key of the same size. Ten rounds (including an initial round) are applied to this data, with each round involving four distinct procedures to modify the data.

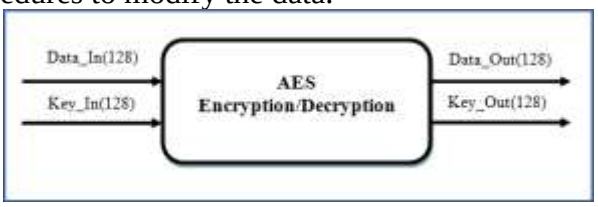

Figure 1. Specifications for implementing AES algorithm

The AES algorithm is implemented in a variety of software programmes. When it comes to dependability and performance, each of them has its benefits and drawbacks. In contrasting those implementations, Bernstein and Schwabe [3] did a fantastic job. Limited hardware implementations of the AES algorithm are now accessible. An idealised implementation technique was described by Marko Mali, Franc Novak, and Anton Biasizzo [4]. We introduced our design version of the AES in section 2. The hardware implementation for the AES architecture is shown in Section 3. The VHDL implementation modules are shown in Section 4. The AES hardware system's encryption and decryption tests are shown in Section 5. Performance evaluation of this algorithm's hardware and software implementations is shown in Section 6. Section 7 concludes with some final thoughts and suggestions for further work.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 11, No. 4, November : 2023

II. Design

After 10 rounds in addition to the first round, our version of the AES algorithm implementation can encrypt 128 bit plaintext data into 128 bit cyphertext data. Up to four distinct procedures are carried out to modify the data during each round. This algorithm can be readily converted to 192 or 256 bits, even though it is just 128 bits. A decryption procedure is carried out to retrieve the original plaintext input from the encrypted cyphertext. Eleven rounds are also included in the decryption process. Up to four operations can be applied to the input data in each cycle of decryption. Since the aim of decryption is to restore the data to its original condition, it is the opposite of the respected encryption processes. The internal workings of the encryption and decryption processes are depicted in the flowcharts in Figure 2. The data becomes cyphertext and is locked after it has gone through the encryption phases.

Shift Rows:

- A data shift operation that creates a 4 X 4 byte matrix out of the data and shifts some rows by a different offset.
- By doing this, the algorithm's data's columns are prevented from becoming linearly independent.

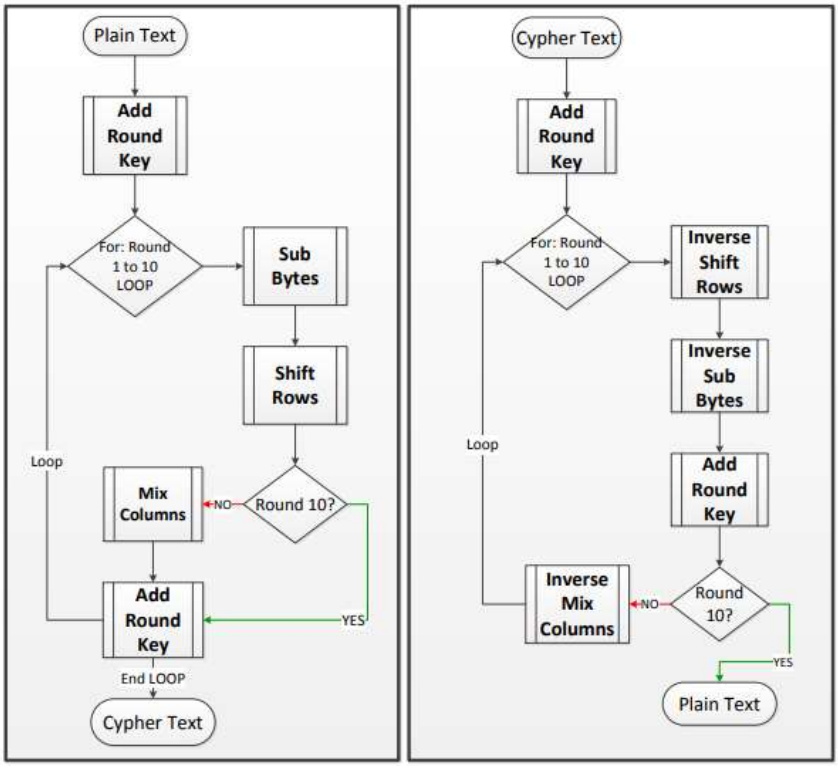

Figure 2. Flowchart showing various steps involved in encryption (left) and decryption (right)

- Mix Columns:

employing the following operations, a key with the same size as the plaintext data (128 bits) is used.

- Add Round Key:
- Uses bitwise operation "XOR" to add the 128-bit Round Key to the 128-bit input.
- Sub Bytes:

o A substitution phase in which a lookup table is used to replace each input byte with a new byte.

o The Sub Bytes step is technically handled by taking the multiplicative inverse of each byte over GF (28), however this Lookup table serves as a more effective technique.

o By adding a nonlinear characteristic, this strengthens the encryption algorithm's defense against attacks.

o Bitwise multiplication of the 128-bit input and a constant 128-bit matrix is used to perform matrix multiplication on the data.

o Diffusion is added to the cypher.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 11, No. 4, November : 2023

III. Hardware Implementation

The design flowchart makes clear that this algorithm requires the data to undergo numerous round changes, with each round requiring the execution of unique operations on the data. The actual data transformation happens in each of these processes, which also serve as the foundation of our solution. The algorithm, as illustrated in figure 2, specifies the sequence in which they must be completed. We used VHDL as a Hardware Descriptive Language to build this hardware design. We used a modular strategy to approach this problem, and we implemented each component piece by piece from the bottom up. A block schematic of the AES algorithm's hardware implementation is shown in Figure 3.

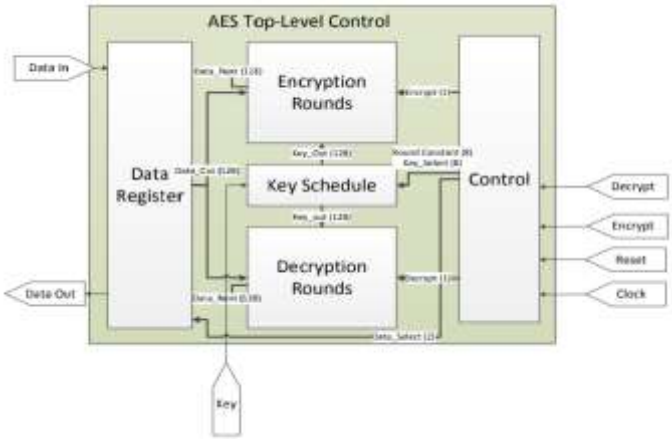

Figure 3. Hardware Implementation

To regulate each round's input and progress, the control module delivers signals. At the clock's positive edge, the data is transferred to the following round and given fresh control signals. After every round, a new key generated by the "Add Round Key" action will be added to the key schedule. We have used a range of VHDL statements and structures along with structural and data flow designs in this implementation. Given that the majority of processes are carried out concurrently, this was the primary driver of the performance increase. Procedural blocks have also been utilized to synchronize the data between rounds.

IV. VHDL Modules

There are various modules that make up the AES encryption scheme. These modules serve as layers through which data is transferred from one module to another, beginning with the top-level implementation and ending with the particular encryption and decryption procedures used in each round. The subprocesses for encryption and decryption are under the direction of the control unit and the top layer module. These subprocesses receive information, and the output is saved in a data register for the subsequent cycle. The following are more particular details that describe the modules:

4.1- Top_Level

The AES algorithm's core module is called the top level code. It takes in the input data, the key, and the encryption or decryption command. Then, it uses a clock-controlled register to store the output of the encryption or decryption operation. The following elements are called by the top level module:

a. Control
b. Encryption Rounds
c. Decryption Rounds
d. Key Schedule

New data is fed into the encryption and decryption registers on the clock's positive edge. At the conclusion of the clock cycle, the data in these registers is put into the encryption/decryption round components, which generate new data.

4.2- Control

With 13 states, the control unit is a linear state machine. For the encryption/decryption algorithm's

other parts to process data and communicate with one another correctly, each state transmits control signals to other parts. The Control Unit flowchart is displayed in Figure 4.

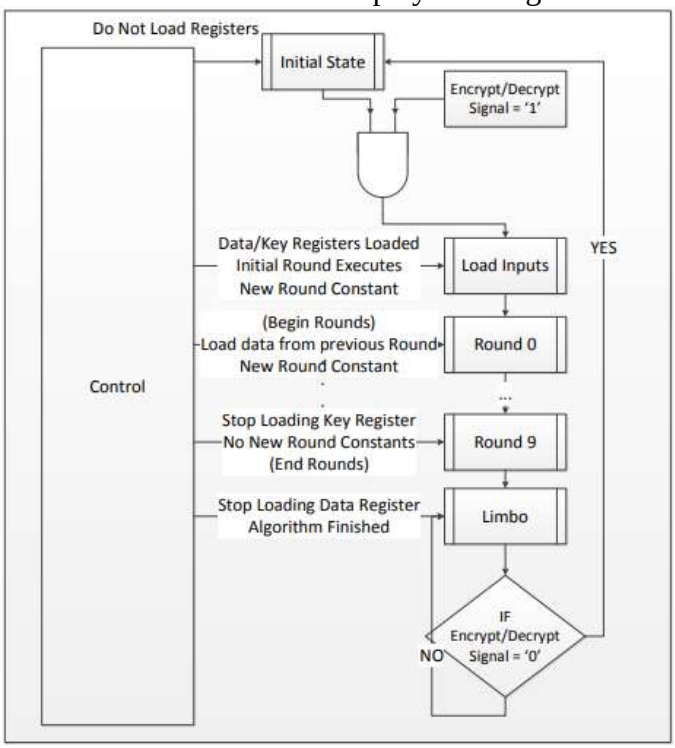

Figure 4. Control Unit Flowchart

The following states make up the Control Unit (CU):

- In the initial state:
- there is an endless loop that awaits the "Encrypt" or "Decrypt" (E/D) signal.
- The inactivity of the Data register and Round Key register occurs until the upper edge of the clock has an active E/D signal.
- Load Inputs State:

o Incoming key_in/data_in inputs are contained in the data register and key register.
o Completes the first encryption and decryption round.
o Sets up the key generator's first new round constant.
o States in Rounds 0–9:
o Following each round, the outcomes of the cypher operations are updated in the data and round key registers.
o Every new round results in a new round constant.
o The signal Round10_Select is given at Round 9, the last round, to make sure the last mix_columns procedure is not carried out.
o Limbo State:
o Turn off the data register's ability to accept new input.
o The encryption/decryption's final results are now included in the data register.
o Verify the E/D signal.

4.3- Key_Generation

For each round of the AES algorithm, a new round key is generated by the Key Generator. The algorithm's "Add Round Key" step, which is carried out a total of 11 times, uses these round keys. The actual key generator is extremely intricate; it divides the input key into four distinct 32-bit words and applies various operations to each word to generate a new key output. We employ the S_Box for a 4-byte operation for the key generator. The flowchart for the encryption key generator is displayed in Figure 5.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 11, No. 4, November : 2023

For the decryption process, which is essentially the same as the encryption version but involves reversing the sequence in which the 32-bit words are XORed, there is an inverse key generator as well. The following are the key generator's inputs:

o Input_Key: input of 128 bits.

o Key_Select: A 1-bit select line that determines whether to load the user key into the key register for the first round and then load fresh keys for each subsequent round.

o Load_Key: A 1-bit input that controls whether the key register is loaded or not

o Round_Constant: The control unit generates an 8-bit input for each round..

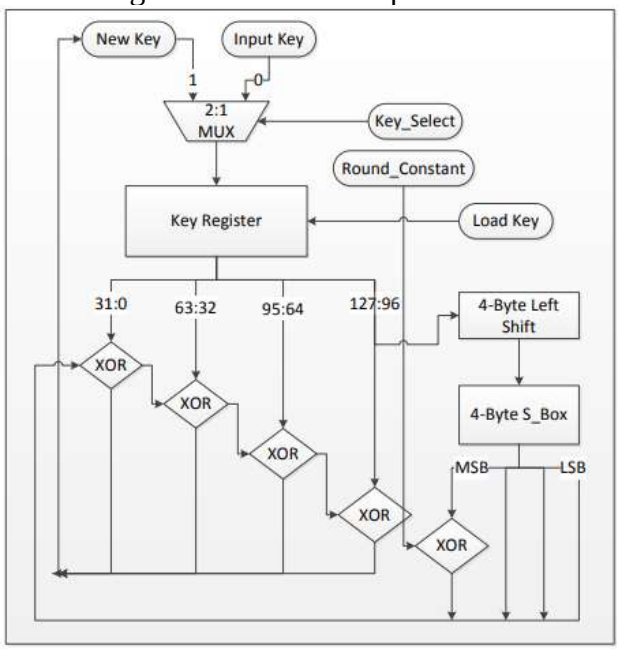

Figure 5. Key Generator Flowchart (Encryption)

4.4- S_bytes

Two data matrices from the algorithm standard [2] are used as substitution boxes in this algorithm. VHDL implements these matrices as lookup tables. An Inverse_S_Box matrix is utilised for decryption, while one matrix is used for encryption. We choose to use the S_Box as a RAM memory in order to have effective memory management. As a result, we may access resources more quickly and realise hardware more effectively. The VHDL code for this component was created using the C# programming language due to the volume of data that S_box contains. Figure 6 displays the block diagram for S_Box.

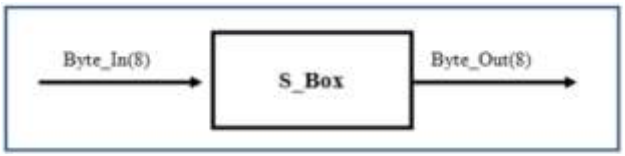

Figure 6. S_Box Block diagram

4.5- Shift_Rows

128 bits of data are transformed into a 16X16 bit matrix for the shift rows operation. The second row is rotated left by one, the third row by two, and the fourth row by three in order to encrypt data. Likewise, we carry out the identical method for the decryption process, but we shift right rather than left.

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 11, No. 4, November : 2023

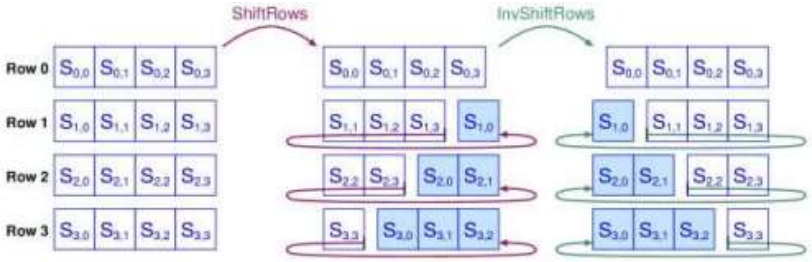

Figure 7. Shift Rows for encryption (center) and decryption (right)[8]

4.6- Mix_Columns

One further activity that takes place during a round is the mix_columns step. A matrix multiplication by two standard matrices is necessary for this operation[5]. The data is initially transformed into a matrix in our implementation, and then it is multiplied by the hexadecimal values 02, 03 for encryption, and 0E, 0B, 0D, 09 for decryption. By choosing the appropriate components, we may assemble the resultant matrix after the multiplications. An overview of the encryption and decryption processes is displayed in Figures 8 and 9, respectively.

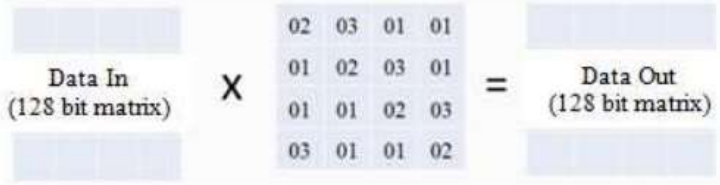

Figure 8: Mix_column operation (encryption)

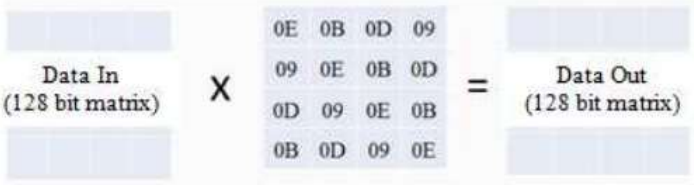

Figure 9. Mix_column operation (decryption)

V. Encryption/decryption Test

Even in various execution contexts, we were able to get remarkable outcomes once the project was developed. A set of data that undergoes encryption and decryption is displayed in the following table. Table 1 makes clear that the information entering the encryption (plaintext) and the information exiting it (decrypted cyphertext) are the same. This demonstrates that the algorithm is being implemented correctly.

More information on the procedures the data travels through and how each round changes the data for both processes is provided in Figures 10 and 11.

Table 1: Encryption / Decryption Phases

	Encryption	Decryption
Data_In	0011223344556677 8899AABBCCDDEEFF	C7AD68D657BEFBE64F2A BA1ECDD7695B
Key_In	F0F0F0F0F0F0F0F0 F0F0F0F0F0F0F0F0	4D59BD8605CC20805740 5E25359D5435
Data_Out	C7AD68D657BEFBE6 4F2ABA1ECDD7695B	00112233445566778899 AABBCCDDEEFF
Key_Out	4D59BD8605CC2080 57405E25359D5435	F0F0F0F0F0F0F0F0F0F0 F0F0F0F0F0F0

Industrial Engineering Journal
ISSN: 0970-2555
Volume : 52, Issue 11, No. 4, November : 2023

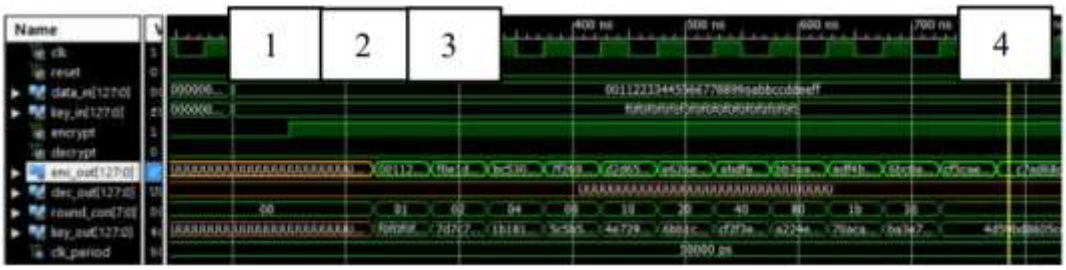

Figure 10. Test Bench I/O results (Below) and Waveform (Above)

Figure 11. Test Bench I/O results (Above) and Waveform (Below)

1. After the "Encrypt" line input reaches logic "1,", the control unit is turned on.
2. The data and key registers are filled with the input data and keys. The first round is played.
3. A set of nine rounds that are completed in order
4. The last encryption round, or "Limbo."

The output is now showing the ciphertext.

VI. Performance Evaluation

The main use for this approach is cloud computing applications, where the execution time (encryption/decryption time) is crucial. To demonstrate the effectiveness of our method, we will compare the algorithm's hardware and software implementations in this section. We have employed a Virtex-5 LX ML501 FPGA board to assess our design. Table 2 displays the clock signals on the board.

Table 2. Virtex-5 LX ML501 FPGA clock signals [11]

Label	Clock Name	FPGA Pin	Description
X1	USER_CLK	AD8	100 MHz single-ended
U8	CLK_33MHZ_FPGA	AB12	33 MHz single-ended
U8	CLK_27MHZ_FPGA	AD13	27 MHz single-ended
U8	CLK_DIFF_FPGA_P	E16	200 MHz differential pair (pos)
U8	CLK_DIFF_FPGA_N	E17	200 MHz differential pair (neg)

The findings of our hardware implementation, which made use of a single-ended clock at the AD8 pin with a frequency of 100 MHz, are displayed in Table 3.

Table 3. Hardware Performance of AES on a Virtex5 FPGA board

Operation	Clock Cycles	Time (Seconds)	Time in (µs)
Encryption (128 Bits)	13	$13*10^{-9}$ Seconds	0.13µs
Decryption (128 Bits)	13	$13*10^{-9}$ Seconds	0.13µs
Encryption (32 Bits)[5]	29*	$29*10^{-9}$ Seconds	0.29µs
Decryption (32 Bits)	29*	$29*10^{-9}$ Seconds	0.29µs

6.1- Software Performance of AES

The AES algorithm is implemented in a variety of software programmes. When it comes to dependability and performance, each of them has its benefits and drawbacks. In contrasting those implementations, Bernstein and Schwabe [3] did a fantastic job. One of the quickest software implementations we could find was compiled on a 2.1 GHz processor running Windows XP SP1 and was written in C++ using Microsoft Visual C++.NET 2003 [1]. Furthermore, 386 assembly routines for multiple-precision addition, subtraction, and code optimisation were employed in the article [1]. When the AES algorithm was implemented in 128 bit mode, it processed 61.010 Mb/s [1]. The time required to calculate the encryption for 128 bits is displayed in the following:
61.010 Mb/s = 6.101 * 10^7 b/s
476640.625 Blocks/s
1 Block (128 bits) =**2.1 µs**

6.2- Other Hardware Implementations

There aren't many hardware versions of the AES algorithm available. An optimal implementation technique was proposed by Marko Mali, Franc Novak, and Anton Biasizzo [4] and is displayed in table 4.

Table 4. Hardware performance [4]

Function	Time **(µs)**	Throughput(**Mbit/s**)
Cipher(encryption)	16.50	7.76
Inverse Cipher (decryption)	20.56	6.23

A Celoxica RC1000 hardware platform with a Xilinx Virtex family BG560 is used for this implementation. This is predicated on a 74.4MHz clock rate. The primary factor affecting performance, frequency, is around 25% lower than our 100MHz frequency used with the Virtex-5.Our hardware solution has encryption and decryption speeds that are 126.9 and 158.1 times quicker, respectively. Even with the PCI bus taken into account, our encryption and decryption speeds are still 56.9 and 70.9 times faster, respectively. Let's assume that we are lowering our frequency to 74.4MHz to match the clock rate in [4] in order to have a comparable comparison between our implementation and the implementation reported in [4]. We can accomplish an execution time of 0.36µs by using the PCI bus, which is still 45.8 times quicker for encryption and 57.1 times faster for decryption.

VII. Conclusion

We have introduced a fast hardware version of the AES algorithm in this research, which is appropriate for cloud computing applications. We compared our AES design and implementation's timing performance to that of competing hardware and software implementations. The software code, which ran on a processor with a frequency of 2.1 GHz, was executed 21 times quicker than the frequency used in the FPGA for our hardware implementation. In contrast, our FPGA hardware implementation was substantially faster. A block of 128 bits was encrypted in 0.13 µs using our FPGA hardware implementation at 100 MHz; whereas, a software-optimized implementation operating at 2.1 GHz took 2.1 µs to encrypt the same block. The hardware implementation we have proposed is **16.15 times quicker** than the software-optimized approach. We also conducted a comparison between our AES hardware design and other hardware implementations of this encryption standard that used the same PCI bus and frequency. An average execution time that is **51.5 times faster** than the present one was achieved. These findings indicate that there is a lot of room for this initiative to be expanded upon and used in other contexts. Deploying this approach in real-time contexts and observing its behaviour is an essential component that we would like to look into in the future. We could access the network's physical layer using a variety of technologies like Cornell University's SoNIC. In this manner, we may target a strong security solution in real-time environments by embedding our solution into the lower tiers of the network stack. [10].

References

[1] Jain, Raj. "CSE567M: Computer Systems Analysis (2006, fall). *CSE567M: Computer Systems Analysis(Fall 2006)*
[2] Daemen, Joan; Rijmen, Vincent (9/04/2003). "AES Proposal: Rijndael". National Institute of Standards and Technology. p. 1. Retrieved 21 February 2013.
[3] Daniel J. Bernstein, Peter Schwabe. "New AES Software Speed Records", in INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008. Proceedings
[4] Marko Mali, Franc Novak and Anton Biasizzo, "Hardware implementation of AES algorithm", Journal of Electrical Engineering, Vol. 56, NO. 9- 10, 2005, 265–269.
[5] Xintong, Kit Choy. (2014, June 26).*Understanding AES Mix-Columns Transformation Calculation*
[6] Gaithersburg, MD: Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology. (2001,November 26). *Announcing the AdvancedEncryption Standard (AES)* [Online].
[7] J.Daemen and V. Rijmen. (1999, September 3). *AES Proposal: Rijndael, AES Algorithm Submission*[Online].
[8] Gürkaynak, Frank K. (2006, December 20). *GALS System Design: Side Channel Attack Secure Cryptographic Accelerators* [Online].
[9] Mell, Peter and Grance, Timothy. (2011September). *The NIST Definition of CloudComputing*
[10] Lee Suh, Ky Wang Han and Hakim Weatherspoon, (2013 May), SoNIC*: Precise Realtime Software Access and Control of Wired Networks*,
[11] Xilinx, Inc. (2014). *Virtex-5 LX FPGA ML501Evaluation Platform* [Online].