**University of Nevada Las Vegas. Department of Electrical and Computer Engineering Laboratories.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Class: | **CPE200L - 1002** | | | Semester: | **Spring 2020** |
|  | | | | | |
| Points |  | Document author: | **David Nakasone, Luis Cardenas** | | |
|  | Author's email: | [**nakasd3@unlv.nevada.edu**](mailto:nakasd3@unlv.nevada.edu) **,** [**Cardel1@unlv.nevada.edu**](mailto:Cardel1@unlv.nevada.edu) | | |
|  | | | |
| Document topic: | **Post Lab 12 Final Project** | | |
| Instructor's comments: | | | | | |

1. **Description:**

This a Group project of 2 people, David Nakasone & Luis Cardenas, in this project two devices are built one for encryption and the other for decryption. To realize these functions Verilog was used to code both the encryption and decryption devices. Two test benches are written respectively for the encryption and decryption device. The test bench stimulates this circuit by acting as the buffer for an arbitrary plain text input stream. Both devices Input 200 words of 16-bit length and then output 200 words of 16-bit length. If the message is less than 200 words, the NULL character will be encrypted. This project is comprised of 5 program files: one for getting the ascii message, a function for encryption, its respective testbench function, a decryption function, and its respective testbench.

1. **Background and Theory:**

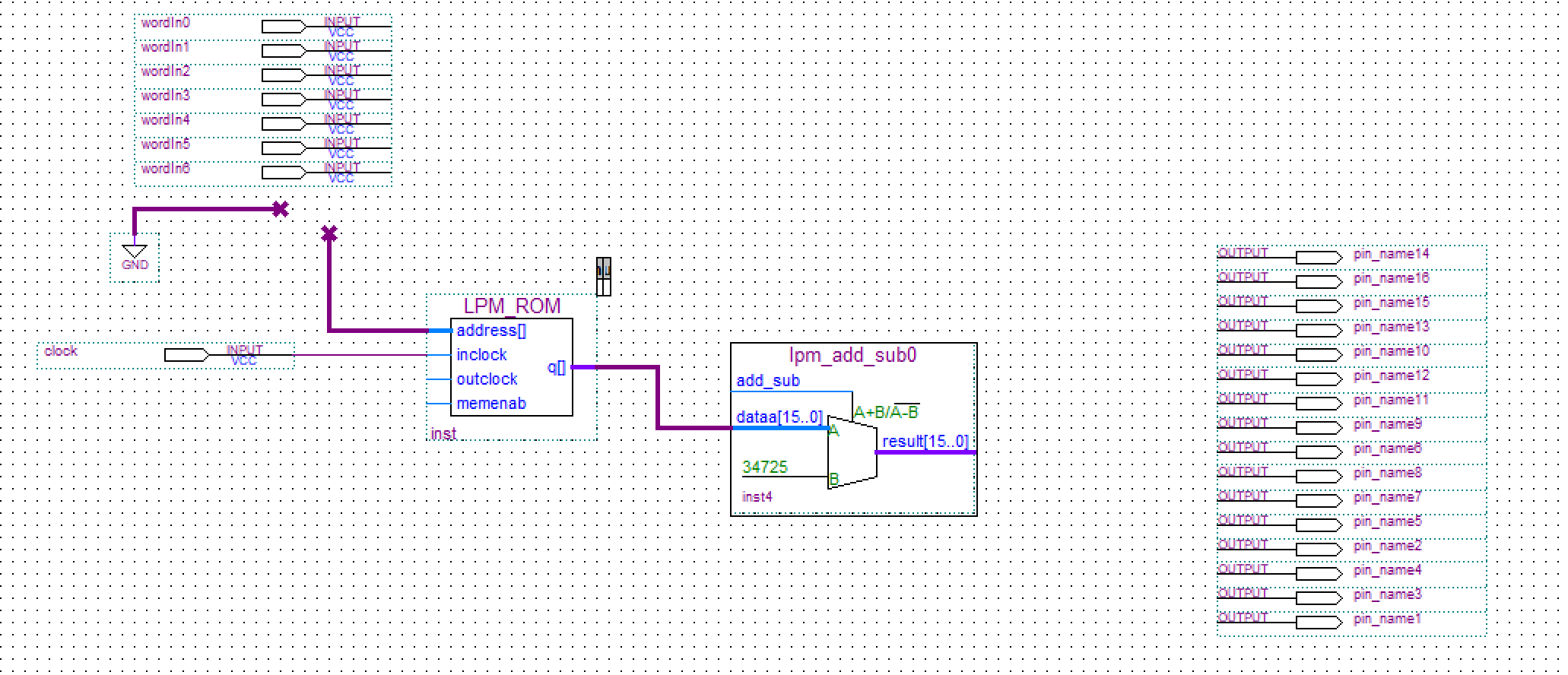
This is a 2-circuit project; for this to work the sender and receiver (both parties) should each have an encryption or/and decryption circuit so they can both transmit and receive. The first circuit is an encryption device is designed to input 200 unencrypted 16-bit words and then output 200 16-bit words; that encrypt the original input.  In each word, the input represents a 7-bit ASCII character that the sender wants to transmit, starting with the LSB of the 16-bit word. The remaining bits are 0. The 16-bit word is encrypted on each rising clock edge. For encryption, the 16-bit word just adds {2^15 + 1957} = \*\*34725\*\* to the input, producing an output. In binary, the key is 16'b 1000 0111 1010 0101; for example, 'A' input = ASCII 65 (decimal), so input is [ 0000 0000 0100 0001]. The input is encrypted to (decimal) 34725 + 65 = 34790, so the output is [ 1000 0111 1110 0110] the process continues until all 200 words have been encrypted. If the message is less than 200 words, the NULL character will be encrypted.

The test bench stimulates this circuit by acting as the buffer for an arbitrary plain text input stream. It is what the sender used to input his message. Using these 200 unencrypted 16-bit words as stimuli, the input is changed between clock edges. As the stimuli provide input to the DUT and output is recorded by the test bench, this represents the point at which the receiver gets the encrypted message. Since the transmission was encrypted, the received data will not be coherent unless the receiver has the proper KEY.

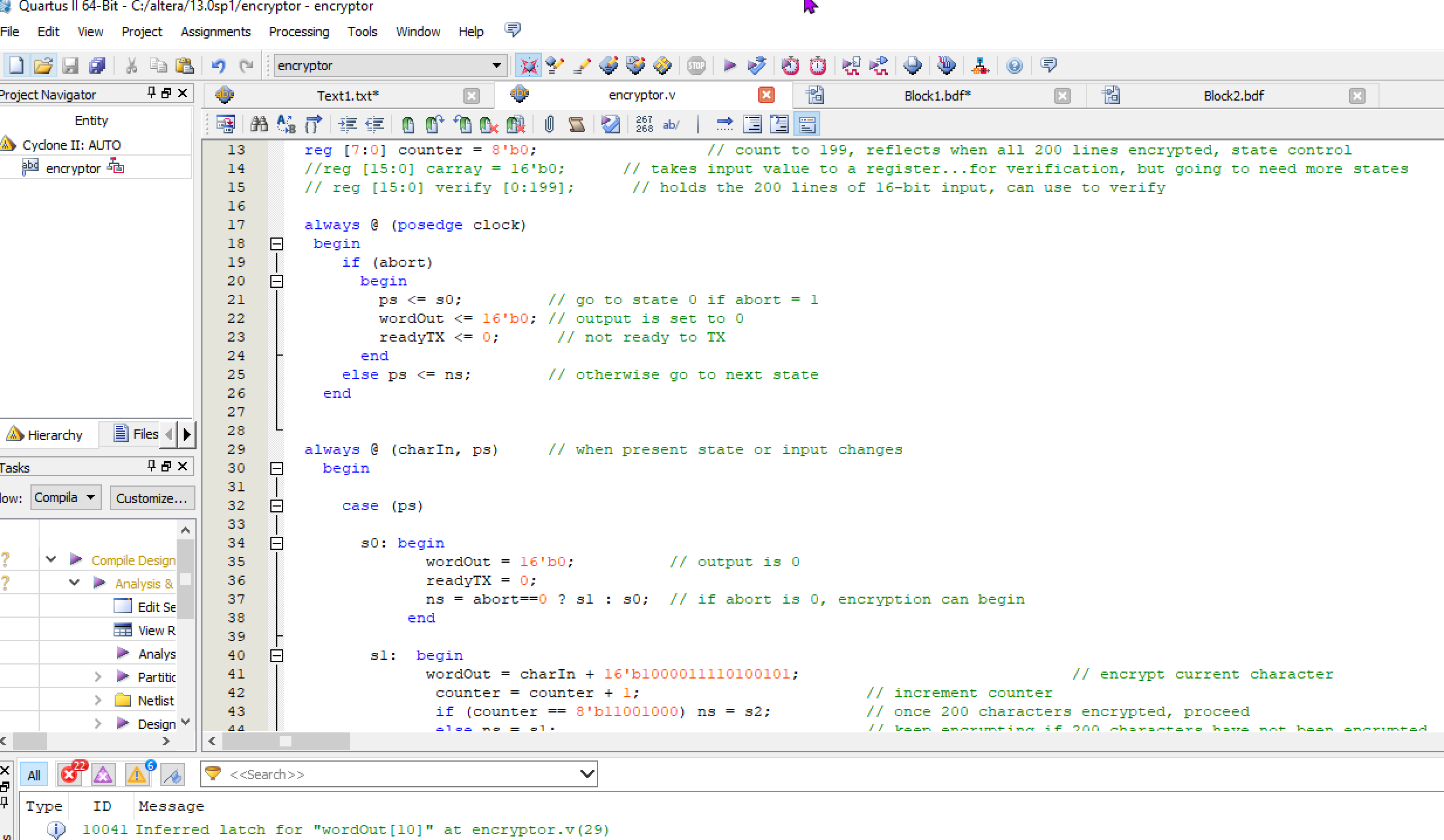
The second circuit is a decryption device; it is customizable to virtually anything, but both parties must use the same encrypt and decrypt method. It is designed to input 200 encrypted 16-bit words and output 200 decrypted 16-bit words, revealing the message from the sender. In each word, the received input represents a character. Since the encrypted input was an unencrypted ASCII character with a key of {2^15 + 1957} = \*\*34725\*\* added to it. For example, if you input [ 1000 0111 1110 0110] to be able to perform the decryption the device will subtract decimal 34725 or binary 16'b 1000 0111 1010 0101. this produces the output of [ 0000 0000 0100 0001], revealing the character 'A'; this process continues until all the characters have been decrypted.

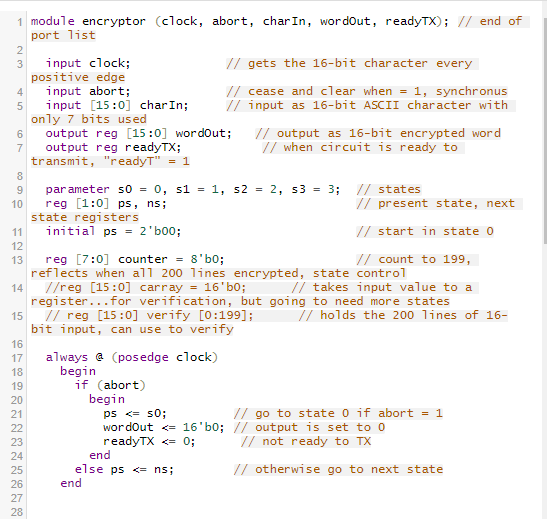
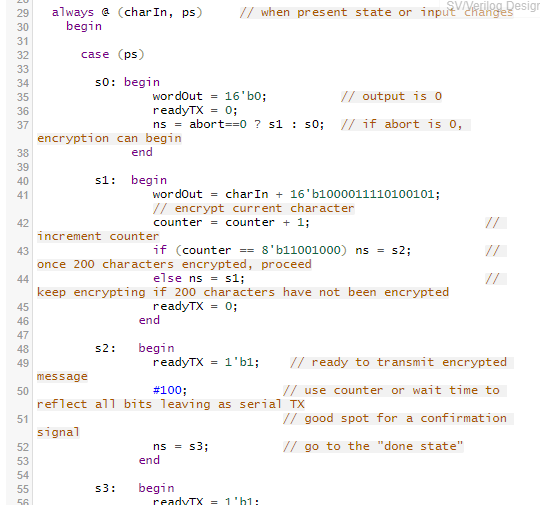
The test bench simulates this circuit with 200 encrypted 16-bit words that would have come from the sending circuit. As the stimuli produce output, the receiver can easily convert the binary ASCII numbers to characters and read the message. The encryption method is customizable to virtually anything, but both parties must use the same encrypt and decrypt method.

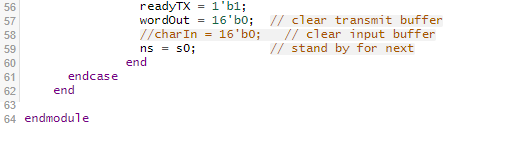
1. **Schematics, Diagrams, and Photos:**

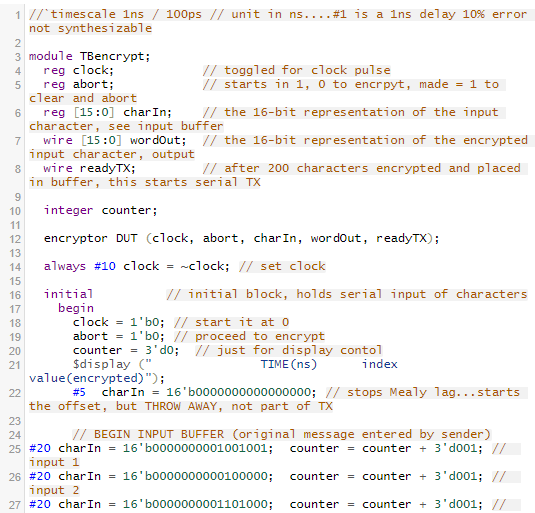
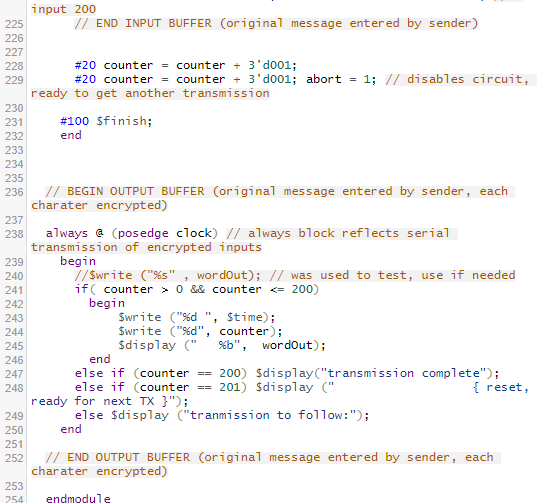
****

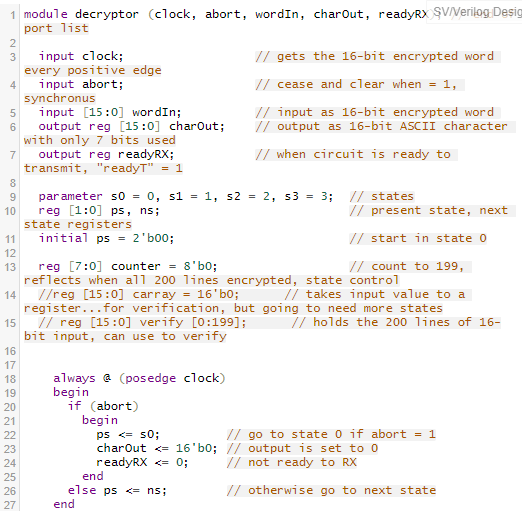
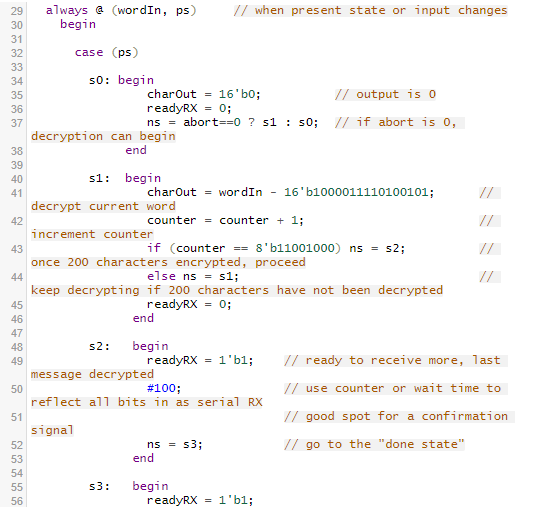
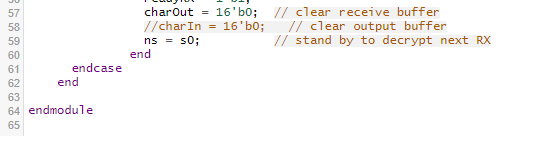
**Fig (1) Encryption Device Code:**

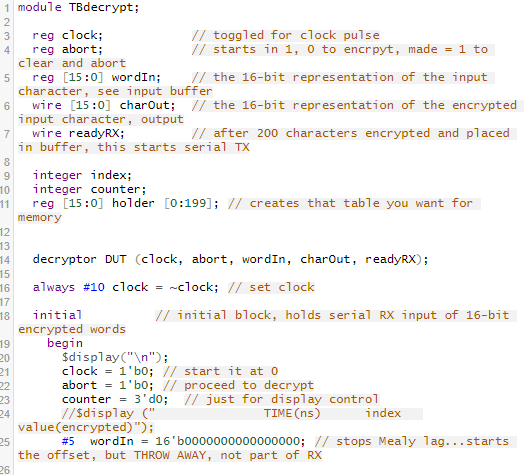
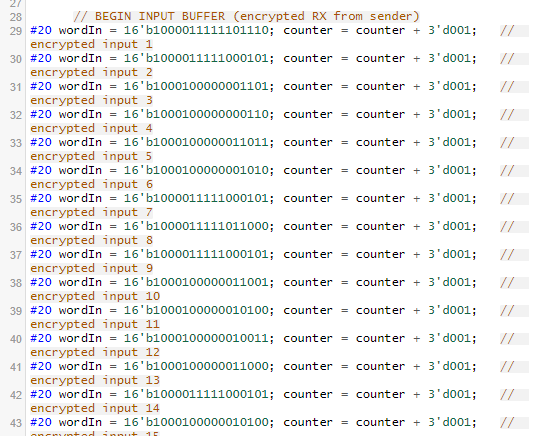
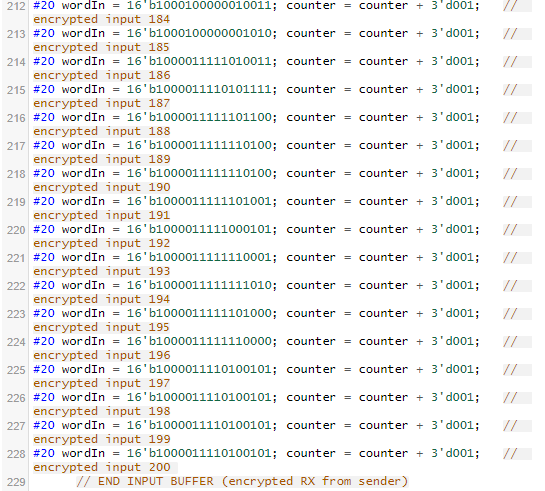
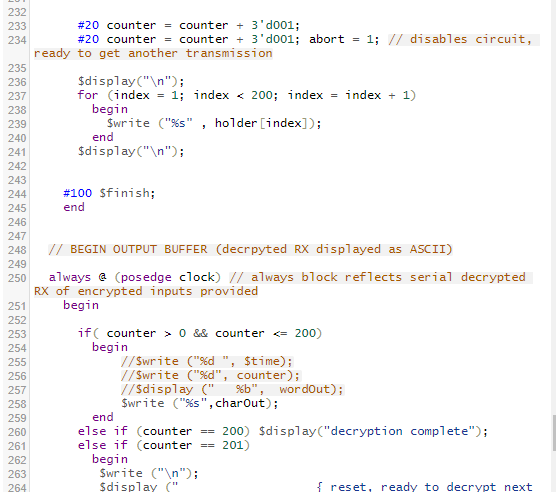
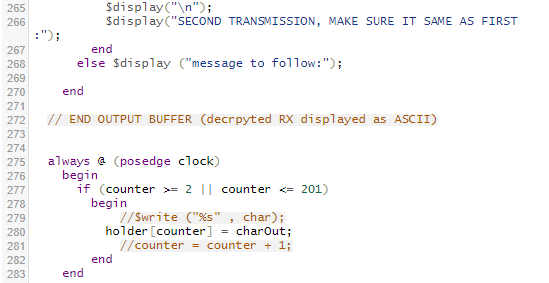
****

**** ****

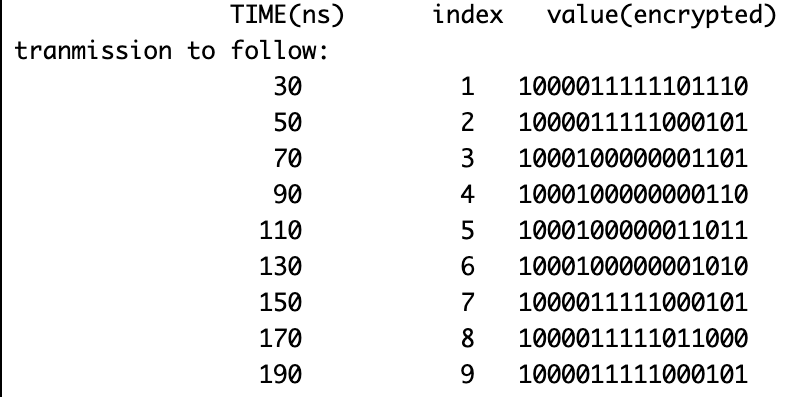


**Fig (2) Encryption Testbench:** ****

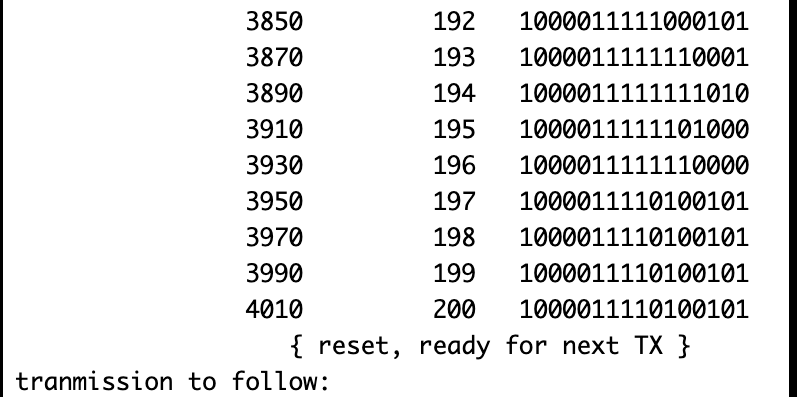
**Fig (3) Decryption Device Code:** **** 

**Fig (4) Decryption Testbench:** ****   

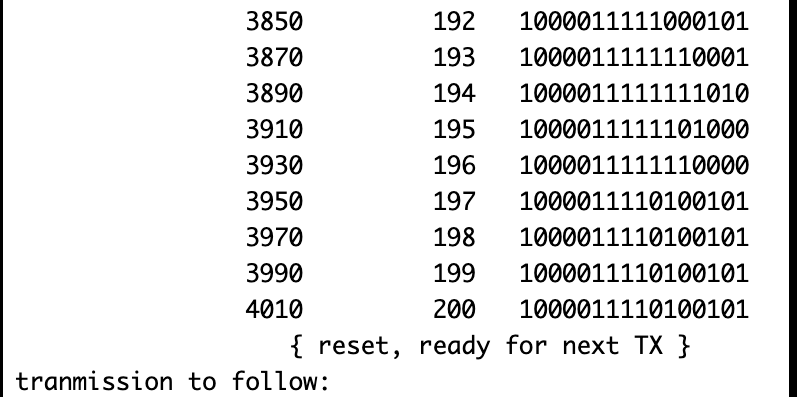
**Fig (5) encryption input (raw):**



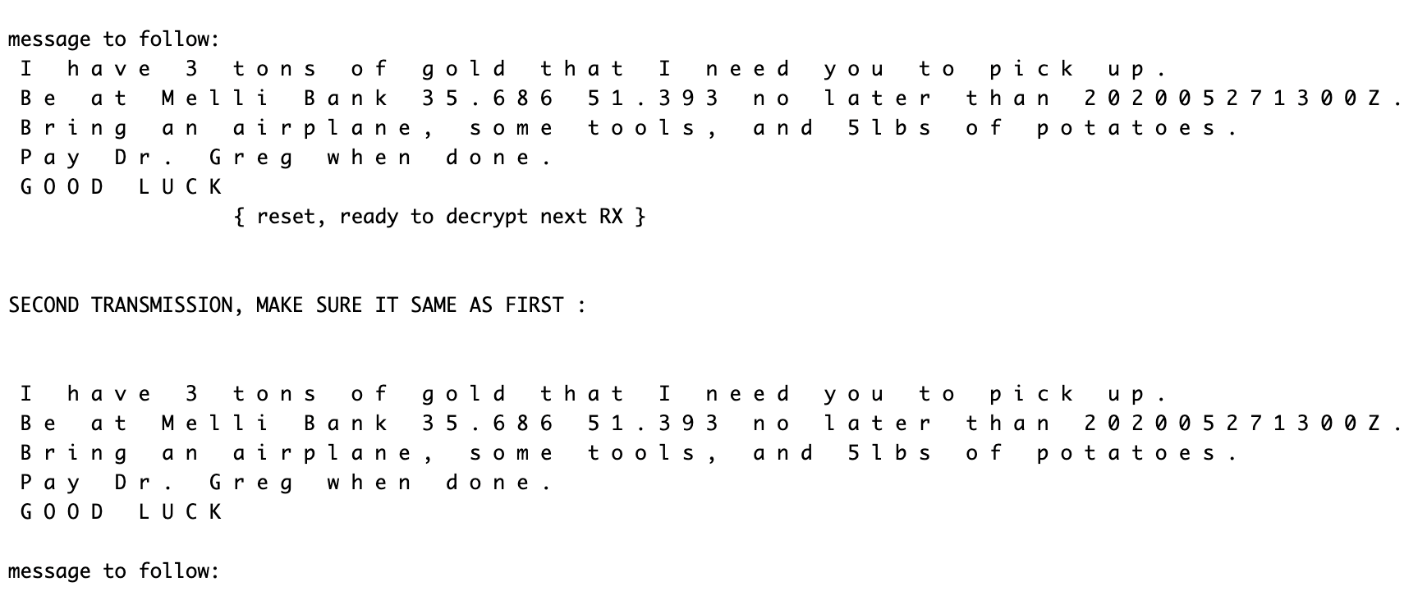
**Fig (6) encryption output(encrypted):**



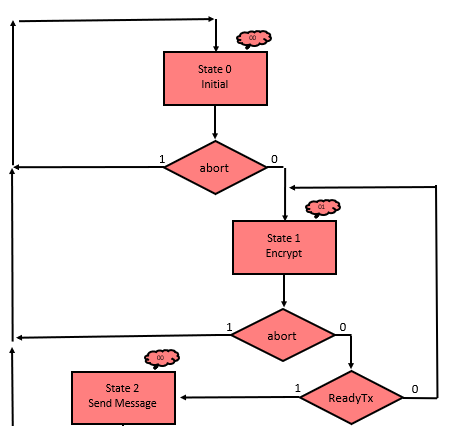
**Fig(7) decryption input is same as encryption output:**

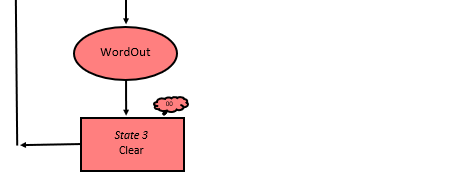


**Fig (8) decrytpor output with verification**

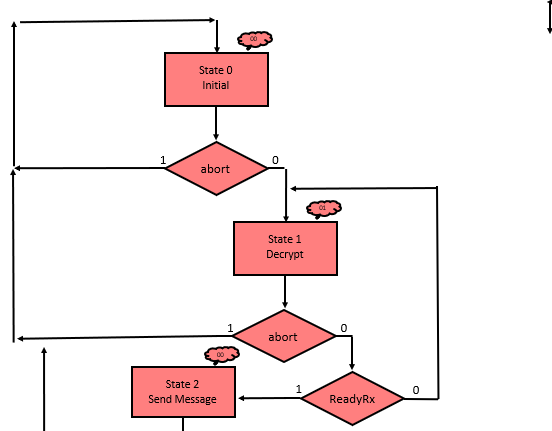
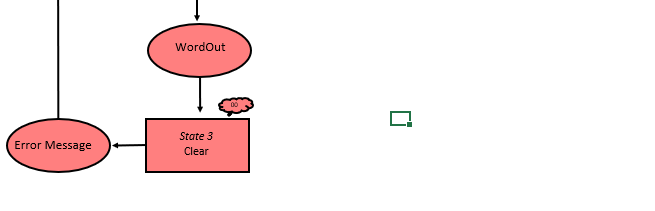


**Fig(9) SM Chart Decryption Device:**



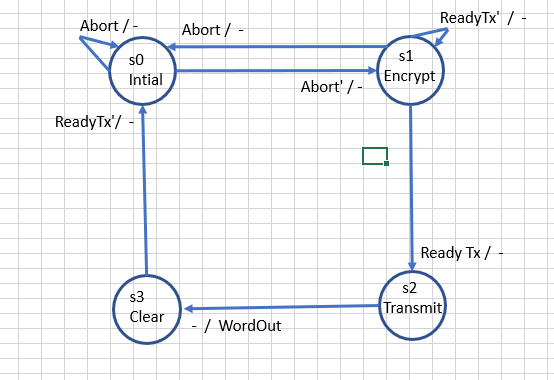


**Fig (10) SM Chart Encryption Device:**

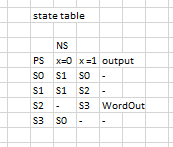
 



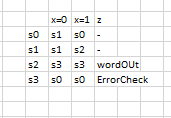
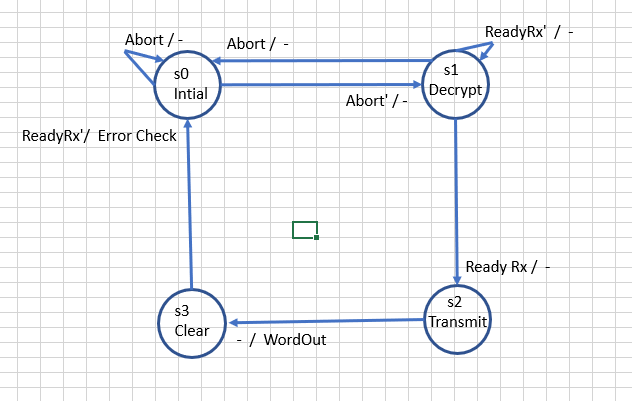
**Fig (11) State graph for encryption and state table:**

****



****

**Fig (12) State Graph for Decryption and State Table:**

****

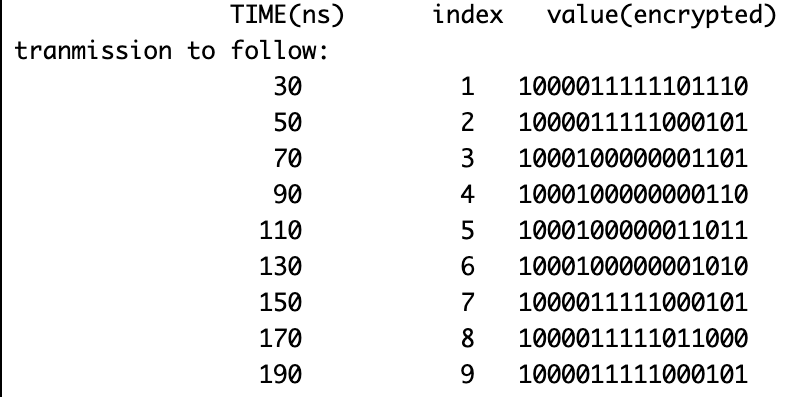
1. **Circuit operation:**

The encryption circuit is constructed by using three inputs Clock, Abort, and CharIn; as well as 2 outputs WordOut and ReadyTx. The clock input will be used to get the 16 bits that comprise the word being encrypted. Abort is a synchronous input being used as a clear when set equal to one, this will cease all operations and reset the circuit to state 0. CharIn is made into a 16-bit array used to store the 16-bit ASCII character with only 7 bits being used. We have WordOut set as a 16-bit array to output the now encrypted ASCII character. The output ReadyTx is being used and initially set to 0 as a controller to tell the circuit when to transmit by changing from 0 to 1. Four states are being used in order to realize the encryption function. When abort is set to 1 or the functions begins state 0; this will set the output to 0 to initialize, and ReadyTx is set to 0 to stop loading; If abort is set to 0 then state 1 can begin. State 1 is being used to encrypt an ASCII character using WordOut = CharIn + 16'b1000011110100101. The counter is increased in state one to read the next variable assigned; this will sequentially continue to read each word until all 200 words are read. Once all 200 words are read the function moves onto the third state, state 2, here the function will transmit the encrypted message to the decryption device. Once state two has been completed the fourth and final state, state 3, in this state the machine will clear the transmit buffer clear the input buffer and reset back to state 0.

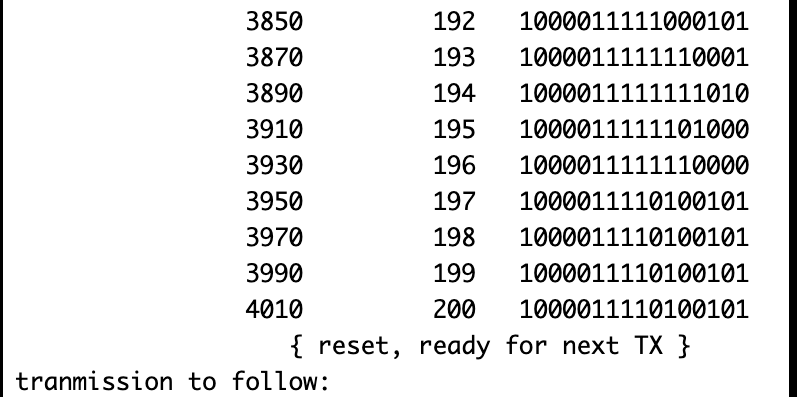
The decryption circuit works similar to the encryption circuit using three inputs clock abort CharIn; as well as two outputs WordOut and ReadyRx. These I/O’s work in similar fashion the difference being this device serves to decrypt. The clock input will be used to get the 16 bits that comprise the word being decrypted. CharIn is made into a 16-bit array used to store the 16-bit ASCII the additional nine bits are now being used for the encryption. The output ReadyRx is still being used and initially set to 0 as a controller to tell the circuit when to transmit by changing from 0 to 1. Four states are being used in order to realize the decryption function. When abort is set to 1 or the functions begin state 0; this will set the output to 0 to initialize, and ReadyRx is set to 0 to stop loading; If abort is set to 0 then state 1 can begin. State 1 is being used to encrypt an ASCII character using WordOut = CharIn - 16'b1000011110100101. The counter is increased in state one to read the next variable assigned; this will sequentially continue to read each word until all 200 words are read. Once all 200 words are read the function moves onto the third state, state 2, here the function will output the message. Once state two has been completed the fourth and final state, state 3, in this state the machine will clear the transmit buffer clear the input buffer and reset back to state 0. The encryption device will output another result with the original text to verify output.

1. **Simulation Results with annotation:**

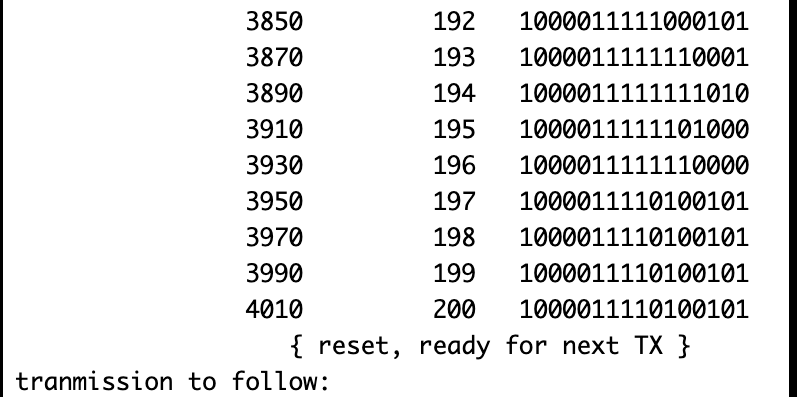
**Fig (5) encryption input (raw):**



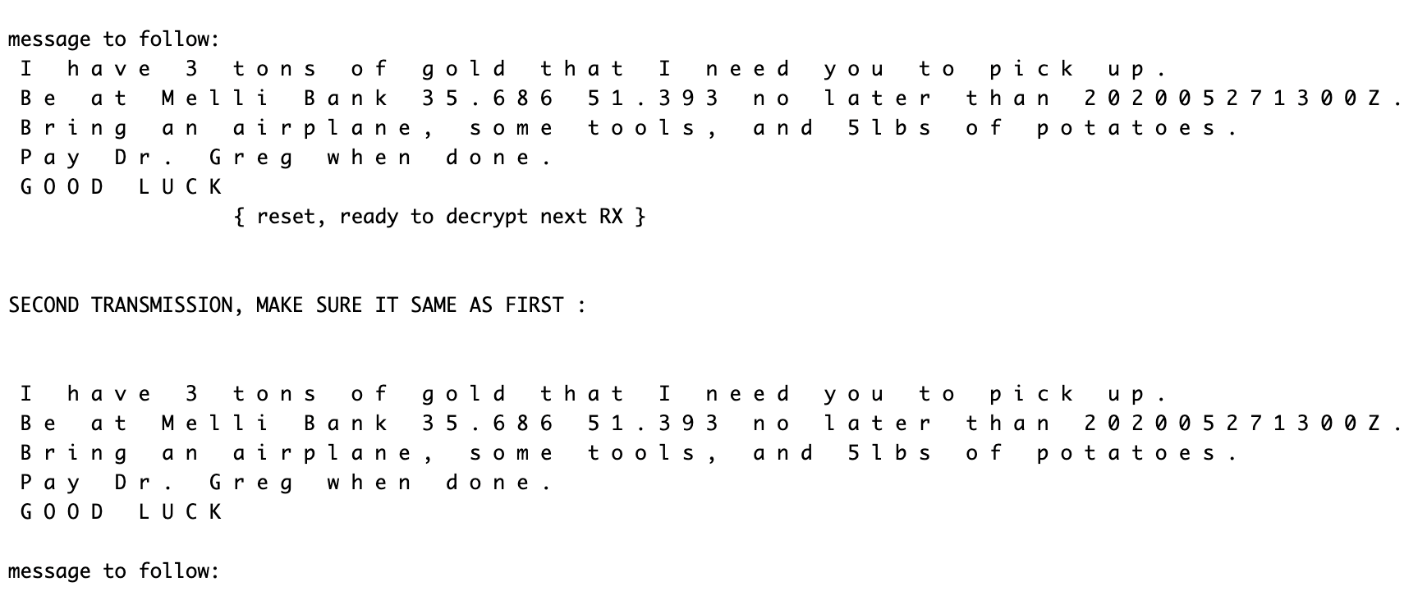
**Fig (6) encryption output(encrypted):**



**Fig(7) decryption input is same as encryption output:**



**Fig (8) decrytpor output with verification**



1. **Encountered Problems and how they were solved:**

ASCII input and output are hard to realize and very tedious to obtain manually. To overcome this obstacle and not have to manually code thousands of entries, we used simple programs to generate the segments of code that we needed. The computer program also helped guide the design, get a working model rapidly, and verify circuit output. Working from home proved difficult when attempting to realize a full hardware implementation so this code is done with only the software in mind and written behaviorally vs structurally.

1. **Any information regarding the project that might be interesting:**

The code used to encrypt the algorithm is done by adding +1957 in binary to the binary ascii value. 1957 was chosen because it is the year UNLV was founded. This project was done in self-isolation. This project was done during self-isolation and was done through video messaging, Google Drive, and EDAplayground. As lab partners throughout the semester, we completed all 12 labs together. The first few labs were very basic, and we did not know much about sequential circuits. The basics are important in logic design; every topic is interrelated and adds a new level of abstraction to build upon in a later lab. Circuitry has to be properly arranged to function as desired.

It is notable that the designer needs to have the skills with both combinational and sequential circuits. With this project, we were able to apply the majority of principles that we learned throughout the semester. Working with encryption forced us to handle much larger data sets, but we found that if you can handle 2 in a row, then 200 is the same thing. The encryption scheme is very weak for simplicity, and because we are short on time and quarantined can be anything you want as long as there is an algorithm or look up table to decrypt. If decryption can't be done, then it defeats the purpose. If someone can decrypt the message, then that also defeats the purpose. But this scheme can be expanded or modified in any way. This started to get very involved as soon as work began.

1. **Conclusions:**

In conclusion there are some big numbers to work with and we Implemented a circuit without any I/O or ability to demonstrate, as this was done during virus outbreak of 2020, this makes it all theoretical. C++ was a good way to get the idea in writing and provides a somewhat easy transition into Verilog code. At least you can use some of the existing functions to generate some of the 1000's of inputs you will need. The array index of C++ being reversed.

To realize these functions Verilog was used to code both the encryption and decryption devices. Two test benches are written respectively for the encryption and decryption device. The test bench stimulates this circuit by acting as the buffer for an arbitrary plain text input stream. Both devices Input 200 words of 16-bit length and then output 200 words of 16-bit length. If the message is less than 200 words, the NULL character will be encrypted. This project is comprised of 5 program files: one for getting the ascii message, a function for encryption, its respective testbench function, a decryption function, and its respective testbench.

1. **Program Files:**

**The c++:**

<https://repl.it/@davenakasone/20200424-lab12-v2>

**The proof of input:**

<https://www.edaplayground.com/x/52_8>

**The encryptor:**

<https://www.edaplayground.com/x/3hWW>

**The decryptor:**

<https://www.edaplayground.com/x/2j7c>

**Google Drive:**

<https://drive.google.com/open?id=1hB0f9nqWGWZmv1ZEqQpXNPdQT1TQsS0r>