Enhanced Throughput AES Algorithm on FPGA

*

H B Naga Chempati
Dept. of Computer Engineering
Arizona State University
1215039241
hchempat@asu.edu

Prasanth Sukhapalli
Dept. of Computer Engineering
Arizona State University
1215358560
psukhapa@asu.edu

Pradeep Chowdary Jampani
Dept. of Computer Engineering
Arizona State University
1215335693
pjampani@asu.edu

***Abstract*—With data transactions happening among Enterprises on a large scale daily, the demand to secure such data from attackers has been a major concern nowadays. Cryptographic Techniques such as AES have been evolved over the years to facilitate secure transport of data. Software implementation of AES algorithm on general purpose CPUs may perform well on a small scale but has significant latency on a large scale. FPGAs are being used widely because they are flexible and can be reprogrammed. Our project involves implementing the illustrious 128-bit AES CryptoAlgorithm in Xilinx Virtex 7 FPGA board and obtain an enhanced throughput performance using architecture optimization techniques such as folding, pipelining, parallelism,etc. The final optimized architecture is unrolled pipelined with parallelism which gave us a throughput of 109 Gbps with considerable area constraints. The FPGA performance of the proposed architecture is comparable to ASIC implementations. The FPGA optimized architecture version of the algorithm is suited for scenarios to securely transfer data in Research labs, to perform bitcoin mining in small scale enterprises, etc.**

***Index Terms*—Xilinx , AES , Folding , Unrolling , pipelining**

I. Introduction

In this modern digital era, with people using the internet for their day to day activities, there are lots of data generated each and every second. With the data flowing through various devices , there is a need to secure the data in order to protect it from adversaries all over the world. Numerous techniques and strategies are available to bypass the data and obtain critical information. Thus, Cryptographic techniques were created in order to encrypt and decrypt the data at the source and the receiver end. In the year 1970, IBM has announced its cryptographic standard called Data Encryption standard which is a symmetric key algorithm having a key length of 56 bits for which it has been subjected to several brute force attack in recent years and can be compromised nowadays. AES is a subset of Rijndael Block Ciphering method invented by two Belgian Cryptographers, Vincent Rijmen and Joan Daemon. The subset consists of symmetric secret keys of length 128 bits, 192 bits and 256 bits to encrypt and decrypt data blocks of 128 bits.

Considering the massive flow of data, large scale companies and organisations have the demand to secure such data on a daily basis from attackers and adversaries. Using general purpose CPUs to encrypt such large amount of data (software implementation of AES) won't give us a better performance when it comes to processing a large amount of data. Thus we are in need of Hardware accelerators such as Cryptoprocessors to process such heavy data within the stipulated time. AES has been used worldwide and hardware implementation of the AES algorithm would give us better performance and efficient computation within constrained time limits when compared to general purpose hardware resources. Our work involves implementing an enhanced throughput AES algorithm that uses the 128-bit secret key on 128-bit blocks on ¡FPGA board¿. The reason for choosing FPGAs is that it is flexible enough to get any digital circuit of the user's choice. We are motivated to help users who rely on limited hardware resource to implement a variety of designs and in situations where there is a constant need to change the design to adapt to varying demands. For our project, MATLAB is used to run the design simulations and testbench.

II. Design Specification

The AES algorithm using 128 bit symmetric key to encrypt the data blocks of 128 bit each is implemented on Xilinx virtex 7 XC7V2000T-2FLG1925 FPGA platform. The design target is to achieve a target throughput of 75 Gbps with an area constraint to be less than 45,000 slices.

III. Architecture

As always, any Encryption algorithm needs the data to go through several transformations. AES also needs its input data to undergo several transformations such as AddRoundkey, Main Rounds, SubBytes, Shiftrows, Mixcolumns. It mainly has three blocks, initial round has only AddRoundKey and the main rounds have SubBytes, ShiftRows, MixColumns, and AddRoundKey. The final round has only SubBytes, ShiftRows and AddRoundKey. Note that final round does not have the MixColumns block in it. The number of rounds that we will have to perform is dependant on the number of keys. Here we are implementing the algorithm for 128-bit keys, so we use only 9 Main rounds. Instead we will have to use 11 Main rounds for 192-bit keys and 13 Main rounds for 256-bit keys.

A. AddRoundKey

The AddRoundKey operation is a simple operation in which we just do the exclusive-or operation between the data and keys respectively. Here, this is the only step in which we

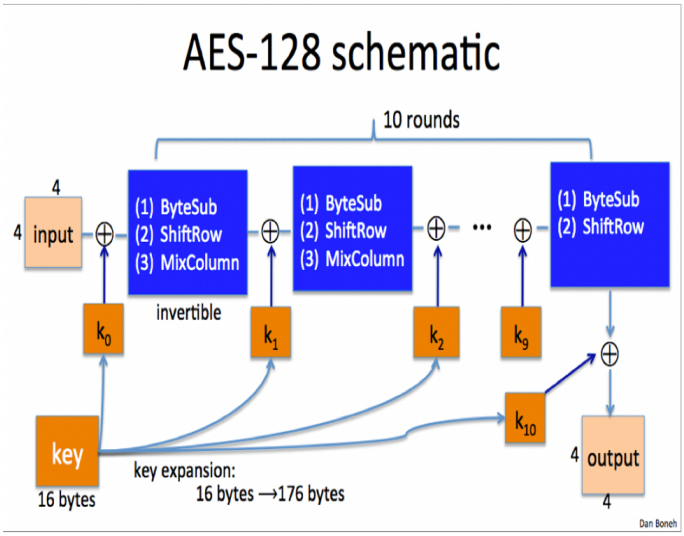

Fig. 1. AES Encryption Model

directly operate on the keys and the data. Here, the key matrix is different and specific for each round.

B. SubBytes

This is a block where the data initially is concealed, i.e it is turned into another data using a trace of it in the lookup table. The 16 input bytes are substituted with the respective element by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns. This block does the substitution on a byte basis due to the addressability of the lookup table. The S-Box for 128-bit plain text comprises of 256 bytes (16x16 matrix) which is created by determining the multiplicative inverse for a given number in GF (2P8) =GF(2)[x]/(x8+x4+x3+x+1). To read this Table, the byte input is broken into two 4-bit halves. The first half determines the row and the second half determines the column. Essentially, AES uses the same S-Box for all bytes.

C. ShiftRows

The state in AES is defined as a matrix. The Shiftrows block randomizes the data matrix. It is like scattering the data points in a matrix. Here, each row in the matrix is internally shifted towards left. Note that, all matrices are defined using a byte data points.i.e 16X16 matrices with 1byte elements. Each row is left shifted by a certain amount. The number of shifts depends on the rank of the row which is basically the index of the row.

D. MixColumn

Until this step, only rows are operated on which makes it easy to revert the effect of randomization. So, operations on columns is also advised. So, we perform multiplication of each column with a matrix, which is always a constant, below to get more level of encryption. The multiplication is simply an exclusive-or operation. The matrix is shown in Figure 2.

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

Fig. 2. Mix Column Matrix

E. AES key Schedule

The SubKey matrix is always specific to each round, which means we will have to generate new keys for each round. This Subkey generation is done taking the current Subkey which is from the previous round. This is applied in a different manner for columns with indices as multiples of 4 and the rest. For columns with indices as multiples of 4, we apply three transformations, S-Box substitution, a Shift operation and exclusive-or with a constant matrix and then exclusive-or with the fourth row to the left. For the rest of the columns, do just the exclusive-or with the fourth row to its left. The output of the key schedule function is input to the AddRoundKey operation in AES encryption which is performed to produce the next round key.

IV. Architecture Design

Our main design consideration is that hardware implementation has more efficiency and speed than the software implementation. The design is implemented in FPGA because of the advantage of reprogramming ability. To simulate the algorithm in hardware blocks we needed a High-Level Synthesis tool, so We have considered implementing it on Synphony Model Compiler (SMC). The DSP design can be synthesized into architecturally-optimized RTL by using Synphony Model Compiler, which has a high-level synthesis (HLS) tool that performs architectural optimizations from the Simulink specification simulating a desired model of FPGA, which we used here is Virtex 7 platform.

The obvious and easy method to implement the algorithm is to loop the 9 rounds after the initial round, which we call it Folded Architecture. In this architecture AES block is implemented as three consecutive blocks with the middle block, main rounds block, looped 9 times as in Figure 4. We operate on hardware blocks here, so blocks operating on bytes are used. Inputs will be represented as 16 bytes instead of 128-bits. All these 16-bytes are concatenated using 4 concatenate blocks and are fed as a serial input to AES block. Corresponding hardware blocks use extract blocks to have their respective bytes to operate on. Similarly, Keys are also fed into AES block as concatenated blocks of bytes, 16-bytes in total. A MATLAB testbench code is written to feed the inputs and Keys to AES block. The block diagram of Folded Architecture is shown in Figure 3.

The AddRoundKey operation is a simple exclusive-or between the given 16 keys and 16 Input data bytes. In first

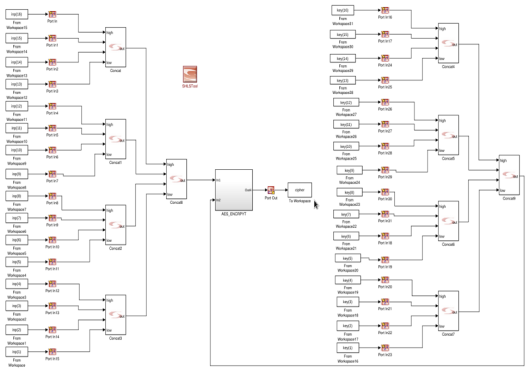

Fig. 3. Folded architecture

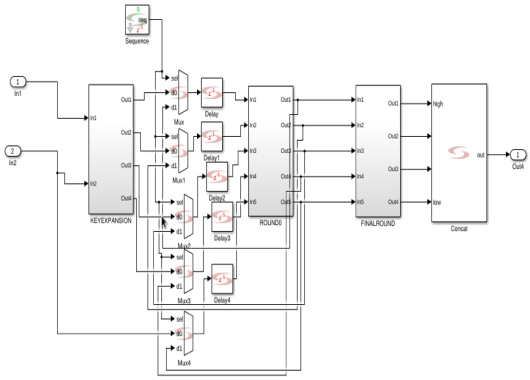

Fig. 4. AES Subsystem showing organisation of rounds

block of AES this is performed on the original Keys and Inputs. In the Main rounds each time a different key matrix is generated, and inputs are also modified using the remaining transformations. Each byte is extracted using an extract block from the stream input of both keys and data. A single hardware block is used to perform exclusive-or between one-byte key and data input. All the computed bytes are again concatenated for next operations. The block diagram can be seen in Figure 5.

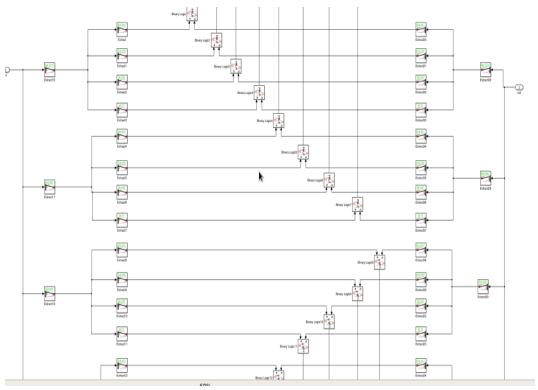

Fig. 5. AddRoundKey

A Sub-block operation is a simple substitution of data by some other data it points to in the look-up table. This clearly means that the actual input data is the address to access the look-up table and the new data is the data to be read. This gives the need to use ROM blocks. So, the 256-element data to be substituted is stored sequentially in dual-port ROM blocks, to minimize the hardware, and to get two-byte substitution per ROM block. The input data will be the address and the ROM data will be the substitution.

The subsequent operation is Shift-row operation. Here, shifting of rows should be done to a row by row index number of times. Unless for any other prodigal ways, wiring is the most effective way. We just feed the outputs of Sub-block operation to the Mixed-column operation by simply changing the wiring as shown in Figure 6.

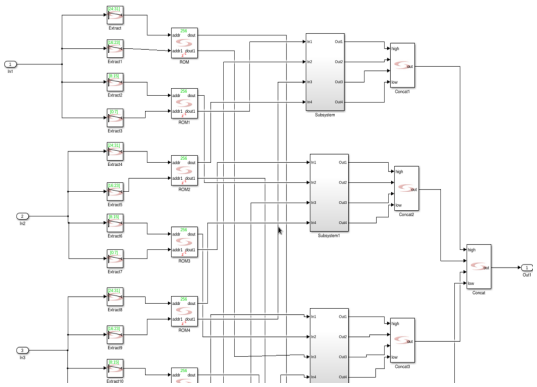

Fig. 6. Subbytes, Shiftrows, Mixcolumns

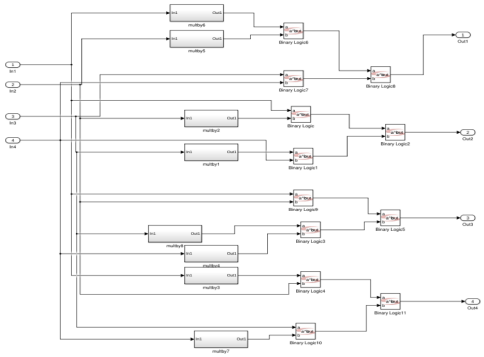

Fig. 7. MixColumnSubsystem

Mix-Column operation is multiplication of the column with a Standard constant matrix such that again a column matrix is obtained. Multiplication with 2 is done by just shifting the data point by 1 bit. Multiplication with 3 is done by shifting the data point by 1 bit and then doing exclusive-or with the data point itself. These are done in isolated systems from one another using shifting operation and then they are added. Obtained outputs must then be exclusive-ored again. To cope up with the overflow effect we do the mod operation by 27 to the input. This operation can be seen in Figure 6.

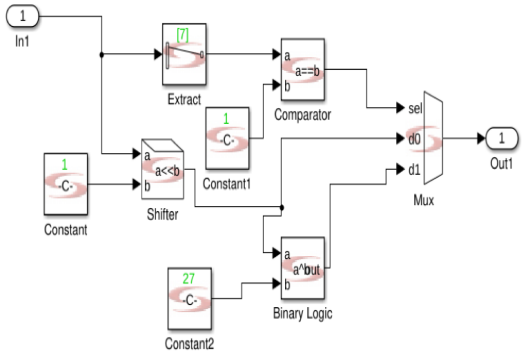

Fig. 8. Multiplication by 2

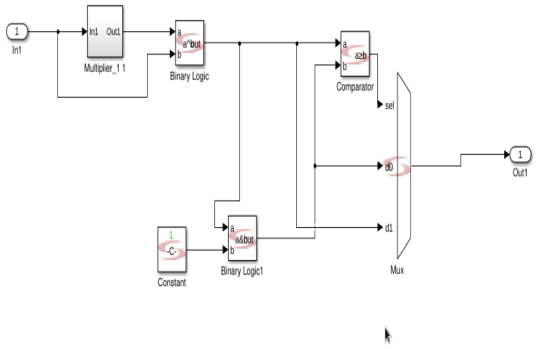

Fig. 9. Multiplication by 3

Key Scheduling operation is used to generate a new Key matrix for AddroundKey operation. The input to this block is the Key matrix from previous Key Scheduling block. This operation will continue basing on the first given input Key inputs. Columns whose indices are multiples of 4 are operated in this way, first they are shifted by 1 byte such that last byte becomes last but one, and first byte becomes last. And then, column is substituted using an S-box, and then it is exclusive-ored with fourth column to the left and with an RCON matrix. For the rest, they are only exclusive-ored with fourth element on the left. This operation is shown in Figure 10.

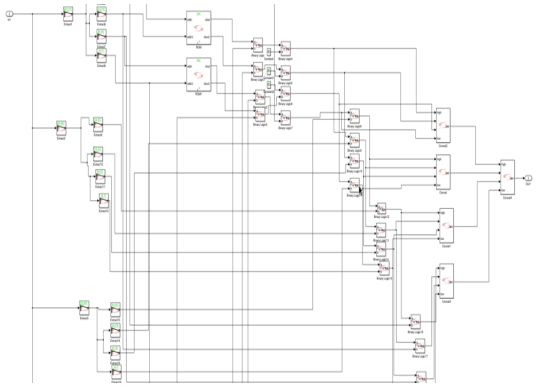

Fig. 10. Subkey Expansion

V. Optimized Architecture

The baseline architecture could not meet our design target of high throughput of 75 mbps . Thus , we apply various architectural level optimization techniques to maximize the throughput . Pipelining the basic architecture gave bit high throughput but this results are menial when compared to the design tagert throughput . Thus , pipelining alongside various techniques like unrolling and parallelism is applied to achieve high throughput . The initial optimization is done using pipelining and the two rules of pipelining have been applied . The pipelining is implemented where the design can be divided into two halves and so on . The implementation of pipelining is done using registers , these registers pipeline architecture by passing the input at alternative clock cycles . The Second

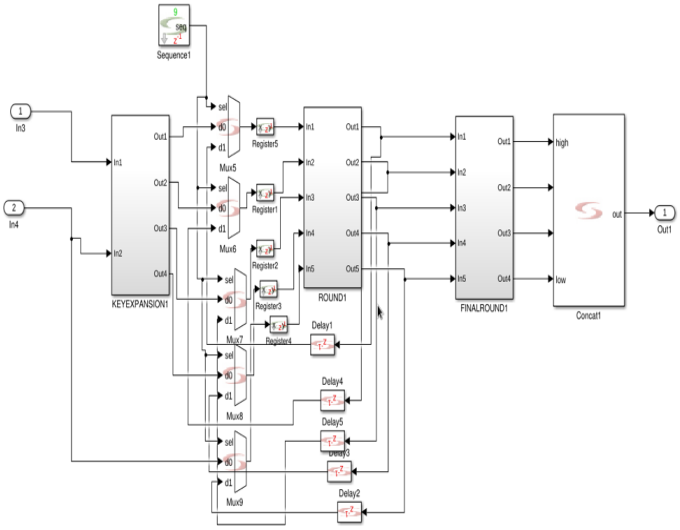

Fig. 11. Folded pipelined architecture

optimization is done using Combining the pipelining in the previous stage with parallelism . Parallelism being a major critique when increasing the throughput , helps us to increase the throughput .

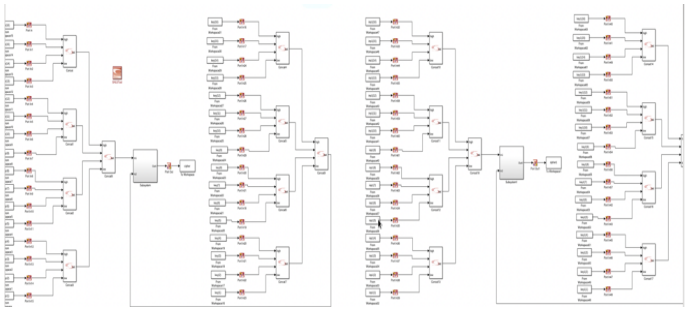

Fig. 12. Folded pipelined parallelism architecture.

Major hindrance which bucks throughput is the decision logic in the folded architecture . This overhead can be removed by eliminating the decision logic and unrolling the loop manually . Thus , third optimization is done by unrolling the

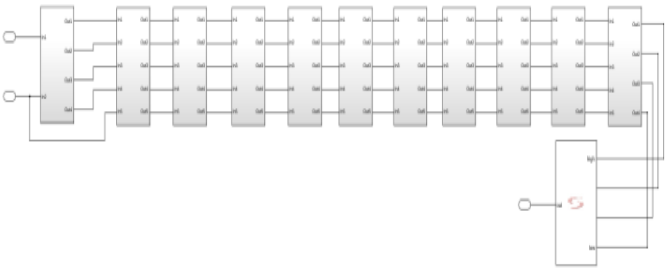

Fig. 13. Unrolled architecture.

architecture . The basic architecture is unrolled and this helps in increasing the throughput .

Fig. 14. Unrolled pipelined

The parallelism between the stages can be implemented which helps in improving the throughput . Thus , the next optimization is done by implementing pipelining to the unrolled architecture . Although we have achieved higher throughput using above architectural techniques , the throughput still falls short of the design target . Thus , the final optimization is done by Combining the unrolled pipeline architecture with parallelism .

VI. Results

Reports of mapping in FPGA target can be used in analyzing the architectures implemented . The primacy design target was to achieve Throughput greater than 75 with area as a constraint . The architecture is mapped on Xilinx XC7V2000T-2FLG1925 . Thus , Frequency , Throughput and area are calculated from Synplify pro .

Thus from Results , we can observe that the baseline architecture of folded design performs worst with a throughput of 3.4Gbps. But , the area resources are well under constrain for the folded design . For the Folded pipelined architecture the throughput has been improved by 1.67 times with the expense of minimal increase in the area . The increase in area is less when compared to increase in the throughput . Further optimization done by unrolling achieved throughput of almost 9 times of base architecture . The combined size of luts and Non-io has also been increased as we are unrolling the loop without any decision factor and manually adding additional stages . Further optimization of this design provided an improvised throughput by pipelining . As each stage now is being pipelined , there is correspondence increase in the area due to additional registers . Finally , we combine unrolling with parallelism and pipelining . This optimization provides an increase in the throughput to a extend which satiates our design

Implementation	Frequency	Throughput (Gbps)	LUT	Non-IO reg
Folded (base design)	265.6	3.4	1250	805
Folded pipelined	444.3	5.688	1291	819
Folded pipelined + parallel (P1)	444.3	11.376	2582	1638
Unrolling	242.2	31.01	11280	7980
Unrolling pipelined	426.09	54.54	13370	8020
Unrolling pipelined + Parallelism(P1)	426.06	109.08	26740	16040
Unrolling pipelined + Parallelism (P2)	426	219.6	53480	32080

Fig. 15. Results from synplify pro.

target . The area constraints are well under our constraints too. The throughput achieved by this design architecture is 109.08 Gbps , Which is almost 32 times of what we have achieved in an baseline architecture . Additionally , a further stage of parallelism is performed . This optimization yield much more higher throughput but the design constraint of area is violated. As Throughput being our primary goal with area constraints we analyze the efficiency of the deign using Throughput efficiency .

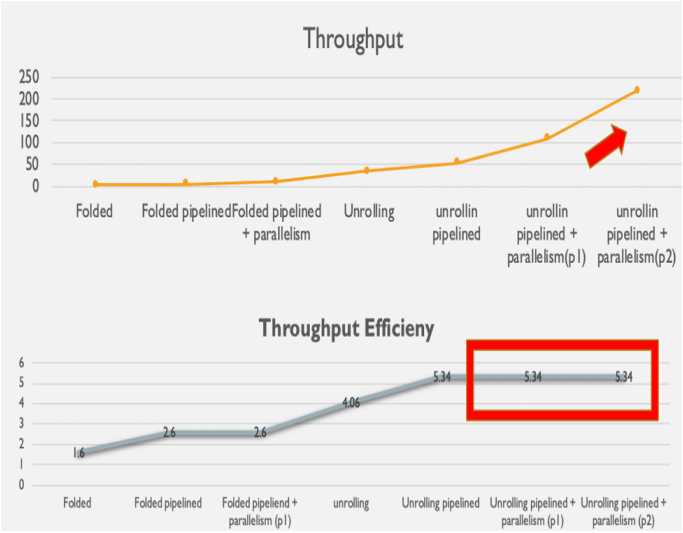

Fig. 16. Throughput efficiency over different architectures

From the above figure , we can analyze that the throughput radically increases in unrolling and there is an exponential rise

when parallelism is applied . But , when the parallelism of 2nd stage is applied there is no change in the throughput efficiency . Thus , going to next levels of parallelism is futile .

VII. Conclusion

We have designed a high throughput AES algorithm on FPGA . A high Throughput AES algorithm with a throughput of 109.08 is been implemented with design constraints being area (LUT + Non-io reg) less than 20,000 slices combined . The High throughput is achieved using Unrolling pipelined parallelism technique . The Throughput obtained using this technique is 32 times the baseline architecture .

VIII. Future Scope

For the Future work , the design parameters can be an impediment for increase the parallelism . But , the sub-pipelining the inner blocks can be used rather than parallelism to increase the throughput efficiency .

IX. References

[1] J. Daemen and V. Rijmen, AES Proposal: Rijndael, AES Algorithm Submission, September 3, 1999.

[2] , Fast implementation and fair comparison of the final candidates for advanced encryption standard using field programmable gate arrays, Lecture Notes in Computer Science, vol. 2020, pp. 84??, 2001. [Online]. Available: citeseer.ist.psu.edu/article/gaj01fast.html

[3] Z. Yuan, Y. Wang, J. Li, R. Li, and W. Zhao, FPGA based optimization for masked AES implementation, in Proc. IEEE 54th Int. Midwest Symp. Circuits Syst. (MWSCAS), Aug. 2011, pp. 14.

[4] D. Canright, A very compact S-box for AES, in Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst. Berlin, Germany: Springer, 2005, pp. 441455

[5]. Li, Xinqiang, Lili Yu, and Lihuan Wei. "The Application of Hybrid Encryption Algorithm in Software Security". 2013 3rd International Conference on Consumer Electronics, Communications and Networks (2013): n. pag. Web.

[6]. Farouk, H.A., and M. Saeb. "An Improved FPGA Implementation of the Modified Hybrid Hiding Encryption Algorithm (MHHEA) For Data Communication Security". Design, Automation and Test in Europe n. pag. Web.

[7] GU Da-wu, XU Sheng-wave translation. The Design of Rijindael: AES-the Advanced Encryption Standard, [Belgium] Joan Daemen [Belgium] Vincent Rijmen , Tsinghua University Press, Beijing, 2003.

[8] P. S. Abhijith, M. Srivastava, A. Mishra, M. Goswami and B. R. Singh, High Performance Hardware Implementation of AES using Minimal Resources, IEEE International Conference on Intelligent Systems and Signal Processing, Gujarat, pp. 338-343, March 2013.

[9] Pritamkumar N. Khose, Prof. Vrushali G. Raut, "Implementation of AES Algorithm on FPGA for Low Area Consumption",2015 International Conference on Pervasive Computing (JCPC).