RIJNDAEL FPGA IMPLEMENTATION UTILIZING LOOK-UP TABLES

Máire McLoone, John V McCanny

DSiP™ Laboratories, School of Electrical and Electronic Engineering,
The Queen's University of Belfast, Northern Ireland.
Email: Maire.McLoone@ee.qub.ac.uk, J.McCanny@ee.qub.ac.uk

Abstract: A FPGA Rijndael encryption design is presented, which utilizes look-up tables to implement the entire Rijndael Round function. A comparison is provided between this design and similar existing implementations. Hardware implementations of encryption algorithms prove much faster than equivalent software implementations and since there is a need to perform encryption on data in real time, speed is very important. In particular, Field Programmable Gate Arrays (FPGAs) are well suited to encryption implementations due to their flexibility and an architecture, which can be exploited to accommodate typical encryption transformations. A look-up table based Rijndael design achieves a speed of 12 Gbits/sec, which is a factor 1.2 times faster than an alternative design in which look-up tables are utilized to implement only one of the Round function transformations, and 6 times faster than other previous implementations.

1 INTRODUCTION

In January 1997, the RSA Data Security company issued a challenge to break the US government's Data Encryption Standard (DES) algorithm [1]. In June of that year, the challenge was solved by the DESCHALL [2] team who successfully recovered the 56-bit DES key. In response, the National Institute of Standards and Technology (NIST) requested candidates for a new Advanced Encryption Standard (AES) algorithm to replace DES, realizing that the algorithm's 56-bit key was no longer sufficient to provide the necessary security in many applications. As an interim measure they adopted and standardized Triple-DES, which uses three passes of the DES algorithm and a 112 or 168-bit key. The AES requirements [3] were for a block cipher capable of supporting a data block size of 128-bits and keys of 128, 192 and 256-bits in length. They wanted an algorithm whose security is at least as good as Triple-DES, but with significantly improved efficiency. In October 2000, following a two year rigorous selection process, the Rijndael algorithm [4] was announced as the AES winner. Rijndael proved a secure and efficient algorithm when implemented in both hardware and software across a wide range of platforms.

0-7803-7145-3/01/$10.00 ©2001 IEEE

This paper describes a 16-stage fully pipelined FPGA implementation of the Rijndael algorithm. The design consists mainly of look-up tables, which are implemented as BlockRAMs on the Xilinx Virtex-E XCV812E device [5]. The implementation is compared to an earlier design [6] in which only the ByteSub transformation of the Rijndael Round function is implemented as a look-up table. Other similar encryption-only designs include an implementation on a Virtex XCV1000 FPGA device by Gay and Chodowiec [7], which achieved a data-rate of 331.5 Mbits/sec. Dandalis, Prasanna and Rolim [8] also carried out an implementation on the XCV1000 device and achieved an encryption rate of 353 Mbits/sec. A partially unrolled design by Elbirt, Yip, Chetwynd and Paar [9] on the XCV1000-BG560 FPGA performed at a data-rate of 1937.9 Mbits/sec.

Section 2 of this paper provides a description of the Rijndael algorithm. Section 3 outlines the design of the look-up table based Rijndael implementation. Performance results and comparisons are given in section 4. Finally, concluding remarks are made in section 5.

2 RIJNDAEL ALGORITHM

The Rijndael algorithm operates on a 128-bit block of data and has a variable number of rounds, 10, 12 or 14, when the key length is 128, 192 or 256-bits in length respectively. The algorithm consists of an initial round-key addition, the required number of rounds and a final round, which is a variation of the typical round. An outline of Rijndael is given in Fig. 1 [6].

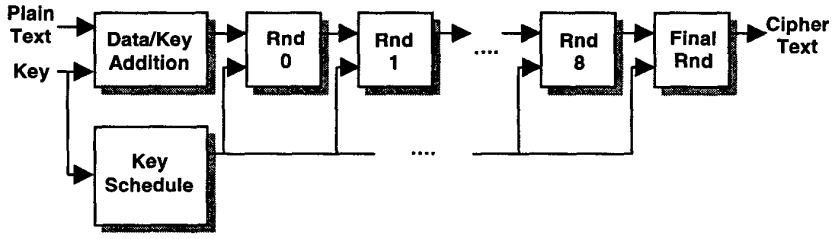

Fig. 1. Outline of 128-bit Key Rijndael Encryption Algorithm

The intermediate cipher result is known as the *State*. The *State* comprises a rectangular array of bytes and a 128-bit plaintext of 16-bytes, B_0, B_1, B_2, B_3 ... B_{15}, is represented as an array of four rows and four columns as illustrated in Fig. 2. Similarly, the key is represented as a rectangular array of bytes, as in Fig. 3, having four rows and a varying number of columns, N_k dependent on the key length. When the key length is 128, 192 or 256-bits, N_k is 4, 6 or 8 respectively. This paper assumes a 128-bit key.

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

Fig.2. State Rectangular Array

$N_k = 4$				$N_k = 6$		$N_k = 8$	
K_0	K_4	K_8	K_{12}	K_{16}	K_{20}	K_{24}	K_{28}
K_1	K_5	K_9	K_{13}	K_{17}	K_{21}	K_{25}	K_{29}
K_2	K_6	K_{10}	K_{14}	K_{18}	K_{22}	K_{26}	K_{30}
K_3	K_7	K_{11}	K_{15}	K_{19}	K_{23}	K_{27}	K_{31}

Fig. 3. Key Rectangular Array

The Rijndael round, as outlined in Fig. 4, consists of four transformations:

- ByteSub transformation
- ShiftRow transformation
- MixColumn transformation
- RoundKey addition

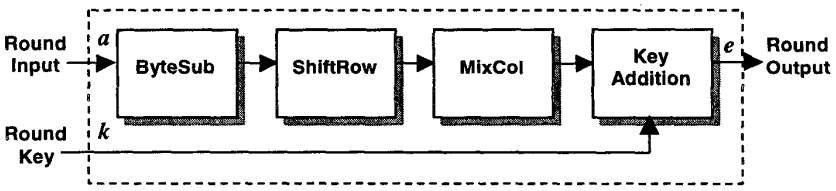

Fig. 4. Rijndael Round

The ByteSub transformation is the *s-box* of the Rijndael algorithm and it operates on each of the State bytes independently. It is constructed by finding the multiplicative inverse of each byte in GF(2^8) and then applying an affine transformation. The affine transformation involves multiplication by a matrix (specified in [4]) followed by addition to the hexadecimal number, '63'. In the ShiftRow transformation, the rows of the State are cyclically shifted to the left. Row 0 is not shifted, row 1 is shifted 2 places and row 3 is shifted 3 places. The columns of the State are considered as polynomials over GF(2^8) for the MixCol transformation, and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$, where,

$$c(x) = '03' x^3 + '01' x^3 + '01' x + '02' \quad (1)$$

and '03', '01' and '02' are hexadecimal numbers. This can be written as a matrix multiplication as given in Equation 2, with a 4-byte input, A_0, A_1, A_2, A_3 and output, B_0, B_1, B_2, B_3. In the final round the MixCol transformation is not included.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad (2)$$

The Round keys are derived from the cipher key and are also represented as an array of four columns and four rows. Each Round key is bitwise XORed to the State in the RoundKey addition transformation.

2.1 Rijndael Key Schedule

The Rijndael key schedule consists of first expanding the cipher key and then from this expansion, selecting the required number of Round keys. Assuming a 128-bit key, the number of rounds in the algorithm is 10 and the number of round keys required is 11. The expanded key is a linear array of 4-byte words, W[0] to W[43]. The first four words contain the cipher key as illustrated in Fig. 5.

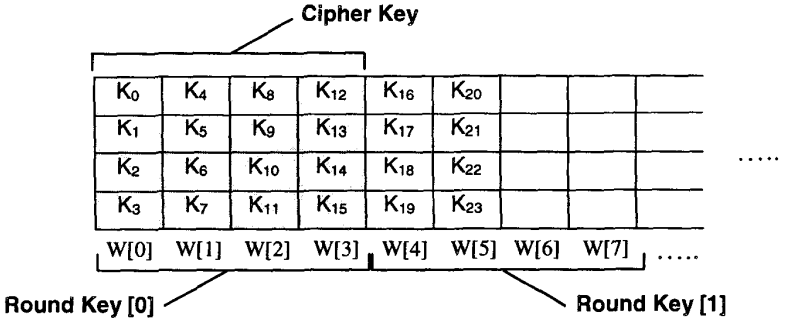

Fig. 5. Expanded Key Array with $N_k = 4$

Each remaining word, W[i] is derived by XORing the previous word, W[i–1] with the word, W[i–4]. For words in positions, which are a multiple of four, a transformation is applied to W[i–1]. Firstly, the bytes in the word are cyclically shifted to the left. For example, a word [a,b,c,d] becomes [b,c,d,a]. Next, each byte in the word is passed through the Rijndael ByteSub transformation and finally, the result is XORed with a round constant. The round constant required for each of the ten rounds is provided in Appendix 3. In the Round key selection process, Round key [0] is taken to be W[0] to W[3], Round key [1] as W[4] to W[7] and so on as shown in Fig. 5 above.

2.2 Decryption

Decrypting data using Rijndael is effectively the inverse of its encryption operation. Data is first passed through the inverse of the final round, then the inverses of the rounds and finally through the initial data/key addition transformation. The key schedule, however, does not change. The Round keys created during key expansion are simply utilized in reverse order. Round key [10] is used in the final round, round key [9] in the round 0, round key [8] in round 1 ... round key [1] in round 8 and round key [0] in the initial data/key addition. The inverse of the round is found by inverting each of the transformations in the round. The inverse ByteSub transformation is the inverse of the affine transformation followed by taking the multiplicative inverse in $GF(2^8)$. In the inverse of the ShiftRow transformation, row 0 is not shifted, row 1 is now shifted by 3 places, row 2 by 2 places and row 3 by only 1 place. To invert the MixCol transformation, the State columns are multiplied modulo $x^4 + 1$ with a fixed polynomial $d(x)$, where,

$$d(x) = \text{'0B'}x^3 + \text{'0D'}x^2 + \text{'09'}x + \text{'0E'} \tag{3}$$

and '0B', '0D', '09' and '0E' are hexadecimal numbers. The initial data/key addition and round key addition are their own inverse.

3 LOOK-UP BASED RIJNDAEL DESIGN

In the design of the encryption-only Rijndael core, it is evident that the ByteSub transformation or *s-box* of the Round function can be implemented as a look-up table (LUT) or ROM. This proves a much more efficient method than implementing the multiplicative inverse operation and affine transformation. The values of the look-up table required are provided in Appendix 1. However, the ShiftRow and MixCol transformations can also be implemented as LUTs rather than using logic. The design is based on the Equation 4 (modified from the equation outlined in [4]),

$$e_j = T_0[a_{0,j}] \oplus T_1[a_{1,j+1}] \oplus T_2[a_{2,j+2}] \oplus T_3[a_{0,j+3}] \oplus k_j \tag{4}$$

where a and k represent the state and key inputs to the round respectively, e represents the output (as shown in Fig. 4) and,

$$T_0 = \begin{bmatrix} S[a]\bullet 02 \\ S[a] \\ S[a] \\ S[a]\bullet 03 \end{bmatrix} \quad T_1 = \begin{bmatrix} S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \\ S[a] \end{bmatrix} \quad T_2 = \begin{bmatrix} S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \\ S[a] \end{bmatrix} \quad T_3 = \begin{bmatrix} S[a] \\ S[a] \\ S[a]\bullet 03 \\ S[a]\bullet 02 \end{bmatrix} \tag{5}$$

($S[a]$ represents a byte, a, being operated on by the Rijndael s-box)

Therefore, two further LUTs are required: LUT_02, containing the values of the ByteSub LUT multiplied in GF(2^8) by the hexadecimal number, '02' (*S[a] • 02*) and LUT_03, containing values of the ByteSub LUT multiplied in GF(2^8) by the hexadecimal number, '03' (*S[a] • 03*). LUT_02 and LUT_03 are given in Appendices 2 and 4 respectively. The Virtex-E family of FPGA devices is utilized for implementation as it contains devices with up to 280 Block RAM (BRAM) memories. A single BRAM can be configured into two single port 256 x 8-bit RAMs. Also, if the BRAM is initialized and both the input data and write enable pins are held low, then the BRAM can be utilized as a ROM. The input round bytes must be passed through each LUT and since a BRAM can accommodate two input bytes, a typical round will require 24 BRAM components to perform an encryption. Fig. 6. illustrates the design required to achieve the first two outputs. The final round excludes the MixCol transformation and hence only needs 8 BRAMs. The 128-bit Rijndael key schedule also incorporates the ByteSub transformation. When expanding the key to create forty words required for the 10 rounds in the algorithm, every fourth word is passed through the s-box with each byte in the word being transformed. Forty 8-bit to 8-bit LUTs or a further 20 BRAMs are required in its implementation. Therefore, the overall design utilizes 244 BRAMs (216 utilized in the 9 typical rounds, 8 in the final round and 20 for the key schedule).

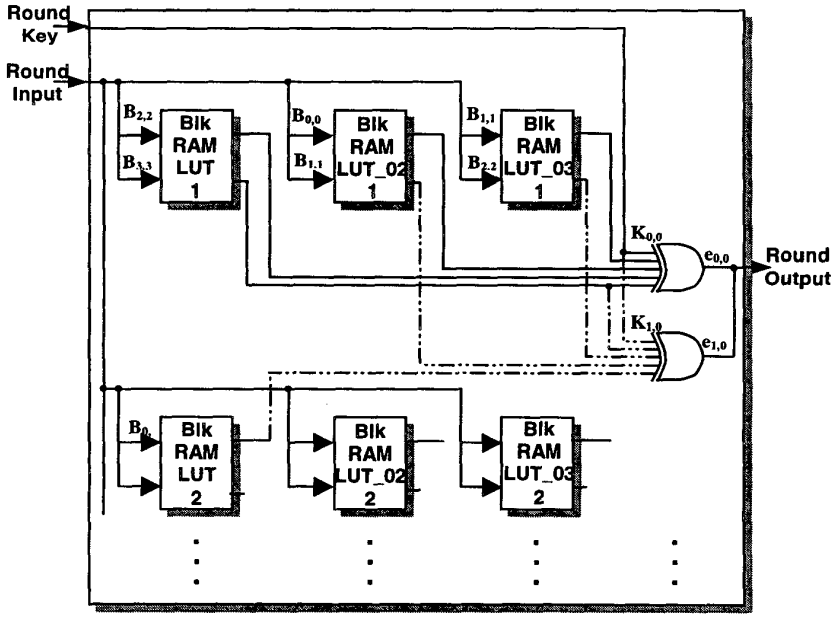

Fig. 6. LUT Based Rijndael Round Design

4 PERFORMANCE RESULTS AND COMPARISON

For the purpose of this paper, only the Rijndael design capable of performing encryption is implemented on an FPGA. Synplify-Pro V.6.0 is used to implement the design and provide pre-placement timing results. In the 16-stage fully pipelined Rijndael encryption-only implementation, data blocks can be accepted every clock cycle and after an initial delay of 10 clock cycles, corresponding to the 10 rounds in the algorithm, the respective encrypted data blocks appear on consecutive clock cycles. The 128-bit key design implemented on an XCV812E-8-bg560 [10] device utilizes 2000 of 9408 CLB slices and 244 of 280 BRAMs. Of IOBs, 384 of 404 are used. The implementation runs at a system clock of 93.9 MHz and achieves a throughput of 12 Gbits/sec. In comparison, the design, which uses BRAMs in the implementation of the ByteSub transformation only [6], when implemented on an XCV812E-8-bg560 device utilizes 2222 CLB slices (23%) at a data-rate of 9.2 Gbits/sec. Timing results from post-placement of this design using Xilinx Foundation Series 3.1i software indicates a throughput of 7 Gbits/sec. Although CLB usage in both these designs is low, many applications will have additional circuitry, which can utilize the remaining CLB resources.

A number of papers that exist describing encryption-only Rijndael FPGA designs are implemented on Xilinx Virtex XCV1000 devices. Throughput comparisons are provided in Table 1.

	Device	RAM Blks	CLB Slices	Data Rate (Mbits/sec)	Data Rate /CLB Slice
Gaj, Chodowiec [7]	XCV1000	-	2902	331.5	0.11
Dandalis *et al* [8]	XCV1000	-	5673	353	0.06
Elbirt *et al*[9]	XCV1000	-	9004	1940	0.22
McLoone, McCanny [6]	XCV812E	100	2222	6956	3.1
McLoone, McCanny LUT-based *Pre-placement timing*	XCV812E	244	2000	12020	4.7

Table 1. Specifications of 128-bit Rijndael Encryption FPGA Implementations

It is evident that post placement, the LUT-based design is the fastest design and the most efficient in terms of CLB slice utilization. Faster Rijndael designs are achieved on the XCV812E FPGA as this device can support a 16-stage fully pipelined design. The speed factor over [6] is achieved through the replacement of the logic required in the multipliers of the MixCol transformation with simple look-up tables.

5 CONCLUSION

Rijndael is set to replace DES as the federal standard and is therefore likely to be used in IPSec protocols, ATM cell encryption and the Secure Socket Layer (SSL) protocol among other applications. Existing Rijndael encryption-only designs have been implemented on Virtex XCV1000 devices. The XCV1000 FPGA consists of 12288 CLB slices and 32 BRAMs. Therefore, the device is not sufficient to support a fully pipelined Rijndael design. The Virtex XCV812E device contains 280 BRAMs and therefore is capable of accommodating the 16-stage pipelined design. When this device is used to implement the design, higher throughputs result. Furthermore, if the RAMs are exploited fully as in the design presented in this paper, an even higher data-rate of 12 Gbits/sec is achieved, a factor 6 times faster than previous implementations on the XCV1000 device.

In the Rijndael round during decryption, the data block and round key are XORed and then passed through the inverse MixCol transformation, the inverse ByteSub transformation and the inverse ShiftRow transformation. The design of a decryption-only implementation is based on Equation 6 where, b is the output of the data/key addition and e is the output of the inverse ShiftRow transformation. Five LUTs are required. The first LUT is the inverse of the LUT utilized in the ByteSub transformation, *InvLUT*. The four other LUTs required are $(0E \bullet b_{i,j})$, $(0B \bullet b_{i,j})$, $(0D \bullet b_{i,j})$ and $(09 \bullet b_{i,j})$.

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} InvS[0E \bullet b_{0,j} \oplus 0B \bullet b_{1,j} \oplus 0D \bullet b_{2,j} \oplus 09 \bullet b_{3,j} \\ InvS[09 \bullet b_{0,j+3} \oplus 0E \bullet b_{1,j+3} \oplus 0B \bullet b_{2,j+3} \oplus 0D \bullet b_{3,j+3}] \\ InvS[0D \bullet b_{0,j+2} \oplus 09 \bullet b_{1,j+2} \oplus 0E \bullet b_{2,j+2} \oplus 0B \bullet b_{3,j+2} \\ InvS[0B \bullet b_{0,j+1} \oplus 0D \bullet b_{1,j+1} \oplus 09 \bullet b_{2,j+1} \oplus 0E \bullet b_{3,j+1} \end{bmatrix} \quad (6)$$

For example, the LUT $(0E \bullet b_{i,j})$ is constructed by multiplying every possible byte from '00' to '11' by '0E' and similarly for the other LUTs. 40 BRAMs per round are required to implement a decryption-only design. Therefore, in the design of a LUT-based Rijndael encryptor/decryptor core, initialization LUTs can be utilized. 40 BRAMs are required in the design of each Round, where only 24 of these need to be operational during encryption. The final round can be implemented using only 8 BRAMs since the MixCol transformation is excluded. 8 additional LUTs or 4 BRAMs are required to initialize the entire design, 3 for encryption (LUT, LUT_02 and LUT_03) and 5 for decryption (InvLUT, LUT_0E, LUT_0D, LUT_0B and LUT_09). The key schedule utilizes 20 BRAMs, hence the overall design requires 392 Block RAMs. The largest Virtex-E device, XCV812E has only 280 BRAMs. However, if the Virtex II family of FPGAs is utilized, which contains devices with a capacity of 392 BRAMs, an extremely fast Rijndael encryptor/decryptor core is achievable.

ACKNOWLEDGEMENTS

This research has been supported by Amphion Semiconductor Ltd. and by a University Research Studentship, which incorporates funding by the European Social Fund.

APPENDICES

Appendix 1: The Hexadecimal values contained in the LUT utilized during encryption are as outlined in the Table 2 below. For example, an input of '00' (hexadecimal) would return the output, '63', an input of '08' would return the output, '30'.

	0	1	2	3	4	5	6	7
0	63	7C	77	7B	F2	6B	6F	C5
1	CA	82	C9	7D	FA	59	47	F0
2	B7	FD	93	26	36	3F	F7	CC
3	04	C7	23	C3	18	96	05	9A
4	09	83	2C	1A	1B	6E	5A	A0
5	53	D1	00	ED	20	FC	B1	5B
6	D0	EF	AA	FB	43	4D	33	85
7	51	A3	40	8F	92	9D	38	F5
8	CD	0C	13	EC	5F	97	44	17
9	60	81	4F	DC	22	2A	90	88
A	E0	32	3A	0A	49	06	24	5C
B	E7	C8	37	6D	8D	D5	4E	A9
C	BA	78	25	2E	1C	A6	B4	C6
D	70	3E	B5	66	48	03	F6	0E
E	E1	F8	98	11	69	D9	8E	94
F	8C	A1	89	0D	BF	E6	42	68

	8	9	A	B	C	D	E	F
0	30	01	67	2B	FE	D7	AB	76
1	AD	D4	A2	AF	9C	A4	72	C0
2	34	A5	E5	F1	71	D8	31	15
3	07	12	80	E2	EB	27	B2	75
4	52	3B	D6	B3	29	E3	2F	84
5	6A	CB	BE	39	4A	4C	58	CF
6	45	F9	02	7F	50	3C	9F	A8
7	BC	B6	DA	21	10	FF	F3	D2
8	C4	A7	7E	3D	64	5D	19	73
9	46	EE	B8	14	DE	5E	0B	DB
A	C2	D3	AC	62	91	95	E4	79
B	6C	56	F4	EA	65	7A	AE	08
C	E8	DD	74	1F	4B	BD	8B	8A
D	61	35	57	B9	86	C1	1D	9E
E	9B	1E	87	E9	CE	55	28	DF
F	41	99	2D	0F	B0	54	BB	16

Table 2. LUT Utilized During Encryption

Appendix 2: Table 3 contains the values of the ByteSub LUT multiplied in GF(2^8) by the hexadecimal number, '02' (*S[a]* •*02* or LUT_02).

	0	1	2	3	4	5	6	7
0	C6	F8	EE	F6	FF	D6	DE	91
1	8F	1F	89	FA	EF	B2	8E	FB
2	75	E1	3D	4C	6C	7E	F5	83
3	08	95	46	9D	30	37	0A	2F
4	12	1D	58	34	36	DC	B4	5B
5	A6	B9	00	C1	40	E3	79	B6
6	BB	C5	4F	ED	86	9A	66	11
7	A2	5D	80	05	3F	21	70	F1
8	81	18	26	C3	BE	35	88	2E
9	C0	19	9E	A3	44	54	3B	0B
A	DB	64	74	14	92	0C	48	B8
B	D5	8B	6E	DA	01	B1	9C	49
C	6F	F0	4A	5C	38	57	73	97
D	E0	7C	71	CC	90	06	F7	1C
E	D9	EB	2B	22	D2	A9	07	33
F	03	59	09	1A	65	D7	84	D0

	8	9	A	B	C	D	E	F
0	60	02	CE	56	E7	B5	4D	EC
1	41	B3	5F	45	23	53	E4	9B
2	68	51	D1	F9	E2	AB	62	2A
3	0E	24	1B	DF	CD	4E	7F	EA
4	A4	76	B7	7D	52	DD	5E	13
5	D4	8D	67	72	94	98	B0	85
6	8A	E9	04	FE	A0	78	25	4B
7	63	77	AF	42	20	E5	FD	BF
8	93	55	FC	7A	C8	BA	32	E6
9	8C	C7	6B	28	A7	BC	16	AD
A	9F	BD	43	C4	39	31	D3	F2
B	D8	AC	F3	CA	CF	F4	47	10
C	CB	A1	E8	3E	96	61	0D	0F
D	C2	6A	AE	69	17	99	3A	27
E	2D	3C	15	C9	87	AA	50	A5
F	82	29	5A	1E	7B	A8	6D	2C

Table 3. LUT_02 Utilized During Encryption

Appendix 3: The round constants used in the Rijndael key schedule are 4-byte vectors, Rcon[i] = (RC[i], '00', '00', 00') where the values of RC[i] are as follows:

RC[1] = '01'	RC[2] = '02'	RC[3] = '04'	RC[4] = '08'	RC[5] = '10'
RC[6] = '20'	RC[7] = '40'	RC[8] = '80'	RC[9] = '1B'	RC[10] = '36'

Appendix 4: Table 4 contains the values of the ByteSub LUT multiplied in GF(2^8) by the hexadecimal number, '03' (*S[a] • 03* or LUT_03).

	0	1	2	3	4	5	6	7
0	A5	84	99	8D	0D	BD	B1	54
1	45	9D	40	87	15	EB	C9	0B
2	C2	1C	AE	6A	5A	41	02	4F
3	0C	52	65	5E	28	A1	0F	B5
4	1B	9E	74	2E	2D	B2	EE	FB
5	F5	68	00	2C	60	1F	C8	ED
6	6B	2A	E5	16	C5	D7	55	94
7	F3	FE	C0	8A	AD	BC	48	04
8	4C	14	35	2F	E1	A2	CC	39
9	A0	98	D1	7F	66	7E	AB	83
A	3B	56	4E	1E	DB	0A	6C	E4
B	32	43	59	B7	8C	64	D2	E0
C	D5	88	6F	72	24	F1	C7	51
D	90	42	C4	AA	D8	05	01	12
E	38	13	B3	33	BB	70	89	A7
F	8F	F8	80	17	DA	31	C6	B8

	8	9	A	B	C	D	E	F
0	50	03	A9	7D	19	62	E6	9A
1	EC	67	FD	EA	BF	F7	96	5B
2	5C	F4	34	08	93	73	53	3F
3	09	36	9B	3D	26	69	CD	9F
4	F6	4D	61	CE	7B	3E	71	97
5	BE	46	D9	4B	DE	D4	E0	4A
6	CF	10	06	81	F0	44	BA	E3
7	DF	C1	75	63	30	1A	0E	6D
8	57	F2	82	47	AC	E7	2B	95
9	CA	29	D3	3C	79	E2	1D	76
A	5D	6E	EF	A6	A8	A4	37	8B
B	B4	FA	07	AF	25	8E	E9	18
C	23	7C	9C	21	DD	DC	86	85
D	A3	5F	F9	D0	91	58	27	B9
E	B6	22	92	20	49	FF	78	7A
F	C3	B0	77	11	CB	FC	D6	3A

Table 4. LUT_03 Utilized During Encryption

REFERENCES

[1] National Bureau of Standards; Data Encryption Standard; Federal Information Processing Standards Publication, *FIPS PUB 46*; January 1977.

[2] RSA Security; RSA's 56-bit DES Challenge; URL: http://www.rsasecurity.com/news/pr/970619-1.htm; April, 2001

[3] AES Development Effort; URL: http://csrc.nist.gov/encryption/aes; April 2001.

[4] J. Daemen, V. Rijmen; The Rijndael Block Cipher: AES Proposal; *First AES Candidate Conference (AES1)*; August 20-22, 1998.

[5] M. McLoone, J.V. McCanny; Apparatus for Selectably Encrypting and Decrypting Data; *UK Patent* Application No. 0107592.8; Filed 27 March 2001

[6] M. McLoone, J.V. McCanny; High Performance Single-Chip FPGA Rijndael Algorithm Implementations; *Third International Cryptographic Hardware and Embedded Systems Workshop – CHES 2001*; May, 2001.

[7] K. Gaj, P. Chodowiec; Comparison of the Hardware Performance of the AES Candidates using Reconfigurable Hardware; *The Third Advanced Encryption Standard (AES3) Candidate Conference*; 13-14 April 2000, New York.

[8] A. Dandalis, V.K. Prasanna, J.D.P. Rolim ; A Comparative Study of Performance of AES Candidates Using FPGAs ; The Third Advanced Encryption Standard (AES3) Candidate Conference, 13-14 April 2000, New York, USA.

[9] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar ; An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists ; The Third Advanced Encryption Standard (AES3) Candidate Conference, 13-14 April 2000, New York, USA.

[10] Xilinx Virtex™-E 1.8V Field Programmable Gate Arrays; URL: http://www.xilinx.com; November 2000.