High Performance Single-Chip FPGA Rijndael Algorithm Implementations

Máire McLoone and J.V McCanny

DSiP™ Laboratories, School of Electrical and Electronic Engineering,
The Queen's University of Belfast, Belfast BT9 5AH, Northern Ireland
Maire.McLoone@ee.qub.ac.uk, J.McCanny@ee.qub.ac.uk

Abstract. This paper describes high performance single-chip FPGA implementations of the new Advanced Encryption Standard (AES) algorithm, Rijndael. The designs are implemented on the Virtex-E FPGA family of devices. FPGAs have proven to be very effective in implementing encryption algorithms. They provide more flexibility than ASIC implementations and produce higher data-rates than equivalent software implementations. A novel, generic, parameterisable Rijndael encryptor core capable of supporting varying key sizes is presented. The 192-bit key and 256-bit key designs run at data rates of 5.8 Gbits/sec and 5.1 Gbits/sec respectively. The 128-bit key encryptor core has a throughput of 7 Gbits/sec which is 3.5 times faster than similar existing hardware designs and 21 times faster than known software implementations, making it the fastest single-chip FPGA Rijndael encryptor core reported to date. A fully pipelined single-chip 128-bit key Rijndael encryptor/decryptor core is also presented. This design runs at a data rate of 3.2 Gbits/sec on a Xilinx Virtex-E XCV3200E-8-CG1156 FPGA device. There are no known single-chip FPGA implementations of an encryptor/decryptor Rijndael design.

Keywords: FPGA Implementation, AES, Rijndael, Encryption

1 Introduction

In September 1997 the National Institute of Standards and Technology (NIST) issued a request for possible candidates for a new Advanced Encryption Standard (AES) to replace the Data Encryption Standard (DES). In August 1998, 15 candidate algorithms were selected and a year later, in August 1999 five finalists were announced: MARS, RC6, Rijndael, Serpent and Twofish. On 02 October 2000, the Rijndael algorithm [1], developed by Joan Daemen and Vincent Rijmen was selected as the winner of the AES development race. In performance comparison studies carried out on all five finalists [2,3,4,7], Rijndael proved to be one of the fastest and most efficient algorithms. It is also easily implemented on a wide range of platforms and is extendable to other key and block lengths.

In this paper two fully pipelined Rijndael algorithm designs are presented. The designs are implemented using Xilinx Foundation Series 3.1i software on the Virtex-E FPGA family of devices [5]. A fully pipelined Rijndael design requires considerable memory, hence, its implementation is ideally suited to the Virtex-E and

Virtex-E Extended Memory range of FPGAs, which contain devices with up to 280 RAM Blocks (BRAMs). The first design presented is an encryption-only design capable of supporting 128-bit, 192-bit and 256-bit keys. The 128-bit key design is implemented on the XCV812E-8-BG560 device. The 192-bit and 256-bits are implemented on XCV3200E-8-CG1156 devices, as they are too large to place on the XCV812E device. The authors are not aware of any other Rijndael hardware design capable of supporting varying key sizes. However, software designs do exist. The fastest known software implementations of Rijndael are by Brian Gladman [6]. On a 933 MHz Pentium III processor, his 128-bit key design achieves a throughput of 325 Mbits/sec, the 192-bit key design runs at 275 Mbits/sec and the 256-bit key design at 236 Mbit/sec. Hardware implementations of a 128-bit key design do exist. An implementation on a Virtex XCV1000 FPGA device by Gaj and Chodowiec [4] achieved a data-rate of 331.5 Mbits/sec. Dandalis, Prasanna and Rolim [2] carried out an implementation on a Xilinx Virtex device and achieved an encryption rate of 353 Mbits/sec. A partially unrolled design by Elbirt, Yip, Chetwynd and Paar [7] on the Virtex XCV1000-BG560 FPGA performed at a data-rate of 1937.9 Mbits. The second design [8] presented is capable of both encryption and decryption operations and is implemented on an XCV3200E-8-CG1156 device. There are no known single-chip FPGA implementations of the Rijndael algorithm, which perform both encryption and decryption. However, Ichikawa, Kasuya and Matsui's [3] implementation on a CMOS ASIC achieves 1950 Mbits/sec. The encryptor/decryptor implementation by Weeks, Bean, Rozylowicz and Ficke [9] performs at a rate of 5163 Mbits/sec on a CMOS ASIC.

Section 2 of the paper provides a description of the Rijndael Algorithm. Section 3 outlines the design of the fully pipelined Rijndael implementations. Performance results are given in section 4. Finally, concluding remarks are made in section 5.

2 Rijndael Algorithm

The Rijndael algorithm is an iterated block cipher. The block and key lengths can be 128, 192 or 256 bits. The NIST requested that the AES must implement a symmetric block cipher with a block size of 128 bits, hence the variations of Rijndael which can operate on larger block sizes will not be included in the actual standard. Rijndael also has a variable number of iterations or '*rounds*': 10, 12 and 14 when the key lengths are 128, 192 and 256 respectively. The transformations in Rijndael consider the data block as a 4 column rectangular array of 4-byte vectors or *State*, as shown in Fig. 1. A 128-bit plaintext consists of 16 bytes, B_0, B_1, B_2, B_3, B_4... B_{14}, B_{15}. Hence, B_0 becomes $P_{0,0}$, B_1 becomes $P_{1,0}$, B_2 becomes $P_{2,0}$ ... B_4 becomes $P_{0,1}$ and so on. The key is also considered to be a rectangular array of 4-byte vectors, the number of columns, N_k, of which is dependent on the key length. This is illustrated in Fig. 2. The algorithm design consists of an initial data/key addition, nine, eleven or thirteen rounds when the key length is 128-bits, 192-bits or 256-bits respectively and a final round, which is a variation of the typical round. The Rijndael key schedule expands the key entering the cipher so that a different sub-key or *round key* is created for each algorithm iteration. An outline of Rijndael is shown in Fig. 3.

$P_{0,0}$	$P_{0,1}$	$P_{0,2}$	$P_{0,3}$
$P_{1,0}$	$P_{1,1}$	$P_{1,2}$	$P_{1,3}$
$P_{2,0}$	$P_{2,1}$	$P_{2,2}$	$P_{2,3}$
$P_{3,0}$	$P_{3,1}$	$P_{3,2}$	$P_{3,3}$

Fig. 1. State Rectangular Array

	$N_k = 4$			$N_k = 6$		$N_k = 8$	
$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$	$K_{0,4}$	$K_{0,5}$	$K_{0,6}$	$K_{0,7}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$	$K_{1,4}$	$K_{1,5}$	$K_{1,6}$	$K_{1,7}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$	$K_{2,4}$	$K_{2,5}$	$K_{2,6}$	$K_{2,7}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$	$K_{3,4}$	$K_{3,5}$	$K_{3,6}$	$K_{3,7}$

Fig. 2. Key Rectangular Array

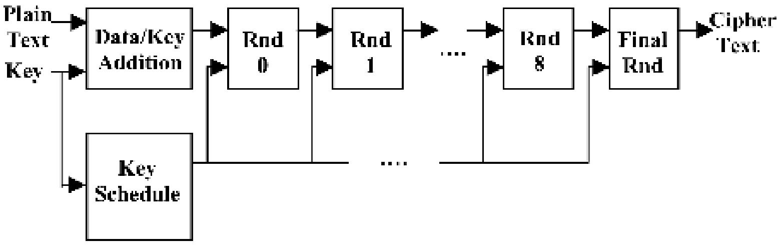

Fig. 3. Outline of 128-bit Key Rijndael Encryption Algorithm

2.1 Rijndael Round

The Rijndael round comprises a ByteSub Transformation, a ShiftRow Transformation, a MixColumn Transformation and a Round Key Addition. The ByteSub transformation is the *s-box* of the Rijndael algorithm and operates on each of the State bytes independently. The s-box is constructed by finding the multiplicative inverse of each byte in GF(2^8). An affine transformation is then applied, which involves multiplying the result by a matrix and adding 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 1 place, row 2 by 2 places and row 3 by 3 places. The MixColumn transformation operates on the columns of the State. Each column is considered a polynomial over GF(2^8) and multiplied modulo x^4+1 with a fixed polynomial $c(x)$, where,

$$c(x) = \text{'03'}x^3 + \text{'01'}x^2 + \text{'01'}x + \text{'02'} \tag{1}$$

Finally the State bytes and round-key bytes are XORed in Round Key Addition. A typical Rijndael round is illustrated in Fig. 4. In the final round the MixColumn transformation is excluded.

2.2 Key Schedule

The Rijndael key schedule consists of two parts: Key Expansion and Round Key Selection. Key Expansion involves expanding the cipher key into a linear array of 4-byte words, the length of which is determined by the data block length, N_b, multiplied

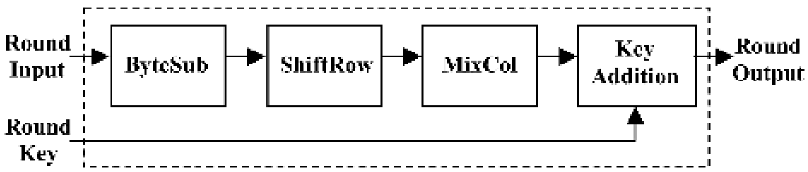

Fig. 4. Rijndael Round

by the number of rounds, N_r plus 1, i.e. N_b * (N_r + 1). The data block length, N_b = 4. When the key block length, N_k = 4, 6 and 8, the number of rounds is 10, 12 and 14 respectively. Hence the lengths of the expanded key are as shown in Table 1.

Table 1. Length of Expanded Key for Varying Key Sizes

Data Block Length, N_b	4	4	4
Key Block Length, N_k	4	6	8
Number of Rounds, N_r	10	12	14
Expanded Key Length	*44*	*52*	*60*

The first N_k words of the expanded key contain the cipher key. When N_k = 4 or 6, each remaining word, W[i], is found by XORing the previous word, W[i-1] with the word N_k positions earlier, W[i-N_k]. For words in positions, which are a multiple of N_k, a transformation is applied to W[i-1] before it is XORed. This transformation involves a cyclic shift of the bytes in the word. Each byte is passed through the Rijndael s-box and the resulting word is XORed with a round constant. However, when N_k = 8, an additional transformation is applied. For words in positions, which are a multiple of ($N_k.i$ + 4), each byte of the word, W[i-1], is passed through the Rijndael s-box. The round keys are selected from the expanded key. In a design with N_r rounds, N_r +1 round keys are required. For example a 10-round design requires 11 round keys. Round key 0 is W[0] to W[3] and is utilized in the initial data/key addition, round key 1 is W[4] to W[7] and is used in round 0, round key 2 is W[8] to W[11] and used in round 1 and so on. Finally, round key 10 is used in the final round.

2.3 Decryption

The decryption process in Rijndael is effectively the inverse of its encryption process. It comprises an inverse of the final round, inverses of the rounds, followed by the initial data/key addition. The data/key addition remains the same as it involves an XOR operation, which is its own inverse. The inverse of the round is found by inverting each of the transformations in the round. The inverse of ByteSub is obtained by applying the inverse of the affine transformation and taking the multiplicative inverse in GF(2^8) of the result. In the inverse of the ShiftRow transformation, row 0 is not shifted, row 1 is now shifted 3 places, row 2 by 2 places and row 3 by 1 place. The polynomial, $c(x)$, used to transform the State columns in the inverse of MixColumn is given by,

$$c(x) = \text{‘0B’}x^3 + \text{‘0D’}x^2 + \text{‘09’}x + \text{‘0E’} \tag{2}$$

Similarly to the data/key addition, Round Key addition is its own inverse. During decryption, the key schedule does not change, however the round keys constructed are now used in reverse order. For example, in a 10-round design, round key 0 is still utilized in the initial data/key addition and round key 10 in the final round. However, round key 1 is now used in round 8, round key 2 in round 7 and so on.

3 Design of Pipelined Rijndael Implementations

The Rijndael algorithm implementations presented in this paper are based on the Electronic Codebook (ECB) mode. Although ECB mode is less secure than other modes of operation, it is commonly used and its operation can be pipelined [10]. The fully pipelined Rijndael implementation will also operate in Counter mode. Counter mode is a simplification of Output Feedback (OFB) mode and it involves updating the input plaintext block, P, as a counter, $P_{j+1} = P_j+1$, rather than using feedback. Hence, the ciphertext block, C, is not required in order to encrypt plaintext block, P+1 [11]. Counter mode provides more security than ECB mode and operation in either mode will achieve high throughputs.

A number of different architectures can be considered when designing encryption algorithms [7]. These are described as follows. Iterative Looping (IL) is where only one round is designed, hence for an n-round algorithm, n iterations of that round are carried out to perform an encryption. Loop Unrolling (LU) involves the unrolling of multiple rounds. Pipelining (P) is achieved by replicating the round and placing registers between each round to control the flow of data. A pipelined architecture generally provides the highest throughput. Sub-Pipelining (SP) is carried out on a partially pipelined design when the round is complex. It decreases the pipeline's delay between stages but increases the number of clock cycles required to perform an encryption.

The Rijndael designs described in this paper are coded using VHDL and are fully pipelined: the encryption design having ten, twelve or fourteen pipeline stages and the encryption/decryption design having ten pipeline stages.

3.1 Design of Generic Rijndael Encryptor Core

The main consideration in both designs is the memory requirement. The Rijndael s-box in the ByteSub transformation can be implemented as a look-up table (LUT) or ROM. This proves a faster and more cost-effective method than implementing the multiplicative inverse operation and affine transformation. Since the State bytes are operated on individually, each Rijndael round requires sixteen 8-bit to 8-bit LUTs. In the key schedule, LUTs can also be used, as words are passed through the s-box. The

Virtex-E and Virtex-E Extended Memory range of FPGAs are utilized for implementation as they contain devices with up to 280 Block SelectRAM (BRAM) memories. A single BRAM can be configured into two single port 256 x 8-bit RAMs, hence, eight BRAMs are used in each round. When the write enable of the RAM is low ('0'), transitions on the write clock are ignored and data stored in the RAM is not affected. Hence, if the RAM is initialized and both the input data and write enable pins are held low then the RAM can be utilized as a ROM or LUT.

The ShiftRow transformation is simply hardwired as no logic is involved. The MixColumn transformation can be written as a matrix multiplication as given in Equation 3, with a 4-byte input, a_0, a_1, a_2, a_3 and output, b_0, b_1, b_2, b_3.

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad (3)$$

The transformation is implemented by XORing the results of the multiplications in GF(2^8) in accordance with Equation (3), as illustrated in Fig. 5.

The flowchart in Fig. 6 outlines the various stages involved in the Rijndael key schedule for key lengths of 128, 192 and 256-bits in length. N_k, N_b and N_r represent the key block length, the data block length and the number of rounds respectively. The input to the key schedule is the cipher key and key block length and the outputs are the Round keys. The Round keys are created as required, hence, Round key [0] is available immediately, Round key [1] is created one clock cycle later and so on. The various functions utilized in the key schedule are as follows:

Rem Function	:	Returns the remainder value in a division
SubByte Function	:	Operates on a 4-byte word and each byte is passed through the Rijndael s-box
RotByte Function	:	Involves a cyclic shift to the left of the bytes in a 4-byte word. For example, an input of x_0, x_1, x_2, x_3, will produce the output x_1, x_2, x_3, x_0.
Rcon Function	:	Returns a 4-byte vector, *Rcon*[*i*] = *RC*[*i*], '00', '00', '00') where the values of *RC*[*i*] are outlined in Table 2.

Table 2. Rijndael Key Schedule Round Constants

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'

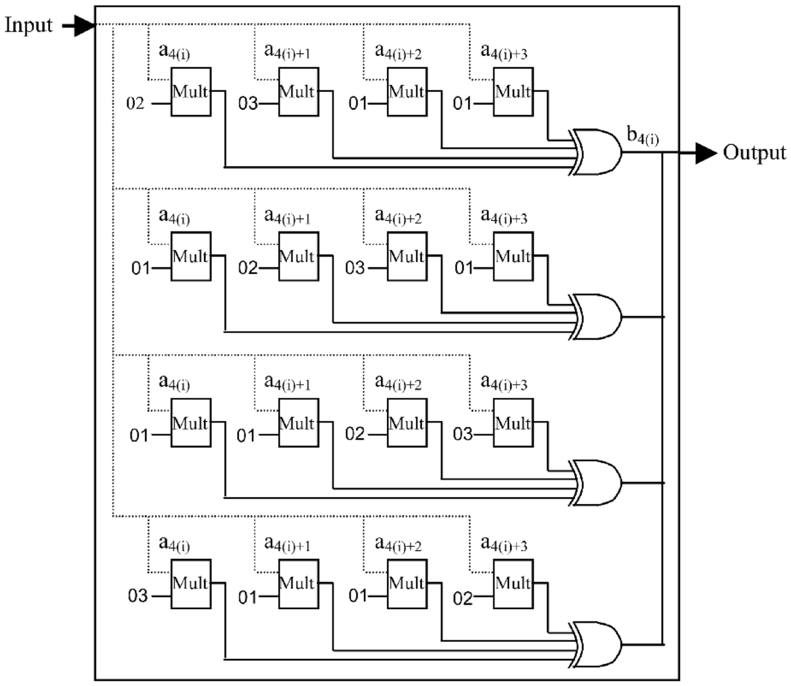

Fig. 5. Design of MixColumn Transformation

When utilizing a 128-bit key, 40 words are created during expansion of the key and every fourth word is passed through the s-box with each byte in the word being transformed. Forty 8-bit to 8-bit LUTs and hence 20 BRAMs are required in its implementation. Similarly, for a 192-bit key 16 BRAMs are required. With a 256-bit key, 26 BRAMs are needed – 14 are utilized for words in positions which are a multiple of 8 and a further 12 are used for words in positions which are a multiple of ($8.i + 4$) for $8<i<60$.

Thus, in the overall 128-bit key design, a total of 100 ROMs are required, 80 ROMs are required for the 10 rounds and a further 20 for the key schedule. Similarly, 112 ROMS are required for the 192-bit design (96 for the 12 rounds and 16 for the key schedule) and 138 for the 256-bit design (112 for the 14 rounds and 26 for the key schedule).

3.2 Design of 128-bit Key Rijndael Encryptor/Decryptor Core

In the decryption operation, the inverse of the ByteSub transformation can also be implemented as a LUT. However the values in this LUT are different to those required for encryption. Therefore, it is necessary to accommodate for both encryption and decryption. One method would involve doubling the number of BRAMs utilized, however, this would prove costly on area. In the Rijndael

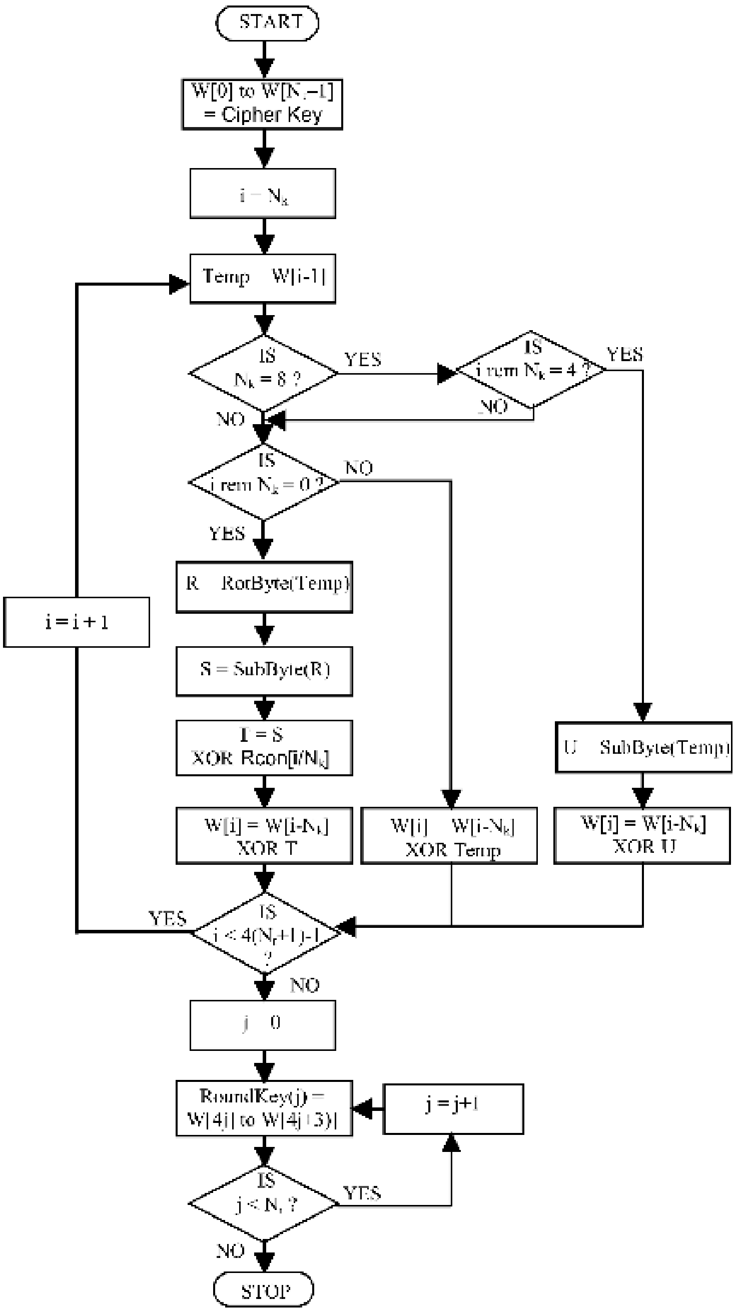

Fig. 6. Rijndael Key Schedule

encryption/decryption design, this was overcome by the addition of two BRAMs, which were utilized as ROMs, one containing the initialization values for the LUTs required during encryption, the other containing the values for the LUTs required during decryption. Therefore, instead of initializing each individual BRAM as a ROM, when the design is set to encrypt, all the BRAMs are initialized with data read from the ROM containing the values required for encryption. When the design is set to decrypt, the BRAMs are initialized with data from the ROM containing the values required for the decryption operation. This initialization procedure is outlined in Fig. 7.

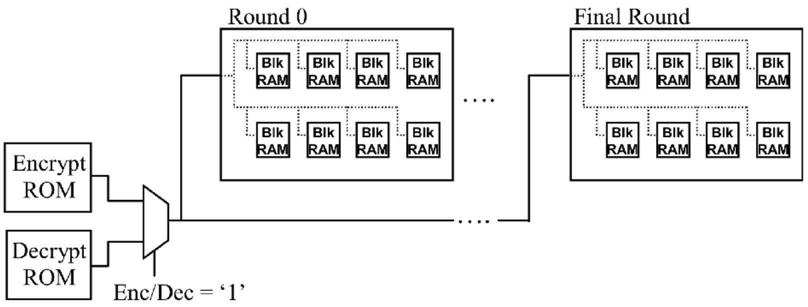

Fig. 7. Initialization of Block RAMs in Rijndael Design

The Inverse ShiftRow transformation is also hardwired. Multiplexors (MUXs) select between the ShiftRow and Inverse ShiftRow wiring. Similarly to Fig. 5, the Inverse MixColumn transformation can be implemented by XORing results of the multiplications in GF(2^8) and again, MUXs are used to select between the values required for encryption and those required during decryption.

Since the encryptor/decryptor core design assumes a key length of 128 bits, the design of the key schedule is a simplification of that shown in the flowchart illustrated in Fig. 6. During decryption, the values of the LUTs utilized in the key schedule do not change, hence, the LUTs can simply be implemented as ROMs. However, the round keys are used in reverse order. The initialization process for either encryption or decryption takes 256 clock cycles as the 256 values contained in each ROM are read. Since the system clock for the encryption/decryption design is 25.3 MHz, this corresponds to an initialization time of only 10 µs. When encrypting data, the keys are produced as each round requires them, therefore, the encryption will take 10 clock cycles corresponding to the 10 rounds when using a 128-bit key. The design assumes that the same key is utilized during a session of data transfer. If decrypting data, the initialization process will be as described above. However, initial decryption will take 20 clock cycles, 10 clock cycles for the required round keys to be constructed and a further 10 corresponding to the 10 rounds. The overall 128-bit key encryptor/decryptor design, therefore, requires 102 BRAMs.

4 Performance Results

The Rijndael designs are implemented using Xilinx Foundation Series 3.1i software and Synplify Pro V6.0 on Xilinx Virtex-E FPGA devices. Data blocks can be accepted every clock cycle and after an initial delay the respective encrypted/decrypted data blocks appear on consecutive clock cycles. The first design implemented is the generic encryptor core, which supports 128-bit, 192-bit and 256-bit keys. The performance results obtained for this implementation will be similar to those of a design with only decryption capabilities. The main difference in the two implementations would be the initial delay time as mentioned in section 3. The 128-bit key encryption design implemented on the Virtex-E XCV812e-8bg560 device, utilizes 2222 CLB slices (23%) and 100 BRAMs (35%). Of IOBs 384 of 404 are used. The design uses a system clock of 54.35 MHz and runs at a data-rate of 7 Gbits/sec (870 Mbytes/sec). This result proves faster than similar existing FPGA implementations, as illustrated in Table 3 below. The implementations included in the table are as outlined in section 1. The design is also the most efficient in terms of CLB utilization although it must be remembered that the previous implementations were limited in their use of device.

Table 3. Specifications of 128-bit Key Rijndael Encryption FPGA Implementations

	Type	**Device**	**Area (CLB Slices)**	**Throughput (Mbits/s)**	**Throughput /Area (Mbits/s*Slices)**
Gaj *et al*[4]	IL	XCV1000	2902	331.5	0.11
Dandalis *et al*[2]	IL	XCV1000	5673	353	0.06
Elbirt *et al*[7]	SP	XCV1000	9004	1940	0.22
McLoone *et al*	P	XCV812E	2222	6956	3.1

Both the 192-bit and 256-bit key encryption designs are implemented on Virtex-E XCV3200e-8-cg1156 devices, as they require a higher number of IOBs than that available on the XCV812E device. The 192-bit key encryption design utilizes 2577 CLB slices (7%) and 112 BRAMs (53%). Of IOBs 448 of 804 are used. The design uses a system clock of 45.44 MHz and runs at a data-rate of 5.8 Gbits/sec (727 Mbytes/sec). The 256-bit key encryption utilizes 2995 CLB slices (9%) and 138 BRAMs (66%). Of IOBs 512 of 804 are used. The design uses a system clock of

39.88 MHz and runs at a data-rate of 5 Gbits/sec (638 Mbytes/sec). There have been no FPGA implementations of Rijndael designs capable of supporting 128-bit, 192-bit and 256-bit keys published to date.

The second design implemented is the Rijndael encryptor/decryptor design. On the Virtex-E XCV3200e-8-cg1156 device, this design utilizes 7576 CLB slices (23%) and 102 BRAMs (49%). Of IOBs 385 of 804 are used. The design uses a system clock of 25.3 MHz and runs at a data-rate of 3239 Mbits/sec (405 Mbytes/sec). There are no known similar single-chip FPGA encryptor/decryptor implementations. Also, the results obtained compare very well with existing ASIC implementations, as illustrated in Table 4 below.

Table 4. Specifications of Rijndael ASIC Implementations

	Device	**Throughput (Mbits/sec)**
Ichikawa, Kasuya ,Matsui [3]	CMOS	1950
Weeks, Bean, Rozylowicz, Ficke [8]	CMOS	5163
McLoone, McCanny	XCV3200E	3239

It is possible to enhance the performance figures of the two designs presented by further optimization of the algorithm specific to the requirements of the FPGA device on which the design is to be implemented. However, this would result in the design being less easy to migrate to other devices and technologies.

5 Conclusions

To conclude, this paper describes high performance single-chip FPGA implementations of the Rijndael algorithm. The generic, parameterisable encryption design is the only hardware Rijndael encryption design that supports varying key sizes, reported to date. When implemented, the 128-bit key encryption design performs at a data-rate of 7 Gbits/sec, which is 3.5 times faster than similar existing FPGA implementations and 21 times faster than software implementations. Previous Rijndael encryption-only designs are implemented on Virtex XCV1000 devices, which consist of only 32 BRAMs and therefore, cannot support a fully pipelined Rijndael design. The Virtex-E and Virtex-E Extended Memory family of FPGAs, however, contains up to 280 BRAMs and can easily accommodate large unrolled designs. The encryptor/decryptor core runs at 3.2 Gbits/sec. This implementation not only compares favorably with similar ASIC designs but is also the only known single-chip FPGA Rijndael design capable of both encryption and decryption. Future work will include parameterising the Rijndael encryption/decryption design so as it may also accept varying key sizes. Rijndael is set to be approved by NIST and replace DES as the Federal Information Processing Encryption Standard (FIPS) in the summer of 2001. It will replace DES in applications such as IPSec protocols, the Secure Socket Layer (SSL) protocol and in ATM cell encryption. In general, hardware implementations of encryption algorithms and their associated key schedules are physically secure, as they cannot easily be modified by an outside attacker. Also, the high speed Rijndael encryptor core and Rijndael encryptor/decryptor core presented, should prove beneficial in applications where speed is vital as with real-time communications such as satellite communications and electronic financial transactions.

Acknowledgements

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

References

1. J. Daemen, V.Rijmen: The Rijndael Block Cipher: AES Proposal : First AES Candidate Conference (AES1) : August 20-22, 1998
2. 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.
3. T. Ichikawa, T. Kasuya, M. Matsui: Hardware Evaluation of the AES Finalists: The Third Advanced Encryption Standard (AES3) Candidate Conference, 13-14 April 2000, New York, USA.
4. 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, USA.
5. Xilinx Virtex^{TM}-E 1.8V Field Programmable Gate Arrays: URL: http://www.xilinx.com: November 2000.
6. Brian Gladman: The AES Algorithm (Rijndael) in C and C++: URL: http://fp.gladman.plus.com/cryptography_technology/rijndael/index.htm: April 2001.
7. 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.
8. M.McLoone, J.V. McCanny: Apparatus for Selectably Encrypting and Decrypting Data: UK Patent Application No. 0107592.8: Filed March 2001.
9. B. Weeks, M. Bean, T. Rozylowicz, C. Ficke: Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms: The Third Advanced Encryption Standard (AES3) Candidate Conference, 13-14 April 2000, New York, USA.
10. J.C.A Van Der Lubbe: Basic Methods of Cryptography: Cambridge University Press, 1998
11. A.Menezes, P. Oorschot, S. Vanstone: Handbook of Applied Cryptography: CRC Press, 1997