A High-Throughput ASIC implementation of Configurable Advanced Encryption Standard (AES) Processor

P.Saravanan
Member IEEE
Assistant Professor
Department of ECE
PSG CT, Coimbatore, India

N.RenukaDevi, G.Swathi
PG Student
Department of ECE
PSG CT, Coimbatore, India

Dr.P.Kalpana
Professor
Department of ECE
PSG CT, Coimbatore, India

ABSTRACT
This paper proposes the Application Specific Integrated Circuit (ASIC) implementation of Advanced Encryption Standard (AES) cryptographic algorithm with reconfigurable 128-bit, 192-bit, 256-bit keys. The proposed implementation has compact 32-bit I/O for both data and key transfer. By using on the fly key generation for encryption process along with efficient implementation of MixColumn and InverseMixColumn operations using finite field GF(2^2) for our 32-bit AES crypto system gives a maximum of 80.1% improvement in operating frequency when compared to the recent implementations. The maximum operating frequency of our proposed pipelined implementation is 333 MHz with high throughput of around 10.656 Gbps in 180 nm standard cell CMOS technology.

Keywords
Keywords - AES, ASIC, Cryptography, Galois Field GF(2^8), On the fly key generation, Throughput.

1. INTRODUCTION
Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables users to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. In order to ensure security in modern wireless communication systems, many cryptographic algorithms have been proposed [1]. In 2001, Rijndael algorithm was selected as the Advanced Encryption Standard (AES) by National Institute of Standards and Technology (NIST) due to the combination of security, performance, efficiency, ease of implementation and flexibility. These features make AES the first choice in many applications such as Wireless LAN and smart cards [2].

AES can be implemented in both software as well as hardware. The advantage of a software implementation includes ease of use, ease of upgrade, portability and flexibility. But the main drawback of software implementation is the limited physical security with respect to key storage [3]. Conversely, hardware implementations are more physically secure, as they cannot be easily read or modified by an outside attacker. Many hardware implementations have been proposed for AES in the literature [4-5]. All these implementations were based on either field programmable gate arrays (FPGA) or application specific integrated circuits (ASIC).

In [6], on the fly S-Box values calculation for both encryption and decryption process has been implemented, which results in reduced operating frequency due to the increase in critical path delay. An AES crypto system implementation using high level code has been discussed in [7]. The maximum throughput achieved in this design was 2.29 Gbps with 173 K gates in 180nm technology and the maximum operating frequency obtained was around 154 MHz. A compact 128 MHz AES implementation has been proposed in [8] which gave a throughput of around 0.14 Gbps with 5.6 K gates in 180 nm technology. In [9], two architectures for the AES algorithm have been proposed through which a maximum throughput of 3.65 Gbps has been achieved. High speed architecture for the hardware implementation of AES algorithm using combinational logic S-Box implementation has been presented [10].

In our proposed AES implementation, on the fly key generation has been used for encryption process. An efficient implementation has been achieved using finite field GF(2^2) in MixColum/InverseMixColumn operations. Shift row and MixColumn operations are combined together in order to reduce the number of registers used in both encryption and decryption process. Pipelining has also been implemented in appropriate operations to deliver high throughput.

The paper is organized as follows. Section 2 explains the basic operations in AES algorithm. The complete overview of the proposed ASIC implementation of AES algorithm is given in Section 3 followed by the comparison of experimental results in Section 4. Section 5 concludes the paper along with noted references.

2. AES ALGORITHM
The AES algorithm is a symmetric block cipher that processes data blocks of 128 bits using a cipher key of length 128, 192, or 256 bits. Each data block consists of a 4×4 array of bytes called the state, on which the basic operations of the AES algorithm are performed. The AES encryption and decryption procedures are shown in Figure 1 and Figure 2. After an initial round key addition, a round function consisting of four different transformations SubByte(), ShiftRow(), MixColumn(), and

AddRoundKey() is applied to the data block (i.e., the state array). The round function is performed iteratively 10, 12, or 14 times, depending on the key length. Note that in the last round MixColumn() is not applied.

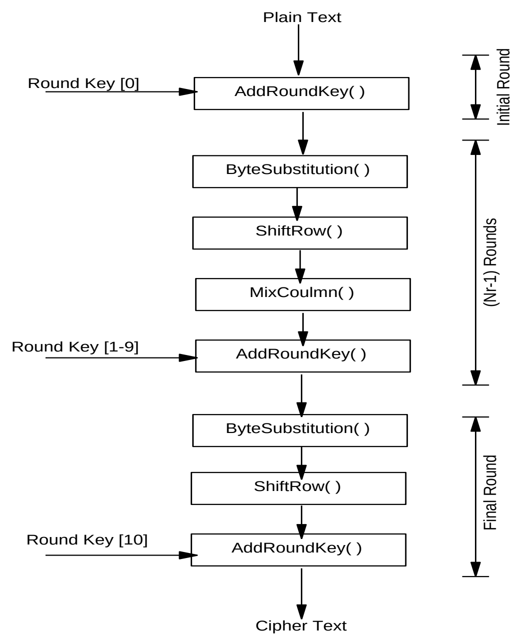

Fig 1: The Encryption procedure of AES Algorithm

The four transformations are described briefly as follows [1]:
• SubByte(): a nonlinear byte substitution that operates independently on each byte of the state using a substitution table (the S-Box).
• ShiftRow(): a circular shifting operation on the rows of the state with different numbers of bytes (offsets).
• MixColumn(): the operation that mixes the bytes in each column by the multiplication of the state with a fixed polynomial modulo $x^4 + 1$.
• AddRoundKey(): an XOR operation that adds a round key to the state in each iteration, where the round keys are generated during the key expansion phase.

The decryption procedure of the AES is basically the inverse of each transformation (InvShiftRow(), InvSub-Byte(), InvMixColumn(), and AddRoundKey()) in reverse order. The decryption procedure thus can be rearranged as shown in Figure 2. Such a structural similarity in both the encryption and decryption procedures makes hardware implementation easier.

3. PROPOSED ASIC IMPLEMENTATION

This section describes the ASIC implementation of AES algorithm for 128-bit key length. In our area efficient implementation, pipelining concepts are included to get maximum throughput along with high speed of operation. Each round of AES is composed of 16 bytes S-Box and four 32-bit MixColumn operations, working on independent data.

3.1 MixColumn() Transformation

The MixColumn transformation operates on the state column-by-column, treating each column as a four-term polynomial [1]. The columns are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)= \{03\}\ x^3 + \{01\}\ x^2 + \{01\}\ x + \{02\}$ is given by the following matrix.

$$\begin{bmatrix} S1' \\ S2' \\ S3' \\ S4' \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S1 \\ S2 \\ S3 \\ S4 \end{bmatrix}$$

The steps to perform the above multiplication are as follows[1]:

$$S1' = 02*S1 + 03*S2 + 01*S3 + 01*S4$$

1. Multiplication by 1 in $GF(2^8)$: Multiplication by one is the identity.
2. Multiplication by 2 in $GF(2^8)$ is performed by :
   Multiplying by a value less than 0x80 → shift all the bits left by 1.Multiplying by a value greater than or equal to 0x80 → shift left by 1 and XOR with 0x1b.
3. Multiplication by 3 in $GF(2^8)$:
   a*0x03 = a*(0x02 + 0x01) = (a * 0x02) + (a * 0x01)

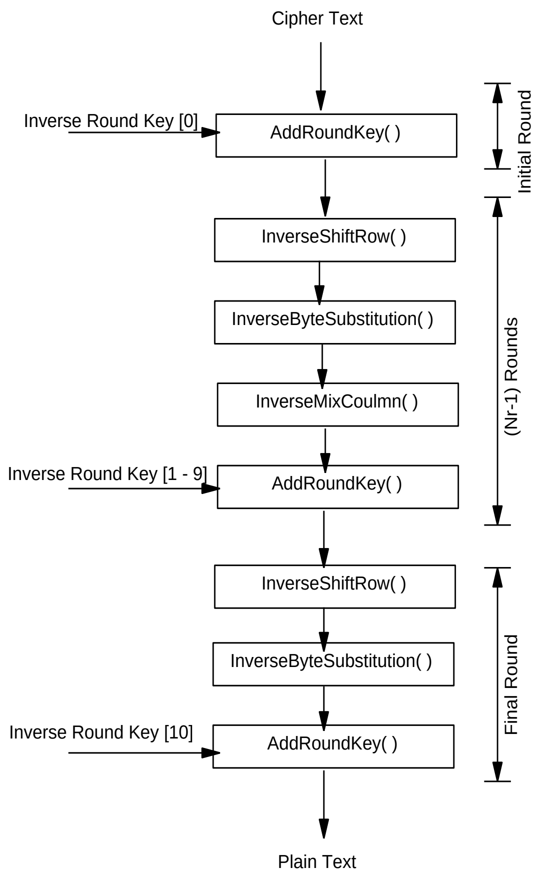

Fig 2: The Decryption procedure of AES Algorithm

3.2 InverseMixColumn () Transformation

Generally affine transformation along with multiplicative inverse in GF(2^8) is used for the implementation of InverseMixColumn operation in decryption process. But the drawback of this technique is the increased complexity due to increase in number of complex operations. Hence, in our proposed implementation, InverseMixColumn operation is performed using multiplication in GF(2^8) method. Here GF(2^2) has been implemented and the same has been used as the basic block for deriving GF(2^8). The matrix multiplication in GF(2^8) for InverseMixColumn is given by the following matrix:

$$\begin{bmatrix} S1' \\ S2' \\ S3' \\ S4' \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S1 \\ S2 \\ S3 \\ S4 \end{bmatrix}$$

where S1'= 0e*S1+ 0b*S2+0d*S3+09*S4.

Multiplication with 0e is obtained by XORing the results of multiplication with 08, 04 and 02 in GF(2^8). The multiplication with 08 is obtained by multiplying 04 and 02 in GF(2^8). The multiplication with 04 is obtained by multiplying 02 and 02 in GF(2^8). The multiplication with 09 is obtained by XORing the results of multiplication with 08 and 01. The multiplication with 0b is obtained by XORing the results of multiplication with 08, 02 and 01. Similarly, the multiplication of 0d is obtained by XORing the multiplication of 08, 04 and 01. The above mentioned operation is explained for a single byte in Figure 3. The same method can be followed for the remaining 3 bytes.

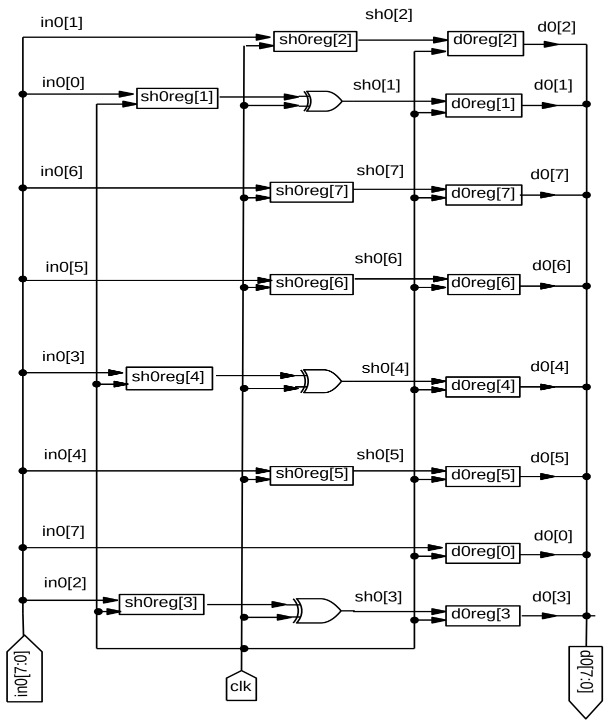

Fig 3: Architecture of single byte multiplication with 02 in GF(2^8).

3.3 Substitution Box (S-Box) implementation

Two approaches are available in the literature for realizing the S-Box. One method uses ROM to store S-Box values and other method calculates the values on the fly. To generate an S-Box value on the fly, two transformations are needed: a multiplicative inverse in GF(2^8) with polynomial, $m(x) = x^8 + x^4 + x^3 + x + 1$ and an affine transformation, $b(x) = (x^7 + x^6 + x^2 + x) + a(x) * (x^7 + x^6 + x^5 + x^4 + 1) \bmod (x^8 + 1)$ where a(x) is the multiplicative inverse in the polynomial form. This approach increases the critical delay of encryption [11]. In order to decrease the critical path delay of the encryption process we used Lookup Table (LUT) implementation.

3.4 Key Generation

Traditionally, most of the papers referred to implementing the key generation by pre-computation method. But the drawback of pre-computation method is the extra memory required to store keys for all rounds which results in increased area. To overcome this problem, in our proposed implementation, on the fly key generation technique has been adopted with pipelined structure. The internal structure of the key generator is shown in Figure 4. Due to the absence of internal storage of keys, the proposed implementation occupies less area which leads to low power consumption when compared to pre-computation method. The Gate count of the design is reduced by 96.2% and power is reduced by 5% with a trade off in latency which increased by 12.6% as shown in Table 1. Since our proposed implementation operates in non-feedback cipher mode, pre-computed key generation technique is used in decryption process.

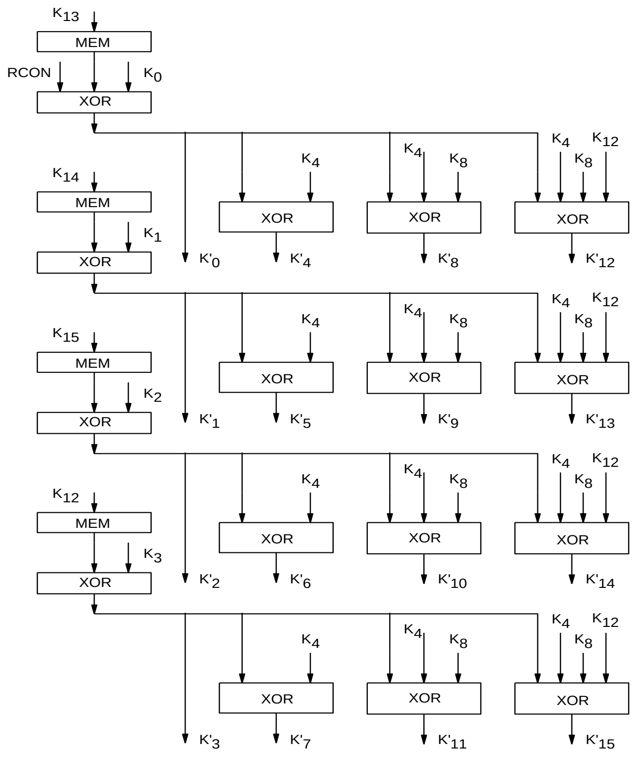

Fig 4: Key generation process

Table 1. Comparison of Key Generation methods in Encryption

Key Generation Method	Area (mm^2)	Latency (Clock Cycles)	Power (mW/MHz)
Pre – computed Key Generation	3.78	71	4.1
On – the fly Key Generation	3.25	80	3.9

3.5 Shift Row / Inverse Shift Row

Instead of performing the shift row operation separately, we combined it with the Substitution Byte operation. Because of this, the internal registers required for shift row operation are avoided in our proposed implementation which results in appreciable area reduction.

In our proposed implementation, 45 pins have been used out of which 32 pins are bidirectional pins (I/O) used for transferring the data and key into the crypto system and get the output (cipher text/plain text) from the crypto system as shown in Figure 5. The data transfer direction in bidirectional pins can be controlled by the pin 'in/out'. When the 'in/out' pin is '1', then data in the bi-directional pins will be loaded into the crypto system and when it is '0', then data in the bi-directional pins can be taken out of the crypto system. Encryption or decryption process can be selected by using the operating mode pin 'func', whose value can be either '0' or '1' respectively. The 'data/key' pin is used to differentiate the data and key inputs given to the crypto system. When the 'data/key' pin is '1', data will be available in the bi-directional pins (Data [31:0]) and when it is '0', key will be available in the bi-directional pins (Data [31:0]) of the crypto system.

The control [3:0] inputs are used to stack the data/key (32-bits) available in the bi-directional pins and store or retrieve from the internal registers with 128 bits and 256 bits for data and key respectively. Load signal is used to make the crypto system wait until all data/key are loaded into the system completely. The security [1:0] is used to select the security level of the Crypto processor. By giving '00', '01', '10' the processor will select 128-bit, 192-bit, 256-bit key length respectively.

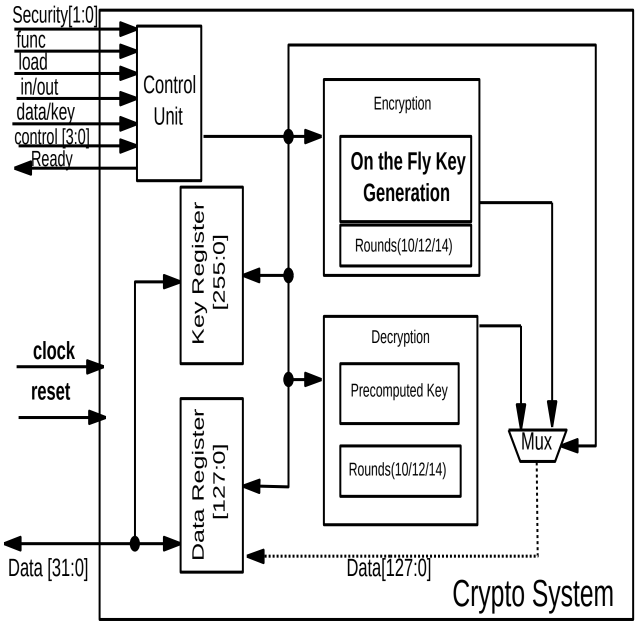

Fig 5: Proposed Configurable AES Architecture

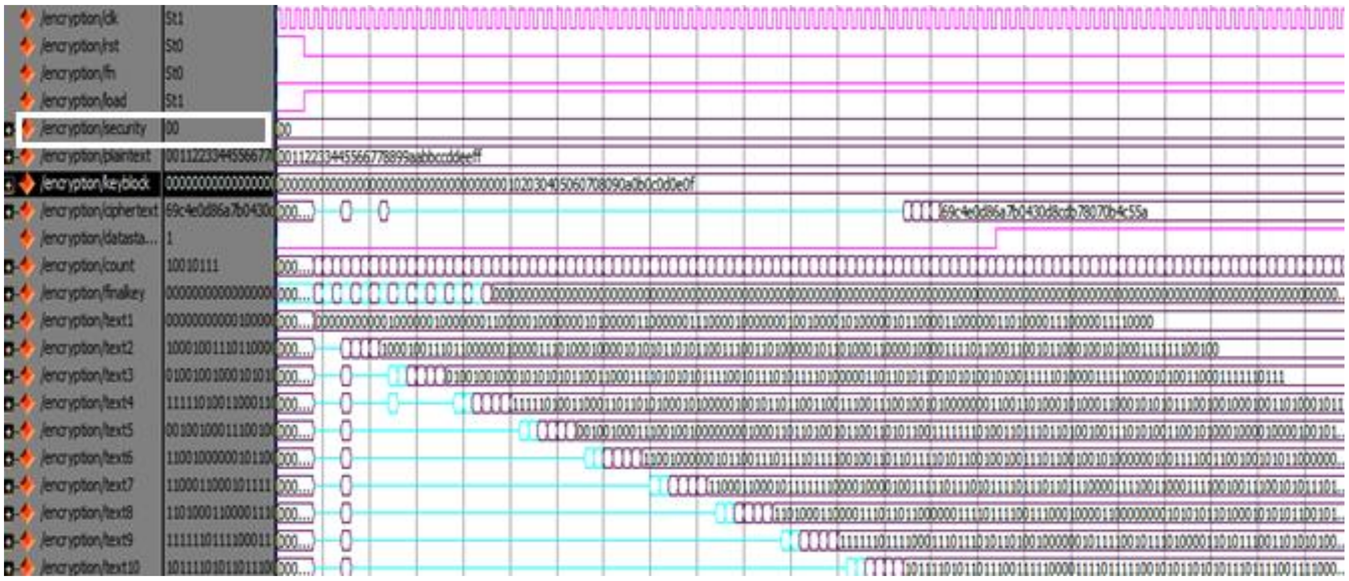

Fig. 6: Simulated Waveform for Encryption with 128 bit key

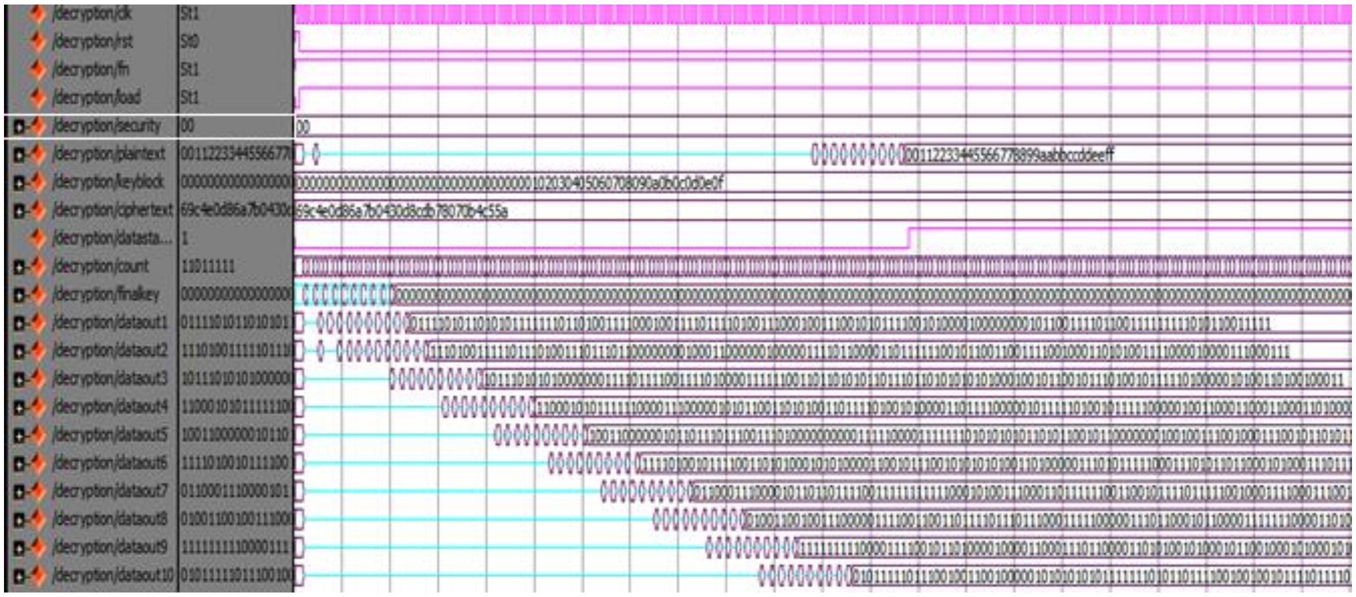

Fig. 7: Simulated Waveform for Decryption with 128 bit key

Table 2. Comparison of AES Design

Parameters	**Verbauwhede[7]**	**Mangard[12]**	**Satoh[13]**	**Su[14]**	**Yen[15]**	**Mao –Yin Wang[16]**	**Ours**
Configurable	No	No	No	No	Yes	Yes	Yes
Technology (nm)	180	600	110	350	180	250	180
Clock Rate (MHz)	125	64	224.22	200	153.84	66	333
Throughput (Gbps)	1.6 1.33 1.14	0.241	2.21	2.381 2.008 1.736	1.7902	0.844 0.704 0.603	10.656

4. SIMULATION RESULTS

The proposed AES architecture is described in Verilog HDL at the register-transfer level. Synthesizing the RTL into the gate level was done by design compiler using 180 nm, standard-cell CMOS technology. Back-end design has been carried out using SOC-encounter. The simulated waveforms for both encryption and decryption process with 128-bit key are shown in Figure 6 and Figure 7 respectively. The comparison results of the proposed implementation with the existing implementations are presented in Table 2 which shows that our proposed implementation is better in all cases when compared to the existing implementations. A maximum operating frequency of 333 MHz and throughput of 10.656 Gbps has been achieved with our proposed implementation. The final layout of the proposed configurable AES processor is shown in Figure 8.

5. CONCLUSION

In this paper, a compact and fully pipelined ASIC implementation of AES cryptography algorithm has been presented. The proposed implementation is configurable to take 128, 192 and 256-bit keys according to the requirement of the security level. The proposed architecture is synthesized in 180 nm standard cell CMOS technology and simulated at gate level to measure the speed of operation. The proposed implementation with 32-bit I/O gives a maximum of 10.656 Gbps throughput with the maximum operating frequency of 333 MHz which outperforms the previously reported schemes.

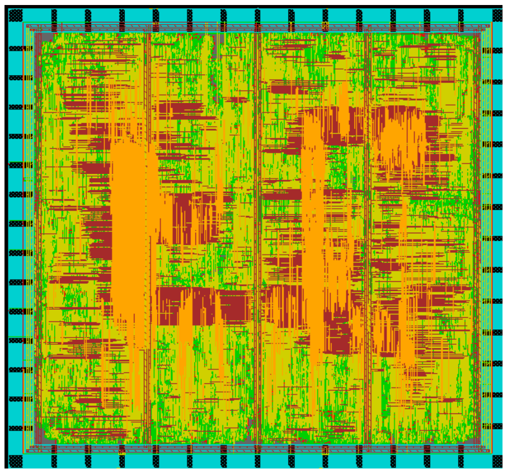

Fig. 8: Final Layout of proposed Configurable AES Processor

6. REFERENCES

[1] William Stallings, *Cryptography and Network Security*, Pearson Education, 2009.

[2] National Institute of Standards and Technology (US), Advanced Encryption Standard, http://csrc.nist.gov/publication/drafts/dfips-AES.pdf.

[3] B. Schnieir, *Applied Cryptography*, Wiley, New York, 1996.

[4] P. Chodowiec and K. Gaj, "Very compact FPGA implementation of the AES algorithm," Proceedings of Cryptographic Hardware and Embedded Systems (CHES), pp. 319-333, 2003.

[5] Gael Rouvroy, Francois-Xavier Standaert and Jean-Jacques Quisquater, "Compact and efficient encryption / decryption module for FPGA implementation of the AES Rijndael very well suited for small embedded applications," Proceedings of the International Conference on Information Technology : Coding and Computing (ITCC'04), 2004.

[6] Chih-Pin Su, Tsung-Fu Lin, Chih-Tsiun Huang and Cheng-Wen Wu, "A high-throughput low cost aes processor," IEEE Communications Magazine, vol. 41, no. 12, pp. 86-91, 2003.

[7] Ingrid Verbauwhede, Patrick Schaumont, and Henry Kuo, "Design and performance testing of a 2.29 Gb/s Rijndael processor," IEEE Journal of solid-state circuits, vol. 38, no. 3, 2003.

[8] F. Haghighizadeh, H. Attarzadeh and M. Sharifkhani, "A Compact 8-bit AES Crypto-Processor," Second International Conference on Computer and Network Technology (ICCNT'10), 2010.

[9] N. Sklavos and O. Koufopavlou, "Architectures and VLSI implementations of the AES-Proposal Rijndael," IEEE Transactions on computers, vol. 51 no. 12, pp. 1454-1459, 2002.

[10] X. Zhang and K. Parhi, "High speed VLSI Architectures for the AES algorithm," IEEE transactions on VLSI systems, vol. 12, no. 9, 2004.

[11] S.M. Yoo, D. Kotturi, D.W. Pan and J. Blizzard, "An AES crypto chip using a high-speed parallel pipelined architecture," Journal of Microprocessors and Microsystems, pp. 317-326, 2005.

[12] S. Mangard, M. Aigner, and S. Dominikus, "A highly regular and scal-able AES hardware architecture," IEEE Trans. Comput. , vol. 52, no. 4,pp. 483–491, Apr. 2003

[13] A. Satoh, S. Morioka, K. Takano, and S. Munetoh, "A compact Rijn-dael hardware architecture with S-box optimization," in ASIA CRYPT 2001 .Berlin, Germany: Springer-Verlag, 2001, vol. 2248, LNCS, pp. 239–254.

[14] C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu, "A high-throughput low-cost AES processor," IEEE Commun. Mag. ,vol. 41, no. 12, pp. 86–91, Dec. 2003.

[15] C.-H. Yen, T.-Y. Pai, and B.-F. Wu, "The implementations of the re-configurable Rijndael algorithm with throughput of 4.9 Gbps," in Proc.16th VLSI Des./CAD Symp. , Hualien, Taiwan, Aug. 2005.

[16] Mao –Yin Wang, Chih –Pin Su, Chia- Lung Horng, Cheng –Wen Wu and Chih- Tsun Huang, " Single- and Multi-core Configurable AEs Architectures for Flexible Security," IEEE Transactions on Very Large Scale Integration(VLSI) Systems, Vol. 18,No.4, April 2010.