ECE 385

Fall 2019

Experiment #9

SOC with Advanced Encryption Standard in SystemVerilog

Eric Dong, Yifu Guo
ericd3, yifuguo3
Section AB3, Thursday 2pm
Gene Shiue

Introduction

In this lab, we implemented an AES encryption and decryption module. The encryption part of the lab is done in software whereas the decryption part of the lab is done in hardware. Through this lab, we are able to see the performance increase when the process is done in hardware, thus the term "hardware acceleration".

Written Description and Diagrams of the AES encryptor/decryptor

Software Encryption:

The software portion of the lab first starts with the console prompting for a message that we would like to encrypt as well as a key that we prefer. Then the main function calls the encrypting function which first converts the two strings into hexadecimal values using the two functions given. After that, the program calls the keyexpansion function which uses the cipher key to generate all 9 other round keys. Following that, we first call the addRoundKey function which does a bitwise XOR between the message and the roundKey. Following that, we go into a loop which runs 9 times, each time executing SubBytes, ShiftRows, MixColumns and addRoundKey functions. For the Subbytes function, we use a lookup table to find the multiplicative inverse of each byte in the message. Then for the ShiftRows function, we do a circular left shift on all the rows with row 0 being shifted 0 times, and row 3 shifting 3 times. Finally for the MixColumns function, we once again use a lookup table, but this time, we multiply each column by a set matrix in the GF(2^8) to get the new column. After this loop ends, we finally call the SubBytes, ShiftRows, and addRoundKey once again to get the final result. As for the NIOS II, all the C program is executed on it, but we have to remember to add a JTAG UART module in order for the scanf function to work from our computer console. The processor also controls what messages to send to the hardware decryption portion of the lab. It communicates to the avalon bus and sends the encrypted message to the bus and receives the decrypted message.

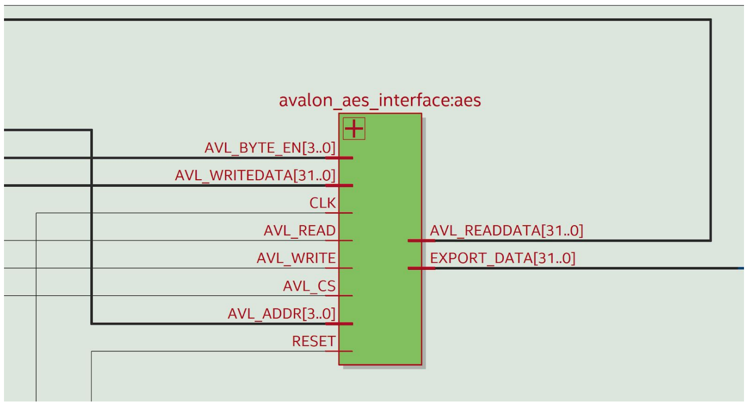
Hardware Decryption:

The hardware decryption part of this lab was done in order to show the hardware acceleration of the decryption process. We were given InvMixColumns, InvShiftRows, KeyExpansion as well as InvSubBytes modules. These modules are able to compute the inverse results of all the encryption functions in order to achieve decryption. In our AES module, we instantiated all these modules, and used a finite state machine to compute do the decryption. For our finite state machines, we used a counter in order to keep track of which round we are in already. We first start with the KeyExpansion module. After the round keys are generated, we start with the 9th round key and XOR it with the encrypted message. Then, we go into a loop where we perform InvShiftRows, InvSubBytes, AddRoundKey and InvMixColumns 9 times. Since we declared all these modules, we have to use a mux to select which output to take into our temporary decrypted message. It is important to note that since we are only allowed to declare the InvMixColumns module once, we had to have 4 states dedicated to the InvMixColumns since each state does 1 column of the matrix. Finally after the loop, we perform InvShiftRows, InvSubBytes and AddRoundKey one last time in order to get the final decrypted message.

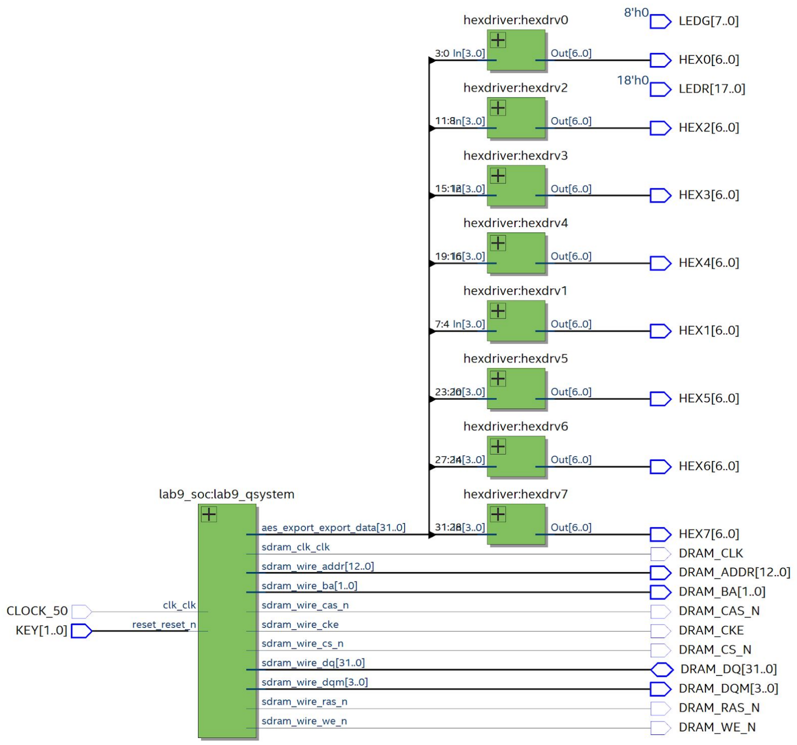
Hardware/Software Interface:

The connection between hardware and software is built upon the avalon_aes_interface module. In this module, we created 16 32bit registers. There is a register map that we can refer to. After the NIOS II is done with encryption, it will store the key into the first 4 registers, and it will also store the encrypted message into registers 4-7. After the decryption module is done with the process, the decrypted message in stored into registers 8-11. Registers 14 is used by the C program to signal hardware decryption to start the process. Register 15 is used by the hardware decryption module to signal the C program that it is done decrypting the message. Any other data is sent through the unused registers, registers 12 and 13. The NIOS II sends the encrypted message by writing to a pointer and it gets sent to the avalon_aes_interface module's register files which the decryptor can then take over.

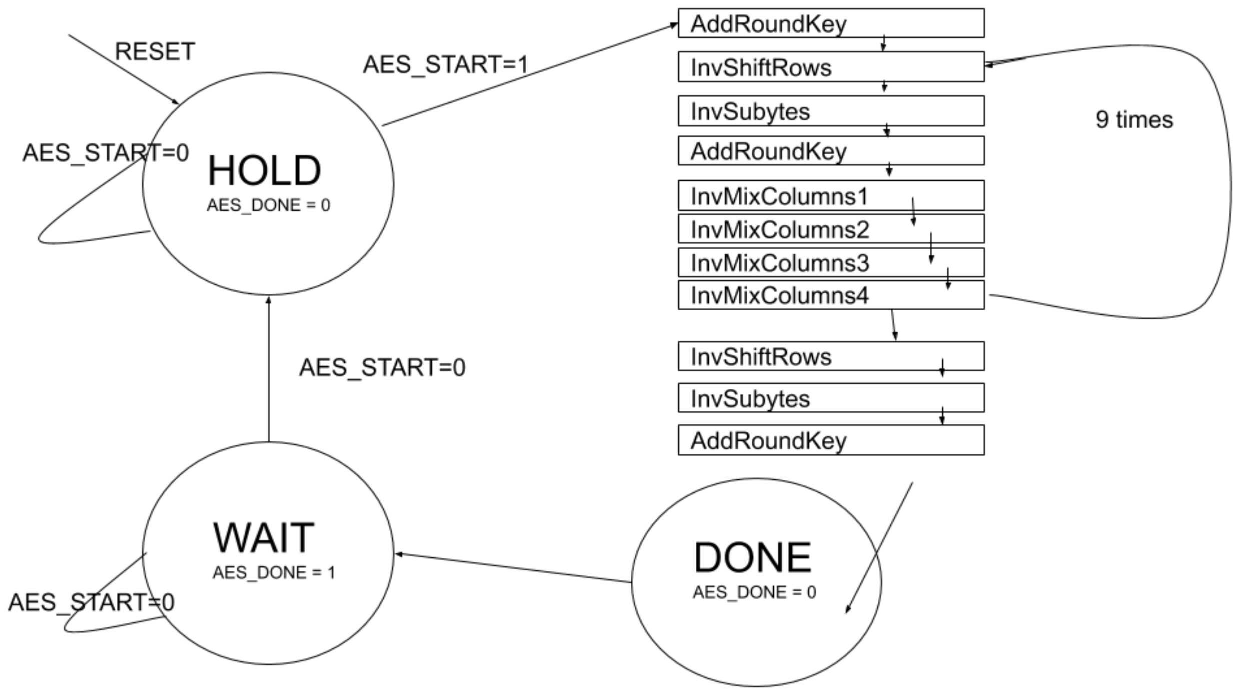
Avalon_aes_interface Diagram

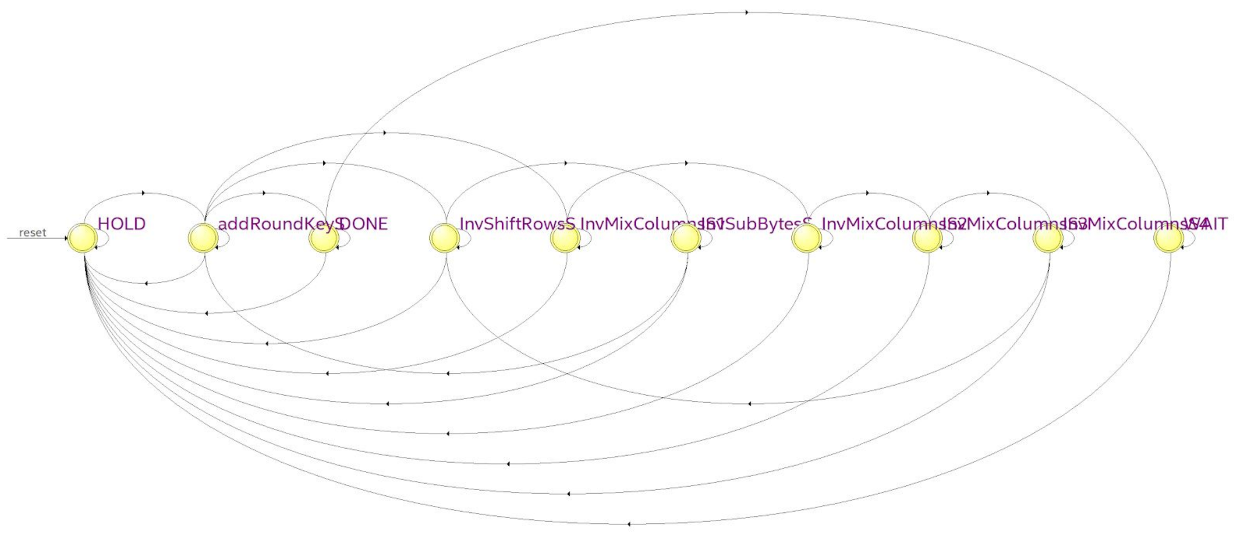


Top Level Diagram



State Diagram





Module Descriptions

- addRoundKey
    - Input
        - in, key
    - Output
        - out
    - Description
        - This module XORs the input message as well as the round key together.
    - Purpose
        - This module is used in the decryption process where we used it to subtract(XOR) out the round key from the message and output the result to the output.
- AES
    - Input
        - CLK, RESET, AES_START, [127:0] AES_KEY, [127:0] AES_MSG_ENC
    - Output
        - [127:0] AES_MSG_DEC, AES_DONE
    - Description
        - This module oversees the entire decryption process. It has a state machine inside which controls the next temporary key.
    - Purpose
        - This module is able to control the data coming from several different decryption modules. It chooses which output from the different modules should become the next temporary message. It also sends the DONE signal once the process is done.

- Avalon_aes_interface
    - Inputs
        - CLK, RESET, AVL_READ, AVL_WRITE, AVL_CS, AVL_BYTE_EN, AVL_ADDR, AVL_WRITEDATA
    - Output
        - AVL_READDATA, EXPORT_DATA
    - Description

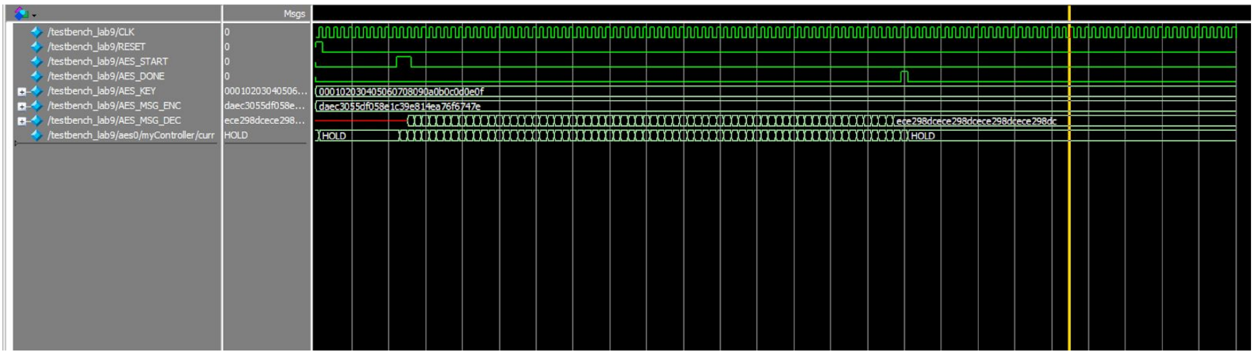
        This module creates a register file of 16 registers with each one having a different purpose. It is able to clear the registers upon pressing the reset button, as well as writing data to the registers based on the AVL_BYTE_EN signal. Finally it is able to send the encrypted message to the AES module to do the decryption, as well as store the final message to the registers for the software to access.
    - Purpose
        - This module is a linker between the software and hardware process of this lab. It controls the data flow from the registers that it creates to the hardware/software parts of this lab. It also controls what data to export to display on the hex drivers.
- HexDriver
    - Input
        - In0
    - Output
        - Out0
    - Description
        - This displays the input to the module to the hexdisplays on the DE2 board.
    - Purpose
        - This module displays the keyboard input ASCII character onto the hex display so we know what we are pressing and for debugging.
- InvMixColumns
    - Input
        - [31:0] In
    - Output
        - [31:0] out
    - Description
        - This module is provided and helps us with the decryption algorithm. It takes 32 bits of input and outputs 32 bits of output with the modified data.
    - Purpose
        - This module does inverse multiplication with 1 column of decrypted message at a time.
- InvShiftRows
    - Input
        - [127:0] data_in
    - Output
        - [127:0] data_out
    - Description

        - This module is provided and helps us with the rows-shift algorithm. It takes 128 bits of input and outputs 128 bits of output with the modified data.
    - Purpose
        - This module does inverse shift rows operations with the input data.
- KeyExpansion
    - Input
        - Clk, [127:0] Cipherkey
    - Output
        - [1407:0] KeySchedule
    - Description
        - This module generated the round key used in the decryption algorithm.
    - Purpose
        - The round key generated is used in the addRoundKey module where each round key is XORed with the temporary message.
- lab9_top
    - Input
        - CLOCK_50, [1:0] KEY
    - Output
        - LEDG, LEDR, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7, DRAM_ADDR, DRAM_BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_DQ, DRAM_DQM, DRAM_RAS_N, DRAM_WE_N, DRAM_CLK
    - Description
        - This module is the top-level of this lab. It connects all the corresponding wires of hexdrivers and sdram.
    - Purpose
        - This module helps to show the data on the hex displays and instantiate the Qsys design to access and operate memory approperly.
- SubBytes
    - Input
        - Clk, [7:0] in
    - Output
        - [7:0] out
    - Description
        - This module is provided and helps to transform the input bytes. It takes 8 bits of input and output 8 bits of output with modified data.
    - Purpose
        - This module helps us to perform the transform in the Rijndael’s finite field.

- Lab9_soc
    - Input
        - Clk_clk, reset_reset_n
    - Output
        - Aes_export_export_data, Sdram_clk_clk, Sdram_wire_addr, Sdram_wire_ba, Sdram_wire_cas_n, Sdram_wire_cke, Sdram_wire_cs_n, Sdram_wire_dq, Sdram_wire_dqm, Sdram_wire_ras_n, sdram_wire_we_n
    - Description
        - This is the system on chip module that holds the NIOS II chip as well as all the other components that communicate with the avalon bus.
    - Purpose
        - With the avalon bus, we are able to add a new module called the avalon_aes_interface module which communicates with the hardware decryption module. This allows the software to write to the register file that is on the hardware, and read the decrypted message. The NIOS II also handles the software encryption part of the lab.

Annotated Simulation of the AES decryptor

- The input message is: ece298dcece298dcece298dcece298dc
- The current state is in curr
- Encrypted message is in AES_MSG_ENC
- Finished message is in AES_MSG_DEC
- The system signals the DONE in AES_DONE


Msgs
/testbench_lab9/CLK 0
/testbench_lab9/RESET 0
/testbench_lab9/AES_START 0
/testbench_lab9/AES_DONE 0
/testbench_lab9/AES_KEY 00010203040506...
/testbench_lab9/AES_MSG_ENC daec3055df058e...
/testbench_lab9/AES_MSG_DEC ece298dcece298...
/testbench_lab9/aes0/myController/curr HOLD
000102030405060708090a0b0c0d0e0f
daec3055df058e1c39e814ea76f6747e
ece298dcece298dcece298dcece298dc
HOLD
HOLD

Document the Design Resources and Statistics from the lab manual.

LUT	6,584
DSP	0
Memory	126,080
Flip-Flop	3002

Frequency	124.04 MHz
Static Power	102.26 mW
Dynamic Power	0.75 mW
Total Power	174.04 mW

Post Lab Questions:

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

I expected the decryption process run on the hardware to be running faster. This is what our results show. The hardware speed was over 100KB/s while the software process was slower than 1 KB/s.

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

One main thing that could have been sped up was the inverse mix matrix. Since in this lab, we were only allowed to declare 1 instance of this module, we had to create 4 different states in order to complete one invmixmatrix operations. If we were able to declare 4 at a time, then we could have done the operation in 1 state in the state machine instead of 4.

Conclusion

a. Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it

Our software portion of the design worked perfectly, but out hardware of the design had flaws. Since our simulation was correct, we thought we were done, but this is not the case. Our result in the console was wrong by 16 bytes. I believe this was a timing issue. To fix this, we should have taken into account for the keyexpansion to be done as well as to make sure that we have a stable answer before we send out the DONE signal.

b. Was there anything ambiguous, incorrect, or unnecessarily difficult in the lab manual or given materials which can be improved for next semester? You can also specify what we did right so it doesn't get changed.

The lab manual did a good job this time. It would be helpful if we added the state machine diagram into the manual. I think noting the timing issue is really important so nobody else can fall into that pitfall.