ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E

DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

HOMEWORK NO	:	3
HOMEWORK DUE	:	22.05.2022
LAB SESSION	:	FRIDAY - 14.00
GROUP NO	:	G25

GROUP MEMBERS:

150210710 : FATİH BASKIN

150190102 : ELVAN TEKE

SPRING 2022

Contents

1 INTRODUCTION

In this homework, we have created ASCII to binary decoder/encoder modules and circular left-right shift modules. By using them, we implemented some historical encryption techniques. In the old days, either people encrypted or deciphered them by hand or they had used electro-mechanical contraptions such as Enigma machines.

2 HOMEWORK

2.1 Helper Modules

In this part, we have been asked to design 4 helper modules.

2.1.1 CharDecoder Module

This module transforms A-Z chars to binary decoded version. To do that, we used switch-case. This module has an 8-bit input "char" and a 26-bit output "decodedChar".

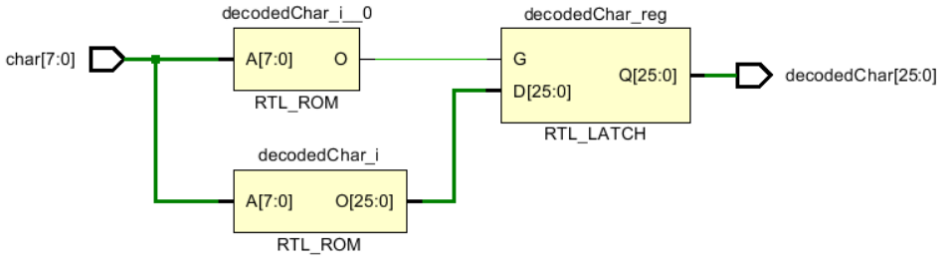

Figure 1: RTL design of CharDecoder module.

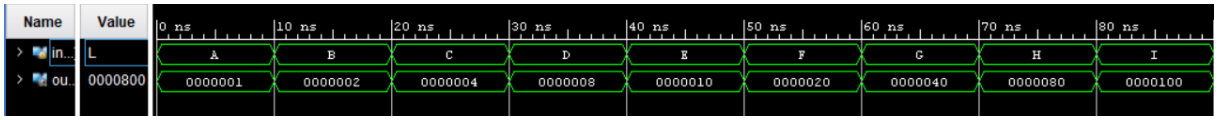

Figure 2: Simulation result of CharDecoder module.

2.1.2 CharEncoder Module

This module transforms binary decoded version of A-Z chars to ASCII codes. Input of this module is 26-bit "decodedChar", and the output is 8-bit "char".

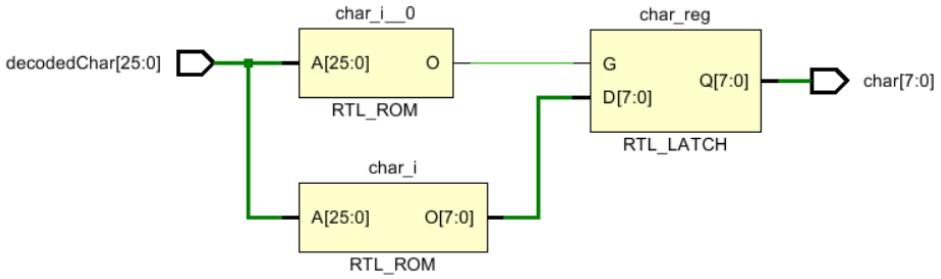

Figure 3: RTL design of CharEncoder module.

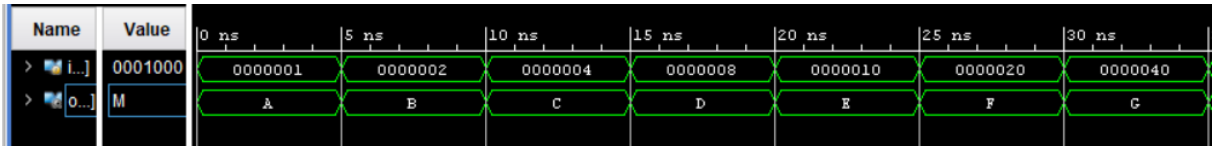

Figure 4: Simulation result of CharEncoder module.

2.1.3 CircularRightShift Module

This module right shifts the input. Shift count is controlled by the shiftAmount input.

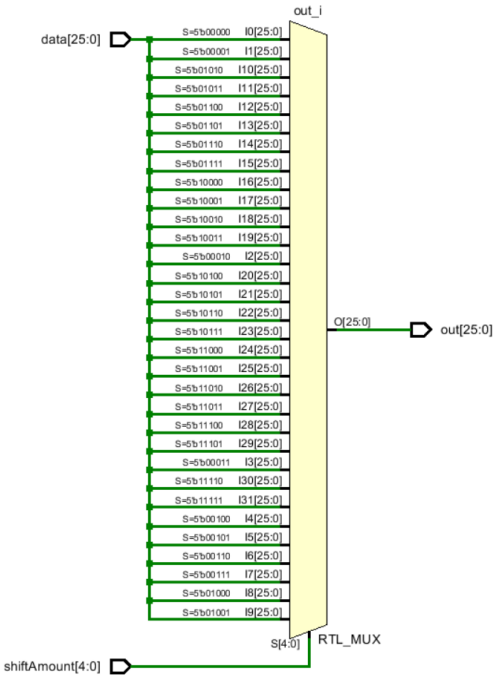

Figure 5: RTL design of CircularRightShift module.

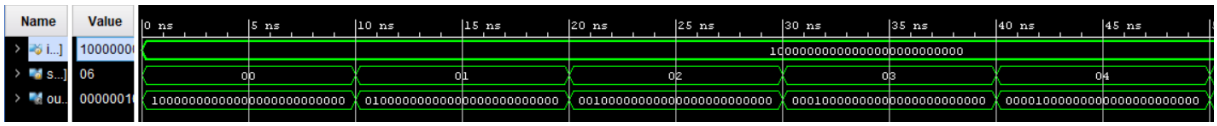

Figure 6: Simulation result of CircularRightShift module.

2.1.4 CircularLeftShift Module

This module left shifts the input. Shift count is controlled by the shiftAmount input.

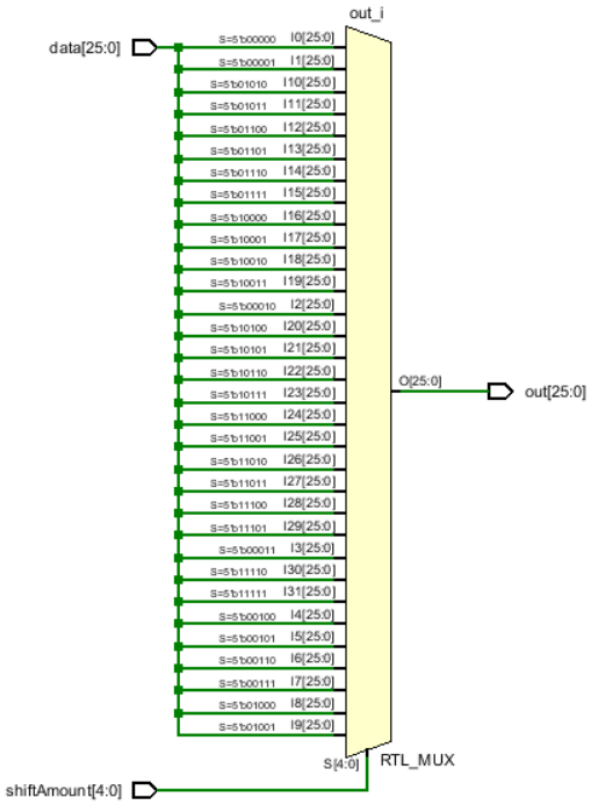

Figure 7: RTL design of CircularLeftShift module.

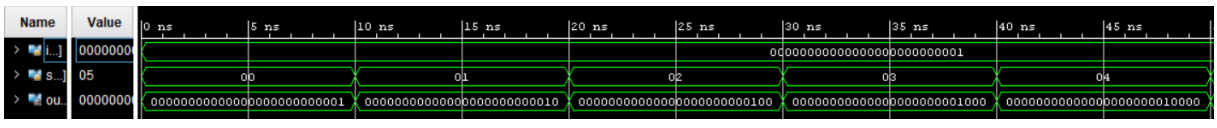

Figure 8: Simulation result of CircularLeftShift module.

2.2 Caesar Cipher

In this part, we implemented Caesar Cipher Encryption and Decryption modules, and Caesar Environment Module. We used first two modules in the Caesar Environment Module to show encrypted and decrypted messages.

2.2.1 CaesarEncryption Module

This module encrypt the input character using Caesar Cipher technique. It takes an ASCII code as input and shifts it by shiftCount.

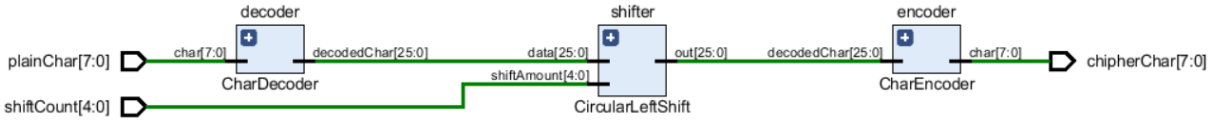

Figure 9: RTL design of CaesarEncryption module.

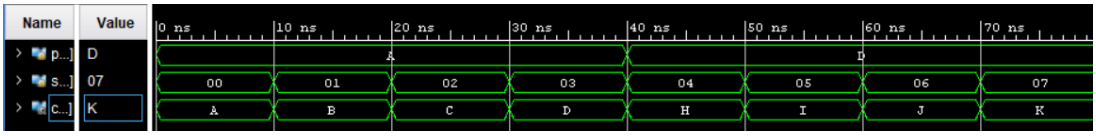

Figure 10: Simulation result of CaesarEncryption module.

2.2.2 CaesarDecryption Module

This module decrypt the encrypted character using Caesar Cipher technique.

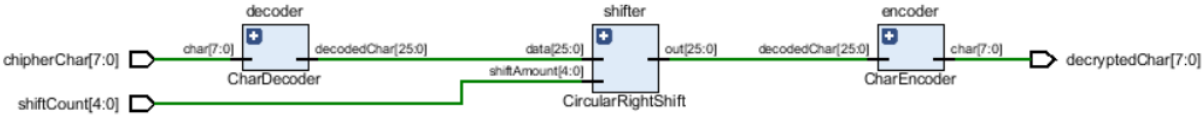

Figure 11: RTL design of CaesarDecryption module.

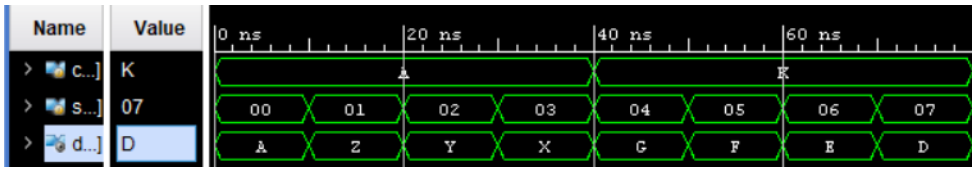

Figure 12: Simulation result of CaesarDecryption module.

2.2.3 CaesarEnvironment Module

This module shows encryption and decryption processes of a character input.

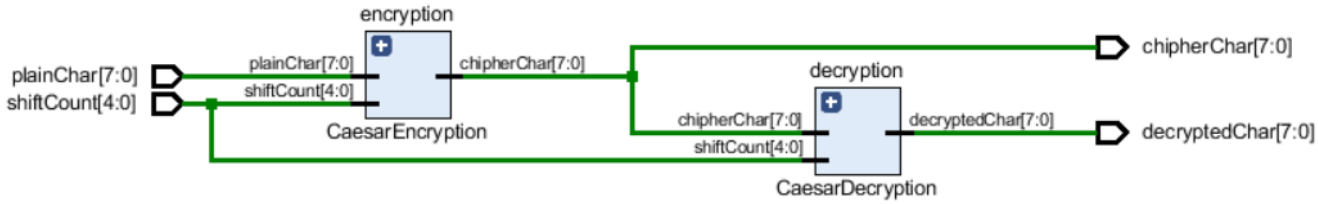

Figure 13: RTL design of CaesarEnvironment module.

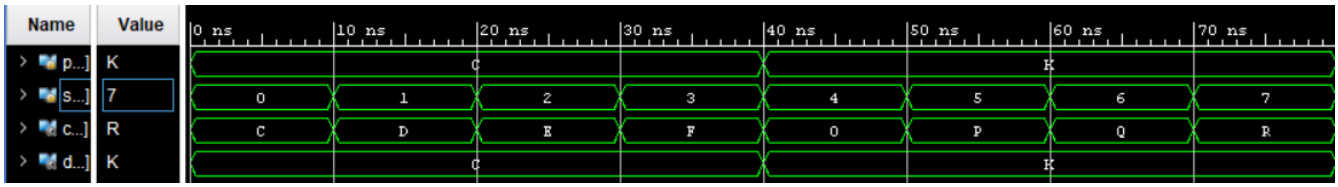

Figure 14: Simulation result of CaesarEnvironment module.

2.3 Vigenere Cipher

In this part, we implemented Vigenere Cipher Encryption and Decryption modules, and Vigenere Environment Module. We used first two modules in the Caesar Environment Module to show encrypted and decrypted messages. Same key message is used in both encryption and decryption modules.

2.3.1 VigenereEncryption Module

This module is designed to encrypt the character using Vigenere Cipher technique.
Encryption formula is: $C_i = (P_i + K_i) \mod 26$

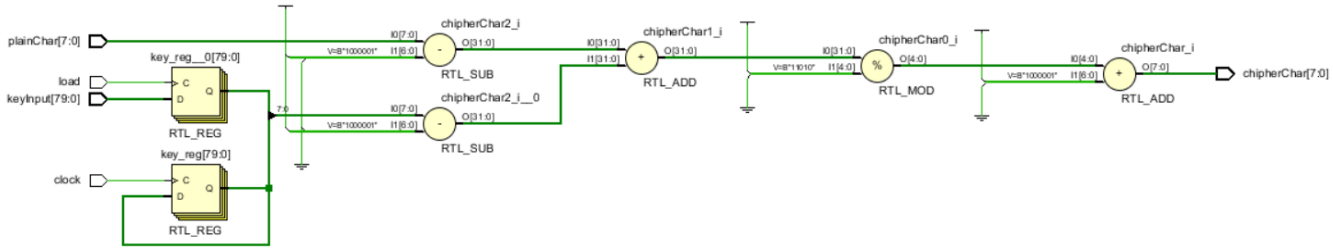

Figure 15: RTL design of VigenereEncryption module.

2.3.2 VigenereDecryption Module

This module is designed to decrypt the encrypted character using Vigenere Cipher technique.
Decryption formula is: $D_i = (C_i - K_i) \mod 26$

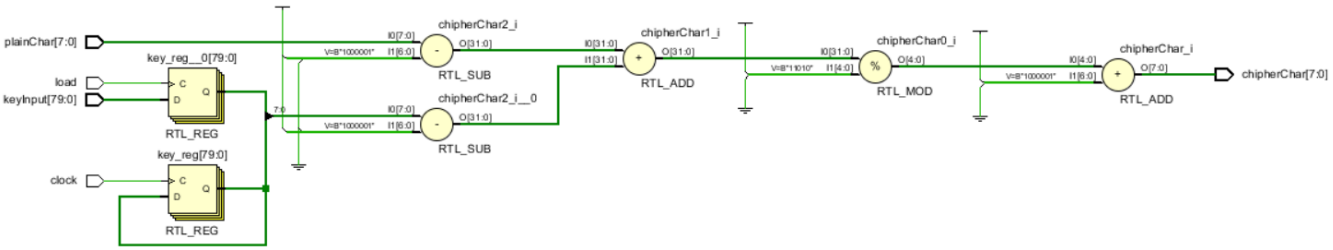

Figure 16: RTL design of VigenereDecryption module.

2.3.3 VigenereEnvironment Module

This module shows encryption and decryption processes of a character input.

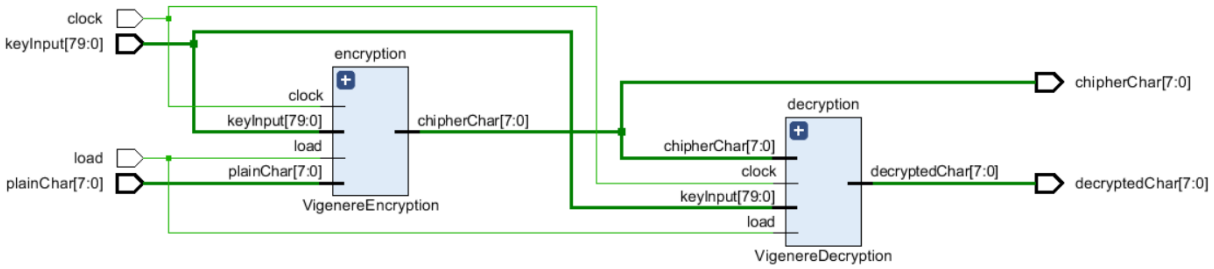

Figure 17: RTL design of VigenereEnvironment module.

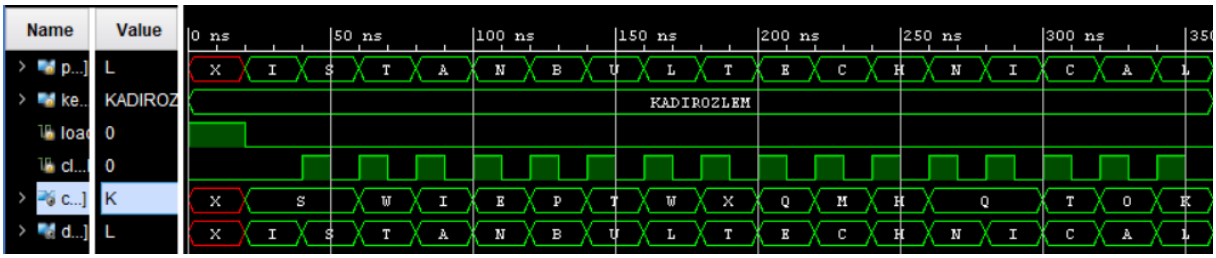

Figure 18: Simulation result of VigenereEnvironment module.

2.4 Enigma

In World War Two, German Army used a novel encryption method to transmit their messages safely. This encrypted messages has created safe communication environment in German Army. Until Alan Turing created a machine to decode those messages, German Army communicated under utter secrecy.

Enigma machine consists of five pieces, a plug-board, 3 rotors that can turn and therefore change the character, making same letters ciphered differently in different times and a reflector that reflects same ciphered message and further ciphers it, passing message under rotors and plug-board again. Also each component has different wiring. This further complicates the ciphering, making deciphering harder.

2.4.1 Enigma Plug-Board

This is a simple contraption that has forward and backward wiring sets and mixes letters with each other. plug-board receives decoded ASCII characters (26-bit input stream) then scrambles them. Here is a table for the supposed wiring:

Forward Output Backward Input	Char Input Char Output	Forward Output Backward Input	Char Input Char Output
0 (A)	4 (E)	13 (N)	22 (W)
1 (B)	10 (K)	14 (O)	24 (Y)
2 (C)	12 (M)	15 (P)	7 (H)
3 (D)	5 (F)	16 (Q)	23 (X)
4 (E)	11 (L)	17 (R)	20 (U)
5 (F)	6 (G)	18 (S)	18 (S)
6 (G)	3 (D)	19 (T)	15 (P)
7 (H)	16 (Q)	20 (U)	0 (A)
8 (I)	21 (V)	21 (V)	8 (I)
9 (J)	25 (Z)	22 (W)	1 (B)
10 (K)	13 (N)	23 (X)	17 (R)
11 (L)	19 (T)	24 (Y)	2 (C)
12 (M)	14 (O)	25 (Z)	9 (J)

Table 1: Scrambling Letters for Plug-Board

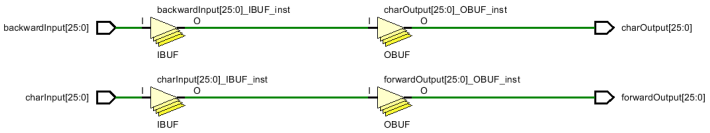

Figure 19: Plug-Board's design.

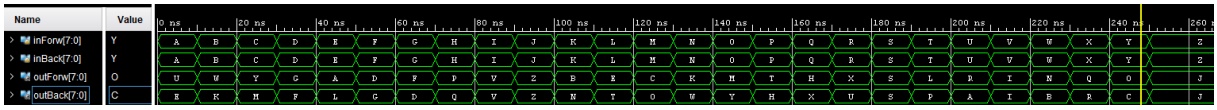

Figure 20: Plug-Board's simulation, this simulation is easier to interpret, we included encoders and decoders for ease of examination.

2.4.2 Enigma Counter

In Enigma machines, rotor1 (fast rotor) rotates one bit at each clock cycle, rotor2 (normal rotor) rotates one bit for every 26 clock cycles, and rotor3 (slow rotor) rotates one bit for every 676 clock cycles. Also they can set to a manual location. To achieve this, we implemented a counter which can be loaded independently from clock cycle, also they have a internal counter where they give a clock-out signal when they reach 26 rotations and sets back to 0.

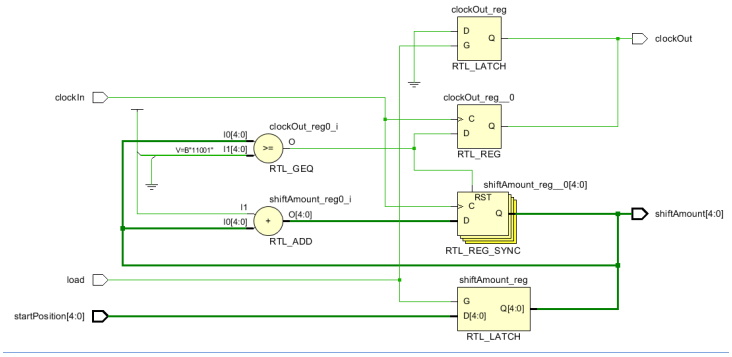

Figure 21: Enigma Counter's design.

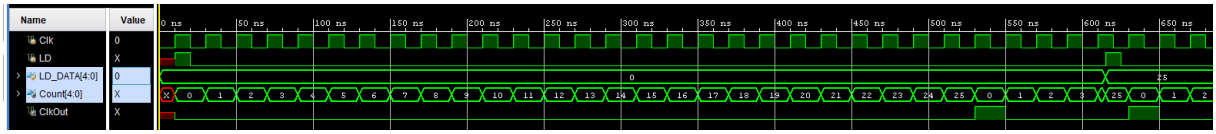

Figure 22: Enigma Counter's simulation.

2.4.3 Enigma Rotor 1

Rotors turn with each clock cycle, also they scramble the letters. With each turn, first they shift right, then scramble, then shift left. Scramble table is given below:

Forward Output Backward Input	Forward Input Backward Output	Forward Output Backward Input	Forward Input Backward Output
0 (A)	7 (H)	13 (N)	3 (D)
1 (B)	12 (M)	14 (O)	14 (O)
2 (C)	21 (V)	15 (P)	13 (N)
3 (D)	17 (R)	16 (Q)	11 (L)
4 (E)	0 (A)	17 (R)	8 (I)
5 (F)	2 (C)	18 (S)	4 (E)
6 (G)	22 (W)	19 (T)	10 (K)
7 (H)	20 (U)	20 (U)	6 (G)
8 (I)	23 (X)	21 (V)	5 (F)
9 (J)	18 (S)	22 (W)	19 (T)
10 (K)	9 (J)	23 (X)	16 (Q)
11 (L)	25 (Z)	24 (Y)	24 (Y)
12 (M)	15 (P)	25 (Z)	1 (B)

Table 2: Scramble Table for Rotor 1

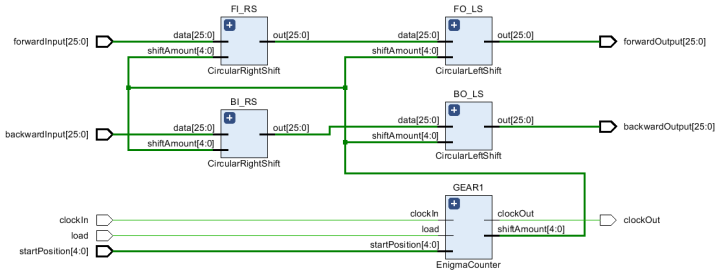

Figure 23: Implementation of Rotor 1

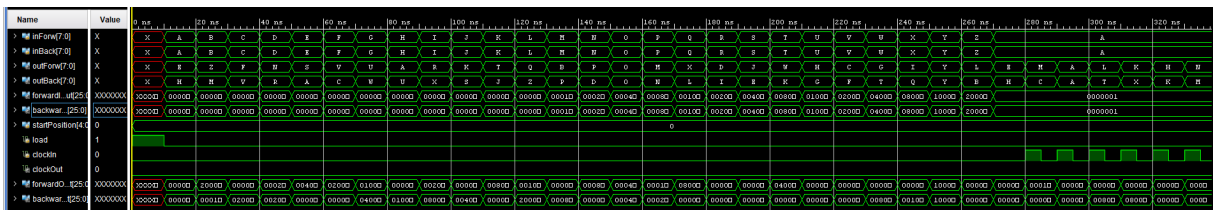

Figure 24: Simulation of Rotor 1 (decoded and encoded I/O)

2.4.4 Enigma Rotor 2

Forward Output	Forward Input	Forward Output	Forward Input
Backward Input	Backward Output	Backward Input	Backward Output
0 (A)	19 (T)	13 (N)	9 (J)
1 (B)	4 (E)	14 (O)	14 (O)
2 (C)	7 (H)	15 (P)	22 (W)
3 (D)	6 (G)	16 (Q)	24 (Y)
4 (E)	12 (M)	17 (R)	18 (S)
5 (F)	17 (R)	18 (S)	15 (P)
6 (G)	8 (I)	19 (T)	13 (N)
7 (H)	5 (F)	20 (U)	3 (D)
8 (I)	2 (C)	21 (V)	10 (K)
9 (J)	0 (A)	22 (W)	21 (V)
10 (K)	1 (B)	23 (X)	16 (Q)
11 (L)	20 (U)	24 (Y)	11 (L)
12 (M)	25 (Z)	25 (Z)	23 (X)

Table 3: Scramble Table for Rotor 2

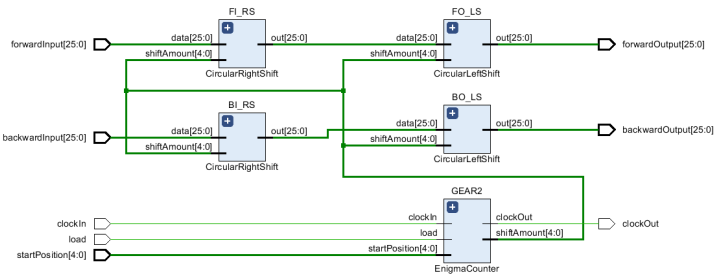

Figure 25: Implementation of Rotor 2

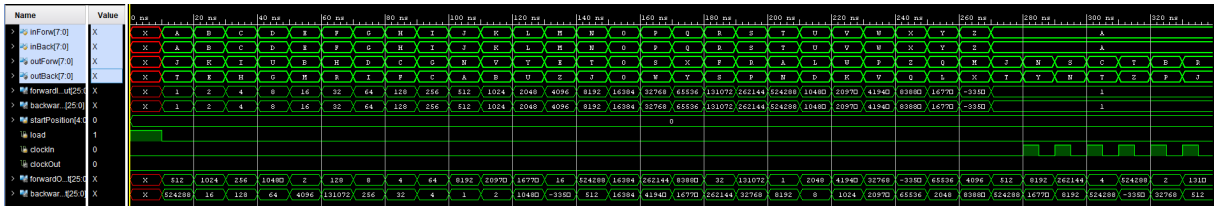

Figure 26: Simulation of Rotor 2 (decoded and encoded I/O)

2.4.5 Enigma Rotor 3

Forward Output Backward Input	Forward Input Backward Output	Forward Output Backward Input	Forward Input Backward Output
0 (A)	19 (T)	13 (N)	13 (N)
1 (B)	0 (A)	14 (O)	25 (Z)
2 (C)	6 (G)	15 (P)	7 (H)
3 (D)	1 (B)	16 (Q)	24 (Y)
4 (E)	15 (P)	17 (R)	8 (I)
5 (F)	2 (C)	18 (S)	23 (X)
6 (G)	18 (S)	19 (T)	9 (J)
7 (H)	3 (D)	20 (U)	22 (W)
8 (I)	16 (Q)	21 (V)	11 (L)
9 (J)	4 (E)	22 (W)	17 (R)
10 (K)	20 (U)	23 (X)	10 (K)
11 (L)	5 (F)	24 (Y)	14 (O)
12 (M)	21 (V)	25 (Z)	12 (M)

Table 4: Scramble Table for Rotor 3

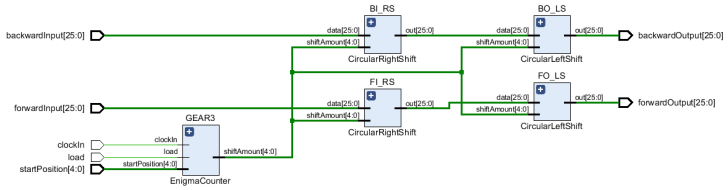

Figure 27: Implementation of Rotor 3

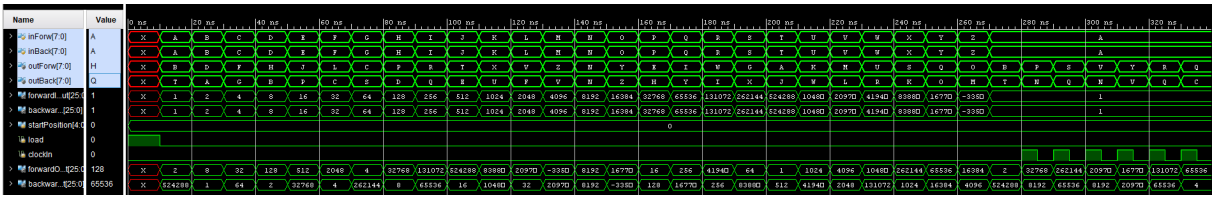

Figure 28: Simulation of Rotor 3 (decoded and encoded I/O)

2.4.6 Enigma Reflector

Enigma ciphering works by first ciphering forward, then reflecting it (again, cables are scrambled) and ciphering it second pass. This further complexes the cipher.

Output Connection	Input Connection	Output Connection	Input Connection
0 (A)	24 (Y)	13 (N)	10 (K)
1 (B)	17 (R)	14 (O)	12 (M)
2 (C)	20 (U)	15 (P)	8 (I)
3 (D)	7 (H)	16 (Q)	4 (E)
4 (E)	16 (Q)	17 (R)	1 (A)
5 (F)	18 (S)	18 (S)	5 (F)
6 (G)	11 (L)	19 (T)	25 (Z)
7 (H)	3 (D)	20 (U)	2 (C)
8 (I)	15 (P)	21 (V)	22 (W)
9 (J)	23 (X)	22 (W)	21 (V)
10 (K)	13 (N)	23 (X)	9 (J)
11 (L)	6 (G)	24 (Y)	0 (A)
12 (M)	14 (O)	25 (Z)	19 (T)

Table 5: Caption

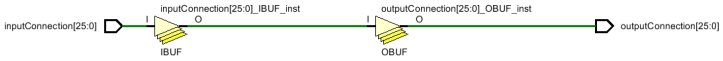

Figure 29: Enigma Reflector's design.

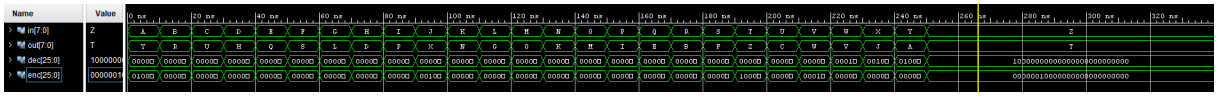

Figure 30: Simulation of Enigma Reflector (decoded and encoded I/O)

2.4.7 Enigma Machine and Enigma Communication Modules

To create a digital Enigma machine, these connections must be made in order:

1. **Decoder:** Decode ASCII character into binary.
2. **Plug-Board (Forward):** Provide decoded letter into plug-board's forward connections.

3. **Rotor 1 (Forward):** The forward output from plug-board must be connected to forward input of rotor 1. Also clock input of the general system (clock pulses when there is a new letter entry).

4. **Rotor 2 (Forward):** The forward output from rotor 1 is connected to forward input of rotor 2. Also clock output from rotor 1 is connected to rotor 2.

5. **Rotor 3 (Forward):** The forward output from rotor 2 is connected to forward input of rotor 3. Also clock output from rotor 2 is connected to rotor 3.

6. **Reflector:** Input should receive forward output of rotor 3 and output of reflector should be connected to rotor 3's backward input.

7. **Rotor 3 (Backward):** The output from reflector is connected to backward input of rotor 3.

8. **Rotor 2 (Backward):** The backward output from rotor 3 is connected to backward input of rotor 2.

9. **Rotor 1 (Backward):** The backward output from rotor 2 is connected to backward input of rotor 1.

10. **Plug-Board (Backward):** The backward output from rotor 1 is connected to backward input of Plug-Board.

11. **Encoder:** Plug-board's backward output is connected to encoder. Encoder then encodes ciphered binary stream into ASCII character.

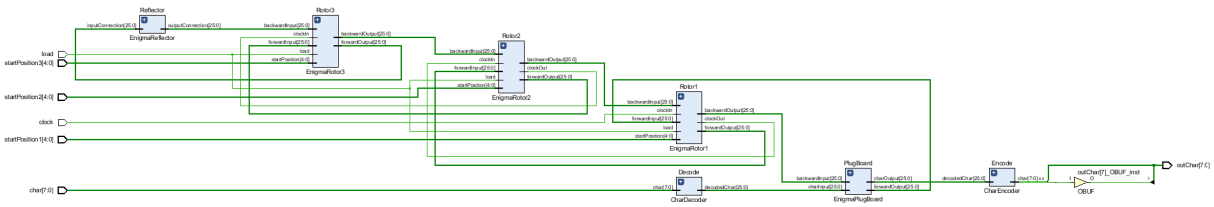

Figure 31: Enigma Machine's design.

By connecting two Enigma machines together in sync (their rotor set inputs and clock inputs are the same), we can make a working enigma communication module.

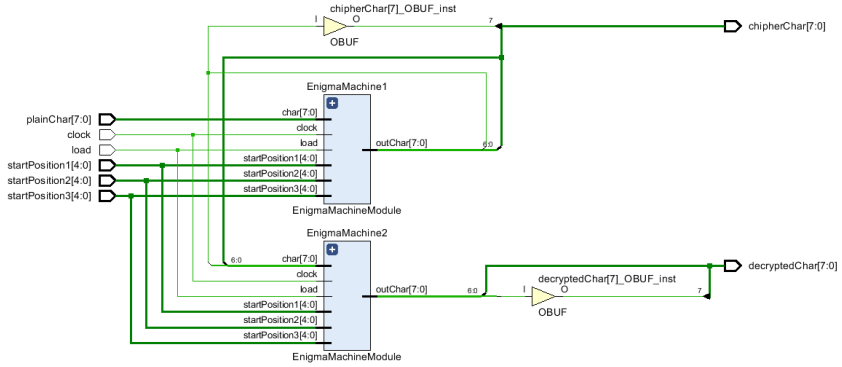

Figure 32: Enigma Communication Module's design.

Clock should be provided with every character input. Here are the simulation results: For text "BUBIRDENEMEYAZISIDIR", we get cipher "EKYKZUFDMBDUMEHHOIYN"

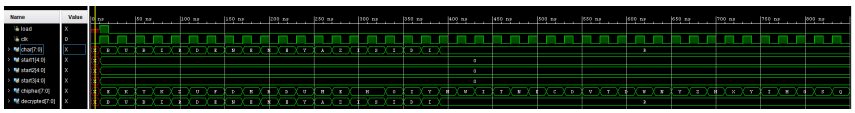

Figure 33: Enigma Communication Module's simulation.

and other Enigma machine successfully deciphers it. Also after the last R, we continued to provide clock input and R's chipher changed with clock cycle but two enigma machines kept communicating right because they were in sync.

3 DISCUSSION

In this homework, we learned that many cryptography techniques are used for secure communication in history. We first implemented helper modules to use in the next parts. We then implemented three cryptography methods.

First method was the Caesar Cipher, named after Julius Caesar. It is one of the first examples of cryptography. Characters are encrypted by shifting them a certain amount.

Second method was the Vigenere Cipher, which is used by the Confederate Army. Vigenere Cipher uses a key to encrypt the characters. To conduct the communication, both parties must have the same key.

Last method was the Enigma, which is used by German Army in the World War 2. Letters are scrambled and with each letter, machine's rotors turn, so this way same letters is not ciphered with same letters in ciphered text. Designing the counter which simulates the rotors turning was quite hard because they must be able to loaded asynchronously, apart from clock input.

4 CONCLUSION

We have used almost every knowledge we had over logic circuits to implement these encryption methods, also we are learned quite about encryption. Also we are familiarized with ASCII table while implementing these circuits. It would be very nice if we had some test cases to compare our results.

REFERENCES