Lab09 Exercise: Design and Verification Using SystemVerilog

Design: Online Shopping Platform Simulation

Data Preparation

1. Extract the test data from TA's directory:

   % tar xvf ~iclabta01/Lab09.tar

2. The extracted LAB directory contains:

   Exercise/

   Practice/

Design Description

In this lab, we are going to build our own online shopping platform named *ICLabee* for students at the National Yang Ming Chiao Tung University (NYCU). On this platform, each user can be a buyer and can also be a seller. Users can perform several operations such as buying items, making deposits, checking their wallet, checking the stocks of other users and returning products to receive refunds.

Operations

- **Buy**

  Input: (User ID), {item ID}, {# of item}, {Seller ID}

  Output: {User info}

- **Check**

  Input: (User ID), (Seller ID, … if needed)

  Output: {16'd0, user's deposit} or {14'd0, seller's stocks}

- **Deposit**

  Input: (User ID), amount of money

  Output: {16'd0, user's deposit}

- **Return**

  Input: (User ID), {item ID}, {# of item}, {Seller ID}

  Output: {14'd0, user's stocks}

ICLabee

Table 1: Format of the shop information (32 bits) (**Exp**: Experience point)

MSB   LSB

6 bits	6 bits	6 bits	2 bits	12bits
User's stocks			User level	Exp
Large items	Medium items	Small items		

Table 2: Format of the user information (32 bits)

MSB   LSB

16 bits	2bits	6bits	8bits
Money	Shopping History		
	Item ID	# of items	Seller ID

Table 3: The price to buy items and the reward Exp after buying the items.

Items		Price	Exp
Large	2'b01	'd300	+'d60 /per items
Medium	2'b10	'd200	+'d40 /per items
Small	2'b11	'd100	+'d20 /per items

Table 4: The required experience to upgrade and the delivery fee corresponding to user level. **Exp**: The required experience to upgrade.

User levels		Exp	Delivery fee
Platinum	2'b00		'd10
Gold	2'b01	'd4000	'd30
Silver	2'b10	'd2500	'd50
Copper	2'b11	'd1000	'd70

Table 5: The corresponding value of actions and error messages. (The number in parentheses indicates the priority. The smaller, the higher. If the operation results in several errors at the same time, output the one with the highest priority).

ACTION		ERROR MESSAGE	
Buy	4'b0001	Seller's inventory is not enough (2)	4'b0010
		Out of money (3)	4'b0011
		User's inventory is full (1)	4'b0100
Check	4'b0010		
Deposit	4'b0100	Wallet is full	4'b1000

Return	4'b1000	Wrong seller ID (2)	4'b1001
		Wrong number (3)	4'b1100
		Wrong item (4)	4'b1010
		Wrong operation (1)	4'b1111
		No error	4'b0000

The following are some rules about the actions:

1. The action will not be complete if the action outputs the error message.

---**Buy**--

2. Users need to spend money to buy items and user's stocks will increase after buying items.

3. Each purchase must include a shipping fee of corresponding value depending on user level, please refer to Table 4.

4. Seller will earn money without containing shipping fee and the inventory will decreases after selling items.

5. Each user can only have at most 63 items of each type of item. If the user wants to '**Buy**', but the user's inventory will be full after buying items, please output the error message 'User's inventory is full'.

6. If the user wants to '**Buy**', but the seller's inventory will be not enough after buying items, please output the error message 'Seller's inventory is not enough'.

7. When the money earned plus the money already owned exceeds 16'd65535 dollars, the seller's deposit will be locked in 16'd65535, no matter how much money is earned.

8. Purchasing items can accumulate Exp, and the amount of Exp gained will be influenced by the quantity of items purchased. Exp can only be obtained when the purchase is completed.

9. When the user level is Platinum, the Exp will be locked in 12'd0, no matter buying any item.

10. When user's Exp reaches the required Exp listed in the Table 4, it will upgrade immediately. If upgrades, the extra Exp will be cleared.

11. After the purchase is completed, the shopping history will be updated.

---**Check**--

12. Users can "**Check**" their own money or the inventory of a certain merchant.

13. If want to check the inventory of other stores, *id_valid* will be valid for 1-5 cycles after the *act_valid* fall.

---**Deposit**--

14. Users can deposit money into their wallet through "**Deposit**".

15. If the amount exceeds the limit (16'b1111_1111_1111_1111) after completing

operation at this time, please output the error message 'Wallet is full'. The operation will fail and the amount in the wallet remains unchanged.

---**Return**--

16. Users can return the last purchased product through "**Return**".
17. The seller will issue a refund and the item will be added back to the inventory. Please note that the refund does not include the shipping fee.
18. "**Return**" must be made immediately after purchase, without any other operations including selling items to others by both the buyer and the seller. If violates this rule, please output the error message "Wrong operation".
19. The seller can only refund to the most recent buyer. If violates this rule, please also output the error message "Wrong operation".
20. If the given seller ID for the return is different from the seller ID in purchase history, please output the error message "Wrong seller ID".
21. The quantity of the returned item needs to match the purchase history, otherwise an error message "Wrong number" needs to be outputted.
22. The item ID for the return needs to match the purchase history, otherwise an error message "Wrong item" needs to be outputted.

More details will be described in 'Lab09_Exercise_note.pdf'.

Design Block Diagram

Here is the rule of this design:
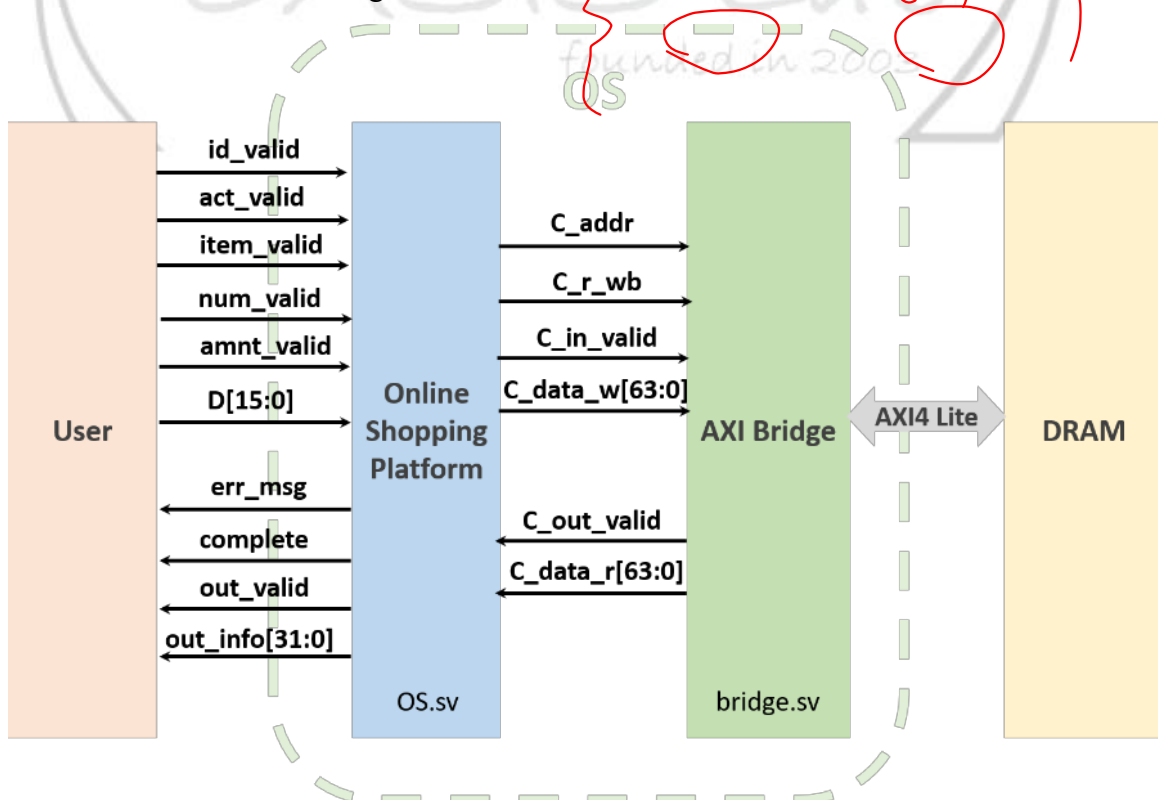
1. After **rst_n**, all output signals (both OS and bridge) should be set to 0.
2. Please initialize DRAM at the beginning. (Ref: Lab09_exercise_note.pdf)
3. DRAM_R_latency, DRAM_W_latency, DRAM_B_latency in pseudo_dram.sv is now set to 1, but you can modify it. **TA will change the value (1<= value <=100) while demo.**
4. The pattern will be inserted using **5 valid signals + 1 data signals**:

id_valid	High when input means user ID. (Buyer or Seller)
act_valid	High when input means action.
amnt_valid	High when input means the amount of money for deposit.
item_valid	High when input means item information.
num_valid	High when input means number of items.
D[15:0]	D = {8'd0, User ID} = {12'd0, action} = {money} = {14'd0, item ID} = {10'd0, # of item}

5. You need to raise *out_valid* only when your design has done current input operation.
6. We have a total of 256 users. Each user information will initially be randomized. Note that the value will not violate the rules.
7. All users have total 3 kinds of item. Therefore, the item ID is from 1 to 3. Note that the value will not violate the rules.
8. The shopping history will initially be also randomized. Shopping history at the initial stage of DRAM will be considered invalid, meaning that "**Return**" operation will output an error message before the user makes their first purchase.
9. We can get user info and shop info from DRAM via AXI Lite protocol.
10. C_in_valid can only be high for one cycle and cannot be pulled high again before C_out_valid.
11. If action is "**Buy**" and it succeeds, out_info should be {User information}.
12. If action is "**Check**" and it is successful, out_info should be {16'd0, user's deposit} or {14'd0, seller's stocks}.
13. If action is "**Deposit**" and it is successful, out_info should be {16'd0, user's deposit}.
14. If action is "**Return**" and it is successful, out_info should be {14'd0, user's stocks}.
15. The 5 valid input signals will not overlap with each other.
16. Out_valid cannot overlap with the 5 input valid signals.
17. Out_valid should be high for exactly one cycle.
18. Out_valid can only be high after giving all necessary input valid signals.
19. TA will check the output signal when the out_valid is high.

20. If actions complete, complete should be high and err_msg should be 4'b0000.
21. If action is not complete, complete should be the low, err_msg should be corresponding value, and out_info should be 32'd0.
22. The next operation will be valid for **2-10** cycles after the out_valid fall.
23. The system will not check the data stored in DRAM.

For the definition of cycles between signals, please refer to Lab09_exercise_note.pdf.

Input: OS.sv (OS: Online Shopping)

name	width	from	note
clk	1-bit	testbench	System clock
rst_n	1-bit	pattern	Asynchronous reset active low reset. Every output signal should be zero after **rst_n**.
id_valid	1-bit	pattern	High when enters user ID.
act_valid	1-bit	pattern	High when enters an action.
item_valid	1-bit	pattern	High when enters an item.
num_valid	1-bit	pattern	High when enters the number of items.
amnt_valid	1-bit	pattern	High when enters the amount of money.
D	16-bit	pattern	Represents the contents of the current input.
C_out_valid	1-bit	bridge	High when the data from the DRAM is ready.
C_data_r	64-bit	bridge	The returned data from DRAM.

Output: OS.sv

name	width	Send to	note
out_valid	1-bit	pattern	Should set to high when your output is ready. **out_valid** will be high for **only one** cycle.
err_msg	4-bit	pattern	err_msg will be 4'0000(No error) if operation is complete, else it needs to be corresponding value.
complete	1-bit	pattern	1'b1: operation complete 1'b0: some error occurred
out_info	32-bit	pattern	Show the corresponding information
C_addr	8-bit	bridge	Indicates which address we want to access.
C_data_w	64-bit	bridge	The data to overwrite DRAM.
C_in_valid	1-bit	bridge	High when the online shopping platform is ready to communicate with the bridge.
C_r_wb	1-bit	bridge	1'b1: Read DRAM. 1'b0: Write DRAM.

Buy: id → act¹ → item → num → id

Check: id → act³ (buyer)
       act² → id₄ (seller)

Deposit: id → act → amnt

Return: id → act → item → num → id

Input: Bridge.sv

name	width	from	note
clk	1-bit	testbench	System clock
rst_n	1-bit	pattern	Asynchronous reset active low reset. Every output signal should be zero after **rst_n**.
C_addr	8-bit	OS	Indicates which address we want to access.
C_data_w	64-bit	OS	The data to overwrite DRAM.
C_in_valid	1-bit	OS	High when the online shopping platform is ready to communicate with the bridge.
C_r_wb	1-bit	OS	1'b1: Read DRAM. 1'b0: Write DRAM.
AR_READY	1-bit	DRAM	AXI Lite signal
R_VALID	1-bit	DRAM	AXI Lite signal
R_DATA	64-bit	DRAM	AXI Lite signal
~~R_RESP~~	2-bit	DRAM	AXI Lite signal
AW_READY	1-bit	DRAM	AXI Lite signal
W_READY	1-bit	DRAM	AXI Lite signal
B_VALID	1-bit	DRAM	AXI Lite signal
B_RESP =0 → 正常	2-bit	DRAM	AXI Lite signal

Output: Bridge.sv

name	width	Send to	note
C_out_valid	1-bit	OS	High when data from DRAM is ready.
C_data_r	64-bit	OS	The returned data from DRAM
AR_VALID	1-bit	DRAM	AXI Lite signal
AR_ADDR	17-bit	DRAM	AXI Lite signal
R_READY	1-bit	DRAM	AXI Lite signal
AW_VALID	1-bit	DRAM	AXI Lite signal
AW_ADDR	17-bit	DRAM	AXI Lite signal
W_VALID	1-bit	DRAM	AXI Lite signal
W_DATA	64-bit	DRAM	AXI Lite signal
~~B_READY~~ =1	1-bit	DRAM	AXI Lite signal

Specifications

Top module
1. Top module name: **OSB** (file name: **bridge.sv** & **OS.sv**)

Reset
2. It is asynchronous reset and active-low architecture. If you use synchronous reset (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 (including OS.sv and bridge.sv) should be reset after the reset signal is asserted.

Design Constraints
4. The maximum clock period is **10 ns**.
5. Your latency should be **less than 10000 cycles** for each operation.
6. All outputs (including OS.sv and bridge.sv) are synchronized at clock rising edge.
7. The type defined in Usertype_OS.sv by TA **should not be modified, but you are encouraged to define a new datatype if needed**.

Synthesis
8. The input delay and the output delay are **'0.5*clock period'**.
9. The output load should be set to **0.05**
10. The synthesis result cannot include any **LATCH and ERROR.**
11. The total area (bridge_area + OS_area) should be **less than 800,000.**

Gate level simulation
12. The gate-level simulation cannot include any timing violations without the *notimingcheck* command.

Supplement
13. Don't use any wire/reg/submodule/parameter name called *error*, *congratulation*, *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.
14. Don't write Chinese or other language comments in the file you sent.
15. Verilog commands
    //synopsys dc_script_begin., //synopsys dc_script_end,
    //sysnopsys translate_off,   //synopsys translate_on
    are only allowed during the usage of including and setting designware IPs, **other design compiler optimizations are forbidden**.
16. Using the above commands are allowed, however **any error messages** during synthesize and simulation, regardless of the result will lead to failure in this lab.
17. Any form of display or printing information in Verilog design is forbidden. You may use this methodology during debugging, but the file you turn in **should not contain any coding that is not synthesized.**

Grading

- **Function: 70%**
- **Performance: (bridge_Area + OS_Area) x Total latency**
  - **Total latency = (# of execution cycles + 1) * cycle time**

Note

- Submit your design in Lab09/EXERCISE/09_SUBMIT
  - 1st_demo deadline: 2023/5/1(Mon.) 12:00:00
  - 2nd_demo deadline: 2022/5/3(Wed.) 12:00:00
- Please upload the following files under 09_SUBMIT:
  - **OS.sv, Usertype_OS.sv, bridge.sv**
  - In this lab, you can adjust your clock cycle time. Consequently, make sure to key in your clock cycle time after the command like the figure below. It's means that the TA will demo your design under this clock cycle time.

    `./01_submit 8.0`
  - If your files violate the naming rule, you will get 5 deduction points.

Reference Waveform

Please refer to Lab09_exercise_note.pdf

※ **Lab10 is about pattern with SystemVerilog, if you need to share your pattern with others, use the following command to encrypt your code. Since we have provided the way to protect your file, there will be no excuse for plagiarism.**

`% ncprotect -autoprotect PATTERN.sv`