NCTU-EE IC LAB - FALL 2019

Lab03 Exercise

Design: Candy Crush

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

Candy Crush is a "match three" game, where the core gameplay is based on swapping two adjacent candies among several candies on the gameboard as to make a row or column of at least 3 matching-colored candies. On this match, the matched candies are removed from the board, and candies above then fall into the empty spaces, with new candies appearing from the top of the board. The screenshot of playing Candy Crush is shown in Fig. 1.

Fig 1. The Candy Crush

In this lab, you are going to design a circuit to play the Candy Crush game. **For simplicity, after you clear some candies at a step, the candies won't fall into the empty spaces, but just stick to the original position**. We will send **36 candies** to each position of **6x6 board** at beginning, and there are **six kinds of colored candies** (red, blue, yellow, green, orange, and purple), and each of them could either be a **normal candy or a stripe candy** as Fig 2 shows. Then, we will send someone's swapping record step by step, illustration of the swapping process shows in Fig 3. After you clear some candies at a step, the candies won't fall into the empty spaces, but just stick to the original position at next step. Finally, you need to calculate the total score of the given gameboard

after finishing all steps. The matching number of candies and the corresponding score is shown in Table 1.

✧ **Be aware that in DESIGN:**

1. When clearing a stripe candy, in addition to the original matching candies, the whole row or column will also be cleared (there will be two types of stripe candy, horizontal type will clear the whole row, vertical type will clear the whole column) at the same time.
2. The function of stripe candy clearing a whole row or column can only be triggered if the stripe candy is one of the original matching candies. This means that if you clear a stripe candy and there is another stripe candy in that row or column but not the original matching candy, the second stripe candy will be cleared as a normal candy.
3. You still need to change the candies even the action will not make a match of candies (different from original game rules).

✧ **Be aware that in PATTERN:**

1. There will be **four stripe** candies given randomly on the game bored in the beginning.
2. Cannot be interchanged with blanks or non-6*6 array areas when the candy is swapped. In other words, the position will not be given to a location where the candy has been cleared. Also, the action will only change with the position that has candy.
3. Pattern should avoid the case when the whole bored is cleared and there aren't any legal actions.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

Fig 2. The 6x6 gameboard

Cleaning horizontal stripe candy

First switching First cleaning Second switching Second cleaning

Fig 3. The process of candy switching and clearing

Note: Purple horizontal stripe candy is one of the original matching candies, so it cleared the whole row. The blue vertical candy is not one of the original matching candies and got cleared by stripe candy, so it will be cleared as a normal candy, not a stripe candy.

Matching number of candies	Score
3	1
4	2
5	3
6	4
Number of candies cleared by stripe candy	Score
1	1
2	2
3	3
4	4
5	5

Table 1. The matching number and corresponding score

Fig 4. Some examples of score counting

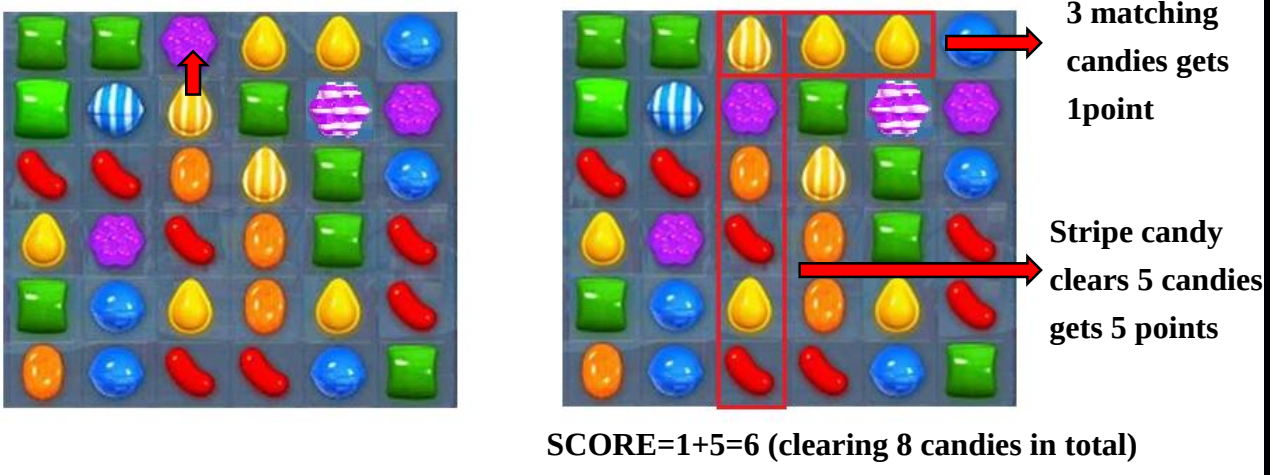

Fig 5. example of score using stripe candy

NOTE: The candies cleared by stripe candies will be counted as the extension of the original

matching candies, and stripe candy itself isn't counted for stripe case. This means that if your stripe candy clear 1 candy, you will get 2 points in total (including the 1 point from original 3 matching candies). If your stripe candy clear 5 candies, you will get 6 points in total (as illustrated by Fig5).

Matching number of candies	Number of candies cleared by stripe candy	Total score
3 (gets 1 point)	1 (gets 1 point)	1+1=2
3 (gets 1 point)	5 (gets 5 point)	1+5=6
4 (gets 2 point)	1 (gets 1 point)	2+1=3
4 (gets 2 point)	5 (gets 5 point)	2+5=7

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	1	High when in_color and in_stripe is valid.
in_valid_2	1	High when in_action and in_starting_pos are valid.
in_color	3	6 kinds of candy colors: 3'b000:Red, 3'001: Blue 3'b010:Green, 3'b011:Yellow 3'b100:Orange, 3'b101:Purple
in_stripe	1	2 kinds of candy types: 1'b0: horizontal stripe, 1'b1: vertical stripe
in_action	2	5 kinds of actions to align candies, 3'b000:Up, 3'b001:Down 3'b010:Left, 3'b011:Right.
in_starting_pos	6	The first 3 bits, [5:3] are for x, others are for y.

The following are the definitions of output signals

Output Signals	Bit Width	Definition
out_valid	1	High when out_score is valid.
out_score	7	The total score after all switch have been done.

1. The input of **in_color** and **in_stripe** is delivered in **raster scan order** for **36 cycles** continuously. When **in_valid_1** is low, input is tied to unknown state.
2. The **first four** cycles when **in_valid_1** is high, **in_starting_pos** will specify the coordinate of the four given stripe candies, and **in_stripe** will specify the type of the stripe candy. Besides the first four cycles of **in_valid_1** being high, inputs are tied to unknown state.
3. The inputs of **in_action** and **in_starting_pos** are delivered for **10 cycles** continuously. When

in_valid_2 is low, inputs are tied to unknown state.

4. All input signals are synchronized at negative edge of the clock.
5. The action of moving candies should not out of the gameboard.
6. The output signal **out_score** must be delivered for **1 cycles**, and **out_valid** should be high simultaneously.
7. The **in_valid_2** will come in **2 cycles** after **in_valid_1** is pulled down
8. The next round of the game will come in **4 negative edge of clock** after your **out_valid** is pulled down. (The new order of candies will be delivered)
9. All operations are unsigned and the **out_score** signal has no the overflow case.

Specifications

1. Top module name: CC (design file name: CC.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_valid** is limited to high only **one** cycle when you want to output the result.
5. The execution latency is limited in **500 cycles**. The latency is the clock cycles between the falling edge of the last **in_valid_2** and the rising edge of the first **out_valid**.
6. The **out_score** should be correct when **out_valid** is high.
7. The **out_score** should be reset after your **out_valid** is pulled down.
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 CC.area and CC.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**. The lower, the better.

Grading Policy

1. Function Validity: 30%
2. Test Bench: 50%

✧ **If any of the above spec is violated, your pattern should terminate the simulation.**

✧ **SPEC 3 means the third specification above.**

✧ **You don't have second chance for test bench demo, it's served only one demo shot.**

✧ **If one of SPEC3~7 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 or you will lose 2 points in demo.**

```
*************************************************************
*                      SPEC 3 IS FAIL                       *
*   Output signal should be 0 after initial RESET at  300   *
*************************************************************
```

3. Performance:20 %
   - Simulation time*Area: 20%

Note

1. **Please upload the following files on new-e3 platform before 12:00 p.m. on Oct. 7:**
   - CC_iclab??.v and PATTERN_iclab??.v If the upload file violating the naming rule, you will get 5 deduct points on this lab.
   - 2nd demo deadline is 12:00 p.m. on Oct. 9
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/CC**.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/your_file.txt", which means that we will put your files into 00_TESTBED for demo. If the demo is fail due to file path, we will punish on the score.

Sample Waveform

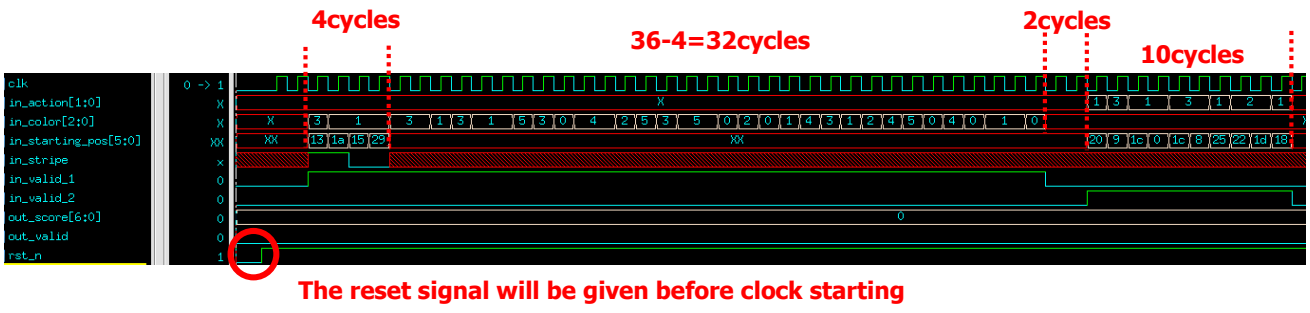

Fig 1. Input waveform