11
11
22
1 1

15
6
90

ac1f 956f
abd6 91e8

100 / 132 / 13 / 90 / 81 / 5 / 84 / 120

505
132
273
120
147
100
47

505
132
373
120
253
100
153
90
63

132
120
100
352
442

NCTU-EE IC LAB – Fall 2024

Lab01 Exercise

Design: Snack Shopping Calculator(SSC)

Data Preparation

1. Extract files from TA's directory:
   % tar -xvf ~iclabTA01/Lab01.tar

Design Description and Examples

One day, the professor asks you to buy some snacks for your lab. He/She gives you a credit card to let you buy the snacks. Also, the professor gives you a list of several choices of snacks, each with a different price and amount needed. However, you're not sure if this credit card is valid or if the money inside your card is enough for all the snacks. So, you decided to design a calculator to help you check if the card is valid and how many snacks you can buy.

✓ Description of this lab

➢ **Input:**

You will receive a number, card_num(64 bits), as the credit card number, representing 16 numbers(range from 0~9) of the credit card(4 bits/digit). The parameter "input_money" (9 bits) represents how much money is inside the credit card. Also, a number, snack_num (32 bits), will represent 8 different types of snacks' amount needed(4 bits/each), and the variable price (32 bits) represents the price of each snack(4 bits/each).

- ➢ [63:0] card_num => 16 digit, 4 bits each.
- ➢ [8:0] input_money
- ➢ [31:0] snack_num => 8 different snacks, 4 bits each.
- ➢ [31:0] price => 8 different price, 4 bits each.

➢ **Calculate Card Number**

First, you must check if the credit card number is valid. The credit card number checking rule is as follows.

➢ Step 1.

Multiply all the odd digits by 2, separate all the digits after multiplying them, and sum it up.

(which means if the digit is 8 and 16 after multiplying, make it 1 and 6 separately)

1776 7787 8998 1960

2 14 14 16 16 18 2 12

10 12 14

56789

1 3 5 7 9

18

0101 ⇓ 0001 0110 ⇓ 0011 0111 ⇓ 0101

➢ Step 2.

Add on the even digits. 184 1001

➢ Step 3. 10010 16 2

If the sum of all the digits can be divided by 10, then it's a valid number.

1000 ⇓ 10000 72

Ex: card_1 72 + 14×8 = 111 card_2

card_num	[63:60]	[59:56]	[55:52]	[51:48]	[47:44]	[43:40]	[39:36]	[35:32]	[31:28]	[27:24]	[23:20]	[19:16]	[15:12]	[11:8]	[7:4]	[3:0]
Digit	4	0	4	8	0	2	2	2	0	7	8	6	8	1	3	4
multiply	8	x	8	x	0	x	4	x	0	x	16	x	16	x	6	x

sum = 8 + 0 + 8 + 8 + 0 + 2 + 4 + 2 + 0 + 7 + (1 + 6) + 6 + (1 + 6) + 1 + 6 + 4 = 70

70 % 10 = 0 => Valid number

16 9 144

➢ **Buy Snack** 先從總價高的開始買

After checking the credit card number, you can now start to buy the snacks.

Buy them from the highest total price until you can't buy them anymore. Be sure that you can only buy the required amount of snacks. If the money is not enough to buy the exact amount, then you don't need to buy it.

Ex: You have 450 dollars (input_money = 450)

snack[7] ⇓ snack_1

	snack_num	price	total	Buy
[31:28]	15	15	225	Yes
[27:24]	3	6	18	No
[23:20]	7	8	56	No
[19:16]	13	10	130	Yes
[15:12]	6	1	6	No
[11:8]	12	7	84	Yes
[7:4]	10	4	40	No
[3:0]	4	4	16	No

snack_8

24 225 8 1800

Total cost = 225 + 130 + 84 = 365

8 7 6 5 4 3 2 1

1800

➢ **Calculate & Output**

There are two output signals, "out_valid" and "out_change".

➢ out_valid: If the credit card number is valid, out_vlaid should be 1, else is 0.

無效　　相同作為輸入貨幣

➢ out_change: If the credit card number is invalid, out_change should be the same as input_money. Otherwise, output the change after you have bought the snacks.

Example 1:
Credit card number(card_num): 1234 1234 1234 1234
Input money(input_money): 400
Snacks needed(snack_num): 1, 2, 3, 4, 5, 6, 7, 8
Snacks' price(price): 8, 7, 6, 5, 4, 3, 2, 1

out_valid: 0 (credit card number invalid)
out_change: 400

459
242
257
165
92
84
8

Example 2:
Credit card number(card_num[0]~[15]): 4556 4976 7312 5561
Input money(input_money): 100
Snacks needed(snack_num[0]~[7]): 1, 2, 3, 4, 5, 6, 7, 8
Snacks' price(price[0]~[7]): 8, 7, 6, 5, 4, 3, 2, 1

8, 14, 18, 22, 22, 18, 14, 8

out_valid: 1
out_change: 10 (100 – 20 – 20 – 18 – 18 -14 = 10)

a 10
b 11
c 12
d 13
e 14

14

The summary of the description and specifications are as followings:

Input Signal	Bit Width	Description
card_num	4 / each (with 16 digits) Total 63 bits	ranged from **1~9** (a digit) **unsigned integer**
input_money	9	ranged from **1~511** **unsigned integer**
snack_num	4 / each (with 8 different type) Total 32 bits	ranged from **1~15** (each type) **unsigned integer**
price	4 / each (with 8 different price) Total 32 bits	ranged from **1~15** (each price) **unsigned integer**

15
15
215

15
15
75
15
225

224
225
8
1800

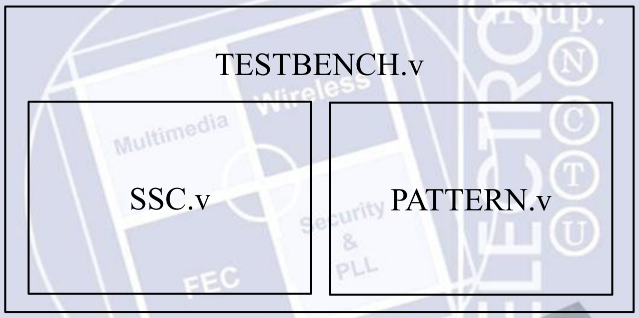
225 224 223

Output Signal	Bit Width	Description
out_valid	1	"1" when the credit card number is valid, else "0".
out_change	9	The change after buying the snacks from the highest total price to the lowest one. Keep the same as "input_money" if the credit card number is invalid.

Specifications

1. Top module name : SSC(File name: SSC.v)
2. After synthesis, check the "SSC.area" and "SSC.timing" in the folder "Report". **The area report is valid only when the slack in the end of "SSC.timing" is "MET".**
3. The synthesis result **cannot** contain any **latch**.
   Note: You can check if there is a latch by searching the keyword "**Latch**" in 02_SYN/**syn.log**

Block Diagram



Grading Policy

The performance is determined by the area of your design. The less area your design has, the higher grade you get. Try to reach better performance by thinking your architecture before coding.

Function Validity: 70%

Performance: area 30%

If you fail Lab01 at the first demo and pass at the second demo, you will get 30% off of your original score. Get no score if you fail both the first and second demos. Note that you will get a 0 score if you are found plagiarism in your code.

Note

1. Tar all your design by run the command **Lab01/09_SUBMIT/00_tar**
2. Submit your design through **Lab01/09_SUBMIT/01_submit**
    a. 1st_demo deadline: 2023/09/18(Wed.) 12:00:00
    b. 2nd_demo deadline: 2023/09/20(Fri.) 12:00:00
3. If your file violates the naming rule, you will lose 5 points.
4. Don't use any wire/reg/submodule/parameter name called *error*, *Congratulations*, *latch* or *FAIL* otherwise you will fail the lab. Note: * means any char in front of or behind the word. e.g: error_note is forbidden.

Be careful about all details!
Template folders and reference commands:
In demo, the reference commands is:

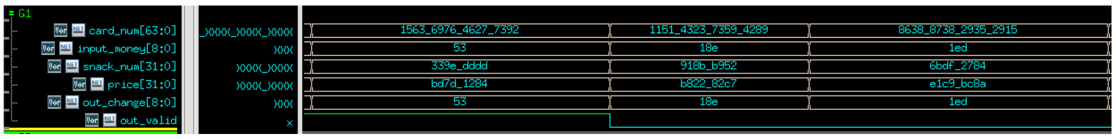
1. 01_RTL (RTL simulation):
    ./01_run_vcs_rtl
2. 02_SYN/ (Synthesis):
    ./01_run_dc_shell
    (Check **latch** by searching the keyword "**Latch**" in 02_SYN/**syn.log**)
    (Check the design's timing in /Report/ SSC.timing)
    (Check the design's area in /Report/ SSC.area)
3. 03_GATE/ (Gate-level simulation):
    ./01_run_vcs_gate
4. 09_SUBMIT/ (submit your files):
    ./00_tar
    ./01_submit
    ./02_check

You can key in **./09_clean_up** to clear all log files and dump files in each folder

Example Waveform

Input and output signal:

* all the input and output are shown in hexadecimal *


G1
card_num[63:0] _XXXX_XXXX_XXXX 1563_6976_4627_7392 1151_4323_7359_4289 8638_8738_2935_2915
input_money[8:0] XXX 53 18e 1ed
snack_num[31:0] XXXX_XXXX 339e_dddd 918b_b952 6bdf_2784
price[31:0] XXXX_XXXX bd7d_1284 b822_82c7 e1c9_bc8a
out_change[8:0] XXX 53 18e 1ed
out_valid x

Hint

Hint1: Try to use **behavior modeling description** instead of gate level description.

Hint2: Try to use **submodule** rather than copy and paste to simply your design. (not necessary in this lab)

```
// -------------------------------------------------
// Example for using submodule
// BBQ bbq0(.meat(meat_0), .vagetable(vagetable_0), .water(water_0),.cost(cost[0]));
// -------------------------------------------------
```

Hint3: Try to think if there is any possible **hardware** that can be **shared** with different mode operation. You can use command dc_shell-gui to examine your design.(not necessary in this lab)

Hint4: Pattern provided by TA will cover only some simple cases, you can try to write your own input / output file by yourself. Here is the format how TA will read in PATTERN:

```
/* input.txt format
1. [PATTERN_NUM]

repeat(PATTERN_NUM)
    1. [card_num_0] [card_num_1] ... [card_num_15}
    2. [input_money]
    3. [snack_num_0] [snack_num_1] ... [snack_num_7]
    4. [price_0] [price_1] ... [price_7]
*/

/* output.txt format
1. [out_valid]
2. [out_change]
*/
```

You can check input.txt and PATTERN.v in 00_TESTBED as a reference, and choose to write either c++/python or Verilog code for generating corner cases.