Do Not Distribute. ALL Rights Reserved.

National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2022

Homework Assignment #2 **(10%)**

Enigma

Assigned on Oct 13, 2022
Due by **Oct 27, 2022**

Assignment Description

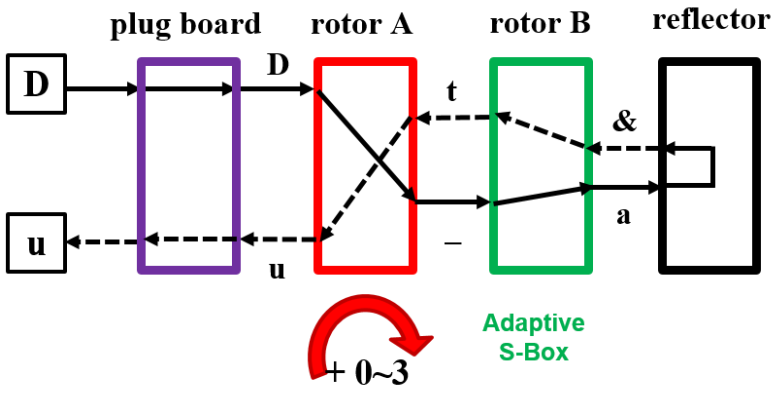


image ref: https://en.wikipedia.org/wiki/Enigma_machine

Fig. 1. A novel Enigma with a plug board, two rotors and a reflector. The solid arrows stand for the forward rotor mapping, and the dotted ones stand for the inverse rotor mapping.

Enigma is the electro-mechanical rotor cipher machine which was used by German military in World War II. In this assignment, you need to implement an improved version of Enigma machine using Verilog RTL. As shown in Fig. 1, for every 6-bit input Enigma code (plaintext) it will generate one 6-bit ciphertext through the plug board, two rotors, the reflector, and the inverse of them which all perform one-to-one substitution functions.

Fig. 2 shows an example for encryption operation. Each 6-bit Enigma code has its corresponding ASCII symbol. For example, 6'h23 means 'D', and through the plug board it will be transformed into 'D' (6'h23). Then through rotorA it becomes '_'(6'h3d); through the rotorB it becomes 'a'(6'h00), and through the reflector you will have '&'(6'h3f) out of it. The inverse rotors do exactly the opposite of the rotors, so the '&' will be transformed to 't'; 't'

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

will be transformed to ‘u’, and finally the output ‘u’. The security of Enigma is provided by the rotation or permutation of the rotors.

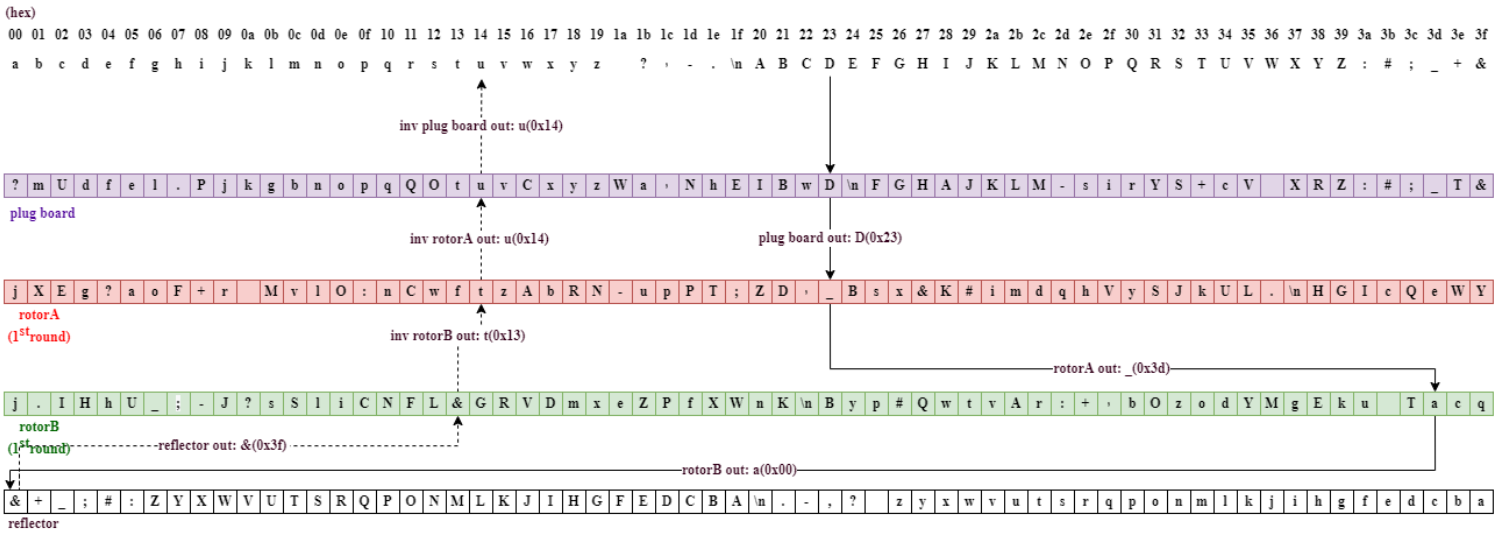


Fig. 2. Example of encryption for an input ‘D’ (the first round)

As shown in Fig. 2, there are one plug board, two rotors, and a reflector in our Enigma machine. To further increase the security of enigma, they all have different behaviors in encryption and decryption modes. Following is a detailed introduction to them.

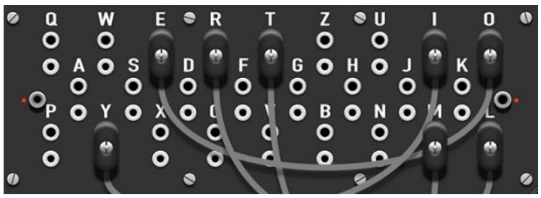


image ref: https://piotte13.github.io/enigma-cipher/

(1) Plug board

The plug board is the first layer in our Enigma machine. It has several cables with a plug at each end that could be used to plug pairs of letters together. If A were plugged to B, then on typing the letter A, the electric current would follow the path that was normally associated with the letter B, and vice versa.

There are 16 cables in our Enigma machine to connect letter pairs. Once cables are settled, the plug board will fix during encryption. In this assignment, you should build the plug board by yourself. Fig. 3 depicts the behavior of the plug board. The Enigma machine sequentially receive 16 letter pairs during the LOAD state.

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

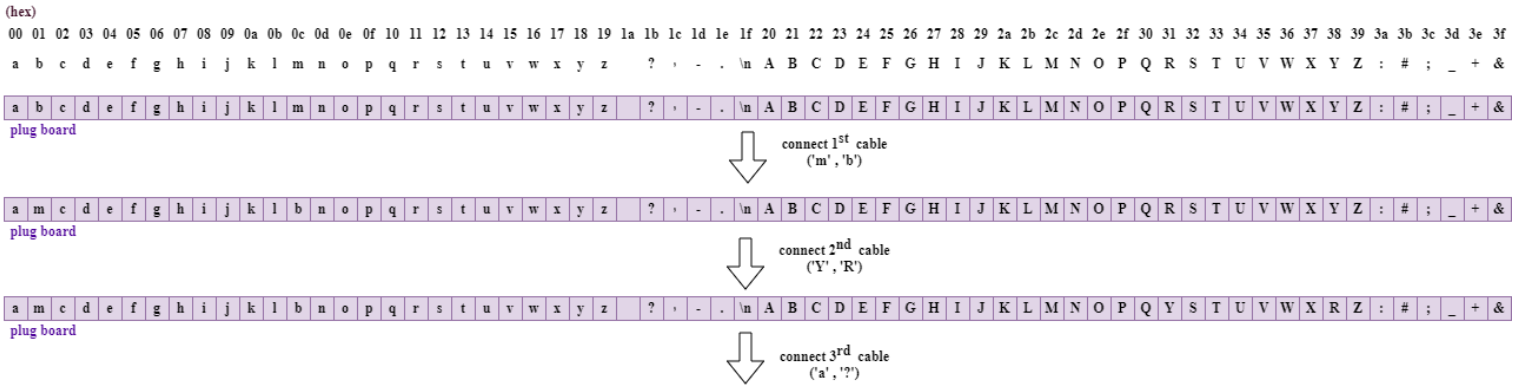


Fig. 3. Behavior of the plug board

(2) RotorA

The rotors form the heart of the Enigma machine. They perform only simple type of encryption, a simple substitution cipher.

In this homework, the Enigma machine receives the initial permutation of rotorA in the LOAD state through the code_in port. After that, in order to avoid the same letters from being encrypted into the same words, rotorA shifts right by 0-3 symbols after encrypting one symbol. The least significant two bits of the 6-bit code of the rotorA output symbol are used to decide the number of rotation (i.e., rotorA mode). As shown in Fig. 4, the rotorA mode in first round is 1, since the rotorA output symbol is '_'(6'b111101). The rotorA will shift right by 1 symbol before the next symbol encryption.

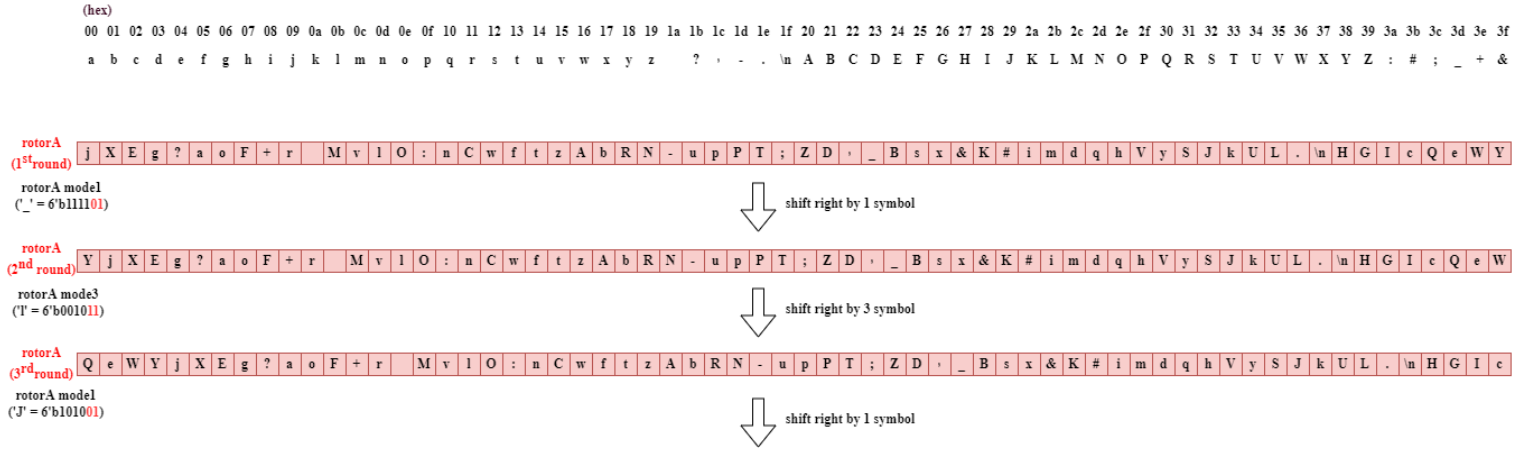


Fig. 4. Behavior of the rotorA

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

(3) RotorB

The rotorB is the novel part in our Enigma machine. It is further improved by applying two-stage adaptive permutation after each symbol encryption.

Same as the rotorA, the rotorB gets its initial permutation through the code_in ports in the READY state. The different part is the rotor reorganization after each symbol encryption/ decryption. Fig. 5 illustrates the behavior of the rotorB. The first-stage S-Box8 applies for each eight contiguous symbols and has eight possible modes as shown in Fig.6. The least significant three bits of the 6-bits code of the rotorB output symbol are used as the selection signal. For example, the output symbol ‘a’ has the value 6’b000000, so the S-Box8 mode is 0 for the first round. The second-stage S-Box64 follows a fixed permutation as shown in Fig. 7.

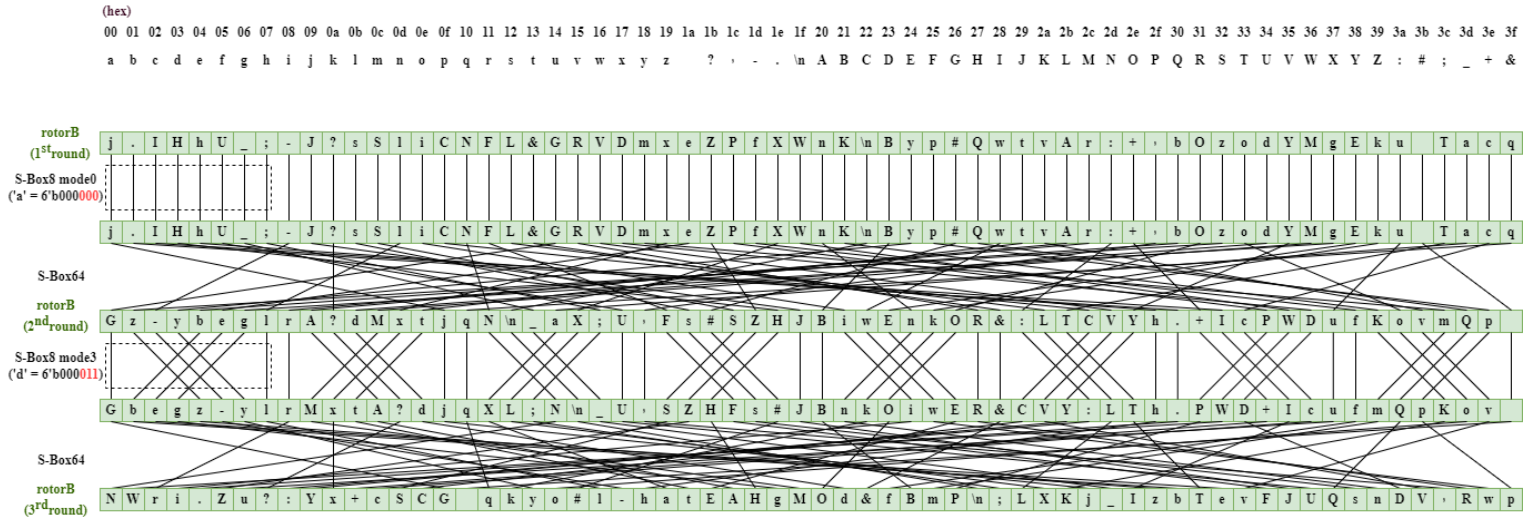


Fig. 5. Behavior of the rotorB

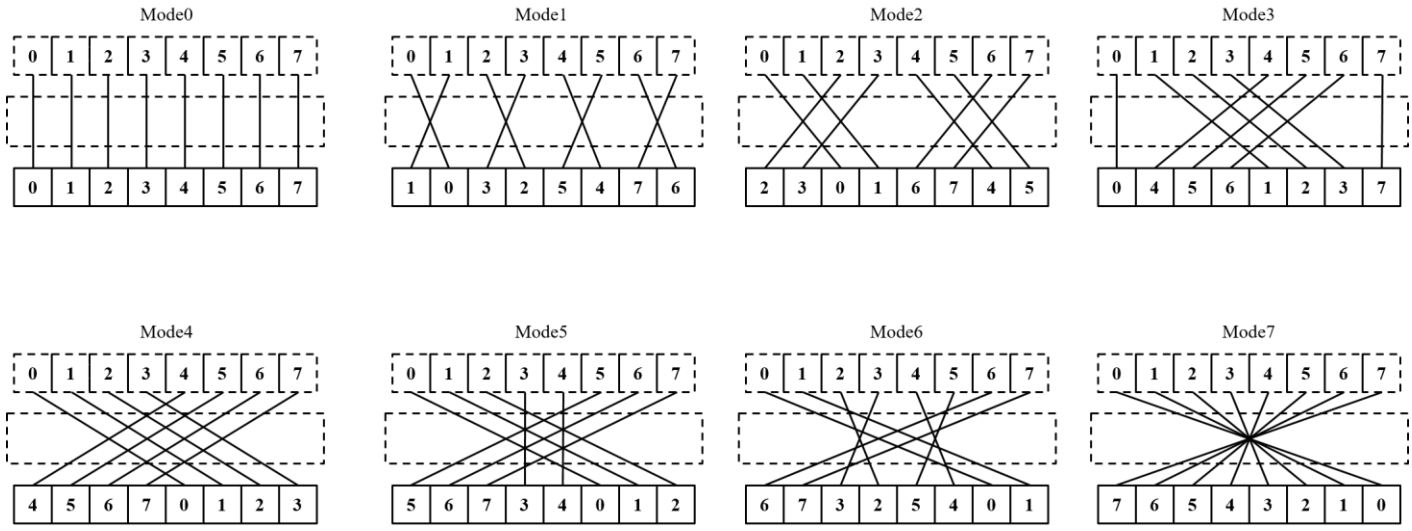


Fig. 6. Eight modes of S-Box8 permutation

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

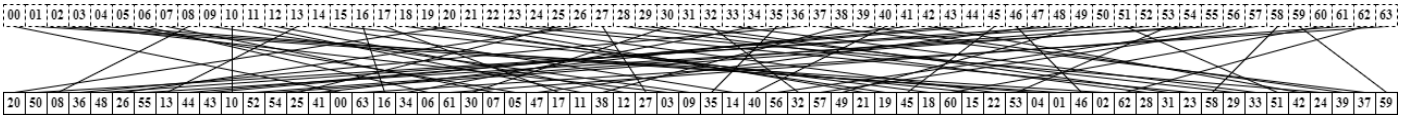


Fig. 7. Fixed S-Box64 permutation

(4) Reflector

The reflector is the last layer in our Enigma machine, being static in encryption and decryption. As illustrated in Fig. 8, the reflector is a fixed table that has the reverse order of the ASCII table.

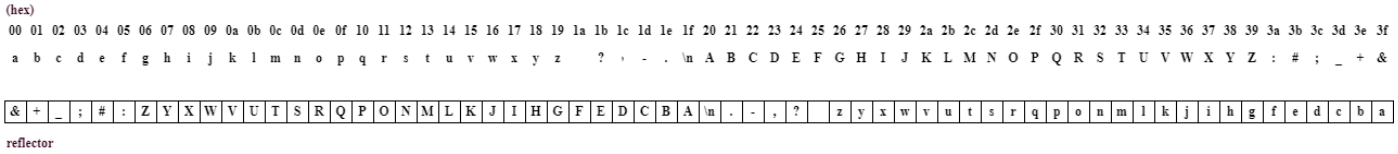


Fig. 8. Fixed reflector

Note: Be careful of the adaptive selection signals you choose for the rotorA and rotorB when the Enigma machine operates in the decryption mode.

For more examples, the encryption operations for two following symbols ‘o’ and ‘ ’(space) are given in Fig. 9 and 10 respectively. A property of Enigma is that it does decryption in the same way as encryption. For instance, if you input ‘u’, ‘t’, and ‘.’ to this Enigma machine sequentially and change the rotors accordingly, you will get ‘D’, ‘o’, and ‘ ’ respectively form the output.

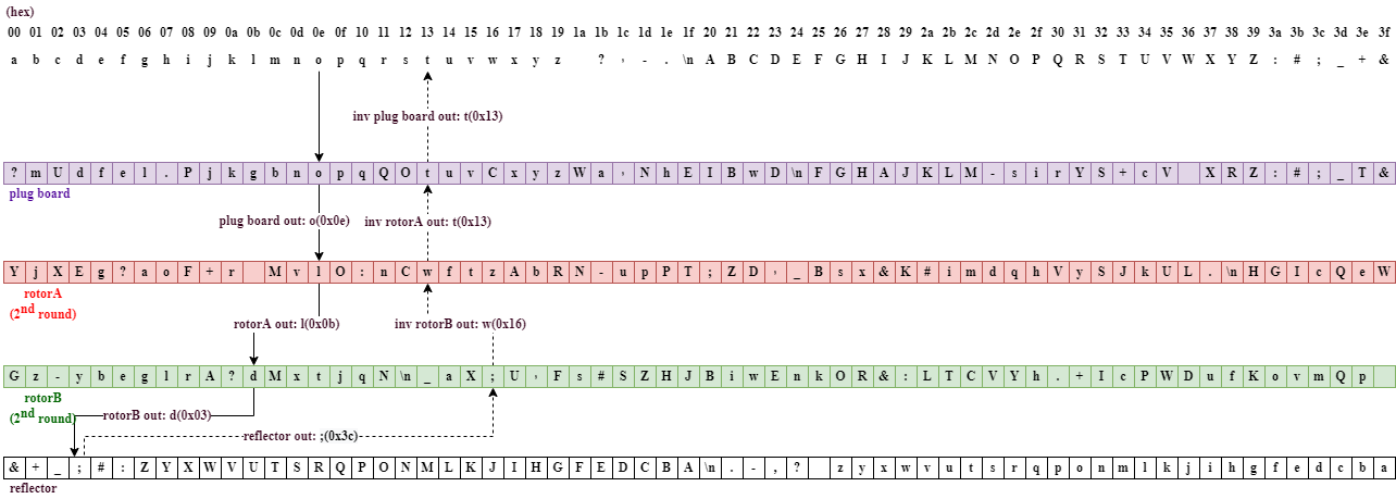


Fig. 9. Example of encryption for an input ‘o’ (the second round)

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

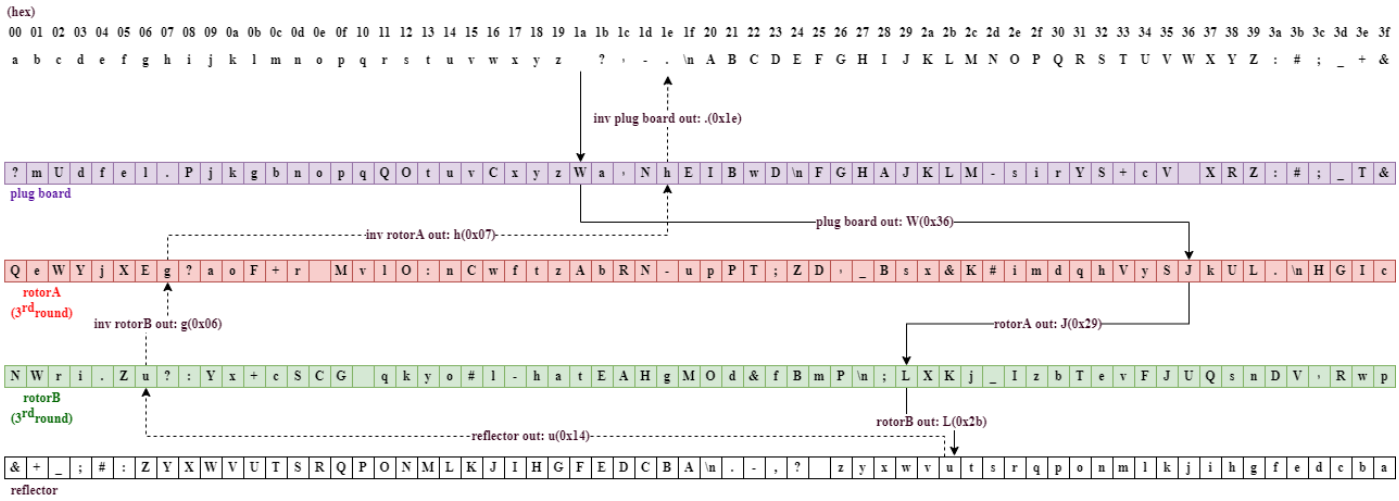
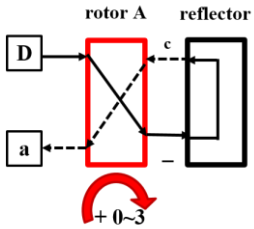


Fig. 10. Example of encryption for an input ‘ ’ (the third round)

Note: Your testbench must follow the reference waveform. Otherwise, your RTL codes might not pass the simulation when TA uses his testbench to evaluate.



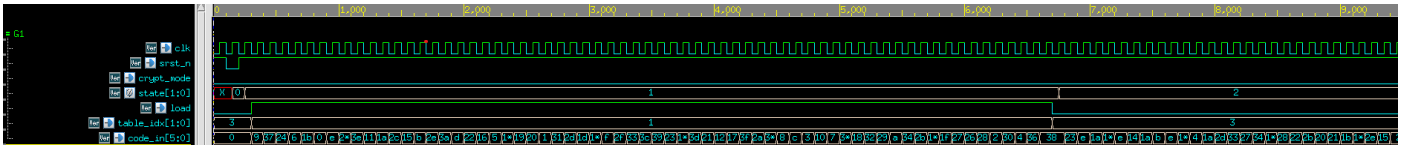
Step in this homework

(1) Part1 (rotorA + reflector) **(2%)**

- **a.** Given the behavior model, implement the testbench **test_enigma_part1.v** for **encrypting patten1 (1%)**
- **b.** Change the behavior model to your synthesizable RTL code **enigma_part1.v (1%)**

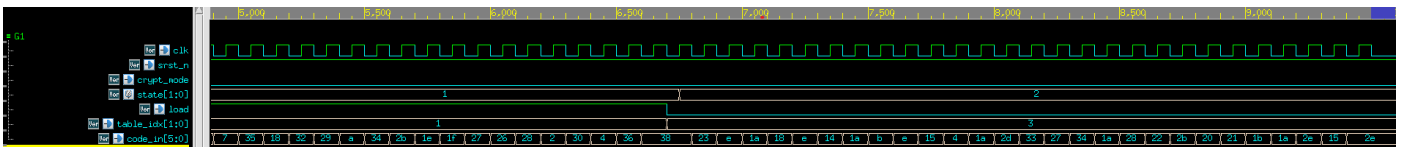
Part1 Reference Waveform

- **IDLE → LOAD (encrypt, pattern1)**



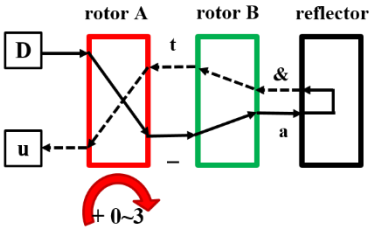
Note: In the part1, the READY state takes 64 cycles to load rotorA.

- **LOAD → READY (encrypt, pattern1)**



Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.



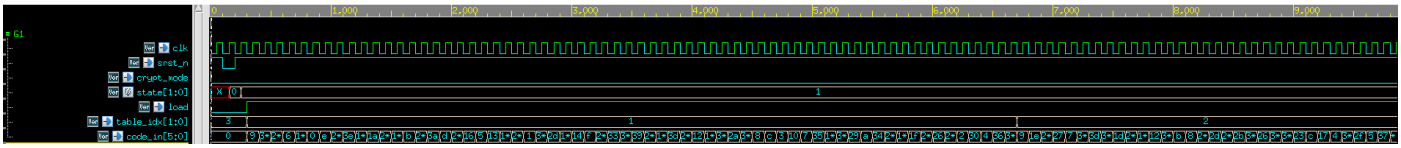
(2) Part2 (rotorA + rotorB + reflector) **(4%)**

a. Implement the synthesizable Enigma RTL code **enigma_part2.v (2%)**

b. Write the testbench **test_enigma_part2.v** for **encrypting/ decrypting pattern1-2 (2%)**

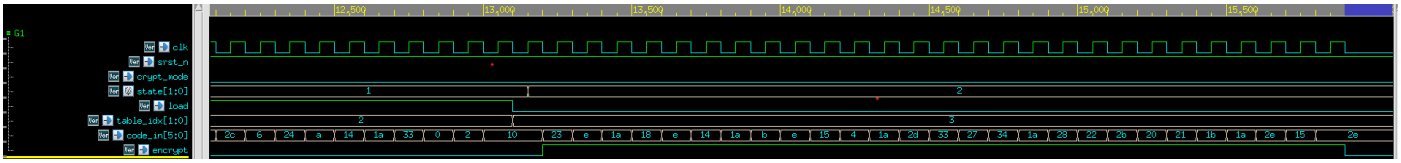
Part2 Reference Waveform

- **IDLE→ LOAD (encrypt, pattern1)**

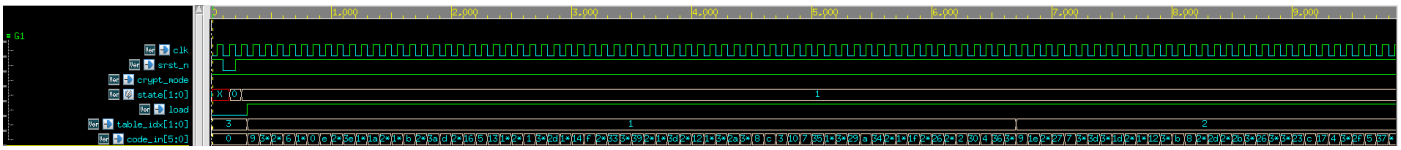


Note: In the part2, the READY state takes 64 + 64 cycles to load rotorA + rotorB.

- **LOAD → READY (encrypt, pattern1)**

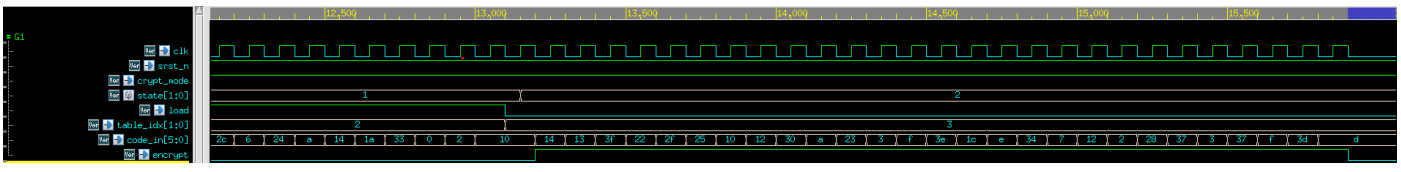


- **IDLE→ LOAD (decrypt, pattern1)**



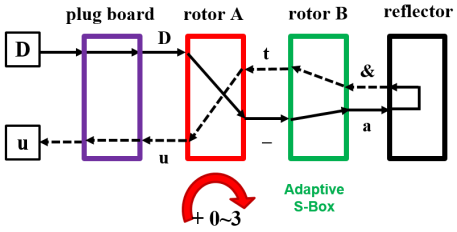
Note: In the part2, the READY state takes 64 + 64 cycles to load rotorA + rotorB.

- **LOAD → READY (decrypt, pattern1)**



Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

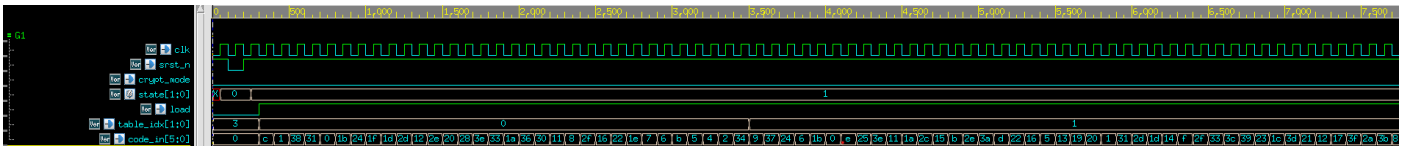


(3) Part3 (plug board + rotorA + rotorB + reflector) **(4%)**

- **a.** Implement the synthesizable Enigma RTL code **enigma_part3.v (2%)**
- **b.** Write the testbench **test_enigma_part3.v** for **encrypting/ decrypting pattern1-2 (1%)**
- **c.** Write the testbench **test_enigma_part3_display.v** to **decrypt** and **save** the results of **pattern2-3 in ASCII format (1%)**

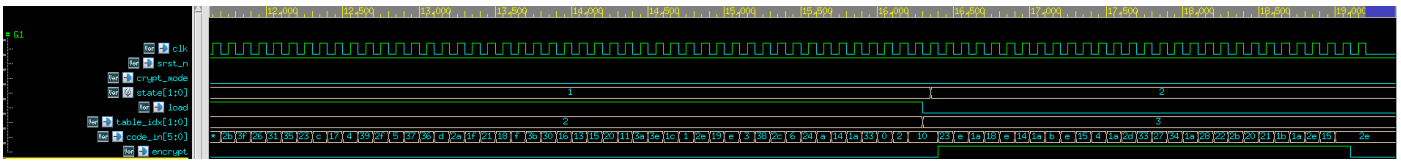
Part3 Reference Waveform

- **IDLE→ LOAD (encrypt, pattern1)**

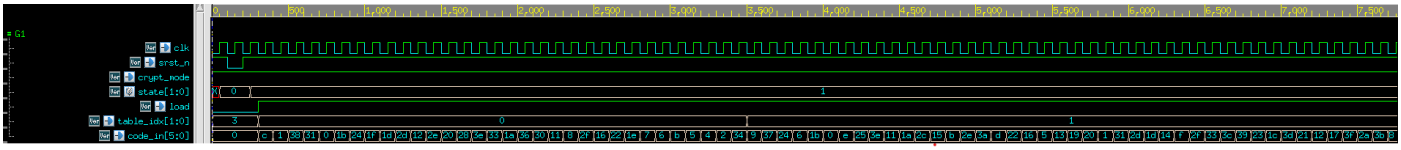


Note: In the part3, the READY state takes 32 + 64 + 64 cycles to load plug board + rotorA + rotorB.

- **LOAD → READY (encrypt, pattern1)**

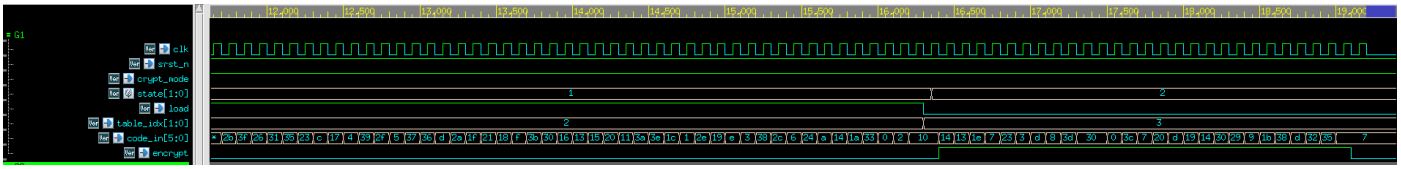


- **IDLE→ LOAD (decrypt, pattern1)**



Note: In the part3, the READY state takes 32 + 64 + 64 cycles to load plug board + rotorA + rotorB.

- **LOAD → READY (decrypt, pattern1)**

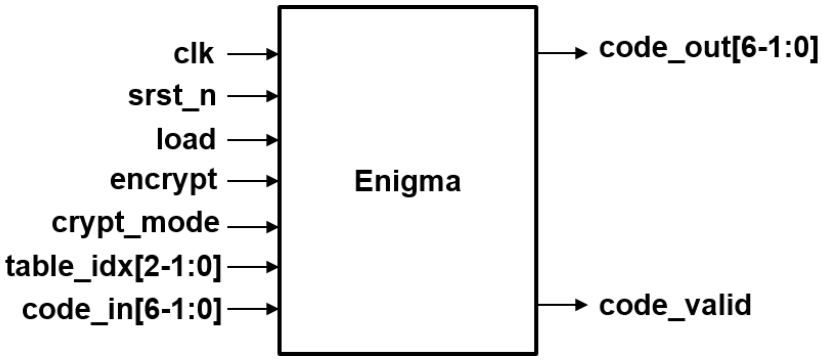


Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

RTL I/O name definition

The RTL module name and I/O definition should be as follows.

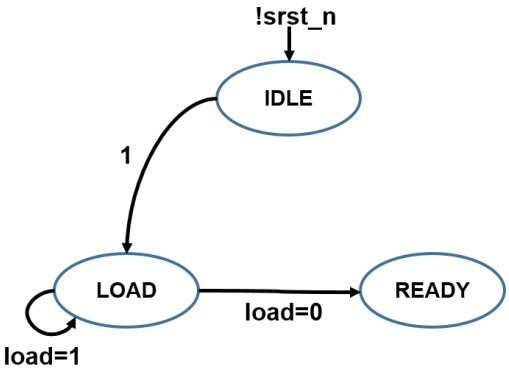


I/O name	Number of bits	Description
clk	1	Clock input
srst_n	1	Synchronous reset (active low)
load	1	Load control signal (level sensitive) 0/ 1: inactive/ active Effective in IDLE and LOAD states
encrypt	1	Encrypt control signal (level sensitive) 0/ 1: inactivate/ active Effective in READY state
crypt_mode	1	0: encrypt, 1: decrypt
table_idx	2	Index of rotor to be load (2'b00: plug board; 2'b01: rotorA; 2'b10: rotorB)
code_in	6	When load is active, then code_in is input of rotors. When encrypt is active, then code_in is input of code words.
code_out	6	encrypted/ decrypted code word (flip-flop output)
code_valid	1	0: non-valid code_out; 1: valid code_out (flip-flop output)

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

Your module shall also obey this finite state machine:



For encrypting a new plaintext, the srst_n signal is used to reset the state to IDLE first. The state will then be changed to LOAD in the following cycle. The LOAD state is used to initialize rotors. You should build the rotors according to the corresponding table_idx. After the all information for rotors are been sent, the load signal will turn to inactive (i.e., load = 0), causing the FSM enter the READY state. In the READY state, the Enigma machine performs the encryption or decryption whenever the signal encrypt signal is set high. (**Notice that the encrypt signal will keep high in READY state)**

Hint: You can use a fixed look-up table for the reflector, and register files for the rotors. The rotorB permutation and the inverse rotors are the tricky parts.

The following attached files are provided for this assignment:

Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

Filename	Description
behavior_model.v	Behavior function of part1 one rotor Enigma machine
partX_ciphertextY.dat	The encrypted text file (used as golden files in encryption mode)
partX_plaintext1Y.dat	The non-encrypted text file (used as golden files in decryption mode)
plug_board_group.dat rotorA.dat rotorB.dat	Hex files of the rotors (Hint: Can be loaded in testbench by $readmemh command)
run_rtl.sh run_display.sh	VCS simulation scripts for each part. (Note: TA will evaluate your homework by executing them)
display_enigma_code.v	A Verilog task for displaying ASCII format in part3
show_enigma_code.v	Example of how to use display_enigma_code.v in your testbench. E.g. $ vcs -R +v2k -full64 show_enigma_code.v +define+PAT1 $ vcs -R +v2k -full64 show_enigma_code.v +define+PAT2

Grading policy

For this assignment, you need to turn in the following files:

1. Functional module enigma_part1.v (1%) and its spyglass report file to show the synthesizability. (If it's not synthesizable, you cannot get the score.)
2. Testbench test_enigma_part1.v (1%) for encrypting pattern1 with behavior_model.v and enigma_part1.v
3. Functional module enigma_part2.v (2%), and its spyglass report file to show the synthesizability. (If it's not synthesizable, you cannot get the score.)
4. Testbench test_enigma_part2.v (2%) for encrypting/ decrypting pattern1-2.
5. Functional module enigma_part3.v (2%), and its spyglass report file to show the synthesizability. (If it's not synthesizable, you cannot get the score.)
6. Testbench test_enigma_part3.v (1%) for encrypting/ decrypting pattern1-2.
7. Testbench test_enigma_display.v (1%) for decrypting ciphertext2.dat and ciphertext3.dat; then saving the results into files **in ASCII format.** (named as plaintext2_ascii.dat and plaintext3_ascii.dat in the **result folder**).

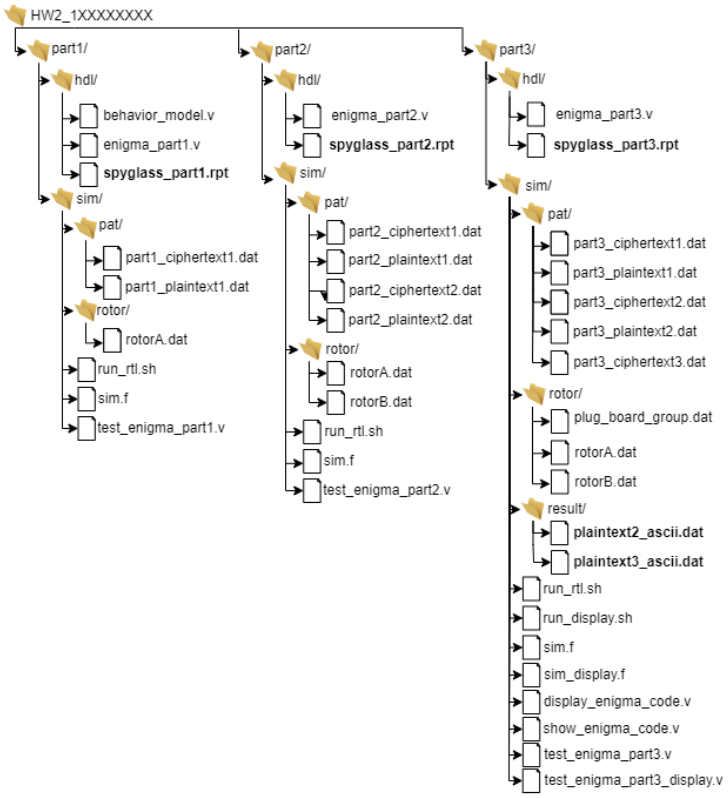
Do Not Distribute. ALL Rights Reserved.

Do Not Distribute. ALL Rights Reserved.

Deliverable

Note: 1% punishment for wrong file delivery.

File Organization



Note1: Bold font indicates new files.

Note2: 1XXXXXXXX is your student ID.

Note3: There are simulation scripts (i.e., run_rtl.sh) in each part.

TA will evaluate your homework by executing them.

Do Not Distribute. ALL Rights Reserved.