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 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 columnby-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.

S1'		02	03	01	01	<i>S</i> 1
S2'		01	02	03	01	<u>S</u> 2
<i>S</i> 3′	-	01	01	02	03	<u>S</u> 3
S4'		03	01	01	02	<i>S</i> 4

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

- 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 $0x80 \rightarrow$ shift all the bits left by 1. Multiplying by a value greater than or equal to $0x80 \rightarrow$ 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:

S1'		0e	0b	0d	09	S1
S2'		09	0e	0 <i>b</i>	0d	<u>S</u> 2
53'	=	0d	09	0e	0 <i>b</i>	53
<u>S4</u> '		lob	0d	09	0e	<u>5</u> 4

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, $b(x) = (x^7 + x^6 + x^2 + x) + a(x) * (x^7 + x^6 + x^5 + x^4 + 1) mod (x^8 + 1)$ 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

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

 Table 1. Comparison of Key Generation methods in Encryption

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 bidirectional 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 bidirectional 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 (32bits) 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

4	/encrypton/dk	81	haann	hooon	henon	hanan	hasen	honon	hone	heene	hooo	booso	honno	haene	honon	hanna	hanna	hanaa	hono	hanar	hanna	hana	hanan	10000	honor
	/encryption/hst	510																		1					
	/encrypton/fn	580								-														1.	
	lencryption/load	511	-																						
0 1	/encryption/security	00	00							-	1										1				
0-1	/encrypton/plantext	0011223344556677	001122	33445566	7788998	abbccdó	eeff			-			-				_	-			-				
÷.) (encryption, keyblock	000000000000000000000000000000000000000	000000	00000000	0000000	0000000	0000000	1020304	0506070	8090a0b	0c0d0e0f	F			1						-				
0-1	/encryption/ciphertext	59c4e0d86a7b0430	200)	0	0	-	-	-	-	-	-	-	-	-	-	-000	169c4e	186476	1430d8o	\$73070	4:550			1	
	/encryption/datasta	1															and the latest								
0- 4	/encryption/count	10010111	200]	IIIII		ш	IIII	um	ш	1000	IIIII	IIII	IIIII	ш	ш			IIII	m	1001	IIIII	ma			
0-1	/encrypton/finalkey	00000000000000000	000[100		100	C (00	000000	00000	100000	000000	000000	0000000	0000000	000000	000000	000000	0000000	000000	000000	\$00000	000000	00000000	0000000	00000.
8-1	/encryption/text1	000000000000000000000000000000000000000	300)	0000000	0001000	0001000	00001100	00001000	0000010	1000001	0000000	1100001	0000000	001000	1010000	0101100	0011000	0001101	0000111	0000011	110000				
0-1	/encryption/text2	1000100111011000	200)	a	110001	0011101	1000000	1000011	010001	00001010	0011010	1100111	0011010	0000101	1010001	0000100	0011110	1100011	0010110	0010010	1000111	11110010	0		
0-1	/encryption/text3	0100100100010101		0	-010	11.01	00100100	010101010	101100	1100011	1010101	0111100	1011101	1111101	00000110	1101011	0010101	0010100	1111101	0000111	1100001	01001100	0111111	0111	
0-1	/encryption/text4	1111101001100011	000)	0	-0-	- (au	111110	001100	1110110	0100010	1000001	0010110	100110	01110011	1001001	0100000	0011001	1010001	0100011	0001010	1011100	0010001	0011010	001011
0-1	/encryption/text5	0010010001110010		0				aa	11 2001	0010001	0001001	0000000	0100011	110100	1011001	0101100	1111111	0100110	1110110	1001001	1101010	0110010	0001000	0100001	00101
0-1	/encryption/texts	110010000010110	000)	0		-	-		-α	m	1001000	00001011	0011101	110111	10010011	0110111	1010110	0100100	1110110	0100101	0000001	00111100	11100100	1010110	00000
0-1	/encryption/text7	1100011000101111	200)	0		-	-	-	-	-	a	111000	1100010	111111	20001000	0100111	1101110	0011110	1110110	1110000	1111001	10001111	0010011	1001010	11101.
8-1	/encryption/text8	1103000130000113		0	-	-	-	-		-	-	-00	11111	100011	00001110	1101100	0000111	1011110	0111000	1000011	0000000	0101010	1010001	0101011	00101
0-1	/encryption/text9	1111110111100011	000)	0				-					0	au	111110	1111000	1110111	0101101	0010000	0010111	1001011	1010000	1010111	0011010	10100
0-4	Jencryption/text10	101111010100110	000)	0	-	-		-			-				111	1 10011	1101011	0111001	1111000	0111101	1111001	01011010	10011101	1110011	11000.

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

/decryption/clk	St1																						
/decryption/rst	510			10.00	1	1.200	100 M	1	-	-	2000	1		10.00	-1-50	- 5.	1.000		1. A. A. A.		and the state of	0.2.00	
/decryption/fin	51									-	-						-			_			
/decryption/load	St1	5	1																				
/decryption/security	00	00																				-	-
/decryption/plaintext	0011223344556677	00	-									0	00000	0000	1122334	4556677	8899aab	bccddeef	f				
/decryption/keyblock	00000000000000000	000000	0000000	000000	000000	0000000	1020304	0506070	8090a0b	(c0d0e0										_		-	
/decryption/ophertext	69c4e0d86a7b0430c	69c4e0	d86a7b04	Godised	7807064	55a										_							_
/decryption/datasta	1					-														-			
/decryption/count	11011111	0000	(Include)	0.001000	(10000)	0000000	10.00000	100.0010	(10000)	100000	0.0000	DENNEND	0.00.000	0000000		101001	0.0000		de la compañía de la	albiitiiti	000000	10000	0.000.00
/decryption/finalkey	00000000000000000000000	0000	100000	000000	000000	0000000	0000000	000000	000000	100000	000000	0000000	0000000	000000	10000000	00000	000000	0000000	0000000	000000	0000000	0000000	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
/decryption/dataout1	011110101101010101	00-0	00000	000011	10010110	1010111	1111011	0100111	1000100	1111011	1101001	10000100	1110010	1011110	01010000	100000	0001011	0111101	1001111	1111101	0110011	11	
ideoryption/dataout2	1110100111110111	0-0-	000000	00000	1101001	1111011	1010011	1011101	1000000	0010001	000000	0000011	1101100	9011011	11110010	1100110	0111100	01000110	1010011	1100001	00001110	00111	_
/decryption/dataout3	10111010101000000	0	1	00000	00000	1011101	0101000	0000111	1011110	0111101	00001111	1100110	1101010	1101110	11010101	0101000	0100101	0010111	0100101	11110100	0000 10 10	0110100	100011
/decryption/dataout4	11000101011111100	0-0			0000	00000	110001	101111	1100001	1000000	0101100	1101010	0110111	1010010	0000110	1111000	0010111	1101001	01111100	0000 100 1	1000110	0011000	1101000
/decryption/dataout5	1001100000010110	0-0	-			0000	00000	1100110	1000000	1101110	1110011	10100000	0000001	1110000	1111111	0101010	1011010	1100101	100000000	1001001	1100100	0111001	01101011
/decryption/dataout6	1111010010111100	0-0			-		000	00000	0 011110	0001011	1100110	10 1000 10	1010000	1 100 10 1	1100101	0101010	0110100	0001110	10111110	0011101	0110110	00101000	1110111
/decryption/dataout7	0110001110000101	0-		-				000	00000	000110	0011100	00101101	1011110	0111111	1111100	0101001	1100013	01111110	0110010	1111011	1110010	0111100	0111001
/decryption/dataout8	01001100100111000	0	-	-	-		-		00	00000	000000	01100100	1110000	0111100	11001101	111011	1011100	01111100	0001110	1100010	1100001	1111100	0011010
/decryption/dataout9	1111111110000111	0	-				-			00	00000	000111	1111110	0001111	00101101	0000010	0011000	1110110	0001101	0100101	0010110	0100010	1000101
/decryption/dataout10	0 10 11 11 10 11 100 100	0	-				-				0	00000	00001	0111110	11 100 100	110010	00001010	101010101	1111110	1011011	100 100 1	0010111	1011110
	jderypton/dk jderypton/tst jderypton/tst jderypton/bad jderypton/becurity jderypton/becurity jderypton/gatecurity jderypton/gatecurit jderypton/gatecurit jderypton/gatecurit jderypton/gatecurit jderypton/gatecurit jderypton/gatecurit jderypton/gatecurit jderypton/gatecurits jderypton/gatecurits jderypton/gatecurits jderypton/gatecurits jderypton/gatecurits jderypton/gatecurits jderypton/gatecurits jderypton/gatecurits	jdecrypton/sk St1 jdecrypton/ist St0 jdecrypton/ist St0 jdecrypton/ist St1 jdecrypton/ist St1 jdecrypton/security 00 jdecrypton/securit St11111 jdecrypton/securit 11011111 jdecrypton/securit 11011111 jdecrypton/securit 11101011010101 jdecrypton/securit 111010111010101 jdecrypton/securit 11101011110101 jdecrypton/securit 1110101010100000 jdecrypton/securit 1100011111000 jdecrypton/securit 1001010000000000000000000000000000000	(decrypton/dk St1 (decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St1 (decrypton/ist St1 (decrypton/ist St1 (decrypton/ist St1 (decrypton/ist St1 (decrypton/istacurity 00 St1 (decrypton/istacurity 0011223344556677	(decrypton/dk S1 (decrypton/int S1 (decrypton/intext 00 (decrypton/idecurity 00 (decrypton/idecurity 000000000000000000000000000000000000	(decrypton/dk St1 (decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St1 (decrypton/ist St1 (decrypton/ist St1 (decrypton/istecurity 00 (decrypton/istecurity 00112233445566770 (decrypton/istecurity 00112233445566770 (decrypton/istecurity 00112233445566770 (decrypton/istecurity 00112233445566770 (decrypton/istecurity 00112233445566770 (decrypton/istecurity 000000000000000000000000000000000000	(decrypton)/dk S11 (decrypton)/ist S60 (decrypton)/ist S60 (decrypton)/ist S11 (decrypton)/ist S11 (decrypton)/ist S11 (decrypton)/ist S11 (decrypton)/ist S11 (decrypton)/istantext 00112233445566770 (decrypton)/istantext 000000000000000000000000000000000000	(decrypton/dx Sti (decrypton/ist Sti (decrypton/istacurity 00 (decrypton/istacurity 00 (decrypton/istacurity 00 (decrypton/istacurity 0011223344556677 (decrypton/istacurity 000000000000000000000000000000000000	(decrypton/dk Sti (decrypton/ist Sti (decrypton/ista Sti	(decrypton)ist Stil (decrypton)istantext 00112233445566770 (decrypton)istantext 00112233445566770 (decrypton)istantext 000000000000000000000000000000000000	(decrypton/dx S1 (decrypton/ist S60 (decrypton/ist S60 (decrypton/ist S11 (decrypton/ist S11 (decrypton/ist S11 (decrypton/ist S11 (decrypton/istantext 0011223344556677 (decrypton/istantext 000000000000000000000000000000000000	(decrypton/dx Sti (decrypton/ist Sti (decrypton/ist Sti (decrypton/ist Sti (decrypton/ist Sti (decrypton/ist Sti (decrypton/ist Sti (decrypton/istaniett 001 (decrypton/istaniett 0011223344556677 (decrypton/istaniett 0011223344556677 (decrypton/istaniett 000000000000000000000000000000000000	(decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St0 (decrypton/ist St1 (decrypton/ist St1 (decrypton/ist St1 (decrypton/istantext 0011223344556677 (decrypton/istantext 0011223344556677 (decrypton/istantext 000000000000000000000000000000000000	(decrypton/ist Still (decrypton/istantext 00112233445566770 (decrypton/istantext 00112233445566770 (decrypton/istantext 00112233445566770 (decrypton/istantext 0010000000000000000000000000000000000	(decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/istantext 00112233445566770 (decrypton/istantext 00112233445566770 (decrypton/istantext 00112233445566770 (decrypton/istantext 00112233445566770 (decrypton/istantext 0010000000000000000000000000000000000	(decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/ist Still (decrypton/istantext 0011223344556677 (decrypton/istantext 0011223344556677 (decrypton/istantext 0011223344556677 (decrypton/istantext 0011223344556677 (decrypton/istantext 000000000000000000000000000000000000	(decrypton/ist Still (decrypton/istantext 0011223344556677 (decrypton/istantext 0011223344556677 (decrypton/istantext 0010000000000000000000000000000000000	Adecryption (dx S11 (decryption (ist S60 (decryption (ist S60 (decryption (ist S11 (decryption (ist) 00 (decryption (ist) 0011223344556677 (decryption (ist) 0011223344556677 (decryption (ist) 0011223344556677 (decryption (ist) 000000000000000000000000000000000000	decrypton/st S1 /decrypton/st S0 /decrypton/st S0 /decrypton/st S1 /decrypton/st S1 /decrypton/stasult 0112234455677 /decrypton/stasult 0112234455677 /decrypton/stasult 00112234455677 /decrypton/stasult 00112234455677 /decrypton/stasult 00112234455677 /decrypton/stasult 00112234455677 /decrypton/stasult 00112234455677 /decrypton/stasult 001122344556778899ad /decrypton/stasult 001122344556778899ad /decrypton/stasult 1011111 /decrypton/stasult 1011111 /decrypton/stasult 1011111 /decrypton/stasult 1011101101010100 /decrypton/stasult 01119011010000 /decrypton/stasult 011190110100000 /decrypton/stasult 011190110100000 /decrypton/stasult 000000000000000000000000000000000000	decrypton/sk Si1 decrypton/sk Si0 decrypton/sk Si0 decrypton/sk Si0 decrypton/sk Si1 decrypton/sk Si1	decrypton/sk Stil /decrypton/st Stil /decrypton/st Stil /decrypton/st Stil /decrypton/st Stil /decrypton/st Stil /decrypton/st Stil /decrypton/stecurity O0 /decrypton/stecurity O0 /decrypton/stecurity O0 /decrypton/stecurity O0 /decrypton/stecurity O0 /decrypton/stect O0/00000000000000000000000000000000000	decryption/st S1 /decryption/st S0 /decryption/fin S1 /decryption/fin/fin/fin/fin/fin/fin/fin/fin/fin/fi	Addromytion (dk SR1 Addromytion (dkn SR1	Merryston/dx Stil Merryston/fxt Stil Stil Stil Merryston/fxt Stil Stil Stil Merryston/fxt Stil Merryston/fxt

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 highthroughput 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.