ECE 385

EXPERIMENT #9

SOC with Advanced Encryption Standard in SystemVerilog

I. OBJECTIVE

In this experiment you will implement a 128-bit Advanced Encryption Standard (AES) using SystemVerilog as an Intellectual Property (IP) core. In the first part of this lab you will implement 128-bit AES encryption on the software IP core, in the second part of this lab you will implement 128-bit AES decryption in SystemVerilog.

II. INTRODUCTION

An Intellectual Property (IP) core is a reusable hardware design unit that is the intellectual property of a party. It can come in many different forms, ranging from hardware description language (HDL) to transistor layouts. An IP core in the hardware world is analogous to a library in the software language. The IP cores are often designed to gear towards specific functionalities rather than complete hardware systems. An IP core is usually specified through its inputs and outputs, with emphasis on the various design specs and performance evaluations such as the ones we have introduced in Experiment 4.

Users of an IP core usually obtain their license from the owner of the IP core to incorporate the IP core into their own design for extended functionality. As a notable example, manufacturers of ARM-based processors obtain their license from ARM Holdings to design their own processors. The owner of the ARM IP core does not manufacture its own processors, but instead focuses on the development of the ARM architecture itself.

In this experiment, you will learn how to design a decryption IP core, and incorporate it with your software interface through the embedded NIOS II processor to form a complete system on chip (SoC). Through the software interface, your circuit will be able to decrypt the input data using the provided cipher key to reconstruct the original message.

Please read the INTRODUCTION TO ADVANCED ENCRYPTION STANDARD (IAES).

Your AES decryption core should have the following inputs and outputs:

Inputs

msg_en – **logic** [127:0]: Cipher text containing the encrypted message
key – **logic** [127:0]: Cipher key used for the encryption and the decryption of the message
run – **logic**: Control signal to start the AES decryption
reset_n – **logic**: Control signal to reset the AES decryption circuit

Outputs

msg_de – **logic** [127:0]: The original message reconstructed from msg_en and key
ready – **logic**: Flag which indicates that the AES decryption has completed, and that the output data msg_de is ready to be accessed

III. PRE-LAB

A. **Week 1**:

Your primary task for the first week of this lab is to implement a 128-bit AES algorithm in C and then to run that algorithm on the NIOS II-e processor. The lookup tables will be provided to you on the course website, while the bulk of the algorithm will be implemented by you.

The individual parts you will need to implement are explained in detail in **Appendix IAES.XX** located in the back of your lab manual.

The sections for encryption include:

1. Your basic encryption input and control loop
2. A KeyExpansion function
3. An AddRoundKey function
4. A SubBytes function
5. A ShiftRows function
6. A MixColumns function

Psuedo-code is provided in the appendix for the progression through these various states, so ideally the process should be as simple as writing each component, and then writing a control loop that applies each step of the algorithm to the plaintext (called the state in the appendix) in the proper sequence.

NOTE: For debugging purposes, the course website contains a table of intermediate results. This file should allow you to figure out what sections of your AES code are having issues.

Once this AES algorithm is working (I would suggest having 2-3 different values for the plaintext and cipherkey that you can swap between to test it), your next task is to get the algorithm to work on the NIOS II-e. The main barrier here is getting the JTAG UART set up to handle input and output properly. You should already have this mostly implemented due to Lab 8, but if you haven't, the same set of steps applies.

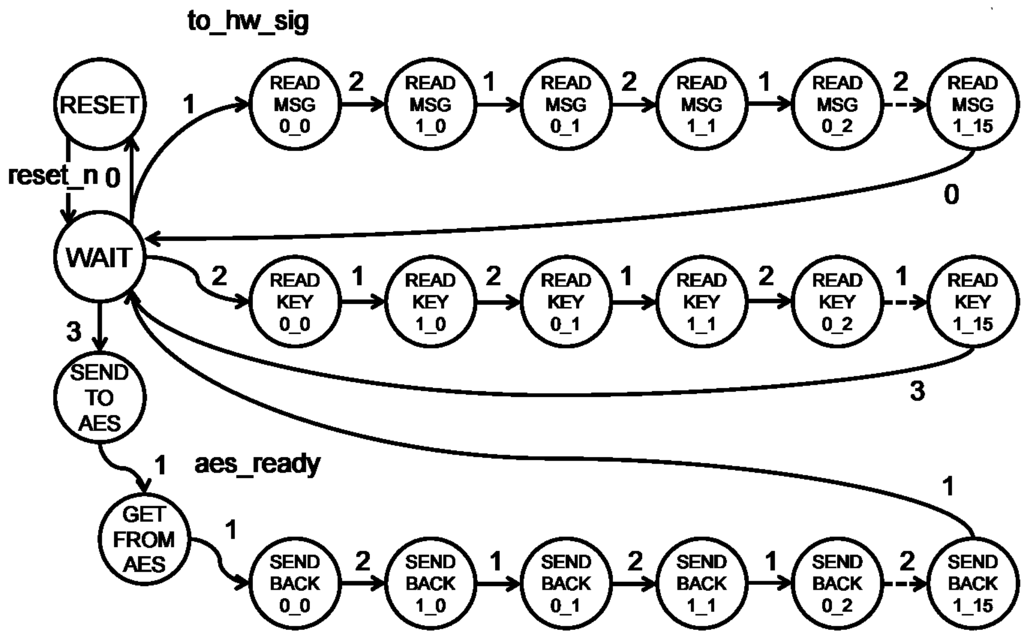
For week 1, your code should have the abilities to receive plaintext and a cipher key from the Eclipse console. It should then use your C code to encrypt the message, with the results appearing in the console and the key on the hex displays. The course website has some example messages and keys (with results) if you want to use them.

In order to display your key on the hex displays, you will also need to complete hardware/software communication in week 1. This should be based on the state machine shown below, it is explained in detail in the lecture slides. You need to be able to send a 128 bit key to the hardware and display the first 8 and last 8 bits on the hex display.

For week 1, you will be required to benchmark your C code on the NIOS II/e software core. It is sufficient to perform many iterations of your AES encryption algorithm and use a hand timer to estimate the throughput. For your benchmark assume that the same key is used for all iterations.

Week 1 Demo Point Breakdown:

1 point: Correct key expansion
1 point: Correct initial round key and looping rounds operations
1 point: Correct AES encryption test vectors
1 point: Benchmark of NIOS II/e based AES in MB/s
1 point: Correct Hardware Software communication to display the correct key on the hex displays



B. **Week 2:**

For the second week, you will need to design, document, and implement a 128-bit hardware decryption core in SystemVerilog that uses the AES algorithm. You will then use Qsys to link the AES IP core to your project. A state machine is **REQUIRED** to transition through the different AES modules, as well as to incorporate the capabilities to initialize, run, reset, and flag the completion of the AES decryption algorithm.

You are provided with IO_module, AES_controller, AES, KeyExpansion, InvMixColumns and SubBytes. In the previous week you should have the completed the io_module to communicate between hardware and software, and should also have a working Qsys environment.

To complete the decryption algorithm you should need to modify AES.sv. You need to make a state machine to handle the looping of the AES algorithm. Since you are only allowed to include one instance of InvMixColumns you need to handle creating additional states in the above state machine or add another state machine. Using the Hardware/Software interface you should be able to output the decrypted plaintext to the terminal.

For week 2 you will benchmark your HDL decryption core, separate from the encryption step. You will need to modify the decryption state machine to perform multiple iterations of the decryption algorithm followed by a message upon completion. You should perform enough iterations so that the console communication time is negligible compared to the time spent decrypting.

You will need to bring the following to the lab:

1. Your code for the Lab.
2. Block diagram of your design with components, ports, and interconnections labeled.
3. An annotated simulation of the AES decryption core showing the correct state transition of the AES state machine.

Week 2 Demo Point Breakdown:

1 point: The AES_terminal program continues after pressing the on-board reset (KEY[1]).
1 point: Correctness of a single looping round operation. (on-board)
1 point: Correctness of stepping through the looping rounds using state machine. (on-board)
1 point: Transmitting correct result to NIOS II-e and displaying plaintext via terminal
1 point: Show benchmark results for HDL based AES decryption in MB/s

IV. LAB

Follow the Lab 9 demo information on the course website.

V. POST-LAB

1.) Refer to the Design Resources and Statistics in IQT.25-27 and complete the following design statistics table.

LUT	
DSP	
Memory (BRAM)	
Flip-Flop	
Frequency	
Static Power	
Dynamic Power	

Total Power	

Document any problems you encountered and your solutions to them, and provide a short conclusion. Before you leave, submit your latest project code including both the .sv files and the software code to your TA on his/her USB drive. The TAs are under no obligation to accept late code or code files that are intermixed with other project files.

2.) Answer the following questions in the lab report

- Which would you expect to be faster to complete encryption/decryption, the software or hardware? Is this what your results show?
- If you wanted to speed up the hardware, what would you do? (Note: restrictions of this lab do not apply to answer this question)

VI. <u>REPORT</u>

In your lab report, should hand in the following:

- An introduction;
- Written description of the operation of your circuit;
- Written purpose and operation of each entity;
- State diagram for the software/hardware interface (io_module);
- State diagram for the decryption process;
- Schematic block diagram with components, ports, and interconnections labeled;
- Annotated prelab simulation waveforms showing the correct state transition of the AES decryption core;
- Answers to postlab questions;
- A conclusion regarding what worked and what didn’t, with explanations of any possible causes and the potential remedies.