FPGA Implementation of Advance Encryption Standard Using Xilinx System Generator

[*]Alia Arshad, Kanwal Aslam, Dur-e-Shahwar Kundi and Arshad Aziz

Department of Electrical Engineering, National University of Sciences and Technology (NUST),
Karachi Campus, H-12 Islamabad, Pakistan
[*]*Email: alia_arshad123 {at} hotmail.com*

ABSTRACT— ***This paper presents a resource efficient reconfigurable hardware implementation of Advance Encryption Standard (AES) algorithm using High Level Language (HLL) approach on Field Programmable Gate Array (FPGA) for rapid development. In this work, we use an approach to directly map the design described in a high level package i.e. System Generator on FPGA platforms. This approach is ideal for Encryption functions where the development of data-path architectures can easily be done to provide bit and cycle accurate models. Our approach fills the gap between performance and flexibility by efficiently applying re-configurability. We use primitive level approach and customize all the operations our design by effectively utilizing conventional blocks of Xilinx System Generator to get optimum performance in terms of speed and area. This approach enables us to minimize critical paths in design and increase the overall frequency of design especially for MixColumn and SubByte transform. Our design shows best performance in terms of speed and area as compared with any other software and hardware/software co-design implementation counterparts, it operates at 288.19 MHZ and offers high throughput of 36.864 Gbps.***

Keywords— AES, System Generator, FPGA, Cryptography

1. NTRODUCTION

In the era of this modern world, secrecy and security are the major concerns which should be kept in mind with respect to digital computer systems. Cryptography provides solutions regarding fool proof secrecy, security, and reliability of given information. It is keeping its vital function in different applications which includes online banking system, Cellular networks, computer hardware emulations, medical imaging, software defined radios, bioinformatics and wireless communication etc. Reconfigurable platform like FPGA are the best for implementation of cryptographic algorithms [1]. These platforms are reconfigurable to provide time and cost effective solutions as compared to Application Specific Integrated Circuit (ASIC) [2]. A reconfigurable platform provides improved performance than software implementations and can also be reconfigured on the fly to store the updated encryption standard.

In recent years, digital hardware design seems to be more similar to the software design, driven by the increased complex design, time-to-market anxiety and demand for an effective participation between various project teams. Platform-based design has become appropriate for IC design projects in this digital world. To minimize the algorithmic process time in term of plenty of data, it is very much inevitable to adopt and implement the algorithm of hardware, despite the fact that software implementation can only meet the requirement of low cost for users. In order to attain a balance between the cost and time, an efficient method must be explored and implement for various combinations of hardware and software to realize algorithmic best solutions of different requisite.

In this work we focus on the efficient and well organized implementation of AES architecture on FPGA using HLL approach i.e. Xilinx System Generator. The proposed FPGA platform for the implementation of this work is Virtex-5 FPGA from Xilinx. HLLs provide a better way to achieve the high-performance in reconfigurable computing to implement any design in hardware [3]. It provides a great deal of functional abstraction and develops highly parallel systems. High level language tool automatically maps the model design to efficient hardware implementation.

Our paper is further structured as follows: Section – 2 presents the literature survey of the related high level implementations of AES. Section – 3 gives the brief overview of AES algorithm while section - 4 shows AES architecture using Xilinx System Generator. In section – 5, the verification of design and validity of results are discussed and Section – 6 presents the implementation results of proposed design. The conclusion of the work is given in section – 7.

2. RELATED WORK

After the approval of AES in 2001, it has been implemented on many different platforms including hardware, software and high level language tools. In this literature we only focus on high level language based implementations of AES related to our work. A large number of AES high level language implementations have been done. Whereas, the earlier implementations mainly focused on the performance criterion but not the time constraints and the flexibility of the designs.

Most recently adopted embedded systems design methodology prototypes a target design in a high-level languages such as C, Handel-C [4], System C [5] and Impulse C [6]. Most of them translate the design manually into an HDL code that can be prolonged and error-prone, thus reduces the flexibility of a prototype. Other high-level languages that can directly translate the design are too time-consuming.

Mostly implementations are focused on software and hardware-based platforms, both have their own pros and cons. Software-based implementations [7] provide cost effective solutions and reduced Time-to-Market but at the cost of performance. Whereas hardware solutions [8] are better in terms of performance but cost of the design and Time-to-Market increases to great extend. High-level language is the platform that provides the features in-between the two. In this study, we have tried to enhance the performance of high-level package by applying some efficient optimization techniques in the design and integrating it with the hardware platform FPGA.

M. Mali et al. [4] implemented AES using high-level design language Handel-C on Celoxica RC1000 development board and Celoxica DK development suite. It runs at the frequency of 74.4 MHz for an external data storage unit. While, in [5] M. Askar et al. presented AES implementation using System C tool and System Crafter tool to translate the System C descriptions into hardware. It takes 153 machine cycles to process 128-bit data. In [6], M. Lukowiak et al. described the AES architecture with data-path of 32-bit using different environment such as Impulse C, C-to-FPGA programming model and Compilation engine to introduce rapid prototyping of digital hardware. The relative performance using Impulse C achieves a speedup of 128.6 times as compared to other software implementations while in [9], D. Osvik et al. has given the AES-128 encryption architecture by targeting not only the low-end processors such as 8-bit AVR microcontroller and 32-bit ARM microprocessor but also on high performance NVIDIA and cell broadband processing engine. On 8-bit AVR microcontroller it offers the frequency of 124.6 cycles/byte while on 32-bit ARM microprocessor it results in the frequency of 34 cycles/byte. The AES architecture is designed to reduce the required number of clock cycles during encryption and decryption using these targeted platforms. In [10], T. Babu et al. reported the hardware together with high-level design tools realization. They implemented the AES algorithm using custom instruction method provided by ARM 7 with Keil platform. Hasamnis et al. [11] has also presented the high-level design architecture for AES implementation where the algorithm is controlled through C-code written in NIOS II IDE that requires 21731020 CPU Cycles to complete the simulation. Bos et. al [12] reported two 128-bit AES implementations on NVIDIA graphics processing units (GPUs) and the Cell broadband engine. The GPU implementation delivers the throughput of 0.17 cycles per byte for encryption module and the Cell broadband engine offer the speed of 11.7 cycles per byte. In [13], Biglari et al present 128-bit pipeline architecture of AES on Maestro platform. This work presents tightly coupled encryption and round key generation modules in the main encryption module that enables the design to reach a throughput of 12.8 Gbps and runs at the speed of 100MHz. Similarly Mourad et al [14] presents the methodology which maps the AES design described in a high level language, Handel-C, to FPGA for low area consideration. This design reported 437 slices for encryption and offers the throughput of 1.716 Gbps.

3. ADVANCED ENCRYPTION STANDARD

In October, 2000 the Rijndael algorithm was selected as the new Advanced Encryption Standards (AES) Algorithm by National Institute of Standards and Technology (NIST) and announced as Federal Information Processing Standard Publication (FIPS PUB) 197 in November, 2001 [15]. AES algorithm used to supports 128-bit data and variable key sizes of 128,192 and 256 bits. The 16 bytes of data arranged in a 4 x 4 array of bytes with four rows and four columns called the state, and is treated as input for AES algorithm as depicted in Figure 1.

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Figure 1: State Array

AES-128 algorithm comprises of ten rounds and each round further consists of four basic byte-oriented transformations i.e. ShiftRow, SubByte, MixColumn and AddRoundKey except the last round where the MixColumn transformation is eliminated. An initial round is added at the start-up which comprises of only AddRoundKey transformation as shown in Figure 2.

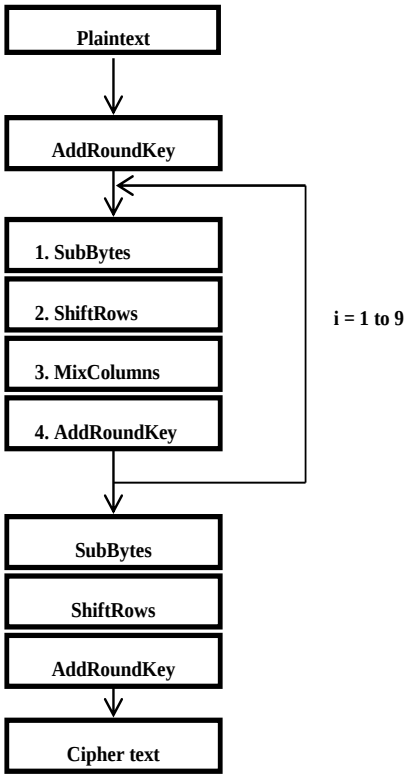

Figure 2: Advance Encryption Standard

3.1 SubByte

SubByte is a non-linear substitution of bytes; each byte of the state is replaced with another using substitution table (S-box). The S-box is invertible and two transformations takes place in order to construct substitution table i.e. to takes the multiplicative inverse in GF(2^8) [16] in which the elements {00} are being mapped itself and then affine transform is applied over GF(2^8).

$$b_{i,j} = S[b_{i,j}] \quad (1)$$

3.2 ShiftRow

ShiftRow transformation cyclically shifts the last three rows of the state with a certain number of steps to the left while first row remained unchanged. It also performs the function of one byte shift in the second row, two bytes shift in the third and three byte shift in fourth row.

3.3 MixColumn

MixColumn is a column-by-column operation on the state. Each column is a fourth-order polynomial over finite field GF(2^8) and multiplied modulo x^4+1 with the fixed polynomial b(x):

$$b(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \quad (2)$$

The matrix representation of above equation is given below.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad (3)$$

As a result of multiplication the four bytes in a column are substituted by the following equations:

$$\begin{aligned} s'_{0,c} &= (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c} \\ s'_{1,c} &= s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c} \\ s'_{2,c} &= s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c}) \\ s'_{3,c} &= (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}) \end{aligned} \quad (4)$$

3.4 AddRoundKey

AddRoundKey is the transformation where Round Key is added to the state by performing simple bitwise XOR operation, and each Round Key is derived by using a key schedule.

4. IMPLEMENTATION

Iterative [17] and pipelined architectures [18] are the two basic reconfigurable architectures that are commonly used for the implementation of encryption functions depending on type of application ranging from low to high speed. In this work we adopted pipelined architecture to implement AES encryption function in order to get best possible results in term of throughput. Further we have used the HLL approach to directly map our design on FPGA. In addition, our design is not device-specific; System Generator is highly scalable and can synthesis a design to different FPGA chips that leads to more flexible and fast design [19].

The block diagram of our full 128-bit AES pipelined architecture is shown in Figure 3, where each round is implemented separately in hardware by enclosing the four transformations as subsystems except in the last round where MixColumn transform has been eliminated. Initial round consists of only AddRoundKey transformation where input data is XORed with the initial Round Key value. Registers are placed at the end of each round forming hierarchical stages within each round of the algorithm. Each 128-bit transform; SubByte, MixColumn, AddRoundkey consists of four parallel copies of 32-bit modules. The detailed implementation and optimization of each is as follows:

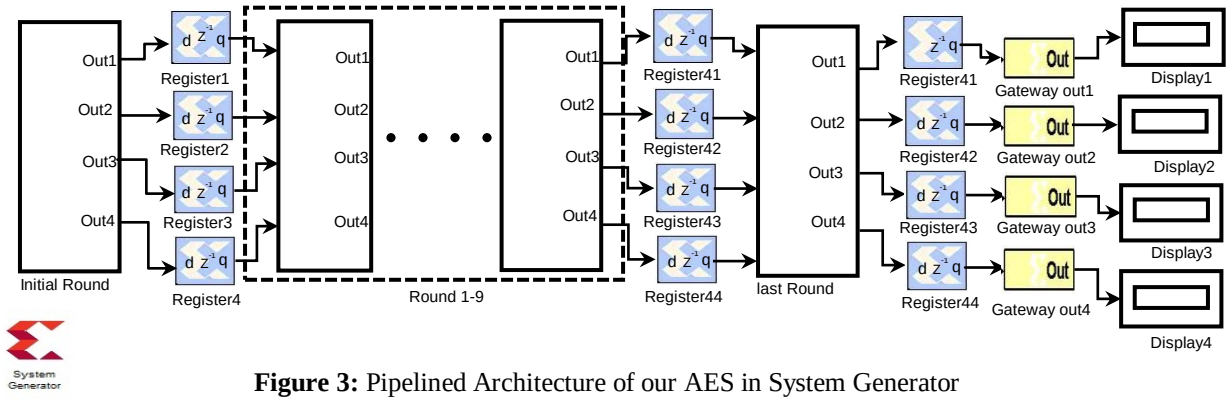

Figure 3: Pipelined Architecture of our AES in System Generator

4.1 SubByte and ShiftRow

There are two basic methods for generating SubByte of AES, either by using multiplicative inverse or by using memory based table lookup. We designed the SubByte using lookup based approach with the help of Dual-Port RAM to store 256 lookup values. Dual-Port RAM is configured as a ROM to access 8-bit lookup values corresponding to each 8-bit input addresses, for this we have operated the Dual-Port RAM in "no read on write" mode. Input of constant zero is

given to the data input pin & write enable pin of RAM as we have not required here to write the data. Lookup values are directly stored in the dual port RAM with the help of coefficient file in the form of decimal numbers.

The input bytes are delivered to the address pins addra [7:0] and addrb [7:0] of the BRAM while corresponding lookup values will be taken from the output pins A [7:0] and B [7:0] of the Dual-Port RAM. Input of 128-bit State is arranged in the four 32-bits words and each is directly given to the 4 32-bit SubByte block. The details of our proposed 32-bits SubByte architecture is shown in Figure 4 which consists of two Dual-Port RAMs.

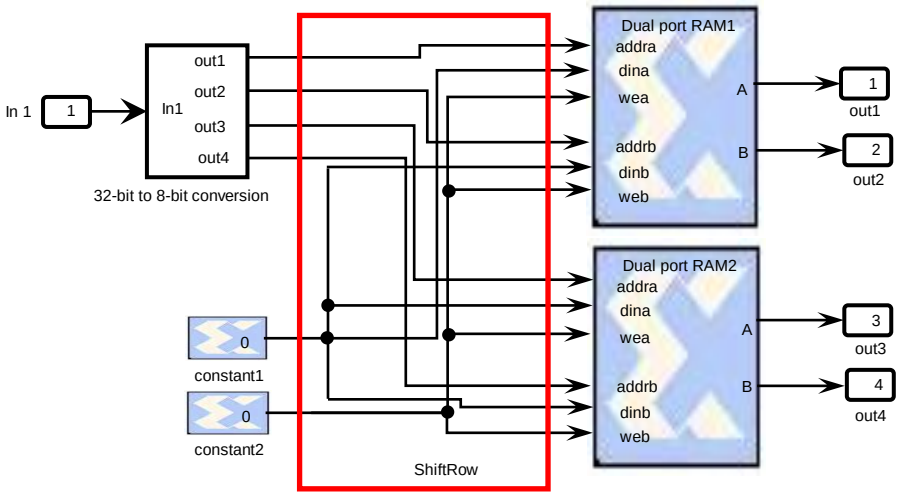

Figure 4: System Generator SubByte

Before applying the 32-bit data to each port of the Dual port RAMs for lookup, we first extract 8-bit data from 32-bit input. Because each RAM is able to lookup 8-bit data and we cannot apply the 32-bit data directly to Dual port RAMs. So for this we have used 32-bit to 8-bit conversion block. We can extract our desired 8 bit data by using the slice blocks. But this approach is not efficient in term of resource as it will require numerous slice resources. Therefore we redesigned this block by using custom logic in which we have used the right and left shift methodology to extract 8 bit data which helped us to save resources up to 1700 slices. The detail of 32-bit to 8-bit conversion module is shown in Figure 5. The ShiftRow transformation is also implemented within the same SubByte block to save the resources. This is done simply by rearranging the wires according to the shifted data as marked by red box in Figure 4 and applied directly to the RAMs.

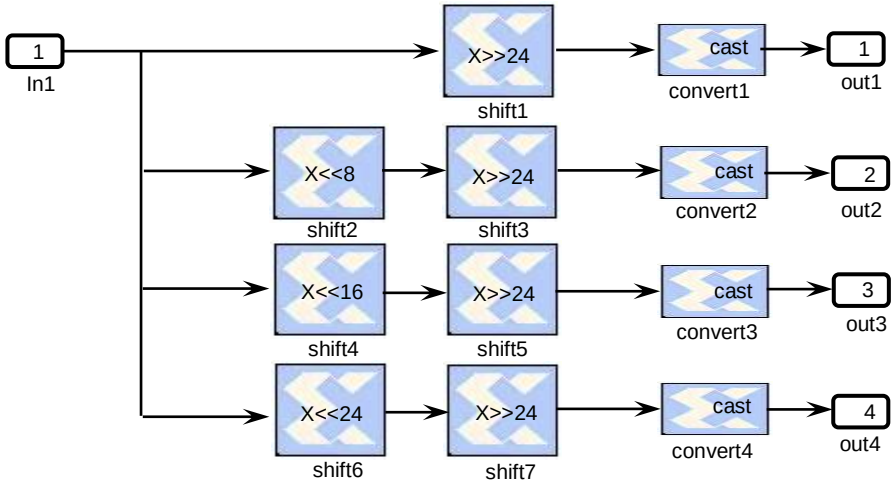

Figure 5: 32 bit to 8 bit Conversion

4.2 MixColumn

In MixColumn block we need a multiplier of 1, 2 and 3. However the implementation of multipliers is very costly in hardware. MixColum use multiply by constant and it can efficiently implemented using multiplier less approach to save

valuable hardware resources. We implement our multipliers by using shift and add algorithm to save the utilization cost. The architecture of MixColumn for first 32-bit data is shown in Figure 6, 32-bit data is passed directly through a connection wire when multiplied by number "1". Here in this block we have designed two multipliers to carry out the multiplication between input data and the numbers "2" and "3".

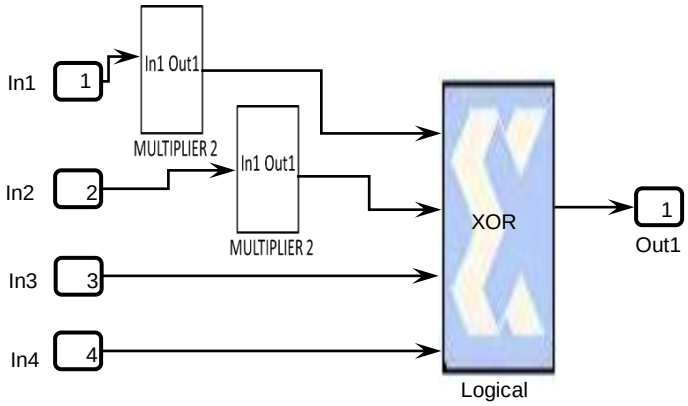

Figure 6: MixColumn

Multiplier 2 is designed by using shift block where left shift is applied to input data which results in multiplication by number "2" as shown in Figure 7.

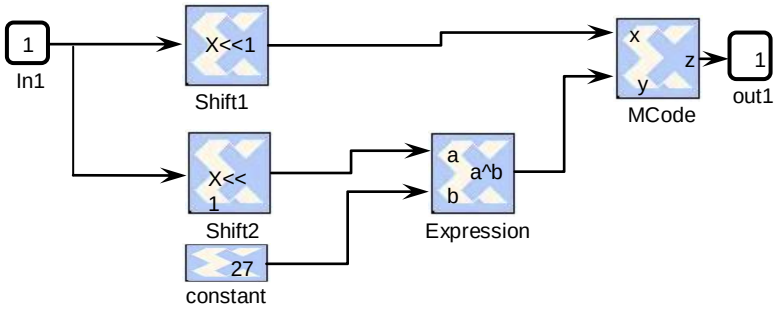

Figure 7: Multiplier 2

While Multiplier 3 is designed by dividing the number "3" into (2+1) where multiplication of data by number "2" is done by single left shift and the resultant number is then added to itself by using XOR operation as shown in Figure 8.

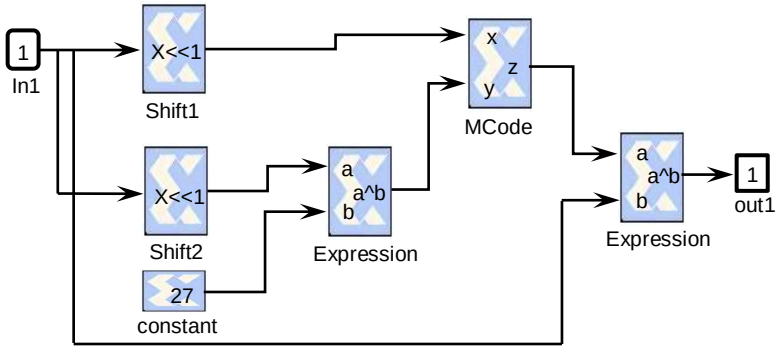

Figure 8: Multiplier 3

In both multipliers in order to satisfy the irreducible condition, the shifted data is XORed with the constant "27" (dec) in hex it is "1b" [2]. The mod 27 must be applied on the results of the multiplication in order to get the number within the range of 255. The MCode block is then designed to select the multiplication result "x" or "y", which is less than 255. It is

programmed by the following code.

```
Function z = x1max(x, y)
if x < 256
z = x
```

4.3 AddRoundKey

AddRoundKey transformation receives 16 bytes of data from MixColumn and performs XORing with the Round Keys. Round Keys are taken from FIPS-197 [2], converted to decimal numbers and are provided on the fly in the form of constants as shown in Figure 9. Here the expression blocks are used for XORing.

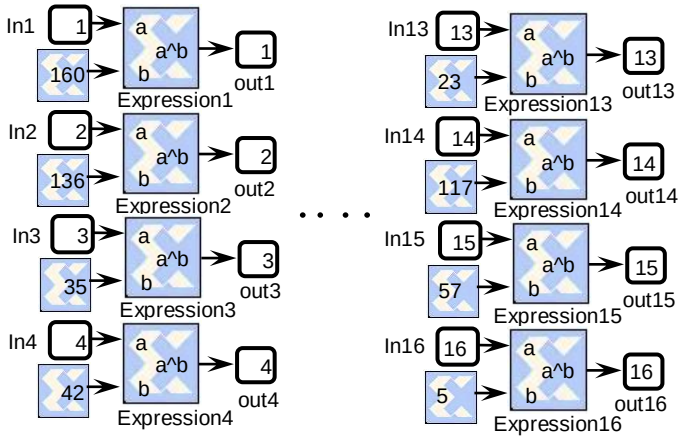

Figure 9: AddRoundKey

5. VALIDATION

Result outputs and behavior of design is verified by giving the pre-defined test vectors defined in AES FIP-197 [15] against standard AES output for testing and validation. Each module is implemented and validated individually. The functionality of each step is tested and verified one by one. Then the verification of output of each round is performed and validated the results after combining all the rounds in the main module.

6. IMPLEMENTATION RESULTS

Hardware implementation results are targeted for Xilinx Virtex-6 xc6vsx315t-3ff1156 FPGA. The design has been implemented using Xilinx System Generator tool in MATLAB to generate verilog code (.v file) alongwith test bench are finally synthesized and simulated using Xilinx ISE Foundation 13.1 and Mentor Graphic ModelSim, respectively. 128-bit AES pipeline architecture occupies 80 BRAM's for the implementation of SubByte and utilized 380 Slices for the remaining logic. The design operates on maximum frequency of 288.19MHz and offers high throughput of 36.864 Gbps as compared to previously reported designs.

Table 1 details the comparison results of previous AES implementations using HLL on different platforms. Parameters chosen for comparison are platform, to clearly indicate different high level implementation platforms, Data-path to indicate numbers of bit simultaneously processed by design. Operating frequency indicates maximum operational frequency of design and throughput to indicate performance of design.

It's evident from Table 1 that our performance results of proposed architecture gives better resource utilization and offers greater operating frequency as compared to all other previously reported high-level language tool implementations of AES. Mali et al [4] presents 128-bit AES implementation on Handle-C environment. This design operates at the maximum frequency of 74.4MHz and offers throughput of 7.76(Mbit/s). M. Askar et al. [5] presented 128-bit AES implementation using System C tool and System Crafter tool to translate the System C descriptions into hardware. It takes 153 machine cycles to process whole data and throughput of 90(Mbit/s). It reported 178 numbers of occupied slices for MixColumn and 197 number of slices for the implementation of ShiftRow, whereas our approach gives 380 slices for the whole architecture of AES. Osvik et al [9] previously reported 8-bit AES implementation on AVR and 32-bit AES

architecture on ARM platform that operates on the frequency of 124.6 cycles/byte and 34 cycles/byte respectively. Babu et al [10] implemented 128-bit architecture of AES using custom instruction method provided by ARM 7 with Keil platform. Hasamnis et al. [11] has given the high-level design architecture for 128-bit AES implementation where the algorithm is controlled through C-code written in NIOS II IDE that operates on frequency of 217.31020M Cycles. Next, Bos et al [12] reported the 128-bit AES implementation on NVIDIA graphics processing units (GPUs) and the Cell broadband engine. The GPU implementation delivers the throughput of 0.17 cycles per byte for encryption module and the Cell broadband engine offer the speed of 11.7 cycles per byte. In Biglari et al [13], the 128-bit pipeline architecture of AES is given on Maestro platform. This work presents tightly coupled encryption and round key generation modules in the main encryption module that enables the design to reach a throughput of 12.8 Gbps and runs at the speed of 100MHz. At the end, Mourad et al [14] presents the methodology which maps the AES design described in a high level language, Handel-C, to FPGA for low area consideration. This design reported 437 slices for encryption and offers the throughput of 1.716 Gbps.

All the implementations discussed above are not efficient enough for high speed applications as effective frequency of these designs ranges from 34 to 153 machine cycles while our design frequency is 288.19MHz. So, it is concluded that this design is faster than the other reported implementation of AES. In addition, the design utilized minimum number of resources i.e. 380 slices and offers high throughput of 36.864 Gbps as compared to previous one .

Table 1: Comparison Results of AES

Implementation	Platform	Data-Path	Frequency	Throughput
Mali et al [4]	Handel-C	128	74.4MHz	7.76(Mbit/s)
Askar et al [5]	SystemC	128	153 Machine Cycles	90(Mbit/s)
Osvik et al [9]	AVR	8	124.6 cycles/byte	-
Osvik et al [9]	ARM	32	34 cycles/byte	-
Babu et al [10]	ARM	128	-	-
Hasamnis et al [11]	NIOS II IDE	128	217.31020M Cycles	-
Bos et al [12]	NVIDIA (GPUs)	128	-	0.17 cycles/byte
Bos et al [12]	Cell broadband engine	128	11.7 cycles/byte	-
Biglari et al [13]	Maestro	128	100 MHz	12.8 Gbps
Mourad et al [14]	Handel-C	128	-	1.716 Gbps
Our Implementation AES-128	System Generator	32	288.19MHz	36.864 Gbps

7. CONCLUSION

In this work we present efficient implementation of AES in term of area and speed using High Level Language tool. Our methodology not only reduces the overall resource utilization but also provide good enough clock frequency. It provides the user friendly design for Matlab users with very short development time and better performance as compared to any software and hardware-software co-design implementations. The important feature of this work is that so far no any implementation of AES has been reported on Xilinx System Generator.

8. REFERENCES

[1] N. A. Saqib, C. K. Koc, A .D. Pérez, F. Rodriguez-Henriquez, “Cryptographic Algorithms on Reconfigurable Hardware”, Signals and Communication Technology, Springer, vol. 26, pp. 362, 2007.

[2] M. Mozaffari-Kermani, A. Reyhani-Masoleh, "Efficient and High-Performance Parallel Hardware Architectures for the AES-GCM," IEEE Transactions on Computers, vol. 61, no. 8, pp. 1165-1178, Aug. 2012.

[3] E. El-Araby, Saumil G. Merchant, T. El-Ghazawi, "A Framework for Evaluating High-Level Design Methodologies for High-Performance Reconfigurable Computers," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 1, pp. 33-45, Jan. 2011.

[4] M. Mali, F. Novak and A. Biasizzo, "Hardware Implementation of AES Algorithm", Journal of Electrical Engineering, vol. 56, pp 265–269, 2005.
[5] M. Askar and T. Egemen, "Design and SystemC Implementation of a Crypto Processor for AES and DES Algorithms", Information Security and Cryptology Conference with International Participation, Dec 2007.
[6] M. Lukowiak, S. Radziszowski and J. Vallino and C. Wood, "Cybersecurity Education: Bridging the Gap Between Hardware and Software Domains".
[7] F. Oboril, I. Sagar, M. B. Tahoori, "A-SOFT-AES: Self-adaptive software-implemented fault-tolerance for AES", On-Line Testing Symposium (IOLTS), IEEE 19th International, pp.104-109, 8-10 July 2013.
[8] Y. Wang, Y. Ha, "FPGA-Based 40.9-Gbits/s Masked AES with Area Optimization for Storage Area Network", Circuits and Systems II: Express Briefs, IEEE Transactions, vol.60, no.1, pp.36-40, Jan 2013.
[9] D. Osvik, J. Bos, D. Stefan and D. Canright, "Fast Software AES Encryption", FSE'10 Proceedings of 17th International Conference on Fast Software Encryption, pp 75-93, 2010.
[10] T. Babu, K. Murthy and G. Sunil, "AES Algorithm Implementation using ARM Processor", 2nd International Conference and workshop on Emerging Trends in Technology (ICWET) Proceedings published by International Journal of Computer Applications (IJCA), 2011
[11] M. Hasamnis, P. Jambhulkar and S. Limaye, "Implementation of AES as a Custom", Advanced Computing: An International Journal (ACIJ), vol.3, No.4, July 2012.
[12] J. Bos, D. Osvik, D. Stefan, "Fast Implementations of AES on Various Platforms", IACR Cryptology ePrint Archive, vol. 501, 2009.
[13] M. Biglari, E. Qasemi, B. Pourmohseni, "Maestro: A high performance AES encryption/decryption system", Computer Architecture and Digital Systems (CADS), 17th CSI International Symposium, pp.145-148, 30-31 Oct. 2013.
[14] O. Mourad, S. Lotfy, M. Noureddine, B. Ahmed, T. Camel, "AES Embedded Hardware Implementation", Adaptive Hardware and Systems, Second NASA/ESA Conference, pp.103-109, 5-8 Aug. 2007.
[15] FIPS-197, "Federal Information Processing Standards Publication FIPS-197, Advanced Encryption Standard (AES)", http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, October 1999.
[16] N.Anitha Christy and P.Karthigaikumar, "FPGA Implementation of AES Algorithm using Composite Field Arithmetic", International Conference on Devices, Circuits and Systems (ICDCS), pp. 713-717, 2012.
[17] A. C. Zigiotto, R. d'Amore, "A Low-Cost FPGA Implementation of the Advanced Encryption Standard Algorithm," 15th Symposium on Integrated Circuits and Systems Design, pp.191, 2002.
[18] S. Qu, G. Shou, Y. Hu, Z. Guo and Z. Qian, "High Throughput Pipelined Implementation of AES on FPGA", International Symposium on Information Engineering and Electronic Commerce, pp. 542-545, 2009.
[19] V. Elamaran and G. Rajkumar, "FPGA Implementation of Point Processes Using Xilinx System Generator", Journal of Theoretical and Applied Information Technology, vol. 41,pp. 201-206, July 2012.