NCTU-EE IC LAB - Spring 2022

Lab03 Exercise

Design: Escape

Data Preparation

1. Extract test data from TA's directory:
   % tar xvf ~iclabta01/Lab03.tar
2. The extracted LAB directory contains:
   a. **00_TESTBED**
   b. **01_RTL**
   c. **02_SYN**
   d. **03_GATE**

Design Description

A *Maze* is a path from an entrance to a goal.

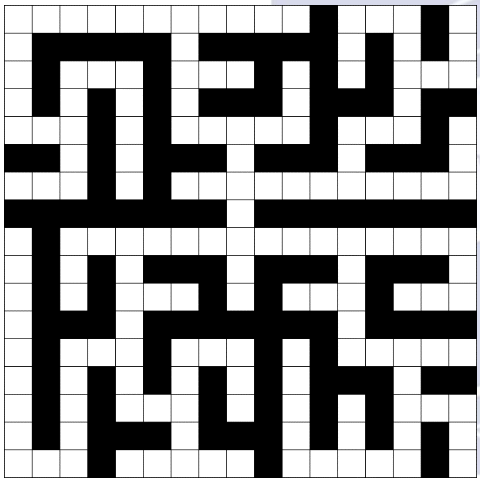


Fig 1. One example of *Maze*

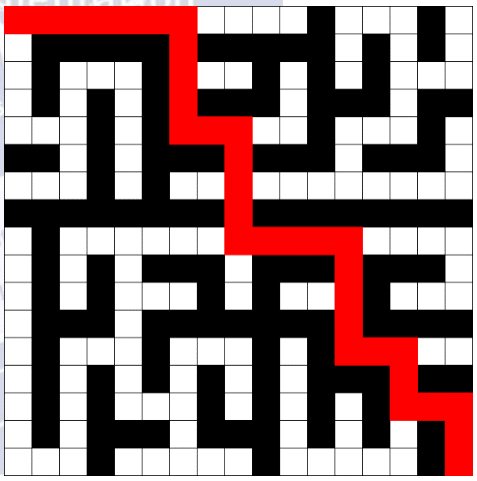


Fig 2. The solution

In this exercise, you need to control the people in the maze to go up, down, left or right. The goal is to spend as few cycles as possible to **rescue hostages** and **leave the maze** from starting point (upper left corner) to the finish point (bottom right corner) with less area. The gameboard is a **17*17 matrix**. The two-bit input is given 17*17 = 289 cycle continuously, from the top left corner to the bottom right corner in raster scan order (from first row left to right, then second row left to right……). Input 0 stands for walls, input 1 stands for paths, input 2 stands for traps and input 3 stands for hostages. To simplify the maze generating algorithm, **the walls will only exist in even row or even column, and only one path will exist from starting point to finish point (see last page for detail explanation)**. Feel free to search maze generate algorithm on the internet (ref. https://github.com/ferenc-nemeth/maze-generation-algorithms). For traps, there are **0 to 8 traps** in each maze. If your location is trap, you must output the direction **"Stall"** and can't move in one clock cycle. For hostages, there

are **0 to 4 hostages** in each maze. They will appear at the end of dead path. When the hostage is rescued, you will obtain a password. Your output should be up, down, left, right or stall depending on the situation. All of traps and hostages will not appear at the starting point and finishing point. Note that you can walk to a dead end then go back, as long as you escape from the maze in 3000 steps.

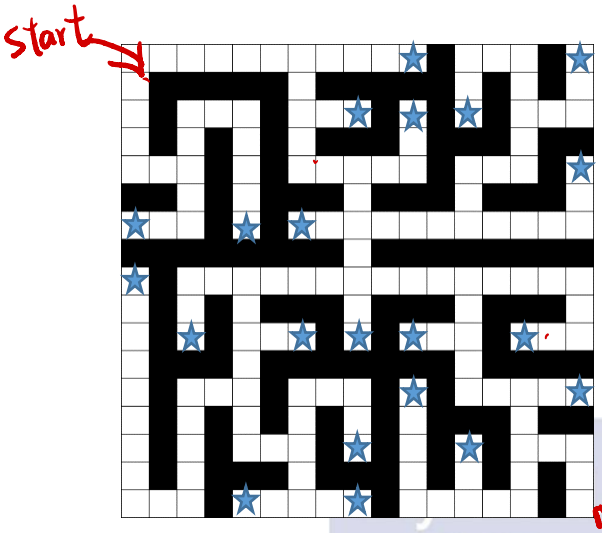


Fig 3. Possible location of *Hostage*

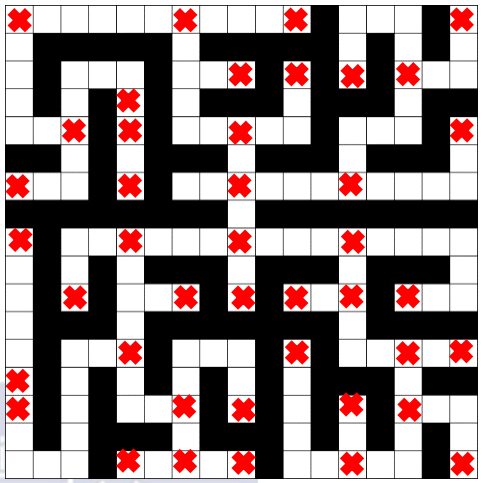


Fig 4. Constraint of traps

1. The trap would not appear at the end of dead path, starting point, finishing point.
2. There are no consecutive traps on the path.
3. Each trap would have two walls around it.

Decode Description

According to the number of hostages, there are different operation for decoding password.

Mode	Definition
Sorting	Sort the sequence from **the largest to the smallest** For example, {-100, 37, -164, 92, 42, -16}, becomes {92, 42, 37, -16, -100, -164}
Excess-3	If the number of hostages is even (not including 0): XS-3 **e.g.** 1: input = 0_1010_1100 represents 79 (decimal). **e.g.** 2: input = 1_0101_1000 represents -25 (decimal). If the number of hostages is odd: Normal signed decimal number **e.g.** 1: input = 0_1000_0001 represents 129 (decimal). **e.g.** 2: input = 1_0000_1101 represents -243 (decimal). ◆ Notice: • in_data has **9** bits. • in_data [3:0],[7:4] deliver 3 to 12 if the amount of hostages are even number.
Subtract half of range	If "the number of hostages is more than 1" indicates to subtract the half of range, which means **(max+min)/2** **e.g.** 1: original series: 12, 1, 5, -7 →max = 12, min = -7 →half of range = (12+(-7)) / 2 = 2 (Round down) →**After subtract the half of range: 10, -1, 3, -9** **e.g.** 2: original series: 7, 5, -5, -12 →max = 7, min = -12 →half of range = (7+(-12)) / 2 = -2 (Round down) →**After subtract the half of range: 9, 7, -3, -10**

Cumulation	If "the number of hostages is more than 2" indicates to do cumulation with the rule like moving average. The ratio of old value to new value is 2:1. **(round-down the answer if it is not integer)** **e.g.** original series: 9, 7, 0, -3 1st number of new series: (9 * 2 + 9 * 1) / 3 = 9 2nd number of new series: (9 * 2 + 7 * 1) / 3 = 8 (round down) 3rd number of new series: (8 * 2 + 0 * 1) / 3 = 5 (round down) 4th number of new series: (5 * 2 + -3 * 1) / 3 = 2 (round down) →After moving average: 9, 8, 5, 2

After doing these processes, you have to output a series of password when out_valid1 is high.

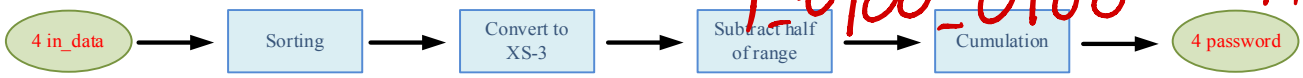
For example, if the number of hostages is 3, your design has to do Sorting, Excess-3, Subtract half of range, Cumulation, sequentially. And then, out_valid1 should maintain three clock cycles and out_data outputs three consecutive password. Especially, if the number of hostages is 0, you also need to output the password. The out_valid1 should maintain one clock cycle and out_data outputs 9'd0.

- Table is following excess-3 code for decimal digits:

Decimal	0	1	2	3	4	5	6	7	8	9
BCD code	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
					+ 0011					
Excess-3	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

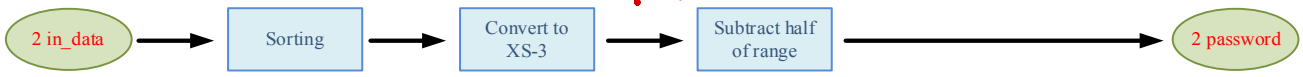
- Following are dataflow of some examples.

e.g. The number of hostages = 4



e.g. The number of hostages = 1

e.g. The number of hostages = 2



Inputs and Outputs

■ The following are the definitions of input signals

Input Signals	Bit Width	Definition
clk	1	Clock.
rst_n	1	Asynchronous active-low reset.
in_valid1 → 存 maze	1	High when "in" is valid.
in_valid2 → 拿 pwd.	1	High when hostage is rescued and in_data is valid.
in_data → password.	9	Password.
in ↓ 先 289 cycles 進入地图	2	2'd0: Wall; 2'd1: Path; 2'd2: Trap; 2'd3: Hostage;

■ The following are the definitions of output signals

Output Signals	Bit Width	Definition
out_valid1 → out data.	1	High when out_data is valid and finish the maze. It should maintain the corresponding cycles.
out_valid2 → out path	1	High when out is valid. (Start running the Maze) Refer the following 5.
out_data	9	Operation result for password data.
out	3	Direction Right : 3'd 0; Down: 3'd 1; Left : 3'd 2; Up : 3'd 3; Stall : 3'd 4;

1. The input signal **in** is delivered in **raster scan order** for **289 cycles** continuously. When in_valid is low, in should be tied to unknown state.
2. All input signals are synchronized at negative edge of the clock.
3. The output signal **out** must be delivered with **out_valid2** high.
4. The output signal **out_data** must be delivered with **out_valid1** high.
5. The output signal out_valid2 will be high during finding the hostages and the exit. When you find out the hostage, the output signal out_valid2 should be low and wait for the high of in_valid2. TA's Pattern will check your current location in this moment.
6. The priority is that you have to rescue all of hostages and collect the password. And then, going to exit and delivering the decoding password.
7. The next round of the game will come in **2~4 negative edge of clock** after your **out_valid1** is pulled down. (The new maze will be delivered)

Specifications

1. Top module name: ESCAPE (design file name: ESCAPE.v)

2. It is asynchronous reset and active-low architecture. If you use synchronous reset (considering reset after clock starting) in your design, you may fail to reset signals.
3. **The reset signal (rst_n) would be given only once at the beginning of simulation. All output signals should be reset after the reset signal is asserted.**
4. **The out should be reset after your out_valid2 is pulled down.**
5. **The out_valid1, out_valid2 should not be high when in_valid1 or in_valid2 is high. The out_valid1 and out_valid2 should not be high at the same time.**
6. **The execution latency is limited in 3000 cycles. The latency is the clock cycles between the falling edge of the last in_valid1 and the rising edge of the out_valid1. (Without adding the cycle of waiting for in_valid2)**
7. **The out should be correct when out_valid2 is high. (Including the trap)**
8. **When pull down the out_valid2, the location of controller should be in the location of hostage or the exit.**
9. **The out_valid1 should maintain the corresponding clock cycles.**
10. **The out_data should be correct when out_valid1 is high.**
11. **The out_data should be reset after out_valid1 is pulled down.**
12. The clock period is **15 ns,** because this exercise's main topic is verification pattern, you don't need to modify timing constraint.
13. The input delay is set to **0.5*(clock period)**.
14. The output delay is set to **0.5*(clock period)**, and the output loading is set to **0.05**.
15. The synthesis result of data type **cannot** include any **latches**.
16. The gate level simulation cannot include any timing violations without the *notimingcheck* command.
17. After synthesis, you can check ESCAPE.area and ESCAPE.timing. The area report is valid when the slack in the end of timing report should be **non-negative (MET)**.
18. The performance is determined by **area** and **latency**. The lower, the better.

Grading Policy

1. Function Validity: 50% (The grade of 2nd demo would be **30% off**.)
2. Test Bench: 30% (The grade of 2nd demo would be **100% off**.)
   - SPEC 3: 2%
   - SPEC 4: 2%
   - SPEC 5: 2%
   - SPEC 6: 2%
   - SPEC 7: 8%
   - SPEC 8: 5%
   - SPEC 9: 2%
   - SPEC 10: 5%
   - SPEC 11: 2%
   - ✧ **SPEC 3~11 means the third to eleven specification above**

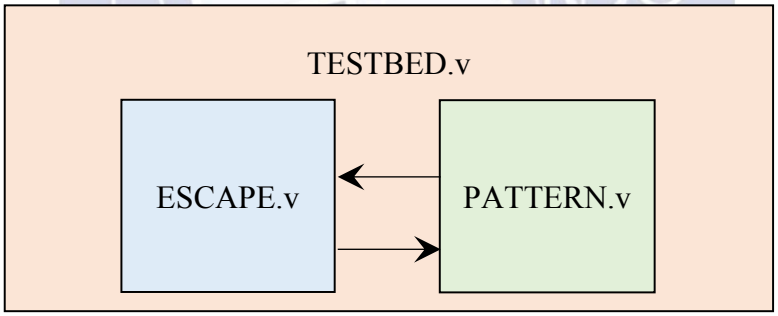
-24

- ✧ **Note that in SPEC 7, correct output means (1) the controller goes from starting point to the finish point and rescues the hostage without hitting the wall. (2) The controller can go back and forth on the same path multiple times. (3) If the controller is trapped, out should be 3'd4. (4) The out_data should be 0.**
- ✧ **Make sure that your design and pattern are robust.**
- ✧ **You don't have second chance for test bench demo. Only one-time demo.**
- ✧ **The number of your patterns can't over 500. (Include all of condition)**
- ✧ **If any spec is violated, you have to show "SPEC X IS FAIL!" on your screen.**
  - ■ **X is the number of the spec.**
  - ■ **Please follow this rule "SPEC X IS FAIL!" when spec is violated or you will lose points in demo. (SPEC3~6,9,11: 1%, SPEC7,8,10: 3%)**

```
**************************************************************
*                        SPEC 3 IS FAIL!                     *
*  Output signal should be 0 after initial RESET  at     2000   *
**************************************************************
```

3. Performance: 20%
   - ● Total latency*Area: 20%
   - ✧ **You will get this part of points only if you pass TA's pattern.**
   - ✧ **The grade of 2nd demo would be 30% off.**

Block diagram



Note

1. **Please upload the following files on new e3 platform before 23:59 on Mar. 20:**
   - ESCAPE_iclab???.v PATTERN_iclab???.v input_iclab???.txt
   - output_iclab???.txt(Optional)
2. **Template folders and reference commands:**

   01_RTL/ (RTL simulation) **./01_run**

   02_SYN/ (Synthesis) **./01_run_dc**

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

   (Check the timing of design in /Report/ ESCAPE**.timing**)

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

3. In this lab, you need to write a pattern file. You may use random system task or high-level language with IO to generate patterns. However, if you use the second approach, you also **need to submit the txt files** you used in your pattern file. The file path should be "../00_TESTBED/input.txt" or "../00_TESTBED/output.txt", which means that we will put your files into 00_TESTBED for demo. If the demo is failed due to file path, we will punish on the score. If the uploaded file violating the naming rule, you will get 5 deduct points on this Lab.

Sample Waveform

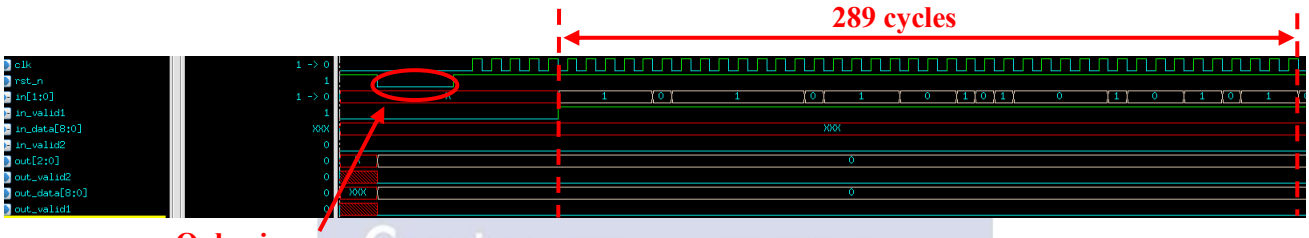


Fig1. Input waveform

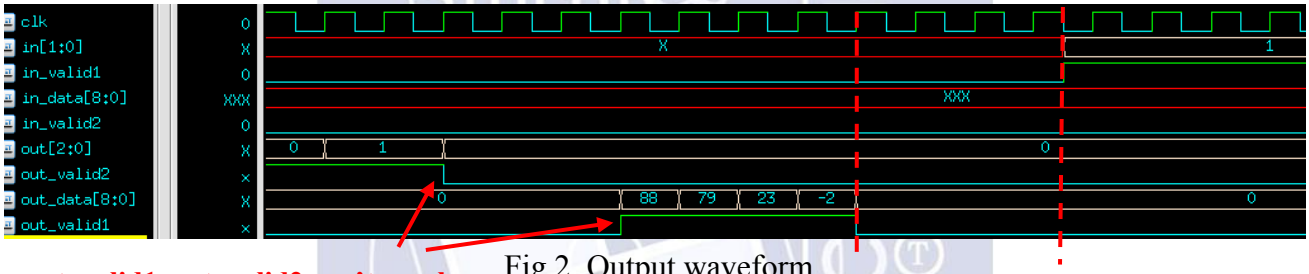


Fig 2. Output waveform

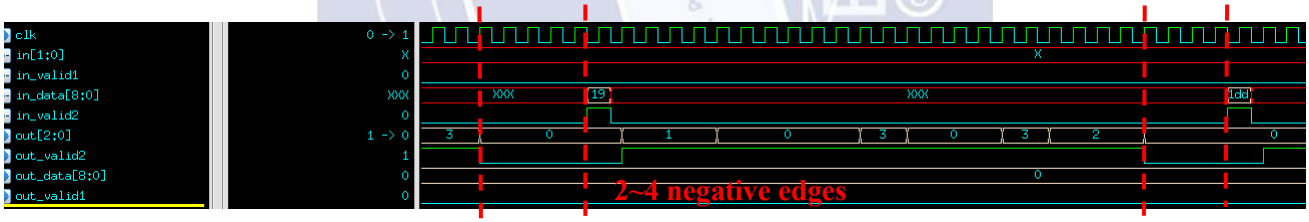


Fig 3. in_valid2, in_data waveform

Fig 4. Some Maze examples

Maze Constrain

Because the walls will block the path, it is eazier to limit the places where walls can exist.
Fig 4. and Fig5. are example of the maze generated with and without the constrian

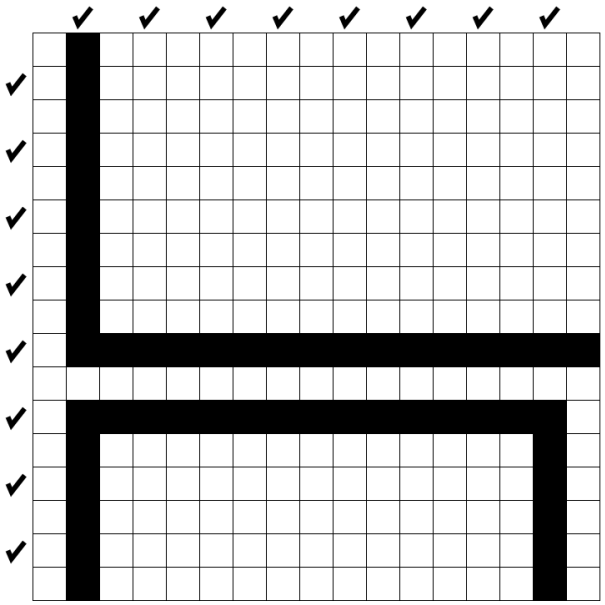


Fig 4. With constrain

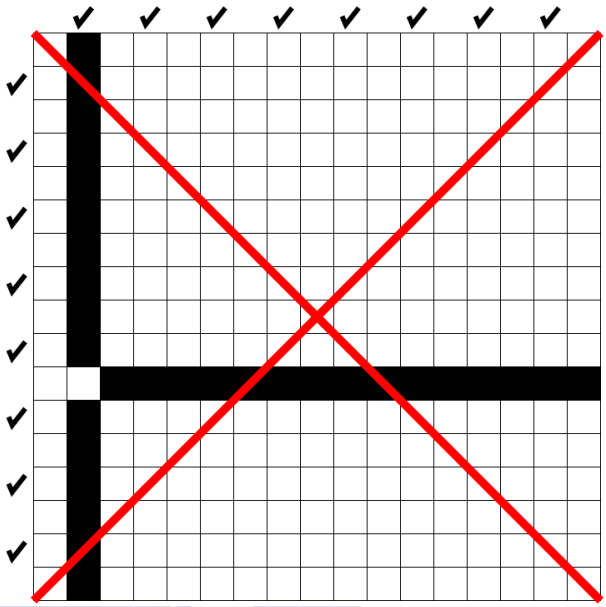


Fig 5. Without constrain

In general, the walls **can not** exist in the white block in Fig 6.

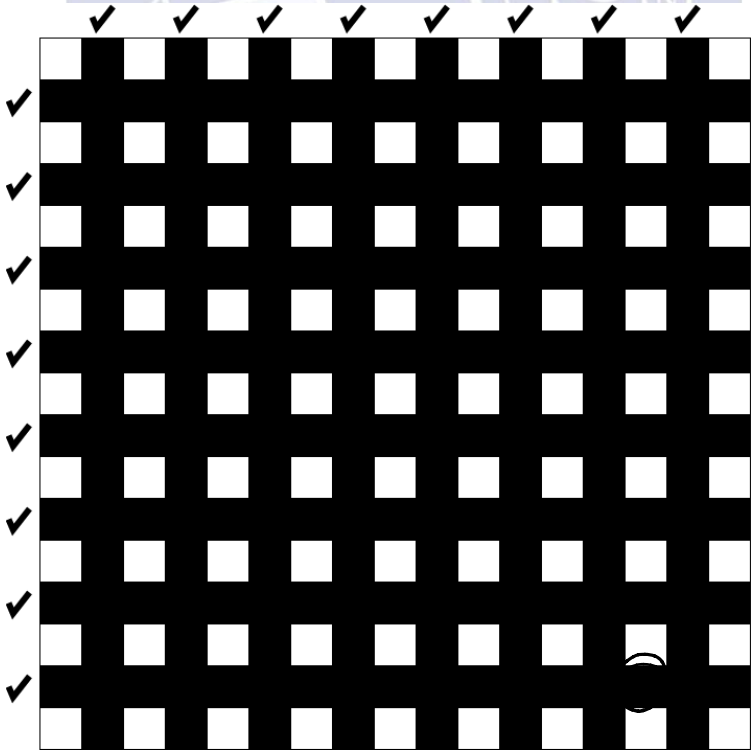


Fig 6. Where the walls can/cannot exist

If you use the maze generate algorithm given in reference (ref. https://github.com/ferenc-nemeth/maze-generation-algorithms), you will generate the right maze.

Encrypt Pattern

>> ncprotect -autoprotect PATTERN.v

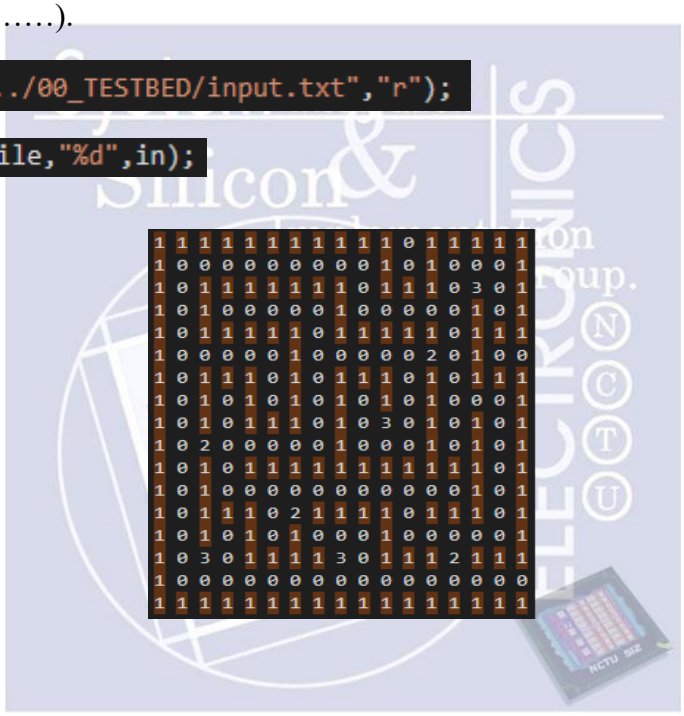
After entering this instruction, your PATTERN.v will be encrypted to PATTERN.vp

Note: If you want to share your pattern to your classmates, make sure that to do this step for avoiding the copy issue. Also, don't upload the PATTERN.v which is encrypted to E3 platform. You will loss all of points in Test Bench part.

Input Pattern

About how to use input.txt in pattern, you can follow this code. **In Lab3, the content of input.txt is only the diagram of maze.** The two-bit input **"in"** is given 17*17 = 289 cycle continuously, from the top left corner to the bottom right corner in raster scan order (from first row left to right, then second row left to right……).

```
input_file=$fopen("../00_TESTBED/input.txt","r");
a = $fscanf(input_file,"%d",in);
```



① 先將沒有 hostage 的路封起來

② 有 hostage 的用 hostage path 封起來

③ 先走 right down left up

maze 0: wall 3: hostage

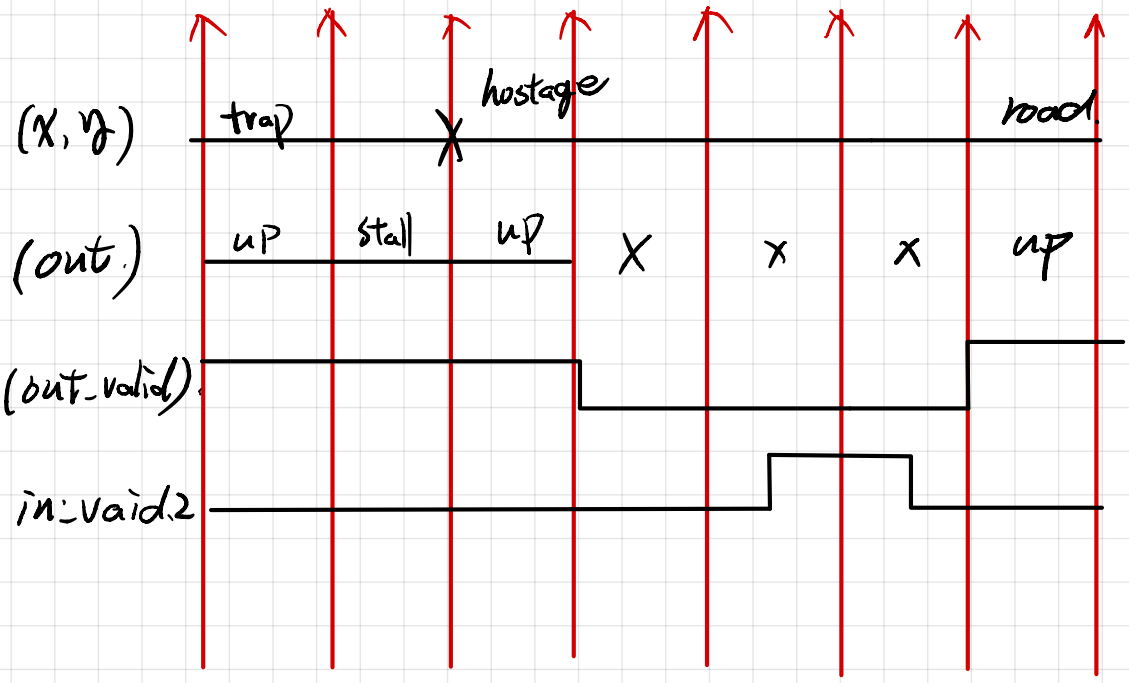
1: path 4: hostage path

2: trap 5: 走過

6 走過气回去

5 走過头回去.

4. 人质 + trap.



Now in hostage road

hostage.

hostage road back,

main road.

优先 ① if (x, y) = 1 or 2 ⇒ look 1 or 2 ⇒ chang to 5

look 3 or 4

② 干 (x, y) = 3 or 4 ⇒ look 3 or 4 ⇒ change to 6

③ 干 (x, y) = 6 ⇒ look 6 or 4 ⇒ change to 5

④ 干 (x, y) = 5 ⇒ look 6 or 4 ⇒ change to 5

判段転角