AES-128 / Rijndael

Author: ssarwono@ieee.org
john@students.ee.itb.ac.id
yusa@vlsi.itb.ac.id

Version 1.0
June 12, 2002

1. Introduction

This is a simple AES/Rijndael Core. We have tried to create an implementation of this standard that would fit in to a low cost FPGA, and still would provide reasonably fast performance. This implementation is with a 128-bit key expansion module only. This document will describe the complete design and implementation of AES-128. It will not talk about the AES standard itself. We have separated the encryption and the decryption block and also provided with figures to make the design simpler and easier to understand. (Pictures can describe thousand of words, hehehehe………….)

2. Architecture

2.1 Encryption Block

By John Purba (john@students.ee.itb.ac.id)

The encryption block comprises 6 blocks:

- BlokInput, interface of input data and key
- KeyExpander, generator the round key
- Kontroler, controls each block
- Fungsi_Round, implements Rijndael algorithm (round calculations)
- BlokOutput, interface of output data

The encryption algorithm has been designed this way that the generation of round key and the round calculations can be parallely executed. The advantage of this design is the fact that we do not need to store the round key since they are currently calculated. Interconnection of each block is depicted in figure 1.

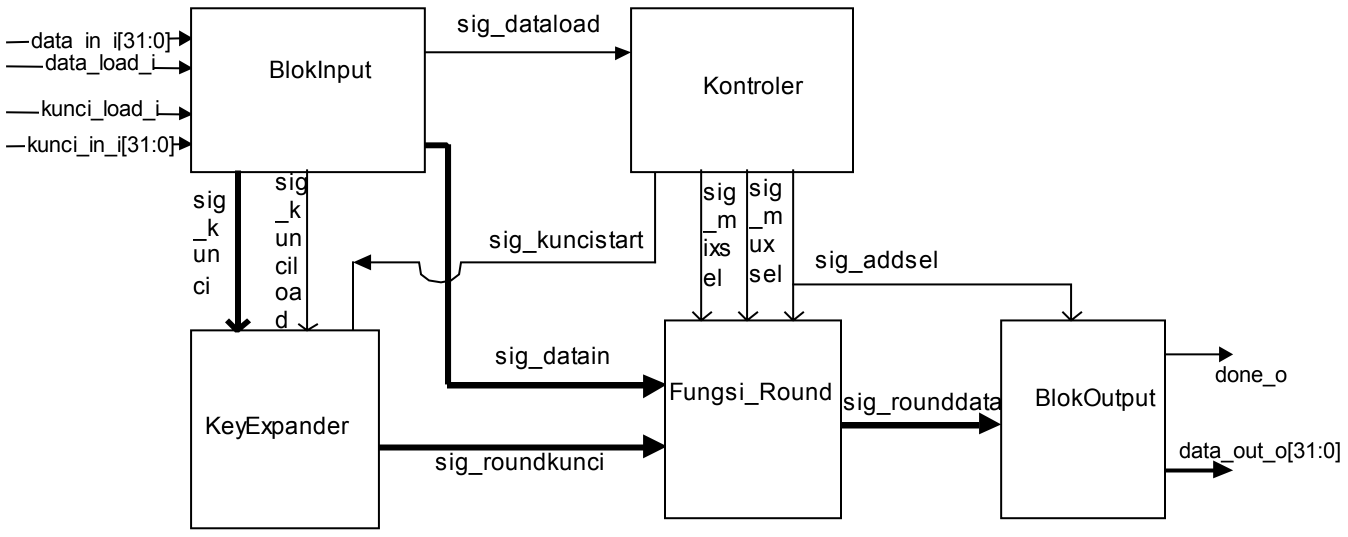

Figure 1

a. BlokInput

BlokInput is an interface for data or key input. The length of data or key supported is 128 bits, so we can use the same design for data input and key input. This block will take 32 bits data and shift them to next register. Four-clock cycle will be needed to take a complete 128 bit data. A simple controller is used here to identify that a complete of 128 bits data has been accepted. The State Diagram of FsmInput and BlokInput are depicted in figure 2.

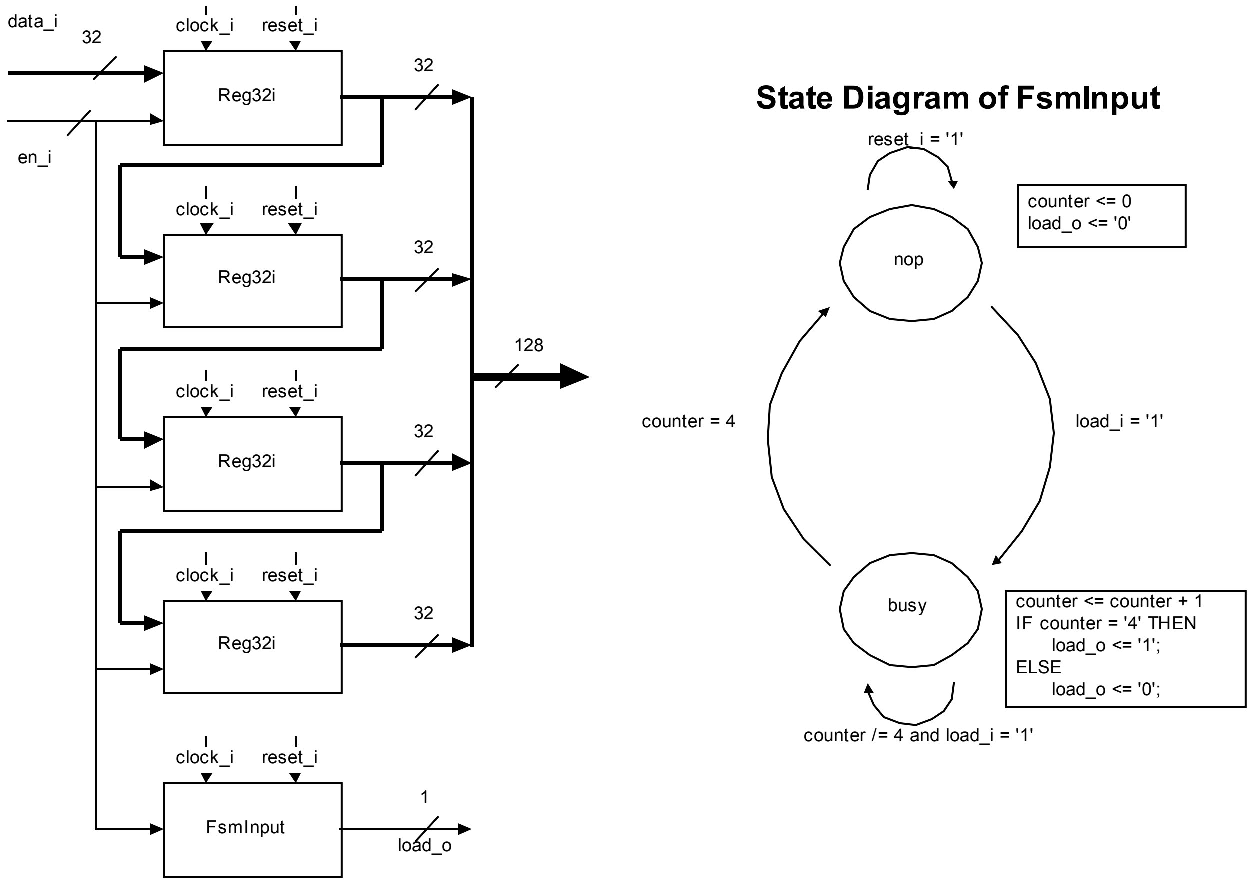

Figure 2

b. BlokOutput

This block takes 128 bit block data to output. The first one block of 128 bits is divided to 4 blocks of 32 bits and then each blocks of data is taken to output every one-clock cycle. Data output register is depicted in figure 3.

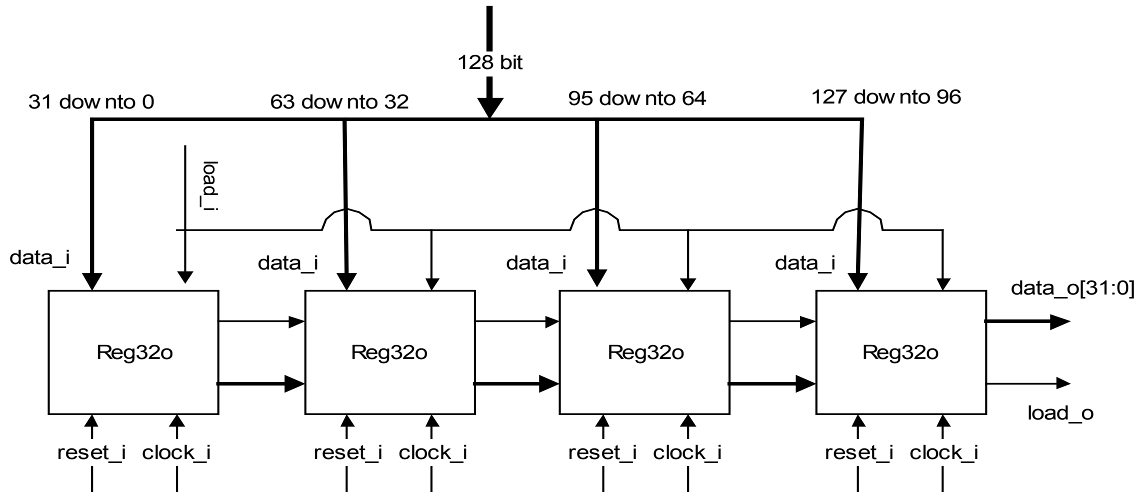

Figure 3

c. Kontroler

Kontroler block controls the **KeyExpander** and **Fungsi_Round** block. The state diagram of Control block is depicted in figure 4.

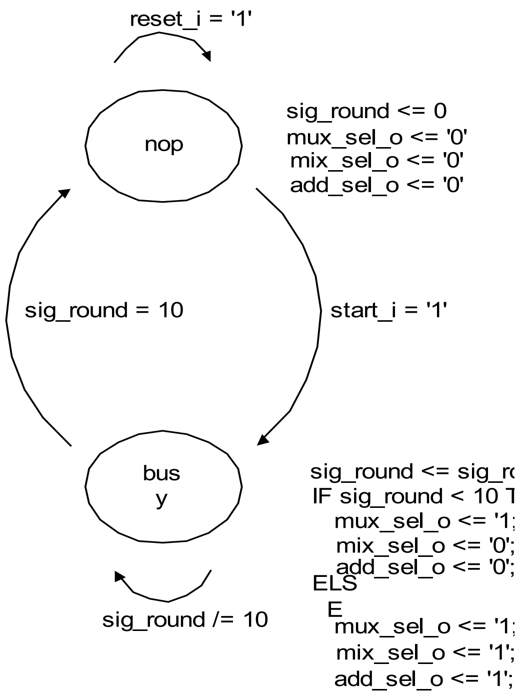

Figure 4

d. Key Expander

Key Expander responsible to generate round key for every round from the initial key. Based on the algorithm specification, if we use 128-bit key, 10 rounds will be needed. Hence the Key Expander will generate 10 round keys. The Key Expander is depicted in figure 5.

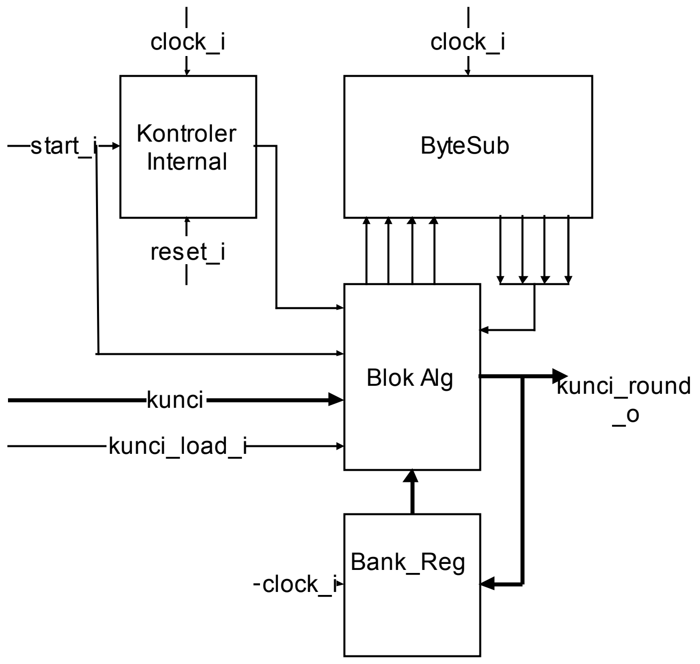

Figure 5

The Blok Alg implements the XOR operations and RotByte function (take a look again FIPS-197). Kontroler Internal controls the Key Expander and its state diagram illustrated in figure 6. The ByteSub block is the same as ByteSub for Rijndael Alg Block.

State Diagram of Kontroler Internal

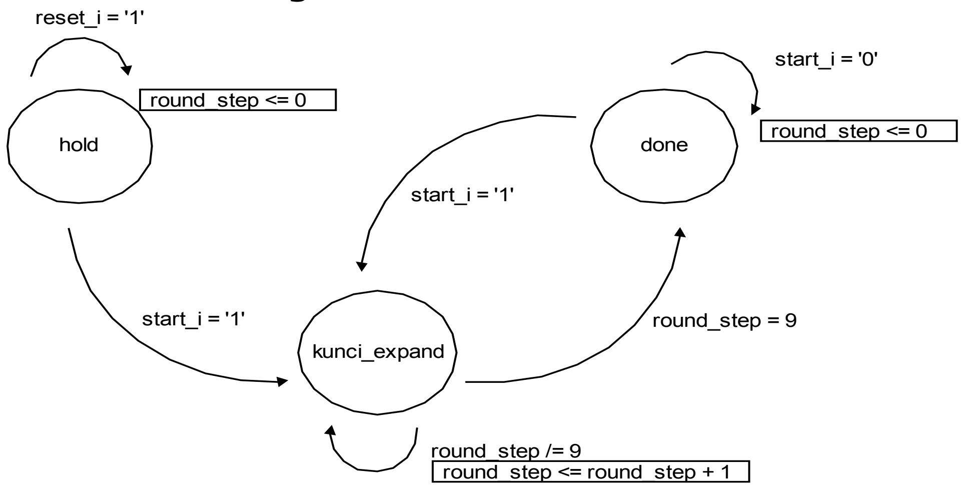

Figure 6

e. Fungsi_Round

The implementation of **Fungsi_Round** has been designed this way that it can work as initial round, standard round, and final round. The Fungsi_Round module is depicted in figure 7.

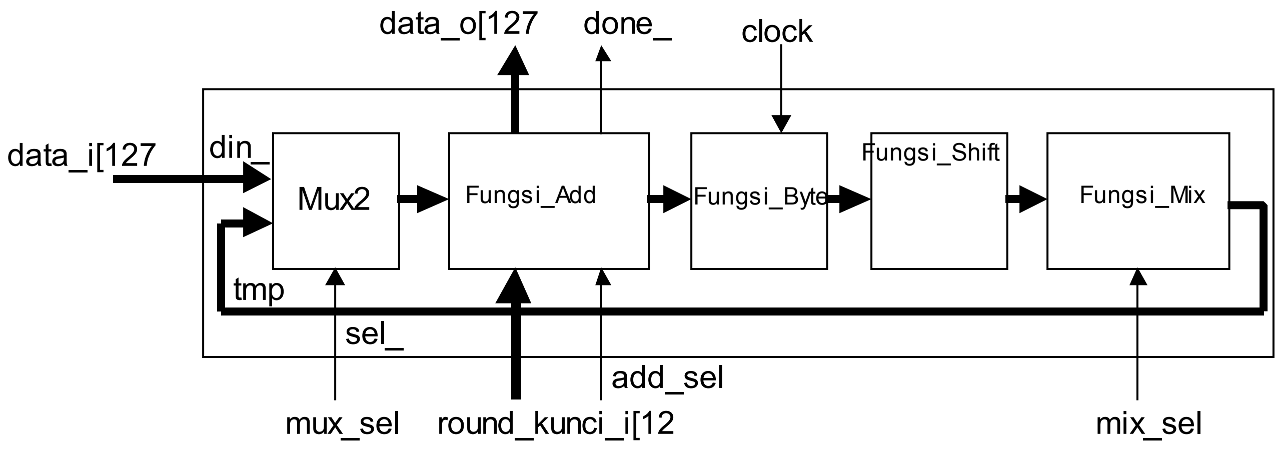

Figure 7

- **Mux21**
  Mux21 is a simple multiplexer 2 to 1. The format of data (din_i) is changed in this block, a block of 128 bits sequence to matrix 4x4 called State to make easier in designing other blocks (AddRoundKey, ByteSub, ShiftRow. MixColumn)

- **Fungsi_Add**
  Implements AddRoundKey transformation, a simple XOR between State and round key.

- **Fungsi_Byte**
  Implements ByteSub transformation, contains 16 S-Box working in parallel. Dual Port Block RAM will be used to implements 2 S-Box, which emulate the ROM memory with configuration of 256x8 bits.

- **Fungsi_Shift**
  Implements ShiftRow transformation. The position of bytes in State will shifted cyclical by offsets. The first row is shifted by zero, the second row is shifter by one, two shifts the third row, and the fourth row is shifted by three. The implementation is realized by hardwiring and do not need gate resource.

- **Fungsi_Mix**
  Implements MixColumn transformation. The columns of State are viewed as the coefficients of polynomial over GF (2^8) of degree smaller than three. This polynomial is multiplied by four terms fixed polynomial a(x), $\{03)x^3 + \{01\}x^2 + \{01\}x + \{02\}$, modulo the polynomial $x^4 + 1$.The multiplication with four terms fixed polynomial can be simplified by using the matrix form. The implementation of this transformation can be realized as shift and XOR operations.

2.2 Decryption Block

By Made Yusadana (yusa@vlsi.itb.ac.id)

Encryption and decryption differ in key expansion and round calculations only. We just modified the **KeyExpander, Fungsi_Round,** and **Kontroler** that depicted in figure 1. Therefore, we can use the same output and input interfaces.

a. Kontroler

Kontroler block controls the **KeyExpander** and **Fungsi_Round** block. The state diagram of Control block is depicted in figure 8.

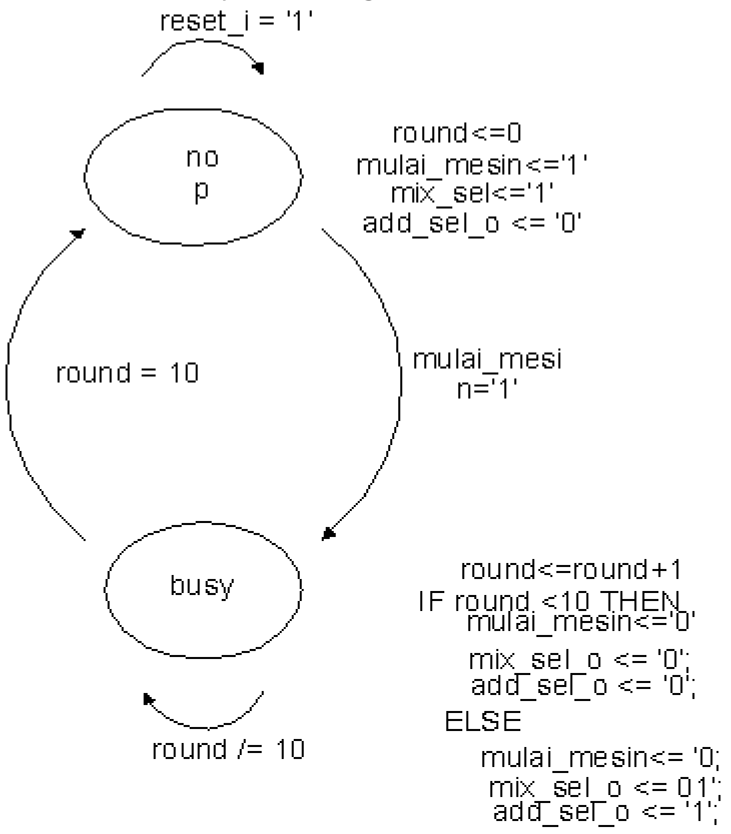

Figure 8

b. Fungsi_Round

Figure 9 depicted the Fungsi_Round for decryption block.

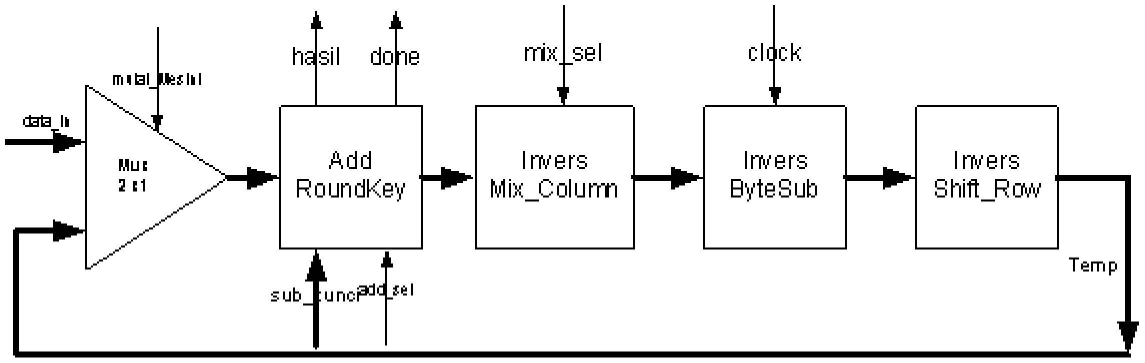

Figure 9

c. KeyExpander

The author wrote the behavioral code for this KeyExpander. He also wrote his own ROM to implement to ByteSub transformation.

3. IOs

Name	Width	Direction	Description
Encryption Block			
clock_i	1	I	Core clock
reset_i	1	I	Active high synchronous reset
data_in_i	32	I	Input text block
data_load_i	1	I	Input load
kunci_in_i	32	I	Key
kunci_load_i	1	I	Key load
data_out_o	32	O	Output text block
done_o	1	O	Output high : valid
Decryption Block			
clock_i	1	I	Core clock
reset_i	1	I	Active high synchronous reset
data_in	32	I	Input text block
data_load	1	I	Input load
key_in	32	I	Key
key_load	1	I	Key load
data_out	32	O	Output text block
done	1	O	Output high : valid

4. Simulation

In this section, we will provide the simulation result for Encryption block.

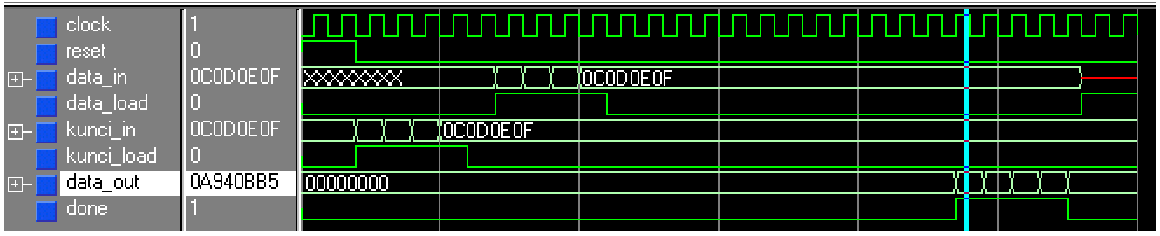

Figure 10

5. Implementation Result

The encryption algorithm has been implemented to FPGA Xilinx Virtex V300pq240. V300pq240 has 20 built-in Dual Port Block RAM so that we did not write our own RAM/ROM to implement to ByteSub transformation (We personally suggest you not to write your own ROM in FPGA implementation unless your FPGA do not has built-in RAM).

The implementation parameters for encryption are:

- Speed
  The maximum frequency is 51 MHz. Hence; the throughput is 593.45 Mbps (51x128/11).
- Area
  The required circuit area is 666 slices.

The implementation parameters for decryption are:

- Speed
  The maximum frequency is 23 MHz. Hence; the throughput is 267.63 Mbps (23x128/11).
- Area
  The required circuit area is 666 slices.