Microelectronics Journal 36 (2005) 139–146

Microelectronics Journal

www.elsevier.com/locate/mejo

An FPGA implementation of the AES-Rijndael in OCB/ECB modes of operation

Cristian Chiţu[a,*], Manfred Glesner[b]

[a]*Infineon Technologies Austria AG, Siemensstrasse 2, A-9500 Villach, Austria*
[b]*Institute of Microelectronic Systems, Darmstadt University of Technology, Karlstrasse 15, D-64283 Darmstadt, Germany*

Received 28 January 2004; received in revised form 18 October 2004; accepted 21 October 2004

Abstract

Implementation in one FPGA of the AES-Rijndael in Offset Codebook (OCB) and Electronic Codebook (ECB) modes of operation was developed and experimentally tested using the Insight Development Kit board, based on Xilinx Virtex II XC2V1000-4 device. The circuit was designed to provide simultaneous data privacy and authenticity in applications which require small area such as wireless LANs, cellular phones, and smart cards. The experimental clock frequency was equal to 50 MHz and translates to the throughputs of 493 Mbit/s for block size and key size of 128 bits, respectively. The circuit combines the efficiency of OCB authentication with the high security of Rijndael encryption/decryption algorithms, offering an authenticated encryption/decryption scheme.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: AES; Rijndael; FPGA; Encryption; Decryption

1. Introduction

Cryptography plays a very important role in the security of data information. 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 October 2000, the Rijndael algorithm [1,2] developed by Joan Daemen and Vincent Rijmen was selected as the winner of the AES development race. Rijndael proved to be one of fastest and most efficient algorithms and can be easily implemented on a wide range of platforms.

Offset Codebook (OCB) [3] is a new proposal by Phillip Rogaway at University of California Davis and is a block cipher mode of operation that provides both authenticity and privacy when combined with encryption/decryption algorithms. OCB is contained in the draft NIST Federal Information Processing Standards (FIPS) for the modes of operation for the symmetric key block ciphers and OCB-AES has been implemented in the IEEE wireless LAN standard 802.11i.

Several papers [4–18] dealing with implementation in FPGA of the AES have been published so far. There are no other AES hardware designs including the OCB mode of operation published until now. However, software design exists and are available at Rogaway's home page.

This paper evaluates the AES-Rijndael implementation in OCB/ECB modes of operation from the viewpoint of its hardware mapping into high performance Xilinx FPGA. An FPGA implementation can be easily upgraded to incorporate any protocol changes without the need for expensive and time consuming physical design, fabrication, and testing as required for ASICs. This paper is organized as follows. A brief overview of OCB mode of operation and its basic building blocks is given in Section 2. Section 3 outlines the design of the pipelined OCB-AES implementation. Performance results and the test setup are given in Section 4. Finally, in Section 5, possible future work is described and concluding remarks are made.

* Corresponding author. Tel.: +43 4242 305 6393; fax: +43 4242 305 6045.
E-mail address: cochitu@ieee.org (C. Chiţu).

0026-2692/$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2004.10.012

2. AES-OCB Encapsulation

2.1. AES algorithm

Rijndael is a symmetric block cypher with a variable key size and a variable input/output block size. Our implementation supports only one key size of 128 bits and is limited to the block size of 128 bits, which is the only block size required by AES. Implementing other block sizes, specified in the original, non-standardized description of Rijndael is not justified from the economical point of view, as it would substantially increase circuit area and cost without any substantial gain in the cipher security.

Implementation of the encryption round of Rijndael requires realization of four component operations: *Substitution*, *ShiftRow*, *MixColumn*, and *KeyAddition*. Implementation of the decryption round of Rijndael requires four inverse operations: *InvSubstitution*, *InvShiftRow*, *InvMixColumn*, and *KeyAddition*.

Substitution and *InvSubstitution* operate on each byte of the state using substitution tables.

ShiftRow and *InvShiftRow* change the order of bytes within a 16 byte (128 bit) word. Both transformations involve only changing the order of signals, and therefore can be implemented using routing only.

The *MixColumn* transformation as well as *InvMixColumn* can be expressed as a matrix multiplication in the Galois Field GF(2^8). The *InvMixColumn* transformation has a longer critical path compared to the *MixColumn* transformation, and therefore the entire decryption is more time consuming than encryption.

KeyAddition is a bitwise XOR of two 128 bit words.

2.2. OCB mode of operation

Today, OCB is a draft standard for IEEE 802.11, a norm for wireless LANs. OCB is also at proposal for the NIST in the USA as mode of operation for block ciphers in general. The algorithm is quite simple and elegant and the proof for its security is mathematically well defined. These are reasons why OCB might be popular in the future, considering its ease of implementation and the expansion of wireless technologies. Actually there is no concurrence from any other mode when using big amounts of data.

OCB illustrated in Fig. 1 is a combined encryption and authentication mode which so far survived analysis. OCB assumes a fixed length block cipher (such as AES). In this case the input plaintext is 128 bits, the output ciphertext is 128 bits, and the block cipher requires a 128 bit key.

The setup procedure for OCB encryption is performed as follows. First, an all zero vector is encrypted producing L. Next, the XOR of L and nonce N is encrypted to get the value R. L and R are used to produce the offsets $Z[i]$ for each message block $M[i]$. In the encryption process, the message block $M[i]$ is XOR-ed with a constant derived from L and R, namely $Z[i]$. Then the block is encrypted and the result is XOR-ed with $Z[i]$ to get the ciphertext block $C[i]$. Each message block $M[i]$ is encrypted in the same way (with different offsets $Z[i]$), for $1 \le i \le m-1$, where $M[m]$ is the last block of M. In practice, $Z[i]$ is computed as follows: $Z[1]=L\oplus R$, and, for $i\ge 2$, $Z[i]=Z[i-1]\oplus L(ntz(i))$ If $i\ge 1$ is an integer then $ntz(i)$ is the number of trailing 0-bits in the binary representation of i (equivalently, $ntz(i)$ is the largest integer z such that 2^z divides i).

The encryption scheme described is only good for the first $m-1$ blocks. To deal with the last block $M[m]$, and produce ciphertext that has the same length as the plaintext, OCB does the following.

The length of $M[m]$ is represented with 128 bits, the size of a regular block. The length is XOR-ed with another full-length constant $L[-1]$, then the result is XOR-ed with $Z[m]$. The ciphertext $C[m]$ is calculated as a XOR between the plaintext and $Y[m]$. The last block can be shorter than usual, or full-length. The ciphertext $C[m]$ has the same length as the original message $M[m]$.

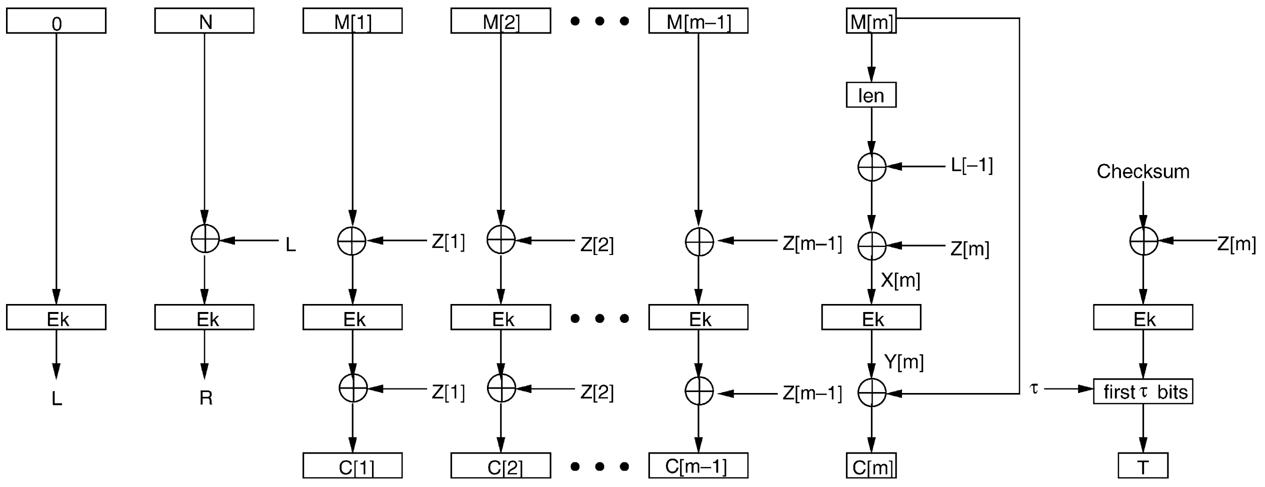


Fig. 1. Illustration of OCB encryption. The message is $M[1]M[2]\ldots M[m-1]M[m]$ and the nonce is N. The resulting ciphertext is $C[1]C[2]\ldots C[m-1]C[m]T$.

We define τ to be the authentication tag length, where $0 \le \tau \le 128$. The number of bits necessary for the tag varies according to the application. To get the *Checksum* we compute $M[1] \oplus M[2] \oplus \ldots \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$. The tag T is the result of encryption of the XOR between the *Checksum* and $Z[m]$. Appending an authentication tag to the ciphertext has several advantages. With this scheme, the size of the tag controls the level of authentication. To verify the signature (authentication tag), the decryptor can recompute the checksum, then recompute the tag, and finally check that is equal to the one that was sent. If the ciphertext passes the test, then OCB produces the plaintext normally.

In review, OCB has the following features:

- Encryption and decryption of arbitrary length messages.
- Single key for confidentiality and authentication.
- Nonce used once, no randomness required.
- Achieves a nearly optimal number of block cipher calls.
- Has provable security.

3. Architecture of the circuit

The architecture proposed for the circuit is based on small area. The organization of the hardware implementation of the circuit is shown in Fig. 2 and includes the following units:

- *EncDecCombined* (Encryption and Decryption Combined), used to encipher and decipher input blocks of data. The controller *EncDecCombinedFSM* has a latency of 11 clocks.
- *KeySchedule* (Key Scheduling), used to compute a set of internal cipher keys based on a single external key. The controller *KeyScheduleFSM* has a latency of 11 clocks.
- *RAMSubKeys* (RAM memory) of internal keys, used to store internal keys computed by the *KeySchedule*, or load the initial key to the FPGA through the Key Entry Interface.
- *Input* (Input Interface), used to load blocks of input data and to store input blocks awaiting encryption/decryption. The controller *InputFSM* has a latency of eight clocks.

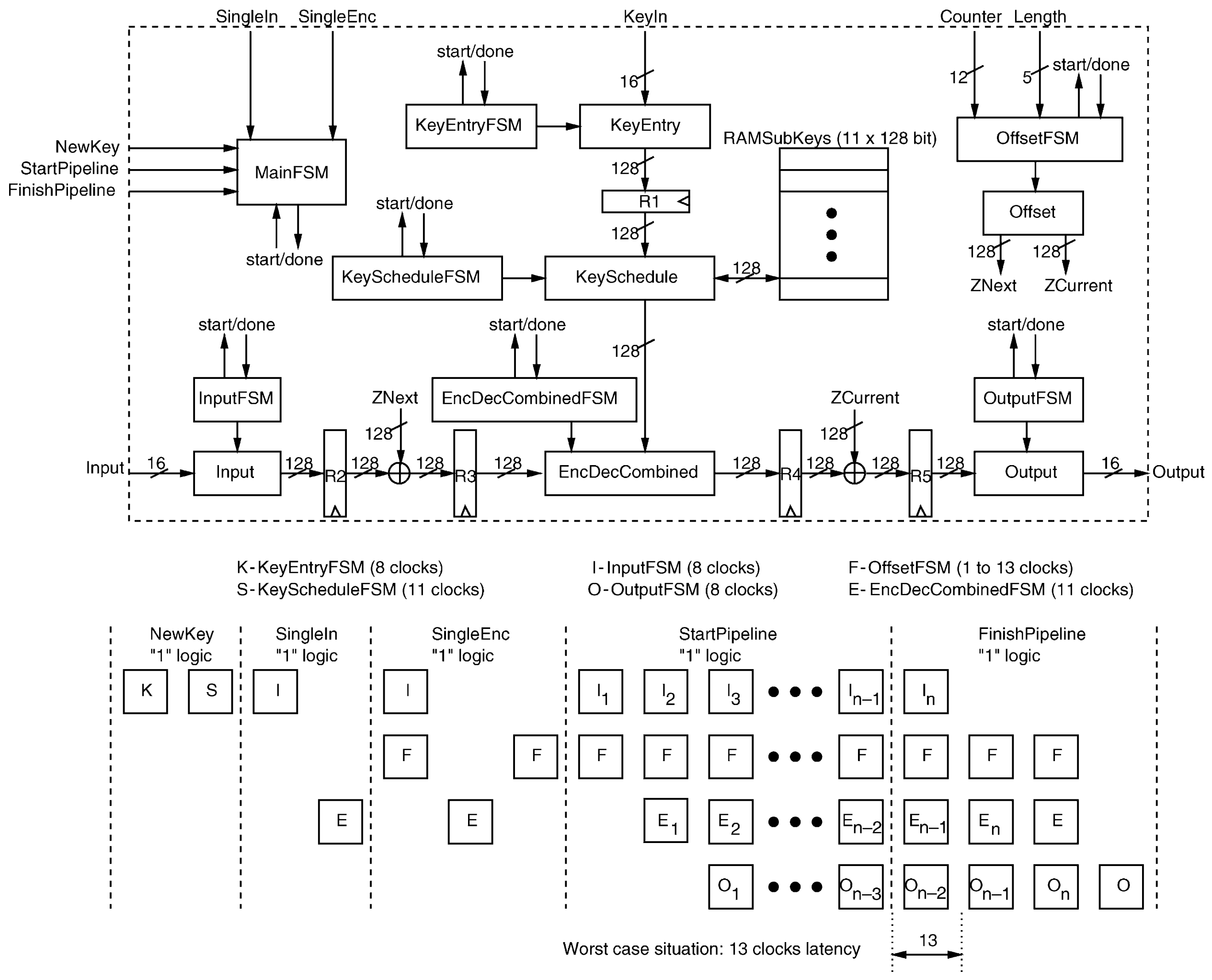


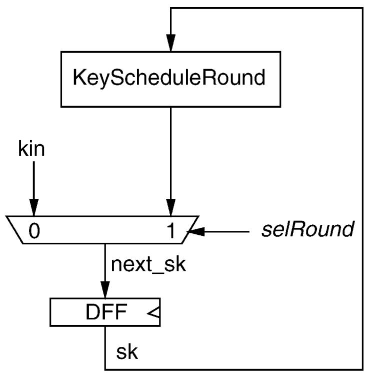
Fig. 2. Upper: architecture of the circuit. Lower: pipelined mode of operation.

- *KeyEntry* (Key Entry Interface), used to load the external key. The controller *KeyEntryFSM* has a latency of eight clocks.
- *Output* (Output Interface), used to temporarily store output from the encryption/decryption unit. The controller *OutputFSM* has a latency of eight clocks.
- *Offset* (Offset Calculation), used to generate the offsets *ZCurrent*(Z[$i-1$]), *ZNext*(Z[i]) necessary for the OCB mode of operation. The *Offset* also calculates the *Checksum* to get the authentication tag T. The latency of the controller *OffsetFSM* varies between 1 and 13 clocks due to the calculation of $L(ntz(i))$, where $1 \leq i \leq 4095$ is the number of packages transmitted in wireless LAN communications.
- *MainFSM* (Main Control Finite State Machine), used to generate control signals for all other units. The top level controller *MainFSM* communicates through handshaking protocols with all other FSMs.

Pipelining is a general method of increasing the amount of data processed by a circuit in a unit of time. The flow of data through the pipeline is shown in Fig. 2. The pipeline is divided in five sequences and the number of iterations is n in this example. In the first sequence, an external key is loaded and the signal *NewKey* is set active. The second and third sequences are used for the initialization phase necessary for the OCB mode of operation. *SingleIn* and *SingleEnc* are active for loading the constants L and R, respectively. During the fourth sequence, *StartPipeline* is active, the data is entered into pipeline, and encryption or decryption is performed. Finally, in the last sequence, all data is processed by setting *FinishPipeline* active. The worst case situation occurs when *OffsetFSM* reaches the maximum latency of 13 clocks which determines the latency of the circuit.

As mentioned in the abstract, we selected 128 bit size for the key as being commonly usable while being more compact than other implementations. Additionally, since fewer rounds are required, it offers greater performance when compared with longer key lengths. On this implementation, *KeyEntry* loads the key and *KeySchedule* produces subkeys of 128 bits. The block diagram of *KeySchedule* is shown in Fig. 3. Initially, the selection signal *selRound* for the multiplexer is set to 0 in a clock cycle for the initial round. For the next clock cycles, *selRound* is set to 1. In a clock cycle, one transformation round is executed and, at the same time, the appropriate subkey for the next round is calculated. The whole process reaches the end when ten rounds of transformation are completed.

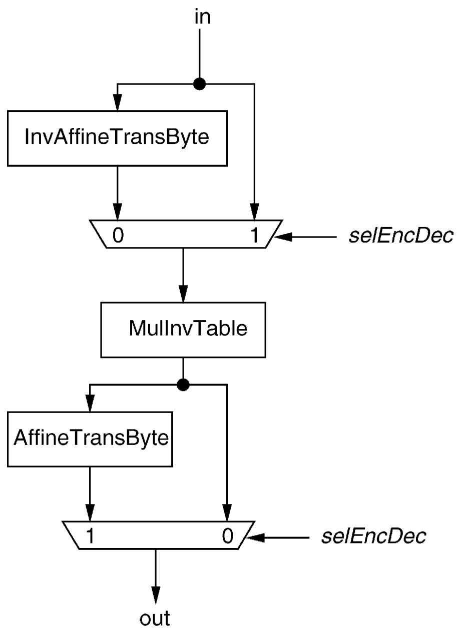
The subkey pieces are passed and stored into the RAM memory *RAMSubKeys*. The memory has 11 locations: 10 locations are for the subkeys and one for the key. It is important to store the initial key since is used in the decryption process. It should be noticed that the subkeys are pre-calculated before encryption/decryption since the circuit contains a RAM memory.



Round	*selRound*
Initial	0
Standard	1
Final	1

Fig. 3. *KeySchedule* block diagram.

Substitution/*InvSubstitution* is usually implemented by look-up tables. Each S-box/Inverse S-box needs a look-up table of 256×8 bit and each round needs 16 S-boxes/Inverse S-boxes, so the area for look-up tables becomes huge. Because the area is critical for the circuit presented, a joint implementation [19,20] of the *SubByte* and *InvSubByte* transformations has been proposed. This implementation requires small area for look-up tables, but has longer delay. The block diagram of the *SubByte* module is shown in Fig. 4. The *SubByte* module is a non-linear byte substitution that acts on every byte of the state to produce a new byte



	selEncDec
Encryption	1
Decryption	0

Fig. 4. *SubByte* module block diagram.

value using a substitution table *MulInvTable*. During encryption, *SubByte* is constructed by composing two transformations [21]:

- First the multiplicative inverse *MulInvTable* in the finite field (with element zero mapped to itself).
- Second the affine transformation *AffineTransByte* over GF(2^8) defined by:

$$b'_i = b_i \oplus b_{(i+4)\text{mod}8} \oplus b_{(i+5)\text{mod}8} \oplus b_{(i+6)\text{mod}8} \oplus b_{(i+7)\text{mod}8} \oplus c_i$$

for $0 \le i \le 8$ where b_i is bit i of the byte and c_i is bit i of a byte c with the value 63 $_h$ or 01100011_b. In matrix form this translates to:

$$\begin{bmatrix} b'_7 \\ b'_6 \\ b'_5 \\ b'_4 \\ b'_3 \\ b'_2 \\ b'_1 \\ b'_0 \end{bmatrix} = \begin{bmatrix} 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \\ 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

During decryption, *SubByte* is constructed by composing two transformations:

- First the inverse affine transformation *InvAffineTransByte* over GF(2^8) defined by:

$$b'_i = b_{(i+2)\text{mod}8} \oplus b_{(i+5)\text{mod}8} \oplus b_{(i+7)\text{mod}8} \oplus d_i$$

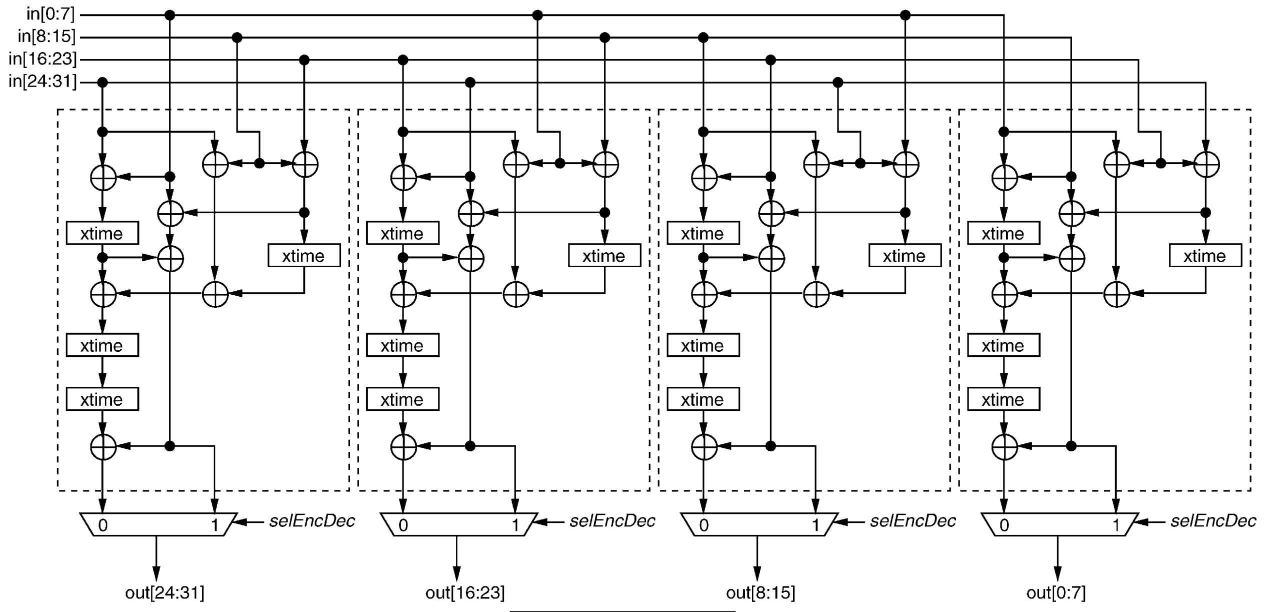
for $0 \le i \le 8$ where b_i is bit i of the byte and d_i is bit i of a byte d with the value 05_h or 00000101_b. In matrix form this translates to:

$$\begin{bmatrix} b'_7 \\ b'_6 \\ b'_5 \\ b'_4 \\ b'_3 \\ b'_2 \\ b'_1 \\ b'_0 \end{bmatrix} = \begin{bmatrix} 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

- Second the multiplicative inverse *MulInvTable* in the finite field (with element zero mapped to itself).

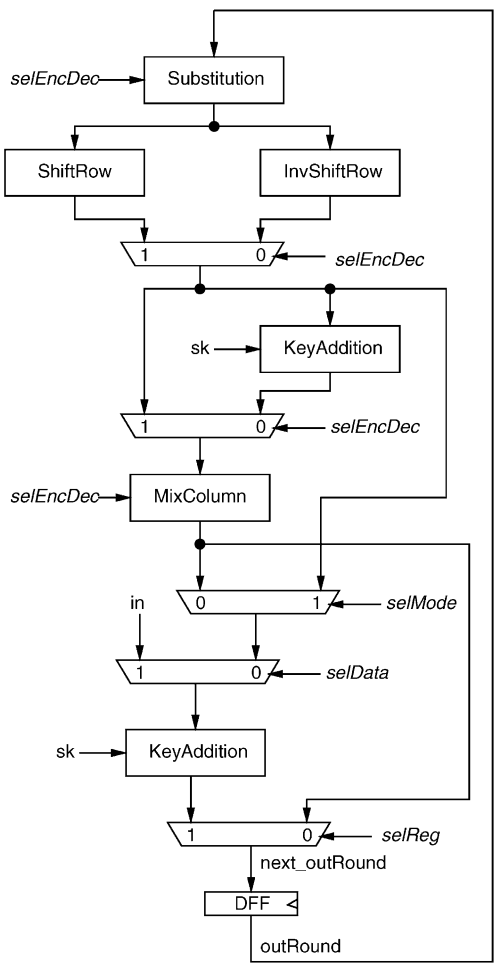
ShiftRow and *InvShiftRow* change the order of bytes within a 16 byte (128 bit) word. Both transformations involve only changing the order of signals, and therefore they can be implemented using routing only, and do not require any logic resources, such as Configurable Logic Blocks (CLBs) or dedicated RAM.

MixColumn/*InvMixColumn* influences usually the cipher area very much. Therefore, we proceeded further with the resource sharing for these blocks. In order to significantly decrease the area of *MixColumn*/*InvMixColumn*, a joint implementation described in detail in the Ref. [19] is proposed in Fig. 5. The four inputs and four outputs



	selEncDec
Encryption	1
Decryption	0

Fig. 5. One quarter of the *MixColumn* block diagram representing one column in the state.



	Round	*selEncDec*	*selData*	*selMode*	*selReg*
Encryption	Initial	1	1	0	1
	Standard	1	0	0	1
	Final	1	0	1	1
Decryption	Initial	0	1	0	1
	Standard	0	0	1	0
	Final	0	0	1	1

Fig. 6. *EncDecCombined* block diagram.

represent single bytes. Four identical blocks like that one shown in Fig. 5, constitute the *MixColumn* block diagram.

KeyAddition is a bitwise XOR of two 128 bit words.

The implementation of the encryption and decryption combined unit is shown in Fig. 6. It requires realization of five component operations: *Substitution*, *ShiftRow*, *InvShiftRow*, *MixColumn*, and *KeyAddition*. The values of the selection signals *selEncDec*, *selData*, *selMode*, and *selReg* for the multiplexers are also described. The architecture shown in Fig. 6 is very compact and is based on the resource sharing for two blocks *Substitution* and *MixColumn* in order to achieve minimum area of the circuit. It has been proven from simulations and further on from implementation that by using the resource sharing of these blocks the area of the circuit is with 16% less. *ShiftRow* and *InvShiftRow* do not require any logic resources in FPGA implementation.

4. Test setup

The Rijndael cipher in OCB/ECB modes of operation was first described in Verilog, and his description verified using the Verilog-XL simulator from Cadence Design Systems. Test vectors from the reference software implementations were used for debugging and verification of Verilog codes. The revised Verilog code became an input to Xilinx ISE Series 4.1 i software performing the logic synthesis, mapping, placing, and routing. In order to fit the whole circuit in one FPGA device Virtex II XC2V1000-4, the option for the Xilinx ISE Series 4.1 i software was set to small area and the design has been flatten. These tools generated reports describing the area and speed of implementation, a netlist used for timing simulations, and a bitstream to be used to program the FPGA device Virtex II XC2V1000-4 [22].

```
Variables in hex for 74 blocks
-----------------------------------------
 Counter=001
    Z[1]=917cf69ebd68b2ec9b9fe9a3eadda692
    M[1]=00000000000000000000000000000000
    C[1]=81d9747326b9ef52f1f10d12b208df9c
-----------------------------------------
 Counter=002
    Z[2]=5cae6137627cea9b8b061d107eb5f0ce
    M[2]=00000000000000000000000000000000
    C[2]=860650ae899c3ab92de65ff4caffebaa
-----------------------------------------
 Counter=003
    Z[3]=3a472ae38df6c6a0034ae749b481dbe0
    M[3]=00000000000000000000000000000000
    C[3]=eaf4f7b6f3aa9d9874cefceaa2233519
-----------------------------------------
 Counter=004
    Z[4]=a1e205b033de764e22790e2e9c5177df
    M[4]=00000000000000000000000000000000
    C[4]=eeac7fd52e028aeb132745faa76c0d22
-----------------------------------------
 Counter=005
    Remaining blocks (bytes)=10
    Z[5]=c70b4e64dc545a75aa35f47756655cf1
   L[-1]=3374a5ea77c5161dc4267d2ce51a1597
    X[5]=f47feb8eab914c686e13895bb37f4936
    Y[5]=e66e766029def4e100abd216167bd219
    M[5]=00000000000000000000
    C[5]=e66e766029def4e100ab
Checksum=00000000000000000000d216167bd219
       T=82a911f68b0a1932abb77b58286f825c
------------------------------------------------------
Test case  OCB-AES-128-74B
Key        00000000000000000000000000000000
Nonce      00000000000000000000000000000000
Plaintext  000000000000   ...   000000000000 [74 bytes]
Ciphertext 81d9747326b9   ...   29def4e100ab [74 bytes]
Tag        82a911f68b0a1932abb77b58286f825c
------------------------------------------------------
```

Fig. 7. Test vectors for OCB mode of operation.

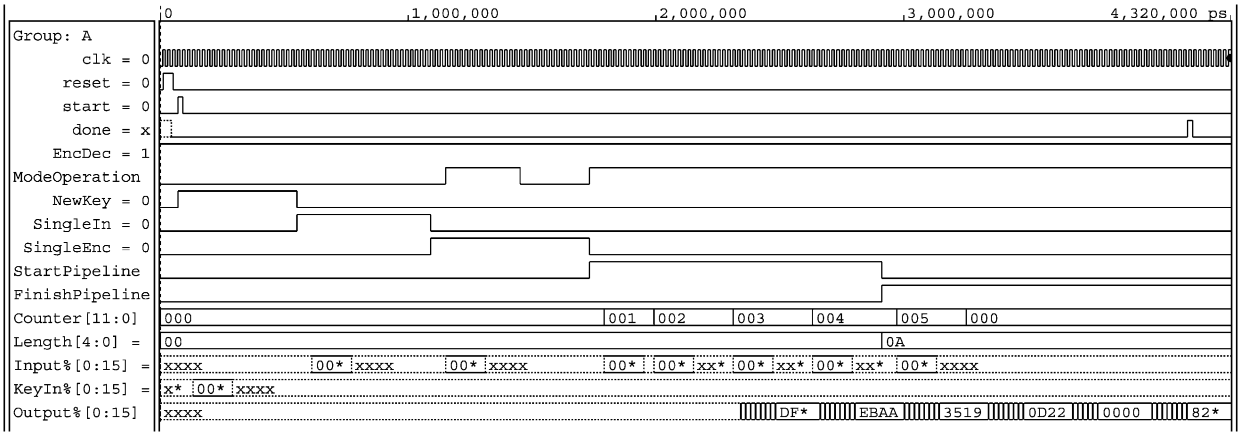


Fig. 8. Timing simulation in Cadence Verilog-XL at 50 MHz clock frequency showing the pipeline with data encrypted.

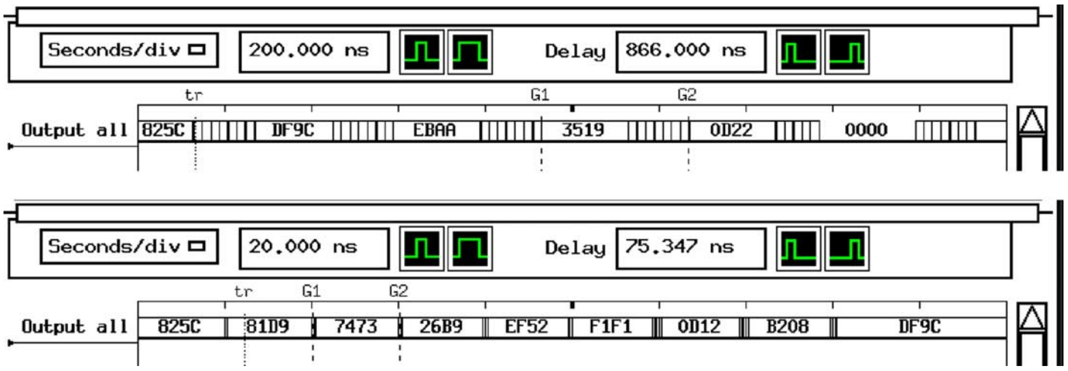


Fig. 9. Experimental testing using Agilent 16702B Logic Analysis System at 50 MHz clock frequency showing the pipeline (upper) and eight clock cycles of $C[1]$ (lower) with data encrypted.

The software [23] used to provide test vectors for OCB mode of operation was written in C and C++ and is available at Rogaway's home page. The test vectors as well as the variables of a pipeline with five inputs ($n=5$ in Fig. 3) containing 74 blocks of data encrypted in OCB are represented in Fig. 7. For simplicity, key, nonce, and plaintext are all set to zero.

The timing simulation results of the test vectors were performed with the key *KeyIn*[0:15] and the input *Input*[0:15] set to zero, as shown in Fig. 8. *ModeOperation* and *EncDec* are '1/0' logic for OCB/ECB and encryption/decryption, respectively. *Length*[4:0] shows the number of remaining blocks of data and *Counter*[11:0] displays the number of packages. The maximum number of packages transmitted in WLANs is 4095 and therefore 12 bits are needed for coding. *NewKey* is '1' logic whenever is desired to load an external key. The reset of the circuit is synchronous through *reset* while *start* and *done* are part of the handshaking protocol.

In order to program the FPGA, a SUN workstation was connected to the Insight Virtex II Development Kit board [24]. The board was connected to the Logic Analysis System Agilent 16702B [25] which provided and displayed signals during measurements. The experimental results are shown in Fig. 9.

The results of the FPGA implementation are summarized in Table 1. The throughput of the circuit in OCB mode is given by:

$$\text{Throughput} = (128\ \text{bits}/13\ \text{clocks})50\ \text{MHz}$$

where 13 clocks is the circuit latency in the worst case situation, as explained in Section 3. In Electronic Codebook (ECB) mode, the throughput is 6400 Mbit/s as a result of

Table 1
Results of FPGA implementation

Target FPGA device	Virtex II XC2V1000-4
Maximum clock frequency	50 MHz
Encryption/decryption throughout OCB mode	493 Mbit/s
Encryption/decryption throughout ECB mode	6400 Mbit/s
Area	
CLB slices	3552
Block ROMs	21
Block RAMs	1
Percentage occupied in device	69%

128 bits multiplied by 50 MHz assuming one encrypted/decrypted block of data exiting the pipeline once the latency has been met.

5. Conclusions

In this paper we have evaluated the Rijndael cipher in OCB/ECB modes of operation from the point of view of its implementation in FPGA. The circuit presented combined the OCB mode with AES-Rijndael algorithm to provide efficient high security functionality for a wide range of operations.

The new architecture presented allows the implementation of the Rijndael cipher in OCB/ECB modes of operation with encryption and decryption. Specific applications for this circuit are in wireless LANs, cellular phones and smart cards. For instance, this circuit can be successfully used for wireless LAN in which the maximum throughput required is of 128 Mbit/s, lower than the throughput obtained from measurements. The experimental procedure demonstrated that the total encryption and decryption throughput of 493 Mbit/s can be achieved using a single FPGA device. Only up to 69% of resources of this single FPGA are required by all cryptographic modules.

The OCB mode of operation is considered secure for transmission of large volumes of data.

References

[1] National Institute of Standards and Technology, Advanced Encryption Standard AES, Federal Information Processing Standards Publication FIPS 197, 2001, http://csrc.nist.gov/publications/fips.

[2] J. Daemen, V. Rijmen, AES Proposal: Rijndael 1999 http://csrc.nist.gov/encryption/aes/rijndael.

[3] P. Rogaway, M. Bellare, J. Black, T. Krovetz, OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption, Eighth ACM Conference on Computer and Communications Security ACM CCS, ACM Press, 2001. pp. 196–205, http://www.cs.ucdavis.edu/rogaway.

[4] P. Chodowiec, K. Gaj, Very Compact FPGA Implementation of the AES Algorithm, Workshop on Cryptographic Hardware and Embedded Systems CHES 2003, Cologne, Germany, 2003 pp. 319–333.

[5] F.X. Standaert, G. Rouvroy, J.J. Quisquater, J.D. Legat, Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs, Workshop on Cryptographic Hardware and Embedded Systems CHES 2003, Cologne, Germany, 2003 pp. 334–350.

[6] K.U. Järvinen, M.T. Tommiska, J.O. Skyttä, A Fully Pipelined Memoryless 17.8 Gbps AES-128 Encryptor, International Symposium on Field-Programmable Gate Arrays FPGA 2003, Monterey, CA, 2003.

[7] N. Sklavos, O. Koufopavlou, Architecture and VLSI Implementations of the AES-Proposal Rijndael, IEEE Transactions on Computers 51 (12) (2002) 1454–1459.

[8] C. Chiţu, D. Chien, C. Chien, I. Verbauwhede, F. Chang, A Hardware Implementation in FPGA of the Rijndael Algorithm, IEEE International Midwest Symposium on Circuits and Systems MWSCAS 2002, Tulsa, OK, vol. 1, 2002 pp. 507–510.

[9] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, An FPGA-Based Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists, IEEE Transactions on VLSI Systems 9 (4) (2001).

[10] M. McLoone, J.V. McCanny, Rijndael FPGA Implementation Utilizing Look-Up Tables, IEEE Workshop on Signal Processing Systems 2001; 349–360.

[11] M. McLoone, J.V. McCanny, Single-Chip FPGA Implementation of the Advanced Encryption Standard Algorithm, Field-Programmable Logic and Applications FPL 2001, Belfast, Northern Ireland, UK, 2001.

[12] M. McLoone, J.V. McCanny, High Performance Single-Chip FPGA Rijndael Algorithm Implementation, Workshop on Cryptographic Hardware and Embedded Systems CHES 2001, Paris, France, 2001 pp. 65–76.

[13] V. Fischer, M. Drutarovsky, Two Methods of Rijndael Implementation in Reconfigurable Hardware, Workshop on Cryptographic Hardware and Embedded Systems CHES 2001, Paris, France, 2001 pp. 77–92.

[14] A. Dandalis, V.K. Prasanna, J.D. Rolim, A Comparative Study of Performance of AES Final Candidates Using FPGAs, Workshop on Cryptographic Hardware and Embedded Systems CHES 2000, Worchester, USA, 2000.

[15] A.J. Elbirt, W. Yip, B. Chetwynd, C. Paar, An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists, Third Advanced Encryption Standard Candidate Conference AES3, New York, USA, 2000.

[16] K. Gaj, P. Chodowiec, Comparison of the Hardware Performance of the AES Candidates using Reconfigurable Hardware, Third Advanced Encryption Standard Candidate Conference AES3, New York, USA, 2000.

[17] T. Ichikawa, T. Matsui, Hardware Evaluation of the AES Finalists, Third Advanced Encryption Standard Candidate Conference AES3, New York, USA, 2000.

[18] K. Gaj, P. Chodowiec, Hardware Performance of the AES Finalists-Survey and Analysis Results, Technical Report, George Mason University, 2000, http://ece.gmu.edu/crypto/AES_survey.pdf.

[19] X. Zhang, K.K. Parhi, Implementation Approaches for the Advanced Encryption Standard Algorithm, IEEE Circuits and Systems Magazine 2 (4) (2002) 25–46.

[20] J. Goodman, I. Verbauwhede, Architectures and Design Methods for Cryptography, IEEE International Solid-State Circuits Conference ISSCC 2002 Tutorial, San Francisco, USA, 2002.

[21] B. Gladman, A Specification for Rijndael, the AES Algorithm, v3.2, 2001, http://fp.gladman.plus.com.

[22] Xilinx, Inc., Virtex II Field Programmable Gate Arrays, http://www.xilinx.com.

[23] P. Rogaway, Reference C Code, http://www.cs.ucdavis.edu/~rogaway.

[24] Insight Electronics, Virtex II Development Kit, San Diego, CA, 2002, http://www.insight-electronics.com/virtexII.

[25] Agilent Technologies, Inc., Training Kit for the Agilent Technologies 16700-Series Logic Analysis System, Making Basic Measurements, 2001, http://www.agilent.com.