Class:	CPE200L - 1002	Semester:	Spring 2020

Points		Document author:	David Nakasone, Luis Cardenas
		Author's email:	nakasd3@unlv.nevada.edu , 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.

2. 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.

3. Schematics, Diagrams, and Photos:

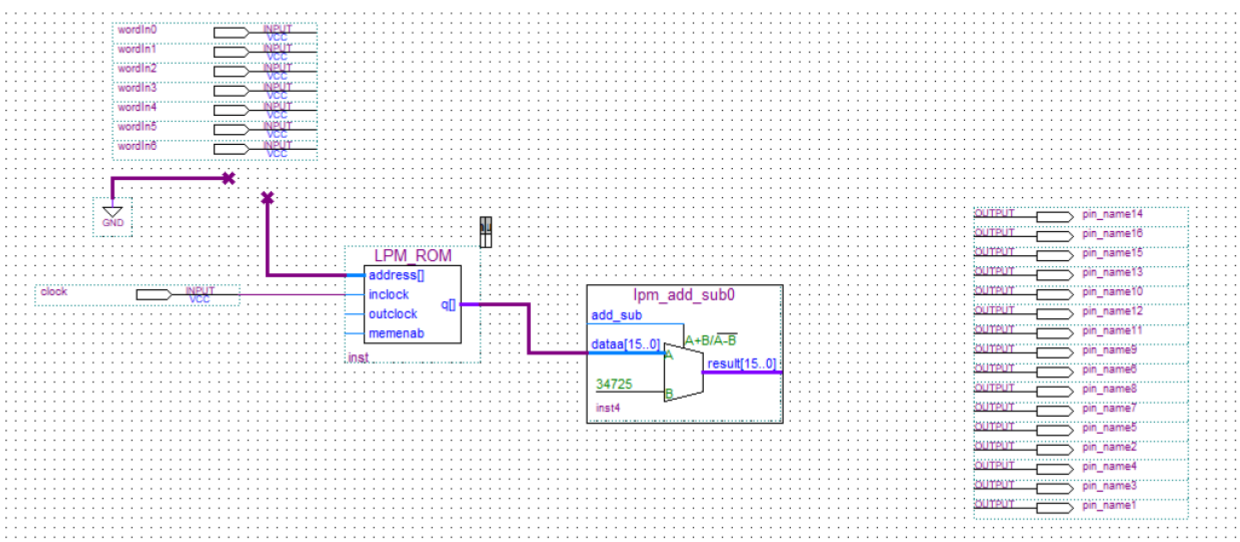
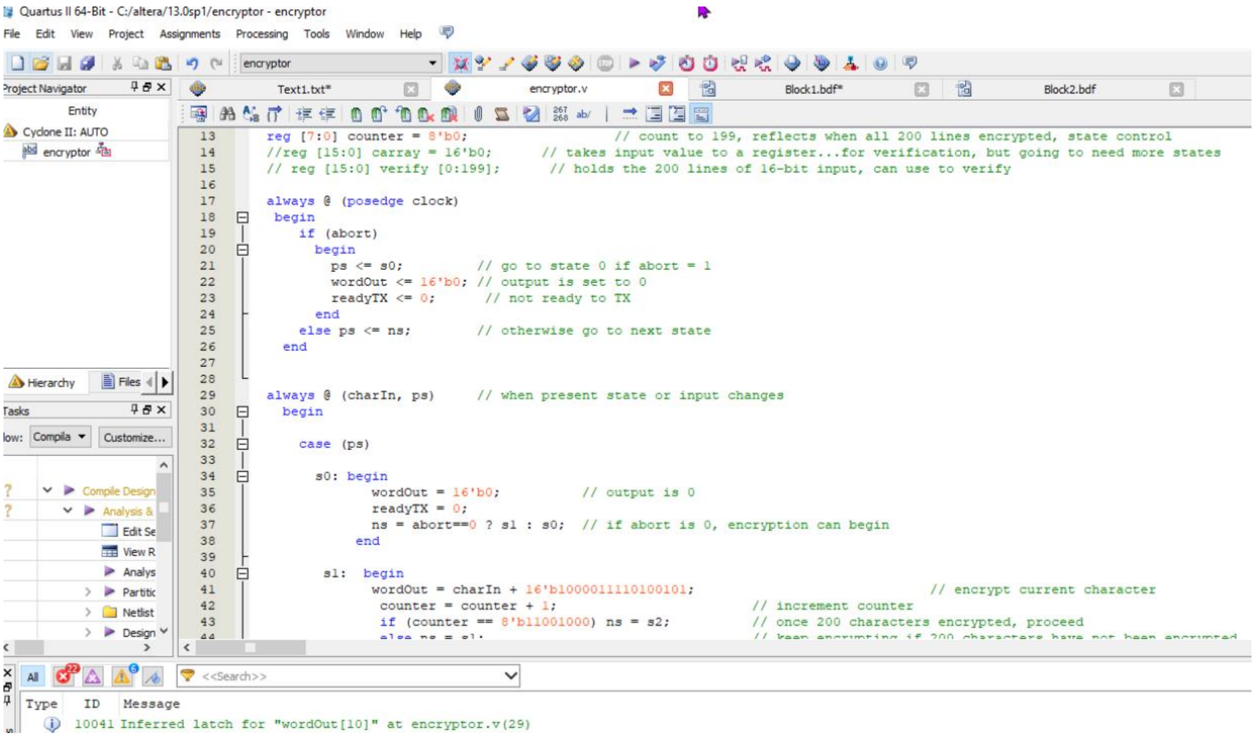


Fig (1) Encryption Device Code:



```
module encryptor (clock, abort, charIn, wordOut, readyTX); // end of port list

  input clock;              // gets the 16-bit character every positive edge
  input abort;              // cease and clear when = 1, synchronus
  input [15:0] charIn;      // input as 16-bit ASCII character with only 7 bits used
  output reg [15:0] wordOut;    // output as 16-bit encrypted word
  output reg readyTX;            // when circuit is ready to transmit, "readyT" = 1

  parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3;  // states
  reg [1:0] ps, ns;                          // present state, next state registers
  initial ps = 2'b00;                        // start in state 0

  reg [7:0] counter = 8'b0;                  // count to 199, reflects when all 200 lines encrypted, state control
  //reg [15:0] carray = 16'b0;       // takes input value to a register...for verification, but going to need more states
  // reg [15:0] verify [0:199];       // holds the 200 lines of 16-bit input, can use to verify

  always @ (posedge clock)
    begin
      if (abort)
        begin
          ps <= s0;          // go to state 0 if abort = 1
          wordOut <= 16'b0; // output is set to 0
          readyTX <= 0;       // not ready to TX
        end
      else ps <= ns;         // otherwise go to next state
    end


```

```
  always @ (charIn, ps)     // when present state or input changes
    begin

      case (ps)

        s0: begin
                wordOut = 16'b0;            // output is 0
                readyTX = 0;
                ns = abort==0 ? s1 : s0;   // if abort is 0, encryption can begin
              end

        s1:  begin
                wordOut = charIn + 16'b1000011110100101;
                // encrypt current character
                counter = counter + 1;                          // increment counter
                if (counter == 8'b11001000) ns = s2;            // once 200 characters encrypted, proceed
                else ns = s1;                                   // keep encrypting if 200 characters have not been encrypted
                readyTX = 0;
              end

        s2:   begin
                readyTX = 1'b1;     // ready to transmit encrypted message
                #100;               // use counter or wait time to reflect all bits leaving as serial TX
                                    // good spot for a confirmation signal
                ns = s3;            // go to the "done state"
              end

        s3:   begin
                readyTX = 1'b1;
                wordOut = 16'b0;  // clear transmit buffer
                //charIn = 16'b0;    // clear input buffer
                ns = s0;          // stand by for next
              end
      endcase
    end

endmodule
```

Fig (2) Encryption Testbench:

```
//`timescale 1ns / 100ps // unit in ns....#1 is a 1ns delay 10% error not synthesizable

module TBencrypt;
  reg clock;             // toggled for clock pulse
  reg abort;             // starts in 1, 0 to encrpyt, made = 1 to clear and abort
  reg [15:0] charIn;     // the 16-bit representation of the input character, see input buffer
  wire [15:0] wordOut;   // the 16-bit representation of the encrypted input character, output
  wire readyTX;          // after 200 characters encrypted and placed in buffer, this starts serial TX

  integer counter;

  encryptor DUT (clock, abort, charIn, wordOut, readyTX);

  always #10 clock = ~clock; // set clock

  initial          // initial block, holds serial input of characters
    begin
      clock = 1'b0; // start it at 0
      abort = 1'b0; // proceed to encrypt
      counter = 3'd0;  // just for display contol
      $display ("               TIME(ns)      index      value(encrypted)");
      #5  charIn = 16'b0000000000000000; // stops Mealy lag...starts the offset, but THROW AWAY, not part of TX

      // BEGIN INPUT BUFFER (original message entered by sender)
#20 charIn = 16'b0000000001001001;  counter = counter + 3'd001; // input 1
#20 charIn = 16'b0000000000100000;  counter = counter + 3'd001; // input 2
#20 charIn = 16'b0000000001101000;  counter = counter + 3'd001; //
```

```
input 200
      // END INPUT BUFFER (original message entered by sender)


      #20 counter = counter + 3'd001;
      #20 counter = counter + 3'd001; abort = 1; // disables circuit, ready to get another transmission

    #100 $finish;
    end



  // BEGIN OUTPUT BUFFER (original message entered by sender, each charater encrypted)

  always @ (posedge clock) // always block reflects serial transmission of encrypted inputs
    begin
      //$write ("%s" , wordOut); // was used to test, use if needed
      if( counter > 0 && counter <= 200)
        begin
            $write ("%d ", $time);
            $write ("%d", counter);
            $display ("   %b",  wordOut);
        end
      else if (counter == 200) $display("transmission complete");
      else if (counter == 201) $display ("                   { reset, ready for next TX }");
      else $display ("tranmission to follow:");
    end

  // END OUTPUT BUFFER (original message entered by sender, each charater encrypted)

  endmodule
```

Fig (3) Decryption Device Code:

```
module decryptor (clock, abort, wordIn, charOut, readyRX); // end of port list

  input clock;                    // gets the 16-bit encrypted word every positive edge
  input abort;                    // cease and clear when = 1, synchronus
  input [15:0] wordIn;            // input as 16-bit encrypted word
  output reg [15:0] charOut;      // output as 16-bit ASCII character with only 7 bits used
  output reg readyRX;             // when circuit is ready to transmit, "readyT" = 1

  parameter s0 = 0, s1 = 1, s2 = 2, s3 = 3;  // states
  reg [1:0] ps, ns;                          // present state, next state registers
  initial ps = 2'b00;                        // start in state 0

  reg [7:0] counter = 8'b0;                  // count to 199, reflects when all 200 lines encrypted, state control
  //reg [15:0] carray = 16'b0;       // takes input value to a register...for verification, but going to need more states
  // reg [15:0] verify [0:199];       // holds the 200 lines of 16-bit input, can use to verify


    always @ (posedge clock)
    begin
      if (abort)
        begin
          ps <= s0;          // go to state 0 if abort = 1
          charOut <= 16'b0;  // output is set to 0
          readyRX <= 0;      // not ready to RX
        end
      else ps <= ns;         // otherwise go to next state
    end
```

```
  always @ (wordIn, ps)      // when present state or input changes
    begin

      case (ps)

        s0: begin
                charOut = 16'b0;           // output is 0
                readyRX = 0;
                ns = abort==0 ? s1 : s0;   // if abort is 0, decryption can begin
             end

        s1:  begin
                charOut = wordIn - 16'b1000011110100101;      // decrypt current word
                counter = counter + 1;                        // increment counter
                if (counter == 8'b11001000) ns = s2;          // once 200 characters encrypted, proceed
                else ns = s1;                                 // keep decrypting if 200 characters have not been decrypted
                readyRX = 0;
              end

        s2:   begin
                readyRX = 1'b1;     // ready to receive more, last message decrypted
                #100;               // use counter or wait time to reflect all bits in as serial RX
                                    // good spot for a confirmation signal
                ns = s3;            // go to the "done state"
              end

        s3:   begin
                readyRX = 1'b1;
                charOut = 16'b0;  // clear receive buffer
                //charIn = 16'b0;   // clear output buffer
                ns = s0;          // stand by to decrypt next RX
              end
      endcase
    end

endmodule
```

Fig (4) Decryption Testbench:

```
module TBdecrypt;

  reg clock;               // toggled for clock pulse
  reg abort;               // starts in 1, 0 to encrpyt, made = 1 to clear and abort
  reg [15:0] wordIn;       // the 16-bit representation of the input character, see input buffer
  wire [15:0] charOut;     // the 16-bit representation of the encrypted input character, output
  wire readyRX;            // after 200 characters encrypted and placed in buffer, this starts serial TX

  integer index;
  integer counter;
  reg [15:0] holder [0:199]; // creates that table you want for memory


  decryptor DUT (clock, abort, wordIn, charOut, readyRX);

  always #10 clock = ~clock; // set clock

  initial          // initial block, holds serial RX input of 16-bit encrypted words
    begin
      $display("\n");
      clock = 1'b0; // start it at 0
      abort = 1'b0; // proceed to decrypt
      counter = 3'd0;  // just for display control
      //$display ("               TIME(ns)      index      value(encrypted)");
      #5  wordIn = 16'b0000000000000000; // stops Mealy lag...starts the offset, but THROW AWAY, not part of RX

```

```
        // BEGIN INPUT BUFFER (encrypted RX from sender)
#20 wordIn = 16'b1000011111101110; counter = counter + 3'd001;    //
encrypted input 1
#20 wordIn = 16'b1000011111000101; counter = counter + 3'd001;    //
encrypted input 2
#20 wordIn = 16'b1000100000001101; counter = counter + 3'd001;    //
encrypted input 3
#20 wordIn = 16'b1000100000000110; counter = counter + 3'd001;    //
encrypted input 4
#20 wordIn = 16'b1000100000011011; counter = counter + 3'd001;    //
encrypted input 5
#20 wordIn = 16'b1000100000001010; counter = counter + 3'd001;    //
encrypted input 6
#20 wordIn = 16'b1000011111000101; counter = counter + 3'd001;    //
encrypted input 7
#20 wordIn = 16'b1000011111011000; counter = counter + 3'd001;    //
encrypted input 8
#20 wordIn = 16'b1000011111000101; counter = counter + 3'd001;    //
encrypted input 9
#20 wordIn = 16'b1000100000011001; counter = counter + 3'd001;    //
encrypted input 10
#20 wordIn = 16'b1000100000010100; counter = counter + 3'd001;    //
encrypted input 11
#20 wordIn = 16'b1000100000010011; counter = counter + 3'd001;    //
encrypted input 12
#20 wordIn = 16'b1000100000011000; counter = counter + 3'd001;    //
encrypted input 13
#20 wordIn = 16'b1000011111000101; counter = counter + 3'd001;    //
encrypted input 14
#20 wordIn = 16'b1000100000010100; counter = counter + 3'd001;    //
encrypted input 15
```

```
#20 wordIn = 16'b1000100000010011; counter = counter + 3'd001;    // encrypted input 184
#20 wordIn = 16'b1000100000001010; counter = counter + 3'd001;    // encrypted input 185
#20 wordIn = 16'b1000011111010011; counter = counter + 3'd001;    // encrypted input 186
#20 wordIn = 16'b1000011110101111; counter = counter + 3'd001;    // encrypted input 187
#20 wordIn = 16'b1000011111101100; counter = counter + 3'd001;    // encrypted input 188
#20 wordIn = 16'b1000011111110100; counter = counter + 3'd001;    // encrypted input 189
#20 wordIn = 16'b1000011111110100; counter = counter + 3'd001;    // encrypted input 190
#20 wordIn = 16'b1000011111101001; counter = counter + 3'd001;    // encrypted input 191
#20 wordIn = 16'b1000011111000101; counter = counter + 3'd001;    // encrypted input 192
#20 wordIn = 16'b1000011111110001; counter = counter + 3'd001;    // encrypted input 193
#20 wordIn = 16'b1000011111111010; counter = counter + 3'd001;    // encrypted input 194
#20 wordIn = 16'b1000011111101000; counter = counter + 3'd001;    // encrypted input 195
#20 wordIn = 16'b1000011111110000; counter = counter + 3'd001;    // encrypted input 196
#20 wordIn = 16'b1000011110100101; counter = counter + 3'd001;    // encrypted input 197
#20 wordIn = 16'b1000011110100101; counter = counter + 3'd001;    // encrypted input 198
#20 wordIn = 16'b1000011110100101; counter = counter + 3'd001;    // encrypted input 199
#20 wordIn = 16'b1000011110100101; counter = counter + 3'd001;    // encrypted input 200
      // END INPUT BUFFER (encrypted RX from sender)
```

```
      #20 counter = counter + 3'd001;
      #20 counter = counter + 3'd001; abort = 1; // disables circuit, ready to get another transmission

      $display("\n");
      for (index = 1; index < 200; index = index + 1)
        begin
          $write ("%s" , holder[index]);
        end
      $display("\n");


    #100 $finish;
    end


  // BEGIN OUTPUT BUFFER (decrpyted RX displayed as ASCII)

  always @ (posedge clock) // always block reflects serial decrypted RX of encrypted inputs provided
    begin

      if( counter > 0 && counter <= 200)
        begin
            //$write ("%d ", $time);
            //$write ("%d", counter);
            //$display ("   %b",  wordOut);
            $write ("%s",charOut);
        end
      else if (counter == 200) $display("decryption complete");
      else if (counter == 201)
        begin
         $write ("\n");
         $display ("                   { reset, ready to decrypt next
          $display("\n");
          $display("SECOND TRANSMISSION, MAKE SURE IT SAME AS FIRST :");
        end
      else $display ("message to follow:");

    end

  // END OUTPUT BUFFER (decrpyted RX displayed as ASCII)


  always @ (posedge clock)
    begin
      if (counter >= 2 || counter <= 201)
        begin
            //$write ("%s" , char);
          holder[counter] = charOut;
            //counter = counter + 1;
        end
    end
```

Fig (5) encryption input (raw):

TIME(ns)	index	value(encrypted)
tranmission to follow:		
30	1	1000011111101110
50	2	1000011111000101
70	3	1000100000001101
90	4	1000100000000110
110	5	1000100000011011
130	6	1000100000001010
150	7	1000011111000101
170	8	1000011111011000
190	9	1000011111000101

Fig (6) encryption output(encrypted):

3850	192	1000011111000101
3870	193	1000011111110001
3890	194	1000011111111010
3910	195	1000011111101000
3930	196	1000011111110000
3950	197	1000011110100101
3970	198	1000011110100101
3990	199	1000011110100101
4010	200	1000011110100101
{ reset, ready for next TX }		
tranmission to follow:		

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

```
                    3850        192   1000011111000101
                    3870        193   1000011111110001
                    3890        194   1000011111111010
                    3910        195   1000011111101000
                    3930        196   1000011111110000
                    3950        197   1000011110100101
                    3970        198   1000011110100101
                    3990        199   1000011110100101
                    4010        200   1000011110100101
                       { reset, ready for next TX }
tranmission to follow:
```

Fig (8) decrytpor output with verification

```
message to follow:
 I have 3 tons of gold that I need you to pick up.
 Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
 Bring an airplane, some tools, and 5lbs of potatoes.
 Pay Dr. Greg when done.
 GOOD LUCK
              { reset, ready to decrypt next RX }


SECOND TRANSMISSION, MAKE SURE IT SAME AS FIRST :


 I have 3 tons of gold that I need you to pick up.
 Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
 Bring an airplane, some tools, and 5lbs of potatoes.
 Pay Dr. Greg when done.
 GOOD LUCK

message to follow:
```

Fig(9) SM Chart Decryption Device:

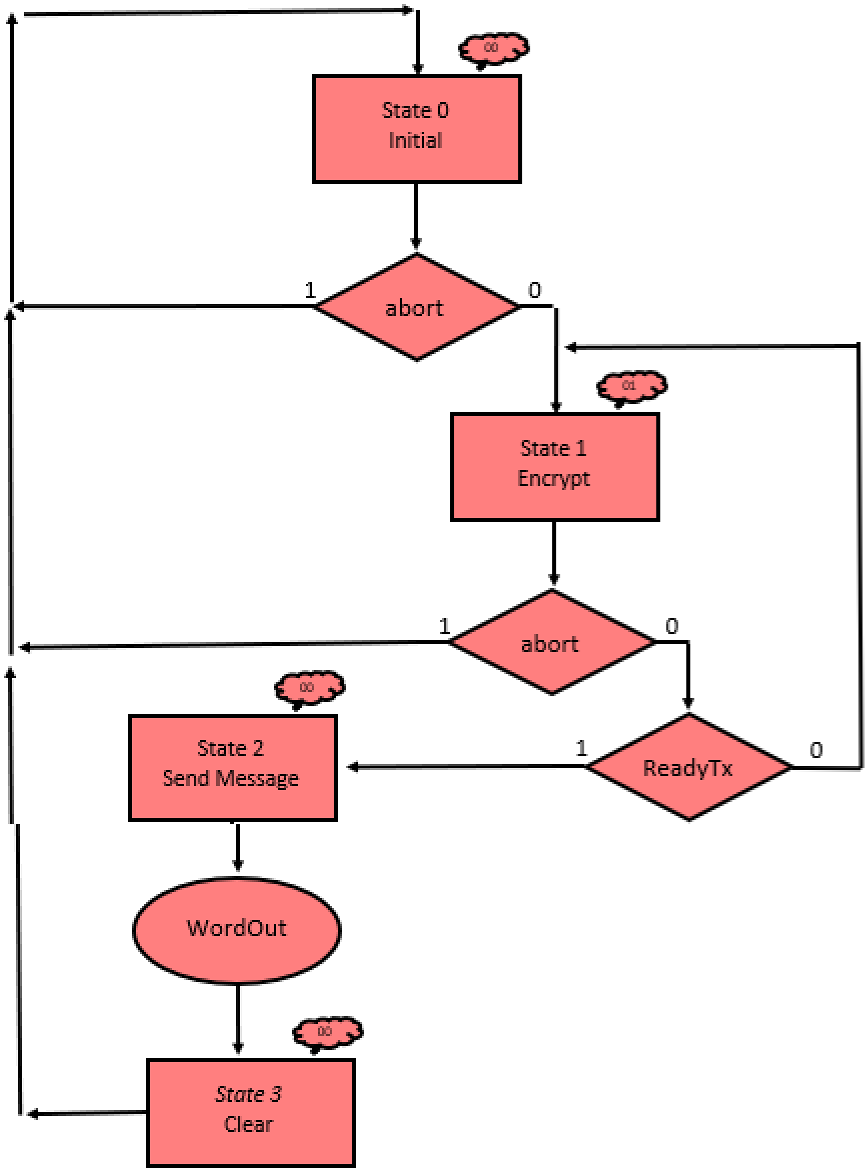


Fig (10) SM Chart Encryption Device:

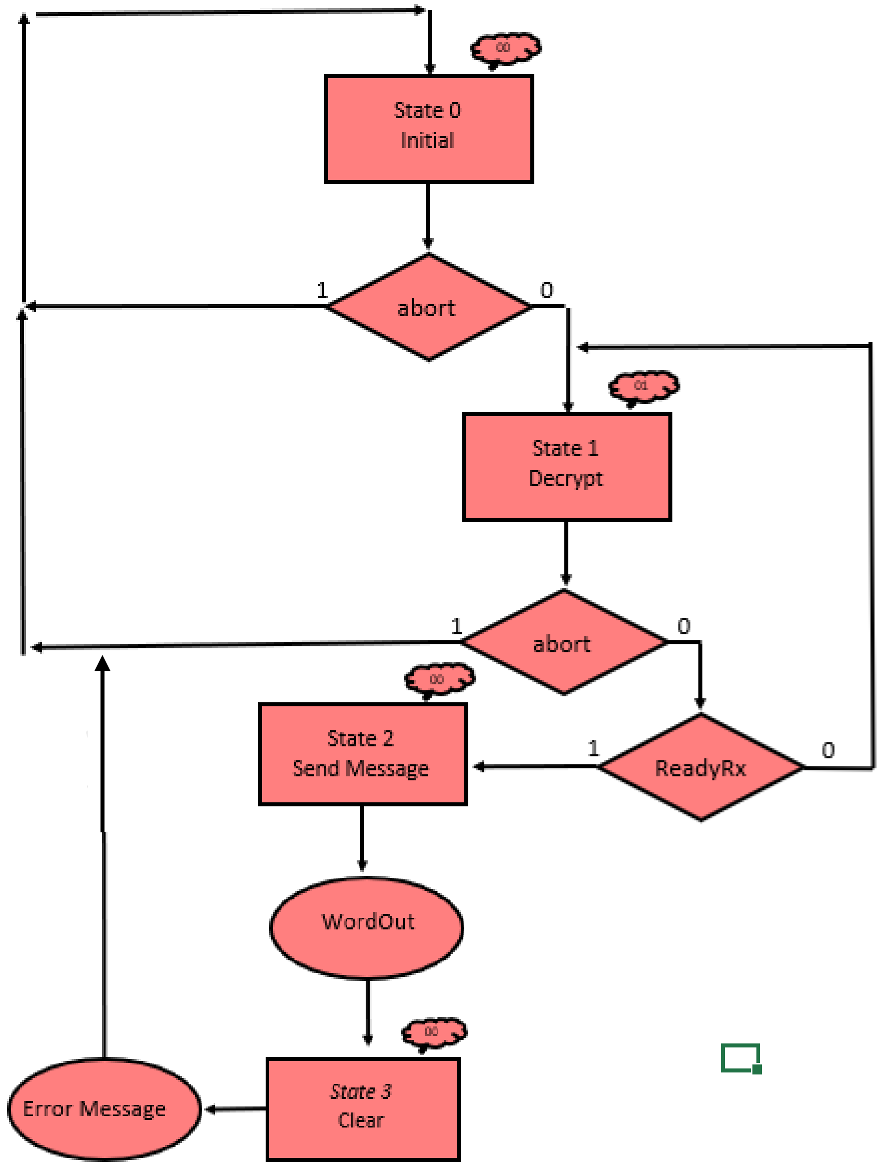
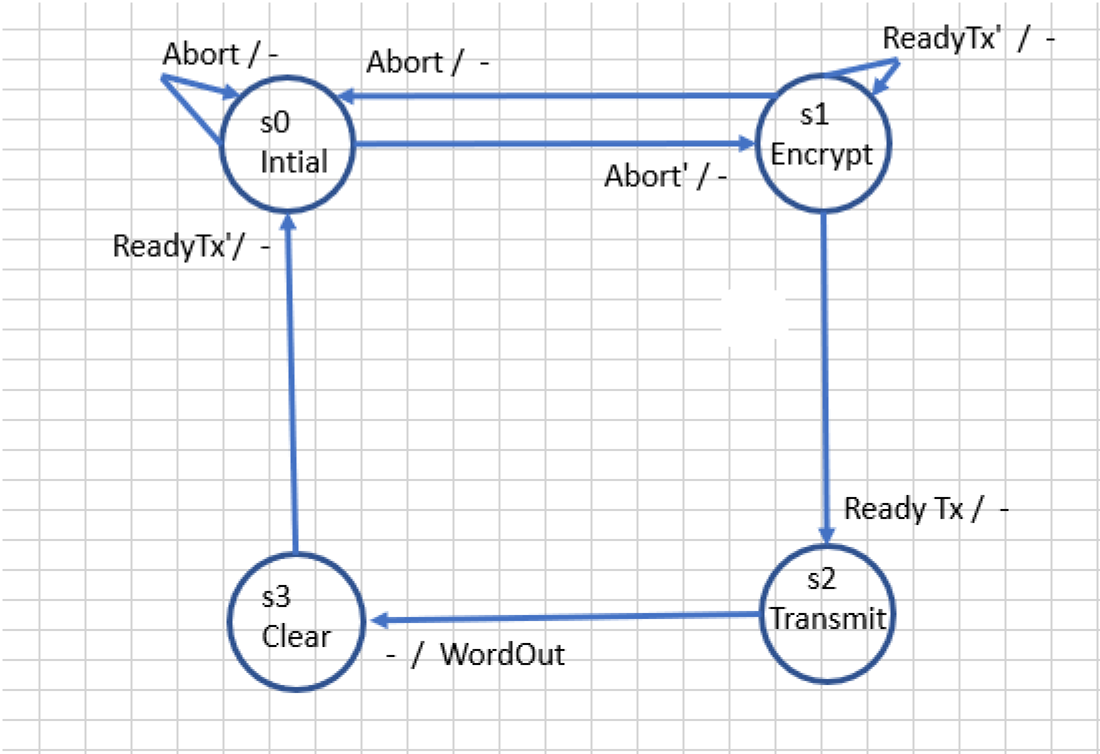


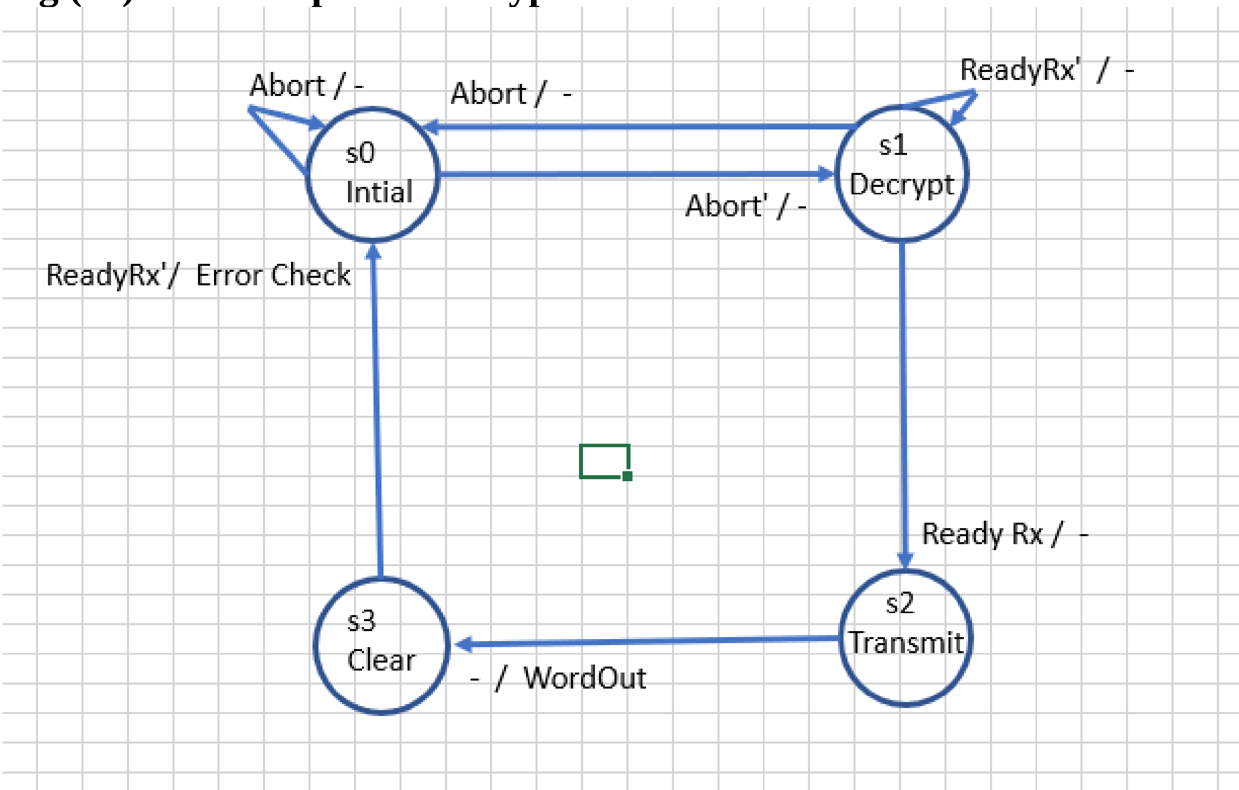
Fig (11) State graph for encryption and state table:



state table

	NS		
PS	x=0	x =1	output
S0	S1	S0	-
S1	S1	S2	-
S2	-	S3	WordOut
S3	S0	-	-

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



	x=0	x=1	z
s0	s1	s0	-
s1	s1	s2	-
s2	s3	s3	wordOUt
s3	s0	s0	ErrorCheck

4. 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.

5. Simulation Results with annotation:

Fig (5) encryption input (raw):

```
                  TIME(ns)       index    value(encrypted)
tranmission to follow:
                     30             1   1000011111101110
                     50             2   1000011111000101
                     70             3   1000100000001101
                     90             4   1000100000000110
                    110             5   1000100000011011
                    130             6   1000100000001010
                    150             7   1000011111000101
                    170             8   1000011111011000
                    190             9   1000011111000101
```

Fig (6) encryption output(encrypted):

```
                   3850           192   1000011111000101
                   3870           193   1000011111110001
                   3890           194   1000011111111010
                   3910           195   1000011111101000
                   3930           196   1000011111110000
                   3950           197   1000011110100101
                   3970           198   1000011110100101
                   3990           199   1000011110100101
                   4010           200   1000011110100101
                      { reset, ready for next TX }
tranmission to follow:
```

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

```
                3850        192    1000011111000101
                3870        193    1000011111110001
                3890        194    1000011111111010
                3910        195    1000011111101000
                3930        196    1000011111110000
                3950        197    1000011110100101
                3970        198    1000011110100101
                3990        199    1000011110100101
                4010        200    1000011110100101
                   { reset, ready for next TX }
tranmission to follow:
```

Fig (8) decrytpor output with verification

```
message to follow:
 I have 3 tons of gold that I need you to pick up.
 Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
 Bring an airplane, some tools, and 5lbs of potatoes.
 Pay Dr. Greg when done.
 GOOD LUCK
            { reset, ready to decrypt next RX }


SECOND TRANSMISSION, MAKE SURE IT SAME AS FIRST :


 I have 3 tons of gold that I need you to pick up.
 Be at Melli Bank 35.686 51.393 no later than 202005271300Z.
 Bring an airplane, some tools, and 5lbs of potatoes.
 Pay Dr. Greg when done.
 GOOD LUCK

message to follow:
```

6. 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.

7. 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.

8. 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.

9. 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