NYCU-EE IC LAB – Spring 2023

Lab03 Exercise

Design: Subway Surfers

Data Preparation

1. Extract files from TA's directory:
   % tar xvf ~iclabta01/Lab03.tar
2. The extracted LAB directory contains:
   a. **Exercise/**
   b. **Practice/**

Design Description

Subway Surfers is a side-scrolling running game, where the player controls a character running on a railway while trying to escape while being chased by the police. The player will see different types of objects on the tracks, such as road, lower obstacles, higher obstacles and trains. You need to run forward and dodge the obstacles and trains by using movements such as straight forward, swiping left, swiping right, and jumping in order to avoid them. The scene of *Subway Surfers* is shown in Fig.1.

Fig.1 *Subway Surfers*

In this Lab, you are going to design a circuit to play *Subway Surfers*. The straight line map is a **4 * 64 matrix**. The character can only move on this map. The PATTERN will send the initial position of the character **(init)** in the beginning of the game. The objects in the map have 4 types (2-bits): **road (2'b00), lower obstacles (2'b01), higher obstacles (2'b10), trains (2'b11)**. The **2-bit** inputs for **4 tracks** (**in0, in1, in2, in3**) is given **64 cycles** continuously. For simplicity, higher and lower obstacles and trains can only be randomly generated in specific regions.

Road

The road can be randomly generated at any position.

Obstacles

The **length unit of the lower and higher obstacles is 1**, and they can only **appear in even columns but excluding columns divisible by 8.**

Trains

The **length unit of the train is 4**, and they can only **appear in intervals where the remainder of the column divided by 8 falls within the range of 0 to 3 (column mod 8 = 0, 1, 2, 3)**. There must be **at least one train and no more than three trains in each interval (1~3 trains in each interval)**.

During the input stage, each cycle follows the above map generation rules (**Note: You must follow the rules to generate the map. If you don't follow the rules, you may end up with a map that has dead ends**), and the corresponding object information of in0, in1, in2, and in3 is continuously transmitted from the PATTERN to the DESIGN. A correct example of the generation of obstacles and trains is shown in Fig.2. The incorrect example of the generation of obstacles and trains is shown in Fig.3. Four types of objects and the height of the character are shown in Fig.4. The action of the character has 4 types: **forward (2'd0)**, **right (2'd1)**, **left (2'd2)**, and **jump (2'd3)**. When the character moves, consider only three individual objects in front of the current position to make decisions about using different movement and avoid obstacles and trains, shown in Fig.5.

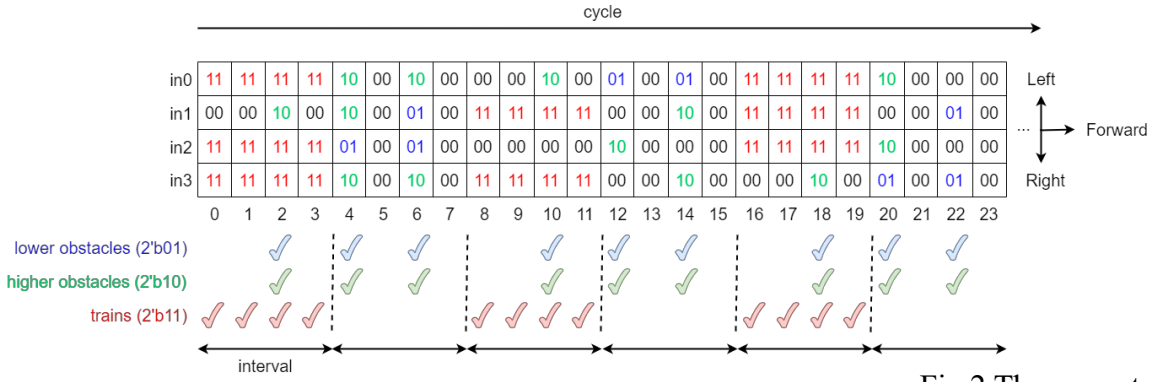


Fig.2 The correct example

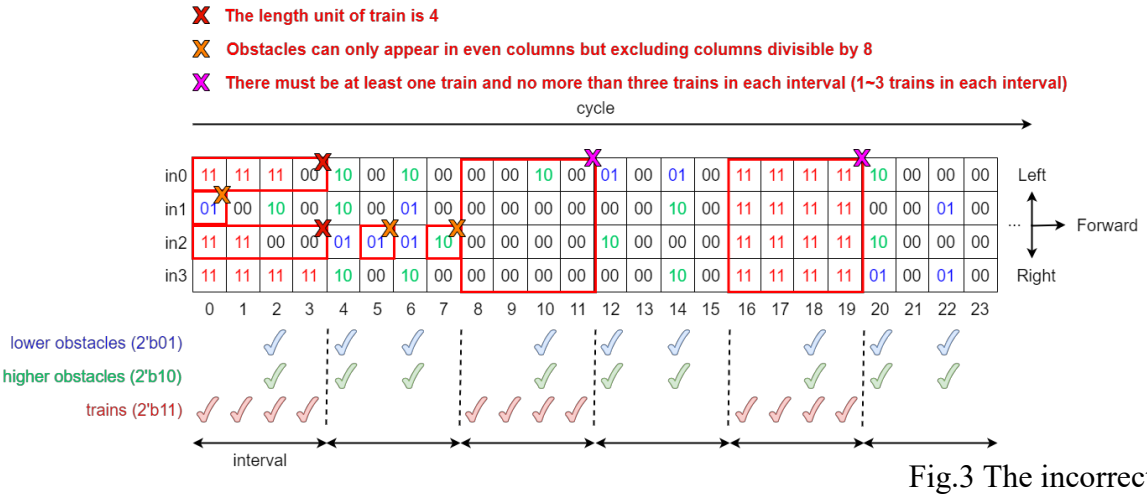


Fig.3 The incorrect example

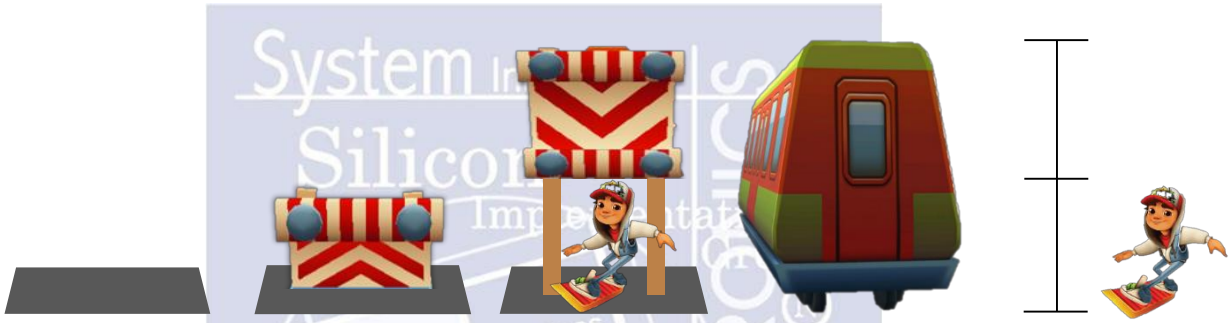


Fig.4 Four types of objects and the height of character

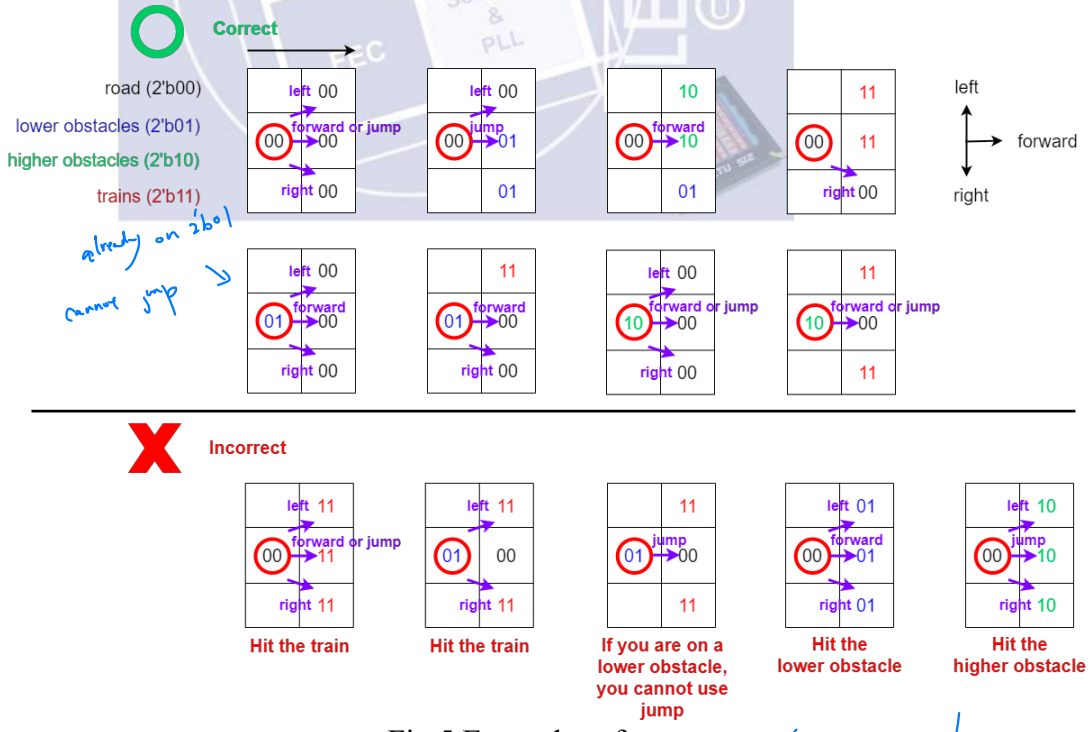
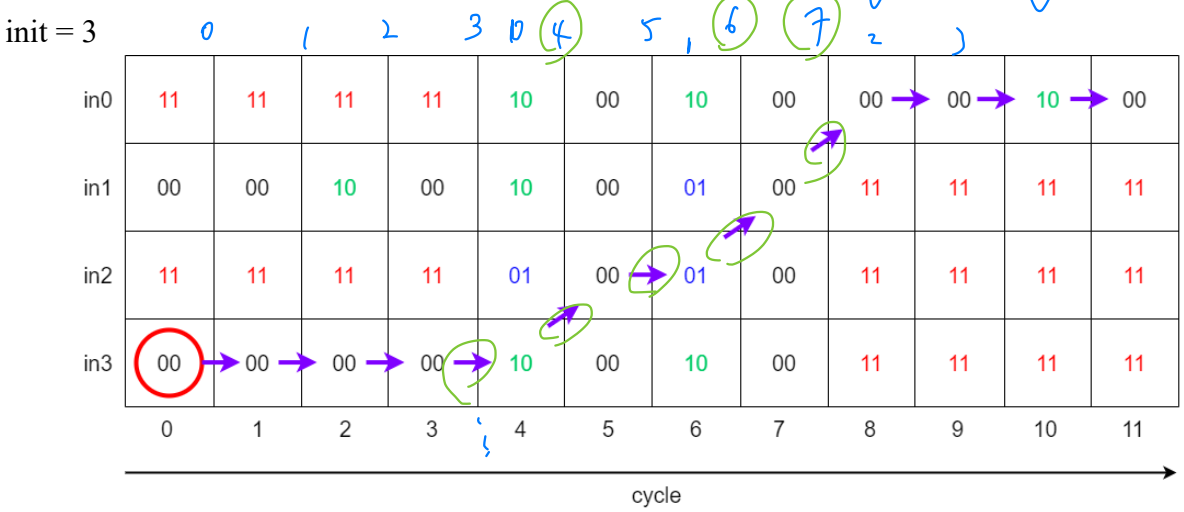


Fig.5 Examples of movement



cycle	0	1	2	3	4	5	6	7	8	9	10
out	2'd0 forward	2'd0 forward	2'd0 forward	2'd0 forward	2'd2 left	2'd3 jump	2'd2 left	2'd2 left	2'd0 forward	2'd0 forward	2'd0 forward

Fig.6 One of the solutions

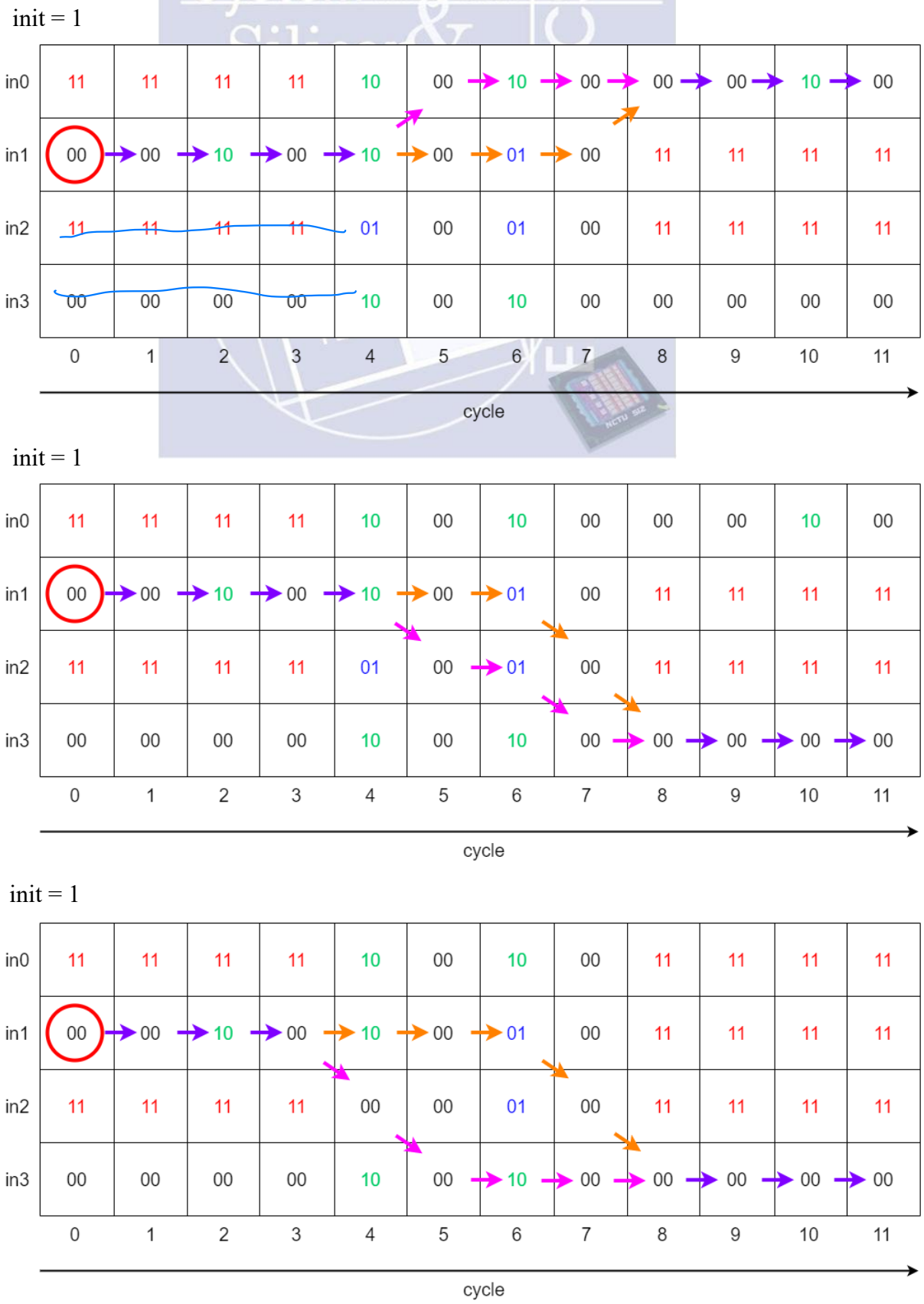


Fig. 7 Example of solution for different paths

■ Be aware that in DESIGN:

1. The output signal **out** has 4 types, **forward (2'd0)**, **right (2'd1)**, **left (2'd2)**, and **jump (2'd3)**.
2. The character must avoid all obstacles and trains and cannot run outside the map. If the character hits the obstacles or train or runs outside the map, you will fail the game.
3. You can jump (2'd3) to avoid lower obstacles (2'b01), but cannot use forward (2'd0).
4. You can forward (2'd0) to avoid higher obstacles (2'b10), but cannot use jump (2'd1).
5. You cannot jump (2'd3) over the trains.
6. If you are on a lower obstacle (2'd01), you cannot use jump (2'd3).
7. As long as all solutions that can avoid all obstacles and trains are the correct solution; your design has to choose a suitable path to avoid obstacles and trains.
8. **The total cost will evaluate the performance of the algorithm based on the movement you choose**, and you should try to use the movement with the lower total cost to traverse the entire map. **The definition of the cost and total cost will be explained in Grading Policy.**

■ Be aware that in PATTERN:

1. **Map generation rules (you must follow this rules)**
   i. **road (2'b00)**: The road can be randomly generated at any position
   ii. **obstacles (2'b01 or 2'b10)**: The length unit of the lower and higher obstacles is 1, and they can only appear in even columns but excluding columns divisible by 8.
   iii. **trains (2'b11)**: The length unit of the train is 4, and they can only appear in intervals where the remainder of the column divided by 8 falls within the range of 0 to 3 (column mod 8 = 0, 1, 2, 3). There must be at least one train and no more than three trains in each interval (1~3 trains in each interval).
2. The character's initial position (**init**) can only be on the road (2'b00) and cannot be on a train.
3. The pattern should check whether the character run outside the map.
4. The pattern should check whether the character avoids all obstacles and trains.
5. The pattern should check whether the movement is legal or not.
6. For the **out** signal, the pattern should have multiple solutions.

Inputs& Outputs

■ The following are the definition of input signals

Input Signals	Bit Width	Definition
clk	1	Clock
rst_n	1	Asynchronous active-low reset
in_valid	1	High when the input is valid
init	2	Initial position of the character
in0	2	The types of objects 2'b00: road 2'b01: lower obstacles 2'b10: higher obstacles 2'b11: trains
in1	2	
in2	2	
in3	2	

■ The following are the definition of output signals

Output Signals	Bit Width	Definition
out_valid	1	High when out is valid
out	2	The types of movements 2'd0: forward 2'd1: right 2'd2: left 2'd3: jump

1. You will receive **init [1:0]** in the first cycle when **in_valid** is high, and then **init [1:0]** is tied to an unknown state.
2. The input signals **in0 [1:0], in1 [1:0], in2 [1:0],** and **in3 [1:0]** are continuously deliver for **64 cycles**. When in_valid is low, all input signals should be tied to an unknown state.
3. All input signals are synchronized at the **negative edge** of the clock.
4. All outputs should be low after **rst_n** assert.
5. **out_valid** is set to high when **out [1:0]** is valid and will be high for **63 cycles** continuously when triggered.
6. **out_valid** should not be raised when **in_valid** is high.
7. The next round of the game will come in **2~4 negative edge of the clock** after your **out_valid** is pulled down. (The new map will be delivered)

Specifications

1. Top module name: SUBWAY (design file name: SUBWAY.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 when your out_valid is low**.
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 time of the clock cycles between the **falling edge of the in_valid and the rising edge of the out_valid**.
7. The **out_valid** and **out** must be asserted successively in **63 cycles**.
8. **The out should be correct when out_valid is high.**
9. The clock period is **10 ns**, because this exercise's main topic is the verification pattern, you don't need to modify the timing constraint.
10. The input delay is set to **0.5*(clock period)**.
11. The output delay is set to **0.5*(clock period)**, and the output loading is set to **0.05**.
12. The synthesis result of the data type **cannot** include any **latches**.
13. Gate-level simulation cannot include any timing violations without the *notimingcheck* command.
14. After synthesis, you can check SUBWAY.area and SUBWAY.timing. The area report is valid when the slack at the end of the timing report should be **non-negative (MET)**.
15. Performance is determined by **area** and **latency**. The lower the better.
16. **Any words with "error", "latch" or "congratulation" can't be used as variable name.**

Grading Policy

1. Function Validity: 40% (The grade of the 2nd demo would be **30% off**.)
2. Test Bench: 40% (The grade of the 2nd demo would be **100% off**.)
   - ■ SPEC 3: 3%
   - ■ SPEC 4: 3%
   - ■ SPEC 5: 3%
   - ■ SPEC 6: 3%
   - ■ SPEC 7: 3%
   - ■ SPEC 8: 25%
   - ✧ **SPEC 3~8** means the **third to the eighth** specification above.
   - ✧ **SPEC 8 is subdivided into 5 parts: (refer to Fig.5)**
     - **SPEC 8-1 (5%): The character cannot run outside the map.**
     - **SPEC 8-2 (5%): The character must avoid hitting lower obstacles.**
     - **SPEC 8-3 (5%): The character must avoid hitting higher obstacles.**
     - **SPEC 8-4 (5%): The character must avoid hitting trains.**
     - **SPEC 8-5 (5%): If you are on a lower obstacle (2'b01), you cannot use jump.**

- ✧ **You don't have a second chance for a test bench demo, it is served only one demo shot.**
- ✧ **The number of your patterns cannot be more than 300**, or you will **lose 5 points**.
- ✧ **If any spec is violated, you must show "SPEC X IS FAIL!" on your screen.**
  - **X is the number of the spec.**
  - **Please follow this rule "SPEC X IS FAIL!" when the specification is violated, or you will lose "all points" in the demo.**

```
SPEC 3 IS FAIL!
```

  - **TA will provide you with a txt file. Please copy and paste the text for display when the SPEC fails.**
  - **If multiple specifications are violated at the same time, you can only display the highest priority on the screen, or you will lose "all points" in the demo.**
  - **The priority of specification: SPEC 3 > SPEC 4 > SPEC 5 > SPEC 6 > SPEC 7 > SPEC 8-1 > SPEC 8-2 > SPEC 8-3 > SPEC 8-4 > SPEC 8-5**
  - **After showing "SPEC X IS FAIL!" on your screen, you have to finish your simulation immediately, or you will lose "all points" in the demo.**

```
$display("SPEC 3 IS FAIL!");
$finish;
```

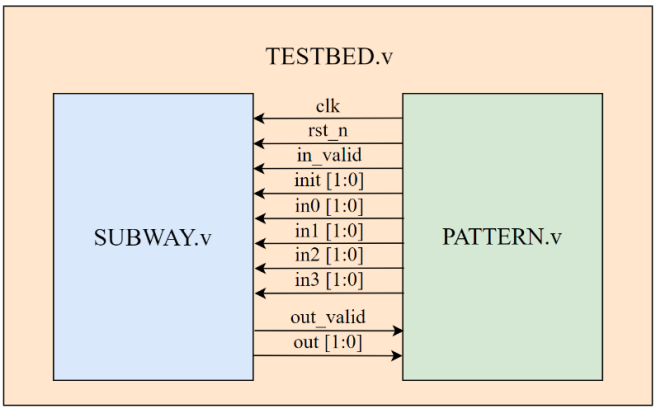
3. Performance: 20 % + 5% (Bonus)
   - ■ **Total latency * Area (20%)**
     - ✧ **You will get this part of points only if you pass TA's pattern.**
     - ✧ **The grade of the 2nd demo will be 30% off.**
     - ✧ **latency = 1 + execution latency**
   - ■ ***(Bonus)* Total cost (5%)**
     - ✧ Design **performance** is primarily evaluated based on **total latency** and **area**. The ***bonus*** section is intended to **encourage you to optimize your algorithm**.
     - ✧ When you have completed all the required items and have extra time available, you can then focus on optimizing your algorithm. **Don't feel pressured!**
     - ✧ The value of **total cost** represents **the quality of your algorithm** and is used to evaluate its movement strategy, such as trying to use forward instead of jump, and choosing the nearest road, etc. The lower the total cost, the better the algorithm.
     - ✧ **Definition of cost & total cost:**

movement	cost
forward (2'd0)	1
right (2'd1)	2
left (2'd2)	2
jump (2'd3)	4

Total cost = $\sum$ cost

     - ✧ TA will test your design with **300 patterns**. **If the total cost of your design is less than 28,000, you will get an extra 5 points.**

Block Diagram

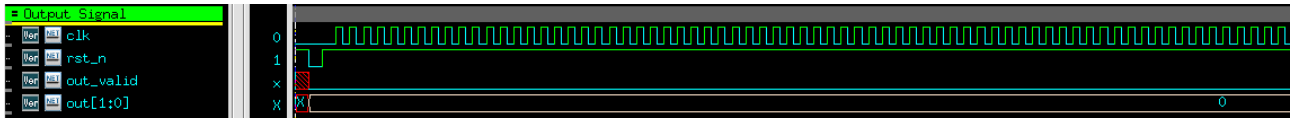


Note

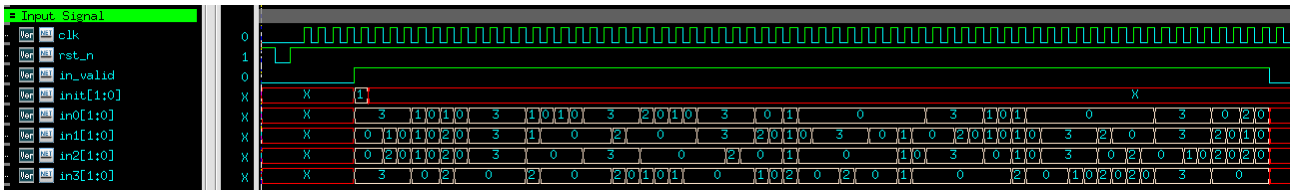
1. Please submit your design (SUBWAY.v) and pattern (PATTERN.v) in **Lab03/EXERCISE/09_SUBMIT**
   a. 1st_demo deadline: 2023/03/13(Mon.) 12:00:00
   b. 2nd_demo deadline: 2023/03/15(Wed.) 12:00:00
2. **Please upload the following file under 09_SUBMIT:**
   - SUBWAY.v and PATTERN.v
   - If your file **violates the naming rule**, you will **lose 5 points**.
3. **Template folders and reference commands:**
   01_RTL/ (RTL simulation) **./01run**
   02_SYN/ (Synthesis) **./01_run_dc**
   (Check if there is any **latch** or **error** in your design in **syn.log**)
   (Check the timing of the design in /Report/**SUBWAY.timing**)
   03_GATE/ (Gate-level simulation) **./01_run**
   09_SUBMIT (submit files) **./01_submit**
   09_SUBMIT (check files) **./02_check**
   You can key in ./09_clean_up to clear all log files and dump files in each folder.
4. In this lab, you need to write a pattern file. You can **only use a random system task** to generate patterns. You **cannot use any IO file to generate patterns**. If you **submit the txt files** you used in your pattern file, this lab will **fail**.

Example Waveform

1. Asynchronous reset and active-low and reset all output



2. 64 cycles for input signals of each round



3. 63 cycles for output signals of each round