ISSN No.: 2454- 2024 (online)

International Journal of Technical Research & Science

IMPLEMENTATION OF HIGH THROUGHPUT CRYPTO CO-PROCESSOR USING AES

A.Saravanakumar[1], R.Evangeline[2]
Solomon.be@gmail.com[1], king.eva7@gmail.com[2]
Assistant Professor, Dream Institute of Technology, Kolkata, Bengal (India)
Ph.D Research Scholar, Sri SRNM College, Sattur, Tamil Nadu (India)

Abstract-The Crypto Co-processor System Design is increasing the number of AES Encryption only. At a system frequency of 400 MHz, increasing the number of AES Encryption from one to two and then to four Modules. Note that at system frequency of 400 MHz, one, two, four lanes AES Modules are running at 400 MHz, 200 MHz, respectively, and the other parts of the crypto coprocessor are running at 400 MHz. Obviously, the consumed area increases with increasing the number of AES Modules with increasing the SDQ depth for each individual curve. More power consumption perspective, the use of AES Encryption pipelines running at 50 MHz results in lower power consumption than two parallel pipelines running at 100 MHz, or one pipeline running at 200 MHz. For more clarity, when the AES pipelines operating frequency is 50 MHz, the effect of increasing the number of parallel lanes on the overall performance of Crypto is dramatically large Moreover, the Crypto throughput when the input data varies from 8 to 3072 blocks. A huge improvement in throughput with increasing the parallel pipelines at constant AES pipelines frequency, where all AES pipelines run at 50 MHz.
Keywords: AES Pipelines, Cryptography, Parallel processing.

1. INTRODUCTION

In cryptography, encryption is the process of transforming information(plaintext) using an algorithm (cipher) to make it unreadable to anyone except those possessing special knowledge(key). The result of the process is encrypted information (ciphertext).Decryption refers to the reverse process to make the encrypted information readable again (converts the cipher text to plaintext using an algorithm called decipher). Encryption has long been used by militaries and governments to facilitate secret communication. However, it is now commonly used in protecting information within many kinds of civilian systems to protect data in transit or in storage. Thus, cryptographic processing has been brought to the forefront of system design. Nowadays, the most important cryptographic algorithm is the Advance Standard Algorithm (AES).AES algorithm is a computationally intensive application based on data-level parallelism (DLP), where the same set of operations are repeated over streams of input data. Among the various forms of parallelism (instruction-level parallelism (ILP), thread-level parallelism (TLP), and DLP) the cheapest and the most prevalent form of parallelism available in many applications is DLP .Exploiting DLP existed in AES algorithm is the key to achieving high throughput by executing multiple, independent operations concurrently.

2. REVIEW OF RELATED WORK

Since AES was accepted as a FIPS (Federal InformationProcessingStandards), it became one of the most important cryptographicalgorithms till today. It received a lot of researchers' attention.Thus, there are various forms for implementing this cryptographicalgorithm on software and hardware. Even though the AES algorithmcan be programmed on general-purpose processors (GPP),its performance is not acceptable to many cryptographic applications.On the other hand, hardware implementations give higherperformances than software; however, they represent special purposehardware. There have been many different hardware implementationsfor ASIC (Application Specific Integrated Circuits) andFPGA to improve the throughput of the AES algorithm.

On a single-chip FPGA (Xilinx Virtex-E), McLoone and Mc-Canny [17] described implementations of AES algorithm. Theypresented generic, parameterizableRijndaelencryptor core capableof supporting varying key sizes. A pipelined single-chip 128-bit key Rijndaelencryptor/decryptor core achieved a data rate of3.2 Gb/s on a system clock of 25.3 MHz.Jrvinen et al. [11] implementeda pipelined AES-128 encryption algorithm on XilinxVirtex-II devices. They implemented the Sboxes combinational,where no internal memory was required. They achieved a throughputof 17.8 Gb/s at 139.1 MHz.Standaert et al. [27] implementeda pipeline version that unrolls the ten AES rounds on FPGA (XilinxVirtex-E) and achieved up to 18.125 Gb/s at 145 MHz.Hodjat andVerbauwhede [7] presented a fully pipelined AES encryption processoron a single-chip FPGA (VirtexII-Pro). By using loop unrolling.and inner-round and outer-round pipelining techniques, a maximumthroughput of 21.54 Gb/s was achieved at 168.3 MHz and apipeline latency of 31 cycles. Qin et al. [20] used the Altera

StratixFPGA to implement the AES encryption circuit. The unrolling implementationachieved a throughput of 20.48 Gb/s at 160.05 MHz.

Wu et al. [31] have designed CryptoManiac, which is acryptographic coprocessor based on VLIW (Very Long InstructionWord) to execute up to four instruction per cycle. Additionally,short latency instructions (e.g. bitwise logical and arithmeticinstructions) can be combined to be executed in a single cycle.CryptoManiac was designed to run at clock rate 360 MHz achievingthroughput equal to 500 Mb/s. Oliva et al. [19] described aprogrammable processor called CRYPTONITE tailored to the needsof crypto algorithms. The CRYPTONITE crypto-processor is a VLIWarchitecture with two 64-bit datapaths, which was designed to runat clock rate 400 MHz and providing a throughput of 700 Mb/s.

3. AES ENCRYPTION ARCHITECTURE

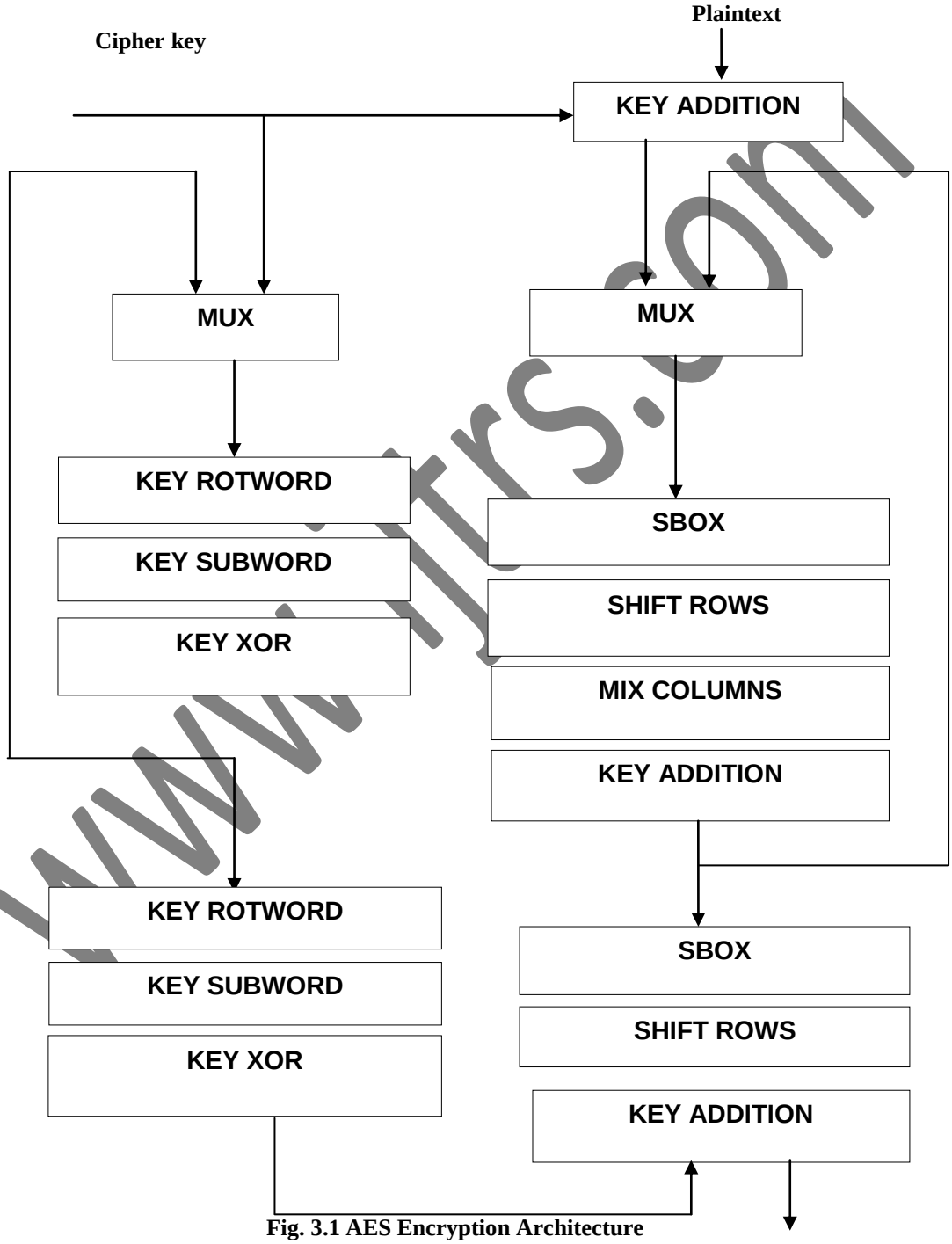


Fig. 3.1 AES Encryption Architecture

AES is a symmetric block cipher. This means that it uses the same keyfor both encryption and decryption. However, AES is quite different from DES in a number of ways. The algorithm Rijndael allows for a variety of block and key sizes and not just the 64 and 56 bits of DES' block and key size. The block and key can in fact be

chosen independently from 128, 160, 192, 224, 256 bits and need not be the same. However, the AES standard states that the algorithm can only accept a block size of 128 bits and a choice of three keys - 128, 192, 256 bits. Depending on which version is used, the name of the standard is modified to AES-128, AES-192 or AES-256 respectively. As well as these differences AES differs from DES in that it is not a feistel structure. Recall that in a feistel structure, half of the data block is used to modify the other half of the data block and then the halves are wapped. In this case the entire data block is processed in parallel during each round using substitutions and permutations. A number of AES parameters depend on the key length. For example, if the key size used is 128 then the number of rounds is 10 whereas it is 12 and 14 for 192 and 256 bits respectively. At present the most common key size likely to be used is the128 bit key.

3.1 Inner Workings of a Round

The algorithm begins with an Add round key stage followed by 9 rounds of four stages and a tenth round of three stages. This applies for both encryption and decryption with the exception that each stage of a round the decryption algorithm is the inverse of it's counterpart in the encryption algorithm.

The four stages are as follows:

- Substitute bytes
- Shift rows
- Mix Columns
- Add Round Key

3.1.1 Substitute Bytes

This stage known as SubBytes is simply a table lookup using a 16×16 matrix of byte values called an s-box. This matrix consists of all the possible combinations of an 8 bit sequence (28 = 16 × 16 = 256). However, the s-box is not just a random permutationof these values and there is a well defined method for creating the s-box tables. The designers of Rijndael showed how this was done unlike the s-boxes in DES for which no rationale was given. We will not be too concerned here how the s-boxes are made up and can simply take them as table lookups.Again the matrix that gets operated upon throughout the encryption is known as state.

It will be concerned with how this matrix is affected in each round.

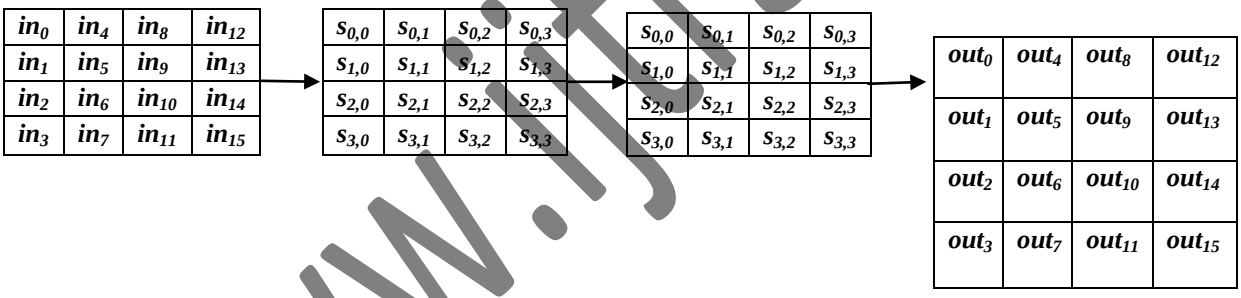
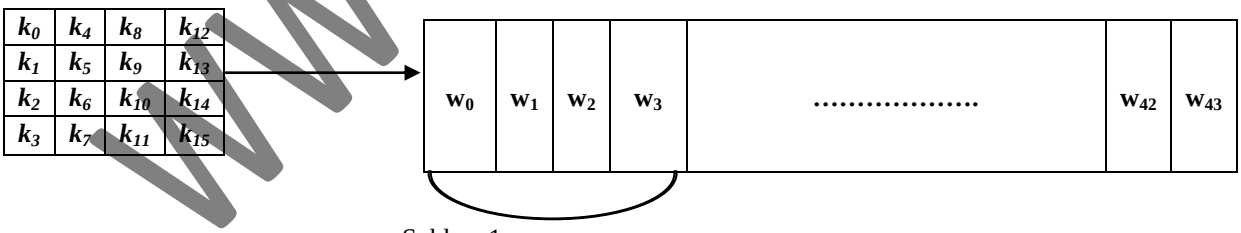


Fig. 3.2(a) Input, State Array and Output



Key and expanded key

Fig. 3.2(b) Data Structures in the AES Algorithm.

Round each byte is mapped into a new byte in the following way: the left most nibble of the byte is used to specify a particular row of the s-box and the rightmost nibble specifies a column. For example, the byte {95} (curly brackets represent hex values in FIPS PUB 197) selects row 9 column 5 which turns out to contain the value {2A}. This is then used to update the state matrix.

The second basic cryptographic primitive is the random generator, also known as a keystream generator or stream cipher. This is also a random function, but unlike in the hash function case it has a short input and a long output. If we had a good pseudorandom function whose input and output were a billion bits long, and we never wanted to handle any objects larger than this, we could turn it into a hash function by throwing away all but a few hundred bits of the output, and a stream cipher by padding all but a few hundred bits of the input with a constant.

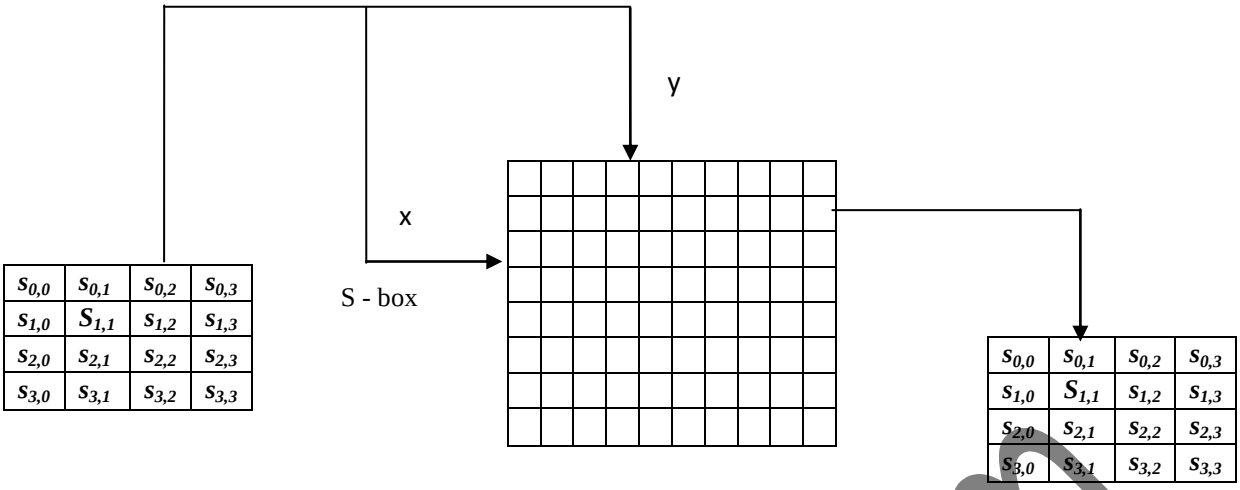


Fig. 3.3 Substitute Bytes Stage of the AES algorithm

The s-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input. In addition, the s-box has no fixed points (s-box(a) = a) and opposite fixed points (s-box(a) =−a) where −a is the bitwise compliment of a. Thes-box must be invertible if decryption is to be possible (Is-box[s-box(a)]= a) however it should not be its self inverse i.e. s-box(a) 6= Is-box(a) Figure

The second basic cryptographic primitive is the random generator, also known as a keystream generator or stream cipher. This is also a random function, but unlike in the hash function case it has a short input and a long output. If we had a good pseudorandom function whose input and output were a billion bits long, and we never wanted to handle any objects larger than this, we could turn it into a hash function by throwing away all but a few hundred bits of the output, and a stream cipher by padding all but a few hundred bits of the input with a constant.

Table-3.1 S- box

X	Y																
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	F2	EB	27	B2	75
	4	09	83	2C	IA	IB	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	D0	D1	00	FF	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	51	EF	AA	F3	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	CD	A3	40	67	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	60	0C	13	DD	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	E0	81	4F	A1	22	2A	90	88	46	EE	B8	14	DE	5E	OB	DB
	A	E7	32	3A	77	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	BA	C8	37	F3	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	70	78	25	62	IC	A6	B4	C6	E8	DD	74	IF	4B	BD	8B	8A
	D	E1	3E	B5	A0	48	03	F6	0E	61	35	57	B9	86	C1	ID	9E
	E	56	F8	98	BC	69	D9	8E	94	9B	IE	87	E9	CE	55	28	DF
	F	8C	A1	89	OD	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

ISSN No.: 2454- 2024 (online)

International Journal of Technical Research & Science

Table-3.2 Inverse S- Box

X	Y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	F	AB	D7	FE	2B	01	6F	30	C5	6F	6B	F2	7B	77	7C	63
	1	76	72	A4	9C	AF	D4	47	AD	F0	47	59	FA	7D	C9	82	CA
	2	C0	31	D8	71	F1	A5	F7	34	CC	F7	3F	36	26	93	FD	B7
	3	15	B2	27	EB	F2	12	05	07	9A	05	96	18	C3	23	C7	04
	4	75	2F	E3	29	B3	3B	5A	52	A0	5A	6E	IB	IA	2C	83	09
	5	84	58	4C	4A	39	CB	B1	6A	5B	B1	FC	20	FF	00	D1	D0
	6	CF	9F	3C	50	7F	F9	33	45	85	33	4D	43	F3	AA	EF	51
	7	A8	F3	FF	10	21	B6	38	BC	F5	38	9D	92	67	40	A3	CD
	8	D2	19	5D	64	3D	A7	44	C4	17	44	97	5F	DD	13	0C	60
	9	73	OB	5E	DE	14	EE	90	46	88	90	2A	22	A1	4F	81	E0
	A	DB	E4	95	91	62	D3	24	C2	5C	24	06	49	77	3A	32	E7
	B	79	AE	7A	65	EA	56	4E	6C	A9	4E	D5	8D	F3	37	C8	BA
	C	08	8B	BD	4B	IF	DD	B4	E8	C6	B4	A6	IC	62	25	78	70
	D	8A	ID	C1	86	B9	35	F6	61	0E	F6	03	48	A0	B5	3E	E1
	E	9E	28	55	CE	E9	IE	8E	9B	94	8E	D9	69	BC	98	F8	56
	F	DF	BB	54	B0	0F	99	42	41	68	42	E6	BF	OD	89	A1	8C

3.1.2 Shift Row Transformation

This stage (known as ShiftRows) is shown in fig. 3.3. This is a simple permutation an nothing more. It works as follow:

- The first row of **state** is *not* altered.
- The second row is shifted 1 bytes to the left in a circular manner.
- The third row is shifted 2 bytes to the left in a circular manner.
- The fourth row is shifted 3 bytes to the left in a circular manner.

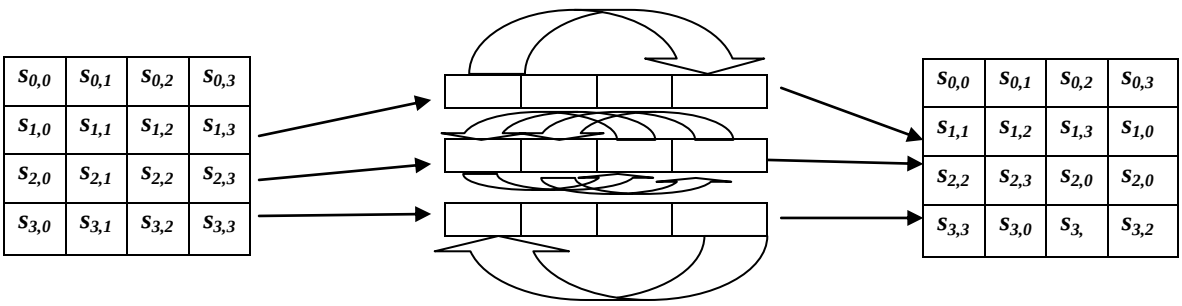


Fig.3.3 Shift Rows Stage

3.1.3 Mix Column Transformation

This stage (known as MixColumn) is basically a substitution but it makes use of arithmeticof GF(28). Each column is operated on individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. The transformation can be determined by the following matrix multiplication on state.

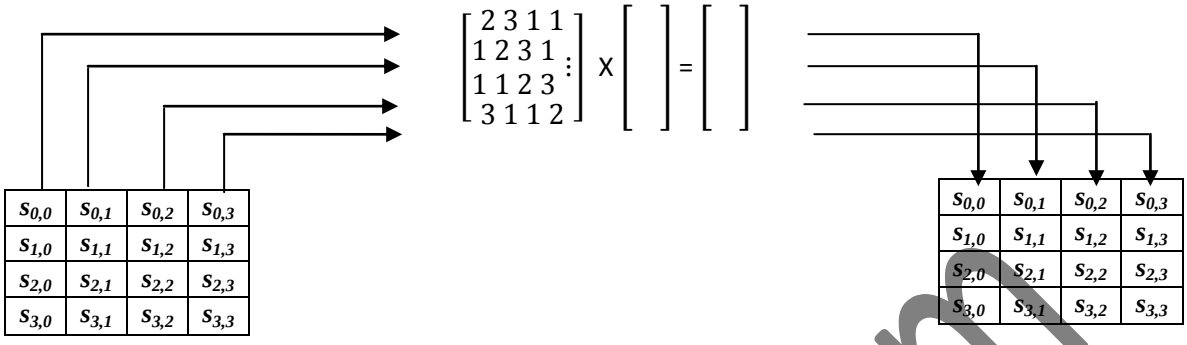


Fig.3.4 Mix Columns Stage

As an example, let's take the first column of a matrix to be s0,0 = {87}, s1,0 ={6E}, s2,0 = {46}, s3,0 = {A6}. This would mean that s0,0 = {87} gets mapped to the value s00,0 = {47} which can be seen by working out the first line of equation3.2 with j = 0. Therefore we have:(02 • 87) _ (03 • 6E) _ 46 _ A6 = 47

3.1.4 Add Round Key Transformation

In this stage (known as addRoundKey) the 128 bits of state are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column wise operation between the 4 bytes of a state column and one word of the round key. This transformation is as simple as possible which helps in efficiency but it also effects every bit of state.

3.2 AES Key Expansion

The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure 3.2. Each word contains 32 bytes which means each subkey is 128 bits long. The program shows the pseudocode for generating the expanded key from the actual key.
Key Expansion (byte key[16], word w[44])

```
{
    Word temp
    For (i=0;i<4;i++)    w[i]=(key[4*i],key[4*i+1], key[4*i+2], key[4*i+3]);
    For (i=4; i<44;i++)
    {
    Temp=w[i];
    If(I mod 4 =0) temp=Sub word (RotWord(temp))+Rcon[i/4];
    W[i]=w[1-4]+temp;
    }
}
```

The program explains the expand key. First it declare size of key and word size, secondly it declare temp function and usinf loop key for compare and rotate, if condition is satisfied sub word compare wit XOR function and decrease the i value this process will contine till compare with sub word after the key expanded.The key is copied into the first four words of the expanded key. The remainder of theexpanded key is filled in four words at a time. Each added word **w**[i] depends on theimmediately preceding word, **w**[i − 1], and the word four positions back **w**[i − 4]. Inthree out of four cases, a simple XOR is used. For a word whose position in the **w**array is a multiple of 4, a more complex function is used. The function g consists of the following sub functions:

- RotWordperforms a one-byte circular left shift on a word. This means that aninput word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
- SubWordperforms a byte substitution on each byte of its input word, using thes-box described earlier.
- The result of steps 1 and 2 is XORed with round constant, Rcon[j].The round constant is a word in which the three rightmost bytes are always 0. Thus theeffect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byteof the word. The round constant is different for each round and is defined as Rcon[j] =(RC[J], 0,0,0), with RC[1]= 1, RC[j]= 2• RC[j −1] and with multiplication definedover the field GF(28).

The key expansion was designed to be resistant to known cryptanalytic attacks. Theinclusion of a round-dependent round constant eliminates the symmetry, or similarity,between the way in which round keys are generated in different rounds. The second basic cryptographic primitive is the random generator, also known as a keystream generator or stream cipher. This is also a random function, but unlike in the hash function case it has a short input and a long output. If we had a good pseudorandom function whose input and output were a billion bits long, and we never wanted to handle any objects larger than this, we could turn it into a hash function by throwing away all but a few hundred bits of the output, and a stream cipher by padding all but a few hundred bits of the input with a constant.

The throughput-area trade-off when the number of stages per one round is changed from one to four. The throughput and area are measured in Giga bits per second (Gb/s) and in slices, respectively. The results show that as the number of stages increases, the area (cost) increases. However, the throughput is saturated when the number of stages is greater than one (two or more stages). In the unrolled AES pipeline, the implementation of the outer pipeline (one pipeline stage per round) achieves a throughput

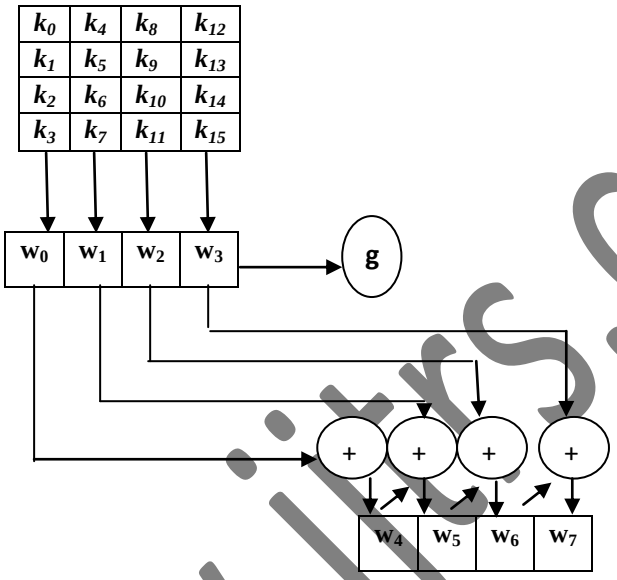


Fig. 3.5 AES Key Expansion

Fig. 3.5 give a summary of each of the rounds. The ShiftRows column is depictedhere as a linear shift which gives a better idea how this section helps in the encryption.The AES algorithm has 10 rounds,where each round could be pipelined with different number of stages. The throughput-area trade-off when the number of stages per one round is changed from one to four. The throughput and area are measured in Giga bits per second (Gb/s) and in slices, respectively.Our results show that as the number of stages increases, the area (cost) increases. However, the throughput is saturated whenthe number of stages is greater than one (two or more stages). In the unrolled AES pipeline, the implementation of the outer pipeline (one pipeline stage per round) achieves a throughput of 45 Gb/s at frequency 360 MHz on Xilinx Virtex V. Since the look-up tables of Sbox are the main critical path in the AES pipeline design,the implementation of the inner pipeline with two, three, or four stages achieves 70 Gb/s at maximum frequency of 557 MHz. From area consumption point of view, one stage per round consumed17% of the FPGA slices, which results in a throughput of 45 Gb/s. The area increases to 23%, 27% and 35% when two, three, and four stages per round are used, respectively, where the throughput remains constant at 70 Gb/s. From this discussion, two stages per round is the best choice for implementation because it gives the best throughput-area trade-off.

3.3 Parallel AES Pipelines

This section analyzes the effect of increasing the number of parallel AES pipelines on the overall performance of Fast Crypto. At a system frequency of 200 MHz, increasing the number of parallel AES pipelines from one to two and then to four pipelines. Note that at system frequency of 200 MHz, one, two, four lanes (AES pipelines) are running at 200 MHz, 100 MHz, and 50 MHz, respectively, and the other parts of the crypto coprocessor are running at 200 MHz. Obviously, the consumed area increases with increasing the number of parallel pipelines, and with increasing the SDQ depth for each individual curve. However, from power consumption perspective, the use of four parallel pipelines running at 50 MHz results in lower power consumption than two parallel pipelines running at 100 MHz, or one pipeline running at 200 MHz .

For more clarity, when the AES pipelines operating frequency is 50 MHz, the effect of increasing the number of parallel lanes on the overall performance of Crypto is dramatically large Moreover, the Crypto throughput when the input data varies from 8 to 3072 blocks. A huge improvement in throughput with increasing the parallel pipelines at constant AES pipelines frequency, where all AES pipelines run at 50 MHz.

The prime numbers are the positive whole numbers with no proper divisors; that is, the only numbers that divide a prime number are 1 and the number itself. By definition, 1 is not prime; so the primes are {2, 3, 5, 7, 11, ...}. The fundamental theorem of arithmetic states that each natural number greater than 1 factors into prime numbers in a way that is unique up to the order of the factors. It is easy to find prime numbers and multiply them together to give a composite number, but much harder to resolve a composite number into its factors. The largest composite product of two large random primes to have been factorized to date was 512 bits 155 digits long; when such a computation was first done, it took several thousand MIPS-years of effort. Recently, however, some Swedish students managed to factor a 512-bit number surreptitiously to solve a challenge cipher, so 512-bit composite numbers are now no more 'secure' than 56-bit DES keys. However, it is believed that a similar number of 1024 bits length could not be factored without an advance in mathematics.

The algorithm commonly used to do public key encryption and digital signatures based on factoring is RSA, named after its inventors Ron Rivest, Adi Shamir, and LenAdleman [649]. It uses Fermat's little theorem, which states that for all primes p not dividing a, a p–1 ≡1 modulo p. (Proof: take the set {1, 2, ... p – 1} and multiply each of them modulo p by a, then cancel out (p – 1)! each side.) Euler's function φ (n) is the number of positive integers less than n with which it has no divisor in common; so if n isthe product of two primes pq then φ(n) = (p – 1) (q – 1) (the proof is similar).The encryption key is a modulus N which is hard to factor (take N = pq for two large randomly chosen primes p and q), plus a public exponent e that has no common factors with either p – 1 or q – 1. The private key is the factors p and q, which are kept secret. Where M isthe message and C is the ciphertext, encryption is defined by:

$$C \equiv M \text{ e modulo } N$$

CONCLUSION

In this project describes implementation of Crypto Co processor using AES. Crypto extends a general-purpose processor with an AES, crypto coprocessor for encrypting data with high throughput. The crypto coprocessor includes parallel AES pipelines with high performance encryption. Moreover, the use of parallel AES pipelines at low frequency reduces the power consumption and provides a scalable system. This article presented a fast and efficient AES cryptography hardware structure that can find many applications. The circuit implementation is very efficient and can be customized to a wide range of applications. The pipelining can be used in faster devices and buses. It represents an improvement over the non-pipeline version and can support many new applications.

REFERENCES

[1] Hodjat, I. Verbauwhede, A 21.54 Gb/s fully pipelined AES processor on FPGA,in: Proc. of 12th Annual IEEE Symposium on Field — Programmable CustomComputing Machines, April, 2004, pp. 308–309.

[2] Hodjat, I. Verbauwhede, Speed-area trade-off for 10 to 100 Gb/s throughputAES processor, in: Proc. of The Thirty-Seventh IEEE Asilomar Conference onSignals Systems and Computers, vol. 2, November 2003, pp. 2147–2150.

[3] Hodjat, I. Verbauwhede, Area-throughput trade-offs for fully pipelined 30to 70 Gb/s AES processors, IEEE Transactions on Computers 55 (4) (2006)366–372.

[4] R. Ho, K. Mai, M. Horowitz, The future of wires, Proceedings of the IEEE 89 (4)(2001) 490–504.

[5] K.U. Jrvinen, M.T. Tommiska, J.O. Skytt, A fully pipelined memoryless 17.8 GBsAES-128 encryptor, in: Proc. of the 2003 ACM/SIGDA Eleventh InternationalSymposium on Field Programmable Gate Arrays, 2003, pp. 207–215.

[6] N.S. Kim, T. Mudge, R. Brown, A 2.3 Gb/s fully integrated and synthesizable AESRijndael core, in: Proc. IEEE Custom Integrated Circuits Conference, September,2003, pp. 193–196.

[7] N.M. Kosaraju, M. Varanasi, S.P. Mohanty, A high-performance VLSI architecturefor advanced encryption standard (AES) algorithm, very large scale integration(VLSI) systems, IEEE Transactions 14 (2) (2006) 111–121.

[8] Kozyrakis, D. Judd, J. Gebis, S. Williams, D. Patterson, K. Yelick,Hardware/compiler codevelopment for an embedded media processor,Proceedings of the IEEE 89 (11) (2001) 1694–1709.

[9] H. Kuo, I. Verbauwhede, Architectural optimization for a 1.82 Gb/s VLSIimplementation of the AES Rijndael algorithm, in: Proc. of CryptographicHardware and Embedded Systems CHES 2001, Paris, France, in: LNCS,vol. 2162, May, 2001, pp. 51–64.[16] F. Mace, F. Standaert, J. Quisquater, FPGA implementation of a scalableencryption Algorithm, in: IEEE Transactions on VLSI Systems, February 2008.