ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 242E

LOGIC CIRCUITS LABORATORY
HOMEWORK REPORT

HOMEWORK NO : 3

HOMEWORK DATE : 22.05.2022

GROUP NO : G5

GROUP MEMBERS:

150200108 : ALP TÜRKBAYRAK

150180025 : HAMZA OĞUZ ŞENSES

SPRING 2022

Contents

1 INTRODUCTION

In this homework, we implemented three different historical ways of encryption and cryptography. We designed four helper modules that were used in the other main modules: CharDecoder, CharEncoder, CircularRightShift, CircularLeftShift. Then, we designed three different ciphering methods: Caesar's Cipher, Vigenere Cipher and finally the Enigma Machine.

2 MATERIALS AND METHODS

2.1 PART 1

For Part 1, we were asked to implement some helper modules to use in the later parts. We used the built-in shift operators and "for" loops for the circular shift circuits. For the decryption and encryption modules, we used a 26-bit variable to represent the encrypted form and an 8-bit variable to represent that letter's ASCII form. The modules we implemented transform these variables into each other respectively.

2.2 PART 2

For Part 2, we implemented the CaesarEncryption and CaesarDecryption modules. How the encryption works is the letter gets encrypted using the helper module from Part 1. Then, the encrypted letters' 26-bit code gets shifted by an amount that is read as input and this creates a new letter. For example, if shiftAmount is "3", then the letter "A" becomes "D". CaesarDecryption works by shifting it back to its original form. Then, we created a CaesarEnvironment module to gather these two modules under a single module and make the necessary connections between them.

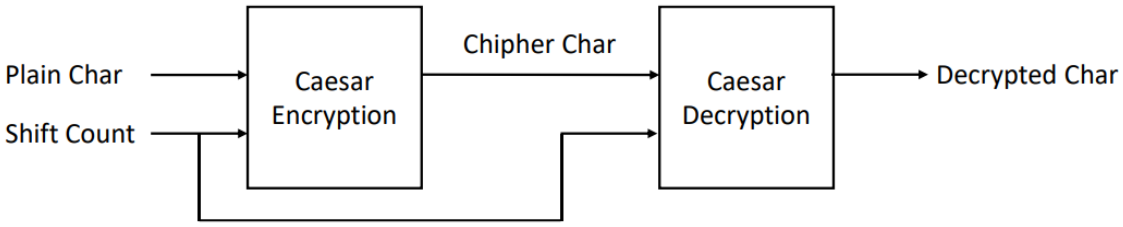

Figure 1: CaesarEnvironment Module

2.3 PART 3

For Part 3, we designed the VigenereEncryption, VigenereDecryption and VigenereEnvironment Modules. Vigenere Cipher works with a plain text and a key text. Key is used for both encrypting and decrypting so both the sending and the receiving parties must know it. The letters in plain text and key text are numbered by their alphabetical order eg. "A" = 0, "B" = 1 etc. Then, to create the encrypted version of the message, the numbers are summed as shown in Figure 2. Ci is the encrypted char, Pi is the plain text's char, Ki is the key text's char. Then, the sum's (Ci's) mod 26 is taken and turned into the letter that corresponds to that alphabetical order. In order to decrypt this message, Ki is subtracted from Ci as show in Figure 3. Di is the decrypted text. Its mod 26 is taken and the resulting number represents the alphabetical order of the letter that was encrypted.

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

Figure 2: Cipher Char Equation

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

Figure 3: Decrypted Char Equation

The VigenereEncryption does the encryption part, VigenereDecryption manages the decryption part and VigenereEnvironment module brings the previous two into one module that makes the necessary connections.

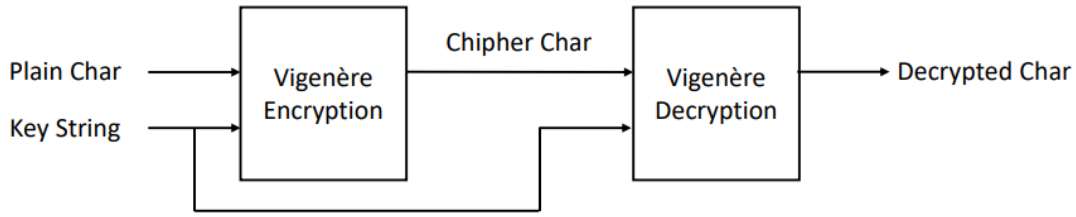

Figure 4: Vigenere Environment Module

2.4 PART 4

For Part 4, we implemented the famous Enigma Machine using 5 modules: PlugBoard, Rotor1, Rotor2, Rotor3 and Reflector. Each one of these modules take an input signal, turn it into another signal by referring to the tables given in the PDF, output it, and give it to the next module as input. The signal gets transformed inside the modules and gets passed to the next one. The connection is: input-PlugBoard-Rotor1-Rotor2-Rotor3-Reflector for encryption and the opposite of this for decryption. The most important point is, the Rotors change their configurance with the clock cycles: Rotor1 changes with every clock signal, Rotor2 changes with every 26 and Rotor3 changes with every 676 (square of 26) cycles. But for this homework's purposes, we used the given table to set the values at each module. We then implemented an EnigmaMachine Module to wire the previous 5 modules accordingly. Finally, we implemented an EnigmaCommunication module to connect two enigma machines together as can be seen from Figure 5.

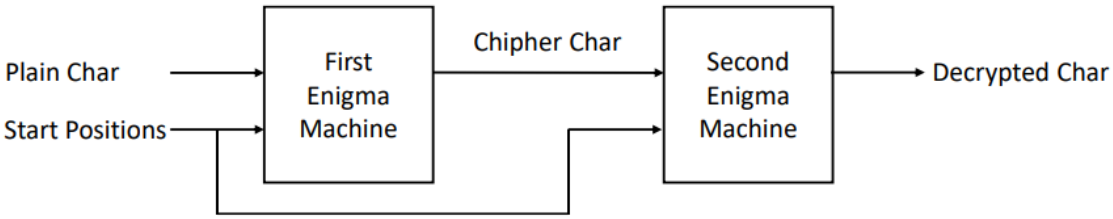

Figure 5: Vigenere Environment Module

3 RESULTS

3.1 PART 1

For the Part 1, we implemented the helper modules to use later. The simulation results can be seen below.

Figure 6 shows how a character that is fed as an input gets encrypted. Figure 7 shows how an encrypted character is decrypted. Figure 8 shows how an encrypted character is shifted left. Figure 9 shows how an encrypted character is shifted right.

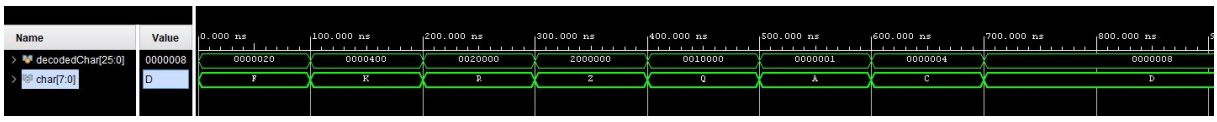

Figure 6: Char Encoder Sim

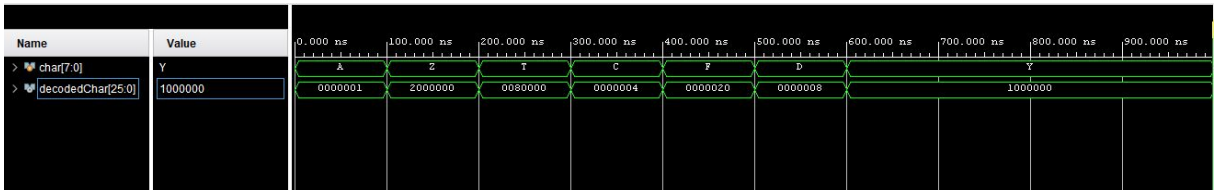

Figure 7: Char Decoder Sim

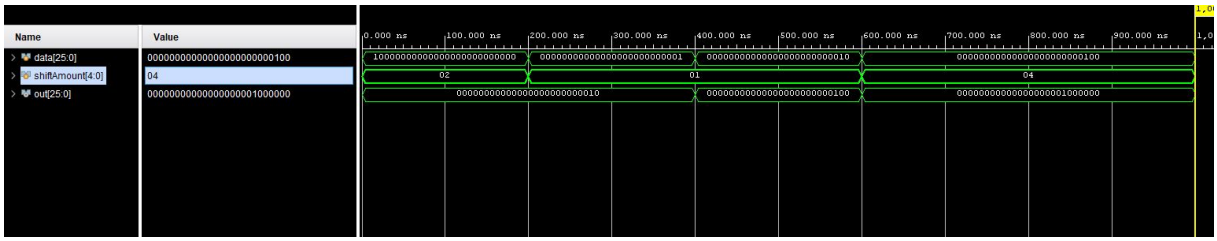

Figure 8: Circular Left Shift Sim

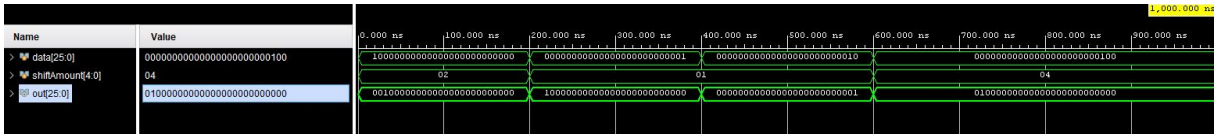

Figure 9: Circular Right Shift Sim

3.2 PART 2

For Part 2, we implemented the Caesar Cipher and its modules. The results of the simulations for each of the modules can be seen below.

Figure 10 shows the Caesar's way of encrypting. Figure 11 shows the Caesar's way of decrypting. Figure 12 shows the Caesar's Environment module working, which is the whole ciphering module.

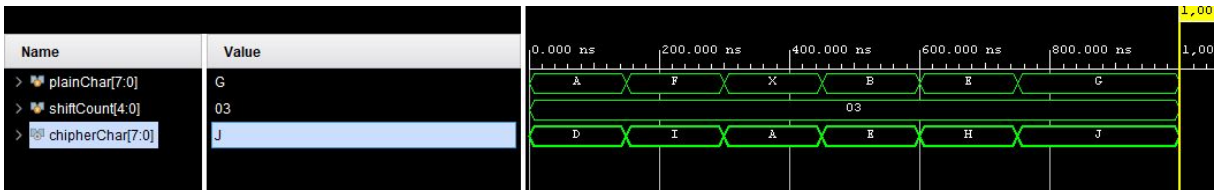

Figure 10: Caesar Encryption Sim

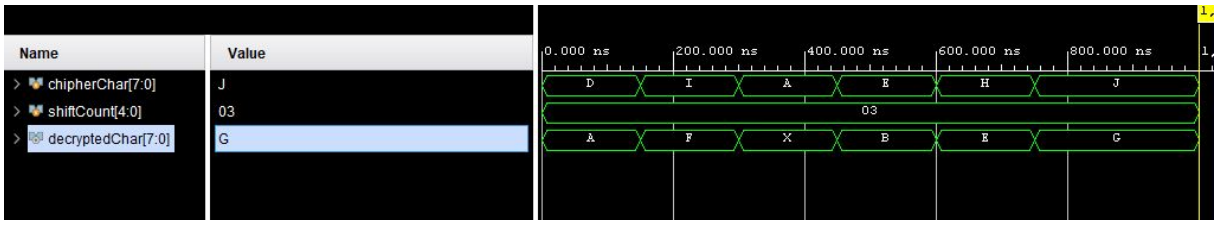

Figure 11: Caesar Decryption Sim

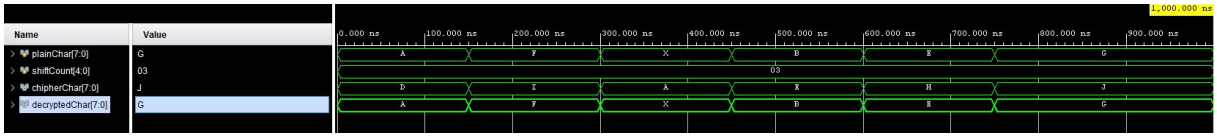

Figure 12: Caesar Environment Sim

3.3 PART 3

For Part 3, we implemented the Vigenere Cipher and its modules. The results of the simulations for each of the modules can be seen below.

Figure 13 shows the Vigenere's way of encrypting. Figure 14 shows the Vigenere's way of decrypting. Figure 15 shows the Vigenere's Environment module working, which is the whole ciphering module.

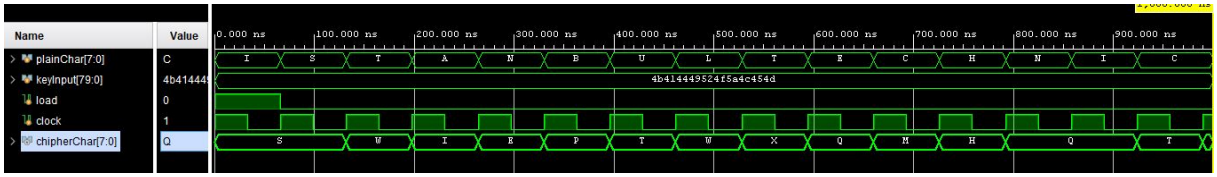

Figure 13: Vigenere Encryption Sim

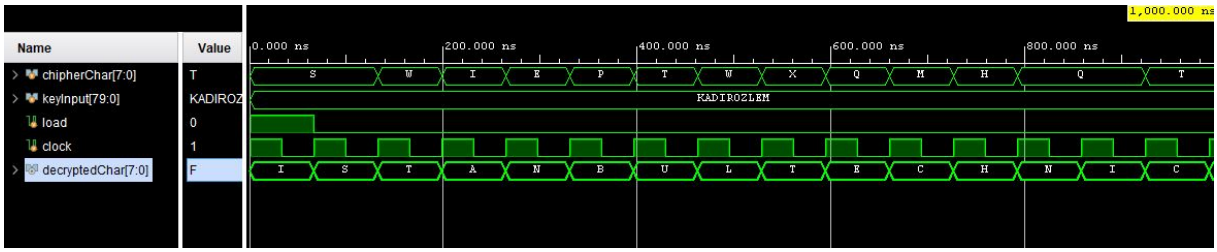

Figure 14: Vigenere Decryption Sim

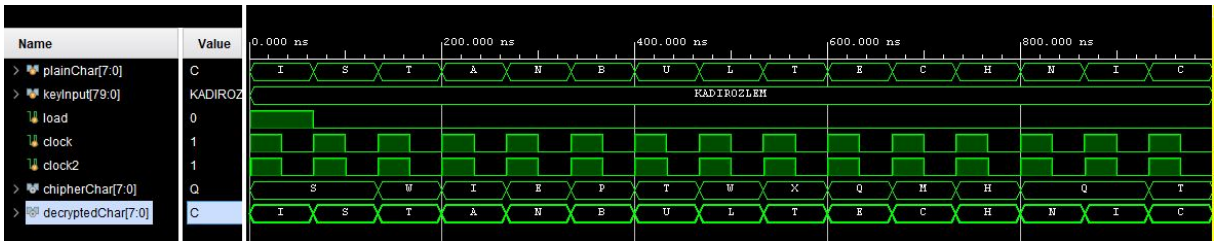

Figure 15: Vigenere Environment Sim

3.4 PART 4

For Part 4, we implemented the Enigma Machine and its modules. The results of the simulations for each of the modules can be seen below.

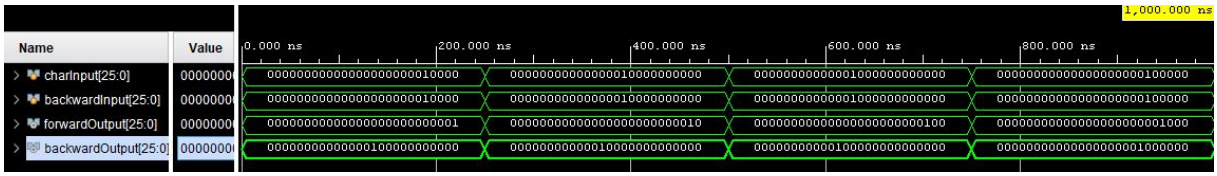

Figure 16: Plugboard Sim

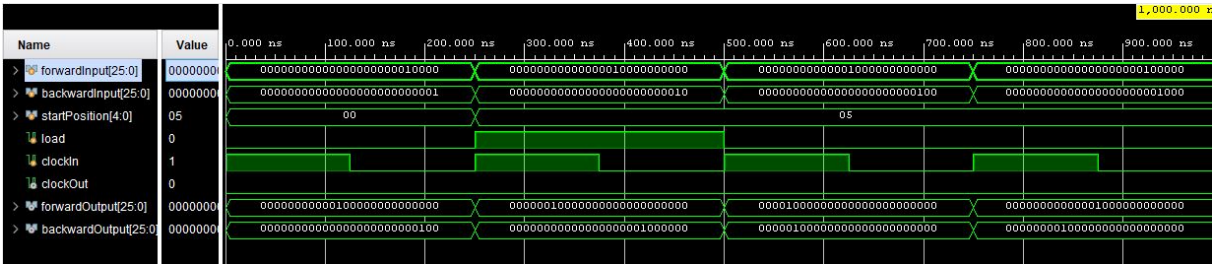

Figure 17: Rotor1 Sim

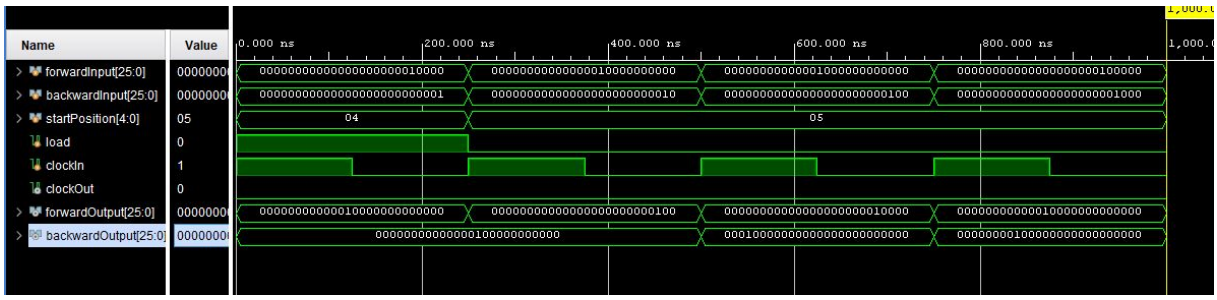

Figure 18: Rotor2 Sim

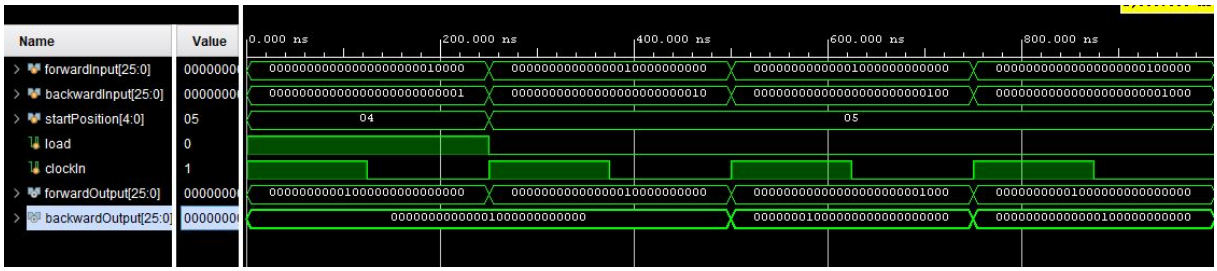

Figure 19: Rotor3 Sim

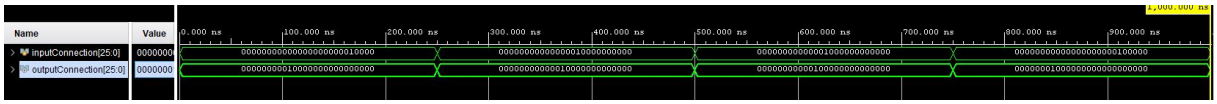

Figure 20: Reflector Sim

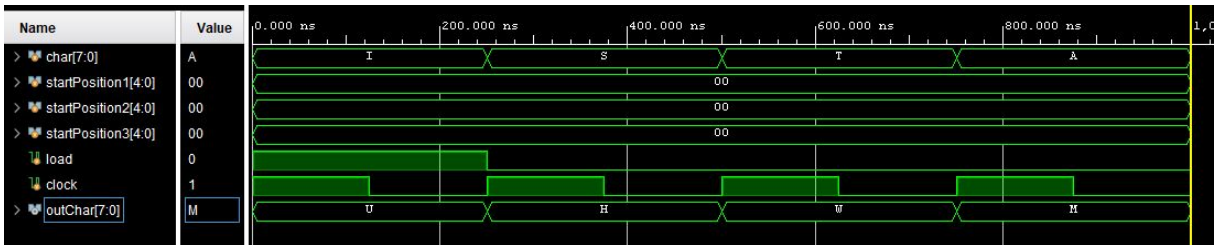

Figure 21: Enigma Machine Sim

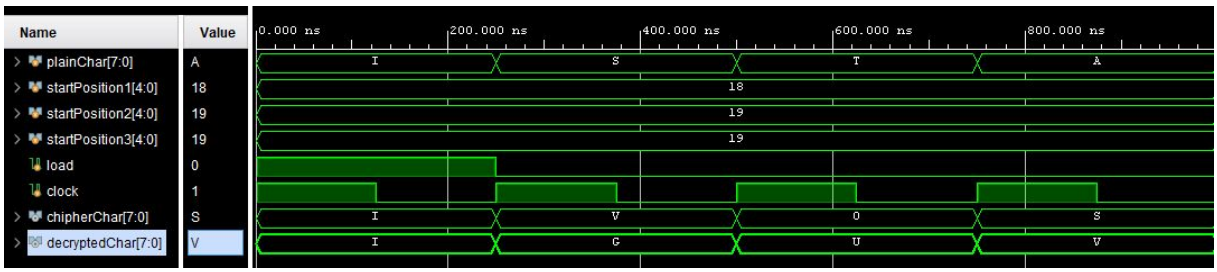

Figure 22: Enigma Communication Sim

4 DISCUSSION

In Part 1, we designed multiple modules. The modules in this part are for general-use and therefore very simple. CharDecoder module takes an 8-bit input that represents the ASCII code for a character ranging from A to Z. It is then transformed into a 26-bit variable of which only the nth bit is '1'. Here, n represents the alphabetical order of the character that needs to be encrypted (n = 1 for A, n = 26 for Z). The CharEncoder works the opposite way: it takes the 26-bit encrypted variable and decodes it into an 8-bit ASCII code for a character ranging from A to Z. CircularRightShift is used to shift a variable using the circular method which means shift it to the right once and carry the rightmost bit (LSB) to the leftmost bit (MSB). CircularLeftShift does the same thing except it shifts left and carries the leftmost bit (MSB) to the rightmost bit (LSB).

In Part 2, we implemented the Ceasar Cipher by utilizing the modules in Part 1. CaesarEncryption module takes a plainChar input which represents the character that we want to encrypt. Then, it requires the shiftCount which represents the shift amount that every char goes through to get encrypted. The module takes plainChar, shifts it left by the shiftCount. The new variable gets outputted as the chipherChar. CaesarDecryption is used to perform the opposite of this operation. It shifts the chipherChar left by the shiftCount to get the plainChar. This is why both of the modules need to have the same shiftCount input in them. This is why we use the CaesarEnvironment module. CaesarEnvironment module uses a CaesarEncryption module and a CaesarDecryption module and feeds the same shiftCount to both of them, making sure that the ciphering is done correctly with the same shiftCount in both modules.

In Part 3, we implemented the Vigenere Cipher. Its workings are explained in the Materials & Methods section of this report. The modules work as follows: The VigenereEncryption Module takes the plainChar as an input which is the character that we want to encrypt. It then takes the 10 character long (80 bits - 8 bits for 1 char) keyInput which is the same for both encrypting and decrypting. Then, we have a clock and a load input of which both imply the timing of the code. On the rising-edge of the load input, the keyInput is read. At the positive edge of the clock, the chipherChar is assigned as Ci = (Pi + Ki) mod 26. The VigenereDecryption does the opposite of this operation. Instead of adding (Pi + Ki = Ci), it subtracts (Pi = Ci - Ki) and takes mod 26 to find the plainChar that was encrypted. Finally, VigenereEnvironment uses one of each of the modules because in order to encrypt and decrypt correctly, both modules have to operate with the same key. Environment makes sure both of the modules get fed the same key.

In Part 4, we implemented the Enigma Cipher Machine which is made up of 7 modules. PlugBoard takes an input char and outputs a char that was given to us in the PDF tables. Then, PlugBoard gives the output to Rotor1 which does the same and gives the output to Rotor2 and that to Rotor3 and finally, Reflector. The output of the Reflector is the encrypted version of the character. Each of these modules have 2 inputs and 2 outputs - one of them is used for forward transfer of the variables and the other is used for backward transfer. Forward transfer is used for encrypting and backward transfer is used for decrypting the characters. Then, we connected these modules and created an EnigmaMachine module. Finally, we created an EnigmaCommunication module to connect two enigma machines - one used for encrypting and one used for decrypting.

5 CONCLUSION

This homework allowed us to learn the inner-workings of some of the most famous ciphering mechanisms that were used historically to shape the future of the world. It was so amazing to see that even though the techniques that were used had changed over the years (centuries), the logical implications of the methods were all the same. And we have learned most of them in our time here in the university and they all seem so basic/simple after having worked with them. It is really great to see what we have learned in our courses are used in solving some real-life problems.