ECE 385

Fall 2020

Experiment 9

SOC with Advanced Encryption Standard in System Verilog

Zhicong Fan/Xin Jin
AB2/Online
Yucheng Liang

Introduction:

In this lab, we implemented the AES encryption and decryption algorithm. Specifically, the encryption algorithm was implemented on NIOS II processor and the decryption algorithm was implemented on the hardware using System Verilog. At the end, we compared the speed of the software encryption algorithm and hardware decryption algorithm. We found that decryption algorithm was hundreds of times faster than the encryption algorithm. Because the decryption process is essentially the inverse of encryption, it is safe to say that the hardware is a hundred times faster than the software.

Descriptions and Diagrams:

Description of the software encryptor:

The software encryptor was implemented on the NOIS II processor. It used C to run the code, but the processor communicated with the hardware to do the actual computation. The encryption algorithm is consisted of the following steps:

1. KeyExpansion: The input key is only 128 bits long. However, the encryption algorithm will need to use the key to encrypt the message eleven times, and in order to ensure the best encrypt performance, we will expand the key to 128*11 bits so each time the encryption will use different set of keys.
2. AddRoundKey: This operation adds the round key with the message by XORing them elementwise. This operation will be performed eleven times.
3. SubBytes: This operation updates each byes of the state by based on a lookup table.
4. ShiftRows: This operation circularly left shifts the nth row of the state by n-1 position.
5. MixColumns: This operation updates the state by doing a linear transformation over GF(2^8). Each four bytes of each word are transformed and combine to generate a new word.

When doing the encryption, message will do the AddRoundKey operation. Then, it will go through the operations SubBytes, ShiftRows, MixColumns, and AddRoundKey in sequence. This sequence will be executed nine times. After these nine rounds of the executions, the message will do the last round of operation: SubBytes, ShiftRows, AddRoundKey. The last round is the same as the previous rounds except a MixColumns operation.

Description of the hardware decryptor:

The decryption algorithm is essentially the same as the encryption except each operation is done in a reversed way. The decryption algorithm is written in System Verilog and run on the hardware directly. The encryption algorithm is consisted of the following steps:

1. KeyExpansion: Similar to the encryption, this operation also expands the key to 128*11 bist.
2. AddRoundKey: Same to the encryption, this operation adds the round key with the message by XORing them elementwise. This operation will be performed eleven times.

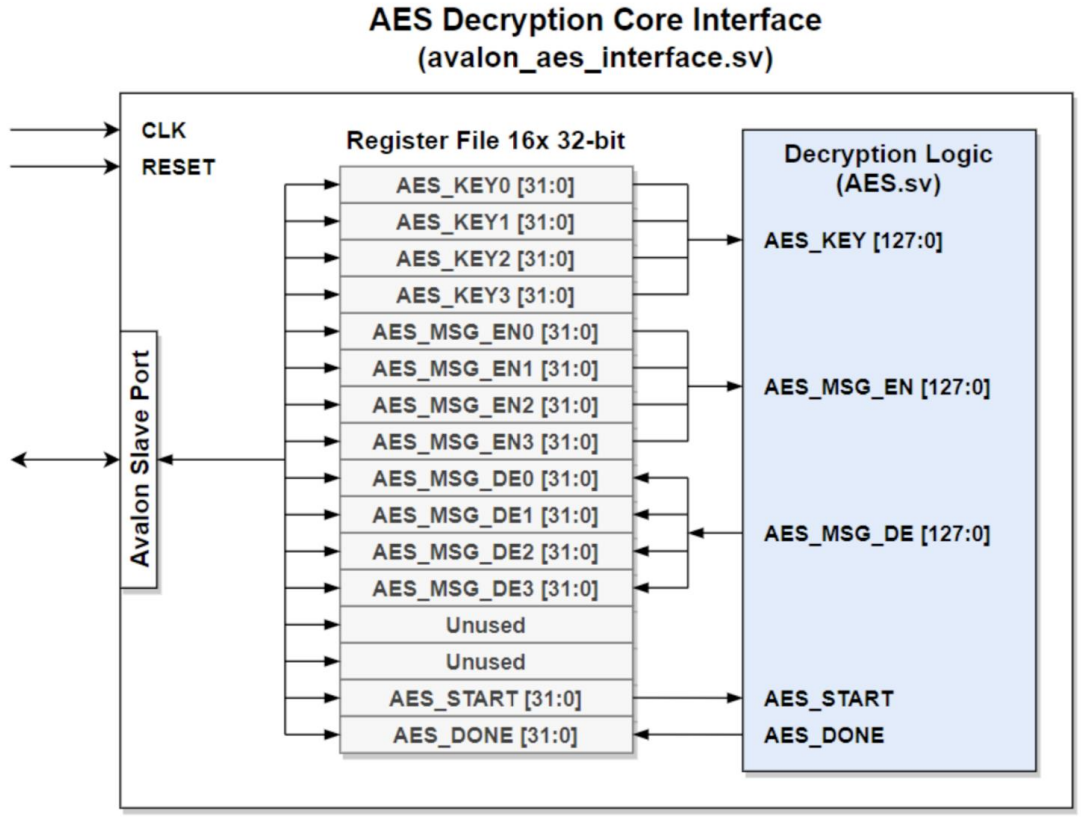
3. InvSubBytes: Similar to the encryption, this operation updates each byes of the state by based on a lookup table, which is the inverse of the table in the encryption.
4. InvShiftRows: Similar to the encryption, this operation circularly right shifts the nth row of the state by n-1 position.
5. InvMixColumns: Similar to the encryption, this operation updates the state by doing a linear transformation over GF(2^8). Each four bytes of each word are transformed and combine to generate a new word. This is the inverse of the MixColumns in encryption.

When doing the decryption, message will do the AddRoundKey operation. Then, it will go through the operations InvShiftRows, InvSubBytes, AddRoundKey, and InvMixColumns in sequence. This sequence will be executed nine times. After these nine rounds of the executions, the message will do the last round of operation: InvShiftRows, InvSubBytes, AddRoundKey. The last round is the same as the previous rounds except an InvMixColumns operation.

Description of the hardware/software interface:

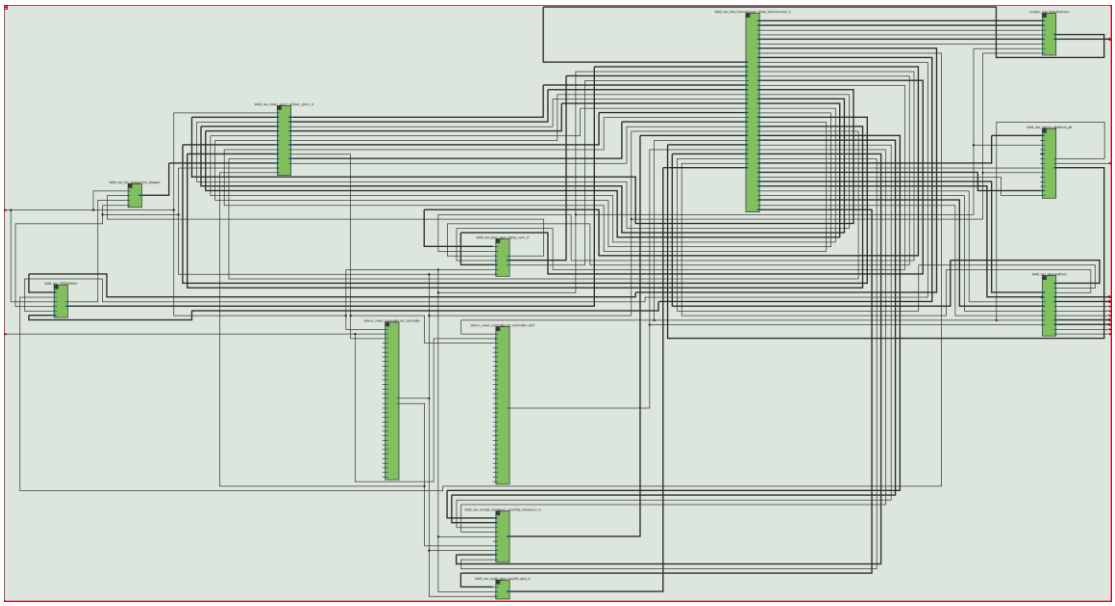
```
module avalon_aes_interface (
    // Avalon Clock Input
    input logic CLK,

    // Avalon Reset Input
    input logic RESET,

    // Avalon-MM Slave Signals
    input  logic AVL_READ,                  // Avalon-MM Read
    input  logic AVL_WRITE,                 // Avalon-MM Write
    input  logic AVL_CS,                        // Avalon-MM Chip Select
    input  logic [3:0] AVL_BYTE_EN,     // Avalon-MM Byte Enable
    input  logic [3:0] AVL_ADDR,            // Avalon-MM Address
    input  logic [31:0] AVL_WRITEDATA,  // Avalon-MM Write Data
    output logic [31:0] AVL_READDATA,   // Avalon-MM Read Data

    // Exported Conduit
    output logic [31:0] EXPORT_DATA     // Exported Conduit Signal to LEDs
);
```

The hardware and software interface are implemented in the Avalon_aes_interface.sv file. The interface is used to connect the software data to the hardware data. There are 16 32-bit registers which contains all the necessary information including: four registers for key, four registers for encrypted message, four registers for decrypted message, one register as the enable bit, and one register as the finish bit. The register file can be accessed by the AVL_ADDR. The RESET signal will clear the all the contents of the register file. When AVL_WRITE and AVL_CS are high, the AVL_WRITEDATA will be written to the register file 0 to 3. The specific register file will be indicated by the AVL_BYTE_EN bites. All of the input signals will be sent to the AES.sv file to do the actual AES decryption operations.

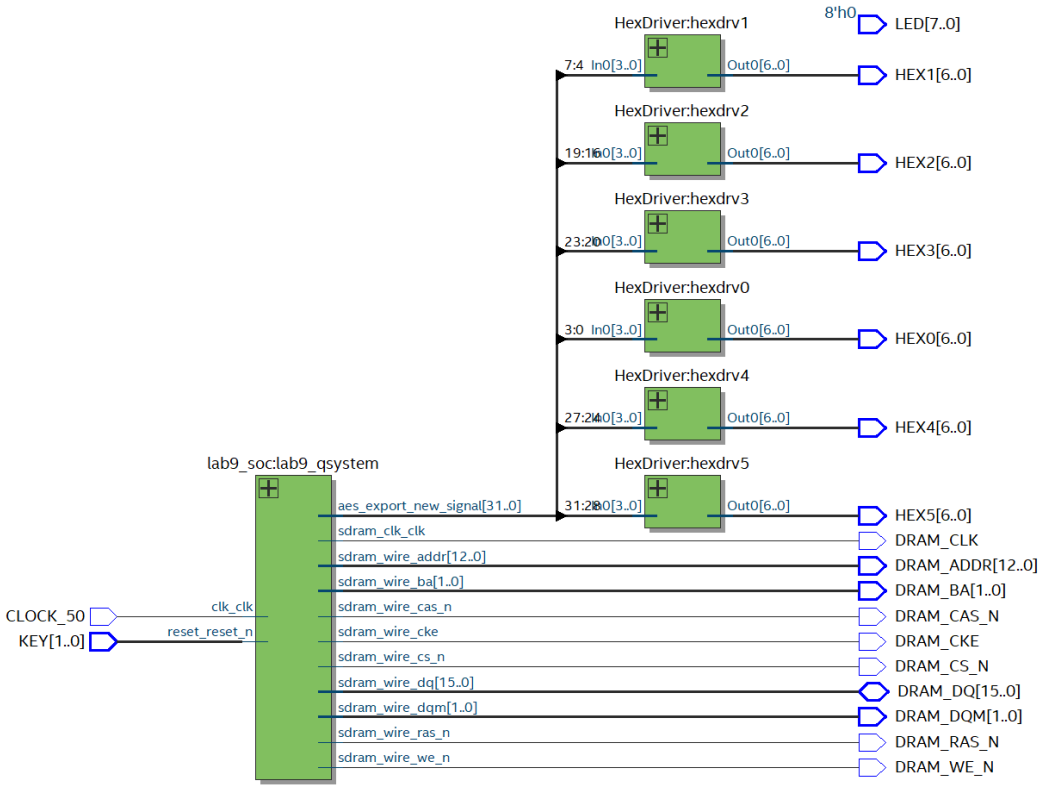


Block Diagrams:

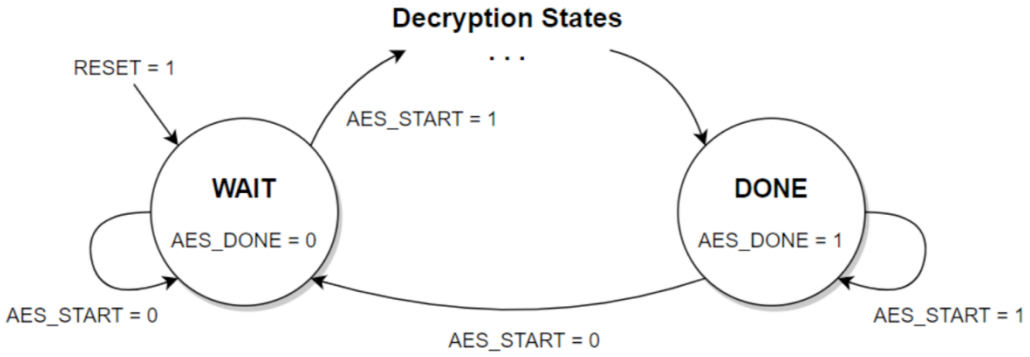
RTL view of avalon_aes_interface.sv

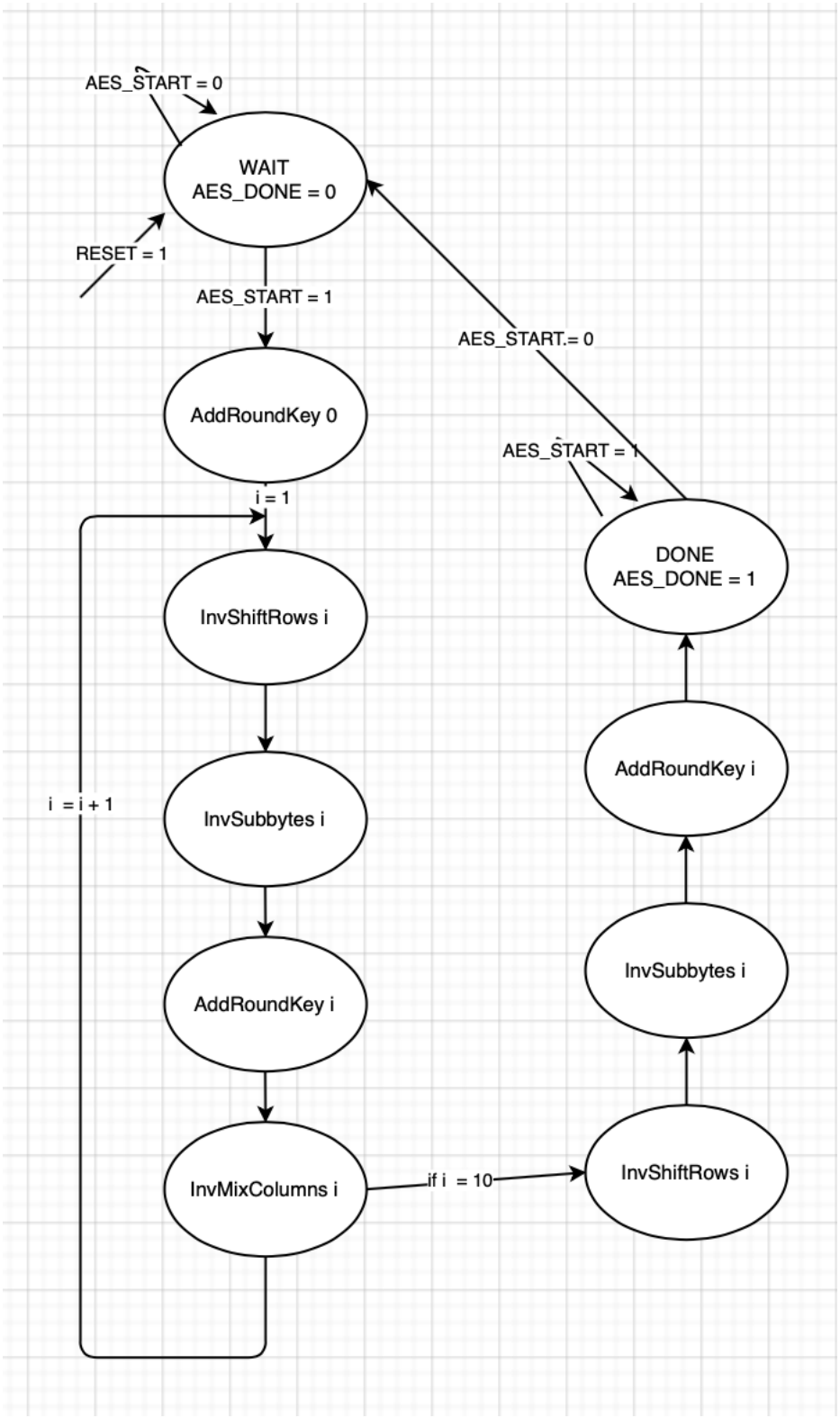


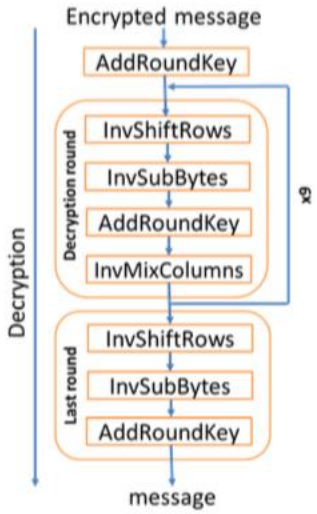
top-level RTL diagram

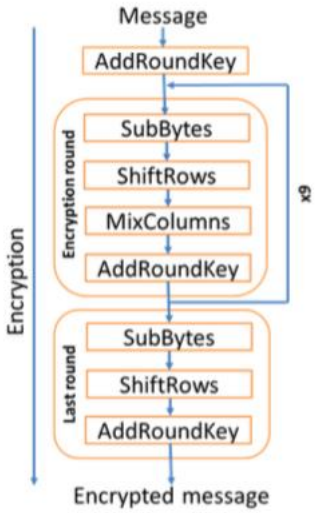


State Diagrams:




AES_START = 0
WAIT
AES_DONE = 0
RESET = 1
AES_START = 1
AES_START.= 0
AddRoundKey 0
AES_START = 1
i = 1
DONE
AES_DONE = 1
InvShiftRows i
AddRoundKey i
i = i + 1
InvSubbytes i
InvSubbytes i
AddRoundKey i
InvMixColumns i
if i = 10
InvShiftRows i





Module Descriptions:

KeyExpansion in KeyExpansion.sv
Input: Clk, [127:0] CipherKey
Output: [1407:0] KeySchedule
Description: This module expands the given 128 bits key into 1408 bits KeySchedule.
Purpose: Generate the key schedule for the eleven rounds decryption process.

KeyExpansionOne in KeyExpansion.sv
Input: clk, [127:0] oldkey, [7:0] Rcon
Output: [127:0] newkey
Description: This is the helper faction for the KeyExpansion, each will generate one set of the key.
Purpose: Helper function for the KeyExpansion.

SubBytes.sv
Input: clk, [7:0] in
Output: [7:0] out
Description: This module is a lookup table that will change the output based on a lookup table.
Purpose: This module is used to substitute bytes from the state from the lookup table.

InvMixColumns in InvMixColumns.sv
Input: [31:0] in
Output: [31:0] out
Description: This module implements the inverse of mix column based on the GF_Mul.
Purpose: This module is one of the decryption operations.

InvShiftColumns.sv
Input: [127:0] in
Output: [127:0] out

Description: This module implements the inverse of shift row operation.
Purpose: This module is one of the decryption operations.

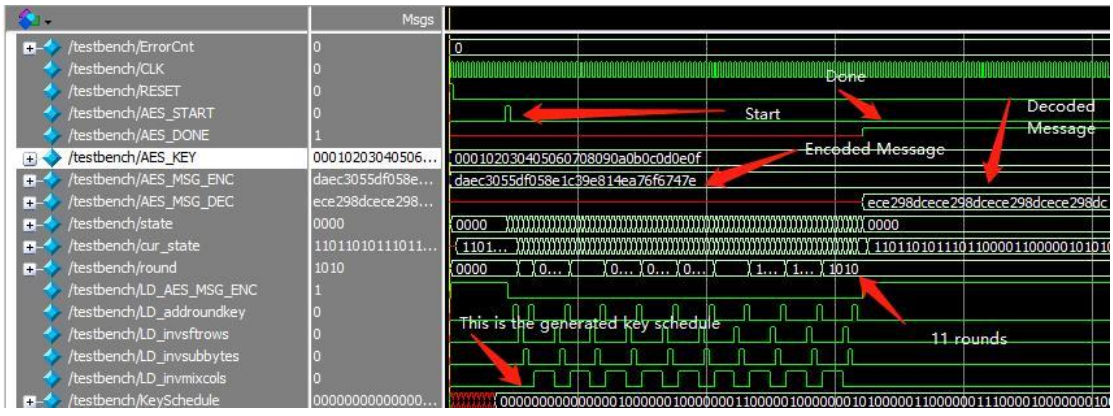
AES in AES.sv
Input: CLK, RESET, AES_START, [127:0], [127:0] AES_MSG_ENC
Output: AES_DONE, [127:0] AES_MSG_DEC
Description: the actual decryption algorithm, which instantiates all the necessary operations and the state machine.
Purpose: This module generates the decrypted message.

AddRoundKey in AES.sv
Input: [127:0] State, [1407:0] KeySchedule, [3:0] loop_num
Output: [127:0] ark_out
Description: performs the AddRoundKey operation. It will output the XORed ouput based on the loop_num to select which round of KeySchedule to perform operation.
Purpose: This module is one of the decryption operations.

InvSubAll in AES.sv
Input: CLK, [127:0] State
Output: [127:0] OUT
Description: do the InvSubBytes on all of 128 bits of the state. This module will call the InvSubBytes 16 times.
Purpose: This module is one of the decryption operations.

decript_states in AES.sv
Input: CLK, RESET, AES_START, [1407:0] KeySchedule
Output: [3:0] cur_state, [3:0] loop_num, [1:0] col_num, AES_DONE
Description: the state machine that controls the state of the next operation. This is the implementation of the state diagram above.
Purpose: control the flow of the decryption process.

Simulation of the AES Descriptor:



Post-Lab Questions:

LUT	5939
DSP	0
Memory (BRAM)	126080 bits
Flip-Flop	2771
Frequency	144.43MHz
Static Power	96.18mW
Dynamic Power	0.65mW
Total Power	105.89mW

Q: Which would you expect to be faster to complete encryption/decryption, the software or hardware? Is this what your results show?

A: We expect the hardware decryption process should be faster than the software encryption process, because the decryption is doing operations directly on the hardware, but the software would need a lot of additional process to transit form hardware to software. When we ran the benchmark, the result showed that the hardware decryption was hundreds of times faster than the software encryption.

Q: If you wanted to speed up the hardware, what would you do? (Note: restrictions of this lab do not apply to answer this question)

A: One way to speed up the hardware performance is to decrease the number of states. For example, we only instantiated the InvMixColumns once and each InvMixColumns will only invert one column from the state. This means that we would need to run four independent states just to finish four columns. In addition, when we do these 10 times for the decryption process, it takes 40 states for this operation. If we can change the current code and do the InvMixColumns for all four columns, we can finish this operation in 10 states total for the decryption process, which should definitely improve the performance.

Conclusions:

In this lab, we implemented the encryption and decryption AES algorithm. The encryption algorithm was written in C and ran on the NIOS II. The decryption algorithm is written in System Verilog and ran on the hardware directly. Since the encryption was written in C, so it was much easier to implement in the high-level language. But, the tradeoff of easy implementation is the performance time. On the other hand, the decryption is written in System Verilog. It took us much more time to implement and we wrote more than ten times lines of code compare to the C code. However, the payback is that decryption ran hundreds times faster than the encryption.