Military Technical College
Kobry El-kobbah,
Cairo, Egypt

5th International Conference
on Electrical Engineering
ICEENG 2006

DESIGN OF PIPELINED AES ENCRYPTION ALGORITHM USING FPGA

Alaa[2]El Din Rohiem , KamelMohamed Hassan , Ahmed[3]M. El-Amin

ABSTRACT:

In this paper, we present developed design procedures for a pipelined Advanced Encryption Standard [AES] encryption algorithm using Field Programmable Gate Array [FPGA].The design procedures starting from entering the design parameters until functional simulation and testing have been introduced in this paper. System throughput of 1.408Gbps has been achieved, whereas the published results for similar systems are much less than this rate [4-7].

KEY WORDS:

FPGA, AES, VHDL, encryption, decryption.

1- INTRODUCTION:

The main factor that this paper is concerned with is to increase the throughput of the design, in another words is to decrease the timing delay between entering two successive inputs and this problem occurs because the design of the AES algorithm depends on the number of rounds in the algorithm and that the data must pass with at least 10 rounds during encryption operation [1] So in the ordinary case we won't be able to enter another data input except after at least 10 rounds which will lead to decrease the throughput of the design so In this paper, we introduce a new design technique that enables the user to enter more than one input without the need to wait until the first input has been encrypted.

The both designs of the encryption and decryption modules of the AES algorithm are introduced such that more than one input with different operation (encryption or decryption) may be applied successively to integrate both the encryption and decryption functions on one chip.Table1 introduces some of the previous trials in implementing the AES encryption algorithm.

Table 1 Pervious designs of AES Encryption Algorithm

Architecture	**Process**	**FPGA device**	**Frequency (MHz)**	**Throughput (Mbps)**
SCHA02[8]	Encryption	FPGA/ASIC	NA	640/ 1280
SKLA02	Encryption/Decryption	XCV300 BG432	22	259
CAST03[9]	Encryption without Key expander	Flex EP1F10K30E	NA	157
SUNG01	128, 192 and 256 bit	ASIC	NA	1024
ELB100	Encryption	XCV1000 BG560	14.1	300
GAJ00	Encryption/Decryption	XILINX Virtex	25.9	331
WEEK00[10]	Encryption/Decryption	ASIC Approach	NA	265

NA:Not available

From Table1, it is shown that the maximum achieved throughput 1280 Mbps which is less than what we have achieved using the proposed design.

The design steps will be accomplished by using the well known package of Mentor Graphics which is *FPGA advantage for HDL design version 5.2* [2].

The rest of this paper is organized as follows; section (1) contains the introduction, section (2) presents the hardware pipelined design of AES encryption algorithm, section (3) presents the AES design functional Simulation and section (4) is the conclusion followed by the Appendices.

2- HARDWARE PIPELINED DESIGN OFAES ENCRYPTION ALGORITHM:

In this section we will discuss the architecture of the proposed design and the function of every main block in pipelined AES encryption algorithm design.

2.1 The Design Architecture:

The basic idea of the design is to allow the entrance of two or more consecutive inputs without the need to wait until the complete encryption/decryption full round has been performed. This idea is accomplished by the main-controller module; which at the beginning receives the first 128 bit (the seed key) then passes them to the key-expander block, to begin the key expanding operation. After this operation is finished, the expanded keys are ready and therefore the design is prepared to receive the input data

(through the same port), to begin encryption/decryption operations. Afterwards, the out-selector passes the output data to the output port and an interruption appears indicating that the output data is ready.

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

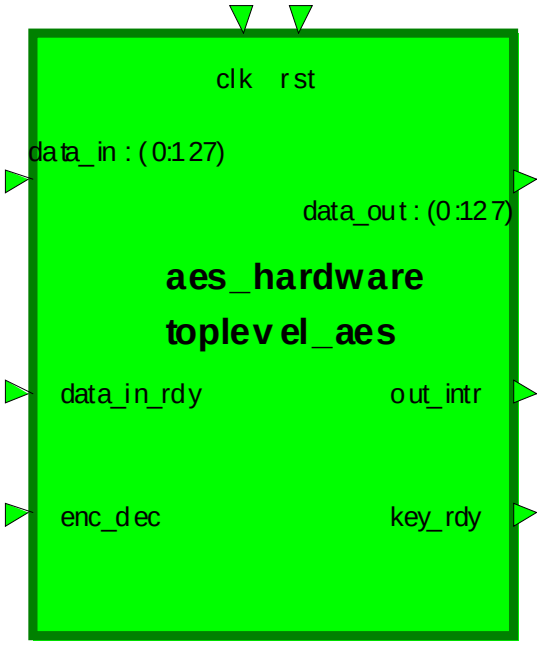


Fig.1shows the toplevel view of the AES algorithm design and the inputs and the

Fig.1 Main Block of Pipelined AES Algorithm design

outputs of the design which will be described later in Table 2.

The internal block diagram of the proposed design will be shown in Fig.5 and the main blocks in this figure are: the AES_encryptor, AES_decryptor, main_controller, key_expander and output_selector.

2.2 The Main_Controller Module:

This module shown in Fig.5 is responsible for receiving the seed key(128 bit) through the data_ip port and passes them to the key_expander through the seed_key port to begin the key expanding operation, then it waits until the key is expanded and then it enables the design to receive the input data through the same port data_ip to begin the encryption\ decryption

operation with the allowance that two or more consecutive inputs can be entered in spite of the operation either (encryption or decryption) in another words the design is enabled to operate in both directions (encryption or decryption).

This module is implemented as a state diagram and the inputs to this module are data_in(128 bit), data_in_rdy(1 bit), enc_dec(1 bit) and key_rdy(1 bit) and the outputs are seed_key(128 bit), key_intr(1 bit), cipher_in(128 bit), decipher_in(128 bit), enc_en(1 bit), dec_en(1 bit), and data_rdy(1 bit).

The inputs of this module are the input to the whole design (Table 2) except key_rdy which indicates the main controller that the expanded keys are ready so that the controller enables the data entrance, and the output ports of the module will be discussed in Table 2.

2.3 The Key_Expander Module:

The main functions of this module are expanding the key and passing the expanded keys to the inputs of the encryption/ decryption units and enabling the main_controller module to begin the data entrance mode by setting key_rdy signal “high” Fig.2 and these functions are achieved through the following blocks: key_controller, rcon_key, s_box_key, and rounds_keys block and the function of every block will be discussed in the following sections.

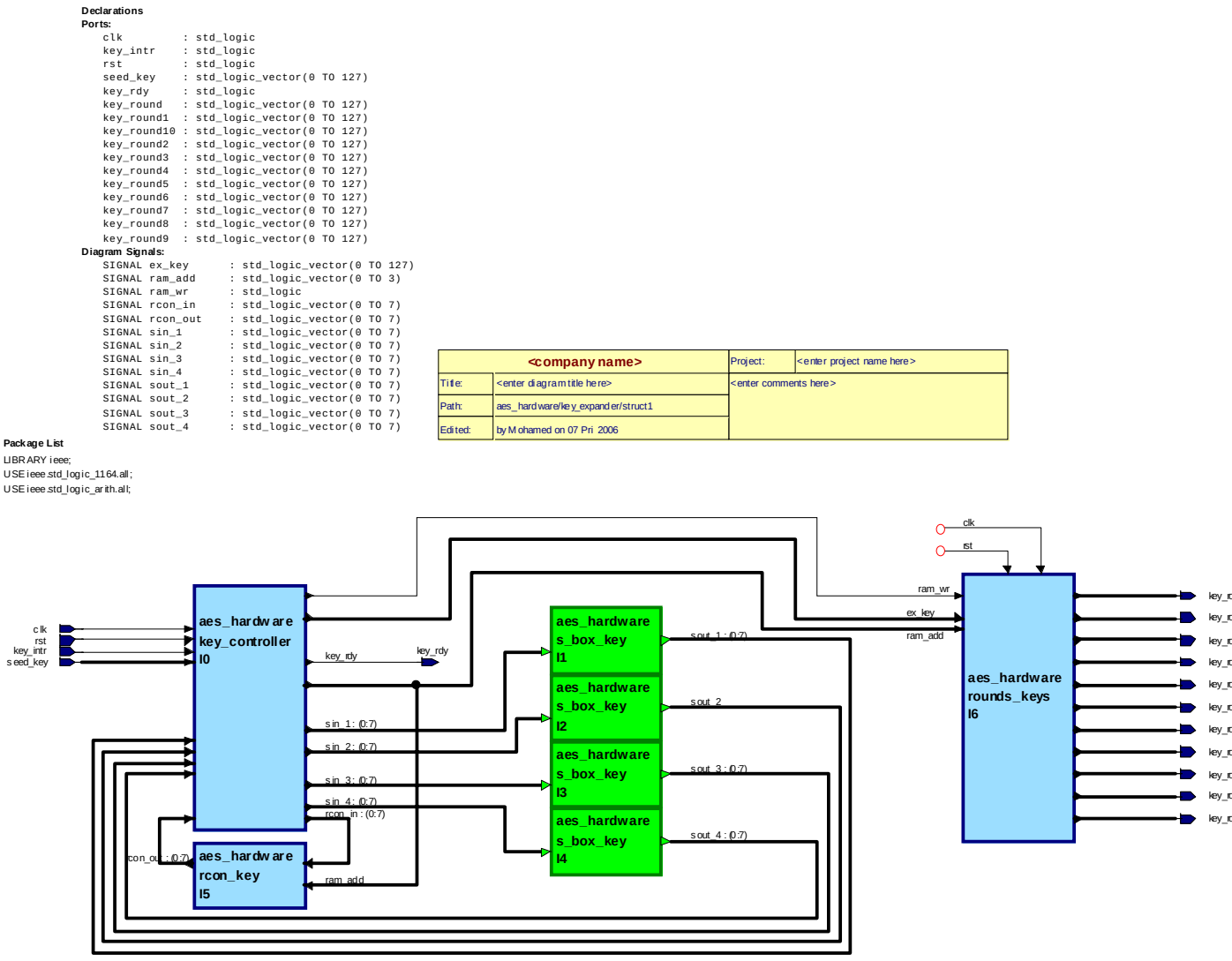


Fig.2 Key_expander Module

2.3.1 The key_controller module:

This is the main block in the key_expander module, it do all the needed operations for expanding the key with the aid of the s_box_key block and rcon_key block as the s_box_key block contain the key S_BOX and the rcon_key block for doing the RCON operations needed for completion of the key expansion.

2.3.2 The rounds_key module:

This block is responsible about monitoring the key_controller block as when it finish the

key expansion it passes the expanded keys to the input of the encryption and decryption units to begin the encryption/ decryption operation.

2.4 The Output_Selector Module:

The main function of this module Fig.5 is to receive the encrypted data or decrypted data and buffers the output data through the output port data_out with an output interrupt to indicate that the output data is ready. The design input ports are: data_rdy, enc_en, dec_en, cipher_out_final, decipher_out_final, clk and rst, and the output ports are: data_out and out_intr.

The output ports are the same as the output ports of the whole design and the function input ports will be discussed in Table 2.The design of this module is introduced as a state diagram.

2.5 AES_Encryptor:

The main function of this module Fig.3 is encrypting the data for only one round of the AES algorithm and the 10 rounds are achieved by repeating this block for 10 times taking into consideration the last special round such that the

Declarations

Ports:

```
Key_in     : std_logic_vector(0 TO 127)
act_mix    : std_logic
cipher_in  : std_logic_vector(0 TO 127)
clk        : std_logic
rst        : std_logic
cipher_out : std_logic_vector(0 TO 127)
```

Diagram Signals:

```
SIGNAL MixCol_OP  : std_logic_vector( 0 TO 31)
SIGNAL MixCol_OP1 : std_logic_vector( 0 TO 31)
SIGNAL MixCol_OP2 : std_logic_vector( 0 TO 31)
SIGNAL MixCol_OP3 : std_logic_vector( 0 TO 31)
SIGNAL sbox_out   : std_logic_vector( 0 TO 7)
SIGNAL sbox_out1  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out10 : std_logic_vector( 0 TO 7)
SIGNAL sbox_out11 : std_logic_vector( 0 TO 7)
SIGNAL sbox_out12 : std_logic_vector( 0 TO 7)
SIGNAL sbox_out13 : std_logic_vector( 0 TO 7)
SIGNAL sbox_out14 : std_logic_vector( 0 TO 7)
SIGNAL sbox_out15 : std_logic_vector( 0 TO 7)
SIGNAL sbox_out2  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out3  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out4  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out5  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out6  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out7  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out8  : std_logic_vector( 0 TO 7)
SIGNAL sbox_out9  : std_logic_vector( 0 TO 7)
```

Package List

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

<company name>		Project:	<enter project name here>
Title:	<enter diagram title here>		<enter comments here>
Path:	aes_hardware/AES_encryptor/struct		
Edited:	by Hesham El Maghraby on 14 Apr 2006		

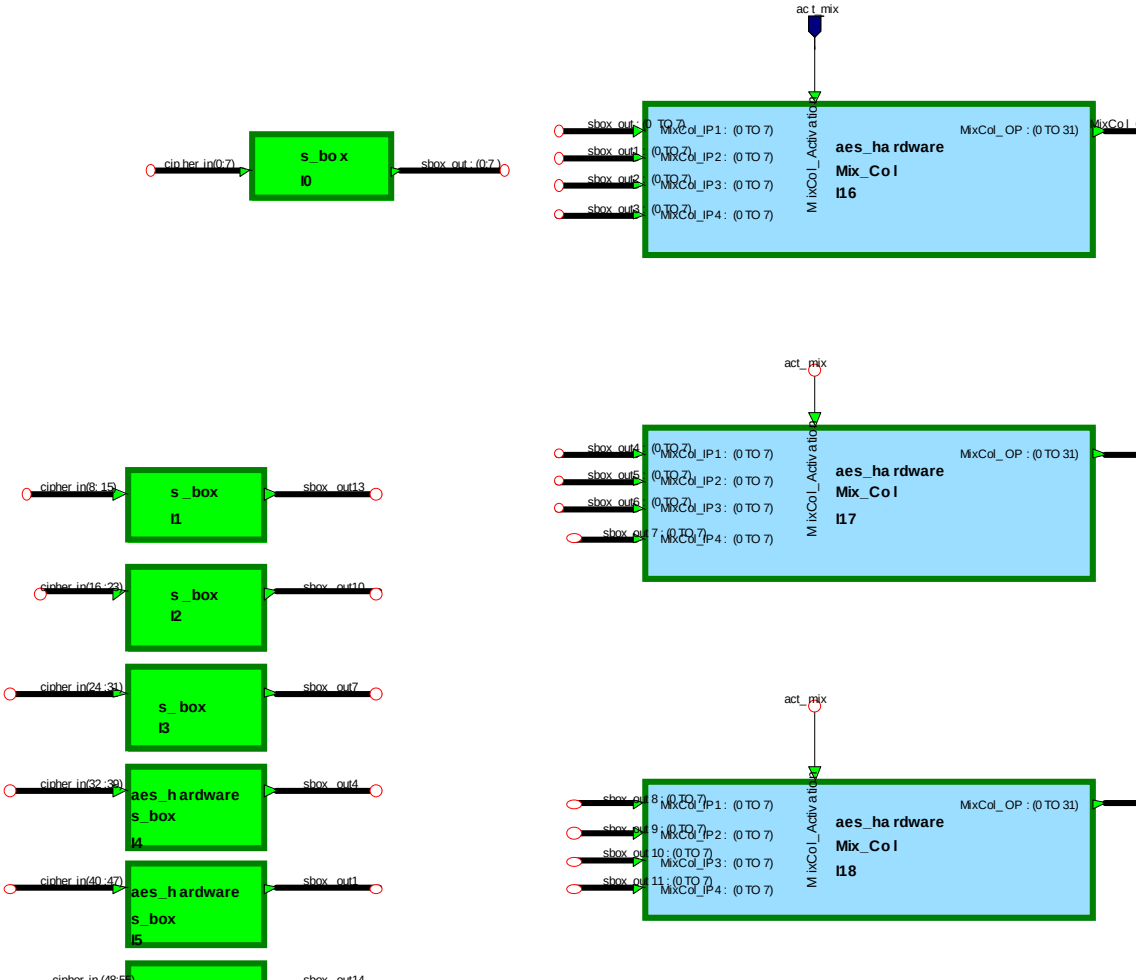


Fig.3 AES_encryptor detailed diagram

Mix_ Col module is deactivated by a signal called act_mix, Fig .3 shows a single module of every block used in the AES_encryptor module, the first block is the s_box module and it is repeated 16 times to every 8 bit of the input and the second block is the Mix_col block and it is repeated 4 times and the third block is the embedded block eb1 and it is only one module in the design.

The main functions of the encryption round are implemented in this module which are: the SBox, shift rows, mix columns, and adding the round key, such that the SBox function is implemented by the s_box block, and the mix of the columns by the Mix_col block and both adding the round key and shifting rows by the embedded block eb1.

2.6 The AES_decryptor module:

The main function of this module Fig.4 is to decrypt the input data for only one round of the AES algorithm and the 10 rounds are achieved by repeating the block for 10 times(as in the encryption operation) taking into consideration the last special round to finish the whole decryption process.

Declarations

Ports:

```
Key_in          : std_logic_vector(0 TO 127)
act_mix         : std_logic
clk             : std_logic
decipher_in     : std_logic_vector(0 TO 127)
rst             : std_logic
decipher_out    : std_logic_vector(0 TO 127)
```

Diagram Signals:

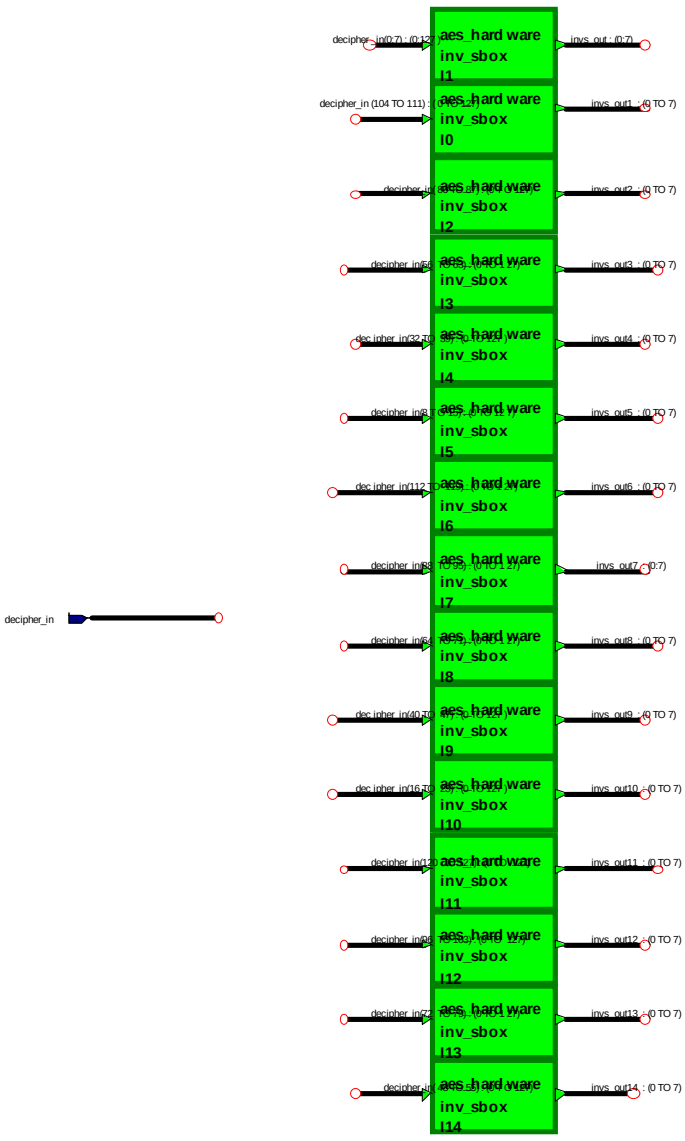
```
SIGNAL Inv_Mic_ColOP1  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP10 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP11 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP12 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP13 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP14 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP15 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP16 : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP2  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP3  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP4  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP5  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP6  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP7  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP8  : std_logic_vector(0 TO 7)
SIGNAL Inv_Mic_ColOP9  : std_logic_vector(0 TO 7)
SIGNAL invs_out        : std_logic_vector(0 TO 7)
SIGNAL invs_out1       : std_logic_vector(0 TO 7)
SIGNAL invs_out10      : std_logic_vector(0 TO 7)
SIGNAL invs_out11      : std_logic_vector(0 TO 7)
SIGNAL invs_out12      : std_logic_vector(0 TO 7)
SIGNAL invs_out13      : std_logic_vector(0 TO 7)
SIGNAL invs_out14      : std_logic_vector(0 TO 7)
SIGNAL invs_out15      : std_logic_vector(0 TO 7)
SIGNAL invs_out2       : std_logic_vector(0 TO 7)
SIGNAL invs_out3       : std_logic_vector(0 TO 7)
SIGNAL invs_out4       : std_logic_vector(0 TO 7)
SIGNAL invs_out5       : std_logic_vector(0 TO 7)
SIGNAL invs_out6       : std_logic_vector(0 TO 7)
SIGNAL invs_out7       : std_logic_vector(0 TO 7)
SIGNAL invs_out8       : std_logic_vector(0 TO 7)
SIGNAL invs_out9       : std_logic_vector(0 TO 7)
SIGNAL xor_out         : std_logic_vector(0 TO 127)
```

Package List

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
```

<co

Title:	<enter dia
Path:	aes_hard
Edited:	by TR C o



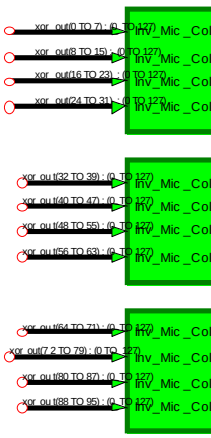


Fig.4 AES_decryptor detailed diagram

Fig .4 shows a single module of every block used in the AES_decryptor module, the first block is the inv_s_box module and is repeated 16 times to every 8 bit of the input and the second block is the embedded block eb1 and is the only one in the AES_decryptor module and the third block is the Inv_Mix_col block and it is repeated 4 times and the fourth block is the act_last_round block and it is only one module in the design.

The block implements the main functions in deciphering process which are Inverse Shift Rows, Inverse SBox, Add round key, and Inverse Mix Columns such that the Inverse shift rows function is implemented by the shifting the inputs to the inv_sbox blocks and the Inverse

SBox function is implemented by the inv_sbox blocks and the add round key function by the embedded block eb1 and the Inverse mix columns function by the Inv_Mix_Col block taking into consideration that the last round is obtained without this function.

3- AES DESIGN FUNCTIONAL SIMULATION:

This step is accomplished by the downstream tool *ModelSim SE 5.7f* in the *FPGA advantage version 5.2* to simulate the design functionality to check if it achieves the same function designed for or not.The testing procedure is discussed in the following section.

3.1 Testing Procedure:

The clock period is 100 ns, the rst of the design is active high.First we enter the key at the input port *data_in* with an interrupt to indicate that the key is ready, and then we wait until the *key_rdy* signal is set high by the design when the expanded keys are generated, Secondly we enter the first input data with the interrupt and the enc_dec signal is set high to enable the encryption circuit, after 3 clocks we enter the second input data with the interrupt and enc_dec signal is set low to enable the decryption circuit, after 7 clocks we enter the third input data with an interrupt and enc_dec signal is set high to enable the encryption circuit, after 14 clocks the first output occurs at the *data_out* port with an interrupt at *out_intr* and after 17 clocks second output occurs at the *data_out* port with an interrupt and after 20 clocks the third output occurs at the *data_out* port with an interrupt, and the inputs and outputs values (testing string) is shown in Table3.

Table (3) the functional simulation testing string

Input no.	**Input data(Hex)**	**Operation**	**Output data(Hex)**
1	00112233445566778899AABBCCDDEEFF	Encryption	69C4E0D86A7B0430D8CDB78070B4
2	69C4E0D86A7B0430D8CDB78070B4C55A	Decryption	00112233445566778899AABBCCDD
3	00112233445566778899AABBCCDDEEF0	Encryption	F00B22FA8ED9C26FBD6A3A691F9
Input Key (constant): 000102030405060708090a0b0c0d0e0f (Hex)			

Fig.6 illustrates the input signals timing and how it is successively entered to the design and Fig.7 shows the output signals and their delay than the input signal.

The throughput of the design is calculated by the following formula:

Throughput = Block size x frequency / (number of clock cycles between two consecutive inputs).

In the proposed design the input needs 3 clock cycle between two successive inputs, the blocksize is 128 bit, and the operating frequency is equal to 33 MHz,

Therefore the throughput = 1.408 Gbps/sec.

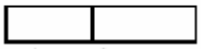
4- CONCLUSION:

The proposed design pipelined AES encryption algorithm improved the system throughput considerably. This achievement is in the expense of increasing the size of the used system.

The future work will focus on reducing the number of used blocks to minimize the system size, to be able to download the proposed design on a single chip and be implemented in different applications.

5-APPENDENCIES:

The appendices in this paper contain the functional simulation of the inputs and outputs of the proposed design as shown in Fig.6 and Fig.7 respectively. Fig. 5 also contains the AES internal design and Table 2 shows all the input and output ports in the proposed design and the function of each port.



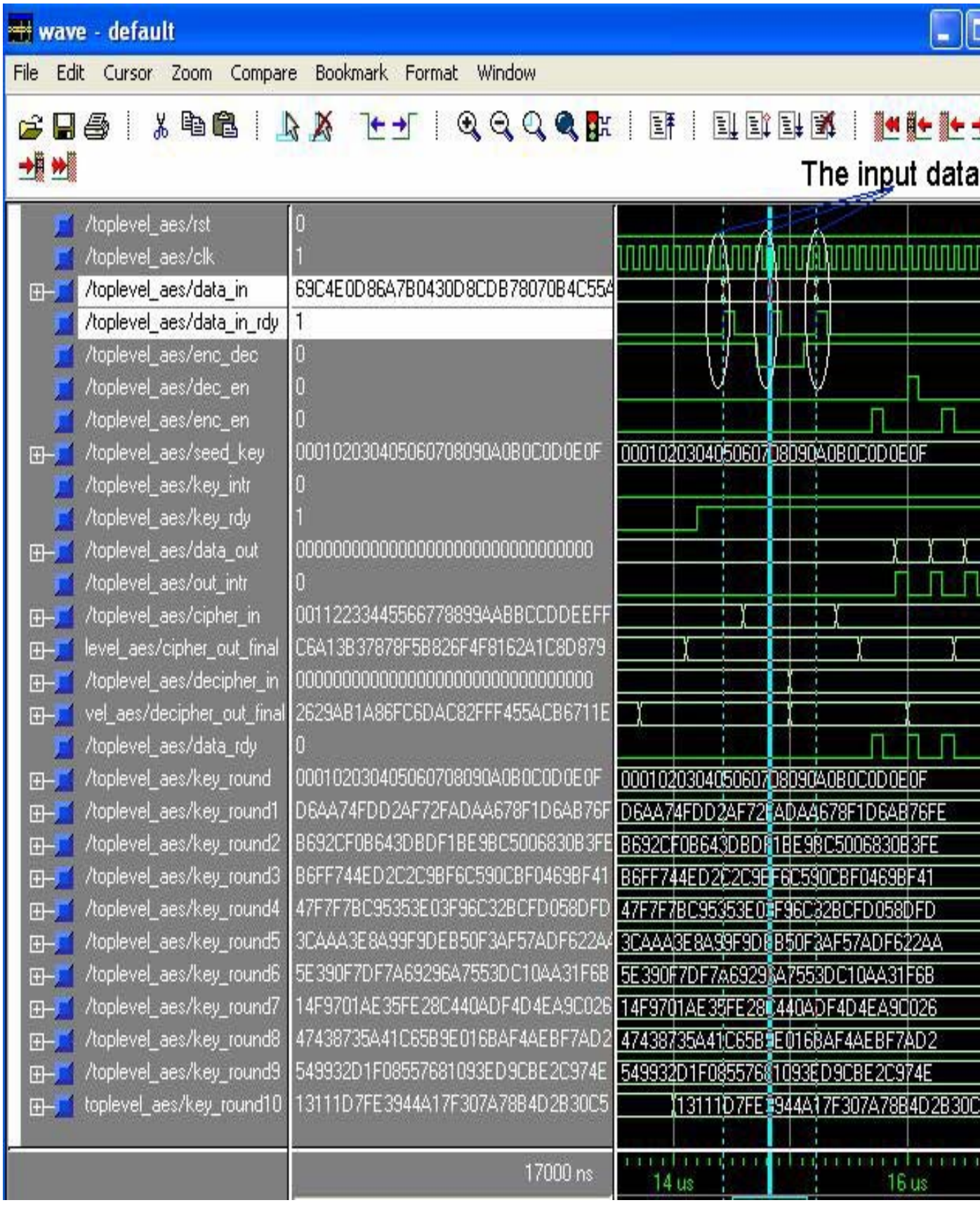
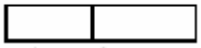

wave - default
File Edit Cursor Zoom Compare Bookmark Format Window
The input data
/toplevel_aes/rst
/toplevel_aes/clk
/toplevel_aes/data_in
/toplevel_aes/data_in_rdy
/toplevel_aes/enc_dec
/toplevel_aes/dec_en
/toplevel_aes/enc_en
/toplevel_aes/seed_key
/toplevel_aes/key_intr
/toplevel_aes/key_rdy
/toplevel_aes/data_out
/toplevel_aes/out_intr
/toplevel_aes/cipher_in
level_aes/cipher_out_final
/toplevel_aes/decipher_in
vel_aes/decipher_out_final
/toplevel_aes/data_rdy
/toplevel_aes/key_round
/toplevel_aes/key_round1
/toplevel_aes/key_round2
/toplevel_aes/key_round3
/toplevel_aes/key_round4
/toplevel_aes/key_round5
/toplevel_aes/key_round6
/toplevel_aes/key_round7
/toplevel_aes/key_round8
/toplevel_aes/key_round9
toplevel_aes/key_round10
0
1
69C4E0D86A7B0430D8CDB78070B4C55A
1
0
0
0
000102030405060708090A0B0C0D0E0F
0
1
00000000000000000000000000000000
0
00112233445566778899AABBCCDDEEFF
C6A13B37878F5B826F4F8162A1C8D879
00000000000000000000000000000000
2629AB1A86FC6DAC82FFF455ACB6711E
0
000102030405060708090A0B0C0D0E0F
D6AA74FDD2AF72FADAA678F1D6AB76FE
B692CF0B643DBDF1BE9BC5006830B3FE
B6FF744ED2C2C9BF6C590CBF0469BF41
47F7F7BC95353E03F96C32BCFD058DFD
3CAAA3E8A99F9DEB50F3AF57ADF622AA
5E390F7DF7A69296A7553DC10AA31F6B
14F9701AE35FE28C440ADF4D4EA9C026
47438735A41C65B9E016BAF4AEBF7AD2
549932D1F08557681093ED9CBE2C974E
13111D7FE3944A17F307A78B4D2B30C5
17000 ns
14 us
16 us

Fig.6 the functional simulation of the design



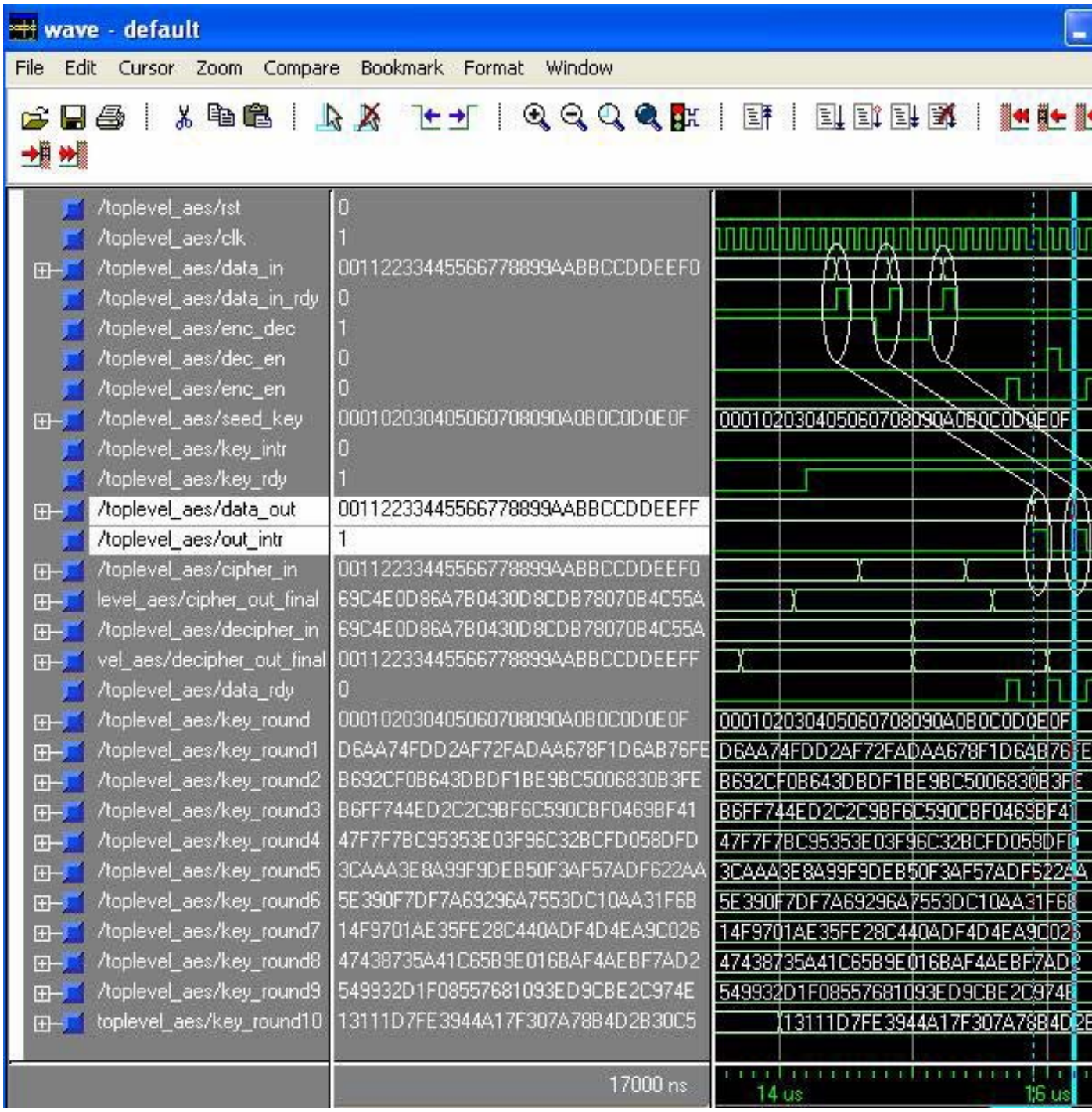
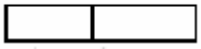


Fig.7 the functional simulation outputs of the design



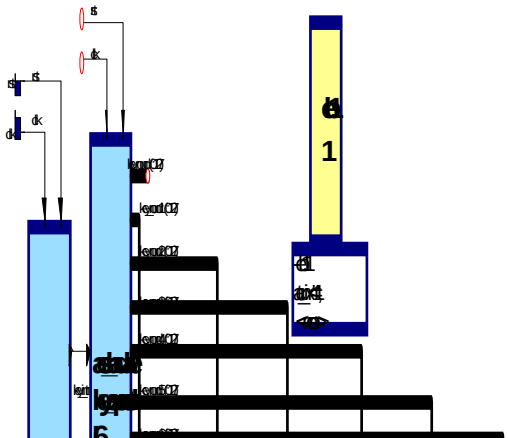


Table2 The design input and output ports

Design Unit	Port Name	Bus Length	Function
The toplevel design ports	rst	1 bit (Active High)	The Design global reset input rst='1' ……..Design is in reset state and no processing occurs. rst='0'…….. Design is ready for processing procedures.
	clk	1 bit	The processing clock of the design and the decision is taken with every rising edge**.**
	enc_dec	1 bit	This port is used for the selection of the mode of operation in the design either Encryption or Decryption , when enc_dec ='0' Encryption enc_dec ='1' Decryption
	data_ip	128 bit	To enter the Key and the data to the Chip respectively.
	data_in_rdy	1 bit	This port informs the design that the input data is ready to enter.
	data_out	128 bit	The output port at which the data is found after processing.
	out_intr	1 bit	The port that indicates that the encryption/decryption operation is completed, out_intr = '0' …….processing mode (data is not ready yet) out_intr = '1'……..data is ready (processing finished)

	key_rdy	1 bit	The port that indicates that the key expansion operation is completed, key_rdy = '0' …….processing mode (key is not expanded yet) key_rdy = '1'……..expanded keys are ready (key expansion operation finished)
	seed_key	128 bit	The port which the seed key enter through to the key_expander block
	key_intr	1 bit	The port which indicates the key_expander module that the seed key is ready at seed_key port
The main_controller ports	cipher_in	128 bit	This is the input port to the encryption unit to begin the encryption operation
	decipher_in	128 bit	This is the input port to the decryption unit to begin the decryption operation.
	enc_en	1 bit	This is the signal that indicates to the out_selector to decide whether the output is from the encryption path.
	dec_en	1 bit	This is the signal that indicates to the out_selector to decide whether the output is from the decryption path.
	data_rdy	1 bit	This is the signal that indicates to the out_selector to decide whether the output is ready or not.
Output_selector ports	cipher_out_final	128 bit	This is the input port at which the data after encryption operation is ready.
	decipher_out_final	128 bit	This is the input port at which the data after decryption operation is ready.

6- REFERENCES:

[1] William Stallings, “Cryptography and Network Security”, 2003

[2] Designing with FPGA Advantage, Mentor Graphic, student workbook, software V5.2, January 2002

[3] Mark Zwolinski, “Digital System Design with VHDL”, 2000

[4] N. Skiavos and 0. Koufopavlou, “Architectures and VLSI Implementations of

the ES-Proposal Rijndael, Implements two different architectures with fixed block and key length", IEEE Transactions on Computers,Vol.51, issue 12, pp. 1463-1472, Dec.2002.

[5] A.J. Elbirt, W. Yip, B. Chetwynd, and C. Paar, "An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists," Proc. Third Advanced Encryption Standard (AES3) Candidate Conf., pp. 13—27, Apr. 2000.

[6] K. Gaj and P. Chodowiec, "Comparison of the Hardware Performance of the AES Candidates Using Reconfigurable Hardware," Proc. Third Advanced Encryption Standard (AES3) Candidate Conf., pp.40-54, Apr. 2000.

[7] N. Sung Kim, R. B. Brown, and T. Mudge, "VLSI Implementation of the Symmetric Key Block Cipher with the Advanced Encryption Standard-Rijndael", University of Michigan, Oct. 2001.

[8] Patrick R. Schaumont, Henry Kuo, Ingrid M. Verbauwhede, " Unlocking the Design Secrets of a 2.29 Gb/s Rijndael Processor ", Proc.39th Design Automation Conf. (DAC2002), pp. 634-639, June 10-14 2002

[9] CAST, Advance Encryption Standard Megafunction 2003, http://www.cast.com

[10] B. Weeks M. Bean, T. Rozylowicz, and C. Ficke, " Hardware Performance Simulations of Rounds 2 Advanced Encryption Standard Algorithms ," Proc. Third Advanced Encryption Standard (AES3) Candidate Conf. , pp. 286-304 , Apr. 2000.