Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm

Máire McLoone , John V. McCanny

DSiP^{TM} 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. A single-chip FPGA implementation of the new Advanced Encryption Standard (AES) algorithm, Rijndael is presented. 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. The FPGA implementation described here is that of a fully pipelined single-chip Rijndael design which runs at a data rate of 7 Gbits/sec on a Xilinx Virtex-E XCV812E-8-BG560 FPGA device. This proves to be one of the fastest single-chip FPGA Rijndael implementations currently available. The high Block RAM content of the Virtex-E device is exploited in the design.

Keywords: FPGA Implementation, AES, Rijndael, Encryption

1 Introduction

On the 2nd October 2000 the US National Institute of Standards and Technology (NIST) selected the Rijndael algorithm [1], developed by Joan Daemen and Vincent Rijmen, as the new Advanced Encryption Standard (AES) algorithm. It proved to be a fast and efficient algorithm when implemented in both hardware and software across a range of platforms. Rijndael is to be approved by the NIST and replace the aging Data Encryption Standard (DES) algorithm as the Federal Information Processing Encryption Standard (FIPS)[2] in the summer of 2001. In the future Rijndael will be the encryption algorithm used in many applications such as:

- Internet Routers
- Remote Access Servers
- High Speed ATM/Ethernet Switching
- Satellite Communications
- High Speed Secure ISP Servers
- Virtual Private Networks (VPNs)
- SONET
- Mobile phone applications
- Electronic Financial Transactions

In this paper a single-chip FPGA implementation of the Rijndael algorithm is presented. The fully pipelined design is implemented using Xilinx Foundation Series 3.1i software on the Virtex-E XCV812E FPGA device [3]. A 10-stage pipelined

Rijndael design requires considerable memory; hence, its implementation is ideally suited to the Virtex-E Extended Memory range of FPGAs, which contain devices with up to 280 RAM Blocks (BRAMs).

The fastest known Rijndael FPGA implementation is by Chodowiec, Khuon and Gaj [4], which performs at 12160 Mbits/sec. Their design is implemented on 3 Virtex XCV1000 FPGA devices. Dandalis, Prasanna and Rolim [5] also carried out an implementation on the XCV1000 device, achieving an encryption rate of 353 Mbits/sec. A partially unrolled design by Elbirt, Yip, Chetwynd and Paar [6] on the same device performed at a data-rate of 1937.9 Mbits/sec. The fastest Rijndael software implementation is Brian Gladman's [7] 325 Mbit/sec design on a 933 MHz Pentium III processor. Whereas, an earlier paper [8] described a high-speed generic Rijndael design, which supported three key lengths, this paper assumes a 128-bit data block and a 128-bit key and concentrates on using an optimum number of Block RAMs to attain high throughput.

Section 2 of this paper describes the Rijndael Algorithm. The design of the fully pipelined Rijndael implementation and the exploitation of the Block RAMs on the Virtex E device are outlined in Section 3. Performance results are given in section 4 and conclusions are provided in section 5.

2 Rijndael Algorithm

The Rijndael algorithm is a substitution-linear transformation network [9]. It can operate on 128-bit, 192-bit and 256-bit data and key blocks. 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 data block sizes will not be included in the actual FIPS standard. An outline of Rijndael is shown in Fig. 2.1.

Rijndael comprises 10, 12 and 14 iterations or rounds 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 (known as the *State* array), as shown in Fig 2.2. 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 2.3. This paper assumes a 128-bit key and therefore a similar rectangular array is considered for the key as for the data block. The algorithm design consists of an initial data/key addition, nine rounds 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. The Rijndael round comprises four transformations:

- ByteSub Transformation
- ShiftRow Transformation
- MixColumn Transformation
- 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’.

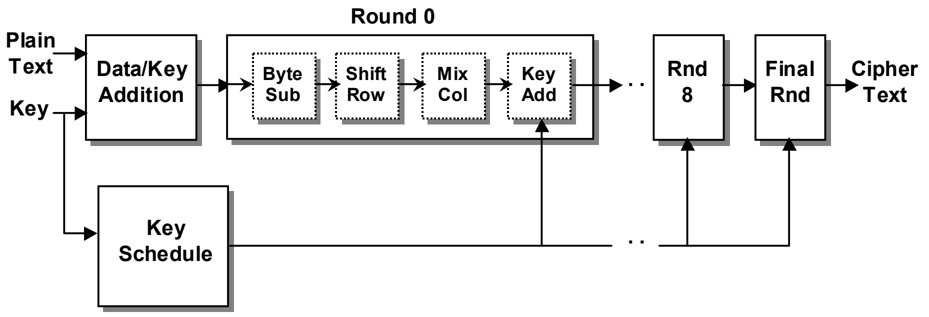

Fig.2.1. Outline of Rijndael Encryption Algorithm

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

Fig. 2.2. 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. 2.3. Key Rectangular Array

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,

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

The Round keys are derived from the cipher key and are also represented as an array of 4-byte vectors. Each round key is bitwise XORed to the State in Round Key Addition. In the final round the MixColumn transformation is excluded.

2.1 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. 2.4. 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.

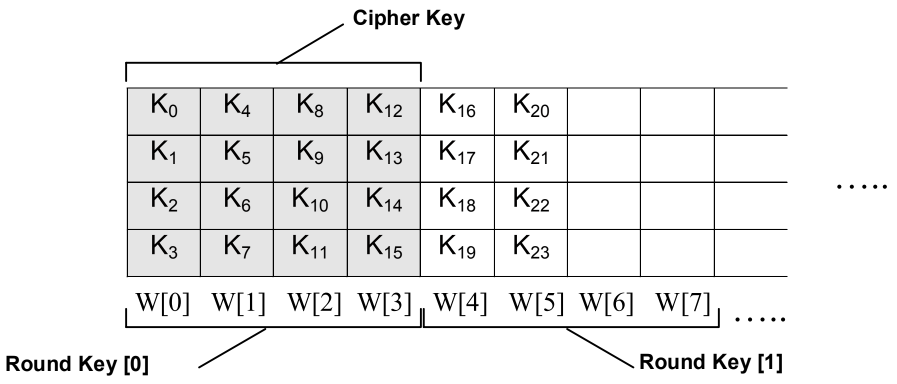

Fig. 2.4. Expanded Key Array with $N_k = 4$

The round constants required for each of the ten rounds are described in [1]. 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. 2.4 above.

2.2 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,

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

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. Round key 0 is still utilized in the initial data/key addition and round key 10 in the inverse of the final round. However, round key 1 is now used in the inverse of round 8, round key 2 in the inverse of round 7 and so on.

3 Design of Pipelined Rijndael Implementation

The Rijndael design described in this paper is fully pipelined, with ten pipeline stages. A number of different architectures can be considered when designing encryption algorithms [6]. 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 main consideration in this design is the memory requirement. The Rijndael s-box - the ByteSub transformation – is utilised in each round and also in the key schedule. This transformation can be implemented as a look-up table (LUT) or ROM. This will prove 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 alone requires sixteen 8-bit to 8-bit LUTs – a total of 160 LUTs. The Virtex-E Extended Memory range of FPGAs is utilized for implementation as it contains devices with up to 280 BlockSelectRAM (BRAM) memories. An outline of the Virtex-E architecture is provided in Fig 3.1 [3].

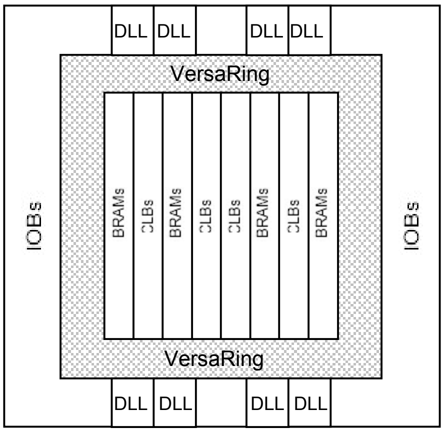

Fig. 3.1. Virtex-E Architecture Overview

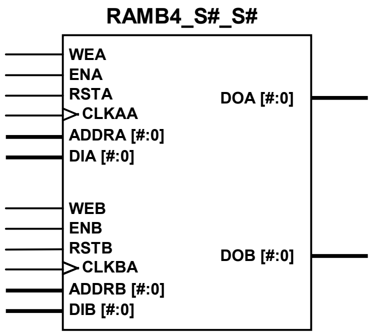

Fig. 3.2. Dual-port Block SelectRAM

A single BRAM can be configured into two single port 256 x 8-bit RAMs, as illustrated in Fig. 3.2 [3]; hence, eight BRAMs are required for 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.

In the key schedule, forty words are created during expansion of the key and every fourth word is passed through the ByteSub transformation with each byte in the word being transformed. Hence, forty 8-bit to 8-bit LUTs or twenty BRAMs can be utilised in its implementation. However, since the round keys are constructed in parallel to the Round operations, only two BRAMs are required. The s-box is only used in the construction of the first word of every Round key (a Round key comprises four words) and each BRAM is used in the construction of two bytes of a word. Therefore an iterative process is used to access the two BRAMs and the Round keys are constructed as they are required by each Rijndael Round. The design assumes that the same key is used in any one data transfer session. The construction of every fourth word, which incorporates the BRAM, is shown in Fig 3.3. As described in Section 2, words which are not a multiple of four are created by XORing the previous word with the word four positions earlier.

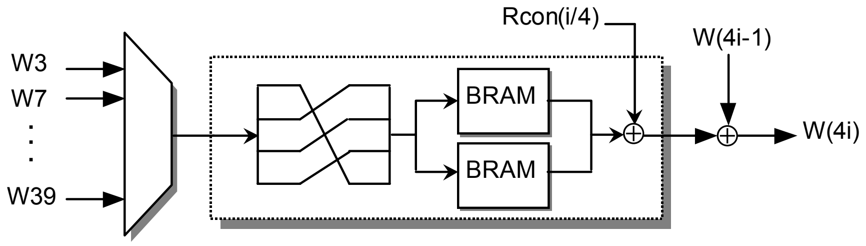

Fig. 3.3. Construction of every Fourth Word in Rijndael Key Schedule

Therefore, in the fully pipelined Rijndael design, a total of 82 ROMs are utilised – 80 ROMs are required for the 10 rounds and a further 2 for the key schedule.

The same design methodology as above can be applied to decryption. The inverse of the ByteSub transformation used in an inverse round operation can also be implemented as a LUT. However, the values in this LUT are different to those required for encryption. During decryption, the values of the LUTs utilized in the key schedule do not change and the round keys are used in reverse order. Therefore, if data decryption is carried out, it is necessary to wait 20 clock cycles before the respective decrypted data appears (10 clock cycles for the construction of the round keys and 10 clock cycles corresponding to the number of rounds in the design). In encryption the latency is only 10 clock cycles. The values contained in the encryption and decryption ROMs are outlined in Appendix 1. Some applications may require a design that can perform both encryption and decryption. One method would involve doubling the number of BRAMs utilized. However, this would prove costly on area. A simple solution [10] is to add two further ROMs to the design outlined here, 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.

4 Performance Results

The outlined Rijndael design is implemented using Xilinx Foundation Series 3.1i software and Synplify Pro V6.0. 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 Rijndael encryptor design, implemented on the Virtex-E XCV812E-8BG560 device, utilizes 2679 CLB slices (28%) and 82 BRAMs (29%). Of IOBs 385 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 to be one of the fastest single-chip Rijndael FPGA implementations currently available, as illustrated in Table 4.1 below. The only faster FPGA implementation is that of Chodowiec, Khuon and Gaj, which has a throughput of 12.16 Gbit/sec. However, this design is implemented over a total of 3 Virtex XCV1000 devices.

Table 4.1. Specifications of Rijndael FPGA Implementations

	Device	**Type**	**Area (CLB Slices)**	**No. of BRAMs**	**Throughput (Mbits/sec)**
Chodowiec, Khuon, Gaj[4] *Over 3 Devices*	XCV1000	P	12600	80	12160
McLoone, McCanny	XCV812E	P	2679	82	6956
Elbirt *et al*[6]	XCV1000	SP	9004	-	1938
Dandalis *et al*[5]	XCV1000	IL	5673	-	353
Gladman [7]	PentiumIII	-	-	-	325

The high performance of the Rijndael design presented is achieved for a number of reasons:

- The design is fully pipelined with data blocks being accepted on every clock cycle.
- The use of dedicated Block RAMs: The complex and slow operations involved in the ByteSub transformation, the multiplicative inverse calculations over GF(2^8) and matrix multiplication and addition, are replaced with simple LUTs.
- The layout of the Virtex-E architecture: From Fig 3.1, it is evident that the BRAMs are located in columns throughout the chip, with each memory column extending the full height of the chip. Each Rijndael round involves

implementation on both CLBs and BRAMs. Therefore, having an architecture where these are located in close vicinity to one another throughout the chip will improve overall performance.

The Rijndael decryptor design is also implemented on the XCV812E-8BG560 FPGA device. It utilises 4304 slices (45%) and 82 BRAMs. It performs at a rate of 6.38 Gbit/sec using a system clock of 49.9 MHz. The variance in the encryption and decryption performances is due to the different multiplier constants required by each design. In encryption the multiplier constants are simply 0x01, 0x01, 0x02 and 0x03 (hexadecimal) while those used in decryption are 0x0B, 0x0E, 0x09 and 0x0D.

5 Conclusions

A high performance single-chip FPGA implementation of the Rijndael algorithm is described in this paper. The encryptor design performs at a data-rate of 7 Gbits/sec, which is 3.5 times faster than existing single-chip FPGA implementations and 21 times faster than software implementations. The decryptor design also achieves a fast throughout of 6.4 Gbits/sec. The pipelined Rijndael design is well suited to the Virtex-E Extended Memory FPGA, since this device can accommodate the 82 BRAMs required in its implementation. The NIST is set to replace DES as the 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 should prove beneficial in applications where speed is important as with real-time communications such as SONET OC-48 networks and satellite communications.

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. NIST; Advanced Encryption Standard (AES) FIPS Draft Publication; URL: http://csrc.nist.gov/encryption/aes/ : 28 February, 2001
3. Xilinx Virtex^{TM}-E Extended Memory 1.8V Field Programmable Gate Arrays ; URL: http://www.xilinx.com : November 2000.
4. P. Chodowiec, P. Khuon, K. Gaj; Fast Implementations of Secret-Key Block Ciphers Using Mixed Inner- and Outer- Pipelining; FPGA 2001, 11-13 February 2001, California.
5. 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.

6. A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar; An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists; AES3 Conference, 13-14 April 2000, New York, USA.
7. Brian Gladman: The AES Algorithm (Rijndael) in C and C++: URL: http://fp.gladman.plus.com/cryptography_technology/rijndael/index.htm: April 2001.
8. M. McLoone, J. V. McCanny; High Peformance Single-Chip FPGA Rijndael Algorithm Implementations; Cryptographic Hardware and Embedded Systems – CHES 2001;
9. K. NechBarker, Bassham, Burr, Dworkin, Foti, Roback; Report on the Development of the Advanced Encryption Standard (AES); URL: http://csrc.nist.gov/encryption/aes/ : 2 October, 2000.
10. M.McLoone, J.V. McCanny: Apparatus for Selectably Encrypting and Decrypting Data: UK Patent Application No. 0107592.8: Filed March 2001.

Appendix 1

The Hexadecimal values contained in the LUT utilised during encryption are outlined below. For example, an input of '00' (hexadecimal) would return the output, '63', an input of '07' would return the output, 'C5', an input of '08' would return the output, '30' and so on.

	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	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

The Hexadecimal values contained in the LUT utilised during decryption are as outlined below. For example, an input of ‘00’ (hexadecimal) would return the output, ‘52’, an input of ‘07’ would return the output, ‘38’, an input of ‘08’ would return the output, ‘BF and so on.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	D0	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D