AES Core
Specification

Author: Homer Hsing
homer.hsing@gmail.com

Rev. 0.1.2
February 21, 2013

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
0.1	10/18/2012	Homer Hsing	First Draft
0.1.1	10/30/2012	Homer Hsing	In section “performance”, change “128 bytes” to “128 bits”, change throughput.
0.1.2	02/21/2012	Homer Hsing	Explain how to encrypt or decrypt a message

Contents

1

Introduction

About AES

AES (Advanced Encryption Standard) is a specification published by the American National Institute of Standards and Technology in 2001, as FIPS 197.[1]

AES describes a symmetric-key algorithm, in which the same key is used for both encrypting and decrypting the data. The block size is restricted to 128 bits. The key size can be 128, 192, or 256 bits. [1]

AES operates on a 4×4 matrix of bytes, called the state. Some rounds of transformation converts the plaintext into the final cipher-text. The number of rounds is six plus the key size divided by 32. One round reads the state into four 4-byte variables y_0, y_1, y_2, y_3; transforms the variables; xor's them by a 16-byte round key; and puts the result into z_0, z_1, z_2, z_3.[3]

When targeting a variable-length plaintext, the plaintext must first be partitioned into separate cipher blocks, and then be encrypted under some mode of operation, generally using randomization based on an additional initialization vector.[4]

The cipher feedback (CFB) mode, output feedback (OFB) mode are specified in FIPS 81. The counter (CTR) mode is specified by NIST in SP800-38A.[4] The advantage of these modes is only using encryption algorithm for both encryption and decryption. So the AES hardware price may be reduced by 50% (not need decryption hardware).

About this project

This project has implemented AES encryption algorithm.

This project provides three cores, doing AES-128, AES-192 and AES-256 encryption separately.

The cores can be used in cipher feedback (CFB) mode, output feedback (OFB) mode, and counter (CTR) mode.

The maximum frequency is 324.6 MHz. The throughput is 37.5 G bytes/second if with a 300 MHz clock.

The features are as follows.

- ✓ Pipeline architecture
- ✓ Only one clock domain in entire core
- ✓ No latch
- ✓ Vendor-independent code

2

Architecture

Architecture of the core

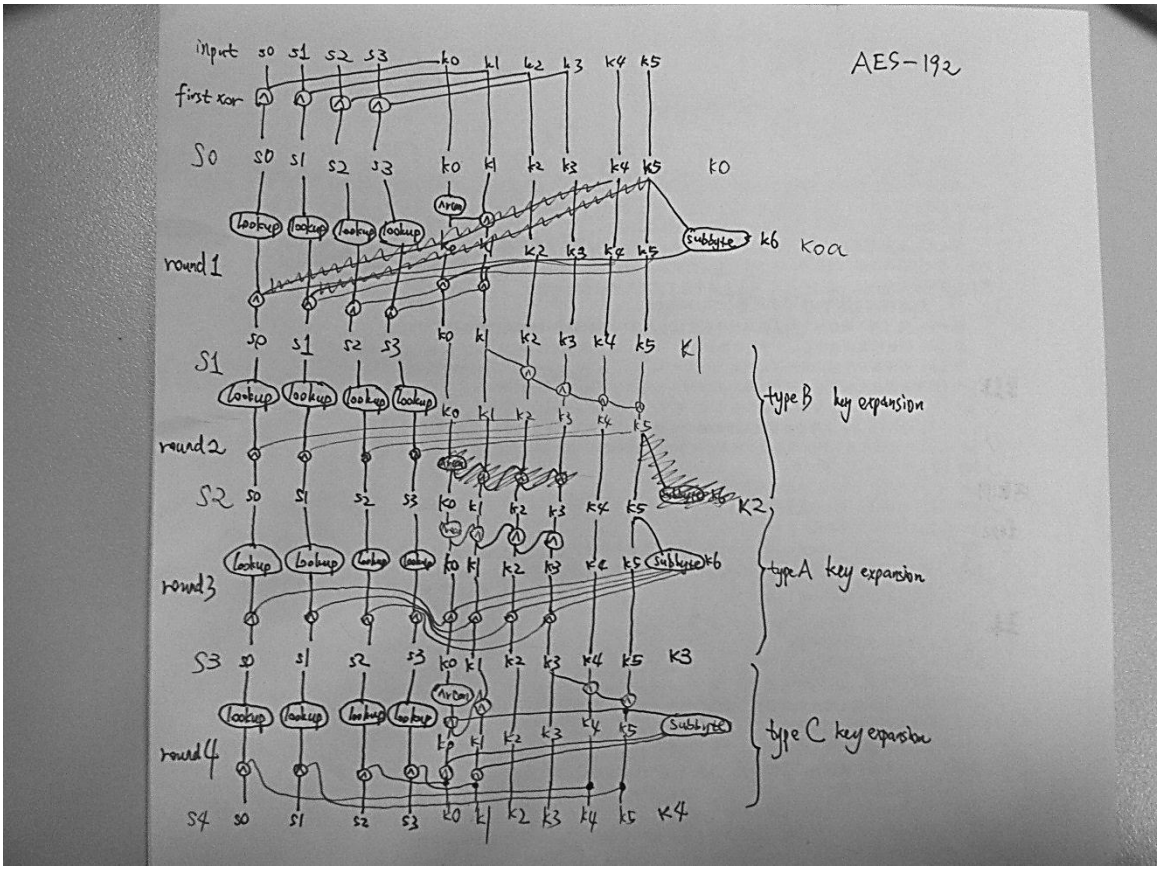

Figure 1: Architecture of AES-192 core

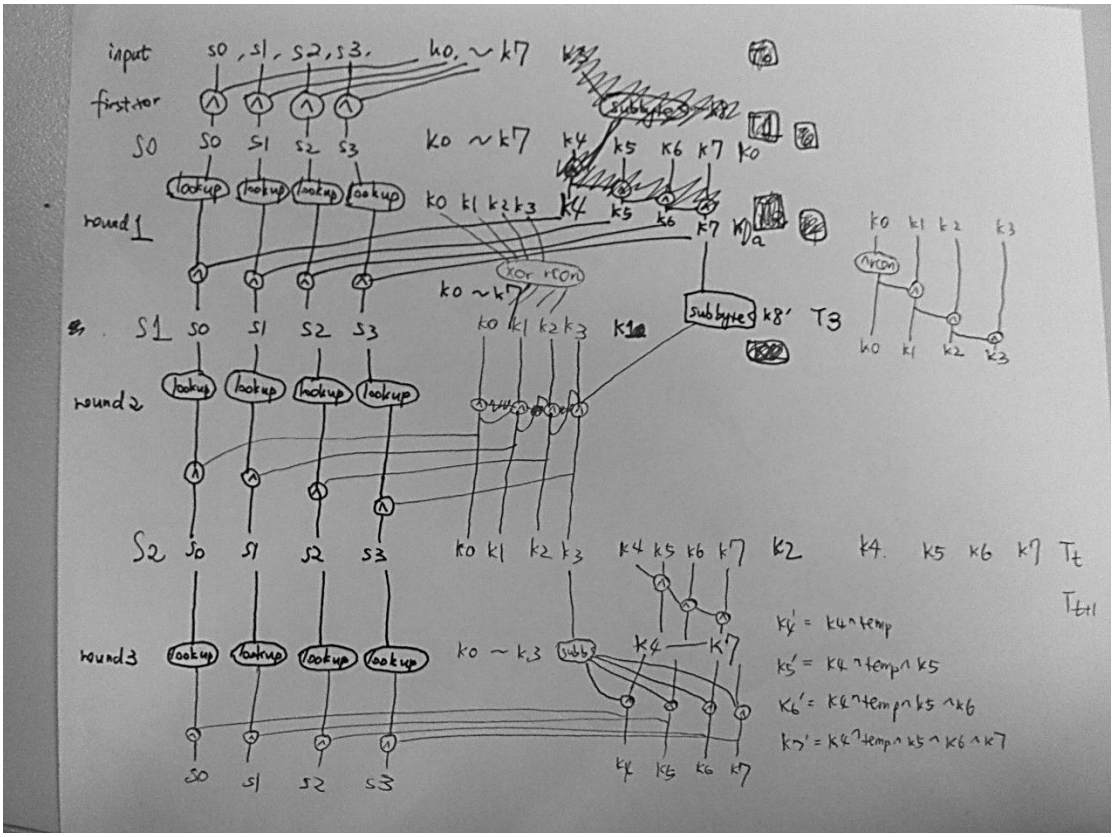

Figure 2: Architecture of AES-256 core

AES-128, AES-192, AES-256 cores have the similar architecture. Each of them consists of two pipelines. The first pipeline transforms the 16 bytes state. The second one computes the 16 bytes key used in each round.

State transformation pipeline

The first pipeline uses two clock cycle for each round of transformation.

In the first clock cycle, the pipeline does three steps, including sub_bytes – a substitution step where each byte is replaced with another according to a lookup table, $shift\_rows$ – a transposition step where each row of the state is shifted cyclically by a certain number of bytes, and $mix\_columns$ – a mixing step which operates on the columns of the state, combining the four bytes in each column.

As described in the original AES proposal, the three steps above are merged into 16 table lookups, see [2].

In the second clock cycle, the pipeline does the step of add_round_key – where each byte of the state is xor'ed with the round key.

Key expansion pipeline

The second pipeline also uses two clock cycle for computing every 16 bytes round key. Different computation methods are used for different AES key size. For brevity, only AES-128 is described here.

Denote the 16 byte round key in the last round as four 4-byte variables k_0, k_1, k_2, k_3. Denote the 16 byte round key in the current round as k'_0, k'_1, k'_2, k'_3. They obey following relation.

$$k'_0 \leftarrow subbyte\big(rotate(k_3)\big)\ xor\ Rconst\ xor\ k_0$$
$$k'_1 \leftarrow k'_0\ xor\ k_1;\ k'_2 \leftarrow k'_1\ xor\ k_2;\ k'_3 \leftarrow k'_2\ xor\ k_3$$

AES-128 core does not adopt the relation above, but use its equivalent relation.

$$v_0 \leftarrow k_0\ xor\ Rconst;$$
$$v_1 \leftarrow v_0\ xor\ k_1;$$
$$v_2 \leftarrow v_1\ xor\ k_2;$$
$$v_3 \leftarrow v_2\ xor\ k_3;$$
$$v_4 \leftarrow subbyte\big(rotate(k_3)\big);$$
$$k'_0 \leftarrow v_0\ xor\ v_4;$$
$$k'_1 \leftarrow v_1\ xor\ v_4;$$
$$k'_2 \leftarrow v_2\ xor\ v_4;$$
$$k'_3 \leftarrow v_3\ xor\ v_4$$

In the first clock cycle, $v_0 \ldots v_4$ are computed. In the second clock cycle, $k'_0 \ldots k'_3$ are computed.

The advantage is the distribution of combination logic in the two clock cycles is more uniform.

3 Interface

The AES cores implement the signals shown in the table below.

Input signals are synchronous and sampled at the rising edge of the clock.

Output signals are driven by flip-flops, and not directly connected to input signals by combinational logic.

For signals wider than 1 bit, the range is most significant bit down to least significant bit.

For example, the first byte of the plaintext is $state[127:120]$. The most significant bit of the first byte is $state[127]$. It is the same for key and out.

Table 1: Interface signals of AES-128 core

Signal name	Width	In/Out	Description
clk	1	In	Clock
$state$	128	In	Plaintext
key	128	In	Key
out	128	Out	Cipher-text. It is valid after 21 clock cycles when $state$ and key signals are valid.

Table 2: Interface signals of AES-192 core

Signal name	Width	In/Out	Description
clk	1	In	Clock
$state$	128	In	Plaintext
key	192	In	Key
out	128	Out	Cipher-text. It is valid after 25 clock cycles when $state$ and key signals are valid.

Table 3: Interface signals of AES-256 core

Signal name	Width	In/Out	Description
clk	1	In	Clock
state	128	In	Plaintext
key	256	In	Key
out	128	Out	Cipher-text. It is valid after 29 clock cycles when *state* and *key* signals are valid.

4

Timing

The number of clock cycles N for calculating the cipher-text is as follows.

Table 4: The value of N for different AES key size

Key-size	The value of N
128 bit	21
192 bit	25
256 bit	29

The signals $state$ and key are valid at the N_0-th clock cycle, if and only if the value of the signal out at the $(N_0 + N)$-th clock cycle is the encryption of plaintext $state$ under the key, for all integer $N_0 \geq 1$.

The value of $state$ and key at the N_0-th clock cycle can be different from the value of $state$ and key at the $(N_0 + D)$-th clock cycle, for all integer $N_0 \geq 1, D \geq 1$.

An example

Table 5: The value of $state$ and key and out

Clock cycle	$state$	key	out
N_0	s_0	k_0	Undefined value
$N_0 + 1$	s_1	k_1	Undefined value
$N_0 + 2$	s_2	k_2	Undefined value
…	…	…	…
$N_0 + N$	Any value	Any value	Cipher-text of s_0 under the key k_0
$N_0 + N + 1$	Any value	Any value	Cipher-text of s_1 under the key k_1
$N_0 + N + 2$	Any value	Any value	Cipher-text of s_2 under the key k_2

5

Usage

How to encrypt or decrypt a message

Suppose you use "aes_256" module, and the operation mode is CTR.

To encrypt something, let "aes_256.state" be a successive values of a "counter".

e.g., in clock cycle $T + 0$, "aes_256.state == $N + 0$";

in clock cycle $T + 1$, "aes_256.state == $N + 1$";

in clock cycle $T + 2$, "aes_256.state == $N + 2$".

Then "aes_256.out" is a binary sequence. Xor the binary sequence to the plain text to get the cipher text.

To decrypt something, let "aes_256.state" be a successive values of the SAME "counter".

Then "aes_256.out" is the SAME binary sequence. Xor the binary sequence to the cipher text to get the plain text.

If it is against your intuition, please read
http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

6

FPGA Implementation

Synthesis results (by Xilinx ISE version 14.2)

AES-128 core

Table 6: synthesis result of AES-128 core

Device	*Xilinx Virtex 6 XC6VLX240T-1FF1156*
Number of Slice Registers	3,968
Number of Slice LUTs	3,536
Number of bonded IOBs	385
Number of Block RAM/FIFO	86
Number of BUFG/BUFGCTRLs	1
Maximum Frequency	324.6MHz

AES-192 core

Table 7: synthesis result of AES-192 core

Device	*Xilinx Virtex 6 XC6VLX240T-1FF1156*
Number of Slice Registers	5,280
Number of Slice LUTs	4,264
Number of bonded IOBs	449
Number of Block RAM/FIFO	100
Number of BUFG/BUFGCTRLs	1
Maximum Frequency	324.6MHz

AES-256 core

Table 8: synthesis result of AES-256 core

Device	*Xilinx Virtex 6 XC6VLX240T-1FF1156*
Number of Slice Registers	6,848
Number of Slice LUTs	6,503
Number of bonded IOBs	513
Number of Block RAM/FIFO	121
Number of BUFG/BUFGCTRLs	1
Maximum Frequency	324.6MHz

Performance

The core can encrypt 128 bit per clock cycle.

The throughput is 38.4 G bit /second (= 4.8 G bytes/sec) if it is working with a 300 MHz clock.

7

Testbench

The file "testbench/simulation.do" is a batch file for ModelSim to compile the HDL files, setup the wave file, and begin function simulation. In order to make it work properly, the working directory of ModelSim must be the directory of "testbench".

The files "testbench/test_aes_128.v", "testbench/test_aes_192.v", and "testbench/test_aes_256.v" are the main test benches for the AES core. The test benches are self-checked. They feed input data to the core and compare the correct result with the output of the core. If the output is wrong, the test benches will display an error message.

Table 9: the object being tested by each test bench

File name	*the object being tested*
test_aes_128.v	the core doing AES-128 encryption
test_aes_192.v	the core doing AES-192 encryption
test_aes_256.v	the core doing AES-256 encryption

8

References

[1] Advanced Encryption Standard, http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

[2] J. Daemen and V. Rijmen. AES proposal: Rijndael. Original AES Submission to NIST, 1999.

[3] D. J. Bernstein and P. Schwabe. New AES software speed records. In INDOCRYPT 2008, volume 5365 of LNCS, pages 322-336, 2008.

[4] Block cipher modes of operation, http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation