OPTIMIZATION OF ADVANCED ENCRYPTION STANDARD (AES) IN FPGA IMPLEMENTATION USING S-BOX INTEGRATION

By

Lee Yi Lin (1520)

Dissertation submitted in partial fulfillment of
The requirement for the
Bachelor of Engineering (Hons)
(Electrical and Electronics Engineering)

MAY 2004

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATE OF APPROVAL

Optimization of Advanced Encryption Standard (AES) in FPGA Implementation using S-Box Integration

by

Lee Yi Lin

Dissertation submitted to the
Electrical and Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfillment of the requirement for the
BACHELOR OF ENGINEERING (Hon)
(ELECTRICAL AND ELECTRONICS ENGINEERING)

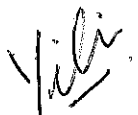
Approved by,

(NOOHUL BASHEER ZAIN ALI)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK
June 2004

CERTIFICATE OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein has not been undertaken or done by unspecified sources or persons.



LEE YI LIN

ACKNOWLEDGEMENT

This project would not have been possible without the help of a number of people, and the author would like to express his heartfelt gratitude to all of them.

First of all, the author would like to express his foremost gratitude to his project supervisor, Mr. Noohul Basheer Zain Ali, for his endless guidance and continuous monitoring throughout the two semester's of final year project. His comments, suggestions, and advice were given serious consideration and were invaluable in determining the final output of this project.

The author is greatly indebted to his parents, Lee Sin Fatt and Chew Lai Chan, for their never-ending support and encouragement. The author would also like to thank his lovely family members, for their love and care which continue to support him through the difficult times

Thanks to Mr. Rudolf Usselmann from asics.ws, for sharing his hard work of the AES IP cores on the webpage of Open Cores.

Last but not least, the author would like to thank his friends and everyone who provided many useful assistance and support to help him in completing his project successfully.

ABSTRACT

Cryptography has a significant role in the security of data transmission. The algorithm of Rijndael was selected and adopted by National Institute of Standards and Technology (NIST) U.S. as Advanced Encryption Standard (AES) in October 2000, in order to replace the old Data Encryption Standard (DES).

As compared to software, hardware implementations provide more physical security as well as faster speed. Thus, in this project, the AES cryptograph was simulated with FPGA, by using Verilog HDL. The main objectives are the architectural and algorithmic optimizations of the AES implementation, which would in turn benefit applications that are both speed and area critical.

The optimization methodology in this project was achieved using *S-Box* integration. *S-Box*, which is for *SubBytes*, and *Inverse S-Box*, which is for *InvSubBytes*, are both constituted of two 256-byte substitution tables. In fact, it is usual that in any high-speed full pipelining AES implementations, it would require 24 *S-Box* tables and 16 *Inverse S-Box* tables at any one time.

Nonetheless, mathematical formulas show that *S-Box* and *Inverse S-Box* could actually be achieved with only g, f and f^{-1}. *Multiplicative inverse*, or g, is a 256-byte look-up table. On the other hand, *affine transformation*, f, and its inverse, f^{-1}, can be implemented with a limited number of XOR gates. Accordingly, the number of substitution tables necessitated could be reduced by half.

Consequently, the new implementation would still obtain the identical *S-Box* and *Inverse S-Box* values, but merely from one look-up table and some simple logic gates. The new design shows that it can deliver a throughput of 203 Mbit/sec with hardware of 78,977 gate counts. Hardware complexity is reduced to 69% of its original while still able to function at core process of only 12 cycles.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

ABBREVIATIONS AND NOMENCLATURES

1. AES – Advanced Encryption Standard
2. Cipher – Series of transformations that converts plaintext to ciphertext using the Cipher Key
3. Cipher Key – Secret, cryptographic key that is used by the Key Expansion routine to generate a set of Round Keys; can be pictured as a rectangular array of bytes, having four rows and *Nk* columns
4. Ciphertext – Data output from the Cipher or input to the Inverse Cipher
5. DES – Data Encryption Standard
6. FPGA – Field Programmable Gate Array
7. Inverse Cipher – Series of transformations that converts ciphertext to plaintext using the Cipher Key
8. IOB – Input Output Blocks
9. Key Expansion – Routine used to generate a series of Routine Keys from the Cipher Key
10. LUT – Look Up Tables
11. MARS – AES finalist block cipher proposed by IBM
12. NIST – National Institute of Standards and Technology
13. Plaintext – Data input to the Cipher or output from the Inverse Cipher
14. RC6 – AES finalist block cipher proposed by RSA
15. Rijndael – 128 bit block cipher, named after its creators, Rijmen & Daemen, selected as AES algorithm in year 2000 by NIST
16. Round Key – Round keys are values derived from the Cipher Key using the Key Expansion routinel they are applied to the State in the Cipher and Inverse Cipher
17. Serpent – AES finalist block cipher proposed by Ross Anderson, Eli Biham and Lars Knudsen
18. State – Intermediate Cipher result that can be pictured as a rectangular array of bytes, having four rows and *Nb* columns.

19. S-box – Non-linear substitution table used in several byte substitution transformation and in the Key Expansion routine to perform a one-for-one substitution of a byte value
20. Twofish – AES finalist block cipher proposed by Counterpane Lab
21. Word – A group of 32 bits that is treated either as a single entity or as an array of 4 bytes

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF STUDY

Cryptography plays an important role in the security of data transmission. In 1997, the RSA Security Company issued a challenge to break the US government's Data Encryption Standard (DES) algorithm. In June of that year, the challenge was solved by the DESCHALL team who successfully recovered the 56-bit DES key. In response, the National Institute of Standards and Technology (NIST) requested candidates for a new Advanced Encryption Standard (AES) algorithm to replace DES, realizing that the algorithm's 56-bit key was no longer sufficient to provide the necessary security in many applications [1].

As an interim measure they adopted and standardized Triple-DES, which uses three passes of the DES algorithm and a 112 or 168-bit key. The AES requirements were for a block cipher capable of supporting a data block size of 128-bits and keys of 128, 192 and 256-bit in length. They wanted an algorithm whose security is at least as good as Triple-DES, but with significantly improved efficiency [1].

Among the 15 preliminary candidates, MARS, RC6, Rijndael, Serpent and Twofish were announced as the finalist candidates on August 9, 1999 for further evaluation. After studying all available information and public comments on these finalist candidates, NIST announced in October 2000 that Rijndael was selected as the AES algorithm [2]. Rijndael proved a secure and efficient algorithm when implemented in both hardware and software across a wide range of platforms.

1.2 PROBLEM STATEMENT

The project will address the efficient hardware implementation approaches for the AES algorithm. The hardware implementation would be FPGA-based. Compared to software implementations, hardware implementations provide more physical security as well as higher speed [2].

Different application of the AES algorithm may require different speed/area trade-offs. Some applications, such as smart cards and cellular phones, require small area. Other applications, such as WWW servers and ATMs, are speed critical. Some other applications, such as digital video recorders, require an optimization of speed/area ratio. Various optimizations for implementation are developed to suit the different demands of applications. Architectural optimizations make use of duplicating the round units, while algorithmic optimizations explore algorithm simplification inside each encryption/decryption round unit [2].

In this project, effort would be paid in order to look into the feasibility of achieving both architectural optimizations and algorithmic optimizations of AES implementation. It would effectively optimize the area utilization of applications that are area-crucial, such as smart cards and mobile phones, while still providing them with essential high speed.

1.3 OBJECTIVES AND SCOPE OF STUDY

1. Implement the cryptographic algorithm of AES, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128 bits.
2. Implement the AES with FPGA technology, using Verilog Hardware Description Language.
3. Investigate the feasibility of both architectural optimizations and algorithmic optimizations of AES with FPGA implementation, in order to benefit applications that are both speed and area crucial, such as smart cards and mobile phones.

4. Integrate the *S-Box* and the *Inverse S-Box* into a unified module, by utilizing a *Multiplicative Inverse* look-up table and simple logic gates.
5. Instantiate the combined module of combined *S-box/Inverse S-Box* with a modified structure of *Cipher* and *Inverse Cipher*, and subsequently implement the design with a top layer data path. The performance in comparison with the implementation of an ordinary design under the same data path would be studied.
6. Reduce the utilization of 256-byte *S-Box/Inverse S-box* tables from 40 to 20, which would eventually reduce the total area utilized.

Please refer to Appendix 1 for Final Year Project Gantt chart of first semester and Appendix 2 for Final Year Project Gantt chart of second semester.

CHAPTER 2
LITERATURE REVIEW AND THEORY

2.1 THE AES ALGORITHM

The AES algorithm is a symmetric-key block cipher in which both the sender and receiver use a single key to encrypt and decrypt the information. Although the block length of Rijndael can be 128, 192, or 256 bits, the AES algorithm only adopted the block length of 128 bits. Meanwhile, the key length can be 128, 192, or 256 bits [2].

The AES algorithm is a substitution linear transformation cipher based on S-boxes and operations in the Galois Fields. Implementation of the encryption round of AES requires realization of four component operations: *Substitution, ShiftRow, MixColumn, and KeyAddition.* Implementation of the decryption round of AES requires four inverse operations: *InvSubstitution, InvShiftRow, InvMixColumn, and KeyAddition* [3].

Substitution is composed of sixteen identical S-boxes working in parallel. *InvSubstitution* is composed of the same number of inverse S-boxes. Each of these S-boxes can be implemented independently using a 256x8 bit look-up table [3].

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 [3].

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 [3].

KeyAddition is a bitwise XOR of two 128 bit words.

2.2 ALGORITHM SPECIFICATIONS OF AES

Since the AES algorithm may be used with the three different key lengths (i.e. 128, 192 and 256 bits), the variances are therefore referred to as "AES-128", "AES-192" and "AES-256" [7].

Internally, the AES algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of bytes, each containing *Nb* bytes, where $Nb = 4$ in this standard. At the start of the *Cipher* and *Inverse Cipher*, the input – the array of bytes $in_0, in_1 \ldots in_{15}$ – is copied into the State array as illustrated in Figure 1. The *Cipher* or *Inverse Cipher* operations are then conducted on this State array, after which its final value is copied to the output – the array of bytes $out_0, out_1 \ldots out_{15}$.

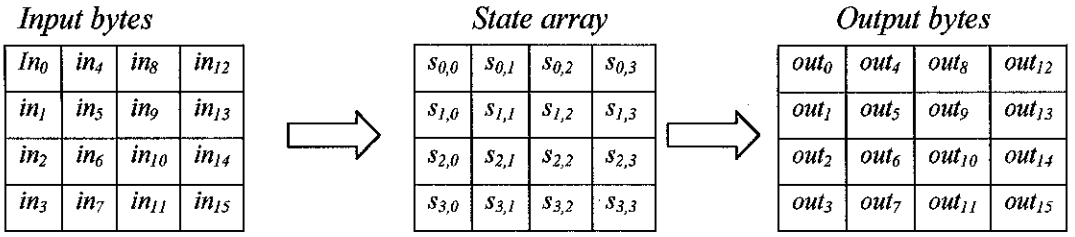


Figure 1: State array input and output

For AES algorithm, the length of the Cipher Key, *K*, is 128, 192, or 256 bits. The key length is represented by $Nk = 4$, 6, or 8, which reflects the number of 32-bit words in the Cipher Key. Meanwhile, the number of round to be performed during the execution of the algorithm is dependent on the key size. The number of rounds is represented by *Nr*, where $Nr = 10$ when $Nk = 4$, $Nr = 12$ when $Nk = 6$, and $Nr = 14$ when $Nk = 8$.

Table 1: Key-Block-Round Combinations

	Key Length (*Nk* words)	Block size (*Nb* words)	Number of rounds (*Nr*)
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

For both its *Cipher* and *Inverse Cipher*, the AES algorithm uses a round function that is composed of four different byte-oriented transformations: 1) byte substitution using a substitution table (S-box), 2) shifting rows of the State array by different offsets, 3) mixing the data within each column of the State array, and 4) adding a Round Key to the State.

2.2.1 Cipher

The *Cipher* is described in the following pseudo code. Notice that all *Nr* rounds are identical with the exception of the final round, which does not include the *MixColumns ( )* transformation.

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
byte state[4,Nb]

state = in

AddRoundKey(state, w[0, Nb-1])

for round = 1 step 1 to Nr–1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
end for

SubBytes(state)
ShiftRows(state)
AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

out = state
end
```

Figure 2: Pseudo Code for the Cipher

i) SubBytes ( ) Transformation

The *SubBytes ( )* transformation is a non-linear byte substitution that operates independently on each byte of the State using a substitution table (*S-box*).

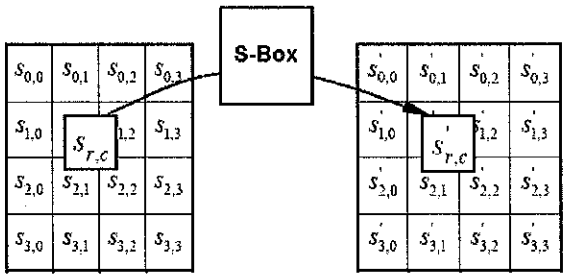


Figure 3: *SubBytes ( )* applies the *S-box* to each byte of the State

The *S-box* used in the *SubBytes ( )* transformation is presented in Appendix 3.

ii) ShiftRows ( ) Transformation

In the *ShiftRows ( )* transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted.

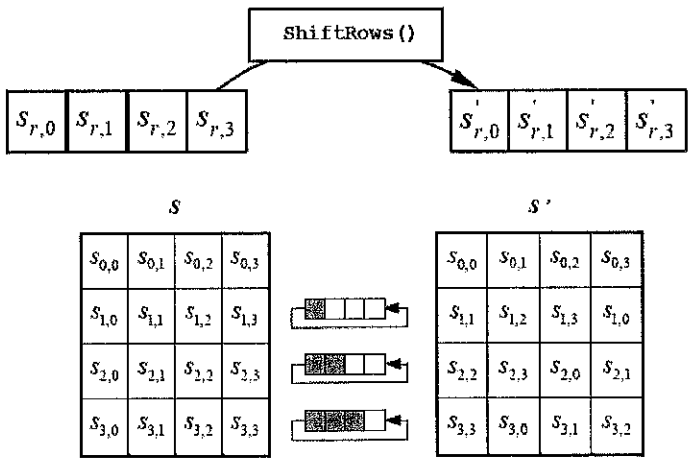


Figure 4: *ShiftRows ( )* cyclically shifts the last three rows in the State

iii) MixColumns ( ) Transformation

The *MixColumns ( )* transformation operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomial over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x)$. This can be written as a matrix multiplication [7]. Let

$$s'(x) = a(x) \otimes s(x)$$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb.$$

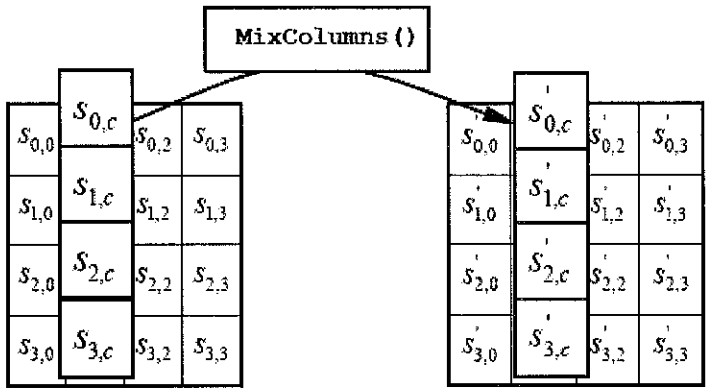


Figure 5: *MixColumns ( )* operates on the State column-by-column

iv) AddRoundKey ( ) Transformation

In the *AddRoundKey ( )* transformation, a Round Key is added to the State by a simple bitwise XOR operation. In the *Cipher*, the initial Round Key addition occurs when *round* = 0, prior to the first application of the round function. The application of the AddRoundKey ( ) transformation to the *Nr* rounds of the *Cipher* occurs when $1 \le round \le Nr$.

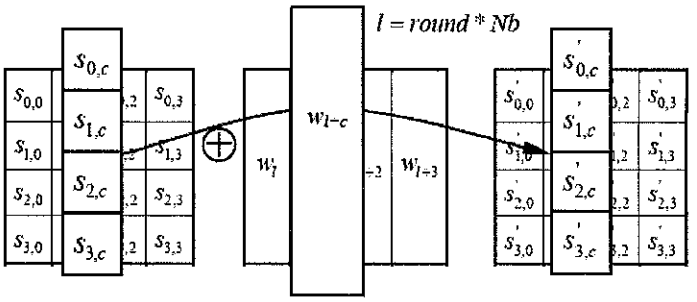


Figure 6: *AddRoundKey ( )* XORs each column of the State with a word from the key schedule

2.2.2 Key Expansion

The AES algorithm takes the Cipher Key, K, and performs a Key Expansion routine to generate a key schedule. The Key Expansion generates a total of $Nb(Nr + 1)$ words. The resulting key schedule consists of a linear array of 4-byte words, denoted $[w_i]$, with I in the range $0 \leq I < Nb(Nr + 1)$ [7].

The expansion of the input key into the key schedule proceeds according to the pseudo code below:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
        word temp

        i = 0

        while (i < Nk)
                w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
                i = i+1
        end while

        i = Nk

        while (i < Nb * (Nr+1)]
                temp = w[i-1]
                if (i mod Nk = 0)
                        temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
                else if (Nk > 6 and i mod Nk = 4)
                        temp = SubWord(temp)
                end if
                w[i] = w[i-Nk] xor temp
                i = i + 1
        end while
end
```

Figure 7: Pseudo Code for Key Expansion

Subword ( ) is a function that takes a four-byte input word and applies the *S-box* to each of the four bytes to produce an output word. The function *RotWord ( )* takes a word $[a_o,a_1,a_2,a_3]$ as input, performs a cyclic permutation, and returns the word $[a_1,a_2,a_3,a_0]$. The round constant word array, *Rcon[i]* is presented in Appendix 6.

2.2.3 Inverse Cipher

The *Cipher* transformation can be inverted and then implemented in reverse order to produce a straightforward *Inverse Cipher* for the AES algorithm [7].

Pseudo code for the *Inverse Cipher* is as follow:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
        begin

        byte state[4,Nb]

        state = in

        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

        for round = Nr-1 step -1 downto 1
                InvShiftRows(state)
                InvSubBytes(state)
                AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
                InvMixColumns(state)
        end for

        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[0, Nb-1])

        out = state
end
```

Figure 8: Pseudo Code for the Equivalent Inverse Cipher

i) InvShiftRows ( ) Transformation

InvShiftRows ( ) is the inverse of the *ShiftRows ( )* transformation. The byte in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row, $r = 0$, is not shifted [7].

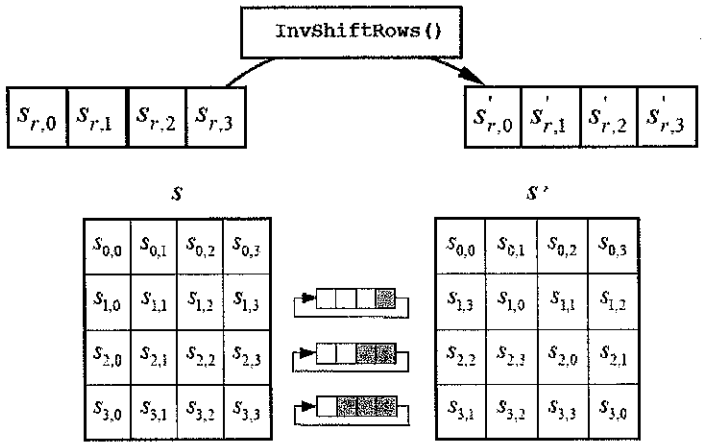


Figure 9: *InvShiftRows ( )* cyclically shifts the last three rows in the State

ii) InvSubBytes ( ) Transformation

InvSubBytes ( ) is the inverse of the byte substitution transformation, in which the inverse S-box is applied to each byte of the State [7]. The inverse S-box used in the *InvSubBytes ( )* transformation is presented in Appendix 4.

iii) InvMixColumns ( ) Transformation

InvMixColumns ( ) is the inverse of the *MixColumns ( )* transformation. *InvMixColumns ( )* operates on the State column-by-column, treating each column as a four-term polynomial. The columns are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial $a^{-1}(x)$. This can be written as a matrix multiplication [7]. Let

$s'(x) = a^{-1}(x) \otimes s(x)$

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb.$$

iv) Inverse of the AddRoundKey ( ) Transformation

AddRoundKey ( ), which was described previously, is its own inverse, since it only involves an application of the XOR operation [7].

2.3 DEDICATED HARDWARE IN AES

Rijndael (AES) is suited to be implemented in dedicated hardware. There are several trade-offs between chip area and speed possible. In dedicated hardware, the *SubBytes* step is the most critical part for a hardware implementation, for two reasons [8]:

1. In order to achieve the highest performance, S_{RD} (S-Box) needs to be instantiated 16 times (disregarding the key schedule). A straightforward implementation with 16 256-byte tables is likely to dominate the chip area requirements or the consumption of logic blocks.
2. Since Rijndael encryption and decryption are different transformations, a circuit that implements Rijndael encryption does not automatically support decryption.

However, when building dedicated hardware for supporting both encryption and decryption, we can limit the required chip area by using parts of the circuit for both transformations [8].

2.3.1 Decomposition of S_{RD}

The Rijndael S-Box S_{RD} is constructed from two transformations [8]:

$$S_{RD}[a] = f(g(a)),$$

where $g(a)$ is the transformation

$$a \rightarrow a^{-1} \text{ in } GF(2^8),$$

and $f(a)$ is an affine transformation. The transformation $g(a)$ is a self-inverse and hence

$$S_{RD}^{-1} = g^{-1}(f^{-1}(a)) = g(f^{-1}(a)).$$

Therefore, when we want both S_{RD} and S_{RD}^{-1}, we need to implement only g, f and f^1. Since both f and f^1 can be implemented with a limited number of XOR gates, the extra hardware can be reduced significantly compared to having to hardwire both S_{RD} and S_{RD}^{-1} [8].

CHAPTER 3
METHODOLOGY / PROJECT WORK

3.1 PROCEDURE IDENTIFICATION

The flowchart below is the methodology conducted throughout the entire project execution:

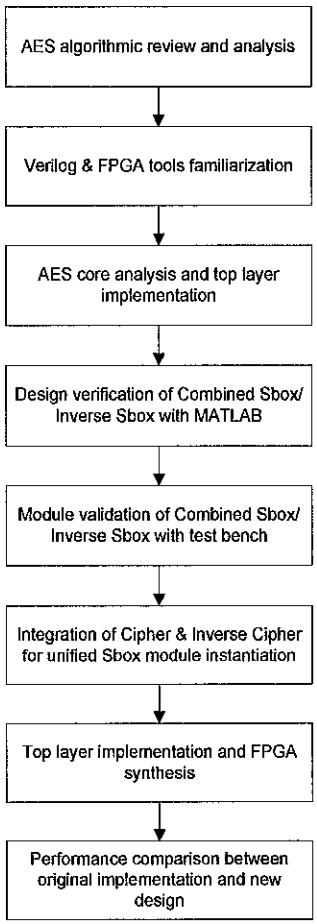


Figure 10: Methodology Flow Chart for the FYP Project Execution

3.1.1 AES Algorithmic Review and Analysis

The project was commenced with the review and analysis of AES-Rijndael algorithm. Both official resources on the algorithm, which are *FIPS PUB 197,* released by the NIST on the specifications of AES [reference 7] and *The Design of Rijndael (AES – Advanced Encryption Standard),* written by its very creators Joan Daemon and Vincent Rijmen [reference 8], were investigated thoroughly.

Next, the mathematical preliminaries in the AES design as well as the algorithmic specifications were studied. All the standard specifications in the *Cipher*, *Inverse Cipher* and *Key Expansion* were analyzed in order to identify the definite algorithm implementations.

Furthermore, instances of AES implementation by other researchers and academia were investigated in order to gain supplementary understanding on the subject studied. The step served as a foundation for the architectural optimization and algorithmic optimization of the cryptography.

3.1.2 Verilog and FPGA Tools Familiarization

Next, the very introduction into Verilog Hardware Description Language was done via the interactive tutorial by Active-HDL, Aldec. The tutorial materials offered a great insight into the increasing-significant hardware description language being used in the industry. Moreover, crucial information on Verilog was obtained from reference books as stated in the Reference section.

On the other hand, it was learned from the FPGA tool familiarization that the FPGA designs described in Verilog would be verified by a simulator (such as Active-HDL, Aldec). The validated Verilog codes would then become an input to the FPGA synthesis software (such as WEB Pack ISE), which performs the logic synthesis, mapping, placing and routing. Eventually, reports describing the area and the speed of the implementation, a net list used for timing simulation, and a bit stream to be used to program the FPGA device will be generated by the tools.

3.1.3 AES Core Analysis and Top Layer Implementation

A simple yet efficient AES/Rijndael IP Core [reference 10] developed by Rudolf Usselmann from ASICS.ws was studied and investigated. Nonetheless, it is an AES implementation with only a 128 bit key expansion (excluding 192 and 256 bit key). All the Verilog modules developed by the mentioned author, i.e. *aes_cipher_top.v*, *aes_inv_cipher_top.v*, *aes_sbox.v*, *aes_inv_sbox.v*, *aes_key_expan_128.v* and *aes_rcon.v* were analyzed thoroughly. Please refer to Appendix 8, 9, 10, 11, 12, 13.

The AES core consists of two blocks, i.e. the AES *Cipher* block which performs encryption and the AES *Inverse Cipher* block which performs decryption. Both block instantiate the same key expansion block.

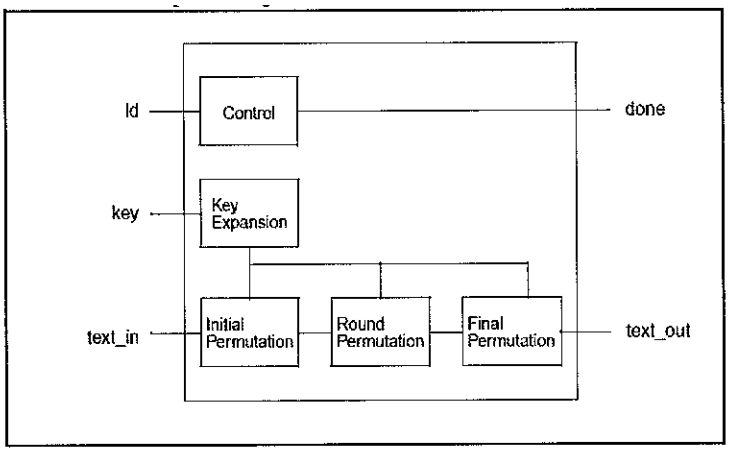


Figure 11: Cipher Core Architecture Overview

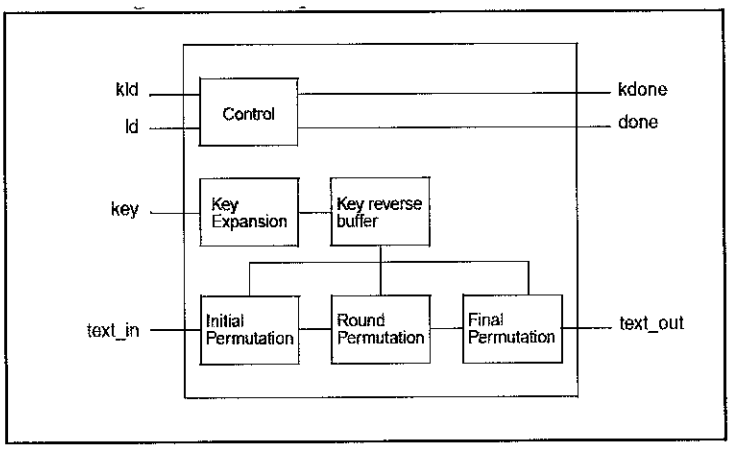


Figure 12: Inverse Cipher Core Architecture Overview

During the first phase of the research project, the cryptograph was implemented with different size of data path. Data input, key input and data output of the cryptograph were implemented with bit size of 64, 32 and 16 bits.

The top layer of the implementation was achieved with Verilog codes in behavioral style, intended to find out the effect of different data path size on the implementation. The arbitrary device chosen for the implementation (FPGA area simulation) is Xilinx VirtexE – PQ240 -6. Despite their variation in the size of data input, key input and data output, all the implementations have the common I/O configurations and internal structure.

The performances in terms of area utilization and speed for each of the distinct implementation were compared and analyzed. The investigation outcome obtained was incorporated into the design and project execution of the second phase.

3.1.4 Design Verification of Combined S-Box with MATLAB

From the discussion in section 2.3, it was realized that both *S-Box* and *Inverse S-Box* could actually be derived from a common *multiplicative inverse* table, with merely some additional logic gates. In other words, the area utilized in the FPGA for the two tables could actually be reduced to only one table.

More discussions on the algorithm simplification and the design rational would be discussed in Chapter 4. Basically, the designing steps involved translating the *affine transformation* matrix into its equivalent mathematical formula. The formula would then be able to perform calculation on the *multiplicative inverse* in order to obtain *S-Box* and *Inverse S-Box* respectively.

The mathematical formula derived, together with the *multiplicative inverse* table, were further translated into MATLAB codes for MATLAB simulation. Two output arrays, which are the *S-Box* and the *Inverse S-box,* were compared against the standard tables under AES specifications. The simulation result would be presented in the next chapter. Nonetheless, the simulation step verified the feasibility of the combination design.

3.1.5 Module Validation of Combined S-Box with Test Bench

Upon the verification test of the combined *S-Box* design, the algorithms in MATLAB codes were translated into two Verilog modules, namely *aes_sbox_inv.v* and *aes_mul_inv.v*. Please refer to Appendix 16 and 17 for the two module files.

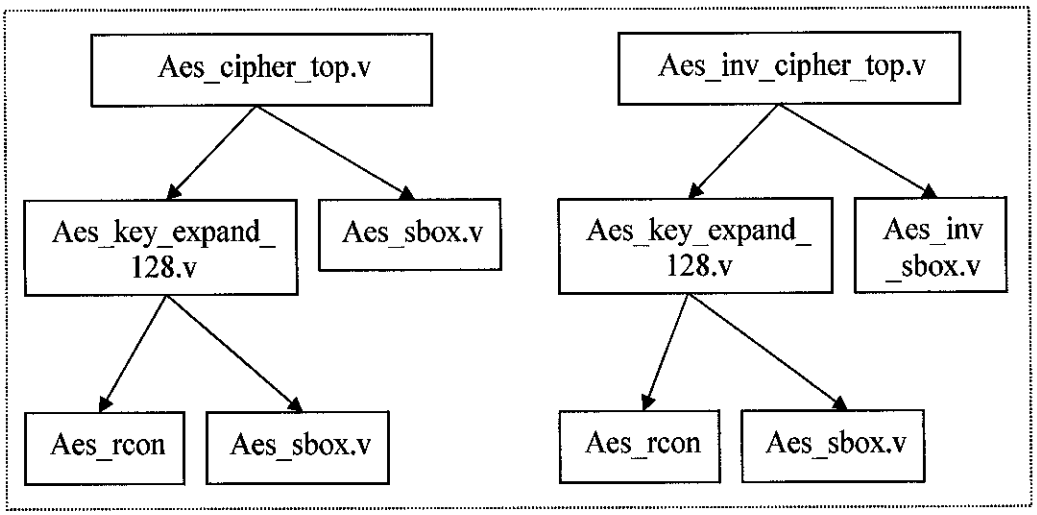


Figure 13: Relationships of Original AES Modules

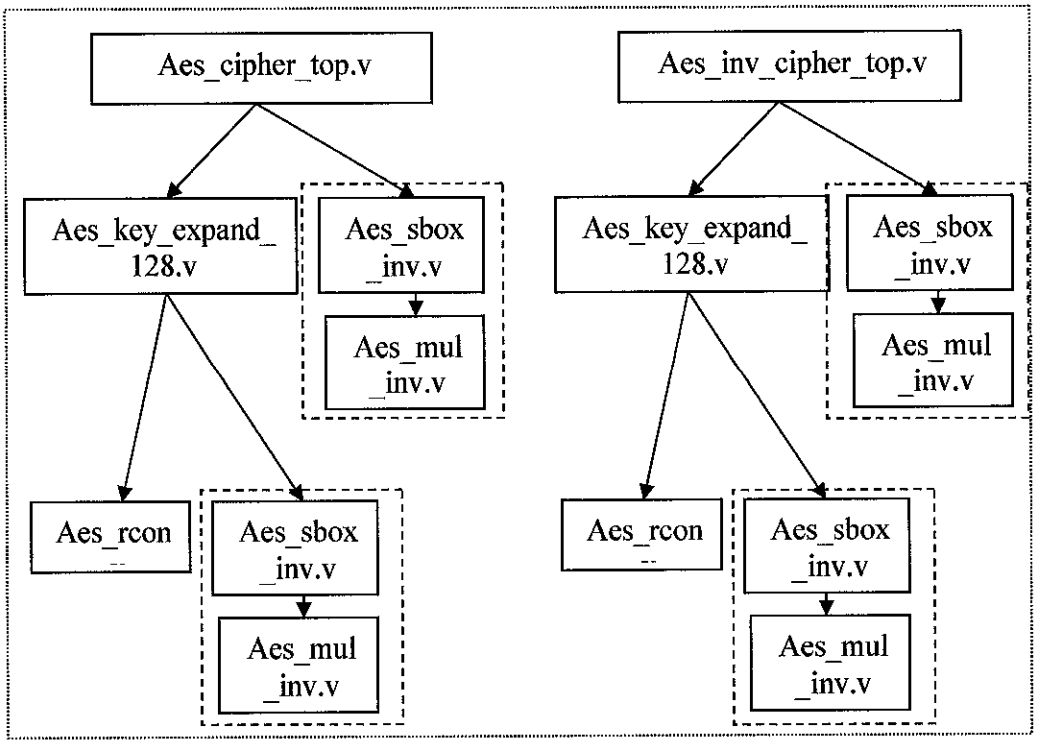


Figure 14: Relationships of Modified AES Modules

The *aes_mul_inv.v* module would be containing the *multiplicative inverse* table while the *aes_sbox_inv.v* module would be having the logical gates for both the *S-Box* and the *Inverse S-Box* computations.

As can be seen from Figure 13 and Figure 14, both the *aes_sbox.v* and the *aes_inv_sbox.v* modules were replaced by the combined, identical *aes_sbox_inv.v* and *aes_mul_inv.v* modules.

Nevertheless, before it can be shown that the gate count could be reduced by having the new design implementation, the new modules devised were required to prove that they would be capable in producing the same functionality and specification as the original *aes_sbox.v* and *aes_inv_sbox.v* did. Validation of the new design was much crucial.

Therefore, a test bench written by Rudolf Usselmann from ASICS.ws was employed for the justification purposes. As can be seen from Appendix 18, the test bench consisted of 283 official test vectors, which were provided by NIST. While the original implementation (Figure 13) developed by Rudolf showed an error-free test output, the new designed modules replacing both the *S-box* and the *Inverse S-Box* as shown in Figure 14 were subjected for the same test.

The test output of this modified AES modules would be discussed in Chapter 4.

3.1.6 Integrated Crypto Module for Unified S-Box Module Instantiation

Upon the verification of the combined *S-Box* concept and the subsequent validation of the new modules developed, test was conducted to review the area utilization performance for the new implementation.

Referred to Figure 13, if the modules structure were subjected for a parallel architecture and a high-speed design, *aes_sbox.v* would be instantiated 16 times under *aes_cipher_top.v*, 4 times under *aes_key_expan_128.v* (under *Cipher*), and another 4 times under *aes_key_expan_128.v* (under *Inverse Cipher*). Meanwhile, *aes_inv_sbox.v* would be instantiated 16 times under *aes_inv_cipher_top.v*. By

having this configuration, it would mean that a total of forty (40) 256-byte tables would be hard-wired into the FPGA. Even with the configuration as in Figure 14, the total repetitions of table required would still be the same. More discussions on this issue would be presented in the subsequent chapter of results and discussion.

In order to be beneficial from the combined *S-Box* and *Inverse S-Box*, we would need the configuration of modules structure as below:

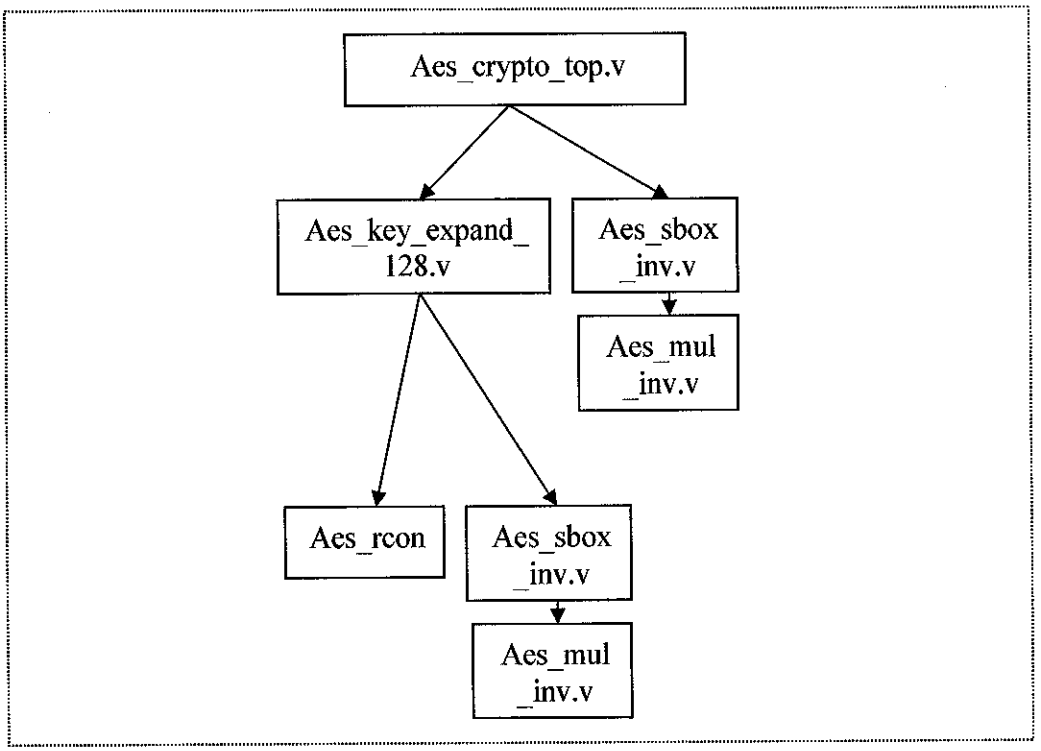


Figure 15: Integrated Crypto Module for Unified S-Box Module Instantiation

The components for both the *Cipher* and the *Inverse Cipher* would be residing within the *aes_crypto_top.v* module. By having this configuration, both *Cipher* and *Inverse Cipher* could instantiate the *aes_sbox_inv.v* module, which would return the *S-Box* and *Inverse S-Box* values by further instantiating the *aes_mul_inv.v* module and performing some simple computation.

Hence, we could see that the total number of 256-byte table required for implementing the *SubBytes* function had been reduced by half.

3.1.7 Top Layer Implementation and FPGA Synthesis

Both the original AES implementation (as in Figure 13) and the integrated AES crypto (as in Figure 15) were implemented with a top layer data path and input/output buffers.

Due to the reasons that would be discussed in the next chapter, the data path size was chosen to be 64-bit for both of the configurations. Besides, research from phase 1 had shown that the bit size of the top layer data path would largely affect the total gate count. Therefore, the data path implementation had to be the same for the two cases so that we could conduct an unbiased area utilization performance and arrive at a justifiable result.

The physical layout, operation and timing of the top layer implementation would be presented in Chapter 4.

Next, both of the Verilog projects (original and modified) were synthesized for syntax checking. Subsequently, they were translated and mapped (with an arbitrary device) under Xilinx Web Pack, in order to obtain the final gate count for performance comparisons.

3.1.8 Performance Comparison

Nevertheless, the execution of this project was only being on the simulation and verification of the RTL codes.

The performances in terms of area and speed for each of the implementation were compared and analyzed. The main objective of this research project was to find out the performance in terms of area utilization for the combined *S-Box* AES design in comparison with the ordinary AES design. The efficiency of space saving under this new approach was found out and the objectives were achieved successfully.

3.2 TOOLS REQUIRED

1. Verilog Hardware Description Language Programming tools – Active-HDL by Aldec
2. FPGA Software – WEB Pack ISE

CHAPTER 4
RESULTS AND DISCUSSION

4.1 OPTIMIZATION VIA S-BOX / INVERSE S-BOX INTEGRATION

There are two large substitution tables in AES. One is for *SubBytes* operation [Appendix 3], and another is for *InvSubBytes* operation [Appendix 4].

The first table, S-box table, was used in two functions, i.e. *SubBytes* and *KeyExpansion*. On the other hand, the inverse S-box table was used in function *InvSubBytes*. We know that these two tables are not the same, thus we must build two different ROMs (256-byte) in order to store the table to realize the S-box and inverse S-box operations [11].

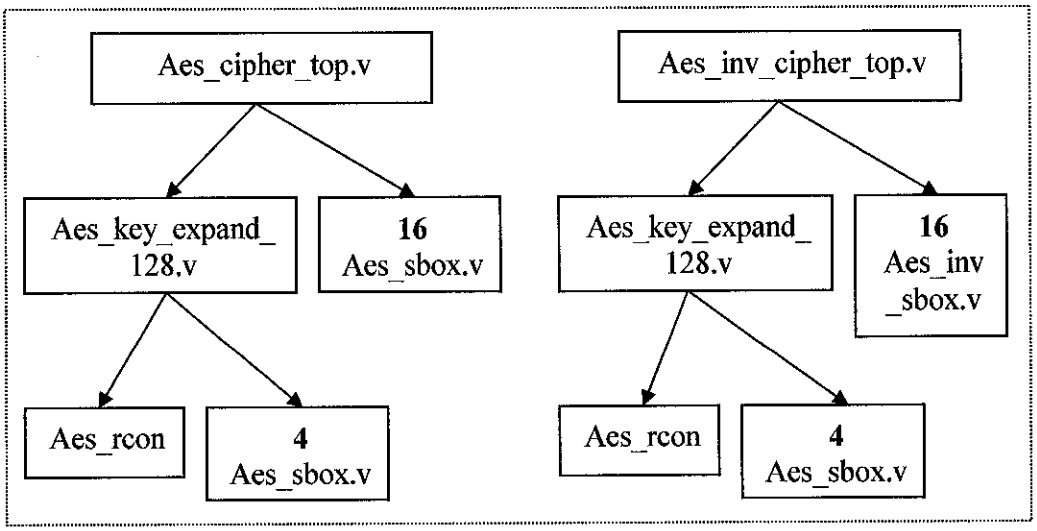


Figure 16: Total Number of S-Box and Inverse S-Box Instantiation

Referring to the figure above, for a parallel architecture of AES design, it usually needs several tables. For example, in a high-speed design of 128-bit AES [reference 10, which is the IP Core being referred to in this project], it needs a total of 24 *S-Box* modules and 16 *Inverse S-Box* modules. 16 *S-Box* modules would be called in the

implementation of *SubBytes* function, another 4 *S-Box* modules would be called in each of the implementation of *KeyExpansion* function under *Cipher* and *Inverse Cipher* and then 16 inverse S-box modules would be called in the implementation of *InvSubBytes* function.

In this case, a substantial amount of hardware resource will be occupied if *SubBytes* and *InvSubBytes* utilize their own tables in encryption (*Cipher*) and decryption (*Inverse Cipher*). It is hence desirable to obtain a simplified way so as to reduce the hardware complexity.

As described in the section 2.3.1, the operation of the S-box can be expressed as

$$S_{RD}[a] = f(g(a)), \tag{1}$$

where $f(a)$ is the affine transformation

$$f(a) = \begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

On the other hand, the operation of the inverse S-box can be expressed as:

$$S_{RD}^{-1} = g^{-1}(f^{-1}(a)) \tag{2}$$

Since the transformation $g(a)$ is a self-inverse and hence

$$S_{RD}^{-1} = g^{-1}(f^{-1}(a)) = g(f^{-1}(a)) \tag{3}$$

where $f^{-1}(a)$ is the inverse affine transformation

$$f'(a) = \begin{bmatrix} b_7' \\ b_6' \\ b_5' \\ b_4' \\ b_3' \\ b_2' \\ b_1' \\ b_0' \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_7 \\ b_6 \\ b_5 \\ b_4 \\ b_3 \\ b_2 \\ b_1 \\ b_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Therefore, when we want both S_{RD} and S_{RD}^{-1}, we need to implement only g, f and f^{-1}. Since both f and f^{-1} can be implemented with a limited number of XOR gates, the extra hardware can be reduced significantly compared to having to hardwire both S_{RD} and S_{RD}^{-1} [8].

By examining Eq.1 and Eq. 3, a common look-up table, i.e. *Multiplicative Inverse, g(xy)*, is employed so the *S-Box* and the *Inverse S-Box* can be integrated to reduce the hardware requirement for *SubBytes* and *InvSubBytes*. Refer to Appendix 5 for the look-up table of *g(xy)*.

Subsequently, the function of *f(a)* could be achieved by:

$b_7' = b_7 \text{ xor } b_6 \text{ xor } b_5 \text{ xor } b_4 \text{ xor } b_3 \text{ xor } 0$
$b_6' = b_6 \text{ xor } b_5 \text{ xor } b_4 \text{ xor } b_3 \text{ xor } b_2 \text{ xor } 1$
$b_5' = b_5 \text{ xor } b_4 \text{ xor } b_3 \text{ xor } b_2 \text{ xor } b_1 \text{ xor } 1$
$b_4' = b_4 \text{ xor } b_3 \text{ xor } b_2 \text{ xor } b_1 \text{ xor } b_0 \text{ xor } 0$
$b_3' = b_7 \text{ xor } b_3 \text{ xor } b_2 \text{ xor } b_1 \text{ xor } b_0 \text{ xor } 0$
$b_2' = b_7 \text{ xor } b_6 \text{ xor } b_2 \text{ xor } b_1 \text{ xor } b_0 \text{ xor } 0$
$b_1' = b_7 \text{ xor } b_6 \text{ xor } b_5 \text{ xor } b_1 \text{ xor } b_0 \text{ xor } 1$
$b_0' = b_7 \text{ xor } b_6 \text{ xor } b_5 \text{ xor } b_4 \text{ xor } b_0 \text{ xor } 1$

On the other hand, the function of *f'(a)* could be achieved by:

$b_7' = (b_6 \text{ xor } 1) \text{ xor } (b_4 \text{ xor } 0) \text{ xor } (b_1 \text{ xor } 1)$
$b_6' = (b_5 \text{ xor } 1) \text{ xor } (b_3 \text{ xor } 0) \text{ xor } (b_0 \text{ xor } 1)$
$b_5' = (b_7 \text{ xor } 0) \text{ xor } (b_4 \text{ xor } 0) \text{ xor } (b_2 \text{ xor } 0)$
$b_4' = (b_6 \text{ xor } 1) \text{ xor } (b_3 \text{ xor } 0) \text{ xor } (b_1 \text{ xor } 1)$
$b_3' = (b_5 \text{ xor } 1) \text{ xor } (b_2 \text{ xor } 0) \text{ xor } (b_0 \text{ xor } 1)$

$b_2' = (b_7 \text{ xor } 0) \text{ xor } (b_4 \text{ xor } 0) \text{ xor } (b_1 \text{ xor } 1)$
$b_1' = (b_6 \text{ xor } 1) \text{ xor } (b_3 \text{ xor } 0) \text{ xor } (b_0 \text{ xor } 1)$
$b_0' = (b_7 \text{ xor } 0) \text{ xor } (b_5 \text{ xor } 1) \text{ xor } (b_2 \text{ xor } 0)$

Thus, we could see that merely 40 XOR gates are required by each of the process for the above implementation. Moreover, a look up table of 16x16 bytes was totally eliminated.

Nevertheless, further optimization proved that the algorithmic structure of *f'(a)* could be simplified. In fact, the XORing of 3 Booleans values within each equation could be reduced to merely 1 finite value. The optimized structure would be as followed.

Function of *f'(a)* could be achieved by:

$b_7' = b_6 \text{ xor } b_4 \text{ xor } b_1 \text{ xor } 0$
$b_6' = b_5 \text{ xor } b_3 \text{ xor } b_0 \text{ xor } 0$
$b_5' = b_7 \text{ xor } b_4 \text{ xor } b_2 \text{ xor } 0$
$b_4' = b_6 \text{ xor } b_3 \text{ xor } b_1 \text{ xor } 0$
$b_3' = b_5 \text{ xor } b_2 \text{ xor } b_0 \text{ xor } 0$
$b_2' = b_7 \text{ xor } b_4 \text{ xor } b_1 \text{ xor } 1$
$b_1' = b_6 \text{ xor } b_3 \text{ xor } b_0 \text{ xor } 0$
$b_0' = b_7 \text{ xor } b_5 \text{ xor } b_2 \text{ xor } 1$

Hence, we could see that merely 24 XOR gates are required, as compared to the 40 XOR gates in the previous implementation. 40% of gate count has been reduced.

Therefore, by merely utilizing one look-up table, which is the *Multiplicative Inverse* table, it was estimated that the amount of hardware for this implementation would have a significant decrease of 50%, as compared with the original hardware requirement without the functional integration.

4.1.1 S-Box Integration Design Verification with MATLAB

```
clear all
g = [
   '00','01','8d','f6','cb','52','7b','d1','e8','4f','29','c0','b0','e1','e5','c7';
   '74','b4','aa','4b','99','2b','60','5f','58','3f','fd','cc','ff','40','ee','b2';
   '3a','6e','5a','f1','55','4d','a8','c9','c1','0a','98','15','30','44','a2','c2';
   '2c','45','92','6c','f3','39','66','42','f2','35','20','6f','77','bb','59','19';
   '1d','fe','37','67','2d','31','f5','69','a7','64','ab','13','54','25','e9','09';
   'ed','5c','05','ca','4c','24','87','bf','18','3e','22','f0','51','ec','61','17';
```

```
  '16','5e','af','d3','49','a6','36','43','f4','47','91','df','33','93','21','3b';
  '79','b7','97','85','10','b5','ba','3c','b6','70','d0','06','a1','fa','81','82';
  '83','7e','7f','80','96','73','be','56','9b','9e','95','d9','f7','02','b9','a4';
  'de','6a','32','6d','d8','8a','84','72','2a','14','9f','88','f9','dc','89','9a';
  'fb','7c','2e','c3','8f','b8','65','48','26','c8','12','4a','ce','e7','d2','62';
  '0c','e0','1f','ef','11','75','78','71','a5','8e','76','3d','bd','bc','86','57';
  '0b','28','2f','a3','da','d4','e4','0f','a9','27','53','04','1b','fc','ac','e6';
  '7a','07','ae','63','c5','db','e2','ea','94','8b','c4','d5','9d','f8','90','6b';
  'b1','0d','d6','eb','c6','0e','cf','ad','08','4e','d7','e3','5d','50','1e','b3';
  '5b','23','38','34','68','46','03','8c','dd','9c','7d','a0','cd','1a','41','1c'
];

h = g';

for n = 1:256

  b(1:4) = dec2bin(base2dec(h(2*n-1),16),4);
  b(5:8) = dec2bin(base2dec(h(2*n),16),4);

  for m = 1:8
    b(m) = b(m) - 48;          %to convert b(m) into an integer
  end

  f(1) = xor(0,(xor(b(1),(xor(b(2),(xor(b(3),(xor(b(4),b(5))))))))));
  f(2) = xor(1,(xor(b(2),(xor(b(3),(xor(b(4),(xor(b(5),b(6))))))))));
  f(3) = xor(1,(xor(b(3),(xor(b(4),(xor(b(5),(xor(b(6),b(7))))))))));
  f(4) = xor(0,(xor(b(4),(xor(b(5),(xor(b(6),(xor(b(7),b(8))))))))));
  f(5) = xor(0,(xor(b(1),(xor(b(5),(xor(b(6),(xor(b(7),b(8))))))))));
  f(6) = xor(0,(xor(b(1),(xor(b(2),(xor(b(6),(xor(b(7),b(8))))))))));
  f(7) = xor(1,(xor(b(1),(xor(b(2),(xor(b(3),(xor(b(7),b(8))))))))));
  f(8) = xor(1,(xor(b(1),(xor(b(2),(xor(b(3),(xor(b(4),b(8))))))))));

  s = sprintf('%d',f);          %write formatted data to a string

  S = dec2hex(bin2dec(s),2)

  x = ceil(n/16);
  if mod(n,16)==0
    y = 48;
  else
    y = 3*(rem(n,16));
  end

  Sbox(x,y-2) = S(1);
  Sbox(x,y-1) = S(2);
  Sbox(x,y) = ' ';

end

for n = 1:256

  hex = dec2hex((n-1),2);
  b(1:4) = dec2bin(base2dec(hex(1),16),4);
  b(5:8) = dec2bin(base2dec(hex(2),16),4);

  for m = 1:8
    b(m) = b(m) - 48;          %to convert b(m) into an integer
  end

  fp(1) = xor(0,(xor(b(2),(xor(b(4),b(7))))));
  fp(2) = xor(0,(xor(b(3),(xor(b(5),b(8))))));
  fp(3) = xor(0,(xor(b(1),(xor(b(4),b(6))))));
  fp(4) = xor(0,(xor(b(2),(xor(b(5),b(7))))));
  fp(5) = xor(0,(xor(b(3),(xor(b(6),b(8))))));
  fp(6) = xor(1,(xor(b(1),(xor(b(4),b(7))))));
  fp(7) = xor(0,(xor(b(2),(xor(b(5),b(8))))));
  fp(8) = xor(1,(xor(b(1),(xor(b(3),b(6))))));


  s = sprintf('%d',fp);

  temp = bin2dec(s);

  S(1) = h(temp*2 + 1);
  S(2) = h(temp*2 + 2);
```

```
  S

  x = ceil(n/16);
  if mod(n,16)==0
     y = 48;
  else
     y = 3*(rem(n,16));
  end

  InvSbox(x,y-2) = S(1);
  InvSbox(x,y-1) = S(2);
  InvSbox(x,y) = ' ';

end
```

Figure 17: MATLAB M-file Codes for Design Verification

Please note from the above MATLAB codes that the subscripts used for the notation of *f(a)* and *f'(a)* in MATLAB codes are in the reversing order. This is due to the differing notation handling of matrix elements in MATLAB. The structure, however, is identical.

From the simulation result, it was found out that:

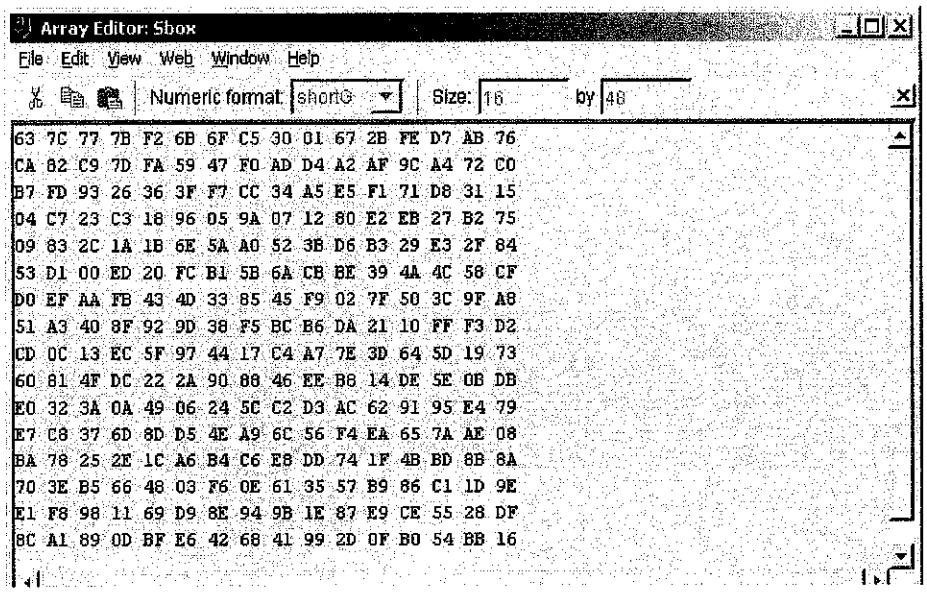


Figure 18: S-box (S_{RD})

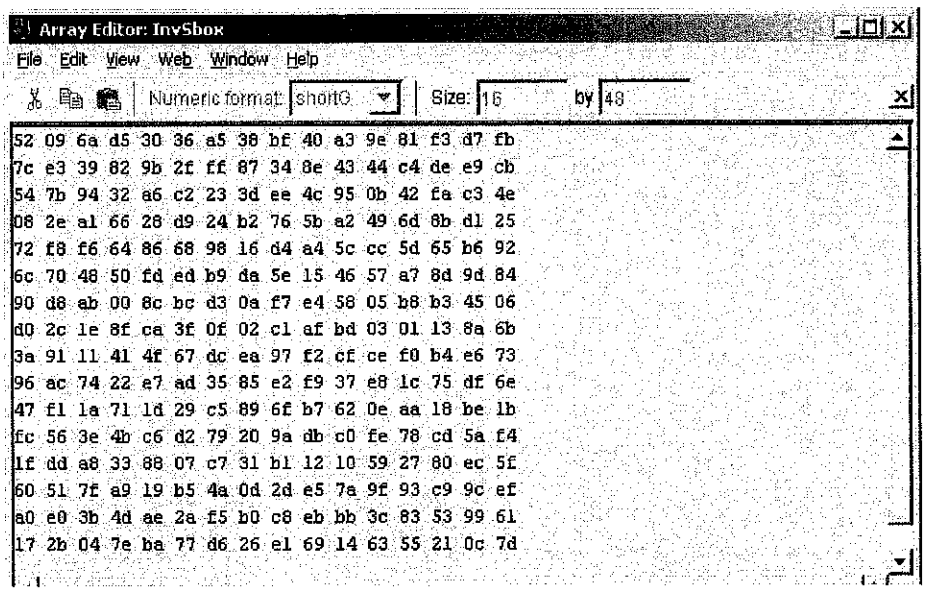


Figure 19: Inverse S-box (S_{RD}^{-1})

From the simulation results shown, it was verified that both the *S-Box* and the *Inverse S-Box* can be achieved effectively by the concept suggested. The logic gates for deriving the two look-up tables would be sharing a common *g(x,y)* table. Therefore, it managed to reduce the necessity from having 2 look-up tables to only 1, while ensuring that the contents within the 2 original tables would be always deliverable.

4.1.2 Integrated S-Box Verilog Modules Construction

With ordinary implementation, *Cipher* would be invoking *S-Box* meanwhile *Inverse Cipher* would be invoking *Inverse S-Box* separately. Besides, *Key Expand* module would be invoking *S-Box* as how *Cipher* does. Please refer to Figure 13 in Section 3.1.5 for the illustration.

By combining both *S-Box* and *Inverse S-Box* under a unified module, a *multiplicative inverse* table would be residing in a lower hierarchy module, and the combined module of *S-Box/Inverse S-Box* would be incorporating some simple logic gates in order to realize the required output. Nevertheless, an additional input would be passed into the module in order to retrieve the out of EITHER *S-Box* OR *Inverse S-Box*.

The equivalence of the modules after the integration would then be as followed:

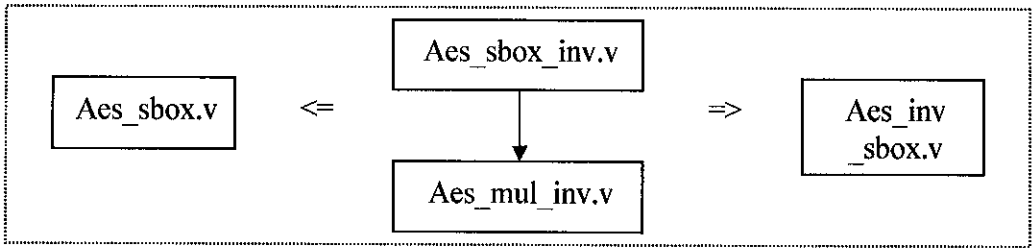


Figure 20: Equivalence of Modules after the Integration

The Verilog module for realizing the *S-Box* and the *Inverse S-Box* computational gates is *aes_sbox_inv.v*. A Boolean value, *b*, in which a value of 1 is for encryption and a value of 0 is for decryption, would be passing into the module for invoking the intended operation. The following is the partial Verilog codes of the module; please refer to Appendix 16 for the complete module.

```
module aes_sbox_inv(b,a,d);
input b;
input [7:0] a;
output [7:0] d;

reg [7:0] d;
reg [7:0] a_in;
wire [7:0] a_sub;

always @(b)
if(b)
        begin
        a_in = a;
        d[7] = a_sub[7]^a_sub[6]^a_sub[5]^a_sub[4]^a_sub[3]^1'b0;
        d[6] = a_sub[6]^a_sub[5]^a_sub[4]^a_sub[3]^a_sub[2]^1'b1;
        d[5] = a_sub[5]^a_sub[4]^a_sub[3]^a_sub[2]^a_sub[1]^1'b1;
        d[4] = a_sub[4]^a_sub[3]^a_sub[2]^a_sub[1]^a_sub[0]^1'b0;
        d[3] = a_sub[7]^a_sub[3]^a_sub[2]^a_sub[1]^a_sub[0]^1'b0;
        d[2] = a_sub[7]^a_sub[6]^a_sub[2]^a_sub[1]^a_sub[0]^1'b0;
        d[1] = a_sub[7]^a_sub[6]^a_sub[5]^a_sub[1]^a_sub[0]^1'b1;
        d[0] = a_sub[7]^a_sub[6]^a_sub[5]^a_sub[4]^a_sub[0]^1'b1;
        end
else
        begin
        a_in[7] = a[6]^a[4]^a[1]^1'b0;
        a_in[6] = a[5]^a[3]^a[0]^1'b0;
        a_in[5] = a[7]^a[4]^a[2]^1'b0;
        a_in[4] = a[6]^a[3]^a[1]^1'b0;
        a_in[3] = a[5]^a[2]^a[0]^1'b0;
        a_in[2] = a[7]^a[4]^a[1]^1'b1;
        a_in[1] = a[6]^a[3]^a[0]^1'b0;
        a_in[0] = a[7]^a[5]^a[2]^1'b1;
        d = a_sub;
        end

aes_mul_inv m0( .b(  b ),       .m( a_in ), .g( a_sub ));

endmodule
```

Figure 21: Partial Verilog codes of aes_sbox_inv.v

Upon selecting the desired operation, the module *aes_mul_inv.v* that contains the *multiplicative inverse* table would be instantiated in order to load the *g(x,y)* values for *S-Box* values or *Inverse S-Box* values computation. The following is the partial Verilog codes of the module, please refer to Appendix 17 for the complete module.

```
module aes_mul_inv(b,m,g);
input b;
input [7:0] m;
output [7:0] g;
reg [7:0] g;

always @(b)
        case(m)
          8'h00: g=8'h00;
          8'h01: g=8'h01;
          8'h02: g=8'h8d;
          8'h03: g=8'hf6;
          8'h04: g=8'hcb;
                  :
                  :
                  :
          8'hfb: g=8'ha0;
          8'hfc: g=8'hcd;
          8'hfd: g=8'h1a;
          8'hfe: g=8'h41;
          8'hff: g=8'h1c;
        endcase

endmodule
```

Figure 22: Partial Verilog codes of aes_mul_inv.v

4.1.3 S-Box Integration Module Validation with Test Bench

As stated in Section 3.1.5, the newly designed Verilog modules for the integrated *S-Box* were subjected to validation test, in order to verify their functionality and validity.

By having the Verilog module hierarchy as shown in Figure 14 (in Section 3.1.5), in which the *S-Box* module and the *Inverse S-Box* module were replaced by the module pair of *S-Box-Inverse* and *multiplicative inverse*, the design was subjected for a simulation test of 80 ms. The simulation output as displayed in the following Figure 23 proved that the integrated *S-Box* module would be capable in producing the identical and error-free *S-Box* values and *Inverse S-Box* values, without having to compromise the speed and the modification of algorithm in the topper hierarchy modules.

Hence, the design was proven in providing a perfect substitution for the original *S-Box* and *Inverse S-Box* modules.

```
asim test
# ELBREAD: Elaboration process.
# ELBREAD: Elaboration time 0.0 [s].
# KERNEL: Main thread initiated.
# KERNEL: Kernel process initializatio phase.
# KERNEL: Time resolution set to 10ps.
# ELAB2: Elaboration final pass...
# ELAB2: Create instances ...
# ELAB2: Create instances complete.
# ELAB2: Elaboration final pass complete - time: 1.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 1385 kB (elbread=250 elab2=857 kernel=278 sdf=0)
# 2:47 PM, Saturday, April 10, 2004
# Simulation has been initialized
# Selected Top-Level: test (test)
run 80 ms
# :
# :
# :
# : ****************************************************
# : * AES Test bench ...
# : ****************************************************
# :
# :
# :
# :
# : Started random test ...
# :
# :
# : Test Done. Found 0 Errors.
# :
# :
# RUNTIME: RUNTIME_0068 test_bench_top.v (437): $finish called.
# KERNEL: stopped at time: 77015 ns
endsim
```

Figure 23: Test Bench Simulation Output

4.2 COMBINATION OF CIPHER AND INVERSE CIPHER

As contrary to Figure 16 in Section 4.1, the following Figure 24 shows that only half of the total numbers of look-up tables is desired. The number displayed an encouraging decrease from (16 + 4 + 16 + 4 = 40) to (16 + 4 = 20).

Nonetheless, this decrease could not be achieved if we separate the *Cipher* module and the *Inverse Cipher* module. The both need to be combined or unified so that they could instantiate the same set of integrated *S-Box* modules. While both the *Cipher* and the *Inverse Cipher* residing in a unified crypto module, all the repetitions of modules in the lower hierarchy could now be eliminated.

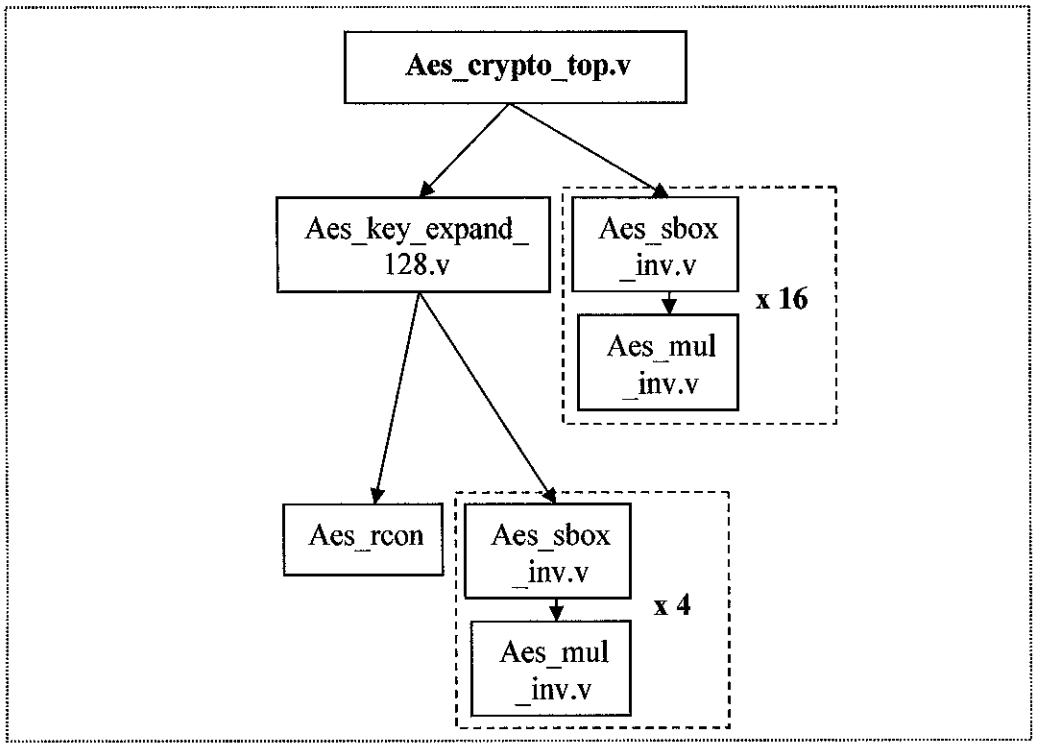


Figure 24: Total Number of S-Box and Inverse S-Box Instantiation

As can be seen from this new relationship, *key expansion* would require 4 *sbox_inv* (inclusive of *mul_inv*) at a time, for the pipelining implementation. Besides, *Cipher* and *Inverse Cipher* would be sharing 16 duplicated *sbox_inv* (inclusive of *mul_inv*) for *SubBytes* and *InvSubBytes* respectively, instead of having 16 *aes_sbox* and 16 *aes_inv_sbox*. In fact, they could share the combined module because they are two mutual exclusive processes and only one process would be invoked at a time.

The combined *S-Box/Inverse S-Box* module would be instantiated within unified crypto module as followed:

Sbox_inv *identifier* (.b (*1 or 0*), .a(*input value*), .d(*output value*));

in which

identifier is to duplicate the module 16 times for pipelining implementation

b is a Boolean value in which 1 is for encryption and 0 is for decryption

The following is the partial Verilog codes of the unified crypto module, please refer to Appendix 15 for the complete module constructed.

```
aes_key_expand_128 u0(
        .clk(           clk     ),
        .kld(           ld      ),
        .key(           key     ),
        .wo_0(          w0      ),
        .wo_1(          w1      ),
        .wo_2(          w2      ),
        .wo_3(          w3      ));

aes_sbox_inv    us00(   .b(     ciph_opt ),     .a(     in_s00  ), .d(  out_s00 ));
aes_sbox_inv    us01(   .b(     ciph_opt ),     .a(     in_s01  ), .d(  out_s01 ));
aes_sbox_inv    us02(   .b(     ciph_opt ),     .a(     in_s02  ), .d(  out_s02 ));
aes_sbox_inv    us03(   .b(     ciph_opt ),     .a(     in_s03  ), .d(  out_s03 ));
aes_sbox_inv    us10(   .b(     ciph_opt ),     .a(     in_s10  ), .d(  out_s10 ));
aes_sbox_inv    us11(   .b(     ciph_opt ),     .a(     in_s11  ), .d(  out_s11 ));
aes_sbox_inv    us12(   .b(     ciph_opt ),     .a(     in_s12  ), .d(  out_s12 ));
aes_sbox_inv    us13(   .b(     ciph_opt ),     .a(     in_s13  ), .d(  out_s13 ));
aes_sbox_inv    us20(   .b(     ciph_opt ),     .a(     in_s20  ), .d(  out_s20 ));
aes_sbox_inv    us21(   .b(     ciph_opt ),     .a(     in_s21  ), .d(  out_s21 ));
aes_sbox_inv    us22(   .b(     ciph_opt ),     .a(     in_s22  ), .d(  out_s22 ));
aes_sbox_inv    us23(   .b(     ciph_opt ),     .a(     in_s23  ), .d(  out_s23 ));
aes_sbox_inv    us30(   .b(     ciph_opt ),     .a(     in_s30  ), .d(  out_s30 ));
aes_sbox_inv    us31(   .b(     ciph_opt ),     .a(     in_s31  ), .d(  out_s31 ));
aes_sbox_inv    us32(   .b(     ciph_opt ),     .a(     in_s32  ), .d(  out_s32 ));
aes_sbox_inv    us33(   .b(     ciph_opt ),     .a(     in_s33  ), .d(  out_s33 ));
```

Figure 25: Partial Verilog codes of aes_crypto_top.v

4.3 TOP LAYER IMPLEMENTATION

In order to simulate and synthesize both of the original design and the new design, a top layer data path for the input/output would be necessary. It would work as an implementation for the AES core.

From the research finding obtained in the first phase, the area utilization of the top layer data path for achieving a constant size of data input/output is inversely proportional to the bit size of the per cycle input/output. In other words, to achieve the AES input/output that is 128-bit, more gates would be required if 32-bit per I/O cycle is employed as compared to 64-bit per I/O cycle. Nevertheless, one would have to consider the total I/O available on the targeted FPGA chip, before deciding on a larger number of data path size for the sake of area saving.

Secondly, despite on the fact that more gates would be required for a smaller data path size, another drawback would be a smaller size of data path would require more clock cycles to complete the data input/output.

During the first phase of the research project, the *Cipher* of the AES Core was implemented with different data path size of 64, 32 and 16 bits. The arbitrary device chosen for the FPGA area simulation was Xilinx VirtexE – PQ240-6. The results of the simulation would evidently illustrate the effects of different data path size on the overall implementation.

4.3.1 Results Comparison of Various Data Path Size

Table 2: Comparison of total clock cycle

	16-bit Data Path	32-bit Data Path	64-bit Data Path
Clock Cycle	28	20	16

It was observed that the more the 128-bit data block is divided into smaller clusters, the more clock cycles are required to complete the operation.

Table 3: Comparison of slice utilization percentage

	16-bit Data Path	32-bit Data Path	64-bit Data Path
Slice Utilization %	96	85	58

It was noticed that more slices are required for relatively small sized data path. Implementation of 16-bit data path almost reaches the full limit of area utilization.

Table 4: Comparison of the number of 4-input LUT

	16-bit Data Path	32-bit Data Path	64-bit Data Path
4-input LUT %	78	68	42

Again, it was found that more 4-input LUTs were required for relatively small sized data path.

Table 5: Comparison of the number of bonded IOB

	16-bit Data Path	32-bit Data Path	64-bit Data Path
Bonded IOB %	32	62	123
Additional JTAG gate count for IOB	2,496	4,800	9,408

Nevertheless, it could be seen that the 64-bit data path implementation exceeds the allowable physical limit of IOB (for the particular Xilinx VirtexE device chosen for

the simulation). Therefore, IOB would be a major factor in determining the data path size despite the undeniable fact that larger data path size would require less area utilization.

Moreover, as listed in the table, the additional JTAG gate count required for IOB was linearly proportional to the data path size; in which JTAG gate for the 32-bit data path was double the size for the 16-bit data path, meanwhile JTAG gate for the 64-bit data path was even quadruple the size for the 16-bit data path.

Table 6: Comparison of total equivalent gate count

	16-bit Data Path	32-bit Data Path	64-bit Data Path
Total Equivalent Gate Count	43,123	38,249	25,894

Conclusively, greater number of total equivalent gate count was required for smaller data path size implementation.

4.3.2 64-bit Top Layer Data Path Implementation

With the goals of:

1) reducing the clock cycle as small as possible
2) not exceeding the bonded IOB
3) remaining at low equivalent gate count
4) utilizing less number of slices
5) minimizing the 4-input LUT

The top layer for both of the original design and the integrated *S-Box* design were constructed with data path size of 64-bit. The arbitrary device chosen, which was Xilinx VirtexE – BG352-8, supported this number of data path size. Please refer to Appendix 7 – *aes_ori64_top.v* for the top layer of the original design and Appendix 14 – *aes_new64_top.v* for the top layer of the integrated *S-Box* design.

The common physical layout and the pin description would be as followed.

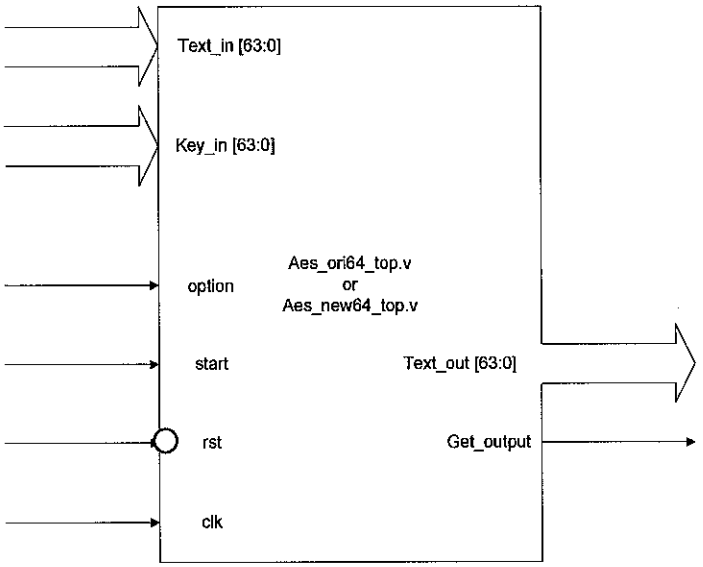


Figure 26: Physical Layout of Top Layer Data Path Implementation

Table 7: Pin Description of Top Layer Data Path Implementation

Name	Type	Description
rst	Input	Core reset, active low
clk	Input	Core clock signal
start	Input	When HIGH, a cryptographic operation is started
option	Input	HIGH for encryption; LOW for decryption
Key_in [63:0]	Input	Input key
Text_in [63:0]	Input	Input data
Text_out [63:0]	Output	Output data
Get_output	Output	Output data valid

A rising input on the 'start' port would trigger the beginning of a cryptographic operation on the data 'text_in', using the 'key_in' as key. The data block (from 'text_in') and the key (from 'key_in') would then be fed into the core serially, 64 bits at a time. All the input blocks would be captured at the rising edge of the external supplied clock 'clk'.

With the proper selection of 'option' for either encryption or decryption, and subsequently upon the receiving of the 128-bit data block, the core would start to process the state according to the AES algorithm.

The timing diagram below would show how the data is fed to the core at the start.

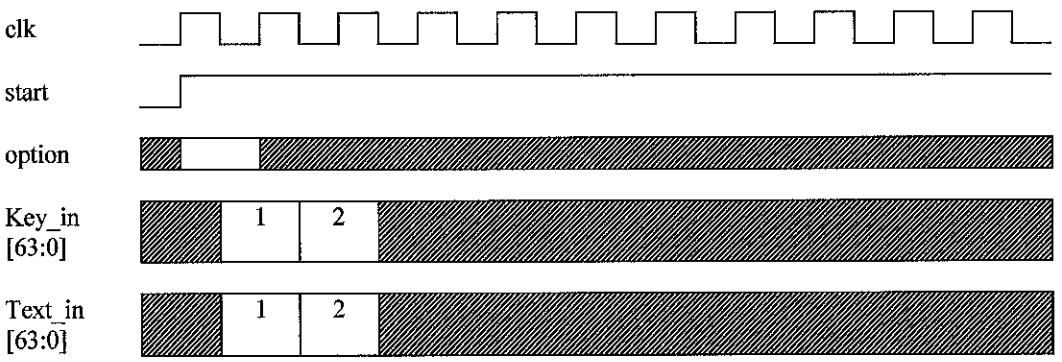


Figure 27: Key and Data Input at the start of Encryption/Decryption

When all the rounds were completed, the 'get_output' signal would be raised and the output data would start to flow out. This would be shown in the following timing diagram.

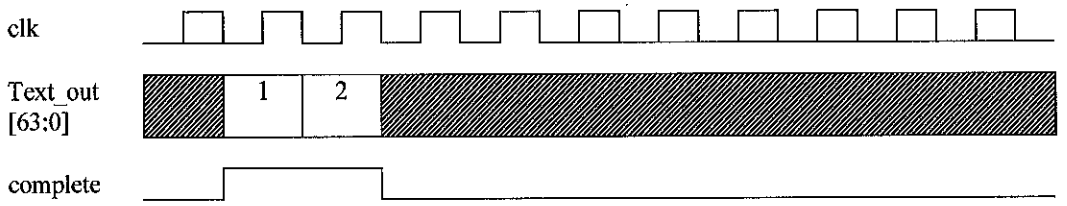


Figure 28: Output Text from an Encryption/Decryption Operation Being Outputted

4.4 PERFORMANCE COMPARISON

Before drawing a conclusion on the overall performance, the gate counts of the individual modules of *S-Box*, *Inverse S-Box* and *Integrated S-Box* would be evaluated.

Table 8: Gate Count of S-Box, Inverse S-Box and Integrated S-Box

Module	Gate Count
S-Box	1836
Inverse S-Box	1815
Integrated S-Box	1974

Therefore, the percentage reduce in gate count in terms of the *S-Box* modules would be

$$[(1836 + 1815) - 1974] / (1836 + 1815) \times 100\% = 45.9\%$$

The *Integrated S-Box* module performed the functions of both *S-Box* and *Inverse S-Box* with only one look-up table so that the amount of hardware for implementation of *SubBytes* and *InvSubBytes* would have a significant decrease of 46%, as compared with the original hardware implementation without the functional integration.

Subsequently, the overall performance in terms of area utilization would be compared. Please refer to Appendix 20 for the Map report of the original design and Appendix 22 for the Map report of the Integrated *S-Box* design.

Table 9: Performance Comparison of Area Utilization

Aspect	Original Design	Integrated *S-Box* Design
Number of Slices	6,979	4,382
Total Number Slice Registers	1,760	1,445
Total Number 4 Input LUT	13,415	8,341
Number of bonded IOBs	196	196
Total Equivalent Gate Count for Design	114,785	78,977

Hence, the percentage of overall gate count for the new design would be

$$78977 / 114785 \times 100\% = 68.8\%$$

In other words, the new design would be just 69% of the original design in area utilization. Nevertheless, the unified crypto module of the new design was merely combining both the *Cipher* and the *Inverse Cipher* modules without further optimizing the module architecturally. In fact, the performance of the area utilization could further be improved if the unified crypto module would be optimized and enhanced beforehand.

Next, the overall performance in terms of speed would be evaluated. Please refer to Appendix 19 for the Synthesis report of the original design and Appendix 21 for the Synthesis report of the Integrated *S-Box* design.

Table 10: Performance Comparison of Speed

Aspect	Original Design	Integrated *S-Box* Design
Critical Path (minimum)	23.042 ns	39.333 ns
Clock Frequency (maximum)	43.399 MHz	25.424 MHz
Clock Cycle	12 (core) + 2 (input data path) + 2 (output data path) = 16 cycles	12 (core) + 2 (input data path) + 2 (output data path) = 16 cycles
Throughput	128-bit / (16 x 23.042 ns) = 347 Mbit/sec	128-bit / (16 x 39.333 ns) = 203 Mbit/sec

Given the fact that the design was synthesized under the optimization goal of *area* and with the optimization effort of *high* (grade 2), the decrease in clock frequency and subsequently the decrease in throughput would be inevitable.

Nonetheless, the data delivery would still be in the range of high data rate. With further architectural optimizations on the unified crypto module, much improvement would be able to be achieved.

CHAPTER 5
CONCLUSION

5.1 REVIEW AND CONCLUSIONS

The complete execution of the AES cryptograph inevitably requires two large substitution tables, in which one is called *S-Box* and another is called *Inverse S-Box*. *S-Box* table is used in two functions, which are *SubBytes* in the encryption process and *KeyExpansion* in the encryption/decryption process. On the other hand, *Inverse S-Box* is used in function *InvSubBytes* in the decryption process. Absolutely, it is apparent that these two tables are not the same, thus requiring two different ROM's of 256-byte in order to store the table for the operations mentioned above.

For a high-speed, full pipelining and parallel architecture of AES implementation, the requirement of the look-up tables can go to an extent of 24 *S-Box* tables and 16 *Inverse S-Box* tables at any one time. Within the circumstances, a substantial amount of hardware resource will be utilized if *SubBytes* and *InvSubBytes* utilize their own tables in encryption (*Cipher*) and decryption (*Inverse Cipher*).

Nevertheless, it is enthralling that the hardware complexity could actually be largely reduced by integrating both the *S-Box* table and the *Inverse S-Box* table under a combined module. Mathematical formulas used to derive the two tables show that when we want both *S-Box* and *Inverse S-Box*, we need to implement only g, f and f^{-1}, in which g is a 256-byte table called *multiplicative inverse* and both f and f^{-1} can be implemented with a limited number of XOR gates. In other words, with merely one look-up table and some simple logic gates, we would still be obtaining both *S-Box* and *Inverse S-Box*.

The design concept was proven with MATLAB at the outset, followed by the construction of *aes_sbox_inv.v* module and *aes_mul_inv.v* module, which were

validated by a test bench with official AES test vectors. The module of *aes_sbox_inv.v* contains the logic gates for realizing the f and f^{-1} functions while the module of *aes_mul_inv.v* is itself the *multiplicative inverse* look-up table.

By implementing with a top layer data path of 64-bit and subsequently subjected under the synthesis and the arbitrary device mapping in Xilinx Web Pack, the new design shows that it can deliver a throughput of 203 Mbit/sec with hardware of 78,977 gate count. Hardware complexity is reduced to 69% of its original yet still functions at core process of merely 12 cycles.

Conclusively, this new design managed to effectively optimize the area utilization of AES in applications that are area-crucial, such as smart cards readers and mobile phones, while still allowing them to run with essential high speed. The objectives of the research project were met through the architectural optimizations and the algorithmic optimizations of the substitution tables.

5.2 SUGGESTED FUTURE WORK FOR EXPANSION & CONTINUATION

In terms of area utilization, it can be further improved by performing the following:

(1) Optimize the architectural structure of the unified crypto module by removing the duplicating data registers of the *Cipher* and the *Inverse Cipher*.

(2) Integrate *MixColumns* and *InvMixColumns* and subsequently share the *xtime()* function and its derivatives (higher order of the function).

(3) Fully utilize the key buffer for *KeyExpansion* function so that continual encryption or decryption process could retrieve their round keys from the buffer without re-generating it.

(4) Implement the design with its full range of key size, i.e. 128-bit, 192-bit and 256-bit key.

(5) Implement pipelining for the top layer data path so that the cryptograph can continuously input / output data while the core is executing the encryption / decryption.

REFERENCES

[1] Maire McLoone, John V McCanny, "Rijndael FPGA Implementation Utilizing Look-up Tables", Signal Processing Systems, IEEE (2001)

[2] Xinmiao Zhang, Keshab K. Parhi, "Implementation Approaches for the Advanced Encryption Standard Algorithm", Circuits and Systems Magazine, IEEE (2002)

[3] Christian Chitu, David Chien, Charles Chien, Ingrid Verbauwhede, Frank Chang, "A Hardware Implementation in FPGA of the Rijndael Algorithm", Circuits and Systems, The 2002 45th Midwest Symposium (2002)

[4] Donald E. Thomas, Philip R. Moorby, "The Verilog Hardware Description Language", 4th Edition, Kluwer Academic Publishers (1998)

[5] Bruce Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd Edition, John Wiley and Sons, Inc (1996)

[6] Joan Daemen and Vincent Rijmen, "AES Submission Document on Rijndael, Version 2", September 1999.
http://csrc.nist.gov/CryptoToolkit/aes.rijndael/Rijndael.pdf

[7] FIPS PUB 197, "Advanced Encryption Standard (AES)", National Institute of Standards and Technology, U.S. Department of Commerce, November 2001.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[8] Joan Daemen and Vincent Rijmen, "The Design of Rijndael, AES – The Advanced Encryption Standard", Springer (2001)

[9] Verilog HDL Language Training, "Active-HDL, Version 5.1", Aldec (2001)

[10] Rudolf Usselmann, "AES (Rijndael) IP Cores", Rev 1.1, ASICS.ws, November 2002.
http://www.opencores.org/projects/aes_core/

[11] Chih-Chung Lu and Shau-Yin Tseng, "Integrated Design of AES (Advanced Encryption Standard) Encrypter and Decrypter", ASAP, IEEE (2002)

APPENDIX 1 – FINAL YEAR PROJECT GANTT CHART OF FIRST SEMESTER

No.	Activity	Week														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.	FYP Briefing	●														
2.	Proposal of Project Title		●													
3.	Preliminary Research Work															
	- Background, problem statement, objectives, project planning - Literature research & review - Procedure identification															
4.	Submission of Preliminary Report				●											
5.	Project Work															
	- Literature research & review - Study AES - Study Verilog HDL															
6.	Submission of Progress Report								●							
7.	Project Work (Cont.)															
	- Study AES IP Core - Architectural optimization - Algorithmic optimization															

No.	Activity	Week														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	- Implement top layer with different data path size (16, 32 & 64 bits) - Analyze output results - Plan goals for phase 2															
8.	Submission of Interim Report Final Draft												●			
9.	Oral Presentation														●	
10.	Submission of Interim Report															●
		●	Milestone													
			Progress													

APPENDIX 2 – FINAL YEAR PROJECT GANTT CHART OF SECOND SEMESTER

No.	Activity	Week																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	19
1.	Project Work (Cont.)																	
	- Investigate integration of cipher/inv																	
	- Feasibility study on combining S-box & combining MixColumns																	
2.	Submission of Progress Report 1				●													
3.	Project Work (Cont.)																	
	- Combine S-box & Inv S-box																	
	- Architectural optimizations																	
4.	Submission of Progress Report 2								●									
5.	Project Work (Cont.)																	
	- Test bench verification																	
	- Data path implementation - Performance comparison - Conclusion & Documentations																	
6.	Submission of Dissertation Final Draft											●						
7.	Exhibition													●				

No.	Activity	Week																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	17	19
8.	Submission of Dissertation (Soft													●				
	Cover)																	
9.	Oral Presentation															●		
10.	Extended Abstracts																●	
11.	Submission of Dissertation (Hard																	●
	Cover)																	
		●	Milestone															
			Progress															

Appendix 3 S-box: substitution values for the byte *xy* (in hexadecimal format)

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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

Appendix 4 Inverse S-box: substitution values for the byte *xy* (in hexadecimal format)

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	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	dd	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

Appendix 5 Multiplicative Inverse, *g(xy)* (in hexadecimal format)

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

Appendix 6 Round constants for the key generation

i	0	1	2	3	4	5	6	7
RC[i]	00	01	02	04	08	10	20	40

i	8	9	10	11	12	13	14	15
RC[i]	80	1B	36	6C	D8	AB	4D	9A

i	16	17	18	19	20	21	22	23
RC[i]	2F	5E	BC	63	C6	97	35	6A

i	24	25	26	27	28	29	30	31
RC[i]	D4	B3	7D	FA	EF	C5	91	39

```
//////////////////////////////////////////////////////////////////////
////                                                              ////
////  AES Top Layer for Original AES Core developed               ////
////  by Rudolf Usselmann (rudi@asics.ws)                         ////
////  (for serial input of 64 bit per cycle)                      ////
////                                                              ////
////  Author: Lee Yi Lin                                          ////
////          leeyilin@yahoo.com                                  ////
////                                                              ////
////                                                              ////
////  Composed for Final Year Project of EE Faculty, UTP          ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
//
//
//  $Date: 2004/4/01 $
//  $Revision: 1.0 $
//  $Author: Lee Yi Lin $
//
//  Change History:
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_ori64_top (clk, rst, option, start, get_output, key_in, text_in, text_out);

input clk, rst;
input option;
input start;
output get_output;
input [63:0] key_in;
input [63:0] text_in;
output [63:0] text_out;

reg get_output;
reg [63:0] text_out;

// local wires

reg [127:0] text_in_buf;
wire [127:0] ciphertext_out;
wire [127:0] plaintext_out;
reg [127:0] key_buf;
reg option_buf;
reg load_ciph, load_inv, load_key;
reg run;
wire done1, done2;
reg complete;
reg [7:0] counter;
reg [7:0] cnt_temp, cnt_temp2;

always @(posedge clk)
          if(!rst)            counter <= #1 8'h00;
          else
          if(complete)        counter <= #1 8'b00;
          else
          if(start)           counter <= #1 8'h01;
          else
          if (run)            counter <= #1 counter + 8'h01;

always @(posedge clk)
          if(!rst)            run <= #1 1'b0;
          else
          if(start)           run <= #1 1'b1;
          else
          if(complete)        run <= #1 1'b0;

always @(posedge clk)
          if(start)           option_buf <= #1 option;

always @(posedge clk)
          if(!((|counter) & start)       text_in_buf[063:000] <= #1 text_in;
          else
          if(counter == 8'h01)                    text_in_buf[127:064] <= #1 text_in;

always @(posedge clk)
          if(!((|counter) & start)       key_buf[063:000] <= #1 key_in;
          else
          if(counter == 8'h01)                    key_buf[127:064] <= #1 key_in;

always @(posedge clk)
          if(option_buf)
                    begin
                              if(start)           load_ciph <= #1 1'b0;
                              else
                              if(counter == 8'h01)          load_ciph <= #1 1'b1;
                              else
                              if(counter == 8'h02)          load_ciph <= #1 1'b0;
                    end
          else
          if(!option_buf)
                    begin
                              if(start)           load_key <= #1 1'b0;
                              else
                              if(counter == 8'h01)          load_key <= #1 1'b1;
                              else
                              if(counter == 8'h02)          load_key <= #1 1'b0;

                              if(load_key)        cnt_temp <= #1 8'h0b;
```

```
                        else
                        if(|cnt_temp)       cnt_temp <= #1 cnt_temp - 8'h01;

                        load_inv <= #1 !(|cnt_temp[7:1]) & cnt_temp[0];
                end

// encryption or decryption process takes place

always @(posedge clk)
        if(done1 || done2)  assign cnt_temp2 = counter;

always @(posedge clk)
        if(done1 || done2)  text_out <= #1 option_buf ? ciphertext_out[063:000] : plaintext_out[063:000];
        else
        if((counter - cnt_temp2) == 8'h01)      text_out <= #1 option_buf ? ciphertext_out[127:064] :
plaintext_out[127:064];

always @(posedge clk)
        if(done1 || done2 || ((counter - cnt_temp2) == 8'h01)) get_output <= #1 1'b1;

always @(posedge clk)
        if((counter - cnt_temp2) == 8'h01)      complete <= #1 1'b1;
        else
        if((counter - cnt_temp2) == 8'h02)      complete <= #1 1'b0;


aes_cipher_top u0(
        .clk(               clk                 ),
        .rst(               rst                 ),
        .ld(                load_ciph ),
        .done(              done1     ),
        .key(               key_buf   ),
        .text_in( text_in_buf         ),
        .text_out(          ciphertext_out      )
        );

 aes_inv_cipher_top u1(
        .clk(               clk                 ),
        .rst(               rst                 ),
        .kld(               load_key  ),
        .ld(                load_inv  ),
        .done(              done2     ),
        .key(               key_buf   ),
        .text_in( text_in_buf         ),
        .text_out(          plaintext_out       )
        );

endmodule
```

```
//////////////////////////////////////////////////////////////////////
////                                                              ////
////  AES Cipher Top Level                                        ////
////                                                              ////
////                                                              ////
////  Author: Rudolf Usselmann                                    ////
////          rudi@asics.ws                                       ////
////                                                              ////
////                                                              ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/   ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
////                                                              ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                     ////
////                         www.asics.ws                         ////
////                         rudi@asics.ws                        ////
////                                                              ////
//// This source file may be used and distributed without         ////
//// restriction provided that this copyright statement is not    ////
//// removed from the file and that any derivative work contains  ////
//// the original copyright notice and the associated disclaimer.////
////                                                              ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY      ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED    ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS    ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR       ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,          ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES     ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE    ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF   ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT   ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT   ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          ////
//// POSSIBILITY OF SUCH DAMAGE.                                  ////
////                                                              ////
//////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_cipher_top.v,v 1.1.1.1 2002/11/09 11:22:48 rudi Exp $
//
//  $Date: 2002/11/09 11:22:48 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_cipher_top.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:48  rudi
//               Initial Checkin
//
//
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_cipher_top(clk, rst, ld, done, key, text_in, text_out );
input           clk, rst;
input           ld;
output          done;
input   [127:0] key;
input   [127:0] text_in;
output  [127:0] text_out;

//////////////////////////////////////////////////////////////////////
//
// Local Wires
//

wire    [31:0]  w0, w1, w2, w3;
reg     [127:0] text_in_r;
reg     [127:0] text_out;
reg     [7:0]   sa00, sa01, sa02, sa03;
reg     [7:0]   sa10, sa11, sa12, sa13;
reg     [7:0]   sa20, sa21, sa22, sa23;
reg     [7:0]   sa30, sa31, sa32, sa33;
wire    [7:0]   sa00_next, sa01_next, sa02_next, sa03_next;
wire    [7:0]   sa10_next, sa11_next, sa12_next, sa13_next;
wire    [7:0]   sa20_next, sa21_next, sa22_next, sa23_next;
wire    [7:0]   sa30_next, sa31_next, sa32_next, sa33_next;
wire    [7:0]   sa00_sub, sa01_sub, sa02_sub, sa03_sub;
wire    [7:0]   sa10_sub, sa11_sub, sa12_sub, sa13_sub;
wire    [7:0]   sa20_sub, sa21_sub, sa22_sub, sa23_sub;
wire    [7:0]   sa30_sub, sa31_sub, sa32_sub, sa33_sub;
wire    [7:0]   sa00_sr, sa01_sr, sa02_sr, sa03_sr;
wire    [7:0]   sa10_sr, sa11_sr, sa12_sr, sa13_sr;
wire    [7:0]   sa20_sr, sa21_sr, sa22_sr, sa23_sr;
wire    [7:0]   sa30_sr, sa31_sr, sa32_sr, sa33_sr;
wire    [7:0]   sa00_mc, sa01_mc, sa02_mc, sa03_mc;
wire    [7:0]   sa10_mc, sa11_mc, sa12_mc, sa13_mc;
wire    [7:0]   sa20_mc, sa21_mc, sa22_mc, sa23_mc;
wire    [7:0]   sa30_mc, sa31_mc, sa32_mc, sa33_mc;
reg             done, ld_r;
reg     [3:0]   dcnt;

//////////////////////////////////////////////////////////////////////
//
```

```
// Misc Logic
//

always @(posedge clk)
          if(!rst)  dcnt <= #1 4'h0;
          else
          if(ld)              dcnt <= #1 4'hb;
          else
          if(|dcnt) dcnt <= #1 dcnt - 4'h1;

always @(posedge clk) done <= #1 !(|dcnt[3:1]) & dcnt[0] & !ld;
always @(posedge clk) if(ld) text_in_r <= #1 text_in;
always @(posedge clk) ld_r <= #1 ld;

////////////////////////////////////////////////////////////////////
//
// Initial Permutation (AddRoundKey)
//

always @(posedge clk)          sa33 <= #1 ld_r ? text_in_r[007:000] ^ w3[07:00] : sa33_next;
always @(posedge clk)          sa23 <= #1 ld_r ? text_in_r[015:008] ^ w3[15:08] : sa23_next;
always @(posedge clk)          sa13 <= #1 ld_r ? text_in_r[023:016] ^ w3[23:16] : sa13_next;
always @(posedge clk)          sa03 <= #1 ld_r ? text_in_r[031:024] ^ w3[31:24] : sa03_next;
always @(posedge clk)          sa32 <= #1 ld_r ? text_in_r[039:032] ^ w2[07:00] : sa32_next;
always @(posedge clk)          sa22 <= #1 ld_r ? text_in_r[047:040] ^ w2[15:08] : sa22_next;
always @(posedge clk)          sa12 <= #1 ld_r ? text_in_r[055:048] ^ w2[23:16] : sa12_next;
always @(posedge clk)          sa02 <= #1 ld_r ? text_in_r[063:056] ^ w2[31:24] : sa02_next;
always @(posedge clk)          sa31 <= #1 ld_r ? text_in_r[071:064] ^ w1[07:00] : sa31_next;
always @(posedge clk)          sa21 <= #1 ld_r ? text_in_r[079:072] ^ w1[15:08] : sa21_next;
always @(posedge clk)          sa11 <= #1 ld_r ? text_in_r[087:080] ^ w1[23:16] : sa11_next;
always @(posedge clk)          sa01 <= #1 ld_r ? text_in_r[095:088] ^ w1[31:24] : sa01_next;
always @(posedge clk)          sa30 <= #1 ld_r ? text_in_r[103:096] ^ w0[07:00] : sa30_next;
always @(posedge clk)          sa20 <= #1 ld_r ? text_in_r[111:104] ^ w0[15:08] : sa20_next;
always @(posedge clk)          sa10 <= #1 ld_r ? text_in_r[119:112] ^ w0[23:16] : sa10_next;
always @(posedge clk)          sa00 <= #1 ld_r ? text_in_r[127:120] ^ w0[31:24] : sa00_next;

////////////////////////////////////////////////////////////////////
//
// Round Permutations
//

assign sa00_sr = sa00_sub;
assign sa01_sr = sa01_sub;
assign sa02_sr = sa02_sub;
assign sa03_sr = sa03_sub;
assign sa10_sr = sa11_sub;
assign sa11_sr = sa12_sub;
assign sa12_sr = sa13_sub;
assign sa13_sr = sa10_sub;
assign sa20_sr = sa22_sub;
assign sa21_sr = sa23_sub;
assign sa22_sr = sa20_sub;
assign sa23_sr = sa21_sub;
assign sa30_sr = sa33_sub;
assign sa31_sr = sa30_sub;
assign sa32_sr = sa31_sub;
assign sa33_sr = sa32_sub;
assign {sa00_mc, sa10_mc, sa20_mc, sa30_mc}  = mix_col(sa00_sr,sa10_sr,sa20_sr,sa30_sr);
assign {sa01_mc, sa11_mc, sa21_mc, sa31_mc}  = mix_col(sa01_sr,sa11_sr,sa21_sr,sa31_sr);
assign {sa02_mc, sa12_mc, sa22_mc, sa32_mc}  = mix_col(sa02_sr,sa12_sr,sa22_sr,sa32_sr);
assign {sa03_mc, sa13_mc, sa23_mc, sa33_mc}  = mix_col(sa03_sr,sa13_sr,sa23_sr,sa33_sr);
assign sa00_next = sa00_mc ^ w0[31:24];
assign sa01_next = sa01_mc ^ w1[31:24];
assign sa02_next = sa02_mc ^ w2[31:24];
assign sa03_next = sa03_mc ^ w3[31:24];
assign sa10_next = sa10_mc ^ w0[23:16];
assign sa11_next = sa11_mc ^ w1[23:16];
assign sa12_next = sa12_mc ^ w2[23:16];
assign sa13_next = sa13_mc ^ w3[23:16];
assign sa20_next = sa20_mc ^ w0[15:08];
assign sa21_next = sa21_mc ^ w1[15:08];
assign sa22_next = sa22_mc ^ w2[15:08];
assign sa23_next = sa23_mc ^ w3[15:08];
assign sa30_next = sa30_mc ^ w0[07:00];
assign sa31_next = sa31_mc ^ w1[07:00];
assign sa32_next = sa32_mc ^ w2[07:00];
assign sa33_next = sa33_mc ^ w3[07:00];

////////////////////////////////////////////////////////////////////
//
// Final text output
//

always @(posedge clk) text_out[127:120] <= #1 sa00_sr ^ w0[31:24];
always @(posedge clk) text_out[095:088] <= #1 sa01_sr ^ w1[31:24];
always @(posedge clk) text_out[063:056] <= #1 sa02_sr ^ w2[31:24];
always @(posedge clk) text_out[031:024] <= #1 sa03_sr ^ w3[31:24];
always @(posedge clk) text_out[119:112] <= #1 sa10_sr ^ w0[23:16];
always @(posedge clk) text_out[087:080] <= #1 sa11_sr ^ w1[23:16];
always @(posedge clk) text_out[055:048] <= #1 sa12_sr ^ w2[23:16];
always @(posedge clk) text_out[023:016] <= #1 sa13_sr ^ w3[23:16];
always @(posedge clk) text_out[111:104] <= #1 sa20_sr ^ w0[15:08];
always @(posedge clk) text_out[079:072] <= #1 sa21_sr ^ w1[15:08];
always @(posedge clk) text_out[047:040] <= #1 sa22_sr ^ w2[15:08];
always @(posedge clk) text_out[015:008] <= #1 sa23_sr ^ w3[15:08];
always @(posedge clk) text_out[103:096] <= #1 sa30_sr ^ w0[07:00];
always @(posedge clk) text_out[071:064] <= #1 sa31_sr ^ w1[07:00];
always @(posedge clk) text_out[039:032] <= #1 sa32_sr ^ w2[07:00];
always @(posedge clk) text_out[007:000] <= #1 sa33_sr ^ w3[07:00];

////////////////////////////////////////////////////////////////////
//
```

```
// Generic Functions
//

function [31:0] mix_col;
input     [7:0]     s0,s1,s2,s3;
reg       [7:0]     s0_o,s1_o,s2_o,s3_o;
begin
mix_col[31:24]=xtime(s0)^xtime(s1)^s1^s2^s3;
mix_col[23:16]=s0^xtime(s1)^xtime(s2)^s2^s3;
mix_col[15:08]=s0^s1^xtime(s2)^xtime(s3)^s3;
mix_col[07:00]=xtime(s0)^s0^s1^s2^xtime(s3);
end
endfunction

function [7:0] xtime;
input [7:0] b; xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction

////////////////////////////////////////////////////////////////////
//
// Modules
//

aes_key_expand_128 u0(
          .clk(               clk       ),
          .kld(               ld        ),
          .key(               key       ),
          .wo_0(              w0        ),
          .wo_1(              w1        ),
          .wo_2(              w2        ),
          .wo_3(              w3        ));

aes_sbox us00(      .a(       sa00      ), .d(    sa00_sub  ));
aes_sbox us01(      .a(       sa01      ), .d(    sa01_sub  ));
aes_sbox us02(      .a(       sa02      ), .d(    sa02_sub  ));
aes_sbox us03(      .a(       sa03      ), .d(    sa03_sub  ));
aes_sbox us10(      .a(       sa10      ), .d(    sa10_sub  ));
aes_sbox us11(      .a(       sa11      ), .d(    sa11_sub  ));
aes_sbox us12(      .a(       sa12      ), .d(    sa12_sub  ));
aes_sbox us13(      .a(       sa13      ), .d(    sa13_sub  ));
aes_sbox us20(      .a(       sa20      ), .d(    sa20_sub  ));
aes_sbox us21(      .a(       sa21      ), .d(    sa21_sub  ));
aes_sbox us22(      .a(       sa22      ), .d(    sa22_sub  ));
aes_sbox us23(      .a(       sa23      ), .d(    sa23_sub  ));
aes_sbox us30(      .a(       sa30      ), .d(    sa30_sub  ));
aes_sbox us31(      .a(       sa31      ), .d(    sa31_sub  ));
aes_sbox us32(      .a(       sa32      ), .d(    sa32_sub  ));
aes_sbox us33(      .a(       sa33      ), .d(    sa33_sub  ));

endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES Inverse Cipher Top Level                              ////
////                                                            ////
////                                                            ////
////  Author: Rudolf Usselmann                                  ////
////          rudi@asics.ws                                     ////
////                                                            ////
////                                                            ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/ ////
////                                                            ////
////////////////////////////////////////////////////////////////////
////                                                            ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                   ////
////                         www.asics.ws                       ////
////                         rudi@asics.ws                      ////
////                                                            ////
//// This source file may be used and distributed without       ////
//// restriction provided that this copyright statement is not  ////
//// removed from the file and that any derivative work contains ////
//// the original copyright notice and the associated disclaimer.////
////                                                            ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY    ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR     ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES   ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE  ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR       ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE        ////
//// POSSIBILITY OF SUCH DAMAGE.                                ////
////                                                            ////
////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_inv_cipher_top.v,v 1.1.1.1 2002/11/09 11:22:53 rudi Exp $
//
//  $Date: 2002/11/09 11:22:53 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_inv_cipher_top.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:53  rudi
//               Initial Checkin
//
//
//
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_inv_cipher_top(clk, rst, kld, ld, done, key, text_in, text_out );
input               clk, rst;
input               kld, ld;
output              done;
input     [127:0]   key;
input     [127:0]   text_in;
output    [127:0]   text_out;

//////////////////////////////////////////////////////////////////
//
// Local Wires
//

wire      [31:0]    wk0, wk1, wk2, wk3;
reg       [31:0]    w0, w1, w2, w3;
reg       [127:0]   text_in_r;
reg       [127:0]   text_out;
reg       [7:0]     sa00, sa01, sa02, sa03;
reg       [7:0]     sa10, sa11, sa12, sa13;
reg       [7:0]     sa20, sa21, sa22, sa23;
reg       [7:0]     sa30, sa31, sa32, sa33;
wire      [7:0]     sa00_next, sa01_next, sa02_next, sa03_next;
wire      [7:0]     sa10_next, sa11_next, sa12_next, sa13_next;
wire      [7:0]     sa20_next, sa21_next, sa22_next, sa23_next;
wire      [7:0]     sa30_next, sa31_next, sa32_next, sa33_next;
wire      [7:0]     sa00_sub, sa01_sub, sa02_sub, sa03_sub;
wire      [7:0]     sa10_sub, sa11_sub, sa12_sub, sa13_sub;
wire      [7:0]     sa20_sub, sa21_sub, sa22_sub, sa23_sub;
wire      [7:0]     sa30_sub, sa31_sub, sa32_sub, sa33_sub;
wire      [7:0]     sa00_sr, sa01_sr, sa02_sr, sa03_sr;
wire      [7:0]     sa10_sr, sa11_sr, sa12_sr, sa13_sr;
wire      [7:0]     sa20_sr, sa21_sr, sa22_sr, sa23_sr;
wire      [7:0]     sa30_sr, sa31_sr, sa32_sr, sa33_sr;
wire      [7:0]     sa00_ark, sa01_ark, sa02_ark, sa03_ark;
wire      [7:0]     sa10_ark, sa11_ark, sa12_ark, sa13_ark;
wire      [7:0]     sa20_ark, sa21_ark, sa22_ark, sa23_ark;
wire      [7:0]     sa30_ark, sa31_ark, sa32_ark, sa33_ark;
reg                 ld_r, go, done;
reg       [3:0]     dcnt;

//////////////////////////////////////////////////////////////////
```

```
//
// Misc Logic
//

always @(posedge clk)
          if(!rst)  dcnt <= #1 4'h0;
          else
          if(done)  dcnt <= #1 4'h0;
          else
          if(ld)              dcnt <= #1 4'hl;
          else
          if(go)              dcnt <= #1 dcnt + 4'hl;

always @(posedge clk)         done <= #1 (dcnt==4'hb) & !ld;

always @(posedge clk)
          if(!rst)  go <= #1 1'b0;
          else
          if(ld)              go <= #1 1'bl;
          else
          if(done)  go <= #1 1'b0;

always @(posedge clk)         if(ld)    text_in_r <= #1 text_in;

always @(posedge clk)         ld_r <= #1 ld;

////////////////////////////////////////////////////////////////////
//
// Initial Permutation
//

always @(posedge clk)         sa33 <= #1 ld_r ? text_in_r[007:000] ^ w3[07:00] : sa33_next;
always @(posedge clk)         sa23 <= #1 ld_r ? text_in_r[015:008] ^ w3[15:08] : sa23_next;
always @(posedge clk)         sa13 <= #1 ld_r ? text_in_r[023:016] ^ w3[23:16] : sa13_next;
always @(posedge clk)         sa03 <= #1 ld_r ? text_in_r[031:024] ^ w3[31:24] : sa03_next;
always @(posedge clk)         sa32 <= #1 ld_r ? text_in_r[039:032] ^ w2[07:00] : sa32_next;
always @(posedge clk)         sa22 <= #1 ld_r ? text_in_r[047:040] ^ w2[15:08] : sa22_next;
always @(posedge clk)         sa12 <= #1 ld_r ? text_in_r[055:048] ^ w2[23:16] : sa12_next;
always @(posedge clk)         sa02 <= #1 ld_r ? text_in_r[063:056] ^ w2[31:24] : sa02_next;
always @(posedge clk)         sa31 <= #1 ld_r ? text_in_r[071:064] ^ w1[07:00] : sa31_next;
always @(posedge clk)         sa21 <= #1 ld_r ? text_in_r[079:072] ^ w1[15:08] : sa21_next;
always @(posedge clk)         sa11 <= #1 ld_r ? text_in_r[087:080] ^ w1[23:16] : sa11_next;
always @(posedge clk)         sa01 <= #1 ld_r ? text_in_r[095:088] ^ w1[31:24] : sa01_next;
always @(posedge clk)         sa30 <= #1 ld_r ? text_in_r[103:096] ^ w0[07:00] : sa30_next;
always @(posedge clk)         sa20 <= #1 ld_r ? text_in_r[111:104] ^ w0[15:08] : sa20_next;
always @(posedge clk)         sa10 <= #1 ld_r ? text_in_r[119:112] ^ w0[23:16] : sa10_next;
always @(posedge clk)         sa00 <= #1 ld_r ? text_in_r[127:120] ^ w0[31:24] : sa00_next;

////////////////////////////////////////////////////////////////////
//
// Round Permutations
//

assign sa00_sr = sa00;
assign sa01_sr = sa01;
assign sa02_sr = sa02;
assign sa03_sr = sa03;
assign sa10_sr = sa13;
assign sa11_sr = sa10;
assign sa12_sr = sa11;
assign sa13_sr = sa12;
assign sa20_sr = sa22;
assign sa21_sr = sa23;
assign sa22_sr = sa20;
assign sa23_sr = sa21;
assign sa30_sr = sa31;
assign sa31_sr = sa32;
assign sa32_sr = sa33;
assign sa33_sr = sa30;
assign sa00_ark = sa00_sub ^ w0[31:24];
assign sa01_ark = sa01_sub ^ w1[31:24];
assign sa02_ark = sa02_sub ^ w2[31:24];
assign sa03_ark = sa03_sub ^ w3[31:24];
assign sa10_ark = sa10_sub ^ w0[23:16];
assign sa11_ark = sa11_sub ^ w1[23:16];
assign sa12_ark = sa12_sub ^ w2[23:16];
assign sa13_ark = sa13_sub ^ w3[23:16];
assign sa20_ark = sa20_sub ^ w0[15:08];
assign sa21_ark = sa21_sub ^ w1[15:08];
assign sa22_ark = sa22_sub ^ w2[15:08];
assign sa23_ark = sa23_sub ^ w3[15:08];
assign sa30_ark = sa30_sub ^ w0[07:00];
assign sa31_ark = sa31_sub ^ w1[07:00];
assign sa32_ark = sa32_sub ^ w2[07:00];
assign sa33_ark = sa33_sub ^ w3[07:00];
assign {sa00_next, sa10_next, sa20_next, sa30_next} = inv_mix_col(sa00_ark,sa10_ark,sa20_ark,sa30_ark);
assign {sa01_next, sa11_next, sa21_next, sa31_next} = inv_mix_col(sa01_ark,sa11_ark,sa21_ark,sa31_ark);
assign {sa02_next, sa12_next, sa22_next, sa32_next} = inv_mix_col(sa02_ark,sa12_ark,sa22_ark,sa32_ark);
assign {sa03_next, sa13_next, sa23_next, sa33_next} = inv_mix_col(sa03_ark,sa13_ark,sa23_ark,sa33_ark);

////////////////////////////////////////////////////////////////////
//
// Final Text Output
//

always @(posedge clk) text_out[127:120] <= #1 sa00_ark;
always @(posedge clk) text_out[095:088] <= #1 sa01_ark;
always @(posedge clk) text_out[063:056] <= #1 sa02_ark;
always @(posedge clk) text_out[031:024] <= #1 sa03_ark;
always @(posedge clk) text_out[119:112] <= #1 sa10_ark;
always @(posedge clk) text_out[087:080] <= #1 sa11_ark;
always @(posedge clk) text_out[055:048] <= #1 sa12_ark;
```

```
always @(posedge clk) text_out[023:016] <= #1 sa13_ark;
always @(posedge clk) text_out[111:104] <= #1 sa20_ark;
always @(posedge clk) text_out[079:072] <= #1 sa21_ark;
always @(posedge clk) text_out[047:040] <= #1 sa22_ark;
always @(posedge clk) text_out[015:008] <= #1 sa23_ark;
always @(posedge clk) text_out[103:096] <= #1 sa30_ark;
always @(posedge clk) text_out[071:064] <= #1 sa31_ark;
always @(posedge clk) text_out[039:032] <= #1 sa32_ark;
always @(posedge clk) text_out[007:000] <= #1 sa33_ark;

//////////////////////////////////////////////////////////////////////
//
// Generic Functions
//

function [31:0] inv_mix_col;
input     [7:0]     s0,s1,s2,s3;
begin
inv_mix_col[31:24]=pmul_e(s0)^pmul_b(s1)^pmul_d(s2)^pmul_9(s3);
inv_mix_col[23:16]=pmul_9(s0)^pmul_e(s1)^pmul_b(s2)^pmul_d(s3);
inv_mix_col[15:08]=pmul_d(s0)^pmul_9(s1)^pmul_e(s2)^pmul_b(s3);
inv_mix_col[07:00]=pmul_b(s0)^pmul_d(s1)^pmul_9(s2)^pmul_e(s3);
end
endfunction

// Some synthesis tools don't like xtime being called recursevly ...
function [7:0] pmul_e;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_e=eight^four^two;
end
endfunction

function [7:0] pmul_9;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_9=eight^b;
end
endfunction

function [7:0] pmul_d;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_d=eight^four^b;
end
endfunction

function [7:0] pmul_b;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_b=eight^two^b;
end
endfunction

function [7:0] xtime;
input [7:0] b;xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction

//////////////////////////////////////////////////////////////////////
//
// Key Buffer
//

reg       [127:0]   kb[10:0];
reg       [3:0]     kcnt;
reg                 kdone;
reg                 kb_ld;

always @(posedge clk)
          if(!rst)  kcnt <= #1 4'ha;
          else
          if(kld)             kcnt <= #1 4'ha;
          else
          if(kb_ld) kcnt <= #1 kcnt - 4'h1;

always @(posedge clk)
          if(!rst)  kb_ld <= #1 1'b0;
          else
          if(kld)             kb_ld <= #1 1'b1;
          else
          if(kcnt==4'h0)      kb_ld <= #1 1'b0;

always @(posedge clk)         kdone <= #1 (kcnt==4'h0) & !kld;
always @(posedge clk)         if(kb_ld) kb[kcnt] <= #1 {wk3, wk2, wk1, wk0};
always @(posedge clk)         {w3, w2, w1, w0} <= #1 kb[dcnt];

//////////////////////////////////////////////////////////////////////
//
// Modules
//

aes_key_expand_128 u0(
          .clk(               clk       ),
          .kld(               kld       ),
          .key(               key       ),
          .wo_0(              wk0       ),
          .wo_1(              wk1       ),
          .wo_2(              wk2       ),
```

```
          .wo_3(              wk3       ));

aes_inv_sbox us00(  .a(       sa00_sr   ),        .d(       sa00_sub  ));
aes_inv_sbox us01(  .a(       sa01_sr   ),        .d(       sa01_sub  ));
aes_inv_sbox us02(  .a(       sa02_sr   ),        .d(       sa02_sub  ));
aes_inv_sbox us03(  .a(       sa03_sr   ),        .d(       sa03_sub  ));
aes_inv_sbox us10(  .a(       sa10_sr   ),        .d(       sa10_sub  ));
aes_inv_sbox us11(  .a(       sa11_sr   ),        .d(       sa11_sub  ));
aes_inv_sbox us12(  .a(       sa12_sr   ),        .d(       sa12_sub  ));
aes_inv_sbox us13(  .a(       sa13_sr   ),        .d(       sa13_sub  ));
aes_inv_sbox us20(  .a(       sa20_sr   ),        .d(       sa20_sub  ));
aes_inv_sbox us21(  .a(       sa21_sr   ),        .d(       sa21_sub  ));
aes_inv_sbox us22(  .a(       sa22_sr   ),        .d(       sa22_sub  ));
aes_inv_sbox us23(  .a(       sa23_sr   ),        .d(       sa23_sub  ));
aes_inv_sbox us30(  .a(       sa30_sr   ),        .d(       sa30_sub  ));
aes_inv_sbox us31(  .a(       sa31_sr   ),        .d(       sa31_sub  ));
aes_inv_sbox us32(  .a(       sa32_sr   ),        .d(       sa32_sub  ));
aes_inv_sbox us33(  .a(       sa33_sr   ),        .d(       sa33_sub  ));

endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES SBOX (ROM)                                            ////
////                                                            ////
////                                                            ////
////  Author: Rudolf Usselmann                                  ////
////          rudi@asics.ws                                     ////
////                                                            ////
////                                                            ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/ ////
////                                                            ////
////////////////////////////////////////////////////////////////////
////                                                            ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                   ////
////                         www.asics.ws                       ////
////                         rudi@asics.ws                      ////
////                                                            ////
//// This source file may be used and distributed without       ////
//// restriction provided that this copyright statement is not  ////
//// removed from the file and that any derivative work contains ////
//// the original copyright notice and the associated disclaimer.////
////                                                            ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY    ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR     ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES   ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE  ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR       ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE        ////
//// POSSIBILITY OF SUCH DAMAGE.                                ////
////                                                            ////
////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_sbox.v,v 1.1.1.1 2002/11/09 11:22:38 rudi Exp $
//
//  $Date: 2002/11/09 11:22:38 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_sbox.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:38  rudi
//               Initial Checkin
//
//
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_sbox(a,d);
input     [7:0]     a;
output    [7:0]     d;
reg       [7:0]     d;

always @(a)
          case(a)             // synopsys full_case parallel_case
             8'h00: d=8'h63;
             8'h01: d=8'h7c;
             8'h02: d=8'h77;
             8'h03: d=8'h7b;
             8'h04: d=8'hf2;
             8'h05: d=8'h6b;
             8'h06: d=8'h6f;
             8'h07: d=8'hc5;
             8'h08: d=8'h30;
             8'h09: d=8'h01;
             8'h0a: d=8'h67;
             8'h0b: d=8'h2b;
             8'h0c: d=8'hfe;
             8'h0d: d=8'hd7;
             8'h0e: d=8'hab;
             8'h0f: d=8'h76;
             8'h10: d=8'hca;
             8'h11: d=8'h82;
             8'h12: d=8'hc9;
             8'h13: d=8'h7d;
             8'h14: d=8'hfa;
             8'h15: d=8'h59;
             8'h16: d=8'h47;
             8'h17: d=8'hf0;
             8'h18: d=8'had;
             8'h19: d=8'hd4;
             8'h1a: d=8'ha2;
             8'h1b: d=8'haf;
             8'h1c: d=8'h9c;
             8'h1d: d=8'ha4;
             8'h1e: d=8'h72;
             8'h1f: d=8'hc0;
             8'h20: d=8'hb7;
             8'h21: d=8'hfd;
```

```
8'h22: d=8'h93;
8'h23: d=8'h26;
8'h24: d=8'h36;
8'h25: d=8'h3f;
8'h26: d=8'hf7;
8'h27: d=8'hcc;
8'h28: d=8'h34;
8'h29: d=8'ha5;
8'h2a: d=8'he5;
8'h2b: d=8'hf1;
8'h2c: d=8'h71;
8'h2d: d=8'hd8;
8'h2e: d=8'h31;
8'h2f: d=8'h15;
8'h30: d=8'h04;
8'h31: d=8'hc7;
8'h32: d=8'h23;
8'h33: d=8'hc3;
8'h34: d=8'h18;
8'h35: d=8'h96;
8'h36: d=8'h05;
8'h37: d=8'h9a;
8'h38: d=8'h07;
8'h39: d=8'h12;
8'h3a: d=8'h80;
8'h3b: d=8'he2;
8'h3c: d=8'heb;
8'h3d: d=8'h27;
8'h3e: d=8'hb2;
8'h3f: d=8'h75;
8'h40: d=8'h09;
8'h41: d=8'h83;
8'h42: d=8'h2c;
8'h43: d=8'h1a;
8'h44: d=8'h1b;
8'h45: d=8'h6e;
8'h46: d=8'h5a;
8'h47: d=8'ha0;
8'h48: d=8'h52;
8'h49: d=8'h3b;
8'h4a: d=8'hd6;
8'h4b: d=8'hb3;
8'h4c: d=8'h29;
8'h4d: d=8'he3;
8'h4e: d=8'h2f;
8'h4f: d=8'h84;
8'h50: d=8'h53;
8'h51: d=8'hd1;
8'h52: d=8'h00;
8'h53: d=8'hed;
8'h54: d=8'h20;
8'h55: d=8'hfc;
8'h56: d=8'hb1;
8'h57: d=8'h5b;
8'h58: d=8'h6a;
8'h59: d=8'hcb;
8'h5a: d=8'hbe;
8'h5b: d=8'h39;
8'h5c: d=8'h4a;
8'h5d: d=8'h4c;
8'h5e: d=8'h58;
8'h5f: d=8'hcf;
8'h60: d=8'hd0;
8'h61: d=8'hef;
8'h62: d=8'haa;
8'h63: d=8'hfb;
8'h64: d=8'h43;
8'h65: d=8'h4d;
8'h66: d=8'h33;
8'h67: d=8'h85;
8'h68: d=8'h45;
8'h69: d=8'hf9;
8'h6a: d=8'h02;
8'h6b: d=8'h7f;
8'h6c: d=8'h50;
8'h6d: d=8'h3c;
8'h6e: d=8'h9f;
8'h6f: d=8'ha8;
8'h70: d=8'h51;
8'h71: d=8'ha3;
8'h72: d=8'h40;
8'h73: d=8'h8f;
8'h74: d=8'h92;
8'h75: d=8'h9d;
8'h76: d=8'h38;
8'h77: d=8'hf5;
8'h78: d=8'hbc;
8'h79: d=8'hb6;
8'h7a: d=8'hda;
8'h7b: d=8'h21;
8'h7c: d=8'h10;
8'h7d: d=8'hff;
8'h7e: d=8'hf3;
8'h7f: d=8'hd2;
8'h80: d=8'hcd;
8'h81: d=8'h0c;
8'h82: d=8'h13;
8'h83: d=8'hec;
8'h84: d=8'h5f;
8'h85: d=8'h97;
8'h86: d=8'h44;
8'h87: d=8'h17;
```

```
8'h88: d=8'hc4;
8'h89: d=8'ha7;
8'h8a: d=8'h7e;
8'h8b: d=8'h3d;
8'h8c: d=8'h64;
8'h8d: d=8'h5d;
8'h8e: d=8'h19;
8'h8f: d=8'h73;
8'h90: d=8'h60;
8'h91: d=8'h81;
8'h92: d=8'h4f;
8'h93: d=8'hdc;
8'h94: d=8'h22;
8'h95: d=8'h2a;
8'h96: d=8'h90;
8'h97: d=8'h88;
8'h98: d=8'h46;
8'h99: d=8'hee;
8'h9a: d=8'hb8;
8'h9b: d=8'h14;
8'h9c: d=8'hde;
8'h9d: d=8'h5e;
8'h9e: d=8'h0b;
8'h9f: d=8'hdb;
8'ha0: d=8'he0;
8'ha1: d=8'h32;
8'ha2: d=8'h3a;
8'ha3: d=8'h0a;
8'ha4: d=8'h49;
8'ha5: d=8'h06;
8'ha6: d=8'h24;
8'ha7: d=8'h5c;
8'ha8: d=8'hc2;
8'ha9: d=8'hd3;
8'haa: d=8'hac;
8'hab: d=8'h62;
8'hac: d=8'h91;
8'had: d=8'h95;
8'hae: d=8'he4;
8'haf: d=8'h79;
8'hb0: d=8'he7;
8'hb1: d=8'hc8;
8'hb2: d=8'h37;
8'hb3: d=8'h6d;
8'hb4: d=8'h8d;
8'hb5: d=8'hd5;
8'hb6: d=8'h4e;
8'hb7: d=8'ha9;
8'hb8: d=8'h6c;
8'hb9: d=8'h56;
8'hba: d=8'hf4;
8'hbb: d=8'hea;
8'hbc: d=8'h65;
8'hbd: d=8'h7a;
8'hbe: d=8'hae;
8'hbf: d=8'h08;
8'hc0: d=8'hba;
8'hc1: d=8'h78;
8'hc2: d=8'h25;
8'hc3: d=8'h2e;
8'hc4: d=8'h1c;
8'hc5: d=8'ha6;
8'hc6: d=8'hb4;
8'hc7: d=8'hc6;
8'hc8: d=8'he8;
8'hc9: d=8'hdd;
8'hca: d=8'h74;
8'hcb: d=8'h1f;
8'hcc: d=8'h4b;
8'hcd: d=8'hbd;
8'hce: d=8'h8b;
8'hcf: d=8'h8a;
8'hd0: d=8'h70;
8'hd1: d=8'h3e;
8'hd2: d=8'hb5;
8'hd3: d=8'h66;
8'hd4: d=8'h48;
8'hd5: d=8'h03;
8'hd6: d=8'hf6;
8'hd7: d=8'h0e;
8'hd8: d=8'h61;
8'hd9: d=8'h35;
8'hda: d=8'h57;
8'hdb: d=8'hb9;
8'hdc: d=8'h86;
8'hdd: d=8'hc1;
8'hde: d=8'h1d;
8'hdf: d=8'h9e;
8'he0: d=8'he1;
8'he1: d=8'hf8;
8'he2: d=8'h98;
8'he3: d=8'h11;
8'he4: d=8'h69;
8'he5: d=8'hd9;
8'he6: d=8'h8e;
8'he7: d=8'h94;
8'he8: d=8'h9b;
8'he9: d=8'h1e;
8'hea: d=8'h87;
8'heb: d=8'he9;
8'hec: d=8'hce;
8'hed: d=8'h55;
```

```
            8'hee: d=8'h28;
            8'hef: d=8'hdf;
            8'hf0: d=8'h8c;
            8'hf1: d=8'ha1;
            8'hf2: d=8'h89;
            8'hf3: d=8'h0d;
            8'hf4: d=8'hbf;
            8'hf5: d=8'he6;
            8'hf6: d=8'h42;
            8'hf7: d=8'h68;
            8'hf8: d=8'h41;
            8'hf9: d=8'h99;
            8'hfa: d=8'h2d;
            8'hfb: d=8'h0f;
            8'hfc: d=8'hb0;
            8'hfd: d=8'h54;
            8'hfe: d=8'hbb;
            8'hff: d=8'h16;
          endcase

endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES Inverse SBOX (ROM)                                    ////
////                                                            ////
////                                                            ////
////  Author: Rudolf Usselmann                                  ////
////          rudi@asics.ws                                     ////
////                                                            ////
////                                                            ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/ ////
////                                                            ////
////////////////////////////////////////////////////////////////////
////                                                            ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                   ////
////                         www.asics.ws                       ////
////                         rudi@asics.ws                      ////
////                                                            ////
//// This source file may be used and distributed without       ////
//// restriction provided that this copyright statement is not  ////
//// removed from the file and that any derivative work contains ////
//// the original copyright notice and the associated disclaimer.////
////                                                            ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY    ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR     ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES   ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE  ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR       ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE        ////
//// POSSIBILITY OF SUCH DAMAGE.                                ////
////                                                            ////
////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_inv_sbox.v,v 1.1.1.1 2002/11/09 11:22:55 rudi Exp $
//
//  $Date: 2002/11/09 11:22:55 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_inv_sbox.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:55  rudi
//               Initial Checkin
//
//
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_inv_sbox(a,d);
input     [7:0]     a;
output    [7:0]     d;
reg       [7:0]     d;

always @(a)
          case(a)             // synopsys full_case parallel_case
             8'h00: d=8'h52;
             8'h01: d=8'h09;
             8'h02: d=8'h6a;
             8'h03: d=8'hd5;
             8'h04: d=8'h30;
             8'h05: d=8'h36;
             8'h06: d=8'ha5;
             8'h07: d=8'h38;
             8'h08: d=8'hbf;
             8'h09: d=8'h40;
             8'h0a: d=8'ha3;
             8'h0b: d=8'h9e;
             8'h0c: d=8'h81;
             8'h0d: d=8'hf3;
             8'h0e: d=8'hd7;
             8'h0f: d=8'hfb;
             8'h10: d=8'h7c;
             8'h11: d=8'he3;
             8'h12: d=8'h39;
             8'h13: d=8'h82;
             8'h14: d=8'h9b;
             8'h15: d=8'h2f;
             8'h16: d=8'hff;
             8'h17: d=8'h87;
             8'h18: d=8'h34;
             8'h19: d=8'h8e;
             8'h1a: d=8'h43;
             8'h1b: d=8'h44;
             8'h1c: d=8'hc4;
             8'h1d: d=8'hde;
             8'h1e: d=8'he9;
             8'h1f: d=8'hcb;
             8'h20: d=8'h54;
             8'h21: d=8'h7b;
```

```
8'h22: d=8'h94;
8'h23: d=8'h32;
8'h24: d=8'ha6;
8'h25: d=8'hc2;
8'h26: d=8'h23;
8'h27: d=8'h3d;
8'h28: d=8'hee;
8'h29: d=8'h4c;
8'h2a: d=8'h95;
8'h2b: d=8'h0b;
8'h2c: d=8'h42;
8'h2d: d=8'hfa;
8'h2e: d=8'hc3;
8'h2f: d=8'h4e;
8'h30: d=8'h08;
8'h31: d=8'h2e;
8'h32: d=8'ha1;
8'h33: d=8'h66;
8'h34: d=8'h28;
8'h35: d=8'hd9;
8'h36: d=8'h24;
8'h37: d=8'hb2;
8'h38: d=8'h76;
8'h39: d=8'h5b;
8'h3a: d=8'ha2;
8'h3b: d=8'h49;
8'h3c: d=8'h6d;
8'h3d: d=8'h8b;
8'h3e: d=8'hd1;
8'h3f: d=8'h25;
8'h40: d=8'h72;
8'h41: d=8'hf8;
8'h42: d=8'hf6;
8'h43: d=8'h64;
8'h44: d=8'h86;
8'h45: d=8'h68;
8'h46: d=8'h98;
8'h47: d=8'h16;
8'h48: d=8'hd4;
8'h49: d=8'ha4;
8'h4a: d=8'h5c;
8'h4b: d=8'hcc;
8'h4c: d=8'h5d;
8'h4d: d=8'h65;
8'h4e: d=8'hb6;
8'h4f: d=8'h92;
8'h50: d=8'h6c;
8'h51: d=8'h70;
8'h52: d=8'h48;
8'h53: d=8'h50;
8'h54: d=8'hfd;
8'h55: d=8'hed;
8'h56: d=8'hb9;
8'h57: d=8'hda;
8'h58: d=8'h5e;
8'h59: d=8'h15;
8'h5a: d=8'h46;
8'h5b: d=8'h57;
8'h5c: d=8'ha7;
8'h5d: d=8'h8d;
8'h5e: d=8'h9d;
8'h5f: d=8'h84;
8'h60: d=8'h90;
8'h61: d=8'hd8;
8'h62: d=8'hab;
8'h63: d=8'h00;
8'h64: d=8'h8c;
8'h65: d=8'hbc;
8'h66: d=8'hd3;
8'h67: d=8'h0a;
8'h68: d=8'hf7;
8'h69: d=8'he4;
8'h6a: d=8'h58;
8'h6b: d=8'h05;
8'h6c: d=8'hb8;
8'h6d: d=8'hb3;
8'h6e: d=8'h45;
8'h6f: d=8'h06;
8'h70: d=8'hd0;
8'h71: d=8'h2c;
8'h72: d=8'h1e;
8'h73: d=8'h8f;
8'h74: d=8'hca;
8'h75: d=8'h3f;
8'h76: d=8'h0f;
8'h77: d=8'h02;
8'h78: d=8'hc1;
8'h79: d=8'haf;
8'h7a: d=8'hbd;
8'h7b: d=8'h03;
8'h7c: d=8'h01;
8'h7d: d=8'h13;
8'h7e: d=8'h8a;
8'h7f: d=8'h6b;
8'h80: d=8'h3a;
8'h81: d=8'h91;
8'h82: d=8'h11;
8'h83: d=8'h41;
8'h84: d=8'h4f;
8'h85: d=8'h67;
8'h86: d=8'hdc;
8'h87: d=8'hea;
```

```
8'h88: d=8'h97;
8'h89: d=8'hf2;
8'h8a: d=8'hcf;
8'h8b: d=8'hce;
8'h8c: d=8'hf0;
8'h8d: d=8'hb4;
8'h8e: d=8'he6;
8'h8f: d=8'h73;
8'h90: d=8'h96;
8'h91: d=8'hac;
8'h92: d=8'h74;
8'h93: d=8'h22;
8'h94: d=8'he7;
8'h95: d=8'had;
8'h96: d=8'h35;
8'h97: d=8'h85;
8'h98: d=8'he2;
8'h99: d=8'hf9;
8'h9a: d=8'h37;
8'h9b: d=8'he8;
8'h9c: d=8'h1c;
8'h9d: d=8'h75;
8'h9e: d=8'hdf;
8'h9f: d=8'h6e;
8'ha0: d=8'h47;
8'ha1: d=8'hf1;
8'ha2: d=8'h1a;
8'ha3: d=8'h71;
8'ha4: d=8'h1d;
8'ha5: d=8'h29;
8'ha6: d=8'hc5;
8'ha7: d=8'h89;
8'ha8: d=8'h6f;
8'ha9: d=8'hb7;
8'haa: d=8'h62;
8'hab: d=8'h0e;
8'hac: d=8'haa;
8'had: d=8'h18;
8'hae: d=8'hbe;
8'haf: d=8'h1b;
8'hb0: d=8'hfc;
8'hb1: d=8'h56;
8'hb2: d=8'h3e;
8'hb3: d=8'h4b;
8'hb4: d=8'hc6;
8'hb5: d=8'hd2;
8'hb6: d=8'h79;
8'hb7: d=8'h20;
8'hb8: d=8'h9a;
8'hb9: d=8'hdb;
8'hba: d=8'hc0;
8'hbb: d=8'hfe;
8'hbc: d=8'h78;
8'hbd: d=8'hcd;
8'hbe: d=8'h5a;
8'hbf: d=8'hf4;
8'hc0: d=8'h1f;
8'hc1: d=8'hdd;
8'hc2: d=8'ha8;
8'hc3: d=8'h33;
8'hc4: d=8'h88;
8'hc5: d=8'h07;
8'hc6: d=8'hc7;
8'hc7: d=8'h31;
8'hc8: d=8'hb1;
8'hc9: d=8'h12;
8'hca: d=8'h10;
8'hcb: d=8'h59;
8'hcc: d=8'h27;
8'hcd: d=8'h80;
8'hce: d=8'hec;
8'hcf: d=8'h5f;
8'hd0: d=8'h60;
8'hd1: d=8'h51;
8'hd2: d=8'h7f;
8'hd3: d=8'ha9;
8'hd4: d=8'h19;
8'hd5: d=8'hb5;
8'hd6: d=8'h4a;
8'hd7: d=8'h0d;
8'hd8: d=8'h2d;
8'hd9: d=8'he5;
8'hda: d=8'h7a;
8'hdb: d=8'h9f;
8'hdc: d=8'h93;
8'hdd: d=8'hc9;
8'hde: d=8'h9c;
8'hdf: d=8'hef;
8'he0: d=8'ha0;
8'he1: d=8'he0;
8'he2: d=8'h3b;
8'he3: d=8'h4d;
8'he4: d=8'hae;
8'he5: d=8'h2a;
8'he6: d=8'hf5;
8'he7: d=8'hb0;
8'he8: d=8'hc8;
8'he9: d=8'heb;
8'hea: d=8'hbb;
8'heb: d=8'h3c;
8'hec: d=8'h83;
8'hed: d=8'h53;
```

```
            8'hee: d=8'h99;
            8'hef: d=8'h61;
            8'hf0: d=8'h17;
            8'hf1: d=8'h2b;
            8'hf2: d=8'h04;
            8'hf3: d=8'h7e;
            8'hf4: d=8'hba;
            8'hf5: d=8'h77;
            8'hf6: d=8'hd6;
            8'hf7: d=8'h26;
            8'hf8: d=8'he1;
            8'hf9: d=8'h69;
            8'hfa: d=8'h14;
            8'hfb: d=8'h63;
            8'hfc: d=8'h55;
            8'hfd: d=8'h21;
            8'hfe: d=8'h0c;
            8'hff: d=8'h7d;
        endcase
endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES Key Expand Block (for 128 bit keys)                   ////
////                                                            ////
////                                                            ////
////  Author: Rudolf Usselmann                                  ////
////          rudi@asics.ws                                     ////
////                                                            ////
////                                                            ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/ ////
////                                                            ////
////////////////////////////////////////////////////////////////////
////                                                            ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                   ////
////                         www.asics.ws                       ////
////                         rudi@asics.ws                      ////
////                                                            ////
//// This source file may be used and distributed without       ////
//// restriction provided that this copyright statement is not  ////
//// removed from the file and that any derivative work contains ////
//// the original copyright notice and the associated disclaimer.////
////                                                            ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY    ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR     ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES   ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE  ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR       ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE        ////
//// POSSIBILITY OF SUCH DAMAGE.                                ////
////                                                            ////
////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_key_expand_128.v,v 1.1.1.1 2002/11/09 11:22:38 rudi Exp $
//
//  $Date: 2002/11/09 11:22:38 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_key_expand_128.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:38  rudi
//               Initial Checkin
//
//
//
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_key_expand_128(clk, kld, key, wo_0, wo_1, wo_2, wo_3);
input               clk;
input               kld;
input     [127:0]   key;
output    [31:0]    wo_0, wo_1, wo_2, wo_3;
reg       [31:0]    w[3:0];
wire      [31:0]    tmp_w;
wire      [31:0]    subword;
wire      [31:0]    rcon;

assign wo_0 = w[0];
assign wo_1 = w[1];
assign wo_2 = w[2];
assign wo_3 = w[3];
always @(posedge clk)         w[0] <= #1 kld ? key[127:096] : w[0]^subword^rcon;
always @(posedge clk)         w[1] <= #1 kld ? key[095:064] : w[0]^w[1]^subword^rcon;
always @(posedge clk)         w[2] <= #1 kld ? key[063:032] : w[0]^w[2]^w[1]^subword^rcon;
always @(posedge clk)         w[3] <= #1 kld ? key[031:000] : w[0]^w[3]^w[2]^w[1]^subword^rcon;
assign tmp_w = w[3];
aes_sbox u0(        .a(tmp_w[23:16]), .d(subword[31:24]));
aes_sbox u1(        .a(tmp_w[15:08]), .d(subword[23:16]));
aes_sbox u2(        .a(tmp_w[07:00]), .d(subword[15:08]));
aes_sbox u3(        .a(tmp_w[31:24]), .d(subword[07:00]));
aes_rcon r0(        .clk(clk), .kld(kld), .out(rcon));
endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES RCON Block                                            ////
////                                                            ////
////                                                            ////
////  Author: Rudolf Usselmann                                  ////
////          rudi@asics.ws                                     ////
////                                                            ////
////                                                            ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/ ////
////                                                            ////
////////////////////////////////////////////////////////////////////
////                                                            ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                   ////
////                         www.asics.ws                       ////
////                         rudi@asics.ws                      ////
////                                                            ////
//// This source file may be used and distributed without       ////
//// restriction provided that this copyright statement is not  ////
//// removed from the file and that any derivative work contains ////
//// the original copyright notice and the associated disclaimer.////
////                                                            ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY    ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED  ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS  ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR     ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,        ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES   ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE  ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR       ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE        ////
//// POSSIBILITY OF SUCH DAMAGE.                                ////
////                                                            ////
////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_rcon.v,v 1.1.1.1 2002/11/09 11:22:38 rudi Exp $
//
//  $Date: 2002/11/09 11:22:38 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_rcon.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:38  rudi
//               Initial Checkin
//
//
//
//
//
//
//

`include "c:\aes_modules\ori\timescale.v"

module aes_rcon(clk, kld, out);
input               clk;
input               kld;
output    [31:0]    out;
reg       [31:0]    out;
reg       [3:0]     rcnt;
wire      [3:0]     rcnt_next;

always @(posedge clk)
          if(kld)             out <= #1 32'h01_00_00_00;
          else                out <= #1 frcon(rcnt_next);

assign rcnt_next = rcnt + 4'h1;
always @(posedge clk)
          if(kld)             rcnt <= #1 4'h0;
          else                rcnt <= #1 rcnt_next;

function [31:0]     frcon;
input     [3:0]     i;
case(i)   // synopsys parallel_case
   4'h0: frcon=32'h01_00_00_00;
   4'h1: frcon=32'h02_00_00_00;
   4'h2: frcon=32'h04_00_00_00;
   4'h3: frcon=32'h08_00_00_00;
   4'h4: frcon=32'h10_00_00_00;
   4'h5: frcon=32'h20_00_00_00;
   4'h6: frcon=32'h40_00_00_00;
   4'h7: frcon=32'h80_00_00_00;
   4'h8: frcon=32'h1b_00_00_00;
   4'h9: frcon=32'h36_00_00_00;
   default: frcon=32'h00_00_00_00;
endcase
endfunction

endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES Top Layer for Combined Sbox Design                    ////
////  (for serial input of 64 bit per cycle)                    ////
////                                                            ////
////  Author: Lee Yi Lin                                        ////
////          leeyilin@yahoo.com                                ////
////                                                            ////
////                                                            ////
////  Composed for Final Year Project of EE Faculty, UTP              ////
////                                                            ////
////////////////////////////////////////////////////////////////////
//
//
//  $Date: 2004/4/01 $
//  $Revision: 1.0 $
//  $Author: Lee Yi Lin $
//
//  Change History:
//
//
//
//

`include "c:\aes_modules\new\timescale.v"


module aes_new64_top (clk, rst, option, start, get_output, key_in, text_in, text_out);

input clk, rst;
input option;
input start;
output get_output;
input [63:0] key_in;
input [63:0] text_in;
output [63:0] text_out;

reg get_output;
reg [63:0] text_out;

// local wires

reg [127:0] text_in_buf;
wire [127:0] text_out_buf;
reg [127:0] key_buf;
reg option_buf;
reg load_crypto, load_key;
reg run;
wire done;
reg complete;
reg [7:0] counter;
reg [7:0] cnt_temp, cnt_temp2;

always @(posedge clk)
          if(!rst)            counter <= #1 8'h00;
          else
          if(complete)        counter <= #1 8'b00;
          else
          if(start)           counter <= #1 8'h01;
          else
          if (run)            counter <= #1 counter + 8'h01;

always @(posedge clk)
          if(!rst)            run <= #1 1'b0;
          else
          if(start)           run <= #1 1'b1;
          else
          if(complete)        run <= #1 1'b0;

always @(posedge clk)
          if(start)           option_buf <= #1 option;

always @(posedge clk)
          if(!(|counter) & start)       text_in_buf[063:000] <= #1 text_in;
          else
          if(counter == 8'h01)                    text_in_buf[127:064] <= #1 text_in;

always @(posedge clk)
          if(!(|counter) & start)       key_buf[063:000] <= #1 key_in;
          else
          if(counter == 8'h01)                    key_buf[127:064] <= #1 key_in;

always @(posedge clk)
          if(option_buf)
                    begin
                              if(start)           load_crypto <= #1 1'b0;
                              else
                              if(counter == 8'h01)          load_crypto <= #1 1'b1;
                              else
                              if(counter == 8'h02)          load_crypto <= #1 1'b0;

                              load_key <= #1 1'b0;
                    end
          else
          if(!option_buf)
                    begin
                              if(start)           load_key <= #1 1'b0;
                              else
                              if(counter == 8'h01)          load_key <= #1 1'b1;
                              else
                              if(counter == 8'h02)          load_key <= #1 1'b0;
```

```
                        if(load_key)        cnt_temp <= #1 8'h0b;
                        else
                        if(|cnt_temp)       cnt_temp <= #1 cnt_temp - 8'h01;

                        load_crypto <= #1 !(|cnt_temp[7:1]) & cnt_temp[0];
                end

// encryption or decryption process takes place

always @(posedge clk)
        if(done)  assign cnt_temp2 = counter;

always @(posedge clk)
        if(done)                                text_out <= #1 text_out_buf[063:000];
        else
        if((counter - cnt_temp2) == 8'h01)      text_out <= #1 text_out_buf[127:064];

always @(posedge clk)
        if(done || ((counter - cnt_temp2) == 8'h01)) get_output <= #1 1'b1;

always @(posedge clk)
        if((counter - cnt_temp2) == 8'h01)      complete <= #1 1'b1;
        else
        if((counter - cnt_temp2) == 8'h02)      complete <= #1 1'b0;

aes_crypto_top u0(
        .ciph_opt(          option_buf          ),
        .clk(               clk                         ),
        .rst(               rst                         ),
        .kld(               load_key  ),
        .ld(                load_crypto         ),
        .done(              done                ),
        .key(               key_buf             ),
        .text_in( text_in_buf       ),
        .text_out(          text_out_buf)
        );

endmodule
```

```
//////////////////////////////////////////////////////////////////////
////                                                              ////
////  AES Cipher Top Level                                        ////
////      (combined with)                                         ////
////  AES Inverse Cipher Top Level                                ////
////                                                              ////
////                                                              ////
////  Author: Rudolf Usselmann                                    ////
////          rudi@asics.ws                                       ////
////                                                              ////
////  Modified by: Lee Yi Lin                                     ////
////                                leeyilin@yahoo.co             ////
////                                                              ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/   ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
////                                                              ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                     ////
////                         www.asics.ws                         ////
////                         rudi@asics.ws                        ////
////                                                              ////
//// This source file may be used and distributed without         ////
//// restriction provided that this copyright statement is not    ////
//// removed from the file and that any derivative work contains  ////
//// the original copyright notice and the associated disclaimer.////
////                                                              ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY      ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED    ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS    ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR       ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,          ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES     ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE    ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF   ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT   ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT   ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          ////
//// POSSIBILITY OF SUCH DAMAGE.                                  ////
////                                                              ////
//////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: aes_cipher_top.v,v 1.1.1.1 2002/11/09 11:22:48 rudi Exp $
//  $Id: aes_inv_cipher_top.v,v 1.1.1.1 2002/11/09 11:22:53 rudi Exp $
//
//  $Date: 2002/11/09 11:22:48 $
//  $Revision: 1.1.1.1 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: aes_cipher_top.v,v $
//               Revision 1.1.1.1  2002/11/09 11:22:48  rudi
//               Initial Checkin
//
//        Modification:
//                                      Both Cipher and Inverse Cipher were merged for combined-Sbox
instantiation
//                                      Modified on 1/4/2004 for Final Year Project of EE Faculty, UTP
//
//

`include "c:\aes_modules\new\timescale.v"

module aes_crypto_top(ciph_opt, clk, rst, kld, ld, done, key, text_in, text_out );
input               ciph_opt;
input               clk, rst;
input               kld, ld;
output              done;
input     [127:0]   key;
input     [127:0]   text_in;
output    [127:0]   text_out;

reg       [127:0]   text_out;
reg                         done;
reg       [7:0]     in_s00, in_s01, in_s02, in_s03;
reg       [7:0]     in_s10, in_s11, in_s12, in_s13;
reg       [7:0]     in_s20, in_s21, in_s22, in_s23;
reg       [7:0]     in_s30, in_s31, in_s32, in_s33;
wire      [7:0]     out_s00, out_s01, out_s02, out_s03;
wire      [7:0]     out_s10, out_s11, out_s12, out_s13;
wire      [7:0]     out_s20, out_s21, out_s22, out_s23;
wire      [7:0]     out_s30, out_s31, out_s32, out_s33;
wire      [31:0]    w0, w1, w2, w3;

//////////////////////////////////////////////////////////////////////
//
// Local Wires for Cipher
//

wire      [31:0]    Cw0, Cw1, Cw2, Cw3;
reg       [127:0]   Ctext_in_r;
reg       [127:0]   Ctext_out;
reg       [7:0]     Csa00, Csa01, Csa02, Csa03;
reg       [7:0]     Csa10, Csa11, Csa12, Csa13;
reg       [7:0]     Csa20, Csa21, Csa22, Csa23;
reg       [7:0]     Csa30, Csa31, Csa32, Csa33;
wire      [7:0]     Csa00_next, Csa01_next, Csa02_next, Csa03_next;
wire      [7:0]     Csa10_next, Csa11_next, Csa12_next, Csa13_next;
```

```
wire      [7:0]     Csa20_next, Csa21_next, Csa22_next, Csa23_next;
wire      [7:0]     Csa30_next, Csa31_next, Csa32_next, Csa33_next;
wire      [7:0]     Csa00_sub, Csa01_sub, Csa02_sub, Csa03_sub;
wire      [7:0]     Csa10_sub, Csa11_sub, Csa12_sub, Csa13_sub;
wire      [7:0]     Csa20_sub, Csa21_sub, Csa22_sub, Csa23_sub;
wire      [7:0]     Csa30_sub, Csa31_sub, Csa32_sub, Csa33_sub;
wire      [7:0]     Csa00_sr, Csa01_sr, Csa02_sr, Csa03_sr;
wire      [7:0]     Csa10_sr, Csa11_sr, Csa12_sr, Csa13_sr;
wire      [7:0]     Csa20_sr, Csa21_sr, Csa22_sr, Csa23_sr;
wire      [7:0]     Csa30_sr, Csa31_sr, Csa32_sr, Csa33_sr;
wire      [7:0]     Csa00_mc, Csa01_mc, Csa02_mc, Csa03_mc;
wire      [7:0]     Csa10_mc, Csa11_mc, Csa12_mc, Csa13_mc;
wire      [7:0]     Csa20_mc, Csa21_mc, Csa22_mc, Csa23_mc;
wire      [7:0]     Csa30_mc, Csa31_mc, Csa32_mc, Csa33_mc;
reg                 Cdone, Cld_r;
reg       [3:0]     Cdcnt;

////////////////////////////////////////////////////////////////////
//
// Local Wires for Inverse Cipher
//

wire      [31:0]    Iwk0, Iwk1, Iwk2, Iwk3;
reg       [31:0]    Iw0, Iw1, Iw2, Iw3;
reg       [127:0]   Itext_in_r;
reg       [127:0]   Itext_out;
reg       [7:0]     Isa00, Isa01, Isa02, Isa03;
reg       [7:0]     Isa10, Isa11, Isa12, Isa13;
reg       [7:0]     Isa20, Isa21, Isa22, Isa23;
reg       [7:0]     Isa30, Isa31, Isa32, Isa33;
wire      [7:0]     Isa00_next, Isa01_next, Isa02_next, Isa03_next;
wire      [7:0]     Isa10_next, Isa11_next, Isa12_next, Isa13_next;
wire      [7:0]     Isa20_next, Isa21_next, Isa22_next, Isa23_next;
wire      [7:0]     Isa30_next, Isa31_next, Isa32_next, Isa33_next;
wire      [7:0]     Isa00_sub, Isa01_sub, Isa02_sub, Isa03_sub;
wire      [7:0]     Isa10_sub, Isa11_sub, Isa12_sub, Isa13_sub;
wire      [7:0]     Isa20_sub, Isa21_sub, Isa22_sub, Isa23_sub;
wire      [7:0]     Isa30_sub, Isa31_sub, Isa32_sub, Isa33_sub;
wire      [7:0]     Isa00_sr, Isa01_sr, Isa02_sr, Isa03_sr;
wire      [7:0]     Isa10_sr, Isa11_sr, Isa12_sr, Isa13_sr;
wire      [7:0]     Isa20_sr, Isa21_sr, Isa22_sr, Isa23_sr;
wire      [7:0]     Isa30_sr, Isa31_sr, Isa32_sr, Isa33_sr;
wire      [7:0]     Isa00_ark, Isa01_ark, Isa02_ark, Isa03_ark;
wire      [7:0]     Isa10_ark, Isa11_ark, Isa12_ark, Isa13_ark;
wire      [7:0]     Isa20_ark, Isa21_ark, Isa22_ark, Isa23_ark;
wire      [7:0]     Isa30_ark, Isa31_ark, Isa32_ark, Isa33_ark;
reg                 Ild_r, Igo, Idone;
reg       [3:0]     Idcnt;


////////////////////////////////////////////////////////////////////
//
// Misc Logic for Cipher
//

always @(posedge clk)
          if(!rst)  Cdcnt <= #1 4'h0;
          else
          if(ld)              Cdcnt <= #1 4'hb;
          else
          if(|Cdcnt)          Cdcnt <= #1 Cdcnt - 4'h1;

always @(posedge clk) Cdone <= #1 !(|Cdcnt[3:1]) & Cdcnt[0] & !ld;
always @(posedge clk) if(ld) Ctext_in_r <= #1 text_in;
always @(posedge clk) Cld_r <= #1 ld;

////////////////////////////////////////////////////////////////////
//
// Initial Permutation (AddRoundKey) for Cipher
//

always @(posedge clk)         Csa33 <= #1 Cld_r ? Ctext_in_r[007:000] ^ Cw3[07:00] : Csa33_next;
always @(posedge clk)         Csa23 <= #1 Cld_r ? Ctext_in_r[015:008] ^ Cw3[15:08] : Csa23_next;
always @(posedge clk)         Csa13 <= #1 Cld_r ? Ctext_in_r[023:016] ^ Cw3[23:16] : Csa13_next;
always @(posedge clk)         Csa03 <= #1 Cld_r ? Ctext_in_r[031:024] ^ Cw3[31:24] : Csa03_next;
always @(posedge clk)         Csa32 <= #1 Cld_r ? Ctext_in_r[039:032] ^ Cw2[07:00] : Csa32_next;
always @(posedge clk)         Csa22 <= #1 Cld_r ? Ctext_in_r[047:040] ^ Cw2[15:08] : Csa22_next;
always @(posedge clk)         Csa12 <= #1 Cld_r ? Ctext_in_r[055:048] ^ Cw2[23:16] : Csa12_next;
always @(posedge clk)         Csa02 <= #1 Cld_r ? Ctext_in_r[063:056] ^ Cw2[31:24] : Csa02_next;
always @(posedge clk)         Csa31 <= #1 Cld_r ? Ctext_in_r[071:064] ^ Cw1[07:00] : Csa31_next;
always @(posedge clk)         Csa21 <= #1 Cld_r ? Ctext_in_r[079:072] ^ Cw1[15:08] : Csa21_next;
always @(posedge clk)         Csa11 <= #1 Cld_r ? Ctext_in_r[087:080] ^ Cw1[23:16] : Csa11_next;
always @(posedge clk)         Csa01 <= #1 Cld_r ? Ctext_in_r[095:088] ^ Cw1[31:24] : Csa01_next;
always @(posedge clk)         Csa30 <= #1 Cld_r ? Ctext_in_r[103:096] ^ Cw0[07:00] : Csa30_next;
always @(posedge clk)         Csa20 <= #1 Cld_r ? Ctext_in_r[111:104] ^ Cw0[15:08] : Csa20_next;
always @(posedge clk)         Csa10 <= #1 Cld_r ? Ctext_in_r[119:112] ^ Cw0[23:16] : Csa10_next;
always @(posedge clk)         Csa00 <= #1 Cld_r ? Ctext_in_r[127:120] ^ Cw0[31:24] : Csa00_next;

////////////////////////////////////////////////////////////////////
//
// Round Permutations for Cipher
//

assign Csa00_sr = Csa00_sub;
assign Csa01_sr = Csa01_sub;
assign Csa02_sr = Csa02_sub;
assign Csa03_sr = Csa03_sub;
assign Csa10_sr = Csa11_sub;
assign Csa11_sr = Csa12_sub;
assign Csa12_sr = Csa13_sub;
assign Csa13_sr = Csa10_sub;
```

```
assign Csa20_sr = Csa22_sub;
assign Csa21_sr = Csa23_sub;
assign Csa22_sr = Csa20_sub;
assign Csa23_sr = Csa21_sub;
assign Csa30_sr = Csa33_sub;
assign Csa31_sr = Csa30_sub;
assign Csa32_sr = Csa31_sub;
assign Csa33_sr = Csa32_sub;
assign {Csa00_mc, Csa10_mc, Csa20_mc, Csa30_mc}  = mix_col(Csa00_sr,Csa10_sr,Csa20_sr,Csa30_sr);
assign {Csa01_mc, Csa11_mc, Csa21_mc, Csa31_mc}  = mix_col(Csa01_sr,Csa11_sr,Csa21_sr,Csa31_sr);
assign {Csa02_mc, Csa12_mc, Csa22_mc, Csa32_mc}  = mix_col(Csa02_sr,Csa12_sr,Csa22_sr,Csa32_sr);
assign {Csa03_mc, Csa13_mc, Csa23_mc, Csa33_mc}  = mix_col(Csa03_sr,Csa13_sr,Csa23_sr,Csa33_sr);
assign Csa00_next = Csa00_mc ^ Cw0[31:24];
assign Csa01_next = Csa01_mc ^ Cw1[31:24];
assign Csa02_next = Csa02_mc ^ Cw2[31:24];
assign Csa03_next = Csa03_mc ^ Cw3[31:24];
assign Csa10_next = Csa10_mc ^ Cw0[23:16];
assign Csa11_next = Csa11_mc ^ Cw1[23:16];
assign Csa12_next = Csa12_mc ^ Cw2[23:16];
assign Csa13_next = Csa13_mc ^ Cw3[23:16];
assign Csa20_next = Csa20_mc ^ Cw0[15:08];
assign Csa21_next = Csa21_mc ^ Cw1[15:08];
assign Csa22_next = Csa22_mc ^ Cw2[15:08];
assign Csa23_next = Csa23_mc ^ Cw3[15:08];
assign Csa30_next = Csa30_mc ^ Cw0[07:00];
assign Csa31_next = Csa31_mc ^ Cw1[07:00];
assign Csa32_next = Csa32_mc ^ Cw2[07:00];
assign Csa33_next = Csa33_mc ^ Cw3[07:00];

////////////////////////////////////////////////////////////////////
//
// Final text output for Cipher
//

always @(posedge clk) Ctext_out[127:120] <= #1 Csa00_sr ^ Cw0[31:24];
always @(posedge clk) Ctext_out[095:088] <= #1 Csa01_sr ^ Cw1[31:24];
always @(posedge clk) Ctext_out[063:056] <= #1 Csa02_sr ^ Cw2[31:24];
always @(posedge clk) Ctext_out[031:024] <= #1 Csa03_sr ^ Cw3[31:24];
always @(posedge clk) Ctext_out[119:112] <= #1 Csa10_sr ^ Cw0[23:16];
always @(posedge clk) Ctext_out[087:080] <= #1 Csa11_sr ^ Cw1[23:16];
always @(posedge clk) Ctext_out[055:048] <= #1 Csa12_sr ^ Cw2[23:16];
always @(posedge clk) Ctext_out[023:016] <= #1 Csa13_sr ^ Cw3[23:16];
always @(posedge clk) Ctext_out[111:104] <= #1 Csa20_sr ^ Cw0[15:08];
always @(posedge clk) Ctext_out[079:072] <= #1 Csa21_sr ^ Cw1[15:08];
always @(posedge clk) Ctext_out[047:040] <= #1 Csa22_sr ^ Cw2[15:08];
always @(posedge clk) Ctext_out[015:008] <= #1 Csa23_sr ^ Cw3[15:08];
always @(posedge clk) Ctext_out[103:096] <= #1 Csa30_sr ^ Cw0[07:00];
always @(posedge clk) Ctext_out[071:064] <= #1 Csa31_sr ^ Cw1[07:00];
always @(posedge clk) Ctext_out[039:032] <= #1 Csa32_sr ^ Cw2[07:00];
always @(posedge clk) Ctext_out[007:000] <= #1 Csa33_sr ^ Cw3[07:00];


////////////////////////////////////////////////////////////////////
//
// Misc Logic for Inverse Cipher
//

always @(posedge clk)
          if(!rst)  Idcnt <= #1 4'h0;
          else
          if(Idone) Idcnt <= #1 4'h0;
          else
          if(ld)              Idcnt <= #1 4'h1;
          else
          if(Igo)             Idcnt <= #1 Idcnt + 4'h1;

always @(posedge clk)         Idone <= #1 (Idcnt==4'hb) & !ld;

always @(posedge clk)
          if(!rst)  Igo <= #1 1'b0;
          else
          if(ld)              Igo <= #1 1'b1;
          else
          if(Idone) Igo <= #1 1'b0;

always @(posedge clk)         if(ld)    Itext_in_r <= #1 text_in;

always @(posedge clk)         Ild_r <= #1 ld;

////////////////////////////////////////////////////////////////////
//
// Initial Permutation Inverse Cipher
//

always @(posedge clk)         Isa33 <= #1 Ild_r ? Itext_in_r[007:000] ^ Iw3[07:00] : Isa33_next;
always @(posedge clk)         Isa23 <= #1 Ild_r ? Itext_in_r[015:008] ^ Iw3[15:08] : Isa23_next;
always @(posedge clk)         Isa13 <= #1 Ild_r ? Itext_in_r[023:016] ^ Iw3[23:16] : Isa13_next;
always @(posedge clk)         Isa03 <= #1 Ild_r ? Itext_in_r[031:024] ^ Iw3[31:24] : Isa03_next;
always @(posedge clk)         Isa32 <= #1 Ild_r ? Itext_in_r[039:032] ^ Iw2[07:00] : Isa32_next;
always @(posedge clk)         Isa22 <= #1 Ild_r ? Itext_in_r[047:040] ^ Iw2[15:08] : Isa22_next;
always @(posedge clk)         Isa12 <= #1 Ild_r ? Itext_in_r[055:048] ^ Iw2[23:16] : Isa12_next;
always @(posedge clk)         Isa02 <= #1 Ild_r ? Itext_in_r[063:056] ^ Iw2[31:24] : Isa02_next;
always @(posedge clk)         Isa31 <= #1 Ild_r ? Itext_in_r[071:064] ^ Iw1[07:00] : Isa31_next;
always @(posedge clk)         Isa21 <= #1 Ild_r ? Itext_in_r[079:072] ^ Iw1[15:08] : Isa21_next;
always @(posedge clk)         Isa11 <= #1 Ild_r ? Itext_in_r[087:080] ^ Iw1[23:16] : Isa11_next;
always @(posedge clk)         Isa01 <= #1 Ild_r ? Itext_in_r[095:088] ^ Iw1[31:24] : Isa01_next;
always @(posedge clk)         Isa30 <= #1 Ild_r ? Itext_in_r[103:096] ^ Iw0[07:00] : Isa30_next;
always @(posedge clk)         Isa20 <= #1 Ild_r ? Itext_in_r[111:104] ^ Iw0[15:08] : Isa20_next;
always @(posedge clk)         Isa10 <= #1 Ild_r ? Itext_in_r[119:112] ^ Iw0[23:16] : Isa10_next;
always @(posedge clk)         Isa00 <= #1 Ild_r ? Itext_in_r[127:120] ^ Iw0[31:24] : Isa00_next;

////////////////////////////////////////////////////////////////////
```

```
//
// Round Permutations Inverse Cipher
//

assign Isa00_sr = Isa00;
assign Isa01_sr = Isa01;
assign Isa02_sr = Isa02;
assign Isa03_sr = Isa03;
assign Isa10_sr = Isa13;
assign Isa11_sr = Isa10;
assign Isa12_sr = Isa11;
assign Isa13_sr = Isa12;
assign Isa20_sr = Isa22;
assign Isa21_sr = Isa23;
assign Isa22_sr = Isa20;
assign Isa23_sr = Isa21;
assign Isa30_sr = Isa31;
assign Isa31_sr = Isa32;
assign Isa32_sr = Isa33;
assign Isa33_sr = Isa30;
assign Isa00_ark = Isa00_sub ^ Iw0[31:24];
assign Isa01_ark = Isa01_sub ^ Iw1[31:24];
assign Isa02_ark = Isa02_sub ^ Iw2[31:24];
assign Isa03_ark = Isa03_sub ^ Iw3[31:24];
assign Isa10_ark = Isa10_sub ^ Iw0[23:16];
assign Isa11_ark = Isa11_sub ^ Iw1[23:16];
assign Isa12_ark = Isa12_sub ^ Iw2[23:16];
assign Isa13_ark = Isa13_sub ^ Iw3[23:16];
assign Isa20_ark = Isa20_sub ^ Iw0[15:08];
assign Isa21_ark = Isa21_sub ^ Iw1[15:08];
assign Isa22_ark = Isa22_sub ^ Iw2[15:08];
assign Isa23_ark = Isa23_sub ^ Iw3[15:08];
assign Isa30_ark = Isa30_sub ^ Iw0[07:00];
assign Isa31_ark = Isa31_sub ^ Iw1[07:00];
assign Isa32_ark = Isa32_sub ^ Iw2[07:00];
assign Isa33_ark = Isa33_sub ^ Iw3[07:00];
assign {Isa00_next, Isa10_next, Isa20_next, Isa30_next} = inv_mix_col(Isa00_ark,Isa10_ark,Isa20_ark,Isa30_ark);
assign {Isa01_next, Isa11_next, Isa21_next, Isa31_next} = inv_mix_col(Isa01_ark,Isa11_ark,Isa21_ark,Isa31_ark);
assign {Isa02_next, Isa12_next, Isa22_next, Isa32_next} = inv_mix_col(Isa02_ark,Isa12_ark,Isa22_ark,Isa32_ark);
assign {Isa03_next, Isa13_next, Isa23_next, Isa33_next} = inv_mix_col(Isa03_ark,Isa13_ark,Isa23_ark,Isa33_ark);

////////////////////////////////////////////////////////////////////
//
// Final Text Output Inverse Cipher
//

always @(posedge clk) Itext_out[127:120] <= #1 Isa00_ark;
always @(posedge clk) Itext_out[095:088] <= #1 Isa01_ark;
always @(posedge clk) Itext_out[063:056] <= #1 Isa02_ark;
always @(posedge clk) Itext_out[031:024] <= #1 Isa03_ark;
always @(posedge clk) Itext_out[119:112] <= #1 Isa10_ark;
always @(posedge clk) Itext_out[087:080] <= #1 Isa11_ark;
always @(posedge clk) Itext_out[055:048] <= #1 Isa12_ark;
always @(posedge clk) Itext_out[023:016] <= #1 Isa13_ark;
always @(posedge clk) Itext_out[111:104] <= #1 Isa20_ark;
always @(posedge clk) Itext_out[079:072] <= #1 Isa21_ark;
always @(posedge clk) Itext_out[047:040] <= #1 Isa22_ark;
always @(posedge clk) Itext_out[015:008] <= #1 Isa23_ark;
always @(posedge clk) Itext_out[103:096] <= #1 Isa30_ark;
always @(posedge clk) Itext_out[071:064] <= #1 Isa31_ark;
always @(posedge clk) Itext_out[039:032] <= #1 Isa32_ark;
always @(posedge clk) Itext_out[007:000] <= #1 Isa33_ark;


always @(posedge clk) text_out <= #1 ciph_opt ? Ctext_out : Itext_out;
always @(posedge clk) done <= #1 ciph_opt ? Cdone : Idone;


////////////////////////////////////////////////////////////////////
//
// Generic Functions for Cipher
//

function [31:0] mix_col;
input     [7:0]     s0,s1,s2,s3;
reg       [7:0]     s0_o,s1_o,s2_o,s3_o;
begin
mix_col[31:24]=xtime(s0)^xtime(s1)^s1^s2^s3;
mix_col[23:16]=s0^xtime(s1)^xtime(s2)^s2^s3;
mix_col[15:08]=s0^s1^xtime(s2)^xtime(s3)^s3;
mix_col[07:00]=xtime(s0)^s0^s1^s2^xtime(s3);
end
endfunction

function [7:0] xtime;
input [7:0] b; xtime={b[6:0],1'b0}^(8'h1b&{8{b[7]}});
endfunction

////////////////////////////////////////////////////////////////////
//
// Generic Functions for Inverse Cipher
//

function [31:0] inv_mix_col;
input     [7:0]     s0,s1,s2,s3;
begin
inv_mix_col[31:24]=pmul_e(s0)^pmul_b(s1)^pmul_d(s2)^pmul_9(s3);
inv_mix_col[23:16]=pmul_9(s0)^pmul_e(s1)^pmul_b(s2)^pmul_d(s3);
inv_mix_col[15:08]=pmul_d(s0)^pmul_9(s1)^pmul_e(s2)^pmul_b(s3);
inv_mix_col[07:00]=pmul_b(s0)^pmul_d(s1)^pmul_9(s2)^pmul_e(s3);
end
```

```
endfunction

// Some synthesis tools don't like xtime being called recursevly ...
function [7:0] pmul_e;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_e=eight^four^two;
end
endfunction

function [7:0] pmul_9;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_9=eight^b;
end
endfunction

function [7:0] pmul_d;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_d=eight^four^b;
end
endfunction

function [7:0] pmul_b;
input [7:0] b;
reg [7:0] two,four,eight;
begin
two=xtime(b);four=xtime(two);eight=xtime(four);pmul_b=eight^two^b;
end
endfunction


//////////////////////////////////////////////////////////////////////
//
// Key Buffer for Inverse Cipher
//

reg       [127:0]   kb[10:0];
reg       [3:0]     kcnt;
reg                 kdone;
reg                 kb_ld;

always @(posedge clk)
          if(!rst)  kcnt <= #1 4'ha;
          else
          if(kld)             kcnt <= #1 4'ha;
          else
          if(kb_ld) kcnt <= #1 kcnt - 4'h1;

always @(posedge clk)
          if(!rst)  kb_ld <= #1 1'b0;
          else
          if(kld)             kb_ld <= #1 1'b1;
          else
          if(kcnt==4'h0)      kb_ld <= #1 1'b0;

always @(posedge clk)         kdone <= #1 (kcnt==4'h0) & !kld;
always @(posedge clk)         if(kb_ld) kb[kcnt] <= #1 {Iwk3, Iwk2, Iwk1, Iwk0};
always @(posedge clk)         {Iw3, Iw2, Iw1, Iw0} <= #1 kb[Idcnt];


//////////////////////////////////////////////////////////////////////
//
// Modules for both Cipher and Inverse Cipher
//

assign Cw0 = w0;
assign Cw1 = w1;
assign Cw2 = w2;
assign Cw3 = w3;
assign Csa00_sub = out_s00;
assign Csa01_sub = out_s01;
assign Csa02_sub = out_s02;
assign Csa03_sub = out_s03;
assign Csa10_sub = out_s10;
assign Csa11_sub = out_s11;
assign Csa12_sub = out_s12;
assign Csa13_sub = out_s13;
assign Csa20_sub = out_s20;
assign Csa21_sub = out_s21;
assign Csa22_sub = out_s22;
assign Csa23_sub = out_s23;
assign Csa30_sub = out_s30;
assign Csa31_sub = out_s31;
assign Csa32_sub = out_s32;
assign Csa33_sub = out_s33;

assign Iwk0 = w0;
assign Iwk1 = w1;
assign Iwk2 = w2;
assign Iwk3 = w3;
assign Isa00_sub = out_s00;
assign Isa01_sub = out_s01;
assign Isa02_sub = out_s02;
assign Isa03_sub = out_s03;
assign Isa10_sub = out_s10;
assign Isa11_sub = out_s11;
assign Isa12_sub = out_s12;
```

```
assign Isa13_sub = out_s13;
assign Isa20_sub = out_s20;
assign Isa21_sub = out_s21;
assign Isa22_sub = out_s22;
assign Isa23_sub = out_s23;
assign Isa30_sub = out_s30;
assign Isa31_sub = out_s31;
assign Isa32_sub = out_s32;
assign Isa33_sub = out_s33;

always @(ciph_opt) in_s00 <= ciph_opt ? Csa00 : Isa00_sr;
always @(ciph_opt) in_s01 <= ciph_opt ? Csa01 : Isa01_sr;
always @(ciph_opt) in_s02 <= ciph_opt ? Csa02 : Isa02_sr;
always @(ciph_opt) in_s03 <= ciph_opt ? Csa03 : Isa03_sr;
always @(ciph_opt) in_s10 <= ciph_opt ? Csa10 : Isa10_sr;
always @(ciph_opt) in_s11 <= ciph_opt ? Csa11 : Isa11_sr;
always @(ciph_opt) in_s12 <= ciph_opt ? Csa12 : Isa12_sr;
always @(ciph_opt) in_s13 <= ciph_opt ? Csa13 : Isa13_sr;
always @(ciph_opt) in_s20 <= ciph_opt ? Csa20 : Isa20_sr;
always @(ciph_opt) in_s21 <= ciph_opt ? Csa21 : Isa21_sr;
always @(ciph_opt) in_s22 <= ciph_opt ? Csa22 : Isa22_sr;
always @(ciph_opt) in_s23 <= ciph_opt ? Csa23 : Isa23_sr;
always @(ciph_opt) in_s30 <= ciph_opt ? Csa30 : Isa30_sr;
always @(ciph_opt) in_s31 <= ciph_opt ? Csa31 : Isa31_sr;
always @(ciph_opt) in_s32 <= ciph_opt ? Csa32 : Isa32_sr;
always @(ciph_opt) in_s33 <= ciph_opt ? Csa33 : Isa33_sr;


aes_key_expand_128 u0(
          .clk(               clk       ),
          .kld(               ld        ),
          .key(               key       ),
          .wo_0(              w0        ),
          .wo_1(              w1        ),
          .wo_2(              w2        ),
          .wo_3(              w3        ));

aes_sbox_inv us00( .b(      ciph_opt  ),      .a(      in_s00    ), .d(    out_s00   ));
aes_sbox_inv us01( .b(      ciph_opt  ),      .a(      in_s01    ), .d(    out_s01   ));
aes_sbox_inv us02( .b(      ciph_opt  ),      .a(      in_s02    ), .d(    out_s02   ));
aes_sbox_inv us03( .b(      ciph_opt  ),      .a(      in_s03    ), .d(    out_s03   ));
aes_sbox_inv us10( .b(      ciph_opt  ),      .a(      in_s10    ), .d(    out_s10   ));
aes_sbox_inv us11( .b(      ciph_opt  ),      .a(      in_s11    ), .d(    out_s11   ));
aes_sbox_inv us12( .b(      ciph_opt  ),      .a(      in_s12    ), .d(    out_s12   ));
aes_sbox_inv us13( .b(      ciph_opt  ),      .a(      in_s13    ), .d(    out_s13   ));
aes_sbox_inv us20( .b(      ciph_opt  ),      .a(      in_s20    ), .d(    out_s20   ));
aes_sbox_inv us21( .b(      ciph_opt  ),      .a(      in_s21    ), .d(    out_s21   ));
aes_sbox_inv us22( .b(      ciph_opt  ),      .a(      in_s22    ), .d(    out_s22   ));
aes_sbox_inv us23( .b(      ciph_opt  ),      .a(      in_s23    ), .d(    out_s23   ));
aes_sbox_inv us30( .b(      ciph_opt  ),      .a(      in_s30    ), .d(    out_s30   ));
aes_sbox_inv us31( .b(      ciph_opt  ),      .a(      in_s31    ), .d(    out_s31   ));
aes_sbox_inv us32( .b(      ciph_opt  ),      .a(      in_s32    ), .d(    out_s32   ));
aes_sbox_inv us33( .b(      ciph_opt  ),      .a(      in_s33    ), .d(    out_s33   ));

endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES Combined Sbox/Inv-Sbox Top Layer Block                ////
////                                                            ////
////                                                            ////
////                                                            ////
////  Author: Lee Yi Lin                                        ////
////          leeyilin@yahoo.com                                ////
////                                                            ////
////                                                            ////
////  Composed for Final Year Project of EE Faculty, UTP           ////
////                                                            ////
////////////////////////////////////////////////////////////////////
//
//
//  $Date: 2004/4/01 $
//  $Revision: 1.0 $
//  $Author: Lee Yi Lin $
//
//  Change History:
//
//
//
//

`include "c:\aes_modules\new\timescale.v"

module aes_sbox_inv(b,a,d);
input b;
input [7:0] a;
output [7:0] d;

reg [7:0] d;
reg [7:0] a_in;
wire [7:0] a_sub;

always @(b)
if (b)
        begin
        a_in = a;
        d[7] = a_sub[7]^a_sub[6]^a_sub[5]^a_sub[4]^a_sub[3]^1'b0;
        d[6] = a_sub[6]^a_sub[5]^a_sub[4]^a_sub[3]^a_sub[2]^1'b1;
        d[5] = a_sub[5]^a_sub[4]^a_sub[3]^a_sub[2]^a_sub[1]^1'b1;
        d[4] = a_sub[4]^a_sub[3]^a_sub[2]^a_sub[1]^a_sub[0]^1'b0;
        d[3] = a_sub[7]^a_sub[3]^a_sub[2]^a_sub[1]^a_sub[0]^1'b0;
        d[2] = a_sub[7]^a_sub[6]^a_sub[2]^a_sub[1]^a_sub[0]^1'b0;
        d[1] = a_sub[7]^a_sub[6]^a_sub[5]^a_sub[1]^a_sub[0]^1'b1;
        d[0] = a_sub[7]^a_sub[6]^a_sub[5]^a_sub[4]^a_sub[0]^1'b1;
        end
else
        begin
        a_in[7] = a[6]^a[4]^a[1]^1'b0;
        a_in[6] = a[5]^a[3]^a[0]^1'b0;
        a_in[5] = a[7]^a[4]^a[2]^1'b0;
        a_in[4] = a[6]^a[3]^a[1]^1'b0;
        a_in[3] = a[5]^a[2]^a[0]^1'b0;
        a_in[2] = a[7]^a[4]^a[1]^1'b1;
        a_in[1] = a[6]^a[3]^a[0]^1'b0;
        a_in[0] = a[7]^a[5]^a[2]^1'b1;
        d = a_sub;
        end

aes_mul_inv m0( .b(    b       ),      .m( a_in ), .g( a_sub ));

endmodule
```

```
////////////////////////////////////////////////////////////////////
////                                                            ////
////  AES Multiplicative Inverse Block                          ////
////  (to be instantiated by aes_sbox_inv for building          ////
////  (both Sbox and Inverse Sbox)                              ////
////                                                            ////
////  Author: Lee Yi Lin                                        ////
////          leeyilin@yahoo.com                                ////
////                                                            ////
////                                                            ////
////  Composed for Final Year Project of EE Faculty, UTP        ////
////                                                            ////
////////////////////////////////////////////////////////////////////
//
//
//  $Date: 2004/4/01 $
//  $Revision: 1.0 $
//  $Author: Lee Yi Lin $
//
//  Change History:
//
//
//
//

`include "c:\aes_modules\new\timescale.v"

module aes_mul_inv(b,m,g);
input b;
input [7:0] m;
output [7:0] g;
reg [7:0] g;

always @(b)
          case(m)
             8'h00: g=8'h00;
             8'h01: g=8'h01;
             8'h02: g=8'h8d;
             8'h03: g=8'hf6;
             8'h04: g=8'hcb;
             8'h05: g=8'h52;
             8'h06: g=8'h7b;
             8'h07: g=8'hd1;
             8'h08: g=8'he8;
             8'h09: g=8'h4f;
             8'h0a: g=8'h29;
             8'h0b: g=8'hc0;
             8'h0c: g=8'hb0;
             8'h0d: g=8'he1;
             8'h0e: g=8'he5;
             8'h0f: g=8'hc7;
             8'h10: g=8'h74;
             8'h11: g=8'hb4;
             8'h12: g=8'haa;
             8'h13: g=8'h4b;
             8'h14: g=8'h99;
             8'h15: g=8'h2b;
             8'h16: g=8'h60;
             8'h17: g=8'h5f;
             8'h18: g=8'h58;
             8'h19: g=8'h3f;
             8'h1a: g=8'hfd;
             8'h1b: g=8'hcc;
             8'h1c: g=8'hff;
             8'h1d: g=8'h40;
             8'h1e: g=8'hee;
             8'h1f: g=8'hb2;
             8'h20: g=8'h3a;
             8'h21: g=8'h6e;
             8'h22: g=8'h5a;
             8'h23: g=8'hf1;
             8'h24: g=8'h55;
             8'h25: g=8'h4d;
             8'h26: g=8'ha8;
             8'h27: g=8'hc9;
             8'h28: g=8'hc1;
             8'h29: g=8'h0a;
             8'h2a: g=8'h98;
             8'h2b: g=8'h15;
             8'h2c: g=8'h30;
             8'h2d: g=8'h44;
             8'h2e: g=8'ha2;
             8'h2f: g=8'hc2;
             8'h30: g=8'h2c;
             8'h31: g=8'h45;
             8'h32: g=8'h92;
             8'h33: g=8'h6c;
             8'h34: g=8'hf3;
             8'h35: g=8'h39;
             8'h36: g=8'h66;
             8'h37: g=8'h42;
             8'h38: g=8'hf2;
             8'h39: g=8'h35;
             8'h3a: g=8'h20;
             8'h3b: g=8'h6f;
             8'h3c: g=8'h77;
             8'h3d: g=8'hbb;
             8'h3e: g=8'h59;
             8'h3f: g=8'h19;
             8'h40: g=8'h1d;
             8'h41: g=8'hfe;
             8'h42: g=8'h37;
```

```
8'h43: g=8'h67;
8'h44: g=8'h2d;
8'h45: g=8'h31;
8'h46: g=8'hf5;
8'h47: g=8'h69;
8'h48: g=8'ha7;
8'h49: g=8'h64;
8'h4a: g=8'hab;
8'h4b: g=8'h13;
8'h4c: g=8'h54;
8'h4d: g=8'h25;
8'h4e: g=8'he9;
8'h4f: g=8'h09;
8'h50: g=8'hed;
8'h51: g=8'h5c;
8'h52: g=8'h05;
8'h53: g=8'hca;
8'h54: g=8'h4c;
8'h55: g=8'h24;
8'h56: g=8'h87;
8'h57: g=8'hbf;
8'h58: g=8'h18;
8'h59: g=8'h3e;
8'h5a: g=8'h22;
8'h5b: g=8'hf0;
8'h5c: g=8'h51;
8'h5d: g=8'hec;
8'h5e: g=8'h61;
8'h5f: g=8'h17;
8'h60: g=8'h16;
8'h61: g=8'h5e;
8'h62: g=8'haf;
8'h63: g=8'hd3;
8'h64: g=8'h49;
8'h65: g=8'ha6;
8'h66: g=8'h36;
8'h67: g=8'h43;
8'h68: g=8'hf4;
8'h69: g=8'h47;
8'h6a: g=8'h91;
8'h6b: g=8'hdf;
8'h6c: g=8'h33;
8'h6d: g=8'h93;
8'h6e: g=8'h21;
8'h6f: g=8'h3b;
8'h70: g=8'h79;
8'h71: g=8'hb7;
8'h72: g=8'h97;
8'h73: g=8'h85;
8'h74: g=8'h10;
8'h75: g=8'hb5;
8'h76: g=8'hba;
8'h77: g=8'h3c;
8'h78: g=8'hb6;
8'h79: g=8'h70;
8'h7a: g=8'hd0;
8'h7b: g=8'h06;
8'h7c: g=8'ha1;
8'h7d: g=8'hfa;
8'h7e: g=8'h81;
8'h7f: g=8'h82;
8'h80: g=8'h83;
8'h81: g=8'h7e;
8'h82: g=8'h7f;
8'h83: g=8'h80;
8'h84: g=8'h96;
8'h85: g=8'h73;
8'h86: g=8'hbe;
8'h87: g=8'h56;
8'h88: g=8'h9b;
8'h89: g=8'h9e;
8'h8a: g=8'h95;
8'h8b: g=8'hd9;
8'h8c: g=8'hf7;
8'h8d: g=8'h02;
8'h8e: g=8'hb9;
8'h8f: g=8'ha4;
8'h90: g=8'hde;
8'h91: g=8'h6a;
8'h92: g=8'h32;
8'h93: g=8'h6d;
8'h94: g=8'hd8;
8'h95: g=8'h8a;
8'h96: g=8'h84;
8'h97: g=8'h72;
8'h98: g=8'h2a;
8'h99: g=8'h14;
8'h9a: g=8'h9f;
8'h9b: g=8'h88;
8'h9c: g=8'hf9;
8'h9d: g=8'hdc;
8'h9e: g=8'h89;
8'h9f: g=8'h9a;
8'ha0: g=8'hfb;
8'ha1: g=8'h7c;
8'ha2: g=8'h2e;
8'ha3: g=8'hc3;
8'ha4: g=8'h8f;
8'ha5: g=8'hb8;
8'ha6: g=8'h65;
8'ha7: g=8'h48;
8'ha8: g=8'h26;
```

```
            8'ha9: g=8'hc8;
            8'haa: g=8'h12;
            8'hab: g=8'h4a;
            8'hac: g=8'hce;
            8'had: g=8'he7;
            8'hae: g=8'hd2;
            8'haf: g=8'h62;
            8'hb0: g=8'h0c;
            8'hb1: g=8'he0;
            8'hb2: g=8'h1f;
            8'hb3: g=8'hef;
            8'hb4: g=8'h11;
            8'hb5: g=8'h75;
            8'hb6: g=8'h78;
            8'hb7: g=8'h71;
            8'hb8: g=8'ha5;
            8'hb9: g=8'h8e;
            8'hba: g=8'h76;
            8'hbb: g=8'h3d;
            8'hbc: g=8'hbd;
            8'hbd: g=8'hbc;
            8'hbe: g=8'h86;
            8'hbf: g=8'h57;
            8'hc0: g=8'h0b;
            8'hc1: g=8'h28;
            8'hc2: g=8'h2f;
            8'hc3: g=8'ha3;
            8'hc4: g=8'hda;
            8'hc5: g=8'hd4;
            8'hc6: g=8'he4;
            8'hc7: g=8'h0f;
            8'hc8: g=8'ha9;
            8'hc9: g=8'h27;
            8'hca: g=8'h53;
            8'hcb: g=8'h04;
            8'hcc: g=8'h1b;
            8'hcd: g=8'hfc;
            8'hce: g=8'hac;
            8'hcf: g=8'he6;
            8'hd0: g=8'h7a;
            8'hd1: g=8'h07;
            8'hd2: g=8'hae;
            8'hd3: g=8'h63;
            8'hd4: g=8'hc5;
            8'hd5: g=8'hdb;
            8'hd6: g=8'he2;
            8'hd7: g=8'hea;
            8'hd8: g=8'h94;
            8'hd9: g=8'h8b;
            8'hda: g=8'hc4;
            8'hdb: g=8'hd5;
            8'hdc: g=8'h9d;
            8'hdd: g=8'hf8;
            8'hde: g=8'h90;
            8'hdf: g=8'h6b;
            8'he0: g=8'hb1;
            8'he1: g=8'h0d;
            8'he2: g=8'hd6;
            8'he3: g=8'heb;
            8'he4: g=8'hc6;
            8'he5: g=8'h0e;
            8'he6: g=8'hcf;
            8'he7: g=8'had;
            8'he8: g=8'h08;
            8'he9: g=8'h4e;
            8'hea: g=8'hd7;
            8'heb: g=8'he3;
            8'hec: g=8'h5d;
            8'hed: g=8'h50;
            8'hee: g=8'h1e;
            8'hef: g=8'hb3;
            8'hf0: g=8'h5b;
            8'hf1: g=8'h23;
            8'hf2: g=8'h38;
            8'hf3: g=8'h34;
            8'hf4: g=8'h68;
            8'hf5: g=8'h46;
            8'hf6: g=8'h03;
            8'hf7: g=8'h8c;
            8'hf8: g=8'hdd;
            8'hf9: g=8'h9c;
            8'hfa: g=8'h7d;
            8'hfb: g=8'ha0;
            8'hfc: g=8'hcd;
            8'hfd: g=8'h1a;
            8'hfe: g=8'h41;
            8'hff: g=8'h1c;
        endcase

endmodule
```

```
//////////////////////////////////////////////////////////////////////
////                                                              ////
////  AES Test Bench                                              ////
////                                                              ////
////                                                              ////
////  Author: Rudolf Usselmann                                    ////
////          rudi@asics.ws                                       ////
////                                                              ////
////                                                              ////
////  Downloaded from: http://www.opencores.org/cores/aes_core/   ////
////                                                              ////
//////////////////////////////////////////////////////////////////////
////                                                              ////
//// Copyright (C) 2000-2002 Rudolf Usselmann                     ////
////                         www.asics.ws                         ////
////                         rudi@asics.ws                        ////
////                                                              ////
//// This source file may be used and distributed without         ////
//// restriction provided that this copyright statement is not    ////
//// removed from the file and that any derivative work contains  ////
//// the original copyright notice and the associated disclaimer.////
////                                                              ////
////     THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY      ////
//// EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED    ////
//// TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS    ////
//// FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL THE AUTHOR       ////
//// OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,          ////
//// INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES     ////
//// (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE    ////
//// GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR         ////
//// BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF   ////
//// LIABILITY, WHETHER IN  CONTRACT, STRICT LIABILITY, OR TORT   ////
//// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT   ////
//// OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE          ////
//// POSSIBILITY OF SUCH DAMAGE.                                  ////
////                                                              ////
//////////////////////////////////////////////////////////////////////

//  CVS Log
//
//  $Id: test_bench_top.v,v 1.2 2002/11/12 16:10:12 rudi Exp $
//
//  $Date: 2002/11/12 16:10:12 $
//  $Revision: 1.2 $
//  $Author: rudi $
//  $Locker:  $
//  $State: Exp $
//
// Change History:
//               $Log: test_bench_top.v,v $
//               Revision 1.2  2002/11/12 16:10:12  rudi
//
//               Improved test bench, added missing timescale file.
//
//               Revision 1.1.1.1  2002/11/09 11:22:56  rudi
//               Initial Checkin
//
//
//
//
//
//
//

`include "timescale.v"

module test;

reg                 clk;
reg                 rst;

reg       [383:0]   tv[512:0];          // Test vectors
wire      [383:0]   tmp;
reg                 kld;
wire      [127:0]   key, plain, ciph;
wire      [127:0]   text_in;
wire      [127:0]   text_out;
wire      [127:0]   text_out2;
reg       [127:0]   text_exp;
wire                done, done2;
integer             n, error_cnt;

initial
   begin
         $display("\n\n");
         $display("*****************************************************");
         $display("* AES Test bench ...");
         $display("*****************************************************");
         $display("\n");
`ifdef WAVES
         $shm_open("waves");
         $shm_probe("AS",test,"AS");
         $display("INFO: Signal dump enabled ...\n\n");
`endif

         kld = 0;
         clk = 0;
         rst = 0;
         error_cnt = 0;
         repeat(4) @(posedge clk);
         rst = 1;
         repeat(20)          @(posedge clk);
```

```
        $display("");
        $display("");
        $display("Started random test ...");

tv[0]= 384'h000000000000000000000000000000000f34481ec3cc627bacd5dc3fb08f273e60336763e966d92595a567cc9ce537f5e;
tv[1]= 384'h0000000000000000000000000000000009798c4640bad75c7c3227db910174e72a9a1631bf4996954ebc093957b234589;
tv[2]= 384'h00000000000000000000000000000096ab5c2ff612d9dfaae8c31f30c42168ff4f8391a6a40ca5b25d23bedd44a597;
tv[3]= 384'h000000000000000000000000000006a118a874519a64e9963798a503f1d35dc43be40be0e53712f7e2bf5ca707209;
tv[4]= 384'h0000000000000000000000000000cb9fceec81286ca3e989bd979b0cb28492beedab1895a94faa69b632e5cc47ce;
tv[5]= 384'h00000000000000000000000000b26aeb1874e47ca8358ff22378f09144459264f4798f6a78bacb89c15ed3d601;
tv[6]= 384'h0000000000000000000000000058c8e00b2631686d54eab84b91f0aca108a4e2efec8a8e3312ca7460b9040bbf;
tv[7]= 384'h10a58869d74be5a374cf867cfb473859000000000000000000000000000006d251e6944b051e04eaa6fb4dbf78465;
tv[8]= 384'hcaea65cdbb75e9169ecd22ebe6e5467500000000000000000000000000006e29201190152df4ee058139def610bb;
tv[9]= 384'ha2e2fa9baf7d20822ca9f0542f764a410000000000000000000000000000c3b44b95d9d2f25670eee9a0de099fa3;
tv[10]= 384'hb6364ac4e1de1e285eaf144a2415f7a0000000000000000000000000005d9b05578fc944b3cf1ccf0e746cd581;
tv[11]= 384'h64cf9c7abc50b888af65f49d521944b2000000000000000000000000f7efc89d5dba578104016ce5ad659c05;
tv[12]= 384'h47d6742eefcc0465dc96355e851b64d900000000000000000000000000306194f666d183624aa230a8b264ae7;
tv[13]= 384'h3eb39790678c56bee34bbcdeccf6cdb50000000000000000000000000858075d536d79ccee571f7d7204b1f67;
tv[14]= 384'h64110a924f0743d500ccadae72c13427000000000000000000000000035870c6a57e9e92314bcb8087cde72ce;
tv[15]= 384'h18d8126516f8a12ab1a36d9f04d68e5100000000000000000000000006c68e9be5ec41e22c825b7c7affb4363;
tv[16]= 384'hf530357968578480b398a3c251cd1093000000000000000000000000f5df39990fc688f1b07224cc03e86cea;
tv[17]= 384'hda84367f325d42d601b4326964802e8e000000000000000000000000bba071bcb470f8f6586e5d3add18bc66;
tv[18]= 384'he37b1c6aa2846f6fdb413f238b089f230000000000000000000000003580d19cff44f1014a7c966a69059de5;
tv[19]= 384'h6c002b682483e0cabcc731c253be567400000000000000000000000043c9f7e62f5d288bb27aa40ef8fe1ea8;
tv[20]= 384'h143ae8ed6555aba96110ab58893a8ae1000000000000000000000000806da964dd29d48deafbe764f8202aef;
tv[21]= 384'hb69418a85332240dc82492353956ae0c0000000000000000000000000a303d940ded8f0baff6f75414cac5243;
tv[22]= 384'h71b5c08a1993e1362e4d0ce9b22b78d50000000000000000000000000c2dabd117f8a3ecabfbb11d12194d9d0;
tv[23]= 384'he234cdca2606b81f29408d5f6da21206000000000000000000000000fff60a4740086b3b9c56195b98d91a7b;
tv[24]= 384'h13237c49074a3da078dc1d828bb78c6f000000000000000000000000146a08e2357f0caa30ca8c94d1a0544;
tv[25]= 384'h3071a2a48fe6cbd04f1a129098e308f800000000000000000000000004b98e06d356deb07ebb824e5713f7be3;
tv[26]= 384'h90f42ec0f68385f2ffc5dfc03a654dce0000000000000000000000007a20a53d460fc9ce0423a7a0764c6cf2;
tv[27]= 384'hfebd9a24d8b65c1c787d50a4ed3619a900000000000000000000000f4a70d8af877f9b02b4c40df57d45b17;
tv[28]= 384'h8000000000000000000000000000000000000000000000000000000edd33d3c621e546455bd8ba1418bec8;
tv[29]= 384'hc000000000000000000000000000000000000000000000000000004bc3f883450c113c64ca42e1112a9e87;
tv[30]= 384'he000000000000000000000000000000000000000000000000000072a1da770f5d7ac4c9ef94d822affd97;
tv[31]= 384'hf000000000000000000000000000000000000000000000000000970014d634e2b7650777e8e84d03ccd8;
tv[32]= 384'hf800000000000000000000000000000000000000000000000000f17e79aed0db7e279e955b5f493875a7;
tv[33]= 384'hfc0000000000000000000000000000000000000000000000009ed5a75136a940d0963da379db4af26a;
tv[34]= 384'hfe0000000000000000000000000000000000000000000000c4295f83465c7755e8fa364bac6a7ea5;
tv[35]= 384'hff0000000000000000000000000000000000000000000000b1d758256b28fd850ad4944208cf1155;
tv[36]= 384'hff8000000000000000000000000000000000000000000042ffb34c743de4d88ca38011c990890b;
tv[37]= 384'hffc0000000000000000000000000000000000000000000009958f0ecea8b2172c0c1995f9182c0f3;
tv[38]= 384'hffe0000000000000000000000000000000000000000000956d7798fac20f82a8823f984d06f7f5;
tv[39]= 384'hfff0000000000000000000000000000000000000000000a01bf44f2d16be928ca44aaf7b9b106b;
tv[40]= 384'hfff8000000000000000000000000000000000000000000b5f1a33e50d40d103764c76bd4c6b6f8;
tv[41]= 384'hfffc0000000000000000000000000000000000000000002637050c9fc0d4817e2d69de878aee8d;
tv[42]= 384'hfffe0000000000000000000000000000000000000000000113ecbe4a453269a0dd26069467fb5b5;
tv[43]= 384'hffff0000000000000000000000000000000000000000097d0754fe68f11b9e375d070a608c884;
tv[44]= 384'hffff8000000000000000000000000000000000000000c6a0b3e998d05068a539977840520 0b4;
tv[45]= 384'hfffffc0000000000000000000000000000000000000000df556a33438db87bc41b1752c55e5e49;
tv[46]= 384'hfffffe0000000000000000000000000000000000000090fb128d3a1af6e548521bb962bf1f05;
tv[47]= 384'hffffff0000000000000000000000000000000000000026298e9c1db517c215fadfb7d2a8d691;
tv[48]= 384'hffffff8000000000000000000000000000000000000a6cb761d61f8292d0df393a279ad0380;
tv[49]= 384'hffffffc0000000000000000000000000000000000012acd89b13cd5f8726e34d44fd486108;
tv[50]= 384'hffffffe0000000000000000000000000000000000095b1703fc57ba09fe0c3580febdd7ed4;
tv[51]= 384'hfffffff0000000000000000000000000000000000de11722d893e9f9121c381becc1da59a;
tv[52]= 384'hfffffff8000000000000000000000000000000000006d114ccb27bf391012e8974c546d9bf2;
tv[53]= 384'hfffffffc0000000000000000000000000000000005ce37e17eb4646ecfac29b9cc38d9340;
tv[54]= 384'hfffffffe0000000000000000000000000000000018c1b6e2157122056d0243d8a165cddb;
tv[55]= 384'hffffffff0000000000000000000000000000000099693e6a59d1366c74d823562d7e1431;
tv[56]= 384'hffffffff8000000000000000000000000000000006c7c64dc84a8bba758ed17eb025a57e3;
tv[57]= 384'hfffffffc0000000000000000000000000000000e17bc79f30eaab2fac2cbbe3458d687a;
tv[58]= 384'hfffffffe0000000000000000000000000000000001114bc2028009b923f0b01915ce5e7c4;
tv[59]= 384'hffffffff0000000000000000000000000000000009c28524a16a1e1c1452971caa8d13476;
tv[60]= 384'hffffffff8000000000000000000000000000000ed62e1636363860fdd6ad62112794f0;
tv[61]= 384'hfffffffc0000000000000000000000000000005a8688f0b2a2c16224c161658ffd4044;
tv[62]= 384'hfffffffe0000000000000000000000000000023f710842b9bb9c32f26648c786807ca;
tv[63]= 384'hffffffff0000000000000000000000000000044a98bf11e163f632c47ec6a49683a89;
tv[64]= 384'hffffffff80000000000000000000000000000f18aff94274696d9b61848bd50ac5e5;
tv[65]= 384'hfffffffc0000000000000000000000000082408571c3e2424540207f833b6dda69;
tv[66]= 384'hfffffffe0000000000000000000000000303ff996947f0c7d1f43c8f3027b9b75;
tv[67]= 384'hffffffff0000000000000000000000007df4daf4ad29a3615a9b6ece5c99518a;
tv[68]= 384'hffffffff8000000000000000000000000c72954a48d0774db0b4971c526260415;
tv[69]= 384'hfffffffc0000000000000000000000001df9b76112dc6531e07d2cfda04411f0;
tv[70]= 384'hfffffffe0000000000000000000000008e4d8e699119e1fc87545a647fb1d34f;
tv[71]= 384'hffffffff0000000000000000000000000e6c4807ae11f36f091c57d9fb68548d1;
tv[72]= 384'hffffffff80000000000000000000000008ebf73aad49c82007f77a5c1ccec6ab4;
tv[73]= 384'hfffffffc0000000000000000000000004fb288cc2040049001d2c7585ad123fc;
tv[74]= 384'hfffffffe0000000000000000000000004497110efb9dceb13e2b13fb4465564;
tv[75]= 384'hffffffff0000000000000000000000075550e6cb5a88e49634c9ab69eda0430;
tv[76]= 384'hffffffff80000000000000000000000b6768473ce9843ea66a81405dd50b345;
tv[77]= 384'hfffffffc0000000000000000000000cb2f430383f9084e03a653571e065de6;
tv[78]= 384'hfffffffe0000000000000000000000ff4e66c07bae3e79fb7d210847a3b0ba;
tv[79]= 384'hffffffff0000000000000000000000007b90785125505fad59b13c186dd66ce3;
tv[80]= 384'hffffffff80000000000000000000000008b527a6aebdaec9eaef8eda2cb7783e5;
tv[81]= 384'hfffffffc0000000000000000000000043fdaf53ebbc9880c228617d6a9b548b;
tv[82]= 384'hfffffffe0000000000000000000000053786104b9744b98f052c46f1c850d0b;
tv[83]= 384'hffffffff0000000000000000000000b5ab3013dd1e61df06cbaf34ca2aee78;
tv[84]= 384'hffffffff80000000000000000000000007470469be9723030fdcc73a8cd4fbb10;
tv[85]= 384'hfffffffc0000000000000000000000a35a63f5343ebe9ef8167bcb48ad122e;
tv[86]= 384'hfffffffe0000000000000000000000fd8687f0757a210e9fdf181204c30863;
tv[87]= 384'hffffffff0000000000000000000007a181e84bd5457d26a88fbae96018fb0;
tv[88]= 384'hffffffff80000000000000000000000653317b9362b6f9b9e1a580e68d494b5;
tv[89]= 384'hfffffffc0000000000000000000000995c9dc0b689f03c45867b5faa5c18d1;
tv[90]= 384'hfffffffe0000000000000000000000077a4d96d56dda39bb9aabecfc75729fd;
tv[91]= 384'hffffffff0000000000000000000000084be19e053635f09f2665e7bae85b42d;
tv[92]= 384'hffffffff80000000000000000000000032cd65284292 6aea4aa6137bb2be2b5e;
tv[93]= 384'hfffffffc0000000000000000000000493d4a4f38ebb337d10aa84e9171a554;
tv[94]= 384'hfffffffe0000000000000000000000d9bff7ff454b0ec5a4a2a69566e2cb84;
tv[95]= 384'hffffffff0000000000000000000003535d565ace3f31eb249ba2cc6765d7a;
tv[96]= 384'hffffffff80000000000000000000000f60e91fc3269eecf3231c6e9945697c6;
tv[97]= 384'hfffffffc0000000000000000000000ab69cfadf51f8e604d9cc37182f6635a;
```

```
tv[98]= 384'hfffffffffffffffffe00000000000000000000000000000000000000000000000007866373f24a0b6ed56e0d96fcdafb877;
tv[99]= 384'hffffffffffffffffff000000000000000000000000000000000000000000000001ea448c2aac954f5d812e9d78494446a;
tv[100]= 384'hffffffffffffffffff800000000000000000000000000000000000000000000acc5599dd8ac02239a0fef4a36dd1668;
tv[101]= 384'hffffffffffffffffffc000000000000000000000000000000000000000000d8764468bb103828cf7e1473ce895073;
tv[102]= 384'hffffffffffffffffffe000000000000000000000000000000000000000001b0d02893683b9f180458e4aa6b73982;
tv[103]= 384'hfffffffffffffffffff000000000000000000000000000000000000000096d9b017d302df410a937dcdb8bb6e43;
tv[104]= 384'hfffffffffffffffffff800000000000000000000000000000000000000ef1623cc44313cff440b1594a7e21cc6;
tv[105]= 384'hfffffffffffffffffffc000000000000000000000000000000000000284ca2fa35807b8b0ae4d19e11d7dbd7;
tv[106]= 384'hfffffffffffffffffffe000000000000000000000000000000000000f2e976875755f9401d54f36e2a23a594;
tv[107]= 384'hffffffffffffffffffff000000000000000000000000000000000000ec198a18e10e532403b7e20887c8dd80;
tv[108]= 384'hffffffffffffffffffff800000000000000000000000000000000000545d50ebd919e4a6949d96ad47e46a80;
tv[109]= 384'hffffffffffffffffffffc00000000000000000000000000000000dbdfb527060e0a71009c7bb0c68f1d44;
tv[110]= 384'hffffffffffffffffffffe00000000000000000000000000000000009cfa1322ea33da2173a024f2ff0d896d;
tv[111]= 384'hfffffffffffffffffffff00000000000000000000000000000000008785b1a75b0f3bd958dcd0e29318c521;
tv[112]= 384'hfffffffffffffffffffff80000000000000000000000000000000038f67b9e98e4a97b6df030a9fcdd0104;
tv[113]= 384'hfffffffffffffffffffffc0000000000000000000000000000000192afffb2c880e82b05926d0fc6c448b;
tv[114]= 384'hfffffffffffffffffffffe000000000000000000000000000000006a7980ce7b105cf530952d74daaf798c;
tv[115]= 384'hffffffffffffffffffffff000000000000000000000000000000ea3695e1351b9d6858bd958cf513ef6c;
tv[116]= 384'hffffffffffffffffffffff800000000000000000000000000006da0490ba0ba0343b935681d2cce5ba1;
tv[117]= 384'hffffffffffffffffffffffc00000000000000000000000000f0ea23af08534011c60009ab29ada2f1;
tv[118]= 384'hffffffffffffffffffffffe000000000000000000000000ff13806cf19cc38721554d7c0fcdcd4b;
tv[119]= 384'hfffffffffffffffffffffff000000000000000000000000006838af1f4f69bae9d85dd188dcdf0688;
tv[120]= 384'hfffffffffffffffffffffff80000000000000000000000036cf44c92d550bfb1ed28ef583ddf5d7;
tv[121]= 384'hfffffffffffffffffffffffc0000000000000000000000d06e3195b5376f109d5c4ec6c5d62ced;
tv[122]= 384'hfffffffffffffffffffffffe0000000000000000000000c440de014d3d610707279b13242a5c36;
tv[123]= 384'hffffffffffffffffffffffff0000000000000000000000f0c5c6ffa5e0bd3a94c88f6b6f7c16b9;
tv[124]= 384'hffffffffffffffffffffffff800000000000000000000003e40c3901cd7effc22bffc35dee0b4d9;
tv[125]= 384'hffffffffffffffffffffffffc000000000000000000000b63305c72bedfab97382c406d0c49bc6;
tv[126]= 384'hffffffffffffffffffffffffe00000000000000000000036bbaab22a6bd4925a99a2b408d2dbae;
tv[127]= 384'hfffffffffffffffffffffffff0000000000000000000307c5b8fcd0533ab98bc51e27a6ce461;
tv[128]= 384'hfffffffffffffffffffffffff800000000000000000000829c04ff4c07513c0b3ef05c03e337b5;
tv[129]= 384'hfffffffffffffffffffffffffc000000000000000000f17af0e895dda5eb98efc68066e84c54;
tv[130]= 384'hfffffffffffffffffffffffffe0000000000000000277167f3812afff1ffacb4a934379fc3;
tv[131]= 384'hffffffffffffffffffffffffff000000000000002cb1dc3a9c72972e425ae2ef3eb597cd;
tv[132]= 384'hffffffffffffffffffffffffff80000000000000036aeaa3a213e968d4b5b679d3a2c97fe;
tv[133]= 384'hffffffffffffffffffffffffffc0000000000000009241daca4fdd034a82372db50e1a0f3f;
tv[134]= 384'hffffffffffffffffffffffffffe0000000000000c14574d9cd00cf2b5a7f77e53cd57885;
tv[135]= 384'hfffffffffffffffffffffffffff000000000000793de39236570aba83ab9b737cb521c9;
tv[136]= 384'hfffffffffffffffffffffffffff8000000000016591c0f27d60e29b85a96c33861a7ef;
tv[137]= 384'hfffffffffffffffffffffffffffc0000000044fb5c4d4f5cb79be5c174a3b1c97348;
tv[138]= 384'hfffffffffffffffffffffffffffe000000000674d2b61633d162be59dde04222f4740;
tv[139]= 384'hffffffffffffffffffffffffffff00000000b4750ff263a65e1f9e924ccfd98f3e37;
tv[140]= 384'hffffffffffffffffffffffffffff8000000062d0662d6eaeddedebae7f7ea3a4f6b6;
tv[141]= 384'hffffffffffffffffffffffffffffc00000070c46bb30692be657f7eaa93ebad9897;
tv[142]= 384'hffffffffffffffffffffffffffffe0000000323994cfb9da285a5d9642e1759b224a;
tv[143]= 384'hfffffffffffffffffffffffffffff000000001dbf57877b7b17385c85d0b54851e371;
tv[144]= 384'hfffffffffffffffffffffffffffff8000000dfa5c097cdc1532ac071d57b1d28d1bd;
tv[145]= 384'hfffffffffffffffffffffffffffffc00000003a0c53fa37311fc10bd2a9981f513174;
tv[146]= 384'hfffffffffffffffffffffffffffffe00000000ba4f970c0a25c41814bdae2e506be3b4;
tv[147]= 384'hffffffffffffffffffffffffffffff0000000002dce3acb727cd13ccd76d425ea56e4f6;
tv[148]= 384'hffffffffffffffffffffffffffffff800000005160474d504b9b3eefb68d35f245f4b3;
tv[149]= 384'hffffffffffffffffffffffffffffffc000000041a8a947766635dec37553d9a6c0cbb7;
tv[150]= 384'hffffffffffffffffffffffffffffffe00000025d6cfe6881f2bf497dd14cd4ddf445b;
tv[151]= 384'hfffffffffffffffffffffffffffffff000000041c78c135ed9e98c096640647265da1e;
tv[152]= 384'hfffffffffffffffffffffffffffffff8000000005a4d404d8917e353e92a21072c3b2305;
tv[153]= 384'hfffffffffffffffffffffffffffffffc0000000002bc96846b3fdc71643f384cd3cc3eaf;
tv[154]= 384'hfffffffffffffffffffffffffffffffe00000000009ba4a9143f4e5d4048521c4f8877d88e;
tv[155]= 384'hffffffffffffffffffffffffffffffff000000000000000000000000a1f6258c877d5fcd8964484538bfc92c;
tv[156]= 384'h0000000000000000000000000000000080000000000000000000000003ad78e726c1ec02b7ebfe92b23d9ec34;
tv[157]= 384'h0000000000000000000000000000000c0000000000000000000000aae5939c8efdf2f04e60b9fe7117b2c2;
tv[158]= 384'h0000000000000000000000000000000e0000000000000000000000f031d4d74f5dcbf39daaf8ca3af6e527;
tv[159]= 384'h0000000000000000000000000000000f0000000000000000000000096d9fd5cc4f07441727df0f33e401a36;
tv[160]= 384'h0000000000000000000000000000000f8000000000000000000000030ccdb044646d7e1f3ccea3dca08b8c0;
tv[161]= 384'h0000000000000000000000000000000fc0000000000000000000000016ae4ce5042a67ee8e177b7c587ecc82;
tv[162]= 384'h0000000000000000000000000000000fe0000000000000000000000b6da0bb11a23855d9c5cb1b4c6412e0a;
tv[163]= 384'h0000000000000000000000000000000ff0000000000000000000000db4f1aa530967d6732ce4715eb0ee24b;
tv[164]= 384'h0000000000000000000000000000000ff8000000000000000000000a81738252621dd180a34f3455b4baa2f;
tv[165]= 384'h0000000000000000000000000000000ffc0000000000000000000077e2b508db7fd89234caf7939ee5621a;
tv[166]= 384'h0000000000000000000000000000000ffe0000000000000000000000b8499c251f8442ee13f0933b688fcd19;
tv[167]= 384'h0000000000000000000000000000000fff0000000000000000000000965135f8a81f25c9d630b17502f68e53;
tv[168]= 384'h0000000000000000000000000000000fff8000000000000000000008b87145a01ad1c6cede995ea3670454f;
tv[169]= 384'h0000000000000000000000000000000fffc0000000000000000000008eae3b10a0c8ca6d1d3b0fa61e56b0b2;
tv[170]= 384'h0000000000000000000000000000000fffe000000000000000000064b4d629810fda6bafdf08f3b0d8d2c5;
tv[171]= 384'h0000000000000000000000000000000ffff0000000000000000000d7e5dbd3324595f8fdc7d7c571da6c2a;
tv[172]= 384'h0000000000000000000000000000000ffff8000000000000000000f3f7275264e167fca9de2c1527d9606;
tv[173]= 384'h0000000000000000000000000000000ffffc000000000000000000ee79dd4f401ff9b7ea945d86666c13b;
tv[174]= 384'h0000000000000000000000000000000ffffe000000000000000000dd35cea2799940b40db3f819cb94c08b;
tv[175]= 384'h0000000000000000000000000000000fffff000000000000000006941cb6b3e08c2b7afa581ebdd607b87;
tv[176]= 384'h0000000000000000000000000000000fffff8000000000000000002c20f439f6bb097b29b8bd6d99aad799;
tv[177]= 384'h0000000000000000000000000000000fffffc000000000000000000625d01f058e565f77ae86378bd2c49b3;
tv[178]= 384'h0000000000000000000000000000000fffffe000000000000000000c0b5fd98190ef45fbb4301438d095950;
tv[179]= 384'h0000000000000000000000000000000ffffff000000000000000013001ff5d99806efd25da34f56be854b;
tv[180]= 384'h0000000000000000000000000000000ffffff8000000000000000003b594c60f5c8277a5113677f94208d82;
tv[181]= 384'h0000000000000000000000000000000fffffffc000000000000000e9c0fc1818e4aa46bd2e39d638f89e05;
tv[182]= 384'h0000000000000000000000000000000fffffffe000000000000000f8023ee9c3fdc45a019b4e985c7e1a54;
tv[183]= 384'h0000000000000000000000000000000ffffffff000000000000000035f40182ab4662f3023baec1ee796b57;
tv[184]= 384'h0000000000000000000000000000000ffffffff8000000000000000aebbad7303649b4194a6945c6cc3694;
tv[185]= 384'h0000000000000000000000000000000fffffffc000000000000000a2124bea53ec2834279bed7f7eb0f938;
tv[186]= 384'h0000000000000000000000000000000fffffffe000000000000000b9fb4399fa4facc7309e14ec98360b0a;
tv[187]= 384'h0000000000000000000000000000000ffffffff000000000000000c26277437420c5d634f715aea81a9132;
tv[188]= 384'h0000000000000000000000000000000ffffffff8000000000000000171a0e1b2dd424f0e089af2c4c10f32f;
tv[189]= 384'h0000000000000000000000000000000fffffffffc000000000000000007cadbe402d1b208fe735edce00aee7ce;
tv[190]= 384'h0000000000000000000000000000000fffffffffe000000000000000043b02ff929a1485af6f5c6d6558baa0f;
tv[191]= 384'h0000000000000000000000000000000ffffffffff000000000000000092faacc9bf43508bf8fa8613ca75dea;
tv[192]= 384'h0000000000000000000000000000000ffffffffff8000000000000000cb2bf8280f3f9742c7ed513fe802629c;
tv[193]= 384'h0000000000000000000000000000000fffffffffc000000000000000215a41ee442fa992a6e323986ded3f68;
tv[194]= 384'h0000000000000000000000000000000fffffffffe000000000000000021e99cf4f0f77cea836e11a2fe75fb1;
tv[195]= 384'h0000000000000000000000000000000ffffffffff000000000000000095e3a0ca9079e646331df8b4e70d2cd6;
tv[196]= 384'h0000000000000000000000000000000ffffffffff8000000000000000004afe7f120ce7613f74fc12a01a828073;
tv[197]= 384'h0000000000000000000000000000000fffffffffc000000000000000827f000e75e2c8b9d479beed913fe678;
tv[198]= 384'h0000000000000000000000000000000fffffffffe000000000000000035830c8e7aaefe2d30310ef381cbf691;
tv[199]= 384'h0000000000000000000000000000000ffffffffff000000000000000191aa0f2c8570144f38657ea4085ebe5;
```

```
tv[200]= 384'h000000000000000000000000000000000fffffffffff80000000000000000000085062c2c909f15d9269b6c18ce99c4f0;
tv[201]= 384'h000000000000000000000000000000000ffffffffffc0000000000000000000678034dc9e41b5a560ed239eeab1bc78;
tv[202]= 384'h000000000000000000000000000000000fffffffffffe000000000000000000c2f93a4ce5ab6d5d56f1b93cf19911c1;
tv[203]= 384'h000000000000000000000000000000000ffffffffffff0000000000000000001c3112bcb0c1dcc749d799743691bf82;
tv[204]= 384'h000000000000000000000000000000000fffffffffffff800000000000000000000c55bd75c7f9c881989d3ec1911c0d4;
tv[205]= 384'h000000000000000000000000000000000ffffffffffffc0000000000000000000ea2e6b5ef182b7dff3629abd6a12045f;
tv[206]= 384'h000000000000000000000000000000000fffffffffffffe000000000000000022322327e01780b17397f24087f8cc6f;
tv[207]= 384'h000000000000000000000000000000000ffffffffffffff0000000000000000c9cacb5cd11692c373b2411768149ee7;
tv[208]= 384'h000000000000000000000000000000000fffffffffffff80000000000000000a18e3dbbca577860dab6b80da3139256;
tv[209]= 384'h000000000000000000000000000000000ffffffffffffffc000000000000000079b61c37bf328ecca8d743265a3d425c;
tv[210]= 384'h000000000000000000000000000000000fffffffffffffe000000000000000d2d99c6bcc1f06fda8e27e8ae3f1ccc7;
tv[211]= 384'h000000000000000000000000000000000ffffffffffffff0000000000000001bfd4b91c701fd6b61b7f997829d663b;
tv[212]= 384'h000000000000000000000000000000000fffffffffffff80000000000000011005d52f25f16bdc9545a876a63490a;
tv[213]= 384'h000000000000000000000000000000000ffffffffffffffc0000000000000003a4d354f02bb5a5e47d39666867f246a;
tv[214]= 384'h000000000000000000000000000000000fffffffffffffe000000000000000d451b8d6e1e1a0ebb155fbbf6e7b7dc3;
tv[215]= 384'h000000000000000000000000000000000fffffffffffffff0000000000000006898d4f42fa7ba6a10ac05e87b9f2080;
tv[216]= 384'h000000000000000000000000000000000ffffffffffffff80000000000000000b611295e739ca7d9b50f8e4c0e754a3f;
tv[217]= 384'h000000000000000000000000000000000ffffffffffffffc0000000000000007d33fc7d8abe3ca1936759f8f5deaf20;
tv[218]= 384'h000000000000000000000000000000000fffffffffffffffe0000000000000003b5e0f566dc96c298f0c12637539b25c;
tv[219]= 384'h000000000000000000000000000000000fffffffffffffff0000000000000000f807c3e7985fe0f5a50e2cdb25c5109e;
tv[220]= 384'h000000000000000000000000000000000ffffffffffffffff80000000000000041f992a856fb278b389a62f5d274d7e9;
tv[221]= 384'h000000000000000000000000000000000ffffffffffffffffc0000000000000010d3ed7a6fe15ab4d91acbc7d0767ab1;
tv[222]= 384'h000000000000000000000000000000000fffffffffffffffe000000000000000021feecd45b2e675973ac33bf0c5424fc;
tv[223]= 384'h000000000000000000000000000000000ffffffffffffffff0000000000000001480cb3955ba62d09eea668f7c708817;
tv[224]= 384'h000000000000000000000000000000000ffffffffffffffff80000000000000066404033d6b72b609354d5496e7eb511;
tv[225]= 384'h000000000000000000000000000000000fffffffffffffffc0000000000000001c317a220a7d700da2b1e075b00266e1;
tv[226]= 384'h000000000000000000000000000000000fffffffffffffffe000000000000000ab3b89542233f1271bf8fd0c0f403545;
tv[227]= 384'h000000000000000000000000000000000fffffffffffffffff0000000000000000d93eae966fac46dca927d6b114fa3f9e;
tv[228]= 384'h000000000000000000000000000000000ffffffffffffffff80000000000000001bdec521316503d9d5ee65df3ea94ddf;
tv[229]= 384'h000000000000000000000000000000000ffffffffffffffffc0000000000000eef456431dea8b4acf83bdae3717f75f;
tv[230]= 384'h000000000000000000000000000000000fffffffffffffffe000000000000006f2519a2fafaa596bfef5cfa15c21b9;
tv[231]= 384'h000000000000000000000000000000000fffffffffffffffff00000000000251a7eac7e2fe809e4aa8d0d7012531a;
tv[232]= 384'h000000000000000000000000000000000ffffffffffffffffc80000000000003bffc16e4c49b268a20f8d96a60b4058;
tv[233]= 384'h000000000000000000000000000000000fffffffffffffffe000000000000e886f9281999c5bb3b3e8862e2f7c988;
tv[234]= 384'h000000000000000000000000000000000ffffffffffffffffe000000000005363bf90d61beef39f48dd625fcef1361;
tv[235]= 384'h000000000000000000000000000000000fffffffffffffffff0000000000004d37c850644563c69fd0acd9a049325b;
tv[236]= 384'h000000000000000000000000000000000ffffffffffffffff800000000000b87c921b91829ef3b13ca541ee1130a6;
tv[237]= 384'h000000000000000000000000000000000ffffffffffffffffc000000000002e65eb6b6ea383e109accce8326b0393;
tv[238]= 384'h000000000000000000000000000000000fffffffffffffffe0000000000009ca547f7439edc3e255c0f4d49aa8990;
tv[239]= 384'h000000000000000000000000000000000ffffffffffffffffff00000000005e652614c9300f37816b1f9fd0c87f9;
tv[240]= 384'h000000000000000000000000000000000fffffffffffffffff800000000014954f0b4697776f44494fe458d814ed;
tv[241]= 384'h000000000000000000000000000000000ffffffffffffffffc00000000007c8d9ab6c2761723fe42f8bb506cbcf7;
tv[242]= 384'h000000000000000000000000000000000ffffffffffffffffe0000000000db7e1932679fdd99742aab04aa0d5a80;
tv[243]= 384'h000000000000000000000000000000000fffffffffffffffffff00000000004c6a1c83e568cd10f27c2d73ded19c28;
tv[244]= 384'h000000000000000000000000000000000ffffffffffffffffff800000000090ecbe6177e674c98de412413f7ac915;
tv[245]= 384'h000000000000000000000000000000000ffffffffffffffffffc00000000090684a2ac55fe1ec2b8ebd5622520b73;
tv[246]= 384'h000000000000000000000000000000000fffffffffffffffffffe0000000007472f9a7988607ca79707795991035e6;
tv[247]= 384'h000000000000000000000000000000000ffffffffffffffffffff00000000056aff089878bf3352f8df172a3ae47d8;
tv[248]= 384'h000000000000000000000000000000000ffffffffffffffffffff80000000065c0526cbe40161b8019a2a3171abd23;
tv[249]= 384'h000000000000000000000000000000000fffffffffffffffffffffc00000000377be0be33b4e3e310b4aabda173f84f;
tv[250]= 384'h000000000000000000000000000000000ffffffffffffffffffffffe000000009402e9aa6f69de6504da8d20c4fcaa2f;
tv[251]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff00000000123c1f4af313ad8c2ce648b2e71fb6e1;
tv[252]= 384'h000000000000000000000000000000000fffffffffffffffffffffff800000001ffc626d30203dcdb0019fb80f726cf4;
tv[253]= 384'h000000000000000000000000000000000ffffffffffffffffffffffffc000000076da1fbe3a50728c50fd2e621b5ad885;
tv[254]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe0000000082eb8be35f442fb52668e16a591d1d6;
tv[255]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff0000000e656f9ecf5fe27ec3e4a73d00c282fb3;
tv[256]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff80000002ca8209d63274cd9a29bb74bcd77683a;
tv[257]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc00000079bf5dce14bb7dd73a8e3611de7ce026;
tv[258]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe0000003c849939a5d29399f344c4a0eca8a576;
tv[259]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff000000ed3c0a94d59bece98835da7aa4f07ca2;
tv[260]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff80000063919ed4ce10196438b6ad09d99cd795;
tv[261]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc000007678f3a833f19fea95f3c6029e2bc610;
tv[262]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe000003aa426831067d36b92be7c5f81c13c56;
tv[263]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff0000009272e2d2cdd11050998c845077a30ea0;
tv[264]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff80000088c4b53f5ec0ff814c19adae7f6246c;
tv[265]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc00004010a5e401fdf0a0354ddbcc0d012b17;
tv[266]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe0000a87a385736c0a6189bd6589bd8445a93;
tv[267]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff0000545f2b83d9616dccf60fa9830e9cd287;
tv[268]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff80004b706f7f92406352394037a6d4f4688d;
tv[269]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc000b7972b3941c44b90afa7b264bfba7387;
tv[270]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe0006f45732cf10881546f0fd23896d2bb60;
tv[271]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff0002e3579ca15af27f64b3c955a5bfc30ba;
tv[272]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff80034a2c5a91ae2aec99b7d1b5fa6780447;
tv[273]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc00a4d6616bd04f87335b0e53351227a9ee;
tv[274]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe007f692b03945867d16179a8cefc83ea3f;
tv[275]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff003bd141ee84a0e6414a26e7a4f281f8a2;
tv[276]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff80d1788f572d98b2b16ec5d5f3922b99bc;
tv[277]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc00833ff6f61d98a57b288e8c3586b85a6;
tv[278]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe08568261797de176bf0b43becc6285afb;
tv[279]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff0f9b0fda0c4a898f5b9e6f661c4ce4d07;
tv[280]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff88ade895913685c67c5269f8aae42983e;
tv[281]= 384'h000000000000000000000000000000000fffffffffffffffffffffffc39bde67d5c8ed8a8b1c37eb8fa9f5ac0;
tv[282]= 384'h000000000000000000000000000000000fffffffffffffffffffffffe5c005e72c1418c44f569f2ea33ba54f3;
tv[283]= 384'h000000000000000000000000000000000ffffffffffffffffffffffff3f5b8cc9ea855a0afa7347d23e8d664e;


for(n=0;n<284;n=n+1)
   begin
         @(posedge clk);
         #1;
         kld = 1;
         @(posedge clk);
         #1;
         kld = 0;
         @(posedge clk);

         while(!done)        @(posedge clk);

         //$display("INFO: (a) Vector %0d: xpected %x, Got %x %t", n, ciph, text_out, $time);

         if(text_out != ciph | (!text_out)==1'bx)
            begin
```

```
                $display("ERROR: (a) Vector %0d mismatch. Expected %x, Got %x",
                        n, ciph, text_out);
                error_cnt = error_cnt + 1;
            end


        while(!done2)       @(posedge clk);

        //$display("INFO: (b) Vector %0d: xpected %x, Got %x", n, plain, text_out2);

        if(text_out2 != plain | (!text_out2)==1'bx)
            begin
                $display("ERROR: (b) Vector %0d mismatch. Expected %x, Got %x",
                        n, plain, text_out2);
                error_cnt = error_cnt + 1;
            end

        @(posedge clk);
        #1;
    end


        $display("");
        $display("");
        $display("Test Done. Found %0d Errors.", error_cnt);
        $display("");
        $display("");
        repeat(10)          @(posedge clk);
        $finish;
end

assign tmp = tv[n];
assign key     = kld ? tmp[383:256] : 128'hx;
assign text_in = kld ? tmp[255:128] : 128'hx;
assign plain   = tmp[255:128];
assign ciph    = tmp[127:0];

always #5 clk = ~clk;

aes_cipher_top u0(
        .clk(               clk             ),
        .rst(               rst             ),
        .ld(                kld             ),
        .done(              done            ),
        .key(               key             ),
        .text_in( text_in           ),
        .text_out(          text_out  )
        );

aes_inv_cipher_top u1(
        .clk(               clk             ),
        .rst(               rst             ),
        .kld(               kld             ),
        .ld(                done            ),
        .done(              done2           ),
        .key(               key             ),
        .text_in( text_out  ),
        .text_out(          text_out2 )
        );

endmodule
```

```
Final Results
Top Level Output File Name         : aes_ori64_top.edn
Output Format                      : EDIF
crit                               : Area
Users Target Library File Name     : Virtex
Keep Hierarchy                     : NO
Macro Generator                    : Macro+

Macro Statistics
# RAMs                             : 1
  1408-bit dual-port RAM           : 1
# Registers                        : 99
  4-bit register                   : 2
  1-bit register                   : 97
# Multiplexers                     : 1
  64-bit 4-to-1 multiplexer        : 1
# Adders/Subtractors               : 8
  4-bit adder                      : 3
  8-bit adder                      : 1
  8-bit subtractor                 : 2
  4-bit subtractor                 : 2

Design Statistics
# IOs                              : 197

Cell Usage :
# BELS                             : 14962
#      INV                         : 17
#      LUT1                        : 1
#      LUT2                        : 907
#      LUT3                        : 4542
#      LUT4                        : 7697
#      MUXCY_L                     : 36
#      MUXF5                       : 1717
#      VCC                         : 1
#      XORCY                       : 44
# FlipFlops/Latches                : 1760
#      FD                          : 1103
#      FDE                         : 580
#      FDR                         : 22
#      FDRE                        : 32
#      FDRSE                       : 6
#      FDS                         : 2
#      FDSE                        : 7
#      LD                          : 8
# RAMS                             : 128
#      RAM16X1D                    : 128
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 196
#      IBUF                        : 131
#      OBUF                        : 65
=========================================================================


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Timing Summary:
---------------
Speed Grade: -6

   Minimum period: 23.042ns (Maximum Frequency: 43.399MHz)
   Minimum input arrival time before clock: 9.007ns
   Maximum output required time before clock: 6.887ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

-------------------------------------------------------------------------
```

```
   Path from Clock 'clk' rising to Clock 'clk' rising : 23.042ns
   (Slack: -23.042ns)
                          Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name
----------------------------------------  ------------
FD:C->Q               59   1.065   4.635  u1_sa20_5_1
LUT3:I2->O             6   0.573   1.665  u1_us22_I_SF69
LUT4:I3->O             1   0.573   0.000  u1_us22_I_203_LUT_11_F
MUXF5:I0->O            1   0.436   1.035  u1_us22_I_203_LUT_11
LUT4:I3->O             1   0.573   1.035  u1_us22_I_199_LUT_136
LUT3:I2->O             1   0.573   0.000  u1_us22_I_d_6_F
MUXF5:I0->O            1   0.436   1.035  u1_us22_I_d_6
LUT2:I0->O            21   0.573   2.925  u1_I_sa22_ark_6
LUT2:I1->O             2   0.573   1.206  u1_I__n0224_4
LUT3:I1->O             1   0.573   1.035  u1_I351_I_Result_4
LUT4:I0->O             1   0.573   1.035  u1_I380_I_Result
LUT4:I3->O             2   0.573   0.000  u1_I__n0008_4
FD:D                       0.342          u1_sa32_4
----------------------------------------
Total                     23.042ns

--------------------------------------------------------------------------------
   Path from Port 'start' to Clock 'clk' rising : 9.007ns
   (Slack: -9.007ns)
                          Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name
----------------------------------------  ------------
IBUF:I->O             11   0.768   2.070  start_IBUF
LUT4:I2->O            64   0.573   4.860  I__n0003_1
FDE:CE                     0.736          key_buf_47
----------------------------------------
Total                      9.007ns

--------------------------------------------------------------------------------
   Path from Clock 'clk' rising to Port 'text_out_0' : 6.887ns
    (Slack: -6.887ns)
                          Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name
----------------------------------------  ------------
FDE:C->Q               1   1.065   1.035  I_text_out_0
OBUF:I->O                  4.787          text_out_0_OBUF
----------------------------------------
Total                      6.887ns

==============================================================================

-->
```

```
Xilinx Mapping Report File for Design 'aes_ori64_top'
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.

Design Information
------------------
Command Line   : map -p V300E-BG352-8 -cm area -gm exact -k 4 -c 100 -tx off
aes_ori64_top.ngd
Target Device  : xv300e
Target Package : bg352
Target Speed   : -8
Mapper Version : virtexe -- D.22
Mapped Date    : Wed Apr 07 11:08:27 2004

Design Summary
--------------
   Number of errors:      1
   Number of warnings:    2
   Number of Slices:              6,979 out of  3,072  227%
   Number of Slices containing
      unrelated logic:               65 out of  6,979    1%
   Total Number Slice Registers:  1,760 out of  6,144   28%
      Number used as Flip Flops:                1,752
      Number used as Latches:                       8
   Total Number 4 input LUTs:    13,415 out of  6,144  218%
      Number used as LUTs:                     13,156
      Number used as a route-thru:                  3
      Number used for Dual Port RAMs:             256
      (Two LUTs used per Dual Port RAM)
   Number of bonded IOBs:           196 out of    260   75%
   Number of GCLKs:                   1 out of      4   25%
   Number of GCLKIOBs:                1 out of      4   25%
Total equivalent gate count for design:  114,785
Additional JTAG gate count for IOBs:  9,456
```

```
Final Results
Top Level Output File Name         : aes_new64_top.edn
Output Format                      : EDIF
crit                               : Area
Users Target Library File Name     : Virtex
Keep Hierarchy                     : NO
Macro Generator                    : Macro+

Macro Statistics
# RAMs                             : 1
  1408-bit dual-port RAM           : 1
# Registers                        : 66
  4-bit register                   : 1
  1-bit register                   : 65
# Adders/Subtractors               : 7
  4-bit adder                      : 2
  8-bit adder                      : 1
  8-bit subtractor                 : 2
  4-bit subtractor                 : 2

Design Statistics
# IOs                              : 197

Cell Usage :
# BELS                             : 8946
#      INV                         : 16
#      LUT2                        : 763
#      LUT3                        : 2890
#      LUT4                        : 4427
#      MUXCY_L                     : 33
#      MUXF5                       : 776
#      VCC                         : 1
#      XORCY                       : 40
# FlipFlops/Latches                : 1445
#      FD                          : 923
#      FDE                         : 458
#      FDR                         : 15
#      FDRE                        : 27
#      FDRSE                       : 6
#      FDS                         : 1
#      FDSE                        : 7
#      LD                          : 8
# RAMS                             : 128
#      RAM16X1D                    : 128
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 196
#      IBUF                        : 131
#      OBUF                        : 65
=========================================================================


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Timing Summary:
---------------
Speed Grade: -6

   Minimum period: 39.333ns (Maximum Frequency: 25.424MHz)
   Minimum input arrival time before clock: 9.727ns
   Maximum output required time before clock: 6.887ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

-------------------------------------------------------------------------
   Path from Clock 'clk' rising to Clock 'clk' rising : 39.333ns
   (Slack: -39.333ns)
                           Gate     Net
```

```
Cell:in->out      fanout   Delay   Delay  Logical Name
----------------------------------------  ------------
FDE:C->Q              99   1.065   6.435  option_buf_2
LUT3:I1->O             5   0.573   1.566  u0_I_in_s03_6
LUT4:I2->O             2   0.573   1.206  u0_us03_I_5_LUT_210
LUT3:I2->O            77   0.573   5.445  u0_us03_I_a_in_4
LUT4:I0->O             1   0.573   1.035  u0_us03_m0_I_157_LUT_218
LUT2:I1->O             1   0.573   1.035  u0_us03_m0_I_156_LUT_6
LUT4:I1->O             1   0.573   1.035  u0_us03_m0_I_155_LUT_109
LUT4:I3->O             1   0.573   1.035  u0_us03_m0_I_146_LUT_109
LUT4:I3->O             4   0.573   1.440  u0_us03_m0_I_g_4
LUT4:I3->O             2   0.573   1.206  u0_us03_I__n0035
LUT3:I2->O             9   0.573   1.908  u0_us03_I_d_7
LUT2:I0->O            13   0.573   2.250  u0_I_Isa03_ark_7
LUT2:I1->O             6   0.573   1.665
u0_I_inv_mix_col_4_pmul_e_13_xtime_177_xtime_1
LUT2:I1->O             1   0.573   1.035  u0_I728_I_Xo10
LUT4:I2->O             1   0.573   1.035  u0_I760_I_Result
LUT4:I0->O             1   0.573   0.000  u0_I__n0155_2
FD:D                       0.342          u0_Isa03_2
----------------------------------------
Total                     39.333ns

--------------------------------------------------------------------------------
   Path from Port 'start' to Clock 'clk' rising : 9.727ns
   (Slack: -9.727ns)
                          Gate     Net
Cell:in->out      fanout  Delay    Delay  Logical Name
----------------------------------------  ------------
IBUF:I->O             19   0.768   2.790  start_IBUF
LUT4:I2->O            64   0.573   4.860  I__n0003
FDE:CE                     0.736          text_in_buf_59
----------------------------------------
Total                      9.727ns

--------------------------------------------------------------------------------
   Path from Clock 'clk' rising to Port 'text_out_0' : 6.887ns
    (Slack: -6.887ns)
                          Gate     Net
Cell:in->out      fanout  Delay    Delay  Logical Name
----------------------------------------  ------------
FDE:C->Q               1   1.065   1.035  I_text_out_0
OBUF:I->O                  4.787          text_out_0_OBUF
----------------------------------------
Total                      6.887ns

================================================================================

-->
```

```
Xilinx Mapping Report File for Design 'aes_new64_top'
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.

Design Information
------------------
Command Line   : map -p V300E-BG352-8 -cm area -gm exact -k 4 -c 100 -tx off
aes_new64_top.ngd
Target Device  : xv300e
Target Package : bg352
Target Speed   : -8
Mapper Version : virtexe -- D.22
Mapped Date    : Wed Apr 07 11:43:50 2004

Design Summary
--------------
   Number of errors:      1
   Number of warnings:    1
   Number of Slices:              4,382 out of  3,072  142%
   Number of Slices containing
      unrelated logic:               68 out of  4,382    1%
   Total Number Slice Registers:  1,445 out of  6,144   23%
      Number used as Flip Flops:                1,437
      Number used as Latches:                       8
   Total Number 4 input LUTs:     8,341 out of  6,144  135%
      Number used as LUTs:                      8,082
      Number used as a route-thru:                  3
      Number used for Dual Port RAMs:             256
      (Two LUTs used per Dual Port RAM)
   Number of bonded IOBs:           196 out of    260   75%
   Number of GCLKs:                   1 out of      4   25%
   Number of GCLKIOBs:                1 out of      4   25%
Total equivalent gate count for design:  78,977
Additional JTAG gate count for IOBs:  9,456
```