A Sub-pipelined Implementation of AES For All Key Sizes

P.V.Sriniwas Shastry
Electronics and Telecommunication Department
Cummins College of Engineering for Women
Pune, India

M.S. Sutaone
Electronics and Telecommunication Department
P.I.E.T's College of Engineering
Pune, India

***Abstract*— In this paper we have proposed three sub-pipelined architectures for Encryption, Decryption and Joint Encryption and Decryption (E/D). These architectures were implemented on Vertex-4 device. The use of Block RAM available in the device for key expansion as well as for the S-Boxes resulted in utilizing less slices and getting higher throughput in all three cases compared to the literature available till date. The encryption architecture clocked a throughput of 35.65Gbps using only 4823 slices while the decryption architecture achieved 33.73Gbps using 6847 slices only. The device used is XC4VLX60. The joint E/D architecture achieved a throughput of 31.62Gbps. Retiming techniques used to balance the computational path delays of encryption and decryption data paths.**

***Keywords*— Sub-pipeline, S-Box, AES, Block RAM, Joint Encryption and Decryption**

I. INTRODUCTION

National Institute of Standards and Technology (NIST) announced the Rijndael as the Advanced Encryption Standard (AES). NIST with the help of cryptographic research community evaluated and found AES-Rijndael [1] to be efficient on VLSI implementations. The algorithm was proposed for 128, 192 and 256-bit key lengths. AES outperforms Data Encryption Standard (DES) due to larger key sizes and larger number of rounds performed while generating cipher text from plain text and key. There are quite a good number of AES implementations on Vertex FPGAs [2][3][4][10] reported and most of them used look-up tables for implementing S-Boxes as there was ample of Block RAM resources available. Also due to the less access time is required compared to on the fly computation of the S-Boxes, FPGA implementations preferred look-up tables for S-Boxes. The algorithm provides convenience in implementing it on 8-bit or 32-bit or higher, bus organization platforms.

All the previous implementations were either tried to produce a throughput optimized or area optimized designs and accordingly they have selected the architectures. Rolled architectures provided an area optimized implementations [6][11] while could not with stand the demand for higher throughput. While implementations based on pipelined architectures could provide very good throughput but with certain area over heads. Hence instead of comparing either on area or throughput parameters, throughput-slice ratio (T/S) became one of the justified parameter for comparison. Over recent years many of the researchers even tried to achieve higher sharing of hardware between encryption and decryption architectures [15] in order to achieve better T/S ratio. Pipelined partial rolled architecture [13] implementation was another effort to improve T/S ratio.

In this paper we implemented a sub-pipelined architecture with an eye on getting better T/S ratio than the earlier implementations to date. Our architecture performs encryption and decryption for all the three key sizes. While implementing, we have also tried to maintain the latency at the lowest possible with sub-pipelined architecture by clubbing some stages into one and hence balancing the clock cycles and optimally utilizing the available clock period in the data as well as in the key expansion path. This was achieved by proper retiming data path and key expansion path so that the combinational path delay in each clock cycle is approximately same.

The rest of the paper is organized in the following manner. Section II describes the AES algorithm briefly and helps in identifying the features available with the algorithm which could be possibly utilized for design optimization. Section III elaborates our approach towards the AES implementation and our proposed architecture. Section IV compares our results with existing ones and which is followed by concluding remarks.

II. AES ALGORITHM

The AES is a symmetric key block cipher algorithm [1] that processes data in 128 bit blocks and can operate with keys that are 128,192 or 256-bits long. The number of rounds performed in the algorithm depends upon the key size. The algorithm performs 10, 12 or 14 rounds of iteration for 128, 192 or 256 bit key sizes respectively to generate final cipher text in encryption. Same number of rounds is performed in decryption algorithm. Each of these rounds performs four types of byte oriented, word oriented and block oriented operations namely as *substitute byte, shift row, mix column* and *add round key* in sequential manner. The outcomes of the last operation of add *round key* is given as input to the first operation *substitute byte* of the next round. The final round consists of only three operations and excludes *mix column.* Figure 1 shows the block diagram of encryption as well as decryption data path.

A. Substitute Byte

This operation primarily substitute each byte of the input state with a substitution byte, computed by finding multiplicative inverse in GF(2^8) and applying affine transformation. This substitution byte can be computed on the fly using combinational logic [12] in GF(2^4). The substitution from S-Box for each byte of the input state could be done byte by byte or all sixteen bytes concurrently. While implementing the algorithm on FPGA, look up tables are generally preferred as the devices available has Block RAM which can host these S-Boxes instead of using the slices. If the whole block of 128 bits has to be performed simultaneously then the number of look-up tables required for S-boxes would be 16. Each of the S-Boxes requires 256 bit memory.

The pipeline architecture is selected for implementation, in all 160, 192 or 224 such S-boxes would be required for 128, 192 or 256 bit key lengths respectively. The decryption algorithm would require totally different set of Inverse S-Boxes in same number.

B. Shift Rows

Shift row cyclically shifts left one byte of second row, two bytes of third row and three bytes of fourth row respectively. While in case of decryption it would be right shift in the similar manner and both cases first row is not shifted.

C. Mix Column

Each column of the row shifted state is treated as a polynomial of GF(2^8) and multiplied modulo x^4+1 by a fixed polynomial a(x)

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} \qquad (1)$$

$$\begin{pmatrix} S_{0i}' \\ S_{1i}' \\ S_{2i}' \\ S_{3i}' \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} * \begin{pmatrix} S_{0i} \\ S_{1i} \\ S_{2i} \\ S_{3i} \end{pmatrix} \qquad (2)$$

The modulo multiplication can be realized as matrix multiplication of a column with a square matrix of (2). While finding inverse Mix Column the fixed polynomial is

$$a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\} \qquad (3)$$

$$\begin{pmatrix} S_{0i}' \\ S_{1i}' \\ S_{2i}' \\ S_{3i}' \end{pmatrix} = \begin{pmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{pmatrix} * \begin{pmatrix} S_{0i} \\ S_{1i} \\ S_{2i} \\ S_{3i} \end{pmatrix} \qquad (4)$$

Even this modulo multiplication can be realized as matrix multiplication of a column with a square matrix of (4). The inverse Mix Column operation can share the matrix multiplication module of (2) as presented in [2].

D. Add Round Key

The Add Round Key is just XOR of each block after mix column with the round key generated by the Key Expansion data path. This operation is same in decryption data path as the XOR operation is inverse to itself.

III. THE PROPOSED ARCHITECTURE

The pipelined architecture suggests introduction of registers after every round [14]. As mentioned in the earlier section each round has four operations to be performed. These rounds when implemented one after one will result into a large combinational path delay which will restrict shorter clock cycle time. Sub-pipelined architecture is like further fine grain pipelining where registers are introduced even in between the operation within and round. This allows the clock to be increased but at the cost of increased latency. The throughput gets improved compared to pipeline architecture.

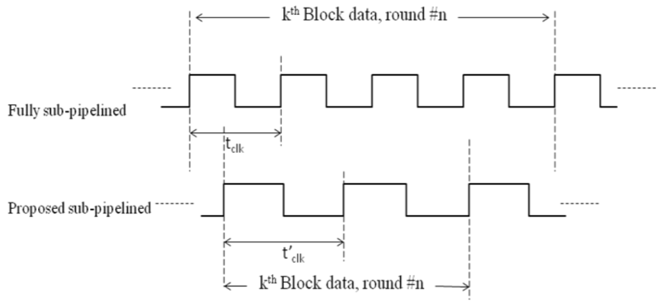


Figure 1. Cycle time for fully sub-pipelined and our proposed subpipelined

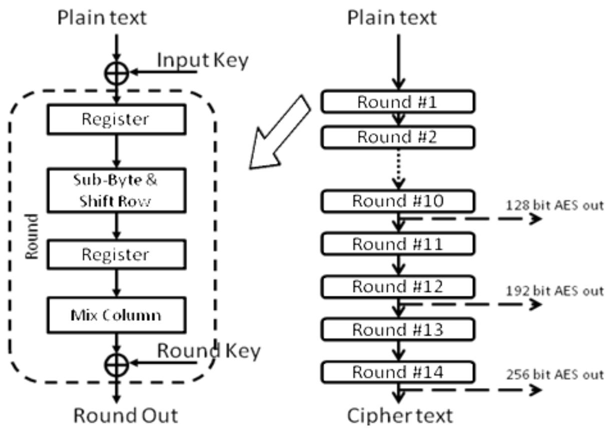
Let the t_{sub}, t_{shft}, t_{mixcol}, and t_{addkey} be the computation time of substitute byte, shift row, mix column and add round key respectively. After implementing the design the conducting static timing analysis it found that these timing were 2.02ns, 0.49ns, 2.65ns and 0.56ns respectively. These timing parameters were obtained after performing certain iterations of optimization while place and route. A fully sub-pipelined architecture would require putting register after every sub process in the each round. This would have directly employed 41, 49 or 57 registers for 128, 192 or 256 bit encryption respectively. Also this would increase the latency of the implementation. The clock time t_{clk} could be calculated as in the (5). The time parameter t_δ includes the setup and hold time of the registers and all other contingencies due to clock distribution network.

$$t_{clk} = t_\delta + \max\{ t_{sub}, t_{shft}, t_{mixcol}, t_{addkey}\} \qquad (5)$$

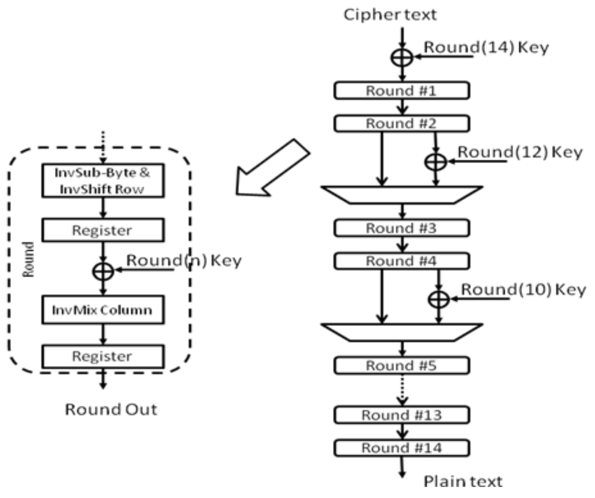
This implementation resulted in 330MHz clock frequency, as computation time of mix column was the largest. After performing retiming on the block architecture [16], reduced the registers required and instead of four consecutive clock cycles it used only two consecutive clock cycles for each round. The substitute byte and shift row were clubbed as a single sub process while mix column and add round key were clubbed together as a single sub process stage. This retiming resulted in clock computation mentioned in (6).

$$t'_{clk} = t_\delta + \max\{(t_{sub} + t_{shft}), (t_{mixcol} + t_{addkey})\} \qquad (6)$$

Even though this reduced the clock frequency and in turn the throughput, but the numbers of registers required were less and also the latency was less. The registers required in this implementation were 21, 25 and 29 for 128bit, 192bit and 256bit encryption respectively. The retiming of the design resulted into more optimal utilization of clock period which has been demonstrated in Figure 1.



(a)



(b)

Figure 2. (a) Encryption Data Path (b)Decryption Data Path

A. 128/192/256 bit AES Encryption Data Path

In our proposed round architecture in Figure.2(a) for encryption data path we have introduced one register in the beginning and the second one after Shift Row is performed. After the Mix Column and Add Round Key are performed the functionality of pipelining register is performed by the initial register of the next Round architecture. Hence only two registers appears in each round. Total of 16 S-Boxes are required for each round so that all 16 bytes are substituted in single clock cycle. Shift Row does not require to employ shift registers because after substitution of each byte the placement of these bytes are done at the new shifted location in the next register. This simplifies the Substitute Byte and Shift Row operation in a single clock cycle. A multiplier less architecture has been used for the Mixed Column. Multiplication by {02} has been realized by modulo shifting the byte once left, while multiplication by {03} by shifting and XOR to itself.

The number of rounds needed for the 128, 192 and 256 bit key lengths are different, which requires an arrangement in the architecture to take the output after 10th, 12th and 14th round and switch to the output port on selection of key size using multiplexer. Including the output registers, in all 30 registers are employed in the architecture.

B. 128/192/256 bit AES Decryption Data Path

Unlike in encryption the decryption data path make use of already expanded keys which are in the key registers while performing encryption. Figure.2(b) illustrates the architecture of decryption data path for all key sizes. As the number of rounds for different key sizes are different hence the decryption round starts at the appropriate stage in the architecture. The number of inverse S-Boxes needed is 16 so that the inverse Substitute Byte is performed in one clock cycle. Here again the Inverse Substitute Byte and Inverse Shift Row has been merged into a single clock cycle by placing the substituted bytes in the shifted location of the next registers.

In order to avoid multiplier for implementing eq. (3), {0b}, {0d}, {09} and {0e} has been realized as in the eq.(7).

$$\{0b\} = \{03\} + \{08\}$$
$$\{0d\} = \{01\} + \{08\} + \{04\}$$
$$\{09\} = \{01\} + \{08\}$$
$$\{0e\} = \{02\} + \{08\} + \{04\} \qquad (7)$$

This also helps in reusing the Mix Column architecture of encryption data path. Each byte is shifted thrice to left and the bytes to be multiplied by {0d} or {0e} are further shifted twice to left before using Mix Column stage of encryption path.

C. Combined Encryption and Decryption Data Path

While combining the encryption and decryption data path the architecture reuses all the registers as well as Mix Column stage of the architecture. The multiplexers placed in between are used to switch between the encryption or decryption data paths. The mode selection control line is used as select line for these multiplexers. Figure.3 shows the round architecture for this combined encryption and decryption data path.

IV. IMPLEMENTATION RESULTS

The three proposed architectures were implemented on Vertex-4 FPGAs. The encryption with key expansion could fit into XC4VLX60 device. We achieved the throughput of 35.65Gpbs at the clock frequency of 278.5MHz. The keys generated by the key expansion unit were stored in the Block RAM whereas the S-Boxes utilized Block RAM in ROM mode. This resulted in to utilizing only 5% of flip flops from the slices. The total number of slices employed for this architecture was 4823(18%).

The decryption data path with key expansion unit achieved 33.73Gbps at the clock frequency of 263.5MHz. The architecture could be successfully implemented by employing mostly Block RAMs rather than flip flops of logic slices. The device utilization reports mentions use of 6847(25%) of slices and around only 4% of flips flops from slices. The decryption data path required more hardware resources then encryption data path. The reason for getting higher latency than encryption data path was due to waiting for the key expansion to be completed before the actual decryption round starts.

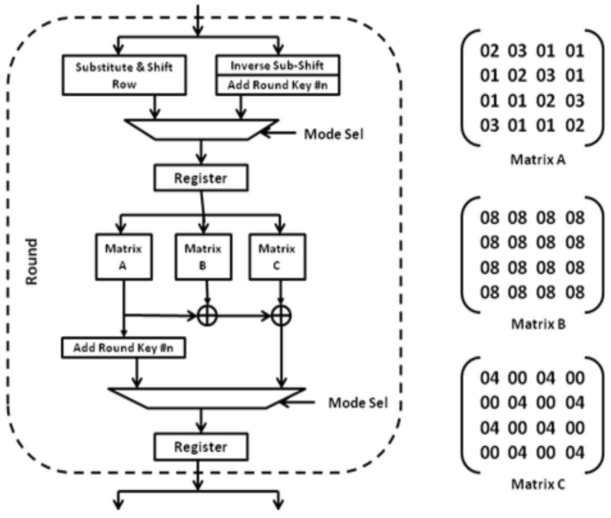


Figure 3. Joint Encryption and Decryption Data Path

The third architecture of combined encryption and decryption with single key expansion unit was successfully clocked at 247MHz providing a throughput of 31.62Gbps. XC4LX60 device fall short to the number of Block RAMs needed, hence the design was pushed into XC4VLX80 device. The total number of slices used was 7667(21%) while 80% of Block RAM in 16 bit mode was employed. The computational delays of both the encryption and decryption path were balanced such that the cycle time between two registers is approximately same.

The timing simulations were performed on ModelSim SE Version 6.6. and Design Synthesis and Implementation was done Xilinx ISE 9.2.

CONCLUSION

The Table I. shown below gives the comparison of our results with the available published literature. The comparison group selected in the table has implemented their design on the Vertex family devices and used pipelined architectures. Comparing only throughputs would have penalized results with less number of slices, hence we have compared Throughput to slice ratio (T/S). Our design has achieved highest T/S ratios for all the three architectures.

Due to the use of less number of slices and higher utilization of Block RAM available on FPGA has enabled us to push the design on smaller device like XC4VLX60 for encryption and decryption. The joint encryption and decryption needed higher number of Block RAM cells hence we have used XC4VLX80 device. Considering the resources utilized and the throughput achieved our design has best results in overall. All the static timing obtained was due to few iterations of optimization while placing and routing the design. A further high level effort of optimization was tried and found setup and holds time violations at many places hence the results were discarded

TABLE I. RESULT COMPARISON

Design	**Device**	**Clock Frequency (MHz)**	**Throughput (Gbps)**	**Device Utilization (slices)**	**T/S**
[2] Enc	XCV1000	168.4	21.56	11022	1.96
[3] Enc	XC2VP20	168.3	21.54	5177	4.2
[5] Enc	XCV2000	241.3	30.88	4626	6.67
[5] Dec	XCV2000	128.07	12.9	19125	0.67
[6] Enc	XCV4LX200	250	32	86806	0.37
[7] Enc	XCV2V600	305.1	39.053	10762	3.66
[4] Enc	XCV2V400	184.2	23.57	16938	1.39
[8] Enc	XC5VLX85	576.06	73.737	22994	3.21
[9] Enc	XC4VLX80	500	64	8901	7.19
[10] Dec	XC4VLX60	180.39	23.09	20155	1.15
Our Designs					
Enc	XC4VLX60	278.5	35.648	4823	**7.39**
Dec	XC4VLX60	263.5	33.728	6847	**4.92**
E/D	XC4VLX80	247	31.616	7667	**4.12**

REFERENCES

[1] NIST, Announcing the Advanced Encryption Standards (AES), Federal Information Processing Standards Publication 197, November 2001.

[2] Xinmiao Shang, Keshab K. Parhi, "High speed VLSI architectures for the AES algorithm", IEEE Transaction on Very Large Scale Integration Systems, Vol.12, No.9, 2004, pp.957-967.

[3] Alizera Hodjat, Ingrid Verbauwhede," A 21.54Gbits/s fully pipelined AES processor", IEEE Symposium on Filed Programmable Custom Computing Machines (FCCM), 2004 .

[4] J.Zambreno, D.Nguyen, A.Choudhary, "Exploring Area/Delay tradeoffs in an AES FPGA implementation", 2004, pp.575-585.

[5] Nalini.C, Nagaraj,Anand Mohan, Poornaih D.V,V.D.Kulkarni, "An FPGA based performance analysis of pipelining and unrolling of AES algorithm",Proceedings of International Conference on Advacned Computing and Communications, 2006, pp.477-482.

[6] Chih-Penh Fan, Ju-Kui Hwang,"Implementations of high throughput sequential and fully pipelined processors on FPGA", Proceedings of International Symposium on Intelligent Signal Processing and Communication systems, 2007, pp.353-356.

[7] Issam Hammad, Kamal El-sankary, Ezz El-Masry, "High speed AES encryptor with efficient merging techniques", IEEE Embedded systems Letters, Vol.2 Issue 3, 2010, pp.67-71.

[8] Shanxin Qu, Guochu Shou, Yihong Hu, Zhigang Guo, Zongjue Qian, "High throughput, pipelined implementation of AES on FPGA", International symposium on Information Engineering and Electronic Commerce, 2009, pp.542-545.

[9] Dong Chen, Guochu Shou, Yihong Hu, Zhigang Guo, "Efficient architecture and implementation of AES", 3rd International Conference on Advanced Computer Theory and Engineering, 2010, pp.295-298.

[10] M.R.M.Rizk, "Optimized area and optimized speed hardware implementation of AES on FPGA", 2nd International Design and Test Workshop, 2007, pp.207-217.

[11] Monjur alam, Santosh Ghosh, Dipanwita RoyChoudhary, Indranil Sengupta, " Single chip encryptor/decryptor core implementation of AES alfotrithm", 21st International Conference on VLSI Design, 2008, pp.693-698.

[12] P.V.S.Shastry,Anuja Agnihotri, Divya Kacchwaha, JayasmitaSingh, M.S.Sutaone, "A combinational logic implementation of S-box of AES", 54th IEEE International Midwest Symposium on Circuits and Systems, Seoul, Korea, 2011.

[13] Hui Qin, Tsutomu Sasao, Yukihiro Iguchi, "An FPGA design of AES encryption circuit with 128-bit keys", GLSVLSI 2005, Chicago, pp.147-151.

[14] N. Sklavos, O. Koufopavlou, "Architectures and VLSI implementation of the AES-proposal Rijndael", IEEE Transaction on Computers, Vol.51,No.12,2002,pp.1454-1459.

[15] S.-F.Hsiao, M.-C.Chen, "Efficient substructure sharing methods for optimizing the inner-product opertaions in Rijndael advanced encryption standard", IEE Proceedings of Computers and Digital Techniques, Vol.152,Issue.5,pp.653-665.

[16] Keshab K. Parhi, "VLSI Digital Signal Processing Systems, Design and Implementation", Wiley India (P) Ltd.