# 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, and will learn to design your own hardware IP core.

## 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) and the INTRODUCTION TO THE AVALON-MM INTERFACE (IAMM).

## III. <u>PRE-LAB</u>

### 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 which contains S-Box, Inverse S-Box, gf_mul (pre-calculated values for all possible GF(2^8) calculations), and Rcon, is be provided to you in aes.h, while the bulk of the algorithm will be implemented by you.

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.

To display your key on the hex displays, you will also need to complete the Avalon-MM Interface, described in IAMM, in week 1. You need to be able to send a 128 bit key to the hardware and display the first and last 2 bytes 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 use the provided codes for benchmarking.

**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 kB/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 IP core in SystemVerilog that uses the AES algorithm. You will then use Platform Designer 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 avalon_aes_interface, AES, KeyExpansion, InvShiftRows, InvMixColumns and SubBytes. In the previous week you should have the completed the avalon_aes_interface to communicate between hardware and software and should also have a working Platform Designer 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 simulating the AES decryption part, you may set AES.sv as the top-level entity temporarily, and write a testbench to simulate AES.sv directly.

For week 2 you will benchmark your HDL decryption core, separate from the encryption step. You can also use the provided codes for this step.

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: 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: Able to run consecutive decryption operations correctly without restarting the program.
1 point: Show benchmark results for HDL based AES decryption in kB/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. REPORT

Write a report, you may follow the provided outline below, or make sure your own report outline includes at least the items enumerated below.

1. Introduction
    a. Briefly summarize the operation of the AES encryptor/decryptor.
2. Written Description and Diagrams of the AES encryptor/decryptor
    a. Written description of the software encryptor
        i. Describe the role of the NIOS processor as well as the basic functionality of your C code
    b. Written description of the hardware decryptor
        i. Describe the basic steps of decryption and how this is controlled and computed in hardware
    c. Written description of the hardware/software interface (avalon_aes_interface.sv)
        i. Describe how the system sends data between NIOS and the hardware decryptor and how the register file is designed

d. Block diagram(s)
   i. Include the RTL view of avalon_aes_interface.sv
   ii. Also include a top-level RTL diagram
e. State Diagram of AES decryptor controller
   i. This is the state machine that was written in AES.sv.
   ii. You may abbreviate the 9 looping rounds in the state diagram like in figure 9 on page IAES.9 of the lab manual.

3. Module Descriptions
   a. A guide on how to do this was shown in the Lab 5 report outline. Do not forget to describe the Qsys generated file for your Nios system! The Qsys view of the NIOS processor or lab9_top.sv is helpful here
4. Annotated Simulation of the AES decryptor
   a. Write a testbench for AES.sv, and set AES.sv as top level for simulation.
   b. In this simulation, you should display the input encrypted message, the input plaintext, the output decrypted message and the current state of the state machine.
   c. Notate various points of interest in the simulation (such as when the decryptor finishes decrypting, etc.).
5. Post-Lab Questions
   a. Fill out the design resources and statistics table
   b. Answers to the post-lab questions
6. Conclusion
   a. Discuss functionality of your design. If parts of your design didn't work, discuss what could be done to fix it
   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.