Lab 7: Advanced Encryption Standard

Peter Johnson[1]

E155 Fall 2018 Section 2, November 1, 2018

***Abstract*— This paper describes the implementation of a hardware accelerator on an FPGA to perform 128-bit AES encryption. The Raspberry Pi sends a plaintext message to the FPGA which returns the cypher to the Pi to be verified.**

I. Introduction

In this lab, the MuddPi Mark IV utility board was utilized in conjunction with the Raspberry Pi. The bulk of the work was done on the FPGA implementing an accelerator for the AES encryption algorithm. Simulation was done in ModelSim, and the SPI communication was examined using a logic analyzer. The encryption was checked using both a testbench and by a test case on the Pi.

II. Design and Testing Methodology

I approached the design of the system by starting at the highest level of abstraction. I split the algorithm up into black boxes with inputs and outputs which I would implement as modules. Basically each function in the provided cipher psuedocode was converted into its own hardware block. This meant that things like swapRows, subBytes and the key expansion would get their own modules. I initially got confused and started to go down the way of thinking like software, but was able to stop this by drawing out a large diagram of the system and the different pieces of hardware I would need. In it, I tried to make a path for my state variable to propagate through each round up to a flip flop which would then pass the state back to the beginning of this path for the next round. In this way, I was able to implement a synchronous sequential version of the algorithm to fit on the FPGA. I also decided that instead of doing the entire key expansion before the encryption took place, I would do it at the same time, only carrying 4 words at a time. In this case, the previous 4 words are used to calculate the next 4 words which is perfect.

I believe my implementation is relatively efficient. I only use each module once, using mux's to take care of the special cases at the first and last rounds of the encryption instead of making another instance of a module for the special case. The signals for the muxes are taken care of by a state machine which just has a single state for each round. The signals are just "start" for the first round in which the subBytes, shiftRows, and mixColumns are skipped, and a "last" for the final round in which the mixColumns is skipped. The FSM also provides the Rcon value to the key expansion module since it changes with each round. This allowed me to just hardcode each value without doing any math since this implementation only needs to work for a single 128-bit case. The final signal from the FSM is to set the done bit to high when the final state is reached. The way I implemented my state machine necessitated having a reset signal so that I would start at state 0 once the load signal went low. I used a modified level to pulse converter to do this..

My testing methodology was simple and based on using the ModelSim simulation waveforms and the examples of the encryption in the AES appendix. My signals were structured such that I could look at the values of my key expansion and state array after each individual transform. I could see the value of the old state, the value after subBytes, shiftRows, and so on. I would then step through the encryption process, comparing the state at each step with the correct value in the appendix. Once I verified I had the correct signal, I would move onto the next step. I did this first for the key expansions before moving onto the overall encryption.

Once the simulation testbench was passed, the last step was getting the accelerator to work on actual hardware. I assigned the pins in the planner, and then wired up the Pi to the FPGA. Testing here was a little more difficult. I tried running it on different people's boards, and with different wiring schemes. In the end I was able to fix my problem which was either swapping a pin on the FPGA or mixing up MISO and MOSI.

III. Code and Schematics

Refer to attached the code and schematics.

IV. Results and Discussions

All of the prescribed tasks were accomplished: the Raspberry Pi is able to send a plaintext message via SPI to the hardware accelerator on the FPGA which sends the ciphertext back to the Pi via SPI where it is successfully verified as the correct encryption of the original message. I was able to view the SPI communication on a logic analyzer.

V. Conclusions

The lab was successfully completed. I learned how to convert a software algorithm into a hardware accelerator to perform the same computations much more efficiently. This is visible in the fact the accelerator took only 11 clock cycles for the encryption. I gained experience with designing complex digital systems using hierarchy and abstraction. I also got practice writing SystemVerilog and debugging using simulation waveforms. In total I spent 13 hours, it was a long lab.

[1]Peter Johnson is with Department of Engineering Harvey Mudd College, Claremont, CA 91711 pjohnson@hmc.edu