# NCTU-EE IC LAB - FALL 2021

## Lab03 Exercise

## Design: Maze

### 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.

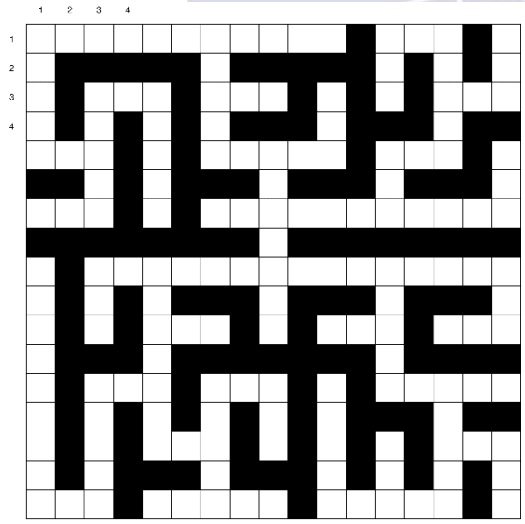


Fig 1. One example of *Maze*

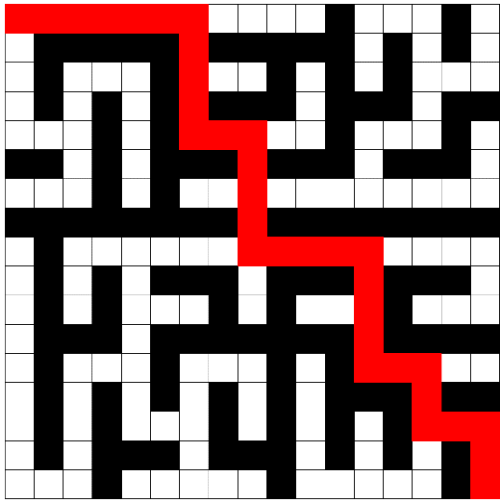


Fig 2. The solution

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 go from starting point (upper left corner) to the finish point (bottom right corner) with less area. The gameboard is a **17*17 matrix**. The one-bit input is given 17*17 = 289 cycle continuously, from the top left corner to the bottom right corner in **raster scan order** (from 1st row left to right, then 2nd row left to right……). Input 0 stands for walls and input 1 stands for paths. 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). Your output should be up, down, left or right depending on the situation. Note that you can walk to a dead end then go back, as long as you finish the maze in 3000 steps.

## 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_valid | 1 | High when input is valid. |
| in | 1 | High stands for path, and Low stands for wall |

■ The following are the definitions of output signals

| Output Signals | Bit Width | Definition |
|---|---|---|
| out_valid | 1 | High when out is valid. |
| out | 2 | Right: 2'd0; Down: 2'd 1; Left: 2'd 2; Up: 2'd 3 |

1. The input signal **in** is delivered in **raster scan order** for **289 cycles** continuously. When in_valid is low, in would 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_valid** high.
4. The next round of the game will come in **2~4 negative edge of clock** after your **out_valid** is pulled down. (The new maze will be delivered)
5. All operations are **unsigned**.

## Specifications

1. Top module name: MAZE (design file name: MAZE.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 to 0** after the reset signal is asserted.
4. The out should be set to 0 after your out_valid is pulled down.
5. **The out_valid should not be high when in_valid is high.**
6. The execution latency is limited in **3000 cycles.** The latency is the clock cycles between the falling edge of the last in_valid and the **falling edge** of the out_valid.
7. **The out should be correct when out_valid is high.**
8. The clock period is **10 ns,** because this exercise's main topic is verification pattern, you don't need to modify timing constraint.
9. The input delay is set to **0.5*(clock period)**.
10. The output delay is set to **0.5*(clock period)**, and the output loading is set to **0.05**.
11. The synthesis result of data type **cannot** include any **latches**.
12. The gate level simulation cannot include any timing violations without the *notimingcheck* command.
13. After synthesis, you can check MAZE.area and MAZE.timing. The area report is valid when the

slack in the end of timing report should be **non-negative (MET)**.

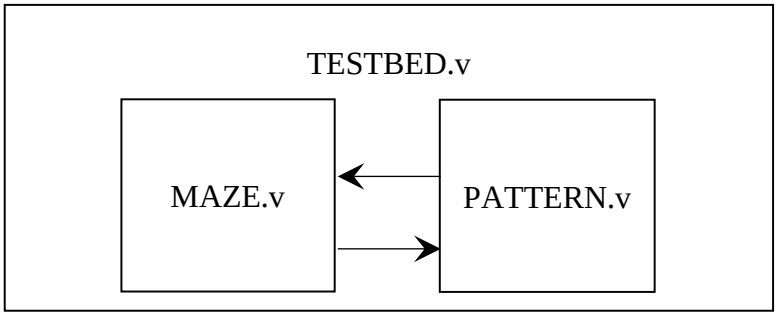
14. The performance is determined by **area** and **latency**. The lower, the better.

## Grading Policy

1. **DESIGN** Function Validity: 50% (The grade of 2nd demo would be **30% off**.)
2. **DESIGN** Performance: 20% (The grade of 2nd demo would be **30% off**.)
   - Total latency*Area: 20%
   - ✧ **You will get this part of points only if you pass TA's pattern.**
3. **PATTERN** Test Bench: 30% (The grade of 2nd demo would be **100% off**.)
   - SPEC 3: 3%
   - SPEC 4: 3%
   - SPEC 5: 3%
   - SPEC 6: 3%
   - SPEC 7: 18% (6%+6%+6%)
   - ✧ SPEC 3~7 means the third to seventh **specification above**
   - ✧ Note that in SPEC 7, correct output means that
     - ✓ (1) The controlled people goes from starting point to the finish point without hitting the wall (6%).
     - ✓ (2) The controlled people can go back and forth on the same path multiple times (6%). (should not print SPEC 7 FAIL)
     - ✓ (3) Must have stimulus that the shortest path is at least 64 steps (6%). (if the stimulus is randomly generated, this specification should be met easily)
   - ✧ You don't have second chance for test bench demo, it's served only one demo shot.
   - ✧ The number of your patterns can't over 300 test cases.
   - ✧ **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: 3%, SPEC7: 6%, 6%, 6%)**

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

## Block diagram



## Note

1. **Please upload the following files on new e3 platform before 12:00 p.m. on Oct. 11:**
   - MAZE _iclab???.v and PATTERN_iclab???.v (input_iclab???.txt, output_iclab???.txt)
   - ( ??? means your iclab account )
   - Ex: MAZE_iclab099.v, PATTERN_iclab099.v(input_iclab099.txt, output_iclab099.txt)
   - input_iclab???.txt and output_iclab???.txt is optional, only if you use it in pattern, you have to submit it
   - note that the file name should be "input_iclab???.txt" and "output_iclab???.txt", TA will copy the file and rename it to "input.txt" and "output.txt" and put it into the folder 00_TESTBED, so your pattern should include the txt file using file path "../00_TESTBED/input.txt", "../00_TESTBED/output.txt",
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/ MAZE**.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 later approach, you also **need to submit the txt files** you used in your pattern file. **The file path should be "../00_TESTBED/input.txt", "../00_TESTBED/output.txt"**, which means that we will put your files into 00_TESTBED for demo. Note that only txt files is available. **If the demo is failed due to file path or file type, 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

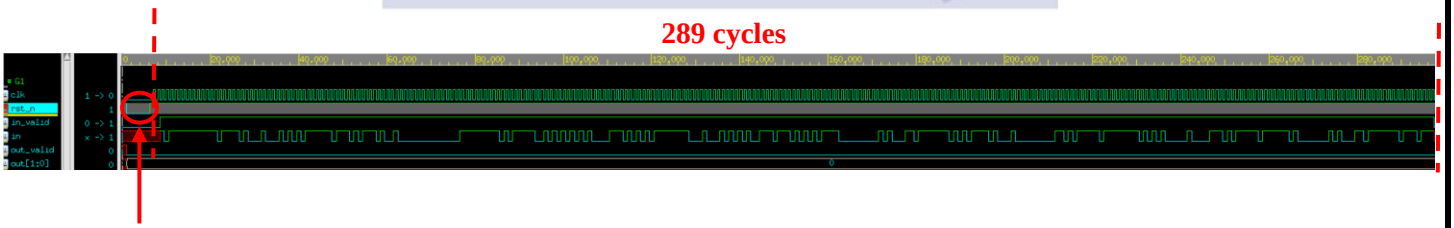



Fig1. Input waveform

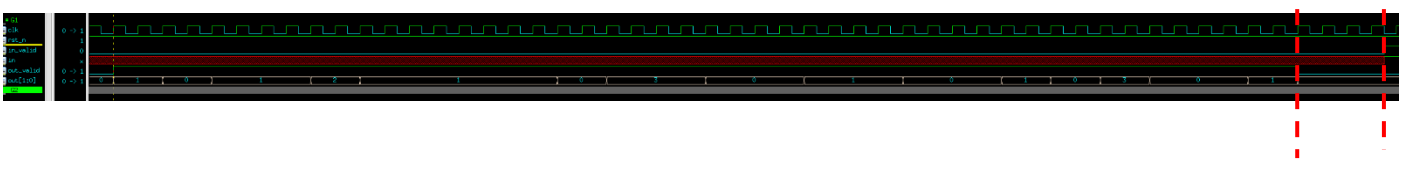


Fig 2. Output waveform

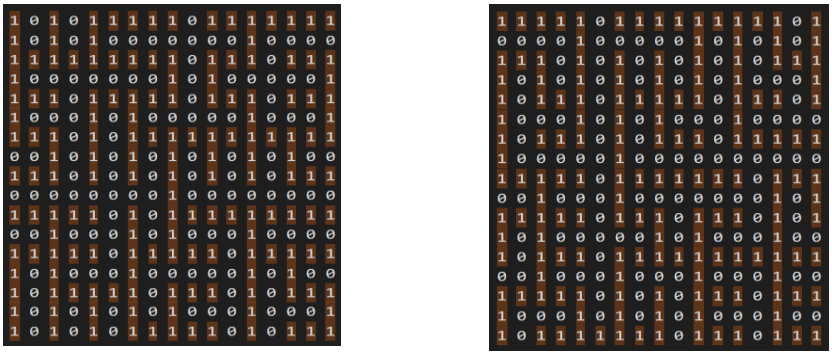


Fig 3. Some Maze examples

## Maze Constrain

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

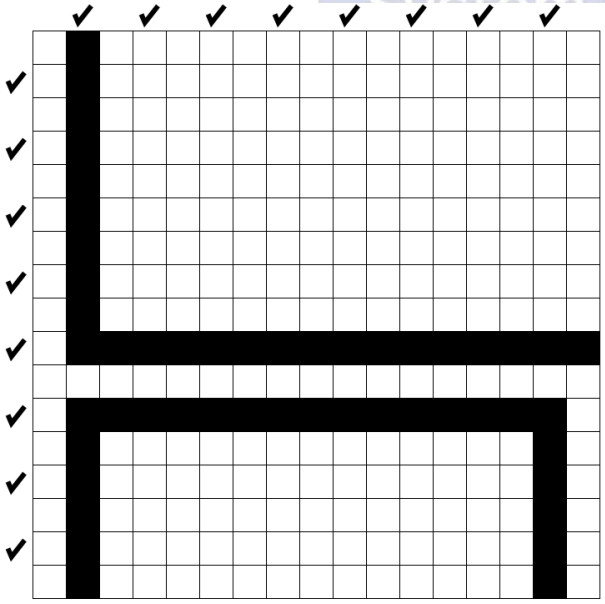


Fig 4. With constrain

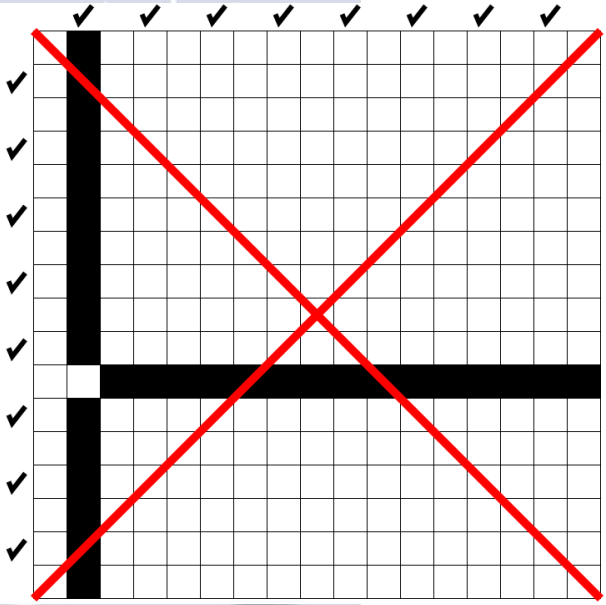


Fig 5. Without constrain

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

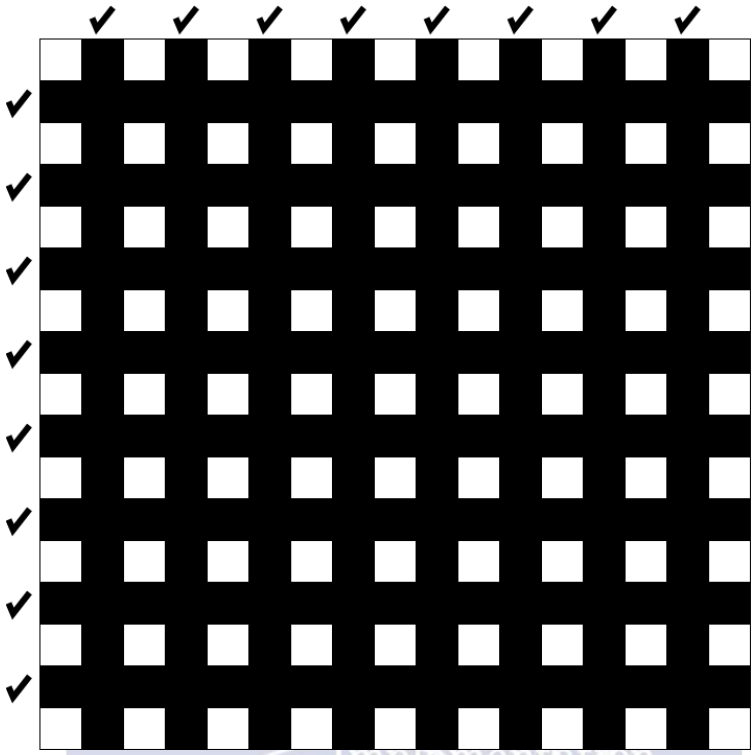


Fig 6. Where the walls can/cannot exist

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.