National Tsing Hua University
Department of Electrical Engineering
EE429200 IC Design Laboratory, Fall 2021

Homework Assignment #2 **(10%)**
Enigma
Assigned on Oct 14, 2021
Due by **Oct 28, 2021**

Assignment Description

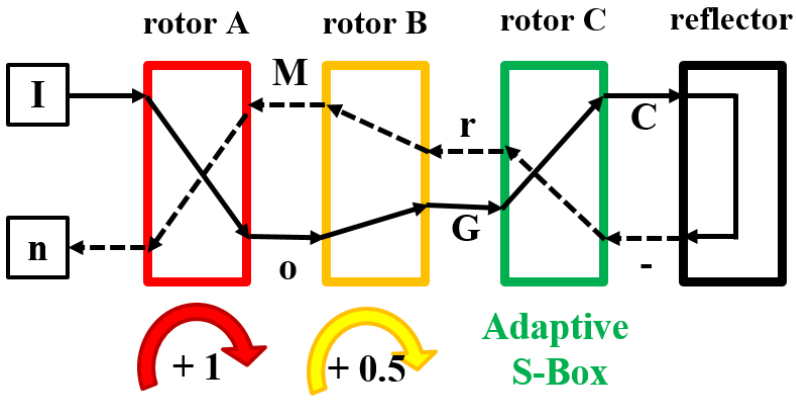


Fig. 1. A novel Enigma with three rotors and a reflector. The solid arrows stand for forward rotor mapping, and the dotted ones for inverse rotor mapping.

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 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 rotors, the reflector, and the inverse rotors 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'h28 means 'I', and through the rotorA it will be transformed into 'o' (6'h0e). Then through rotorB it becomes 'G'(6'h26); through the rotorC it becomes 'C'(6'h22), and through the reflector you will have '-'(6'h1d) out of it. The inverse rotors do exactly the opposite of the rotors, so the '-' will be transformed to 'r'; 'r' will be transformed to 'M', and finally the output 'n'. The security of Enigma is provided by the rotation or permutation of the rotors. After encrypting one symbol, the rotorA shifts

right by one symbol as shown in Fig. 3. After encrypting every two symbols, the rotorB shifts right by one symbol as shown in Fig. 4. Note that the reflector is always fixed.

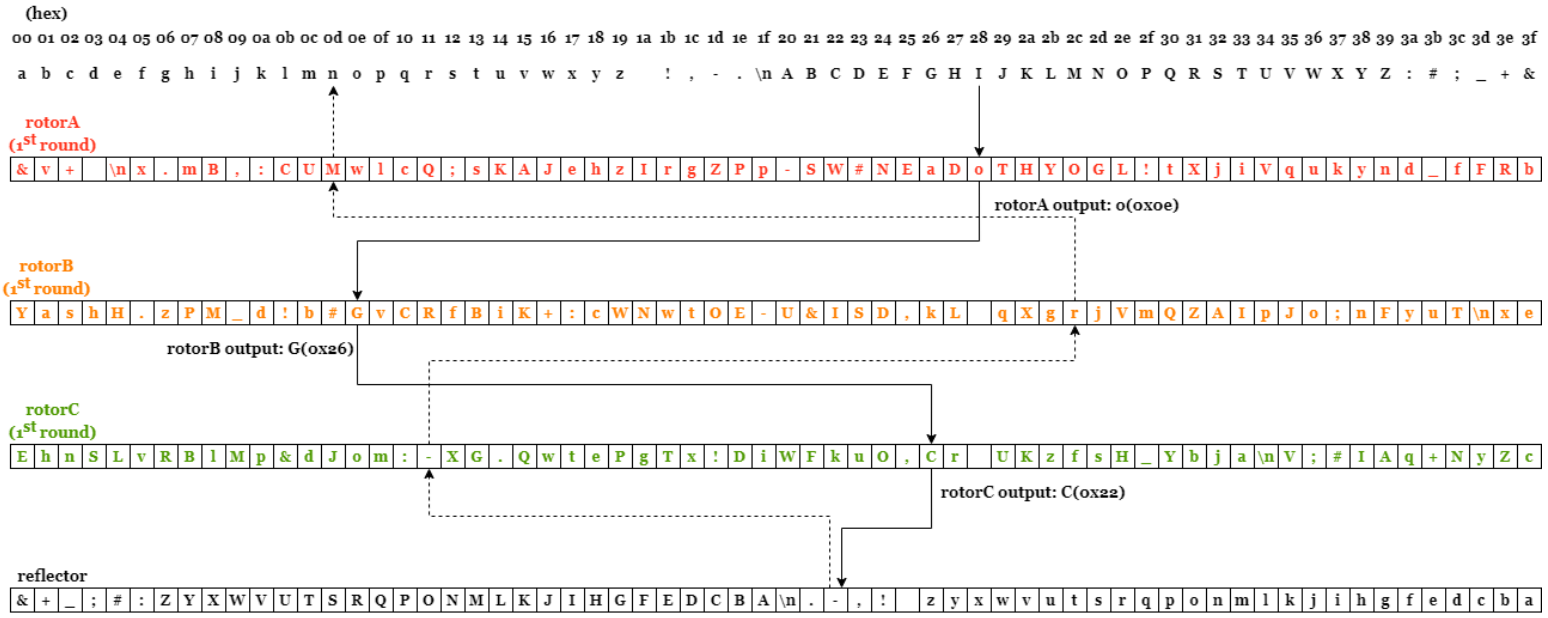


Fig. 2. Example of encryption for an input 'I' (the first round)

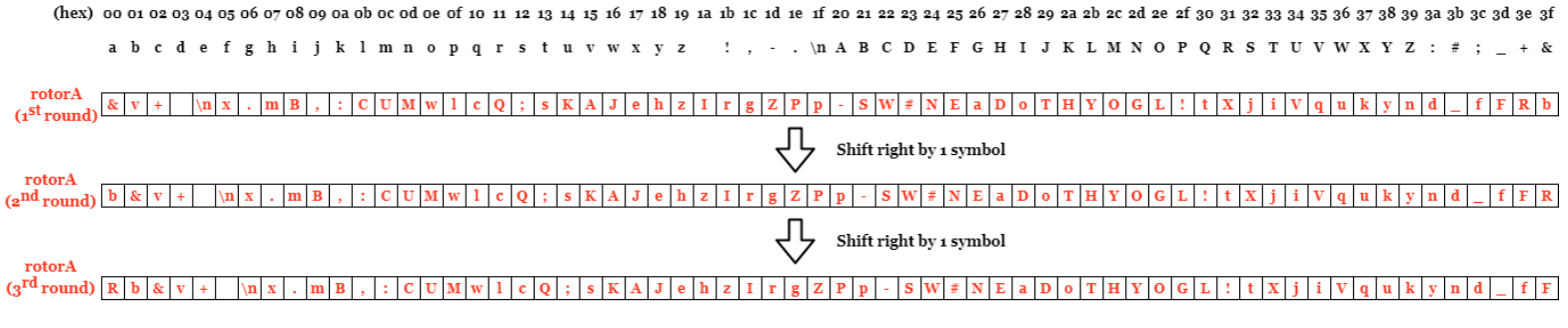


Fig. 3. Rotation of the rotorA.

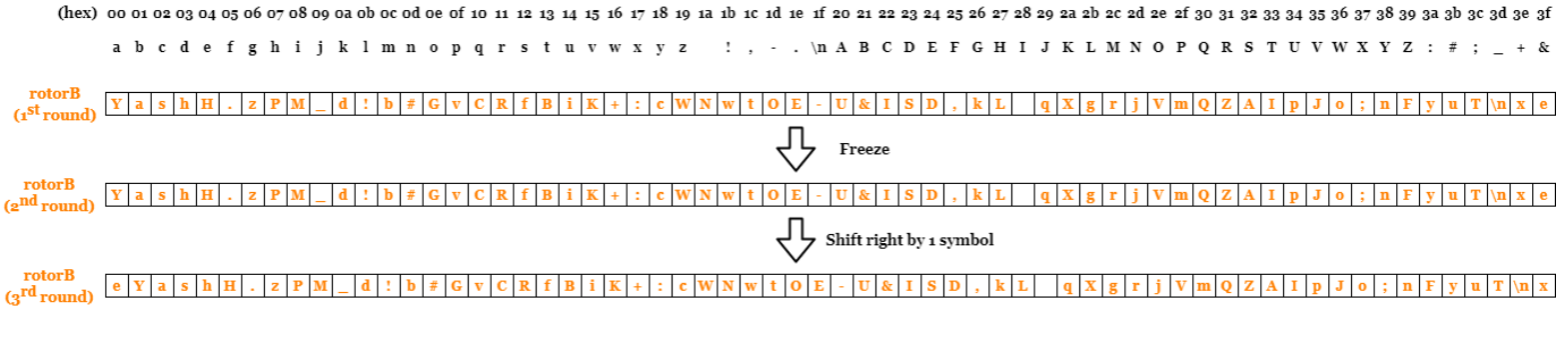


Fig. 4. Rotation of the rotorB.

This novel Enigma is improved by applying adaptive rotation and permutation for the rotorC. Fig.5 shows the two-stage adaptive permutation for the rotorC. The first-stage S-Box4 applies for each four contiguous symbols and has four possible modes as shown in Fig. 6. The least significant two bits of the 6-bit code of the output symbol of the rotorC are used as the selection signal. For example, the output symbol 'C' has the value 6'b100010, so the S-Box4 mode is 2 for the first round. The second-stage S-Box64 follows a fixed permutation as shown in Fig. 7.

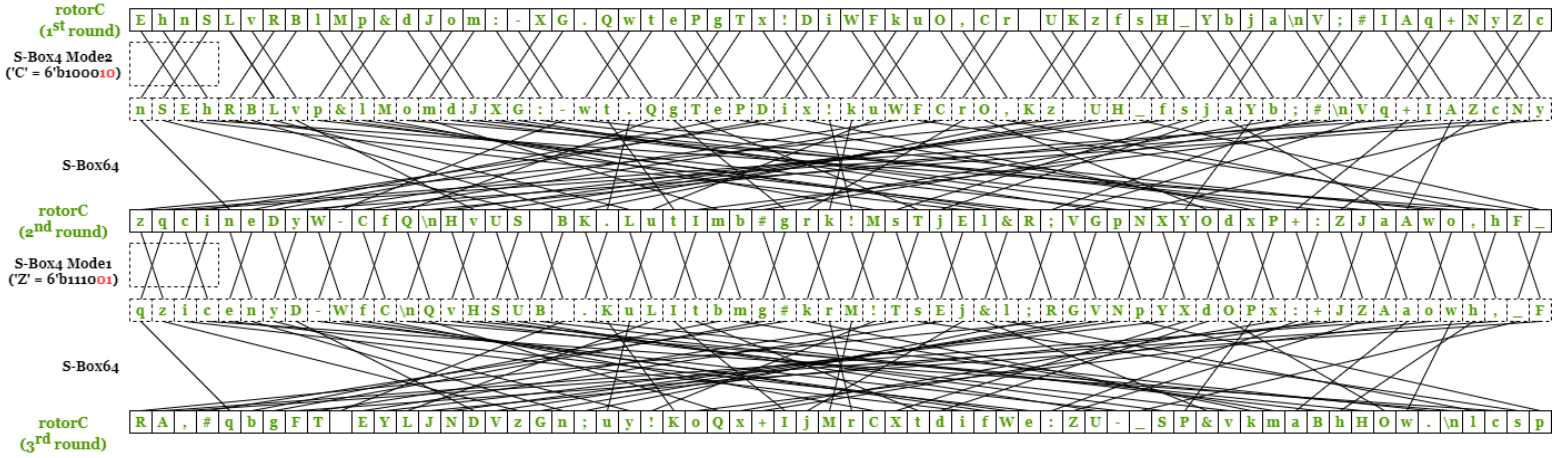


Fig. 5. Adaptive permutation of the rotorC

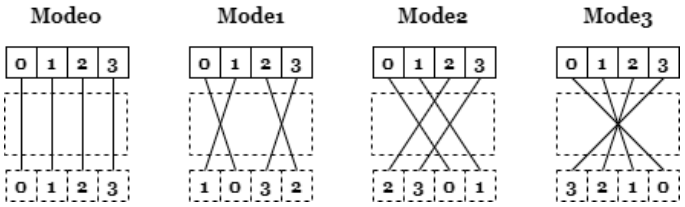


Fig. 6. Four modes of S-Box4 permutation

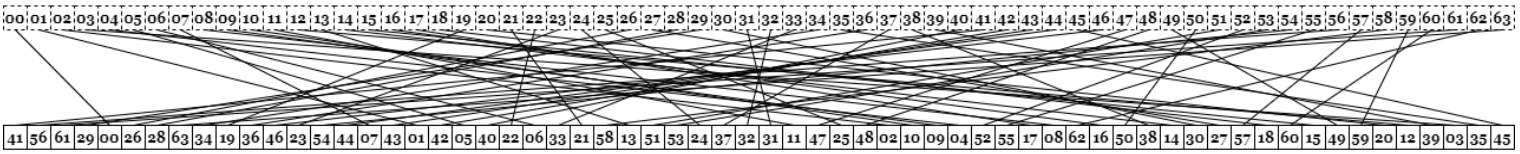


Fig. 7. Fixed S-Box64 permutation

For more examples, the encryption operations for two following symbols ' ' (space) and 'l' are given in Figs. 8 and 9 respectively. A property of Enigma is that it does decryption in the same way as encryption. For example, if you input 'n', 'f', and 'Q' to this Enigma sequentially and change the rotors accordingly, you will get 'I', ' ', and 'l' respectively from the outputs.

Note: Be careful of the adaptive selection signals you choose for the rotorC when the Enigma operates in the decryption mode.

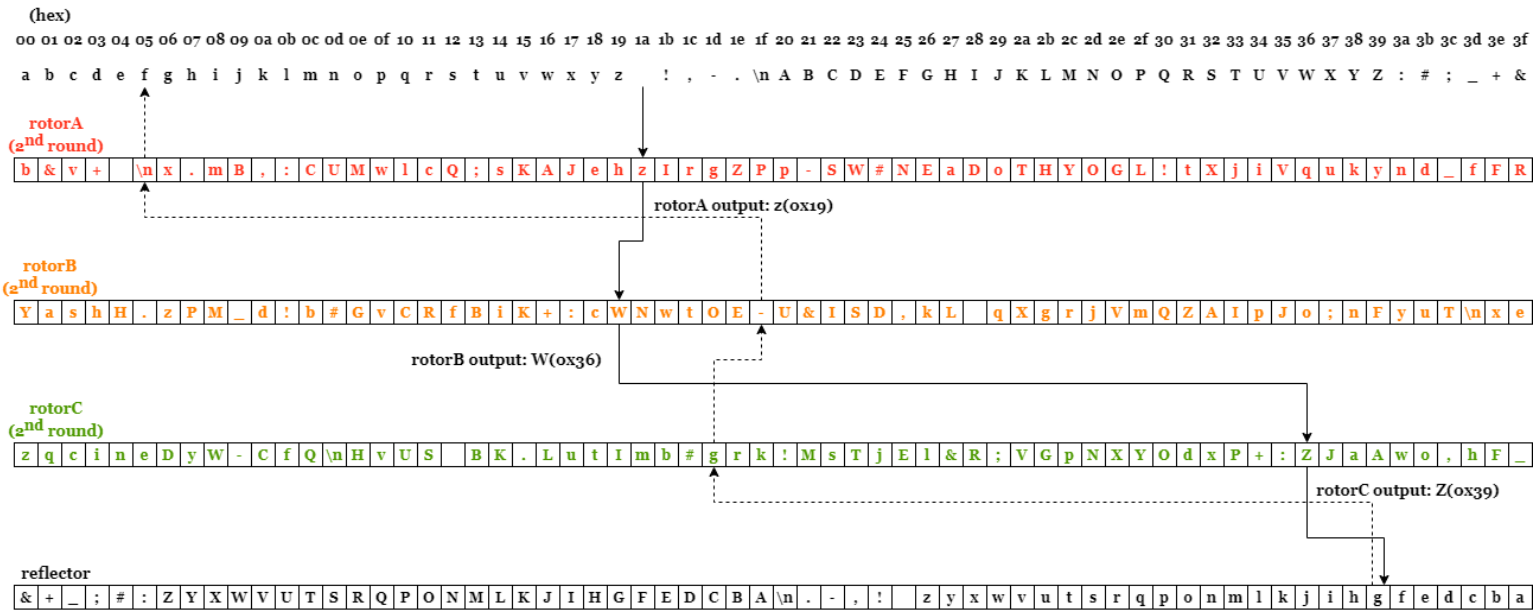


Fig. 8. Example of encryption for an input ‘ ’ (the second round)

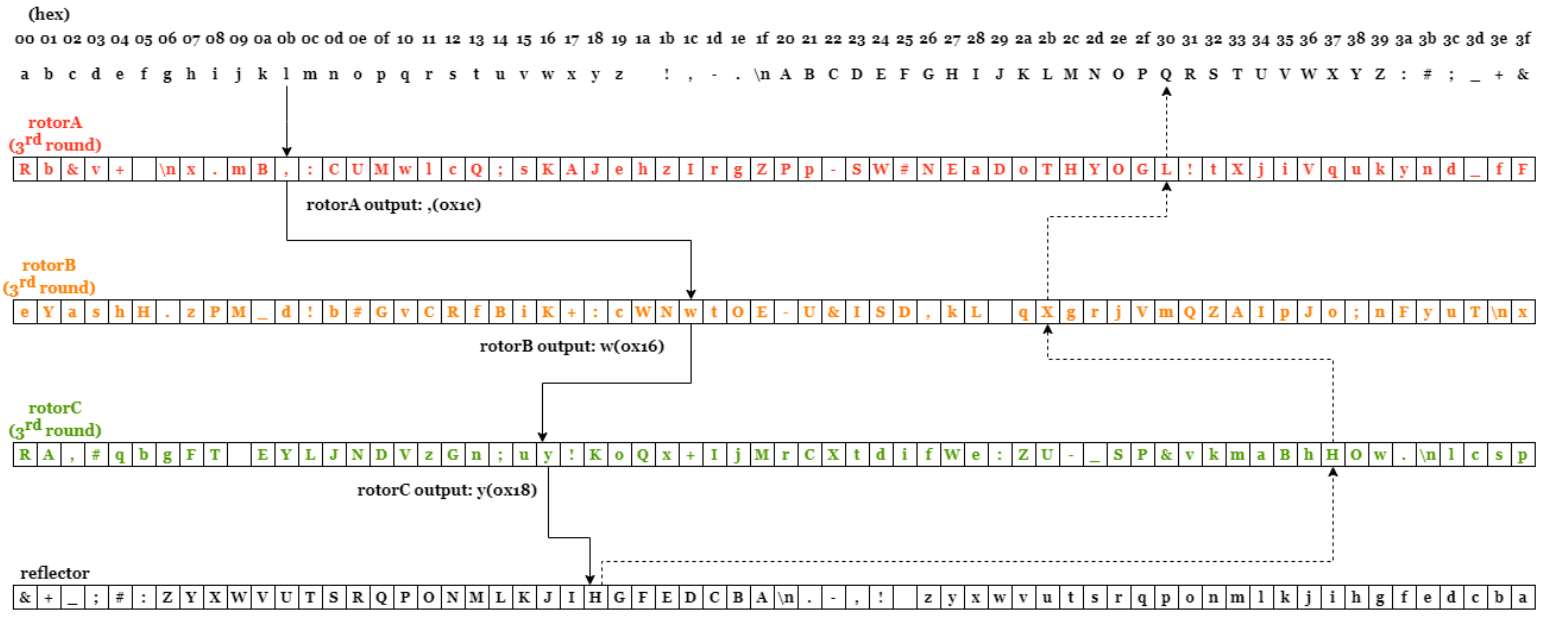
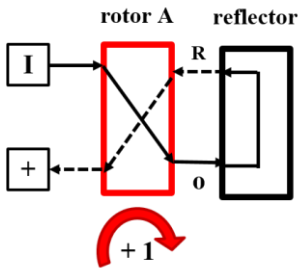


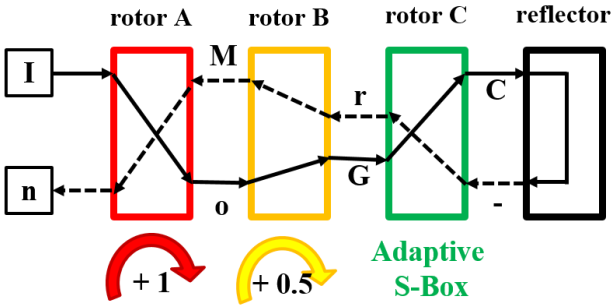
Fig. 9. Example of encryption for an input ‘l’ (the third round)



Step in this homework

Part1 (rotorA + reflector)

a. Given the behavior model, implement the testbench **test_enigma_part1.v** (1%)
b. Change the behavior model to your synthesizable RTL code **enigma_part1.v** (2%)

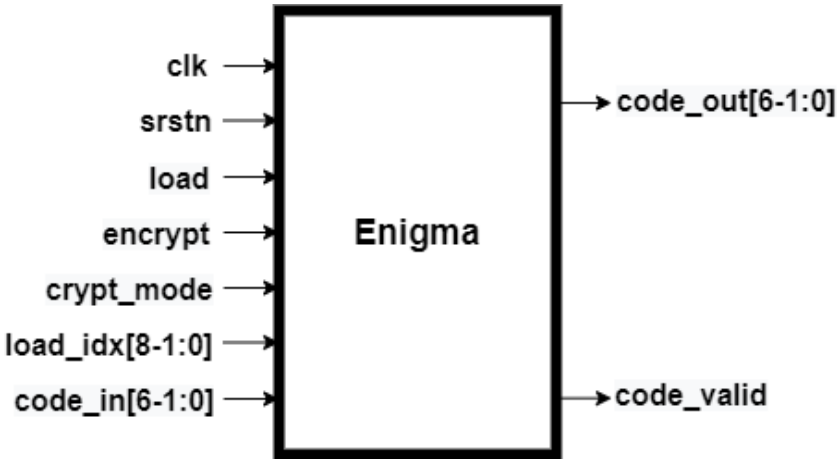


Part2 (rotorA + rotorB + rotorC + reflector)

a. Implement the complete enigma RTL code **enigma_part2.v** (4%)
b. Write the testbench **test_enigma_part2.v** for encrypting and decrypting (1%)
c. Write the testbench **test_enigma_display.v** to decrypt and save the results **in ASCII format** (2%)

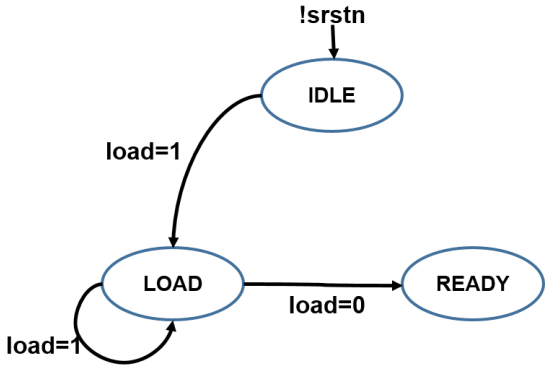
RTL I/O name definition

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



I/O name	Number of bits	Description
clk	1	clock input
srstn	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: inactive/active effective in READY state
crypt_mode	1	0: encrypt, 1:decrypt
load_idx	8	index of rotor table to be loaded (A: 0~63; B: 64~127; C: 128~191)
code_in	6	When load is active, rotorA[load_idx[5:0]] <= code_in if load_idx[7:6]==2'b00 rotorB[load_idx[5:0]] <= code_in if load_idx[7:6]==2'b01 rotorC[load_idx[5:0]] <= code_in if load_idx[7:6]==2'b10
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)

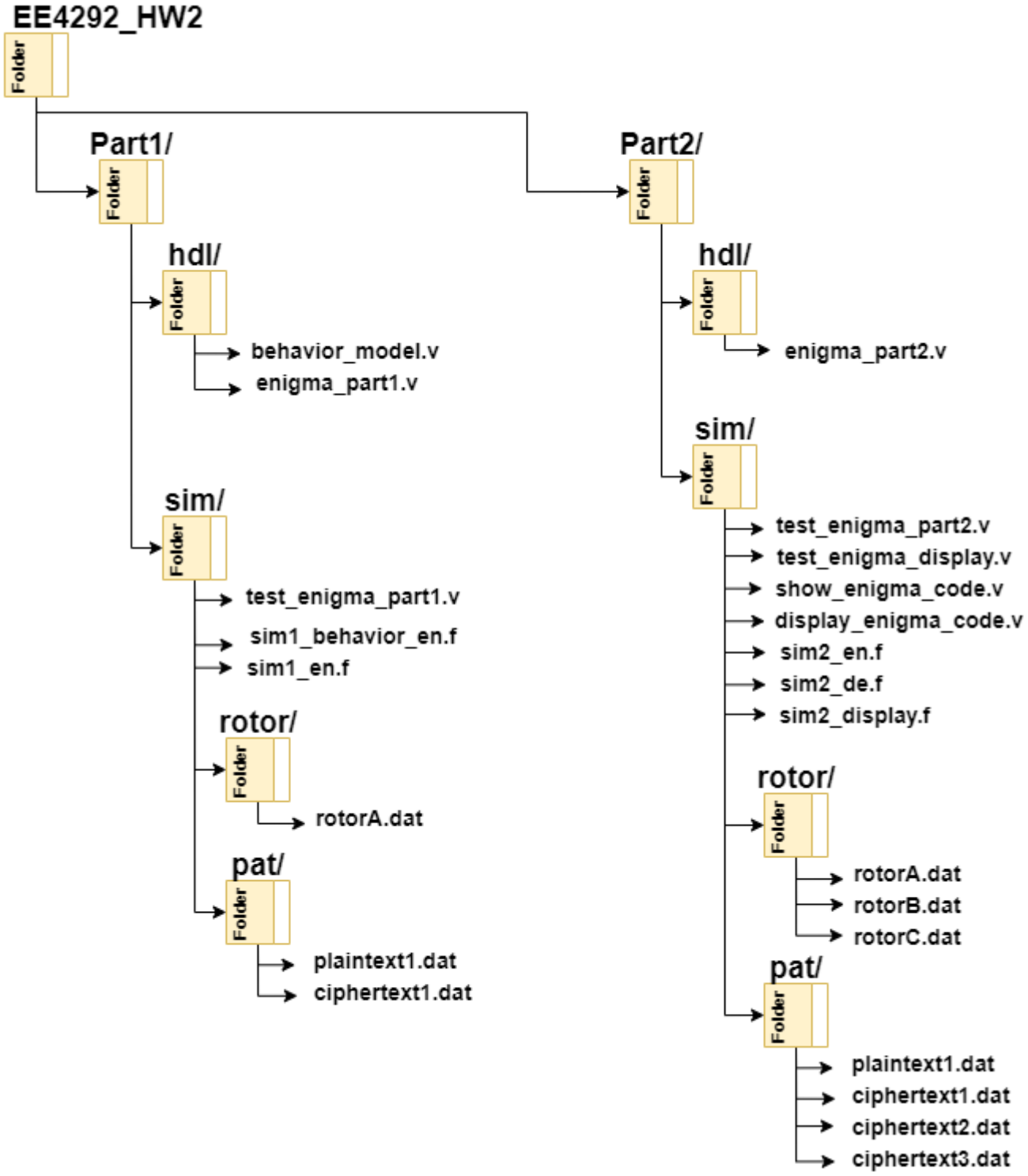
Your module shall also obey this finite state machine:



For encrypting a new plaintext, the srstn signal is used to reset the state to IDLE at first. Then the level sensitive load signal is used to initialize the rotor table by setting rotorA/B/C[load_idx[5:0]] = code_in with sequentially incremented load_idx. After 192 consecutive loading cycles, the state goes to READY after load is turned off. Then the

Enigma module performs the encryption and rotates the rotors whenever the signal encrypt is set high. (Hint: You can use a fixed look-up table for the reflector, and register files for the rotors. The rotorC permutation and the inverse rotors are the tricky parts.)

The following attached files are provided for this assignment:



Filename	Description
behavior_model.v	behavior function of one rotor Enigma
rotorA.dat, rotorB.dat, rotorC.dat	Hex files of the rotors for $readmemh
plaintext1.dat ciphertext1.dat	First plaintext “I love NTHU iclab!!! :D\n” The encrypted text for plintext1.dat
ciphertext2.dat ciphertext3.dat	The encrypted texts
display_enigma_code.v	Verilog tasks for displaying plaintext or ciphertext in ASCII format
show_enigma_code.v	Example of how to use display_enigma_code.v in your testbench. You can display plaintext1.dat by executing $ncverilog show_enigma_code.v +define+PLAINTEXT1

Grading policy

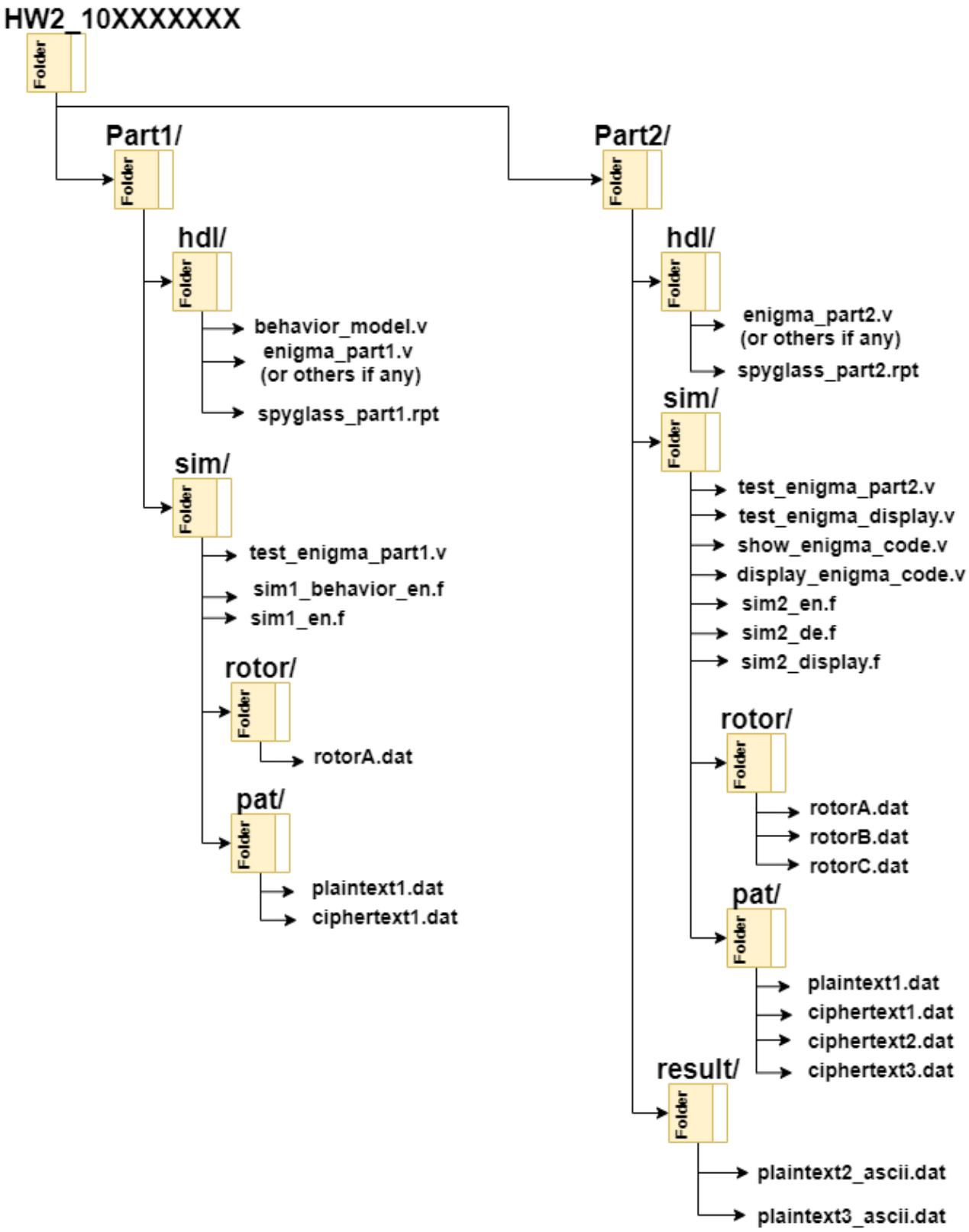
For this assignment, you need to turn in the following files:

1. Testbench test_enigma_part1.v (1%) for encrypting plaintext1.dat to ciphertext1.dat with the behavior_model.v
2. Functional module enigma_part1.v (2%) (or others if any) and its spyglass report file to show the synthesizability. (If it’s not synthesizable, you cannot get the score.)
3. Functional module enigma_part2.v (4%) (or others if any) for encrypt and decrypt mode, 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 (1%) for encrypting plaintext1.dat and also decrypting ciphertext1.dat for verification.
5. Testbench test_enigma_display.v (2%) 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).
   (you can refer to show_enigma_code.v and display_enigma_code.v)

Deliverable

Note: 1% punishment for wrong file delivery.

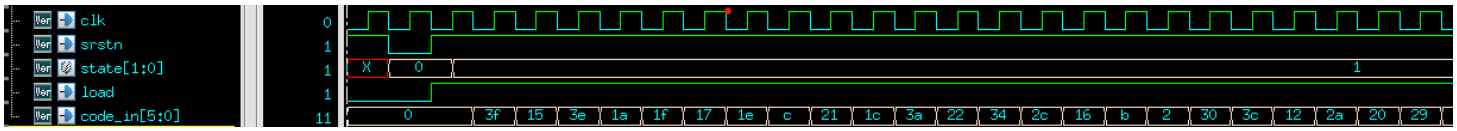
File Organization



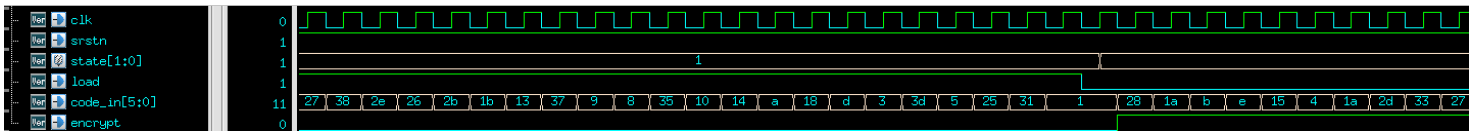
Directory	Filename	Description
HW2/part1/hdl/	behavior_model.v	Provided enigma behavior model
HW2/part1/hdl/	enigma_part1.v (or others if any)	Enigma with rotorA & reflector
HW2/part1/hdl/	spyglass_part1.rpt	Spyglass report of enigma_part1.v (or others if any)
HW2/part1/sim/	test_enigma_part1.v	Encrypt testbench
HW2/part1/sim/	sim1_behavior_en.f	Pre-sim file list for behavior_model.v (see .f file for more details)
HW2/part1/sim/	sim1_en.f	Pre-sim file list for enigma_part1.v (see .f file for more details)
HW2/part2/hdl/	enigma_part2.v (or others if any)	Enigma with rotorA & rotorB & rotorC & reflector
HW2/part2/hdl/	spyglass_part2.rpt	Spyglass report of enigma_part2.v (or others if any)
HW2/part2/sim/	test_enigma_part2.v	Encrypt and decrypt testbench
HW2/part2/sim/	test_enigma_display.v	Decrypt and save testbench
HW2/part2/sim/	sim2_en.f	Pre-sim file list for enigma_part2.v (see .f file for more details)
HW2/part2/sim/	sim2_de.f	Pre-sim file list for enigma_part2.v (see .f file for more details)
HW2/part2/sim/	sim2_display.f	Pre-sim file list for enigma_part2.v (see .f file for more details)
HW2/part2/result/	plaintext2_ascii.dat	Decrypted message – 2 (ASCII format)
HW2/part2/result/	plaintext3_ascii.dat	Decrypted message – 3 (ASCII format)

Referenced waveform (Modified in V2!!)

IDLE→LOAD



LOAD→READY



It is highly recommended to do it as early as possible.