CSE 140L Lab 5

NG Zhe Wee, A16389707

Academic Integrity

Your work will not be graded unless the signatures of all members of the group are present beneath the honor code.

To uphold academic integrity, students shall:

- Complete and submit academic work that is their own and that is an honest and fair representation of their knowledge and abilities at the time of submission.
- Know and follow the standards of CSE 140L and UCSD.

Please sign (type) your name(s) below the following statement:

I pledge to be fair to my classmates and instructors by completing all of my academic work with integrity. This means that I will respect the standards set by the instructor and institution, be responsible for the consequences of my choices, honestly represent my knowledge and abilities, and be a community member that others can trust to do the right thing even when no one is watching. I will always put learning before grades, and integrity before performance. I pledge to excel with integrity.

NG Zhe Wee, A16389707

Free Response

Please answer the following questions.

1. Please describe, at a high level, how each stage of decryption using LFSR works. (4 pts)

Word limit: 200 words.
The stages we are looking for you to describe are the preamble stage, training stage, and decryption stage.

The LFSR is responsible for generating a pseudo-random bit pattern, which can have a significantly long repeating pattern, effectively resembling a random sequence. To initialize and set the state of the LFSR, an initial value is required, and the resulting random pattern will be the same for a given initial value and state. This similarity between generating random patterns and encryption keys is notable. By applying XOR with a random pattern, it becomes possible to encrypt a message. Conversely, by knowing the initial value and state of the LFSR, the encrypted message can be decoded back into its original form by reversing the XOR operation.

To identify a matched decoded value, a continuous operation of six LFSRs is initiated for a duration of 7 to 12 cycles. Once one of the LFSRs successfully matches the desired value, it assumes the responsibility of decoding the remaining portions of the message. This chosen LFSR continues to process the message until its completion, ensuring a seamless decoding process.

2. Why do we use 6 LFSRs in our top level module? (4 pts)

Word limit: 200 words.

We utilize six LFSRs in our design, each starting with a unique initial value. This approach significantly reduces the time required to identify a matching value by employing multiple LFSRs simultaneously. While it is technically feasible to employ a single LFSR, doing so would necessitate a greater number of clock cycles to accomplish the same task.

3. Please explain how the testbench tests your design. (4 pts)

Word limit: 200 words.
You can briefly describe the major steps taken by the testbench.

The testbench is responsible for generating an original string before encryption. It then proceeds to calculate the encrypted messages and writes them to the data memory. Next, the testbench reads the content of the data memory, which stores the decrypted message generated by the

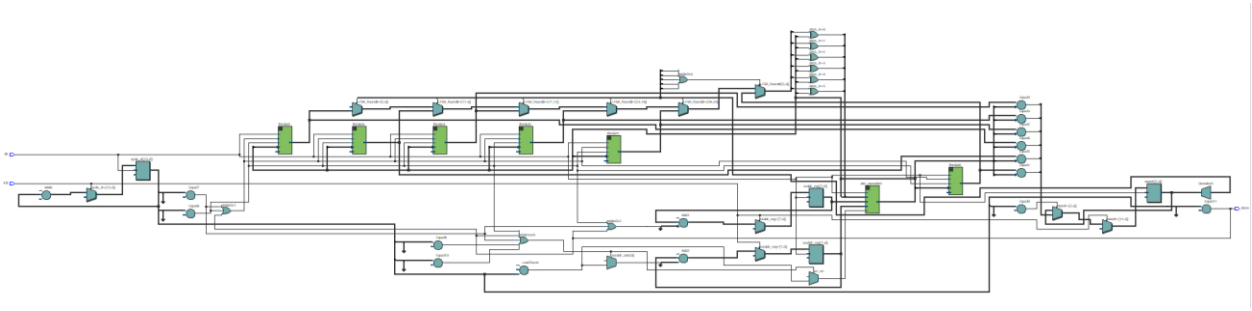
DUT (Device Under Test). Finally, the testbench compares the obtained result with the expected result to ensure the accuracy of the decryption process.

4. In our implementation, the preamble is an underscore character. Is it possible to decrypt messages with an unknown preamble character? How would you edit your decryption design to accommodate for different character preambles? (3 pts)

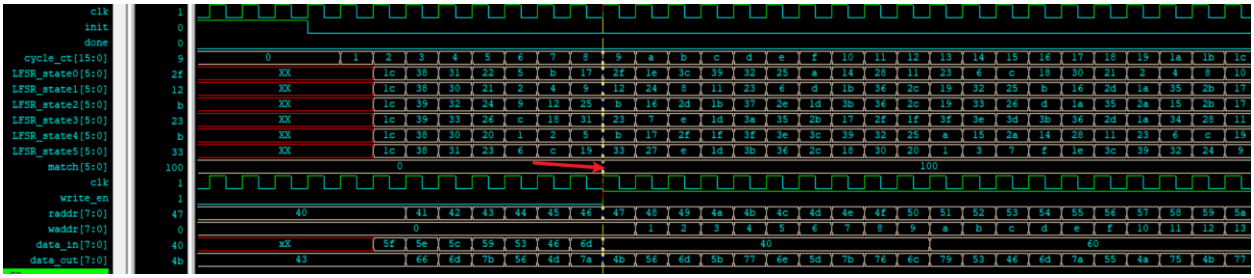
Word limit: 200 words.
Please answer both questions. For the second question, just explain at a high level what you would do. No need to give us extra code for this question. Unknown means that the receiver does not know what the preamble character is.
In our implementation, the preamble is set as an underscore character. Even when the preamble character is unknown, it is still possible to decrypt messages successfully. The decryption process does not depend on the specific preamble character. To adapt the decryption design to accommodate different character preambles, it is necessary to modify the design by storing all decrypted messages, including all potential preambles, in the data memory. By doing so, the decryption design becomes flexible enough to handle different preamble characters, ensuring that the entire decrypted message, regardless of the specific preamble, is stored and available for further processing.

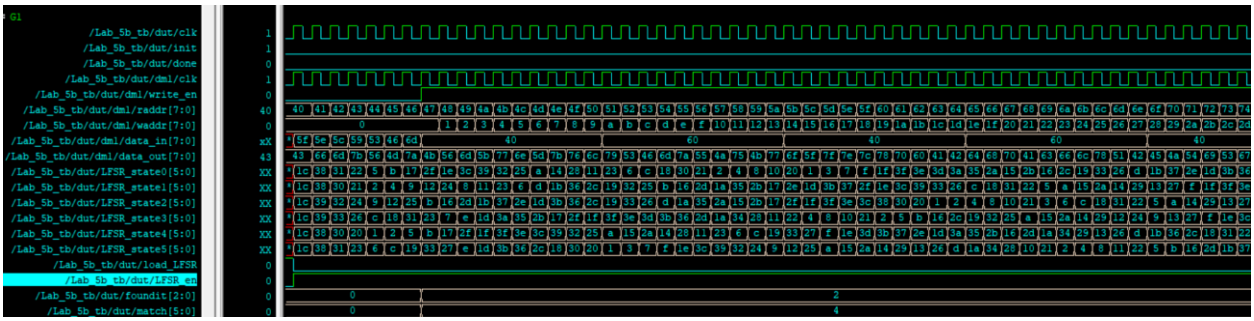
Screenshots

Screenshot of the RTL viewer top level schematic/block diagram in Quartus Or submit your Mentor Precision netlist file if using EDA Playground (5 pts)



Screenshot of your waveform viewer, including variables from both the testbench and the design under test. Or submit your Mentor Precision netlist file if using EDA Playground (5 pts)





Please edit your testbench to pipe the transcript to an output file in your submission, and name the output file “output.txt” (5 pts)

We will be looking for a text file with that name specifically, so be sure to rename it. Nothing is required in the writeup for this question.