Ryan Hagler

CMPE 415

Homework 6

Project Report

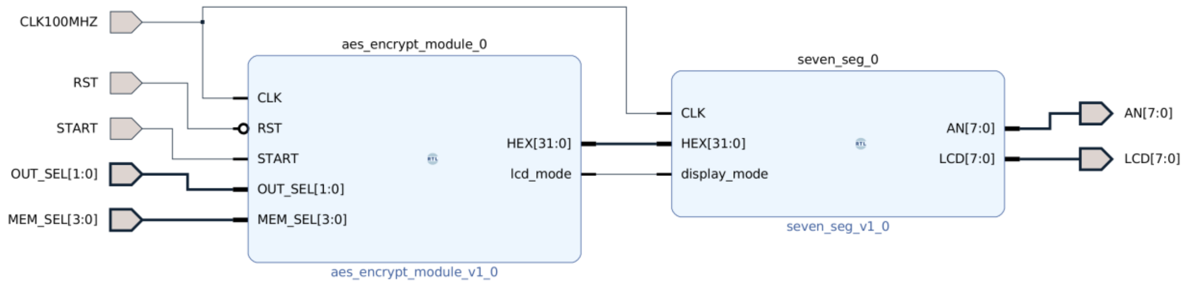
This document contains a brief summary of the design for my implementation of Homework 6. The project requires the FPGA to create a 10-round AES core and display the ciphertext of the final round to the user. In my implementation of this project, I created modules to handle each part of the encryption in a way that allowed me to easily scale the project. In addition to the three provided .v files, I created:

- *aes_encrypt_module.v*
- *key_schedule.v*
- *per_round_sbyte.v*
- *seven_seg.v*
- *sub_byte.v*

I have also renamed IO from the FPGA constraint file to make the block design easier to understand.

- RST <- SW[0]
- START <- SW[1]
- OUT_SEL <- SW[3:2]
- MEM_SELECT <- SW[15:12]
- AM[7:0] <- 7 Segment Select
- LCD[7:0] <- Display Select
-

All files are located in the /src directory within the project. The figure below is my top Block Design, which only uses *aes_encrypt_module* and *seven_seq* at the highest level.



aes_encrypt_module is the function that does (or calls) all of the AES core calculations, while *seven_seg* is nothing more than a driver for the 7 segment LEDs. Within this main module, no functions can happen when the RST switch is low. Once the RST switch is high, the START switch is able to start the AES core.

One known misbehavior in the module is that the display will show the previous AES ciphertext upon start up if the START switch is low. I have done testing with the display on at all times, and no matter what memory bank is selected at the time of programming, the 10th roundkey of mem[0] is sent to the 7 segment. It is also important to note that the AES core does not calculate new keys using the

START switch when RST is low. I believe this “ghost value” has to do with the memory selection module being initiated to 0, therefore sending the first plaintext to the core upon programming the device.

With that being said, once the first encryption pass is complete, the START switch behavior completely matches the project description, in which the switch must be pulled down and then back up in order to start a new pass. As the document does not state one way or the other, the user will still be able to flip between byte chunks using SW[3:2] while START is low (only is RST is high).

In order to complete the first round of encryption, which I made sure was working before moving on to the loop, I read through the project document carefully to plan which pieces needed to be their own modules. The S_Byte step seemed to be a good candidate, as it is a simple case statement. I created this and instantiated it inside of *aes_encrypt_module*. Key Scheduling was another step that was easy to separate into its own module, returning the g value in order to create the next roundkey. Once these were instantiated, I used the Column_Mix file provided, and followed the rest of the steps to produce the first roundkey. I also used all blocking assignments within the AES core.

After the first round was successfully produced, I created 2D array from the 1D regs that held my data, created instantiations for each round, and created a loop that would populate the 2D array for each round. I also created a nested instantiation for the s_byte module, as each byte needed its own module. This prevented there from being 16 s_byte modules for each round in the main module. Instead of creating another case for the final round, I only looped through for 9 rounds, and had separate logic for the final round to avoid unnecessary debugging. With minimal problems, I was able to correctly produce the 10th roundkey for all 16 plaintexts in memory.