ECE 385

2021 FA

LAB 9

SOC with Advanced Encryption Standard in SystemVerilog

Zhou Qishen, Xie Mu

Introduction

The **AES** (**Advanced Encryption Standard**) is the most common symmetric encryption algorithm. In the Lab 9, we realize an AES-128 encryption/decryption on the NIOS 2 soc with FPGA board. The encryption part is done in software while the description part is done in the hardware. We also write benchmarks of the whole progress, and we can find that the hardware description is much faster than the software encryption.

Description & Diagram

Summarize the operation of the AES encryption and decryption

The AES is a symmetry encryption algorithm, as a result, both the sender and the receiver should know the Cipher key. In order to use the AES, you should prepare a Cipher key and the Plaintext, that is the message to be encrypted, and load it to the AES encryption, and then we get the encrypted message called Ciphertext. The receiver who has the Cipher key and the Cyphertext can goes through the reverse algorithm to produce the original meaningful Plaintext message.

Description and the operation of the AES encryptor/decryptor.

a) Software encryption

The message is encrypted by a C program running on the NIOS 2. With the JTAG UART module, we can get a scanf function, so we can interact to it with the console to send Plaintext and the Cipher key that we prefer. Once you send a valid message and a key, the main function will call the function CharToHex to convert the strings to the hex values, Then the key expansion function will expand the key to 10 round keys, and then we add the first Round Key with XOR to the message. Then we will loop the following steps for 9 times to encrypt the message totally: execute the SubBytes, ShiftRows, MixColumns and then addRoundKey. The SubBytes function will substitute the value based on the look up table with High four bits and the low 4 bits of the original value. The ShiftRows function will left shift the values in each row based on the number of the row. The MixColumns function will multiply each column with a fix matrix in the GF(2^8) to get the fixed value. After the loop, we call SubBytes, ShiftRows, and addRoundKey again to output the final Ciphertext.

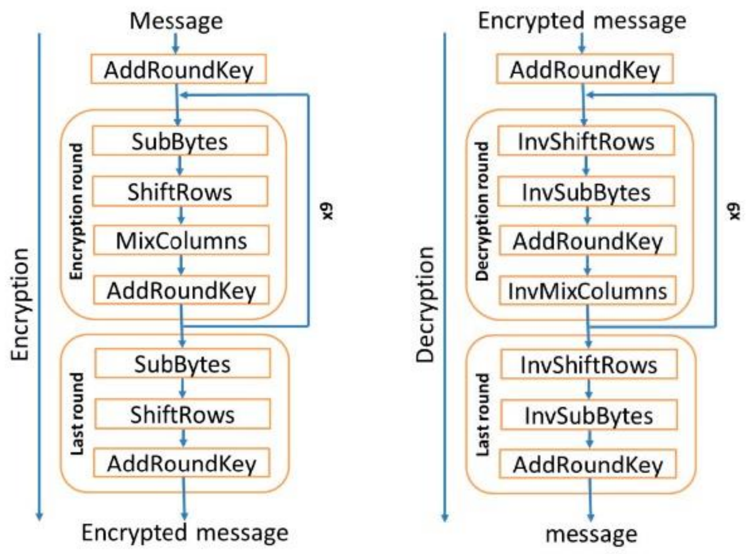
b) Hardware description

The Ciphertext will be decrypted by hardware logics written by System Verilog on the

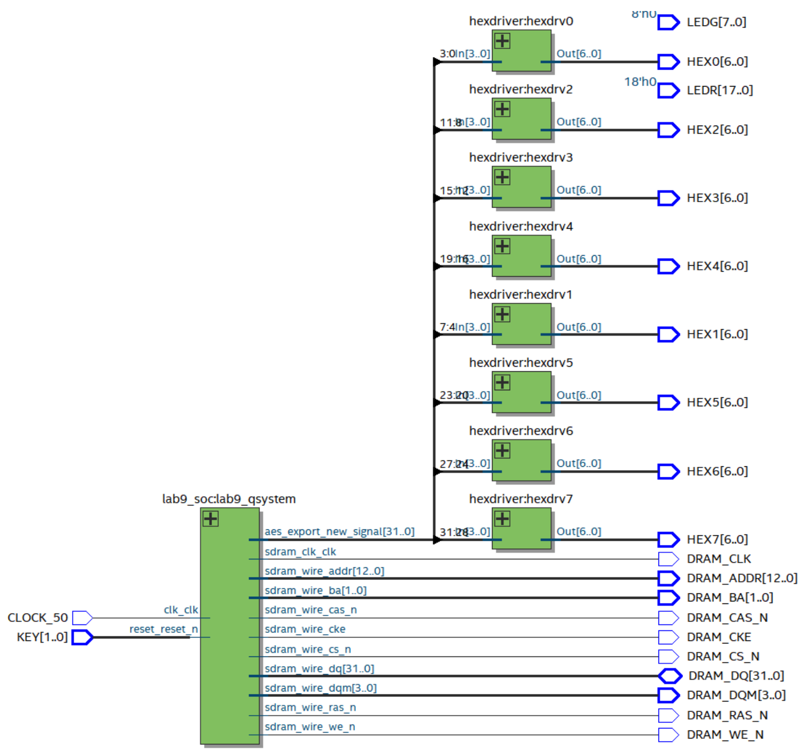
FPGA. When we type in the CipherKey and the Ciphertext in the console, the software will write them to the register file that defined in the AES decryption module. Then we initiate the AES module and all the linked module and then define states to decryption step by step. According to the diagram, we should start with the key expansion module to expand 9 round keys. Then we run addRoundKey module to do XOR to the keys. After that we will loop the invSubBytes, InvShiftRows, AddRoundKeys, and InvMixColumns 9 times to completely reverse the encryption. Then We will do InvShiftRows, InvSubBytes and AddRoundKeys again to get the Plaintext again.

c) Description of the hardware/software interface

We connect the hardware and software with the module Avalon_aes_interface.sv. We create a 32-bit reg file with 16 slot to hold the intermediates. After the software finish the encryption. The key will be stored in the first 4 registers and the encrypted message will be stored in the next 4 registers. After the hardware finish the description, the decrypted message will be store in the next 4 registered. Then we will use the 12 and 13 slot to hold signals that sent through registers, and register 14 will be used to hold START signal that given by the C program. To control the registers, the NIOS 2 will write pointers to sent messages to the target slot.

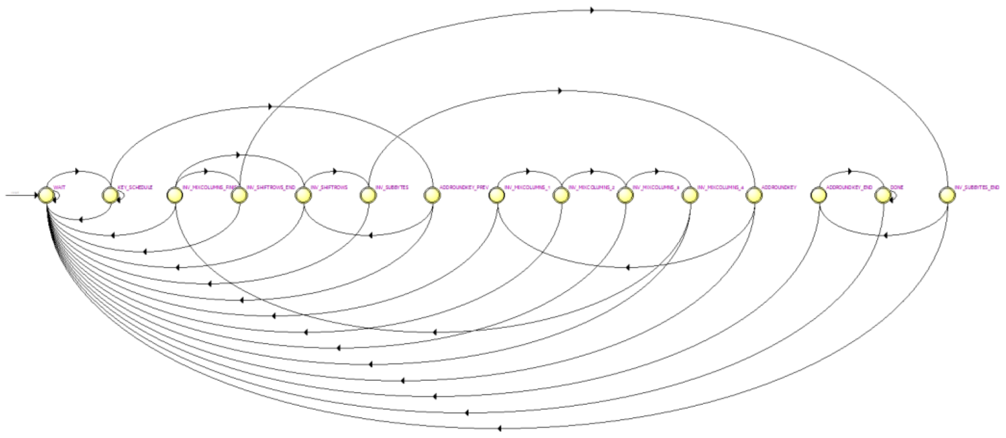


System Level Block Diagram



Block Diagram

State Diagram of AES decrypted controller



State Diagram

Hardware Module Description

Module: sdram

Input: clk, reset, s1

Output: wire

Description: This module is a 32-bit dynamic memory of SOC, which contains 1 chip, 4 banks, 13 rows and 10 columns, with the size of 1GB data.

Purpose: It is used for store data and address.

Module: onchip_memory2_0
Input: clk, reset, s1, wire
Output: readdata
Description: This module is writable RAM with memory size of 16 byte and datawidth of 32.
Purpose: This module is the on chip memory of NOIS II processor.

Module: led
Input: clk, reset, s1
Output: external_connection
Description: The module of LED on DE2 board with 8-bit datawidth.
Purpose: This module is used for control the LED on board to output data.

Module: sdram_pll
Input: inclk_interface, inclk_interface_reset, pll_slave, c0
Output: c1
Description: Clock module with the offset of -3 ns.
Purpose: This module is used for adjusting the clock input and I/O of sdram.

Module: sysid_qsys_0
Input: clk, reset, control_slave
Output: almost_empty, almost_full, empty, full, rd_data
Description: This module is system ID checker to ensure the compatibility between hardware and software.
Purpose: It can prevents us from loading software onto an FPGA which
has an incompatible NIOS II configuration.

Module: nios2_gen2_0
Input: clk, reset, data_master, instruction_master, irq, debug_deset_request, debug_mem_slave
Output: custrom_instruction_master
Description: The build-in NIOS II/e processor.
Purpose: The main processor of the SOC system.

Module: jtag_uart_0
Input: clk, reset, avalon_jtag_slave
Output: irq
Description: It is a build-in JTAG UART peripheral module.
Purpose: This module enable use the terminal of the host computer to communicate with the NIOS II

Module: timer

Input: clk, reset, s1
Output: irq
Description: Nois II built-in timer that can track the program operation time.
Purpose: This module is used to record the hardware decrypting time.

Module: AES
Input: CLK, RESET, AES_START, AES_KEY AES_MSG_ENC,
Output: AES_DONE, AES_MSG_DEC
Description: AES algorism that be implemented in hardware.
Purpose: This module is used to carry out the main job in hardware decrypting message.

Module: AddRoundKey
Input: state, roundKey
Output: out
Description: It is one of submodule of AES.
Purpose: This module is used to apply a Round Key of 4-Word matrix to the updating State through a simple XOR operation in every round.

Module: avalon_aes_interface
Input: CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, AVL_BYTE_EN, AVL_WRITEDATA
Output: AVL_READDATA, EXPORT_DATA
Description: This is the interface between Avalon and AES module.
Purpose: This module enable the communication between NOIS II and AES module.

Module: hexdriver
Input: In
Output: Out
Description: It is used for storing the value in hex display.
Purpose: This module is used to display the highest digest and lowest digits in message.

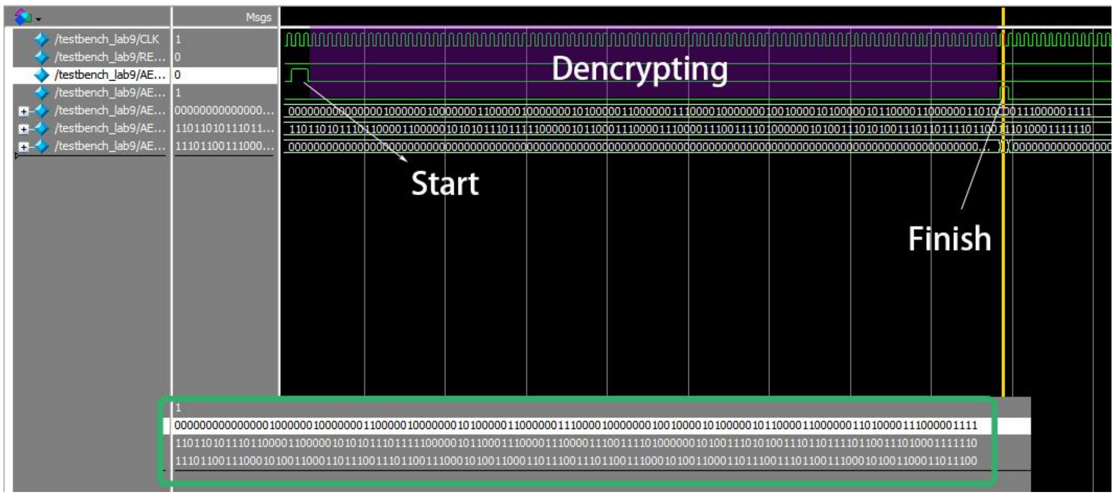
Module: InvMixColumns
Input: in
Output: out
Description: It is one of submodule of AES.
Purpose: This module is used to perform the inverse operation of MixColumns.

Module: InvShiftRows
Input: data_in
Output: data_out
Description: It is one of submodule of AES.
Purpose: This module is designed for carrying out the inverse operation of ShiftRows in encrypting.

Module: KeyExpansion
Input: clk, Cipherkey
Output: KeySchedule
Description: It is one of submodule of AES.
Purpose: This module is used to takes the Cipher Key and performs a Key Expansion to generate a series of Round Keys and store them into Key Schedule.

Module: SubBytes
Input: clk, in
Output: out
Description: It is one of submodule of AES.
Purpose: Each Byte of the updating State is non-linearly transformed by taking the multiplicative inverse in Rijndael's finite field then applying an affine transformation. This module provides a look-up table for the transformation.

Simulation



```
                                Msgs
/testbench_lab9/CLK             1
/testbench_lab9/RE...           0
/testbench_lab9/AE...           0
/testbench_lab9/AE...           1
/testbench_lab9/AE...           000000000000000...
/testbench_lab9/AE...           11011010111011...
/testbench_lab9/AE...           11101100111000...
```

```
1
00000000000000010000001000000011000000100000001010000011000000111000010000000100100001010000010110000110000001101000011100000011 11
11011010111011000011000001010101110111110000010110001110000111000011100111101000000101001110101001110110111101100111010001111110
11101100111000101001100011011100111011001110001010011000110111001110110011100010100110001101110011101100111000101001100011011100
```

Post-Lab

Document the Design Resources and Statistics in following table.

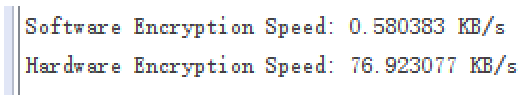
Property	Value
LUT	7175
DSP	0
Memory	125952
Flip-Flop	2806

Static Power	102.53 mW
Dynamic Power	72.09 mW
Total Power	242.72 mW

Answer to two hidden questions

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

We think that the hardware will operate a faster speed, since the algorism is implemented much directly, without much delay at interfaces or ports.



```
Software Encryption Speed: 0.580383 KB/s
Hardware Encryption Speed: 76.923077 KB/s
```

And our test result backs up our idea that hardware encryption is much more efficient.

Q: If you wanted to speed up the hardware, what would you do? (Not

We can speed up by modifying the InvMixMatrix part. In the lab, we use 4 states to calculate the inverse mix matrix column by column. As a result, if we can reduce the calculation state, we can save much clock edge to decrypt the message.

Conclusion

The project works correctly on both the weeks, but it was hard to design the decryption states for hardware decryption since there are a lot of modules to design. This lab help me understand the hardware can process data much faster than the software when the date procession can be design in logics. I think we can do some different encryption next time like RSA.