ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

HOMEWORK NO	:	3
EXPERIMENT DATE	:	22.05.2022
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G8

GROUP MEMBERS:

150180089	:	Harun Çifci
150200705	:	Helin Aslı Aksoy

SPRING 2021

Contents

1 INTRODUCTION [10 points]

In this experiment we are asked to implement three important cryptography applications in the history of crypthography. First we have implemented Caesar Cipher named after Roman Republic general and dictator Julius Caesar. Caesar Cipher was one of the first examples of cryptography in history. Julius Caesar used this encryption technique to communicate with his generals during war. The second technique we are asked to implement was Vigenere Cipher used by Confederate Army. Unlike Caesar Cipher, the Cigenere Chiper encrypts messages using a key. Both partiesi, the sender and receiver musth have the same key for conducting communication. Finally we are asked to implement Enigma machine used by German Army in the World War 2 and we are asked to realize a sample communication environment. The Enigma machine was used to encrypt and decrypt secret messages. According to experts, the defeat of the German Army was accelerated by 2 years thanks to the work of Ultra group, which deciphered the Enigma and other encryption machines.

2 MATERIALS AND METHODS [40 points]

2.1 Part 1

In this part we have designed 4 helper modules. Detailed information is given below.

- CharDecoder Module: It transforms ASCII codes of to decoded binary character.It takes 8-bit input which is the ASCII code, gives 26-bit output as decoded char.
  - char (8-bit input): ASCII code of the character.
  - decodedChar (26-bit output): Decoded output of the character.
- CharEncoder Module: It transforms the binary decoded version of the char to ASCII code. It takes 26-bit input, gives 8-bit output.
  - decodedChar (26-bit input): Decoded input of the character.
  - char (8-bit output): ASCII code of the character.
- CircularRightShift Module: It takes 26-bit input data and 5-bit shiftAmount input, gives 26-bit output. The output is data shifted to the right 'shiftAmount' times.
  - data (26-bit input): Input data.
  - shiftAmount (5-bit input): Shift amount.
  - out (26-bit output): Data shifted to the right 'shiftAmount' times.

- CircularLeftShift Module: It takes 26-bit input data and 5-bit shiftAmount input, gives 26-bit output. The output is data shifted to the left ‘shiftAmount’ times.
  - data (26-bit input): Input Data.
  - shiftAmount (5-bit input): Shift amount.
  - out (26-bit output): Data shifted to the left ‘shiftAmount’ times

2.2 Part 2

In this part, we have implemented Caesar Cipher Encryption and Decryption modules. Then we have0 created a Caesar Module to show encrypted and decrypted messages. Figure 1 shows the encryption technique of Caesar Cipher when the shift count is 3.

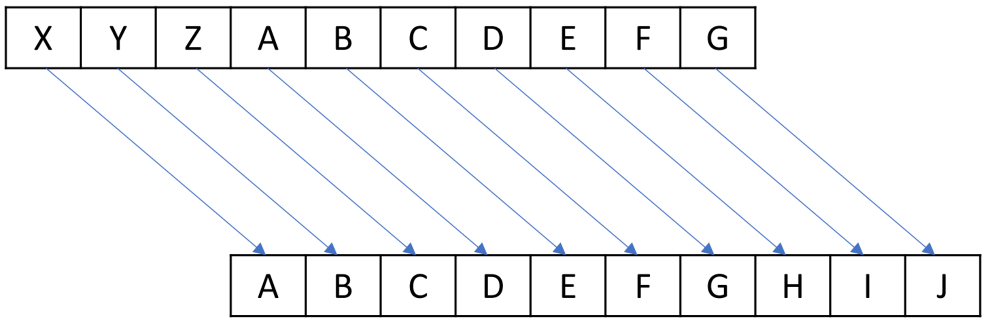


Figure 1: Caesar Encryption

- CaesarEncryption Module: It works like a shifter and encrypt the char using Caesar Cipher technique.
- CaesarDecryption Module: It works like a shifter and decrypt the encrypted char using Caesar Cipher technique.
- CaesarEnvironment Module: It is a module that contains CaesarEncryption and CaesarDecryption modules inside.

To implement these modules we have used helper modules that we have implemented at the previous part. The abstract diagram of the Caesar Encryption and Decryption Process is as follows.

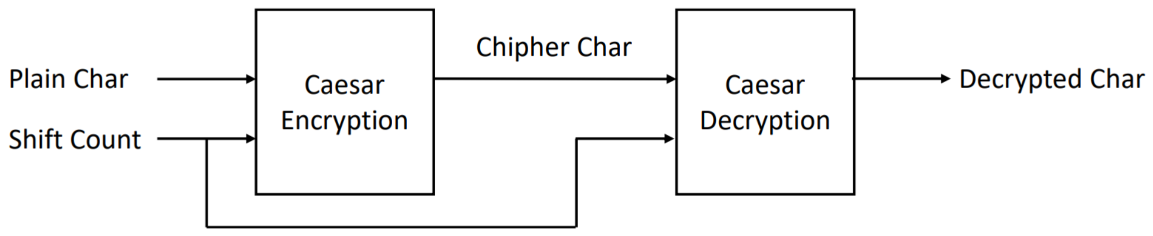


Figure 2: Caesar Encryption

2.3 Part 3

In this part, we have implemented the Vigenere Cipher Encryption and Decryption modules. Then, we have created a Vigenere Module to show encrypted and decrypted messages. Vigenere Cipher uses the same key message for encryption and decryption processes. Equation 1 and Equation 2 show encryption and decryption calculations of the Vigenere Cipher, respectively. Vigenere Cipher's key message is "KADIROZLEM" and the plain text is "ISTANBULTECHNICAL".

$$C_i = (P_i + K_i) \quad mod \quad 26$$

Figure 3: Cipher Char Equation

where Ci is the cipher char, Pi is the plain char, and Ki is the key char.

$$D_i = (C_i - K_i) \quad mod \quad 26$$

Figure 4: Decrypted Char Equation

where Di is the decrypted char, Ci is the cipher char, and Ki is the key char.
The modules that we have implemented for the Vigenere Chipher as given below.

- VigenereEncryption Module: It encrypts the char using Vigen‘ere Cipher technique. It has 2 char input, one of them is plain char which is going to be encrypted. The other one is key char. It has 1 bit clock and load input that allows registers to work. It gives 8-bit encrypted char output.

- VigenereDecryption Module: It decrypts the char using Vigenere Cipher technique. It has 2 char input, one of them is plain char which is encrypted. The other one is key char. It has 1 bit clock and load input that allows registers to work. It gives 8-bit decrypted char output.

- VigenereEnvironment Module: It is a module that contains VigenereEncryption and VigenereDecryption modules inside.

2.4 Part 4

In this part, we have tried to implement the Enigma machine. The Enigma machine can both encrypt and decrypt a message. For secure communication, receiver and transmitter must have Enigma machines which are configured and initiated identically. The rotors in the Enigma machine rotate every time a new input character is given and this rotation changes the configuration of the machine entirely. This way, a character is decrypted differently every time it is given as an input.

3 RESULTS [15 points]

3.1 Part 1

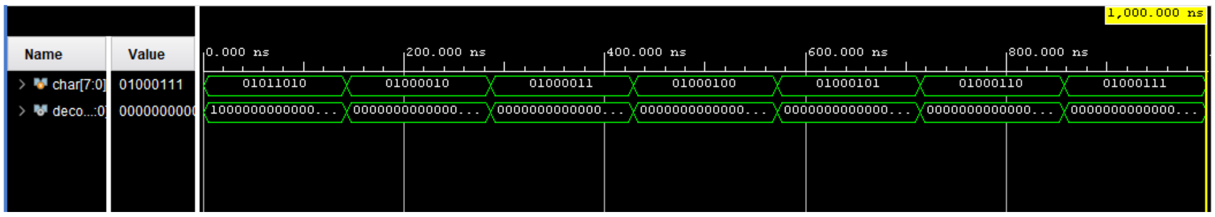
```
module CharDecoder_simulation();
   reg [7:0] char;
   wire [25:0] decodedChar;

   CharDecoder uut(char, decodedChar);

   initial
   begin
       char = 8'd90; #142.85;
       char = 8'd66; #142.85;
       char = 8'd67; #142.85;
       char = 8'd68; #142.85;
       char = 8'd69; #142.85;
       char = 8'd70; #142.85;
       char = 8'd71; #142.85;
   end

endmodule
```

Listing 1: CharDecoder Module



Name	Value	0.000 ns			200.000 ns		400.000 ns		600.000 ns		800.000 ns		1,000.000 ns
char[7:0]	01000111	01011010	01000010	01000011	01000100	01000101	01000110	01000111					
deco...:0]	0000000000	1000000000000...	0000000000000...	0000000000000...	0000000000000...	0000000000000...	0000000000000...	0000000000000...					

Figure 5: CharDecoder Simulation

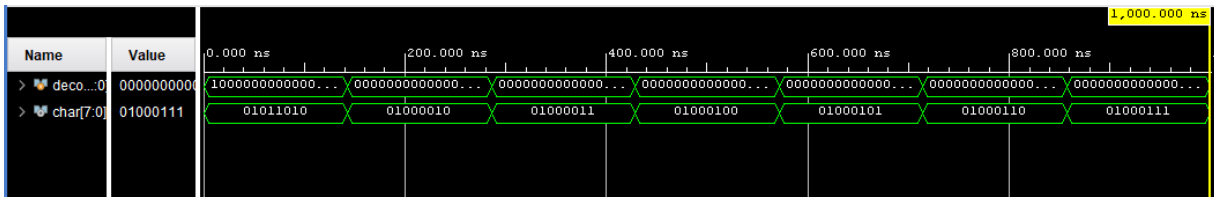
```
module CharEncoder_simulation();
   reg [25:0] decodedChar;
   wire [7:0] char;

   CharEncoder uut(decodedChar, char);

   initial
   begin
        decodedChar = 26'b10000000000000000000000000; #142.85;
        decodedChar = 26'b00000000000000000000000010; #142.85;
        decodedChar = 26'b00000000000000000000000100; #142.85;
        decodedChar = 26'b00000000000000000000001000; #142.85;
        decodedChar = 26'b00000000000000000000010000; #142.85;
        decodedChar = 26'b00000000000000000000100000; #142.85;
        decodedChar = 26'b00000000000000000001000000; #142.85;
   end

endmodule
```

Listing 2: CharEncoder Module



Name	Value							
deco...:0]	0000000000	1000000000000...	0000000000000...	0000000000000...	0000000000000...	0000000000000...	0000000000000...	0000000000000...
char[7:0]	01000111	01011010	01000010	01000011	01000100	01000101	01000110	01000111

Figure 6: CharEncoder Simulation

```
module CircularLeftShiftSimulation();
   reg [25:0]data;
   reg [4:0]shiftAmount;
   wire [25:0]out;
   CircularLeftShift uut(shiftAmount, data, out);

   initial begin
```

```
        data = 26'b10000000000000000000000000; shiftAmount = 5'b00001; #250;
        data = 26'b01000000000000000000000000; shiftAmount = 5'b00001; #250;
        data = 26'b00100000000000000000000000; shiftAmount = 5'b00001; #250;
        data = 26'b00010000000000000000000000; shiftAmount = 5'b00001; #250;
   end
endmodule
```

Listing 3: LeftShift Module

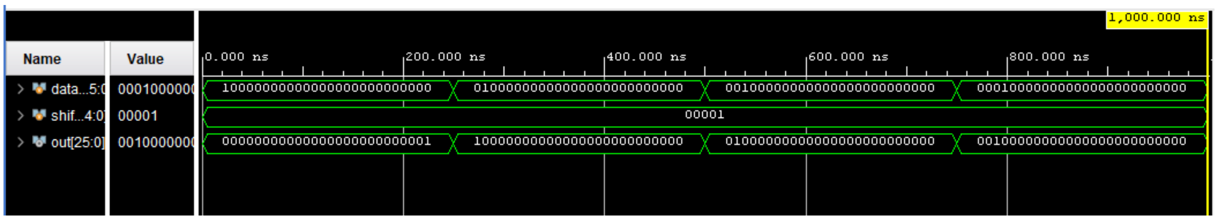


Figure 7: CircularLeftShift Module Simulation

```
module CircularRightShiftSimulation();
   reg [25:0]data;
   reg [4:0]shiftAmount;
   wire [25:0]out;
   CircularRightShift uut(shiftAmount,data, out);

   initial begin
        data = 26'b10000000000000000000000000; shiftAmount = 5'b00001; #250;
        data = 26'b01000000000000000000000000; shiftAmount = 5'b00001; #250;
        data = 26'b00100000000000000000000000; shiftAmount = 5'b00001; #250;
        data = 26'b00010000000000000000000000; shiftAmount = 5'b00001; #250;
   end
endmodule
```

Listing 4: RightShift Module

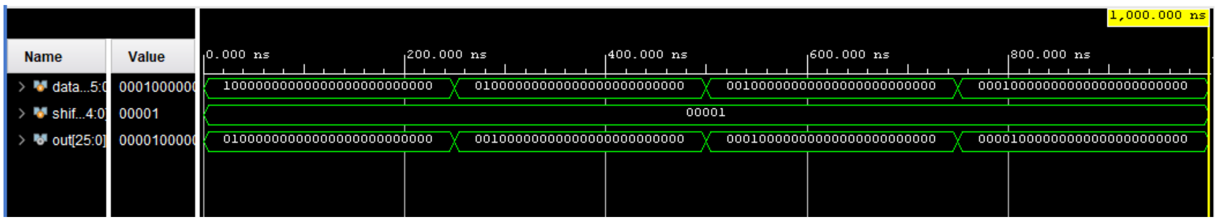


Figure 8: CircularRightShift Module Simulation

3.2 Part 2

```
module CaesarCipherEncryption_simulation();
    reg [7:0] plainChar;
    reg [4:0] shiftCount;
    wire [7:0] chipherChar;

   CaesarCipherEncryption uut(plainChar, shiftCount, chipherChar);

   initial
   begin
        plainChar = 8'd72; shiftCount = 0; #200;
        plainChar = 8'd65; shiftCount = 4; #200;
        plainChar = 8'd82; shiftCount = 20; #200;
        plainChar = 8'd85; shiftCount = 14; #200;
        plainChar = 8'd78; shiftCount = 0; #200;
   end

endmodule
```

Listing 5: Decryption Module

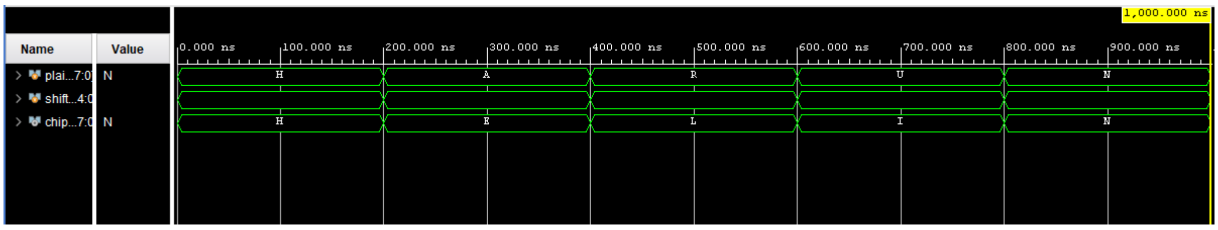


Figure 9: CaesarCipherEncryption Module Simulation

```
module CaesarCipherDecryption_simulation();
    reg [7:0] chipherChar;
    reg [4:0] shiftCount;
    wire [7:0] decryptedChar;

   CaesarCipherDecryption uut(chipherChar, shiftCount, decryptedChar);
```

```verilog

   initial
   begin
        chipherChar = 8'd72; shiftCount = 0; #200;
        chipherChar = 8'd69; shiftCount = 4; #200;
        chipherChar = 8'd76; shiftCount = 20; #200;
        chipherChar = 8'd73; shiftCount = 14; #200;
        chipherChar = 8'd78; shiftCount = 0; #200;
   end

endmodule
```

Listing 6: Encryption Module

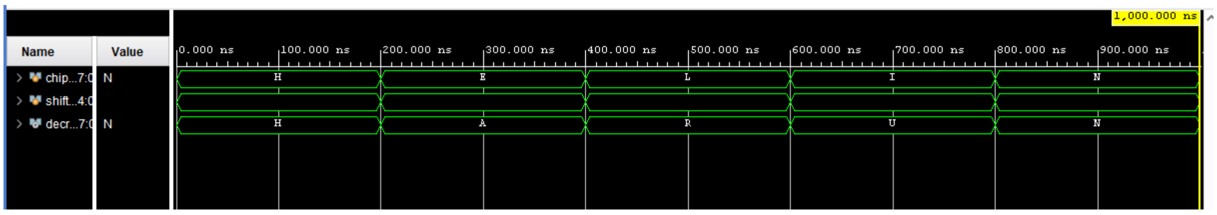


Figure 10: CaesarCipherDecryption Module Simulation

```verilog
module CaesarEnvironment_simulation();
    reg [7:0] plainChar;
    reg [4:0] shiftCount;
    wire [7:0] chipherChar;
    wire [7:0] decryptedChar;

   CaesarEnvironment uut(plainChar, shiftCount, chipherChar, decryptedChar);

   initial
   begin
        plainChar = 8'd75; shiftCount = 3; #200;
        plainChar = 8'd65; shiftCount = 1; #200;
        plainChar = 8'd68; shiftCount = 15; #200;
        plainChar = 8'd73; shiftCount = 4; #200;
        plainChar = 8'd82; shiftCount = 3; #200;
   end

endmodule
```

Listing 7: Environment Module

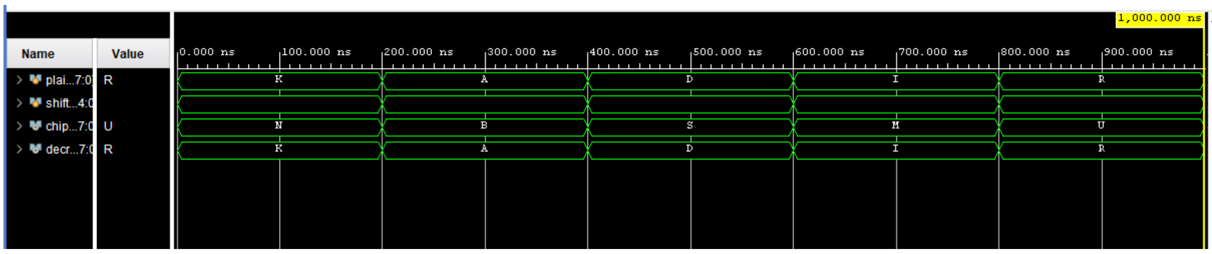


Figure 11: CaesarEnvironment Module Simulation

3.3 Part 3

```
module Vigenere_Cipher_Decryption_simulation();
   reg [7:0] chipherChar;
   reg [79:0] keyInput;
   reg load;
   reg CLK;
   wire [7:0] decryptedChar;
   Vigenere_Cipher_Decryption uut(chipherChar, keyInput, load, CLK,
   decryptedChar);

   initial begin
       CLK = 0; load = 1; chipherChar = "O"; keyInput = "KADIROZLEM";
   #100;
                 load = 0; chipherChar = "R"; #100;
                 load = 0; chipherChar = "L"; #100;
                 load = 0; chipherChar = "S"; #100;
                 load = 0; chipherChar = "P"; #100;
                 load = 0; chipherChar = "S"; #100;
                 load = 0; chipherChar = "Q"; #100;
                 load = 0; chipherChar = "T"; #100;
                 load = 0; chipherChar = "Q"; #100;
                 load = 0; chipherChar = "Y"; #100;
   end

   always begin
       CLK <= ~CLK; #50;
   end

endmodule
```

Listing 8: Decryption Module

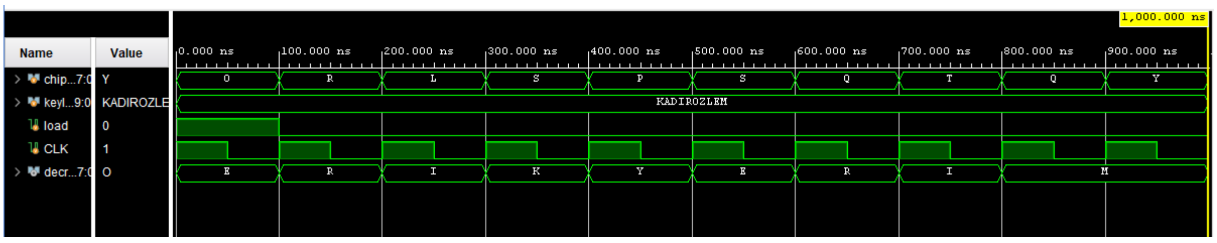


Figure 12: Vigenere Decryption Simulation

```verilog
module Vigenere_Cipher_Encryption_simulation();
   reg [7:0] plainChar;
   reg [79:0] keyInput;
   reg L;
   reg Clock;
   wire [7:0] chipherChar;
   Vigenere_Cipher_Encryption uut(plainChar, keyInput, L, Clock,
   chipherChar);

   initial begin
       Clock = 0; L = 1; plainChar = "E"; keyInput = "KADIROZLEM"; #100;
                L = 0; plainChar = "R";  #100;
                L = 0; plainChar = "I";  #100;
                L = 0; plainChar = "K";  #100;
                L = 0; plainChar = "Y";  #100;
                L = 0; plainChar = "E";  #100;
                L = 0; plainChar = "R";  #100;
                L = 0; plainChar = "I";  #100;
                L = 0; plainChar = "M";  #100;
                L = 0; plainChar = "M";  #100;

   end

   always begin
       Clock <= ~Clock; #50;
   end

endmodule
```

Listing 9: Encryption Module

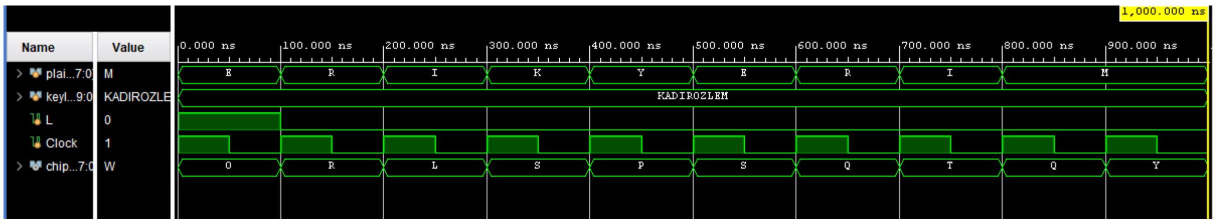


Figure 13: Vigenere Encryption Simulation

```
module VigenereEnvironment_simulation();
   reg [7:0] plainChar;
   reg [79:0] keyInput;
   reg load;
   reg CLK;
   reg CLK2;
   wire [7:0] chipherChar;
   wire [7:0] decryptedChar;
   VigenereEnvironment_Module uut(plainChar, keyInput, load, CLK, CLK2 , chipherChar, decryptedChar);

   initial begin
       CLK = 0; CLK2 = 0; load = 1; plainChar = "P"; keyInput = "DIGITALLAB"; #100;
                   load = 0; plainChar = "A"; #100;
                   load = 0; plainChar = "L"; #100;
                   load = 0; plainChar = "Y"; #100;
                   load = 0; plainChar = "A"; #100;
                   load = 0; plainChar = "C"; #100;
                   load = 0; plainChar = "O"; #100;
                   load = 0; plainChar = "L"; #100;
                   load = 0; plainChar = "U"; #100;
                   load = 0; plainChar = "K"; #100;

   end

   always begin

       CLK <= ~CLK; #50;
   end
   always begin
       #0.4
       CLK2 <= ~CLK2; #49.6;
   end

```

```
endmodule
```

Listing 10: Environment Module

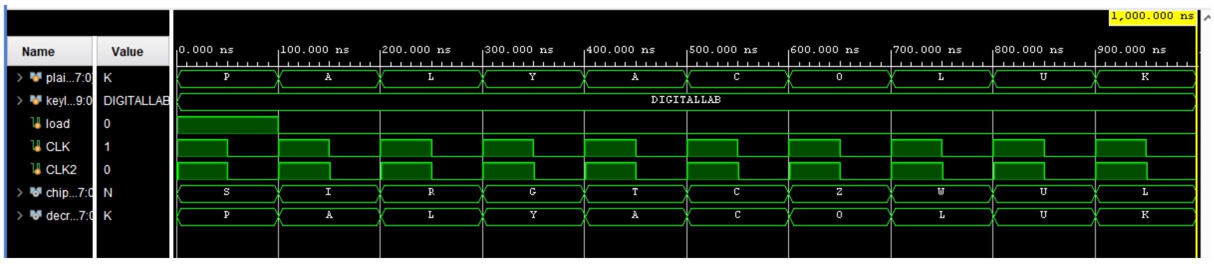


Figure 14: Vigenere Environment Simulation

4 DISCUSSION [25 points]

In the first part of experiment, we are asked to build some helper modules for part 2 such as; encoder, decoder, right shifter and left shifter. Encoder transforms ASCII to encoded binary char. Decoder works vice versa. Shifters are used to map the characters to another character as Caesar Method works like that.

In the second part of experiment, we have implemented Caesar Cipher with the help of modules we have created in previous part. We have implemented 3 modules in this part such as: Caesar Encryption, Caesar Decryption and Caesar Environment. In Caesar Encryption module, we have 2 inputs plainChar and shiftCount and 1 output chipherChar. First, we have used Decoder module and decode plainChar input. Then, we have used Right Shift module and shift our decoded char to right. Lastly, we have encoded our decoded char with Encoder module. Caesar Decryption module has the same structure with Caesar Encryption module but instead of shifting right between decoder and encoder we are shifting to left. In the Caesar Environment we have used Encryption and Decryption Modules. We have taken a plainChar and shiftCound as input and gave chipherChar and decryptedChar output.

In the third part, We have implemented Vigenere Module. Vigenere Cipher uses the same key message for encryption and decryption processes. We have implemented 3 modules: Vigenere Encryption, Vigenere Decryption and Vigenere Environment. In Vigenere Encryption module, we have taken 4 inputs plainChar, keyInput, LOAD and CLK and gave output chipherChar. We have used “Ci = (Pi + Ki) mod 26” equation to calculate chipherChar if LOAD is 0. If LOAD is 1, it loads the keyInput into keyRegister. Vigenere Decryption module has same structure with Encryption but it uses a different equation to calculate decryptedChar ,“Di = (Ci Ki) mod 26”. In Vigenere Environment we have used both Encryption and Decryption modules consecutively and we have taken

4 inputs such as Load, CLK, plainChar and keyInput and gave 2 outputs, chipherChar and decryptedChar.

5 CONCLUSION [10 points]

Since it is an interesting working area we have thought about the topic and made some searches on it. We have learned many different Cryptology methods other than Caesar, Vigenere. Also story of Enigma and it's connection with computer engineering was impressive. Unfortunately, due to lack of time we could not make the Enigma but we have learned workflow of it.