NYCU-DCS-2025

OT

Design: Enigma Machine

Data Preparation

1. Extract files from TA's directory:

   % tar xvf ~DCSTA01/OT.tar

2. The extracted LAB directory contains:
   a. **00_TESTBED**
   b. **01_RTL**
   c. **02_SYN**
   d. **03_GATE**
   e. **09_SUBMIT**

Design Introduction

■ Enigma Machine

Enigma is the rotor cipher machine for encryption and decryption which was used by the German military in World War II.

■ Architecture and Flow

Fig. 1 shows the structure. It has been simplified for implementation easily. For every 6-bit input code (plaintext) it will generate one 6-bit output ciphertext through the two rotors and the reflector, and the inverse of them which all perform one-to-one substitution functions. The rotation or permutation of the rotors provides the security.

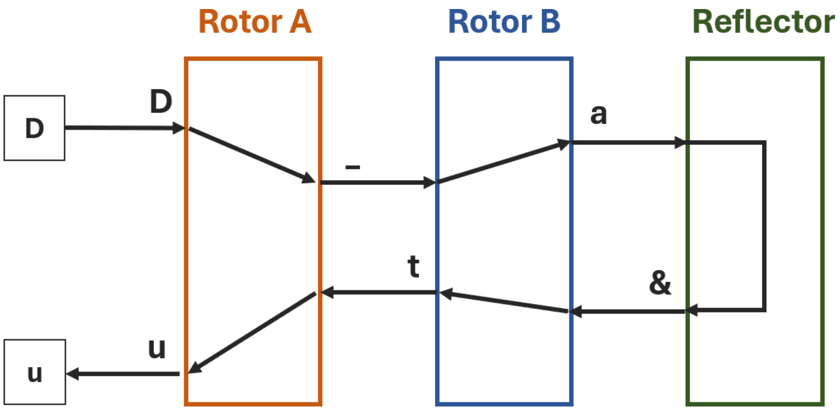


Fig. 1 Structure of the simplified enigma

Each 6-bit Enigma code has its corresponding ASCII symbol. Take Fig. 2 for example:

1. Input (plaintext) 6’h23 means “D”
2. Through the rotor A it becomes “_” (6’h3d)
3. Through the rotor B it becomes “a” (6’h00)
4. Through the reflector it becomes “&” (6’h3f) out of it.
5. Afterwards, keep performing transformation through rotors, which do exactly the opposite way.
6. Through the **inverse** rotor B it becomes “t”
7. Through the **inverse** rotor A it becomes “u”
8. Finally output “u” (6’h14) as ciphertext.

As for decryption, it has the same logic. Depicted in Fig. 3.

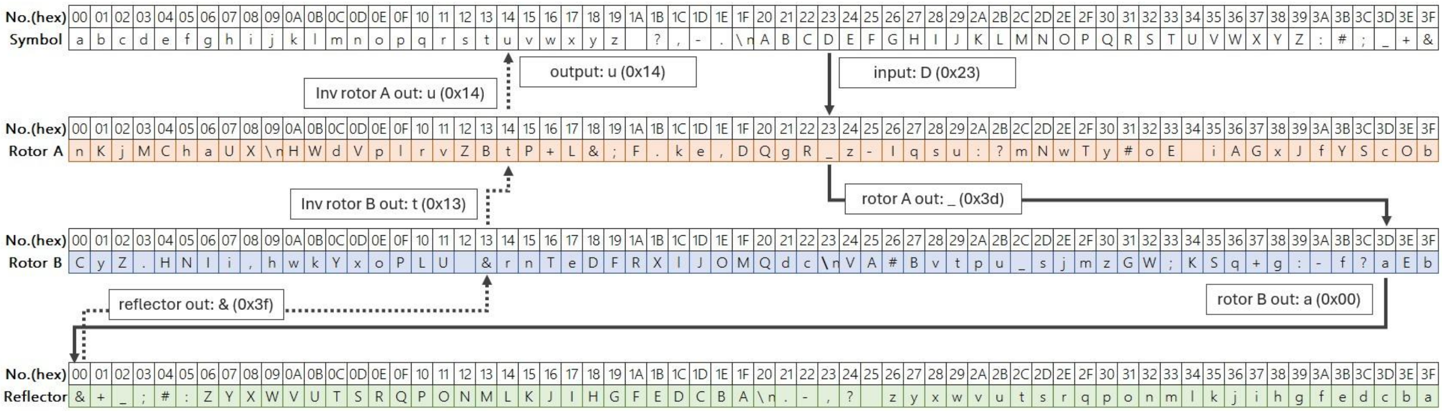


Fig. 2. Example of encryption for input “D”

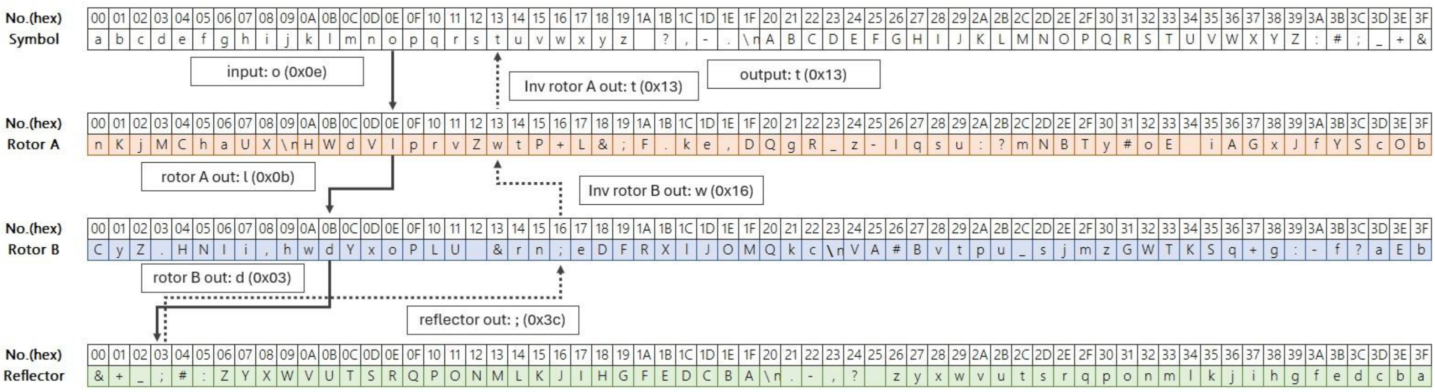


Fig. 3. Example of decryption for input “o”

■ How to transform symbol through the table

Here is the extra explanation for the example of Fig. 2. At the positive direction, when a symbol is input to the rotor and reflector, just recognize its value and look up the location of the table to get the output value. As for the inverse direction, vice versa. For example:

Input “D” (0x23)→rotor A has “_” (0x3d) at location 0x23→rotor A out = ”_”(0x3d)

Input “_” (0x3d)→rotor B has “a” (0x00) at location 0x3d→rotor B out = ”a”(0x00)

Input “a” (0x00)→reflector has “&” (0x3f) at location 0x00→reflector out = ”&”(0x3f)

Inverse direction:

Input “&” (0x3f)→rotor B location “t” 0x13 has “&” (0x3f)→reflector out = ”t”(0x13)….

■ Rule of the arrangement of rotors and the reflector

The rotors are Enigma’s core, performing a simple substitution cipher. Encrypting/Decrypting with the same permutation all the time would lead to identical outputs. Therefore, the permutation of rotors will change.

1. Rotor A

During the “in_valid” is high, the Rotor A receives its initial arrangement via the “code_in” port.

The permutation of rotor A should **shift right** by 0~3 symbols after encrypting one symbol. The rotation amount (i.e. shift num) is determined by **the least significant two bits** of the 6-bit code of the **rotor A output** symbol.

For instance, in the first round (process the first symbol), the rotor A output symbol is “_” represented as 6’b111101. Shift num is set to 1, then rotor A shifts right one symbol before encrypting the next symbol, as depicted in Fig. 4.

No.(hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Rotor A (1st round)	n	K	j	M	C	h	a	U	X	\n	H	W	d	V	p	l	r	v	Z	B	t	P	+	L	&	;	F	.	k	e	,	D

No.(hex)	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Rotor A (1st round)	Q	g	R	_	z	-	l	q	s	u	:	?	m	N	w	T	y	#	o	E		i	A	G	x	J	f	Y	S	c	O	b

rotor A out:
_ (6'b111101)

shift right by 1 symbol

No.(hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Rotor A (2nd round)	b	n	K	j	M	C	h	a	U	X	\n	H	W	d	V	p	l	r	v	Z	B	t	P	+	L	&	;	F	.	k	e	,

No.(hex)	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Rotor A (2nd round)	D	Q	g	R	_	z	-	l	q	s	u	:	?	m	N	w	T	y	#	o	E		i	A	G	x	J	f	Y	S	c	O

rotor A out:
l (6'b001011)

shift right by 3 symbol

No.(hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Rotor A (3rd round)	S	c	O	b	n	K	j	M	C	h	a	U	X	\n	H	W	d	V	p	l	r	v	Z	B	t	P	+	L	&	;	F	.

No.(hex)	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Rotor A (3rd round)	k	e	,	D	Q	g	R	_	z	-	l	q	s	u	:	?	m	N	w	T	y	#	o	E		i	A	G	x	J	f	Y

rotor A out:
.....

....

Fig. 4. Behavior of the rotor A

2. Rotor B

After Rotor A receives its initial arrangement for 64 cycles, Rotor B also receives its initial arrangement via the code_in ports when the “in_valid” is high. Like rotor A, rotor B has a different arrangement after each symbol encryption.

Fig. 5 illustrates how rotor B behaves. In the first round, the permutation is given by the input port, and change permutation for the next symbol. Fig. 6 lists a total of eight possible modes for rearranging. The rearranging format applies to every eight consecutive symbols. To decide which mode is applied to the next symbol, you need to look up the **least significant three bits** of the 6-bit code of the **rotor B output** symbol. For example, the rotor B output symbol “a” has the value 6’b000000, then use Mode 0 to arrange rotor B for the next round.

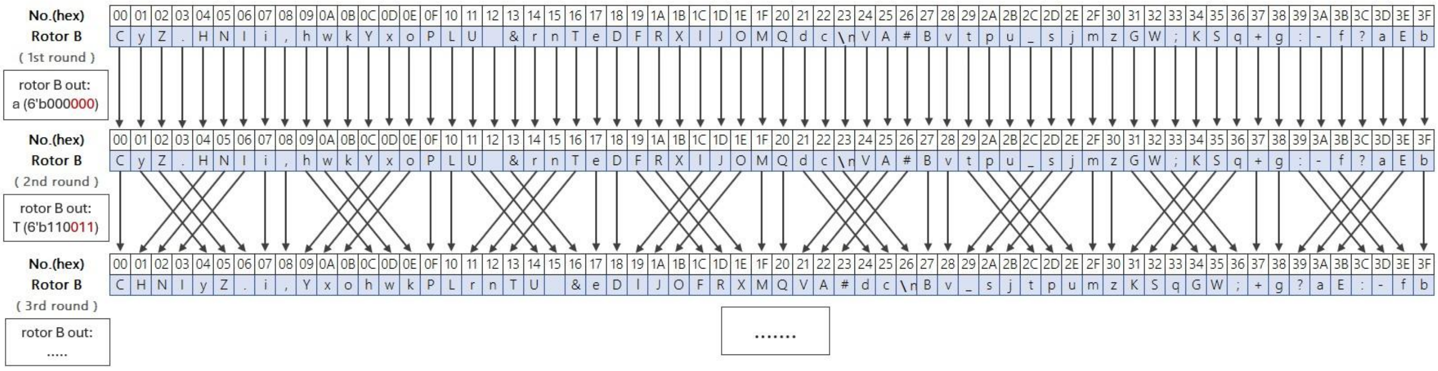


Fig. 5. Behavior of the rotor B

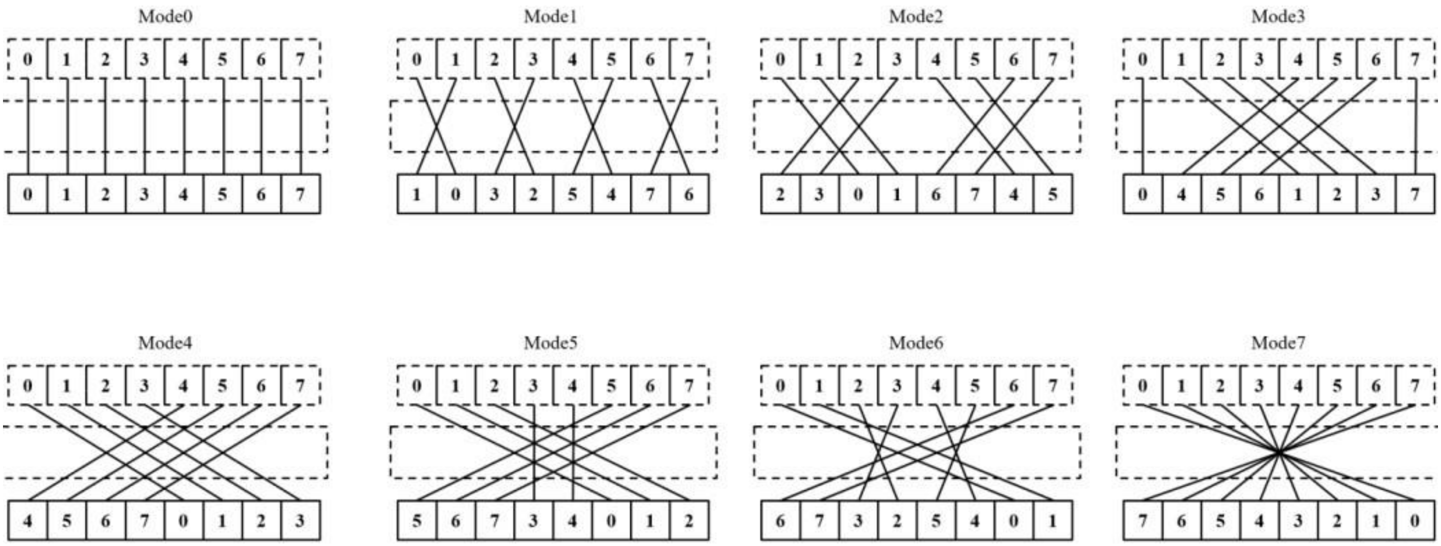


Fig. 6. Eight modes of S-Box 8 permutation

※ Note: When encryption, selection signals for permutation changing are rotor A output and rotor B output. But when decryption mode, selection signals for permutation changing are Inv rotor B output (for A) and reflector output (for B).

3. Reflector

The reflector is the final layer in our Enigma machine, remaining constant in both encryption and decryption processes. As shown in Fig. 6, the reflector is a **fixed** table that has the reverse order of the ASCII table.

No.(hex)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
Reflector	&	+	_	;	#	:	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

No.(hex)	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
Reflector	\n	.	-	,	?		z	y	x	w	v	u	t	s	r	q	p	o	n	m	l	k	j	i	h	g	f	e	d	c	b	a

Fig. 6. Fixed reflector

Inputs

Input	**Bit Width**	**Definition and Description**
clk	1	Clock.
rst_n	1	Asynchronous active-low reset.
in_valid	1	High when input signals are valid. Total high for 128 cycles Former 64 cycles code_in for initial rotor A Later 64 cycles code_in for initial rotor B
in_valid_2	1	High when codes to be processed are valid.
crypt_mode	1	0: encrypt. 1: decrypt. Only valid for 1 cycle when in_valid is active
code_in	6	When in_valid is active, then code_in is the input of rotors. When in_valid_2 is active, then code_in is the input of code words.

Outputs

Output	**Bit Width**	**Definition and Description**
out_valid	1	High when output is valid. Need to keep high for the number of input text cycles
out_code	6	Output code of encrypted/ decrypted code word. When the out_valid is low, the out_code should be 0.

Specifications

1. Top module name: ENIGMA (design file name: ENIGMA.v)
2. It is an asynchronous reset and active-low architecture. If you use synchronous reset (reset after the clock starts) in your design, you may fail to reset signals.
3. The reset signal (rst_n) would be given only once at the beginning of the simulation. All output signals should be reset to 0 after the reset signal is asserted.
4. Any words with "error", "latch" or "congratulation" can't be used as a variable name.
5. The execution latency is limited to **100 cycles**. The latency is the clock cycle between the rising edge of the first **in_valid_2** and the rising edge of the first **out_valid**. (half cycle would be treat as one cycle)
6. The clock period of the design is fixed at **10ns.**
7. The in_valid_2 will come in 2~5 cycles after in_valid pull-down.
8. The out_valid **cannot** overlap in_valid **but can** overlap in_valid_2.
9. The out_valid should be high until the whole text are output.
10. The next group of inputs will come in 2~5 cycles after your out_valid pull-down.
11. The synthesis result of the data type **cannot** include any latches.
12. The slack at the end of the timing report should be **non-negative (MET).**
13. Check out the log file in 03_GATE. The gate-level simulation **cannot** include any **timing violations**.

Notes

1. **Homework upload**
   (1). 1st_demo deadline: **2025/06/05(Thr.) 18:30:00**
   (2). 2nd_demo deadline: **2025/06/05(Thr.) 23:59:59**
   (3). 3rd_demo deadline: **2025/06/06(Fri.) 23:59:59**
2. **Grading policy**
   (1). RTL and gate-level simulation correctness: 100%
   (2). 1st_demo: 100
   (3). 2nd_demo: 80
   (4). 3rd_demo: 60
3. **Commands list**
   (1). 01_RTL/**./01_run_vcs_rtl**
   (2). 02_SYN/ **./01_run_dc_shell**

(Can execute 02_SYN/08_check for checking)

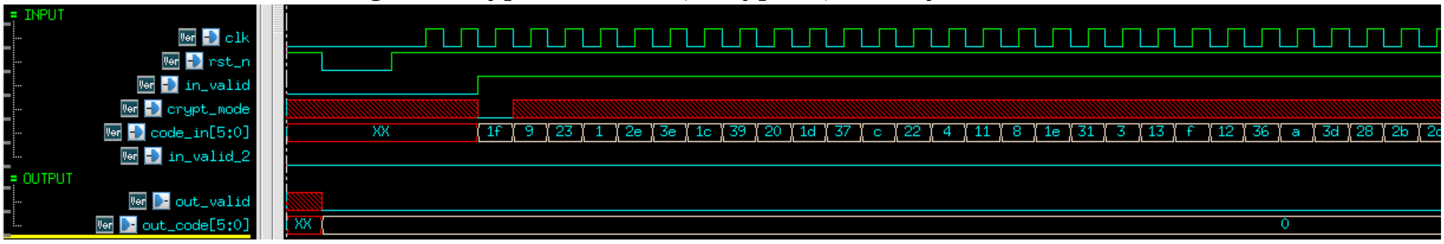
(Check if there is any **latch** in your design in **syn.log**)

(3). 03_GATE / (Gate-level simulation) **./01_run_vcs_gate**

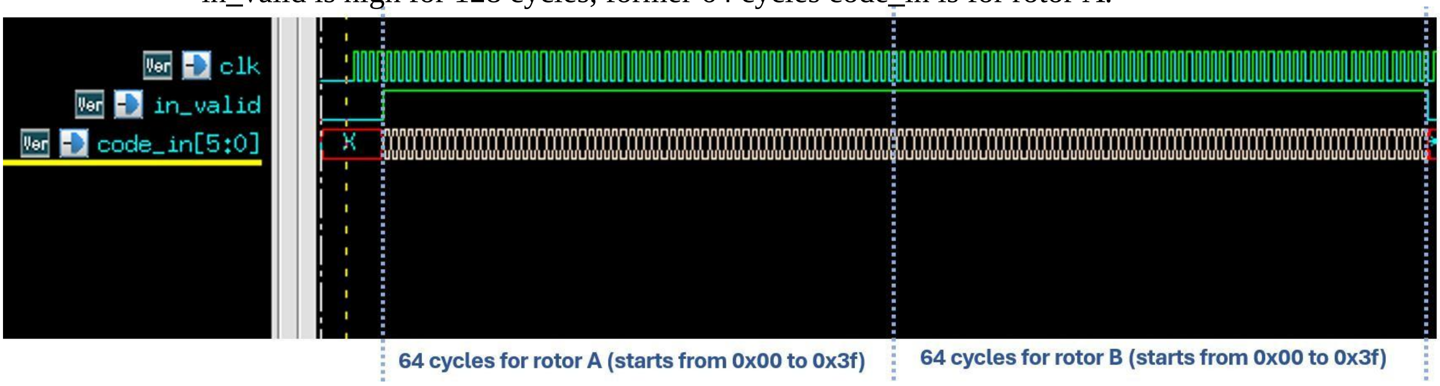
(4). 09_SUBMIT/ **./01_submit**

Example Waveform

- in_valid is high, the crypt_mode is 0 (encryption) for 1 cycle.



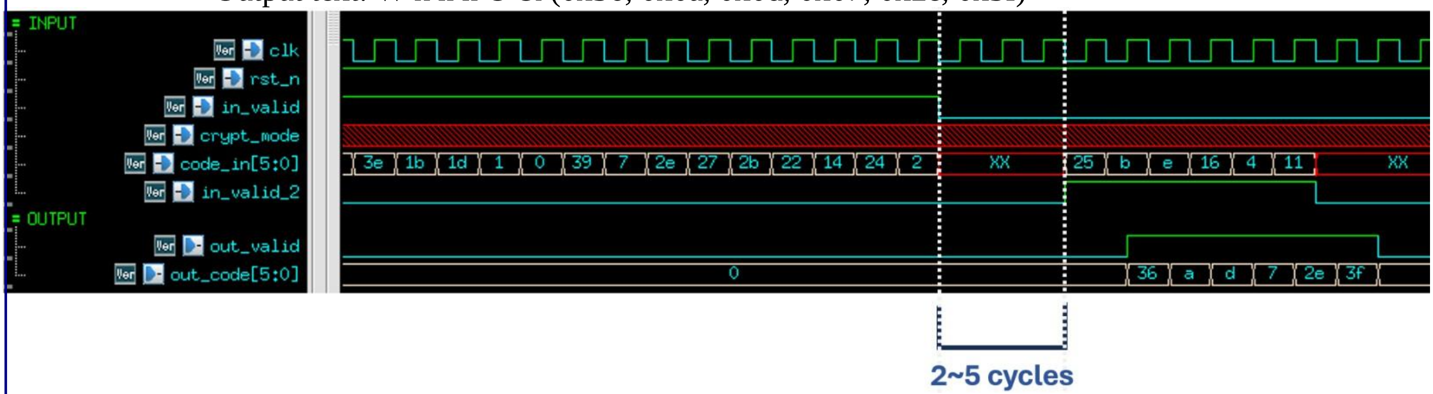
- in_valid is high for 128 cycles, former 64 cycles code_in is for rotor A.



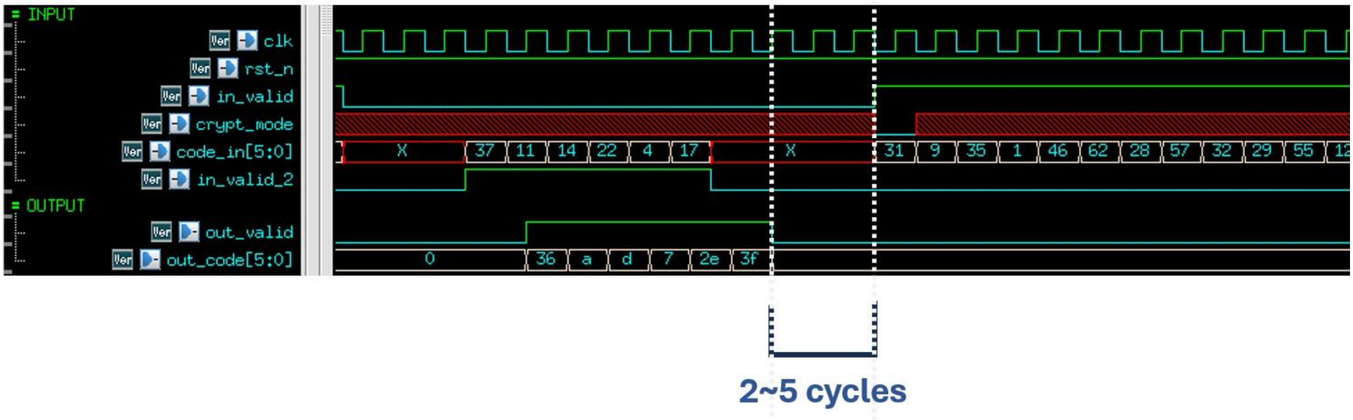
- Encryption: in_valid_2 is high and the output is ready

  Input text: F l o w e r (0x25, 0x0b, 0x0e, 0x16, 0x04, 0x11)

  Output text: W k n h O & (0x36, 0x0a, 0x0d, 0x07, 0x2e, 0x3f)



- Encryption: out_valid is low then the next pattern comes, the in_valid is high.



Hints for your reference

Here is the FSM design for your reference, you don't need to follow the design if you have a better idea. Good luck with everything!!!