NCTU-EE IC LAB - FALL 2022

Lab03 Exercise

Design: Block Party mode in Fall Guys

Data Preparation

1. Extract the 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

Block Party mode in Fall Guys is a particular challenge, where all you have to do is stay on a platform and avoid obstacles coming at you. You will see blocks coming at you from the front, and as more time passes, you will need to find a way to survive; in theory, it is easier said than done, and you must survive until the clock runs out. The screenshot of playing Block Party mode in Fall Guys is shown in Fig.1.

Fig.1 The Block Party mode in Fall Guys

In this Lab, you are going to design a circuit to play Block Party mode in Fall Guys. The platform is a **1 * 8 matrix**; the guy can only move on this platform. We will send the starting position of the guy in the beginning. The guy can appear on this platform anywhere. The limit clock in this game is **64 cycles**. It means that the obstacles will come at you for 64 cycles continuously. The obstacles are 2-bit and have four types, **no obstacles (2'b00)**, **obstacles in low places (2'b01)**, **obstacles in high places (2'b10)**, and **full obstacles (2'b11)**. For simplicity, the next cycle after the type of obstacles in low places(2'b01), obstacles in high places (2'b10), and full obstacles (2'b11) should be no obstacles (2'b00). Each cycle refers to a **combination of only no obstacles (2'b00)** or a **combination of**

obstacles in low places (2'b01) and full obstacles (2'b11) or a **combination of obstacles in high places (2'b10) and full obstacles (2'b11)**. Also, you only have **one exit** to survive in any cycle that the obstacles come at. The correct example of obstacles is shown in Fig.2. The incorrect example of obstacles is shown in Fig.3. Four types of obstacles and the height of the guy are shown in Fig.4. The action of the guy has 4 types, **stop (2'd0), right (2'd1), left (2'd2),** and **jump (2'd3)**. You need to use these actions to avoid obstacles.

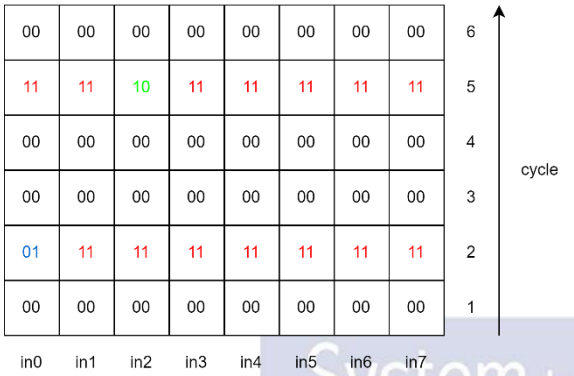


Fig.2 The correct example

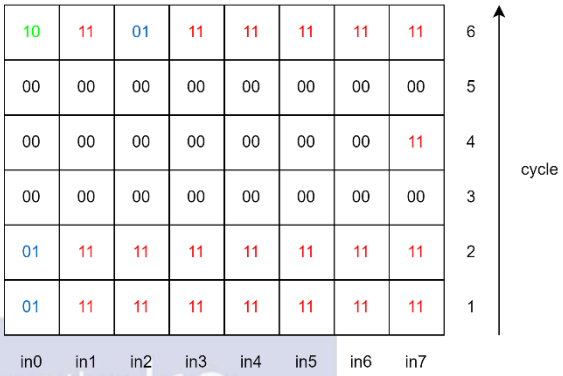


Fig.3 The incorrect example

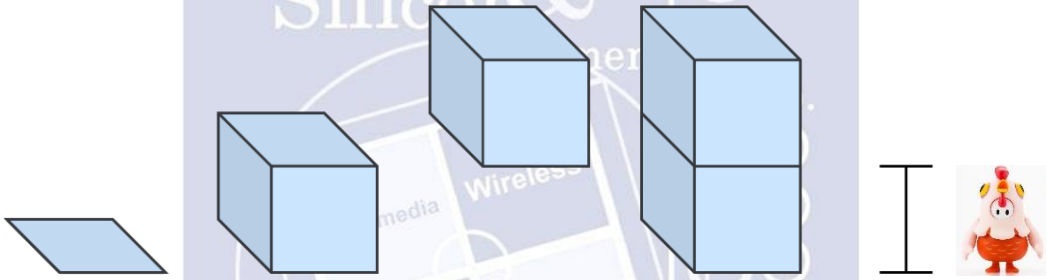


Fig.4 Four types of obstacles and the height of the guy.

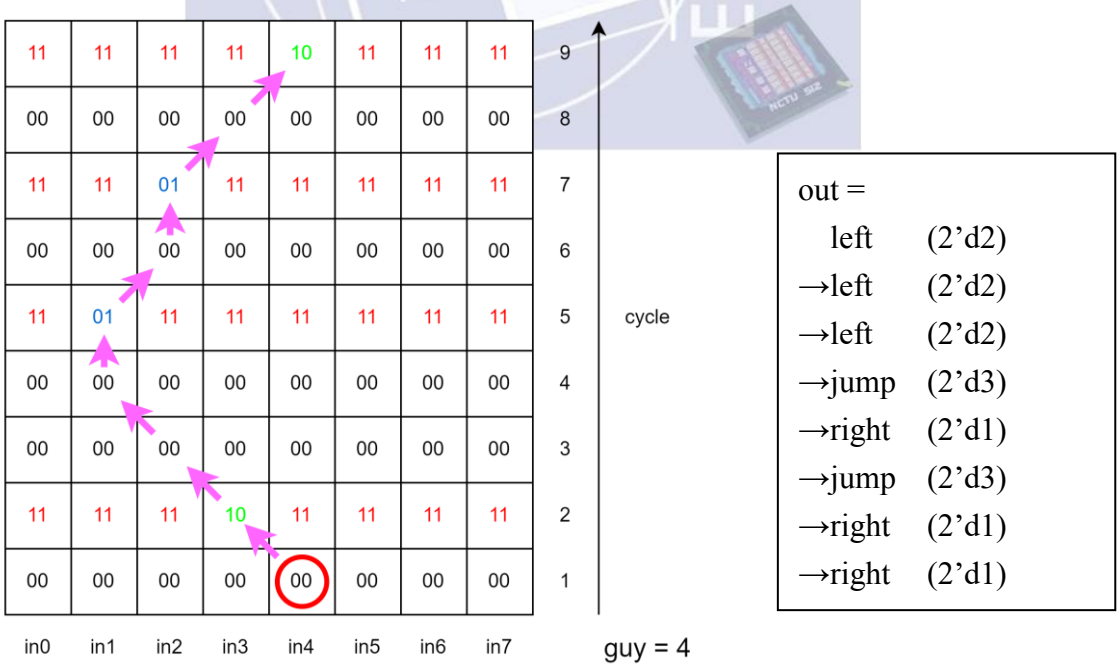
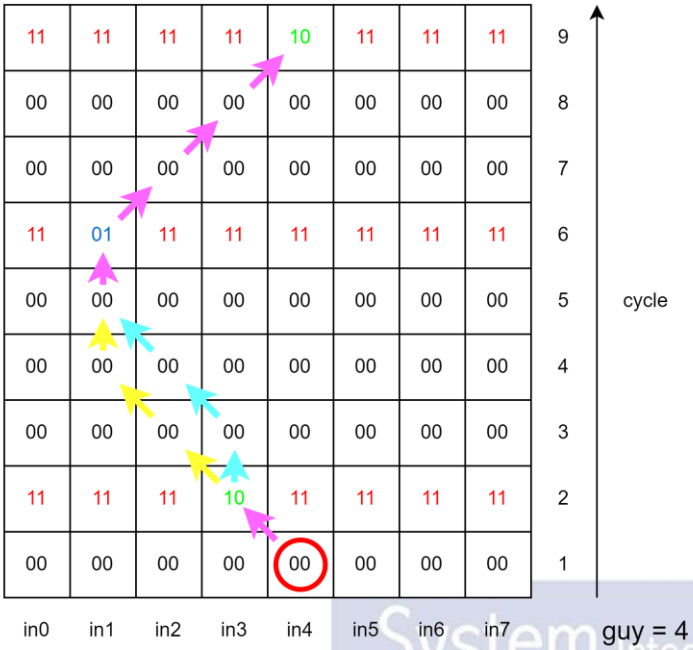


Fig.5 The one solution example.



out =		out =	
left	(2'd2)	left	(2'd2)
→left	(2'd2)	→stop	(2'd0)
→left	(2'd2)	→left	(2'd2)
→stop	(2'd0)	→left	(2'd2)
→jump	(2'd3)	→jump	(2'd3)
→right	(2'd1)	→right	(2'd1)
→right	(2'd1)	→right	(2'd1)
→right	(2'd1)	→right	(2'd1)

Fig.6 Multiple solution example.

- ✧ Be aware that in DESIGN:
  1. The output signal **out** have 4 types, **stop (2'd0)**, **right (2'd1)**, **left (2'd2)**, and **jump (2'd3)**.
  2. The guy must avoid obstacles that come at him and cannot leave the platform. If the guy hits the obstacles or leave the platform, you will fail this game.
  3. You have to jump (2'd3) to avoid obstacles in low places (2'b01).
  4. If the guy **jumps from high to low place**, **out** must be **2'b00** for **2 cycles** (Fig.7).
  5. If the guy **jumps to the same height**, **out** must be **2'b00** for **1 cycle** (Fig.7 & Fig.8).
  6. If the guy jumps from low to high place, out can be anything you want in the next cycle.
  7. There is more than one or only one solution; your design has to choose a suitable path to avoid obstacles (Fig.8 & Fig.9).
- ✧ Be aware that in PATTERN:
  1. Combination of each cycle:
     - i. **8 no obstacles (2'b00)**.
     - ii. **1 obstacles in low places (2'b01) + 7 full obstacles (2'b11)**.
     - iii. **1 obstacles in high places (2'b10) + 7 full obstacles (2'b11)**.
  2. The next cycle after combination ii (1 obstacles in low places + 7 full obstacles) or combination iii (1 obstacles in high places + 7 full obstacles) should be combination i (8 no obstacles).
  3. The cycle between obstacles and obstacles should be long enough to move yourself to avoid obstacles (Fig.12).
  4. The first cycle must be 8 no obstacles (2'b00) (Fig.13).
  5. The pattern should check whether the guy avoids all obstacles.
  6. Pattern should have multiple solutions.

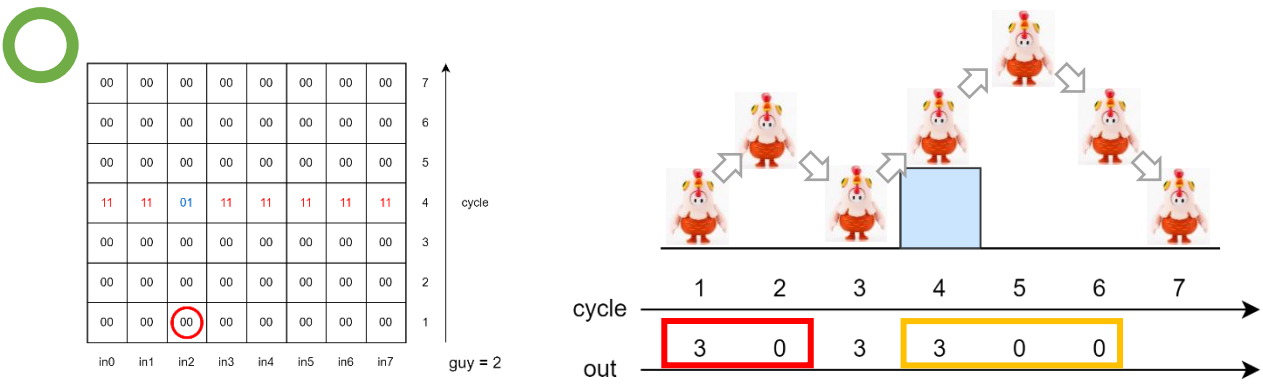


Fig.7 The correct action of the guy. The **red** frame is an example of the guy **jumping to the same height**. In the **orange** frame is the guy **jumping from high to low.**

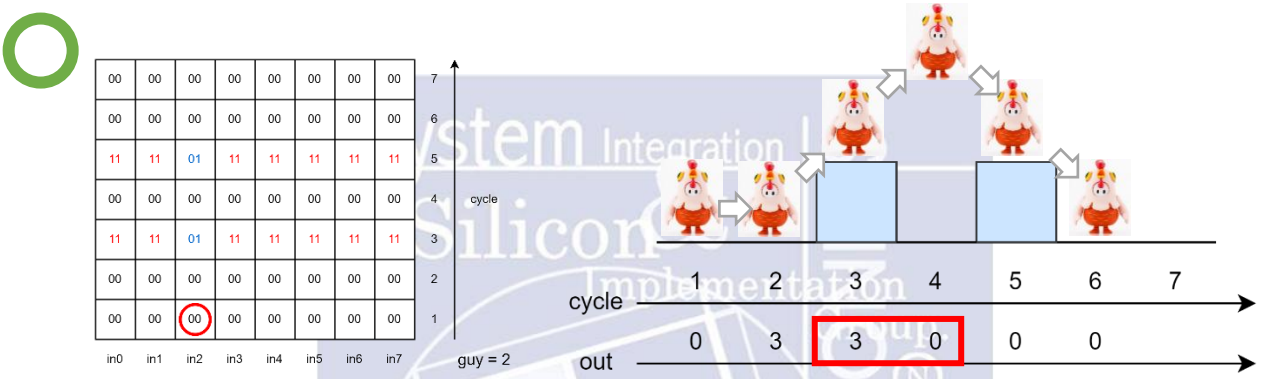


Fig.8 The correct action of the guy. The **red** frame is an example of the guy **jumping to the same height**.

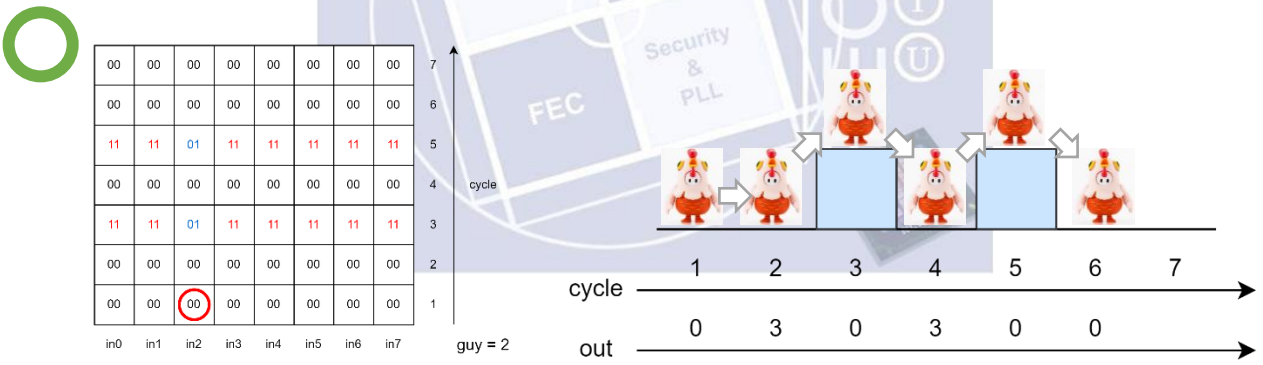


Fig.9 Another correct action of the guy.

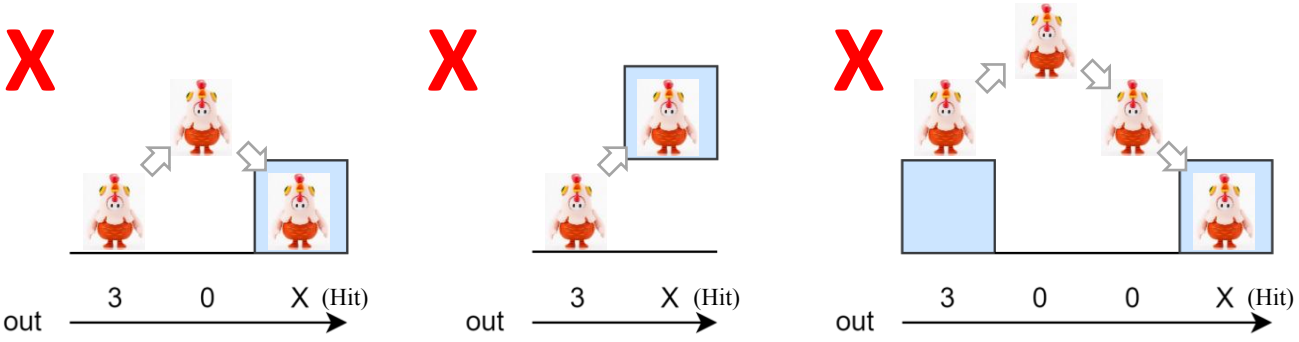


Fig.10 Some incorrect action of the guy.

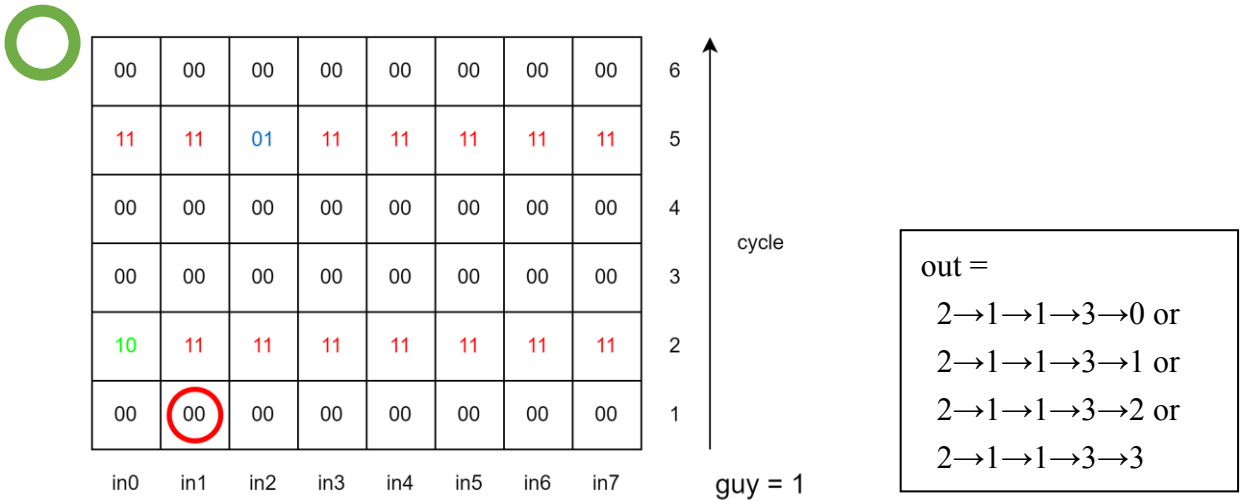


Fig.11 The suitable pattern.

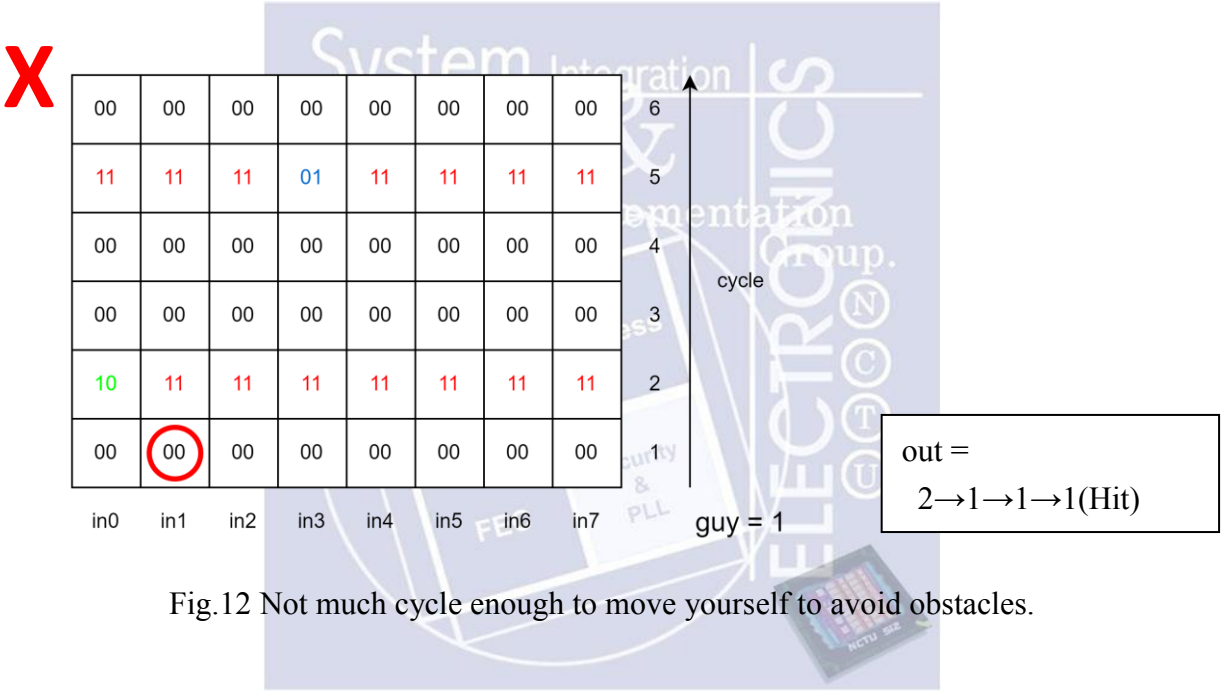


Fig.12 Not much cycle enough to move yourself to avoid obstacles.

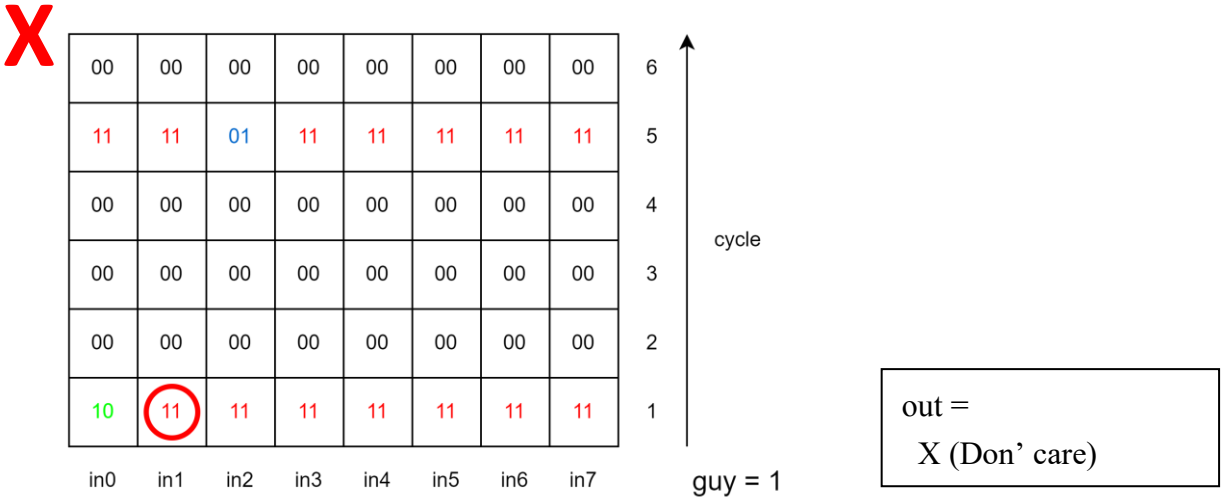


Fig.13 The first cycle must be 8 no obstacles (2'b00).

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 the input is valid.
guy	3	Start position of the guy.
in0	2	The type of obstacles. 2'b00: no obstacles. 2'b01: obstacles in low places. 2'b10: obstacles in high places. 2'b11: full obstacles.
in1	2	
in2	2	
in3	2	
in4	2	
in5	2	
in6	2	
in7	2	

■ The following are the definitions of output signals.

Output Signals	Bit Width	Definition
out_valid	1	High when out is valid.
out	2	Stop: 2'd0; Right: 2'd1; Left: 2'd2; Jump: 2'd3

1. You will receive **guy [2:0]** in the first cycle when **in_valid** is high, and then guy [2:0] is tied to an unknown state.
2. The input signals **in0 [1:0], in1 [1:0], in2 [1:0], in3 [1:0], in4 [1:0], in5 [1:0], in6 [1:0]** and **in7 [1:0]** are continuously delivered 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 **3~5 negative edge of the clock** after your **out_valid** is pulled down.
8. All operations are **unsigned**.

Specifications

1. Top module name: BP (design file name: BP.v)
2. It is an asynchronous reset and active-low architecture. If you use synchronous reset (considering reset after clock starting) in your design, you may not be able to reset the signals.
3. The **reset signal (rst_n)** would be given **only once at the beginning** of the 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 to 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 BP.area and BP.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: 18% (each 6%)
    - Correct Functionality: 7%
    - ✧ **SPEC 3~8** means the **third to the eighth** specification above.
    - ✧ **SPEC 8 is subdivided into 3 items.**
        - ■ **SPEC 8-1: The correct output means that the guy has to avoid all obstacles and cannot leave the platform (6%).**
        - ■ **SPEC 8-2: If the guy jumps from high to low place, out must be 2'b00 for 2 cycles (6%).**
        - ■ **SPEC 8-3: If the guy jumps to the same height, out must be 2'b00 for 1 cycle (6%).**
    - ✧ **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.**

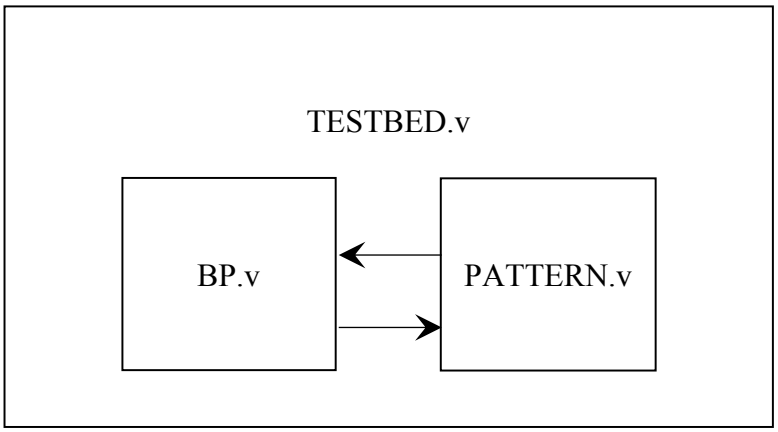
```
**************************************************************
*                        SPEC 3 IS FAIL!                     *
*  Output signal should be 0 after initial RESET  at     2000   *
**************************************************************
```

  - ■ **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 specifications: SPEC 3 > SPEC 4 > SPEC 5 > SPEC 6 > SPEC 7 > SPEC 8-3 > SPEC 8-2 > SPEC 8-1**
  - ■ **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 ("------------------------------------------------------");
$display ("                                                      ");
$display ("                  SPEC 5 IS FAIL!                     ");
$display ("  out_valid should not be high when in_valid is high   ");
$display ("                                                      ");
$display ("------------------------------------------------------");
$finish;
```

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

Block diagram



Note

1. Submit your design (BP.v) and pattern (PATTERN.v) in Lab03/EXERCISE/09_SUBMIT
   a. 1st_demo deadline: **2022/10/10(Mon.) 12:00:00**
   b. 2nd_demo deadline: **2022/10/12(Wed.) 12:00:00**
2. **Please upload the following files under 09_ SUBMIT:**
   - BP.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)	**./01_run**
02_SYN/	(Synthesis)	**./01_run_dc**

   (Check if there is any **latch** in your design in **syn.log)**
   (Check the timing of the design in /Report/ BP**.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**.

Sample Waveform

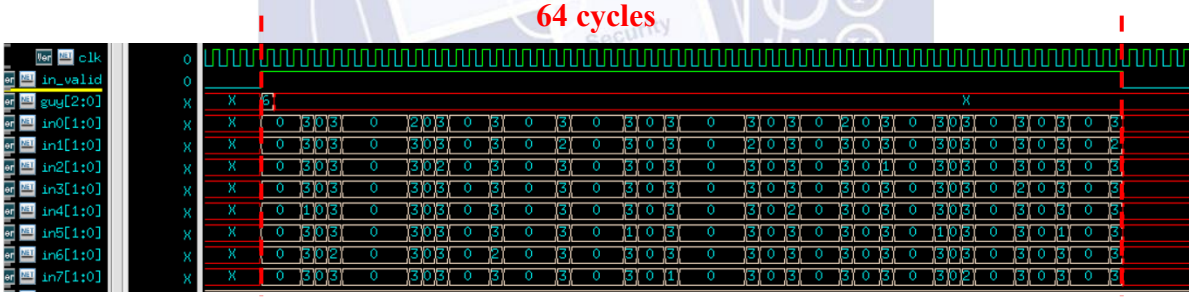


Fig.14 Input waveform

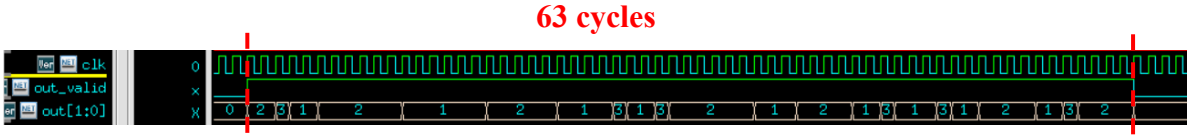


Fig.15 Output waveform

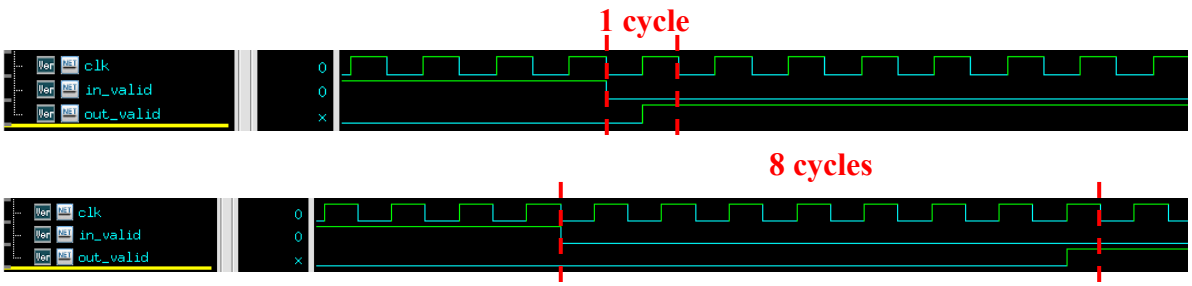


Fig.16 Execution latency example. The above image is 1 cycle. The image below is 8 cycles.