# Application Specific Scalable Architectures for Advanced Encryption Standard (AES) Algorithm

S.S.Naqvi, S.R.Naqvi, S.A Khan, S.A. Malik
Department of Electrical Engineering
COMSATS Institute of Information Technology
Department of Electrical Engineering, Chak Shazad Campus Islamabad
Pakistan
{saud_naqvi, shaihdk, smalik}@comsats.edu.pk, rameeznaqvi83@hotmail.com

*Abstract:* - The work presented proposes two diverse FPGA based architectures with high-speed and low area constraints for suitable implementation of Advanced Encryption Standard (AES). The main focus of this paper is to compare different design architectures existing in literature with the proposed ones, based on application specific constraints. The high speed design presented here proposes a good engineering solution to high speed applications where area constraint can not be totally neglected. The high speed design manages to achieve a reasonable 6 Gbps throughput despite of the fact that it only covers mere 5800 slices in area .Low area architecture achieves a decent throughput of 1.98 Mbps with low slices count of 297. Some common applications of high speed design include broadband switches and firewall, whereas the low area design mainly focuses on compact applications like PDAs and cell phone in which area and power constraints are critical. Both the designs are implemented and tested using a Xilinx Spartan-III (XC3S2000) target device.

Key-Words: - Advanced Encryption Standard (AES), Subpipelined, High Speed, Low Area, Unrolled architecture, Cryptography, Throughput, FPGA, and Data Encryption Standard (DES).

## 1 Introduction

Cryptography has been used for years and plays a vital role in security of data transmission [1]. In October 1997, Advanced Encryption Standard (AES) replaced the older Data Encryption Standard (DES) and two years later Cipher-Rijndael algorithm was named AES Algorithm. Its major applications include smart cards, high-speed networks, ATMs and DVDs [1]. There are two possible hardware implementations of AES algorithm. First is to provide higher throughput and second is to utilize as small area on chip as possible. This paper presents both of the possible implementations, one favoring higher throughput and the other with a low area constraint. The main focus of this work is to investigate various design architectures associated with the Advanced Encryption Standard (AES) in context to their specific application areas. Hardware designs and architectures have been the major focus for AES afterwards. Enormous research efforts have been done in order to find suitable hardware implementation specific to certain applications. Choice of a design for a particular application depends on system specifications such as speed and the number of resources utilized in terms of area [2]. Common terminologies that can be found in literature describing system resources and specifications for AES architectures are area and throughput.

Previous efforts [1],[2],[3],[11],[12],[13] done for high speed pipelined architectures mainly focuses on high throughput constraint without giving much focus to area constraints. The work presented here proposes a high speed sub-pipelined architecture which also focuses on area constraint in order to provide an overall good engineering design. Three optimization approaches are commonly used in term of hardware architectures: pipelining, subpipelining and loop unrolling. This design employ loop unrolling architecture thus maintaining a high throughput while the low area is achieved using Non-LUT based approaches thus involving complex hardware for inversions in Galois field GF ($2^8$). This design mainly focuses on applications where area is also a secondary constraint in addition to high speed.

A detailed comparison of some low area architectures [2],[4],[5],[10] is also presented. Most of the quoted designs are built around a 32 bit datapath, but this work proposes a very compact architecture based on 8-bit design. The proposed design is mainly based on the work proposed by Tim in [2] with major modifications made in the design of ShiftRows. Some major modifications are also proposed in mixcoloumns module which considerably reduces the count of slices. Again the

design employs Galois field GF ($2^8$) inversions thus reducing overall area in slices as compared to LUT based approaches.
The rest of the paper is organized as follows:
Section 2 briefly describes the AES algorithm. Section 3 gives the overall design and architectural modification details, for both high speed and low area specialized architectures. Section 4 provides a performance based comparison of proposed architectures with other FPGA based designs. Section 5 concludes with comparing different quoted design architectures with the proposed ones based on their target applications.

## 2 AES Algorithm

Unlike asymmetric-key algorithms, AES is a symmetric key cipher thus only a single key is used at the encryption as well as decryption end. Data block is limited to 128 bits, while the key length is adjustable between 128, 192 or 256 bits respectively. The data block is arranged in a 4x4 array called the state [1] on which all the subsequent internal operations are performed. Every byte present in the state block is considered to be an element of GF($2^8$). The specific irreducible polynomial used by the AES algorithm is $p(x) = x^8 + x^4 + x^3 + x + 1$. Encryption can be achieved by applying the four transformations in each round: SubBytes, ShiftRows, MixColumn, and the AddRounKey. However final round does not involve the ColoumnMixing transform. Subbyte is a non-linear transformation and acts separately on each byte of the state. Main operations include computing the multiplicative inverse for each byte subsequently followed by affine transformation [1]. SubBytes can be mathematically described by

$$S'_{i,j} = MS'_{i,j} + C \qquad (1)$$

ShiftRows is a transformation which involves simple shifting of bytes. First row of the state remain unchanged, while the second, third and fourth row undergo cyclic shifting by one, two and three bytes respectively [1]. The MixColumn transformation can be mathematically expressed as

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} \{02\}_{16} & \{03\}_{16} & \{01\}_{16} & \{01\}_{16} \\ \{01\}_{16} & \{02\}_{16} & \{03\}_{16} & \{01\}_{16} \\ \{01\}_{16} & \{01\}_{16} & \{02\}_{16} & \{03\}_{16} \\ \{03\}_{16} & \{01\}_{16} & \{01\}_{16} & \{02\}_{16} \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \quad 0 \le c \le 4 \quad (2)$$

Finally the AddRounKey is a simple bit-wise XOR operation of the modified state array and the Cipher key. The decryption process is the inverse process of encryption and uses exactly similar structure. The only change is the sequence of transformations involved in the decryption process. This feature can be exploited by resource sharing in encryptor and decryptor architectures [6],[14],[15]. Key expansion process produces 4-byte words $(w_0, w_1, ....., w_{4(Nr+1)-1})$ [6]. The initial key comprising of Nk words is used to generate the rest of the words. Nk can be computed to be 4, 6 or 8 depending upon respective key lengths (128, 192 or 256-bits respectively). Detailed description of the key expansion process can be found in [1].

## 3 Proposed Application Specific High Speed and Low Area Designs

This section provides detailed architectural designs for both application specific implementations. Some major modifications in the previous designs and subsequent performance improvements are highlighted. Both the implementations are specifically meant and best suited to certain applications.

### 3.1 Proposed High Speed Unrolled Architecture

This section presents the design of high speed unrolled architecture for both encryption/decryption. The key factors of the design are:

#### 3.1.1 32 bit main data path / 128 bits key input

The data input/output ports of the system are 32 bits wide. This architecture was chosen as it proved to provide the best utilization of the Spartan-3 FPGA resources. It was chosen to be column oriented, mainly to allow a simple and efficient implementation MixColumn which in turn facilitates performance improvements (in terms of Throughput VS Area) in comparison to [1]. The key data input is 128 bits wide facilitating a simple key input procedure for the user. Calculated roundkeys are distributed through a 32 bit internal bus.

### 3.1.2 Subpipelined architecture

Every individual module of our architecture is subpipelined as in [1], providing the potential of higher clock speeds, thus higher data throughput.

### 3.1.3 On the Fly round keys calculation

A fundamental process of AES encryption/decryption is key expansion. Our architecture inputs a key from the user and calculates the 10 roundkeys on the fly and supplies them when and where required.

### 3.1.4 Continuous data flow

All loops are completely unrolled, thus there is no module waiting to be reused in the course of the processing of a single block. The flow of data is completely linear. On condition that the key remains the same, it will keep calculating one column per clock cycle indefinitely.

### 3.1.5 Implementation of Individual modules

This section explains all the individual modules required to be executed in a single round. The individual modules like SubBytes and MixColumn are implemented according to Zhangs architecture [1]. The design of ShiftRows and AddRoundKey are modified in order to improve overall design of individual modules in comparison to similar approaches. The design modifications allow reasonable throughput achievements considering the small area covered. Detailed architecture for individual modified modules is discussed below. ShiftRows is the module which performs the ShiftRows step in encryption and inverseShiftRows in decryption. Both these steps involve no mathematical operations. They are simple horizontal rearranging of bytes. In our column oriented 32bit architecture, the rearranging of bytes within the same row becomes an issue requiring some consideration. Unlike [1],[2],[3],[16],[17] our architecture makes use of a programmable tap 16bit shift register, which performs two functions. The first is to store the bytes of each row as they come in one by one and output them in the correct rearranged order required by ShiftRows/invShiftRows.

As the amount of clock cycles for doing that is not the same for each row, another shift register is daisy chained in order to provide programmable delay. Fig.1 shows the general architecture of the row shifting module. This will ensure that all bytes at the output belong to the same (transformed) column and the result is uniform. Since our architecture makes use of seven programmable elements, an appropriate mechanism is required to control their behavior. This is done by an accurately designed state machine which determines the tap position and delay for each row, according to the column being processed and the selected mode (encryption or decryption). It is obvious that synchronization is a key concept of this design. On the module level, it is achieved by a single input, which asynchronously resets the state machine back to its initial state.

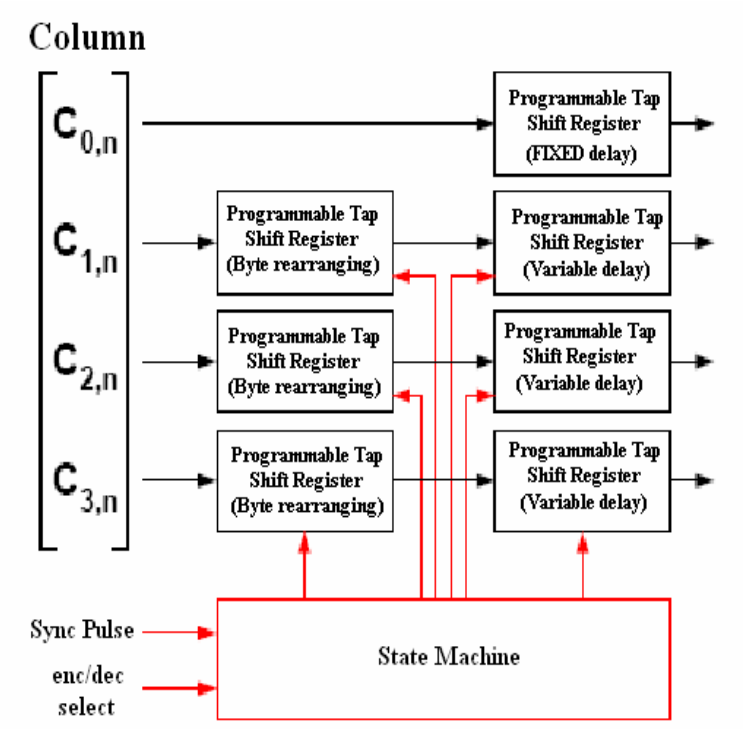


Fig.1 General Architecture for row shifting module

Proper synchronization requires a single, accurately timed pulse prior to every data feed session. By exploiting the resource capabilities of SRL16 “16-Bit Shift Register Look- Up Table” and by proper scheduling and flow of data through the row shifting block, we managed to improve the overall performance of the system by restricting the number of cycles consumed by the row shifting to a very low count as compared to [1],[3]. Considerable amount of area optimization was also achieved by using SRL16 as it requires minimal slices. The most crucial issue concerned to Roundkey addition is the proper synchronization which will ensure that every clock cycle, the current 32 bit column of the state will always be added to the corresponding column of the round key. Our AddroundKey module is correctly “tuned” by a carefully timed pulse prior to

the arrival of the input data stream as shown in Fig.2.
The main characteristics of our implementation which makes this module comparable to the previous implementations in terms of throughput are given below:

- Processes 32 bits (a single column) per clock cycle. Sub-pipelined, therefore increased throughput.
- Roundkey storage is managed directly with a series of registers thus providing fast access. Whenever a change of key occurs these subsequent registers are automatically overwritten.
- Synchronization is achieved by a single pulse per uninterrupted data stream. This refers to a single round's AddRoundKey module.

For all 11 AddRoundKey modules, 11 pulses will have to be produced. Thus mitigating the problem of synchronization.

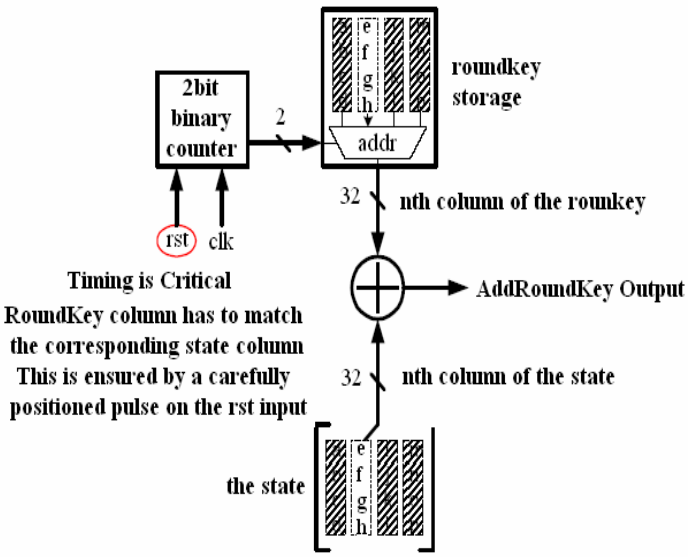


Fig.2 General Architecture for AddRoundKey module

Fig.3 shows the architecture for key expansion module. The stage of Key expansion includes roundkey generation from the plain Cipher key. There were two possible approaches to achieve this task.

Roundkeys can be either generated beforehand and stored in memory or generated on the fly. In our design we chose to generate the Roundkeys on the fly as in [2] and the obvious advantage was to improve throughput of the overall design.

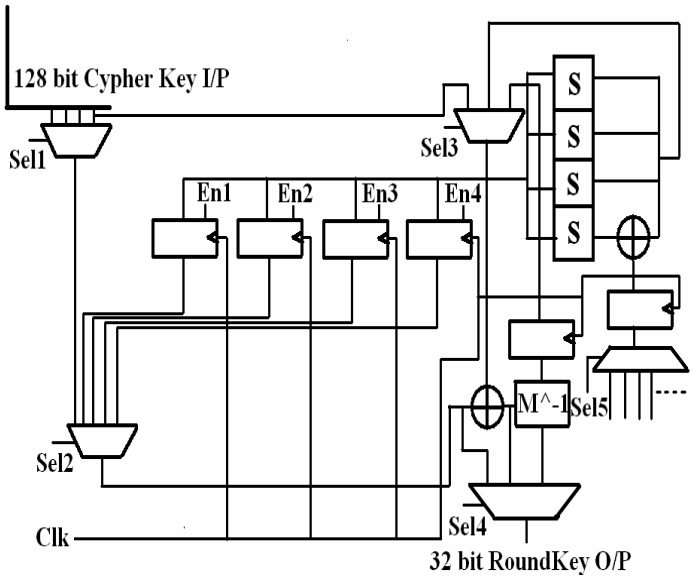


Fig.3 Architecture for Key expansion module

However it was a tradeoff between area, speed and power on the cost of increasing the complexity of the overall architecture. The main motivation behind generating keys on the fly was target applications in which the key changes at regular intervals [19].

### 3.2 Proposed Low Area Implementation

Modern communication systems require faster processing speeds i.e. greater throughput and at the same time small area utilization. It has always been designers challenge to find an appropriate tradeoff either in speed and/or area utilization in order to achieve an optimal solution.

Before making a final decision about our architecture, we thought of implementing a smallest possible design and then determine the stages where it might be possible to tradeoff area against throughput. Previous research [2],[4],[5],[18] suggested several architectures which we analyzed and found Tim's solution probably the most optimum one.

Fig.4 shows proposed organization of modules, where feedback logic is used which is believed to minimize additional delays and pipeline registers thus reducing slices count significantly.

First multiplexer allows selection of any row or column at a time and the second multiplexer allows to select any operation as required. On completion

of every step, updated values or intermediate results are loaded back into state memory.

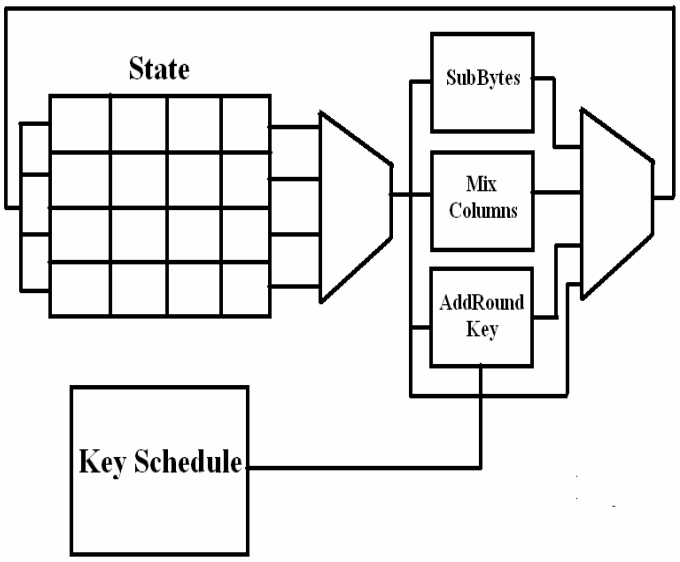


Fig.4 General Organization of Proposed Modules

Fig.5 shows the organization of shift registers and MixColumn unit that enable the data from state memory to be moved into the unit in round robin sequence.

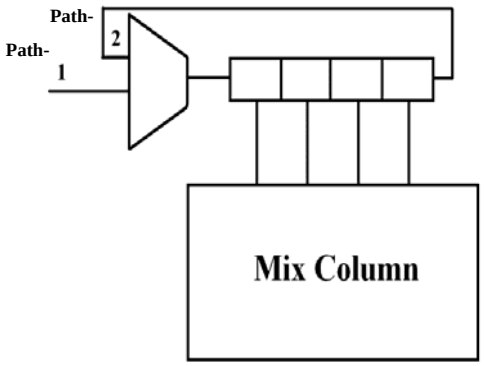


Fig.5 Organization of Shift Registers and MixColumn Unit .

Path-1 is used to load data (4 bytes) during first four clock cycles. After that, path-2 is used to reduce these bytes four times again.

### 3.2.1 Data Path

Since our foremost target was to utilize as small area as possible, we decided to make our input bus only 8-bit wide. Although it requires sixteen clock cycles each for the Cipher key and the State input to load, it saves a large number of device resources as far as input/output port utilization is concerned.

### 3.2.2 Internal Architecture of Modules

Internal architectures of SubByte and MixColumn are almost similar to those suggested by Zhang [1]. However, our organization makes it possible to avoid a unit required to perform ShiftRows since a feed back path is already supported which allows us to rotate any row as many times as required.

Previous works [2],[4],[5] suggests a feedback shift register at the top of MixColumn unit. During first four cycles, data are chosen from path-1 until the shift registers are occupied. Later during each clock cycle, shift register is right shifted by one byte and the mux chooses data from path-2 as shown in Fig. 5. Fig.6 shows the modified ShiftRows unit.

The Shift Row logic helps in reducing number of slices used by Mix Column Unit. During the first four cycles, bytes are entered based on similar principle as before. However, using this design it is possible to keep using Shift Row logic in order to rotate the bytes as many times as required.

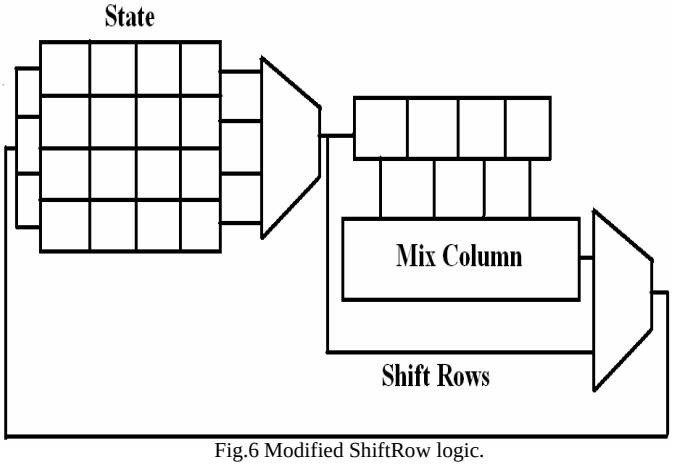


Fig.6 Modified ShiftRow logic.

We suggest a modification in Zhangs [1] architectures by removing this feed back logic. Since our state memory already supports feed back path, we keep on rotating each column until operation completes by making use of ShiftRow logic.

This way we reduce MixColumn architecture by not less than six slices resulting in a much reduced architecture utilizing thirty three slices only. Device utilization summary for this design is given in Table 1.

| Module | No. of Slices Utilized |
|---|---|
| **Sub Bytes** | 52 |
| **Mix Columns** | 33 |
| **Key Schedule** | 36 |
| **Control Unit plus Design** | 80 |
| **Memory Units** | 48x2 = 96 |

Table 1. Device Utilization Summary

## 4 Comparison With Other FPGA Designs

This section provides a brief comparison of our two proposed architectures with other implementations existing in literature. The criteria for comparing various high speed and low area designs depends on the design parameters formulated in Table 2 and Table 3.

Graphical comparisons in terms of Throughput Vs Area and Design frequency Vs Throughput/slice are depicted in Fig 7 and Fig 8 respectively. When comparing different design parameters referenced here for various implementations the following attributes are considered [20],[21]:

- Targeted Device (Which FPGA part is used).
- Key expansion process included or not.
- Only Encryption or Encryption/Decryption datapath is included.

Table 2. shows that first three high speed designs in [3],[7],[9] respectively, give performance details for encryption architecture only thus justifying high throughput but area covered still remains unjustified.

High speed designs from Tim [2] and Zhang [1] are very impressive in terms of throughput but at the cost of area covered in slices. Our design offers a significant tradeoff between area covered in slices and high throughput. The noticeable balance between high throughput and low area makes our proposed design a good engineering solution.

| Design | FPGA Part | Freq MHz | Throughput Mbps | Area Slices | Mbps/ Slice | Data Path |
|---|---|---|---|---|---|---|
| Jarvinen et al [7] | Virtex-E XCV1000e-8 | 129.2 | 16,500 | 11,719 | 1.408 | Enc |
| Standaert et al [8] | Virtex-E XCV3200e-8 | 145 | 18,560 | 15,112 | 1.228 | Enc |
| Hodjat et al [4] Excl.key expand | Virtex-II Pro XC2VP20 | 169.1 | 21,640 | 9,446 Excl.KE | 2.290 | Enc |
| Zhang(r=7) [2] Excl.key expand | Virtex-E XCV1000E-8 | 168.4 | 21,556 | 11,022 Excl.KE | 1,956 | Enc/ Dec |
| Tim 3 LUT cut [2] Key change support1 | Virtex-E XCV 2000E-8 | 184.8 | 23,654 | 16,693 | 1.417 | Enc/ Dec |
| Tim 3 LUT cut [2] Key change support2 | Spartan-III XC3S2000-5 | 196.1 | 25,107 | 17,425 | 1.441 | Enc/ Dec |
| This Work | Spartan-III XC3S2000-5 | 190 | 6080 | 5800 | 1.048 | Enc/ Dec |

Table 2. Comparison of proposed and previous high speed design.

Fig.7 provides a comparison of various high-speed architectures in terms of throughput achieved and the area in slices covered. It can be observed that our design manages considerable amount of throughput in Mbps while maintaining a minimal slice count.

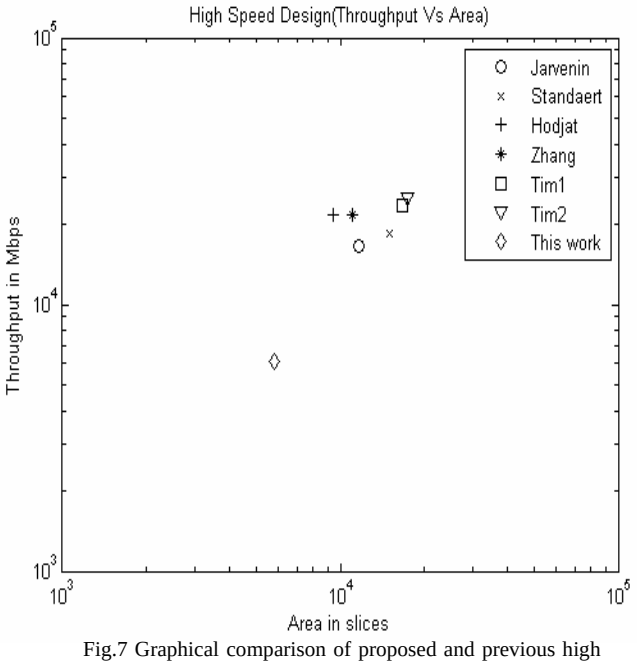


Fig.7 Graphical comparison of proposed and previous high speed designs (Throughput Vs Area)

Fig.8 shows that despite of low Mbps/slice factor our design still manages to have almost the highest frequency in all designs. This result becomes even more acceptable when considering that our design includes both encryption and decryption.

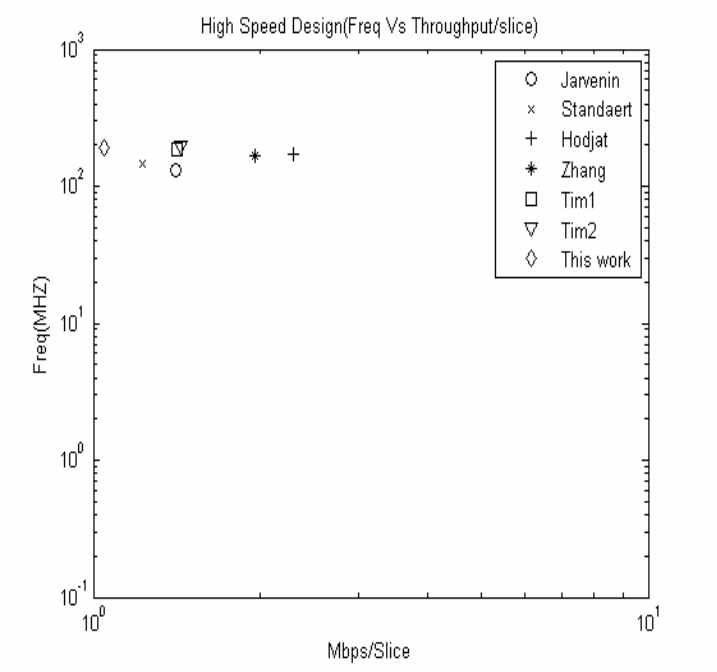


Fig.8 Graphical comparison of proposed and previous high speed designs (Freq(MHZ) Vs Throughput/Slice)

Table 3 provides a comparison of proposed low area design with various existing architectures. The results show that the proposed design is quite efficient in terms of efficiency in comparison to previous designs.

| | This Design | T.Good & Benaissa | Picoblaze based | Chodowiec & Gaj | Rouvroy et al |
|---|---|---|---|---|---|
| FPGA | Spartan-II XC2S15-6 | Spartan-II XC2S15-6 | Spartan-II XC2S15-6 | Spartan-II XC2S30-6 | Spartan-III XC3S50-4 |
| Clock Freq (MHz) | 56 | 67 | 90 | 60 | 71 |
| Datapath | 8 | 8 | 8 | 32 | 32 |
| Slices Utilized | 201 | 124 | 119 | 222 | 163 |
| Block Rams Used | 2 | 2 | 2 | 3 | 3 |
| Block Ram Size (Kbits) | 4 | 4 | 4 | 4 | 18 |
| Bits of Block Rams Used | 5120 | 4480 | 10666 | 9600 | 34176 |
| Slices for Memory Utilized | 96 | 140 | 333 | 300 | 1068 |
| Total Slices | 297 | 264 | 452 | 522 | 1231 |
| Max Throughput (Mbps) | - | - | - | 166 | 208 |
| Av. Throughput (Mbps) | 1.98 | 2.2 | 0.71 | 69 | 87 |
| Throughput/Slice (Kbps/Slice) | 6.67 | 8.3 | 1.9 | 132 | 70 |
| Summary | Second Smallest | Smallest | Software | Best Speed/Area | Fastest |

Table 3. Comparison of proposed design with previous low area deigns.

Although, the throughput of the proposed design is less when compared with Chodowiec & Gaj [4] and Rouvroy [5] designs, but the proposed design is placed among the two smallest designs.

Fig.9 gives a comparison for various low area designs in terms of throughput achieved and the area covered in slices. It is clearly evident from the figure that our design achieves a respectable throughput to other designs [2],[4],[5].

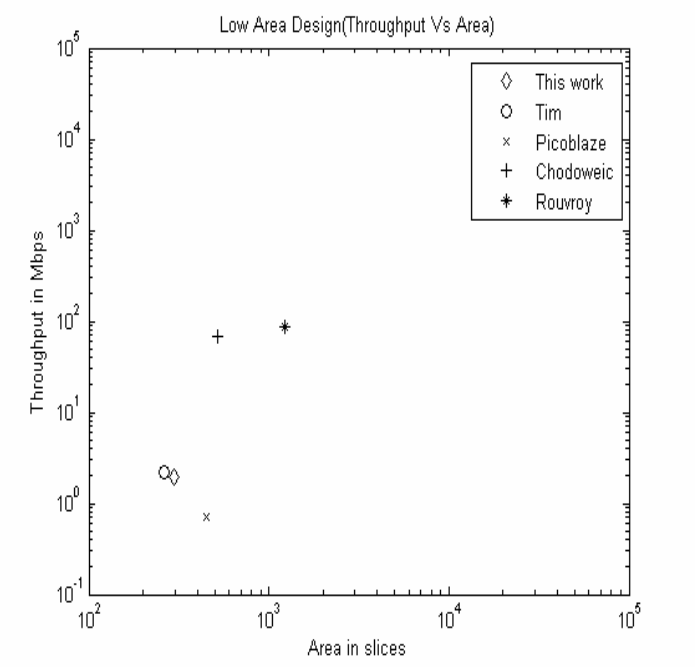


Fig.9 Graphical comparison of proposed and previous low area designs (Throughput Vs Area)

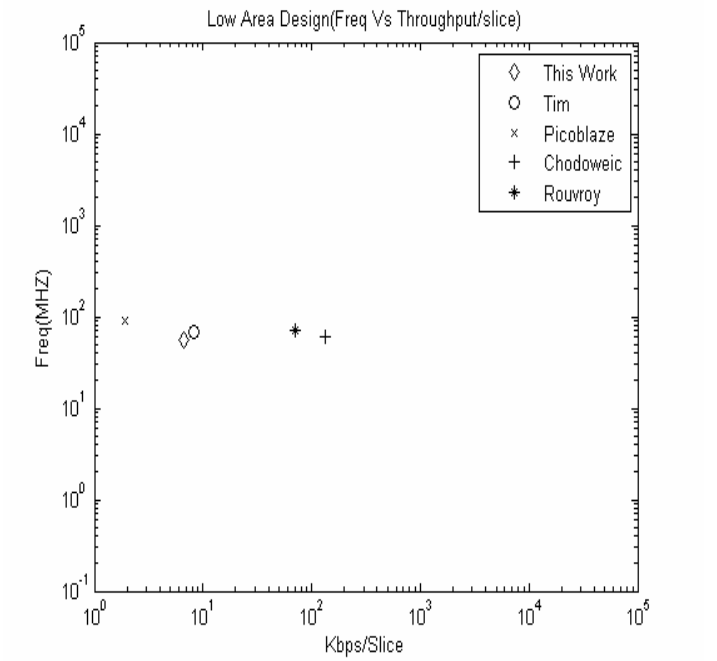


Fig.10 Graphical comparison proposed and previous low area designs (Freq(MHZ) Vs Throughput/Slice)

Fig.10 gives a comparison of clock rate of the design and throughput/slice for various low area architectures. The results show that the proposed design achieves considerable throughput/slice factor with a restricted 8-bit datatpath.

# 5 Conclusions

The work presented provides a comprehensive study and comparison of various high speed and low area designs reported in literature with the design proposed in this paper.

Comparative study and analysis show that high speed design reported in literature need million of logic gates to synthesize them, whereas it is evident from the performance parameters that our high speed design provides a moderate throughput with extraordinary low area factor despite the fact that it includes the whole encryption/decryption designs. Furthermore our design also manages key expansion procedure in the quoted number of slices.

All high speed designs are targeted to high-speed applications providing data rates more than 4000 Mbps e.g. High-speed networks. This work suggests a high speed and area constrained, overall good engineering design particularly targeting fast applications where area is also an auxiliary constraint in addition to high speed. Some of such common applications include broadband switches, firewalls and remote access concentrators.

Proposed low area design is particularly dedicated for applications where area is of vital importance. Furthermore the design achieves a respectable throughput considering the datapath is only 8-bits wide. The low area design being very small is particularly concentrated to compact applications where throughput doesn't mean a great deal in comparison to area constraints. Some common targeted applications where such a design can be deployed are wireless applications such as PDAs and cell phones where area and power constraints are crucial.

*Acknowledgments*

This work was supported by the University of Sheffield, UK. The authors would like to thank Dr. Luke seed and Dr. Mohammad Bennaisa for their useful suggestions and discussion.

*Appendix*

Due to wide range of applications Rijndael algorithm supports various implementations favoring diverse scenarios. Currently AES is targeted to four different solutions depending upon applications.

**Very Fast**

These solutions aim applications with data rates even faster than 2000Mbits/sec.

**Fast**

These solutions target applications like VPN security products deployed in routers, firewalls and switches. They are best suited for applications supporting data rates up to 2000Mbits/sec. Fig. 11 exhibits a practical example. Fast AES solution is incorporated in a secure wireless communication system.

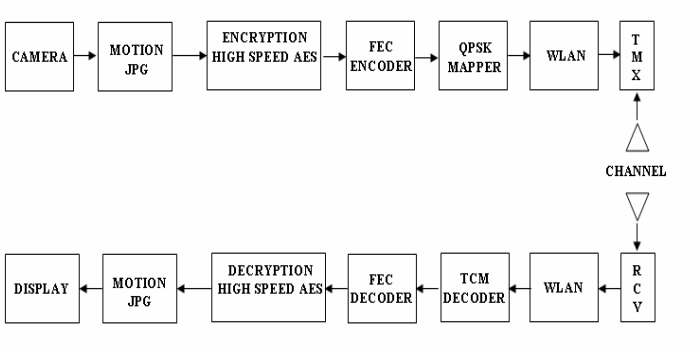


Fig. 11 Block Diagram of a secure wireless communication system.

**Standard**

It can provide rates up to 500 Mbits/sec and is targeted for applications like VOIP.

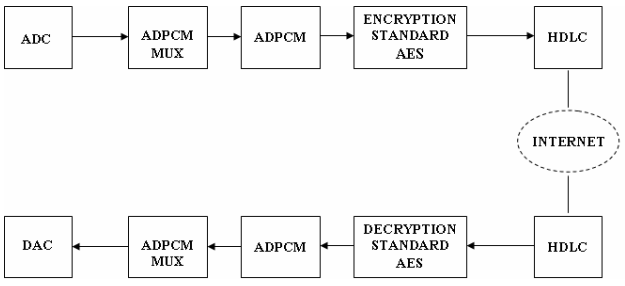


Fig. 12 Block Diagram of a secure VOIP system using AES.

**Compact**

The compact or low area applications are perfectly suited to wireless products like PDAs and cell phones, where power requirements are the greatest concern.

*References:*


[1] X. Zhang, K. K. Parhi, *High-speed VLSI architectures for the AES algorithm*, IEEE Trans. VLSI Systems, Vol. 12, Iss. 9, pp. 957 - 967, Sept. 2004.

[2] Tim good, Mohammad Benaissa, " AES on FPGA from fastest to the smallest", Department of Electronic and Electrical Engineering, University of Sheffield Mappin Street Sheffield S1 3JD, UK.

[3] A. Hodjat, I. Verbauwhede, A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA, 12th Annual IEEE Sypmosium on Field-Programmable Custom Computing Machines (FCCM'04), pp. 308-309, April 2004.

[4] P. Chodowiec, K. Gaj, Very Compact FPGA Implementation of the AES Algorithm, Cryptographic Hardware and Embedded Systems (CHES 2003), LNCS Vol. 2779, pp. 319 – 333, Springer-Verlag, October 2003.

[5] G. Rouvroy, F. X. Standaert, J. J. Quisquater, J. D. Legat, Compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications, Procedings of the international conference on Information Technology: Coding and Computing 2004 (ITCC 2004), pp. 583 – 587, Vol. 2, April 2004.

[6] Advanced Encryption Standard (AES), Nov. 26, 2001.

[7] K. U. Jarvinen, M. T. Tommiska, and J. O. Skytta, A fully pipelined memoryless 17.8 Gbps AES-128 encryptor, Proc. Int. Symp. Field-Programmable Gate Arrays (FPGA 2003), Monterey, CA, pp. 207–215, Feb. 2003.

[8] J. Zambreno, D. Nguyen, A. Choudhary, Exploring Area/Delay Trade-offs in an AES FPGA Implementation, Proc. FPL 2004, 2004

[9] F. Standaert, G. Rouvroy, J. Quisquater, and J. Legat, Efficient implementation of rijndael encryption in reconfigurable hardware: Improvements & design tradeoffs, Proc. CHES 2003, Cologne, Germany, Sept. 2003.

[10] A. Satoh, S. Morioka, K. Takano, S. Munetoh, A Compact Rijndael Hardware Architecture with S-Box Optimization, Proceedings of ASIACRYPT 2001, LNCS Vol. 2248, pp. 239 - 254, Springer-Verlag, December 2001.

[11] G. P. Saggese, A. Mazzeo, N. Mazocca, and A. G. M. Strollo, An FPGA based performance analysis of the unrolling, tiling and pipelining of the AES algorithm, Proc. FPL 2003, Portugal, Sept. 2003.

[12] M. McLoone and J.V. McCanny, High Performance Single-Chip FPGA Rijndael Algorithm Implementations, CHES 2001, Paris, France, 2001.

[13] N. Pramstaller and J. Wolkerstorfer, A Universal and efficient AES co-processor for Field Programmable Logic Arrays, FPL 2004, LNCS Vol. 3203, pp. 565-574, Springer-Verlag, 2004.

[14] A. J. Elbirt, W. Yip, B. Chetwynd, and C. Paar. An FPGA implementation and performance evaluation of the AES block cipher candidate algorithm finalist. Presented at Proc. 3rd AES Conf. (AES3). [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html.

[15] V. Fischer and M. Drutarovsky, "Two methods of Rijndael implementation in reconfigurable hardware," in Proc. CHES 2001, Paris, France, May 2001, pp. 77–92.

[16] K. Gaj and P. Chodowiec. Comparison of the hardware performance of the AES candidates using reconfigurable hardware. presented at Proc. 3rd AES Conf. (AES3). [Online]. Available: http://csrc.nist.gov/encryption/aes/round2/conf3/aes3papers.html.

[17] H. Kuo and I. Verbauwhede, "Architectural optimization for a 1.82 Gbits/sec VLSI implementation of the AES Rijndael algorithm," in Proc. CHES 2001, Paris, France, May 2001, pp. 51–64.

[18] M. McLoone and J. V. McCanny, "Rijndael FPGA implementation utilizing look-up tables," in IEEEWorkshop on Signal Processing Systems, Sept. 2001, pp. 349–360.

[19] A. Rudra, P. K. Dubey, C. S. Jutla, V. Kumar, J. R. Rao, and P. Rohatgi, "Efficient implementation of Rijndael encryption with composite field arithmetic," in Proc. CHES 2001, Paris, France, May 2001, pp. 171–184.

[20] M. H. Jing, Y. H. Chen, Y. T. Chang, and C. H. Hsu, "The design of a fast inverse module in AES," in Proc. Int. Conf. Info-Tech and Info-Net, vol. 3, Beijing, China, Nov. 2001, pp. 298–303.

[21] X. Zhang and K. K. Parhi, "Implementation approaches for the advanced encryption standard algorithm," IEEE Circuits Syst. Mag., vol. 2, no. 4, pp. 24–46, 2002.