Mini AES 128

Arif E. Nugroho

arif_endro@opencores.org

VLSI Research Group
LabTek VIII Institut Teknologi Bandung
Jl. Ganesha 10 Bandung 40141
West Java, Indonesia

Contents

1

AES 128

1.1. Introduction

The National Institute of Standards and Technology (NIST) choose the Rijndael algorithm as the new Advanced Encryption Standard (AES) in 2001. Rijndael algorithm is a symmetric block cipher that can process data block of 128 bits, using cipher keys with length of 128, 192, and 256 bits. The algorithm can be used on different key length and may be referred to as ″AES-128″, ″AES-192″, and ″AES-256″. This crypto core is a hardware implementation of Rijndael algorithm that process 128 bit block of data using 128 bit key or ussually called as AES-128.

1.2. Circuit Architecture

The architecture of this implementation is based on the paper described by P. Chodowiec [1], the schematic diagram of circuit implementation is as follows:

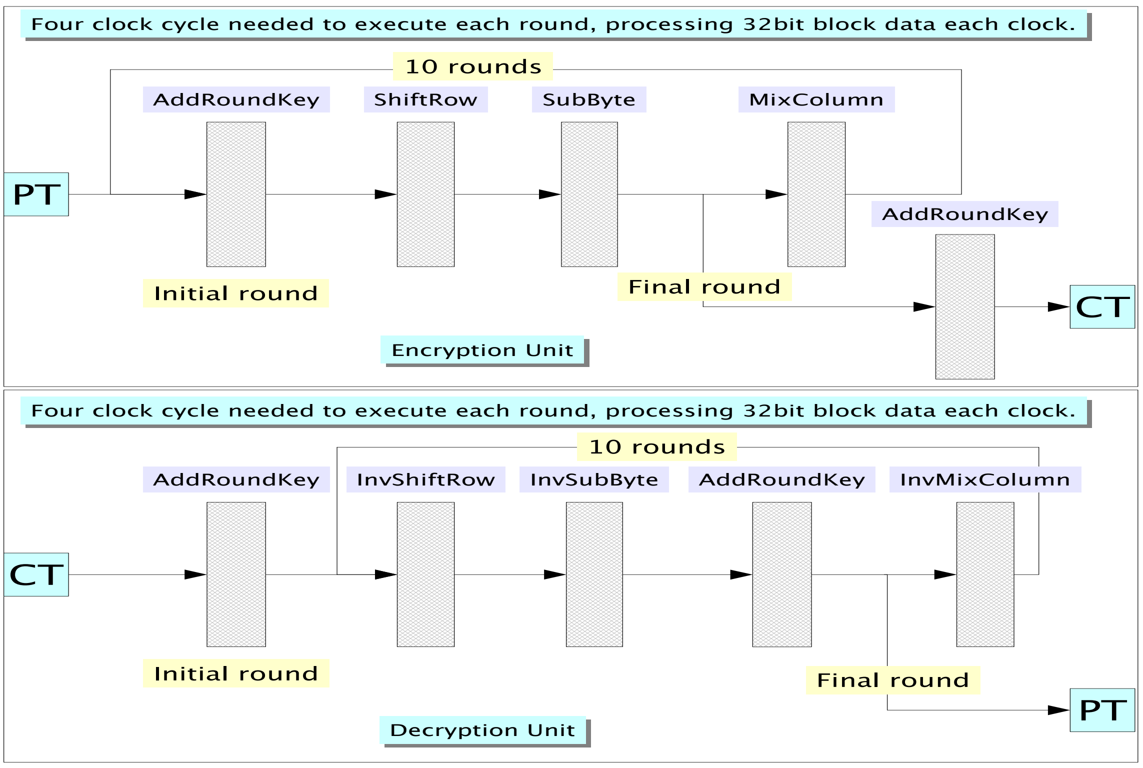

Figure 1-1: Circuit schematic

1.3. Simulation

Simulation is done by ModelSim 6.0 SE, the simulation is performed to verify the correctness of design. The encryption and decrption units has been verified using Electronic Codebook (ECB) method and passed 128 test vector verification phase of Tables Known Answer Test (KAT).

1.4. Synthesize

This design has been synthesized using ISE Xilinx 6.3i, here is the summary of the area utilization in FPGA Xilinx:

XC2V2000–FF896–4	
Slices	03973/10752
Slices Flip Flops	03940/21504
4 input LUT	04231/21504
Block RAM	00008/00056

Table 1-1: Area utilizations summary

The maximum clock frequency is 50.594 MHz (Minimum period 19.765ns)

1.5. Circuit Explanation

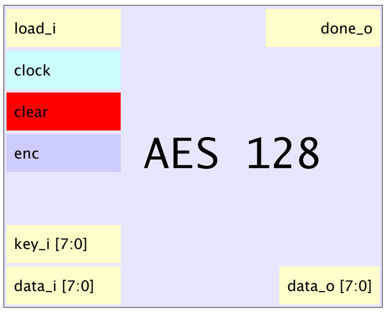

Figure 1-2: AES 128 Input - Output Pin

The AES 128 is composed from four subcircuit, these subcircuit are :

- ShiftRow
- SubByte
- MixColumn
- KeyScheduler

1.5.1. ShiftRow

ShiftRow transformation is performed by arranging the sequence of input data to be processed, these transformation is performed this way:

$$\left\{\begin{array}{lcl} input & & output \\ 0,5,a,f & => & 0,1,2,3 \\ 4,9,e,3 & => & 4,5,6,7 \\ 8,d,2,7 & => & 8,9,a,b \\ c,1,6,b & => & c,d,e,f \end{array}\right\}$$

the InvShiftRow transformations operations is performed using the following sequence of input data:

$$\left\{\begin{array}{lcl} input & & output \\ 0,d,a,7 & => & 0,1,2,3 \\ 4,1,e,b & => & 4,5,6,7 \\ 8,5,2,f & => & 8,9,a,b \\ c,9,6,3 & => & c,d,e,f \end{array}\right\}$$

1.5.2. SubByte

SubByte transformation is implemented using dedicated block RAM, SubByte transformation occupy 4 Kb Block RAM in 512x8 configurations.

1.5.3. MixColumn

The MixColumn is implemented using matrix calculation of following equation:

$$c(x) = \, '03' \, x^3 + \, '01' \, x^2 + \, '01' \, x + \, '02'. \qquad (1\text{-}1)$$

matrix form of above equation is:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The InvMixColumn operations is performed using following equation:

$$d(x) = \ '0b' \, x^3 + \ '0d' \, x^2 + \ '09' \, x + \ '0e'. \tag{1-2}$$

in matrix representation is:

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} 0e\ 0b\ 0d\ 09 \\ 09\ 0e\ 0b\ 0d \\ 0d\ 09\ 0e\ 0b \\ 0b\ 0d\ 09\ 0e \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

The InvMixColumn can be implemented using resource sharing, by following equation:

$$c(x) \otimes d(x) = \ '01' \tag{1-3}$$

then if we multiply both side with $d^2(x)$ then it become:

$$c(x) \otimes d^2(x) = d(x) \tag{1-4}$$

above equation state that we can get the InvMixColumn using multiplication of MixColumn operations and $d^2(x)$:

$$d^2(x) = \ '04' \, x^2 + \ '05'. \tag{1-5}$$

and matrix representation of above equation is:

$$\begin{bmatrix} e_0 \\ e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} 05\ 00\ 04\ 00 \\ 00\ 05\ 00\ 04 \\ 04\ 00\ 05\ 00 \\ 00\ 04\ 00\ 05 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

1.5.4. KeyScheduler

The KeyScheduler is implemented using the following schematic diagram:

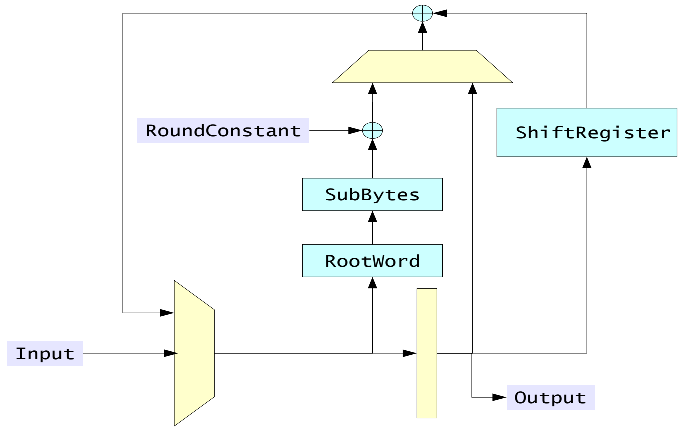

Figure 1-3: KeyScheduler Schematics

1.6. TODO

- Optimize Key Scheduler storage allocations implementations.
- Optimize folded register implementation.
- Implement TriState Buffer for switching SubBytes utilization.
- Update documentation.
- CleanUp code.

Bibliography

[1] Pawel Chodowiec and Kris Gaj, **Very Compact FPGA Implementation of the AES Algorithm**, CHES 2003, LNCS 2779, pp. 319-333

[2] Federal Information Processing Standards Publication 197, **Advanced Encryption Standard (AES)**, National Institute of Standards and Technology, 2001.

[3] Daemen J. and Rijmen V., **AES Proposal: The Rijndael Block Cipher**, http://www.esat.kuleuven.ac.be/˜ rijmen/rijndael/Rijndael.pdf

A

Informations

A.1. Tools

- **ModelSim 6.0** The Simulator
- **Xilinx 6.3i** The Synthesizer
- **VIM** (Vi IMproved) / **Emacs** The Editor
- **LaTeX** The Typesetter
- **OpenOffice.org 2.0** The Drawer

Version: 1.0