# NCTU-EE IC LAB – Spring 2022

## Lab08 Exercise–Design and Verification Using SystemVerilog

## Design: Pokemon Simulation Game

### Data Preparation

1. Extract test data from TA's directory:

   **% tar xvf ~iclabta01/Lab08.tar**

2. The extracted LAB directory contains:

   Exercise/

   Practice/

### Design Description

In this lab, we are going to build a well-known game called Pokemon. In this game, player can perform several operations like buy, sell Pokemons, make a deposit, use items, check status or fight with other player.

**Operations**

- **Buy**

  Input: (Player ID), Pokemon type or Item type

  Output: {bag info, Pokemon info}

- **Sell**

  Input: (Player ID), Pokemon type or Item type

  Output: {bag info, Pokemon info}

- **Deposit**

  Input: (Player ID), amount of money

  Output: {bag info, Pokemon info}

- **Check**

  Input: (Player ID)

  Output: {bag info, Pokemon info}

- **Use_item**

  Input: (Player ID), item category

  Output: {bag info, Pokemon info}

- **Attack**

  Input: (Player ID), Other player's ID

  Output: {player Pokemon info, defender Pokemon info}

Table 1: The data format of bag information (32 bits)

MSB LSB

| 4 bits | 4 bits | 4 bits | 4 bits | 2 bits | 14 bits |
|---|---|---|---|---|---|
| Berry | Medicine | Candy | Bracer | Evolutionary stone | Money |

0~15

Table 2: The data format of Pokemon information (32 bits)

MSB LSB

31 28 27 24

| 4 bits | 4 bits | 8 bits | 8 bits | 8 bits |
|---|---|---|---|---|
| Stage | Type | Current HP | Attack damage (Atk) | Experience (Exp) |

Tabel 3: The price to buy or sell Pokemons.

| Pokemon type | | Buy Price | Middle Stage(4'b0010) | Highest Stage(4'b0100) |
|---|---|---|---|---|
| | | | Sale Price | Sale Price |
| Grass | 4'b0001 | 'd100 | 'd510 | 'd1100 |
| Fire | 4'b0010 | 'd90 | 'd450 | 'd1000 |
| Water | 4'b0100 | 'd110 | 'd500 | 'd1200 |
| Electric | 4'b1000 | 'd120 | 'd550 | 'd1300 |
| Normal (Eevee) | 4'b0101 | 'd130 | | |

Table 4: The information of different type and different stage of Pokemons

HP: The maximum HP, Atk: Attack damage, Exp: The required experience to evolve

| **Pokemon type** | | **Lowest(4'b0001)** | | | **Middle(4'b0010)** | | | **Highest(4'b0100)** | |
|---|---|---|---|---|---|---|---|---|---|
| | | **HP** | **Atk** | **Exp** | **HP** | **Atk** | **Exp** | **HP** | **Atk** |
| Grass | 4'b0001 | 'd128 | 'd63 | 'd32 | 'd192 | 'd94 | 'd63 | 'd254 | 'd123 |
| Fire | 4'b0010 | 'd119 | 'd64 | 'd30 | 'd177 | 'd96 | 'd59 | 'd225 | 'd127 |
| Water | 4'b0100 | 'd125 | 'd60 | 'd28 | 'd187 | 'd89 | 'd55 | 'd245 | 'd113 |
| Electric | 4'b1000 | 'd122 | 'd65 | 'd26 | 'd182 | 'd97 | 'd51 | 'd235 | 'd124 |
| Normal | 4'b0101 | 'd124 | 'd62 | 'd29 | | | | | |

形態

Table 5: The corresponding value of actions and error messages. (Number in parentheses means 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 | Already have a Pokemon (2) | 4'b0001 |
| | | Out of money (1) | 4'b0010 |
| | | Bag is full (3) | 4'b0100 |
| Sell | 4'b0010 | Do not have a Pokemon (1) | 4'b0110 |
| | | Pokemon is in the lowest stage (3) | 4'b1000 |
| | | Do not have item (2) | 4'b1010 |
| Use_item | 4'b0110 | Do not have a Pokemon (1) | 4'b0110 |
| | | Do not have item (2) | 4'b1010 |
| Attack | 4'b1010 | Do not have a Pokemon (1) | 4'b0110 |
| | | HP is zero (2) | 4'b1101 |
| Deposit | 4'b0100 | | |
| Check | 4'b1000 | | |
| | | No error | 4'b0000 |

Table 6: Pokemon type chart, the actual damage should multiply the number in the chart, round down to the integer

| Attacker \ Defender | Grass | Fire | Water | Electric | Normal |
|---|---|---|---|---|---|
| **Grass** | 0.5 | 0.5 | 2 | 1 | 1 |
| **Fire** | 2 | 0.5 | 0.5 | 1 | 1 |
| **Water** | 0.5 | 2 | 0.5 | 1 | 1 |
| **Electric** | 0.5 | 1 | 2 | 0.5 | 1 |
| **Normal** | 1 | 1 | 1 | 1 | 1 |

Table 7: The Exp reward after the "Attack"

| Character \ Opponent's stage | Lowest | Middle | Highest |
|---|---|---|---|
| Attacker | 'd16 | 'd24 | 'd32 |
| Defender | 'd8 | 'd12 | 'd16 |

28

Table 8: The code of evolutionary stone when storing in the bag

| None | Water Stone | Fire Stone | Thunder Stone |
|---|---|---|---|
| 'b00 | 'b01 | 'b10 | 'b11 |

Table 9: item that can be used to benefit Pokemon

| Item | Code when Buy/Sell /Use | Price (Buy/Sell) | Description | Note |
|---|---|---|---|---|
| Berry  | ’b0001 | ‘d16/’d12 | Current HP + ’d32 | HP can’t exceed max HP |
| Medicine  | ’b0010 | ‘d128/’d96 | Recover full HP | HP can’t exceed max HP |
| Candy  | ’b0100 | ‘d300/’d225 | Exp + ‘d15 | If evolution, the extra Exp will be cleared. |
| Bracer  | ’b1000 | ‘d64/’d48 | Atk + ‘d32 | The effect will disappear after successfully performing “Attack”, “Sell pokemon” action, “evolution” or change user. The effect can’t stack. |
| Water Stone  | ’b1001 | ’d800/’d600 | When your Pokemon is Eevee (Normal) and it has enough Exp to evolve, it will evolve and become Vaporeon (water), Flareon (fire) or Jolteon (electric) depending on the evolutionary stone | You can only have one evolutionary stone at the same time. If your pokemon is not Eevee or Eevee doesn’t reach the required Exp, using the stone will not have any effect and will still cost the stone. |
| Fire Stone  | ’b1010 | | | |
| Thunder Stone  | ’b1100 | | | |

00 ⇒ no stone

讓進來再給atk就好

eevee 要用 stone 才能進化

Following are some rules about the actions (12, 19, 20, 26 are the constraint for pattern, your pattern should avoid those situations):

1. Player need to spend money to buy Pokemon or item.
2. Player will earn money after selling Pokemon or item.
3. Player information includes "Bag information" and "Pokemon information".
4. If player already have a Pokemon and still want to "Buy". Then output the error message "Already have a Pokemon"
5. If player wants to "Buy" but running out of money. Then output the error message "Out of money"
6. The max number of each item (except evolutionary stone) and evolutionary stone are 15 and 1, respectively. If certain item has already reached the limit and player still want to "Buy" it. Then output the error message "Bag is full"
7. If player wants to perform "Sell pokemon", "Use_item", "Attack" without having a Pokemon, output the error message "Do not have a Pokemon"
8. If the Pokemon is in the lowest stage and the player wants to "Sell". Then output the error message "Pokemon is in the lowest stage"
9. The information of Pokemon should be reset after "Sell pokemon"
10. If player wants to "Use_item" or "Sell item" but his/her bag doesn't have that item. Then output the error message "Do not have item"
11. There are 2 roles during "Attack": "attacker" and "defender". Attacker means the one who performs "Attack" action, defender means the target of the attacker.
12. Attacker and defender can't be the same player.
13. If player wants to "Attack" but his/her Pokemon's HP is equal to 0, then output error message "HP is zero"

14. If player wants to “Attack” but defender doesn’t have a Pokemon, then output error message “Do not have a Pokemon”
15. If player wants to “Attack” but defender’s Pokemon’s HP is equal to 0, then output error message “HP is zero”
16. After “Attack”, defender’s HP = HP before attack – Attacker’s ATK * coefficient in Table 6. Notice that the HP can’t be negative.

就算 HP=0 还是可以進化

17. When “Attack”, the system will first calculate the HP, then calculate the EXP and then decide whether to evolve or not.

先計 HP 再算 exp

18. Player(x) will enter the “Player ID” “x” and do series of action until next player(y) enter “Player ID” “y”. However, in “Attack” the system requires the player to enter another player ID, in this case the player will not change.
19. Each player will perform at least one action before changing player.
20. When changing the player, the new player ID won’t be the same as the previous one.
21. The initial exp of the purchased Pokemon is set to 8’d0
22. When the Pokemon is in the highest stage, the Exp will be locked in 8’d0, no matter using the item or having a fight.
23. When Pokemon’s (not including Eevee) EXP reaches the required EXP listed in the Table 4, it will evolve.

只要 exp 到就会自动進化

24. When Eevee’s EXP reaches the required EXP listed in the table4 and the player uses the evolutionary stone, it will evolve and directly become the highest stage. If player doesn’t use the evolutionary stone, the EXP of Eevee will be locked in 8’d29, no matter using the item or having a fight.

eevee 只会從 lowest 到 highest

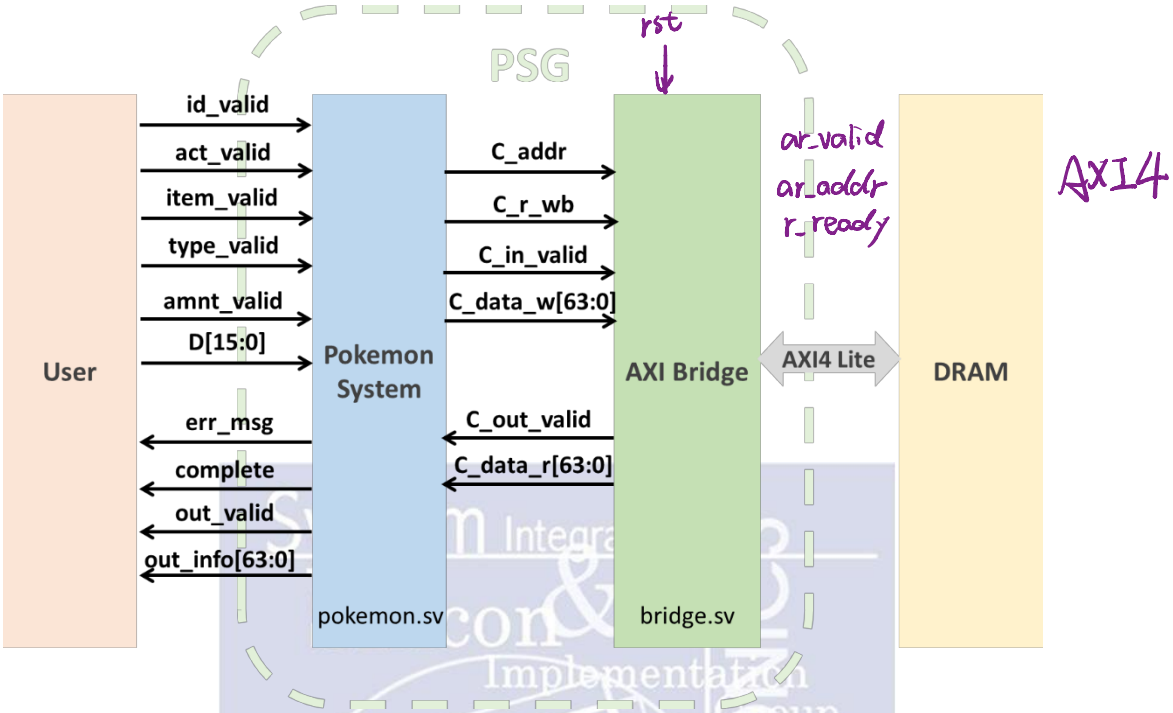
25. When the Pokemon evolves, it will recover full HP. The effect of items and the extra Exp will be cleared.
26. The money of each player won’t overflow.

↑最多就 14bit

More details will be described in “Lab08_Exercise_note.pdf”.

## Design block diagram

Here is the rule of this design:




1. After **rst_n**, all the output signals (both pokemon and bridge) should be set to 0.
2. Please initialize DRAM at beginning. (please refer to Lab08_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 signal**:

| | |
|---|---|
| id_valid | High when input means player ID |
| act_valid | High when input means action |
| item_valid | High when input means item |
| type_valid | High when input means Pokemon's type |
| amnt_valid | High when input means the amount of money for deposit |
| D[15:0] | D = {2'b0,money} = {8'b0, player_ID} = {12'b0, action}<br>= {12'b0, item} = {12'b0, type} |

5. You need to raise **out_valid** when your design has been done for the input supply operation.
6. We have total 256 players. The status of each player information will be randomized initially. Note that the value won't violate the rules.
7. We get player info (bag info + pokemon info) from DRAM via AXI Lite protocol.
8. C_in_valid can only be high for one cycle, and can't be pulled high again before C_out_valid.

9. If action is “Buy”, “Sell”, “Check”, “Use_item” or “Deposit” and it successes, out_info should be {bag_info, pokemon_info}
10. If action is “Attack” and it successes, out_info should be {player_pokemon_info, defender_pokemon_info}
11. If player doesn’t have a Pokemon, the Pokemon info should be 32’b0.
12. The 5 input valid signals won’t overlap with each other.
13. Out_valid cannot overlap with the 5 input valid signals
14. Out_valid can only be high for exactly one cycle.
15. Out_valid can only be high after given all necessary input valid signals.
16. Check the output signal only when the out_valid is high.
17. If action complete, complete should be high and err_msg should be 4’b0.
18. If action not complete, complete should be low, err_msg should be corresponding value and out_info should be 64’b0.
19. Next operation will be valid **2-10** cycles after out_valid fall.
20. System will not check the data stored in DRAM. → 只會 check output DRAM Don't care

For the definition of cycles between signals, please refer to Lab08_exercise_note.pdf

## Input: pokemon.sv

| name | width | from | note |
|---|---|---|---|
| clk | 1-bit | testbench | System clock |
| rst_n | 1-bit | pattern | Asynchronous reset active low reset.<br>Every output signal should be zero after **rst_n**. |
| id_valid | 1-bit | pattern | High when user enter player ID. |
| act_valid | 1-bit | pattern | High when user enter action. |
| item_valid | 1-bit | pattern | High when user enter item. |
| type_valid | 1-bit | pattern | High when user enter Pokemon’s type. |
| amnt_valid | 1-bit | pattern | High when user enter the amount of money. |
| D | 16-bit | pattern | Represents the contents of current input. |
| C_out_valid | 1-bit | bridge | High when data from DRAM is ready. |
| C_data_r | 64-bit | bridge | The returned data from DRAM. |

## Output: pokemon.sv

| name | width | Send to | note |
|---|---|---|---|
| out_valid | 1-bit | pattern | Should set to high when your output is ready.<br>**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 | 64-bit | pattern | Show the corresponding infomation |
| 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 pokemon system is ready to communicate with bridge. |
| C_r_wb | 1-bit | bridge | 1’b1: Read DRAM. 1’b0: Write DRAM. |

## Input: Bridge.sv

| name | width | from | note |
|---|---|---|---|
| clk | 1-bit | testbench | System clock |
| rst_n | 1-bit | pattern | Asynchronous reset active low reset.<br>Every output signal should be zero after **rst_n**. |
| C_addr | 8-bit | pokemon | Indicates which address we want to access. |
| C_data_w | 64-bit | pokemon | The data to overwrite DRAM. |
| C_in_valid | 1-bit | pokemon | High when pokemon system is ready to communicate with bridge. |
| C_r_wb | 1-bit | pokemon | 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 | 2-bit | DRAM | AXI Lite signal |

## Output: Bridge.sv

| name | width | Send to | note |
|---|---|---|---|
| C_out_valid | 1-bit | pokemon | High when data from DRAM is ready. |
| C_data_r | 64-bit | pokemon | 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-bit | DRAM | AXI Lite signal |

## Specifications

***Top module***

1. Top module name: **PSG** (File name: **bridge.sv** & **pokemon.sv**)

***Reset***

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 (including pokemon.sv and bridge.sv) should be reset after the reset signal is asserted.

***Design Constraints***

4. Maximum clock period is **15 ns**.
5. Your latency should be **less than 1200 cycles** for each operation.
6. All outputs (including pokemon.sv and bridge.sv) are synchronized at clock rising edge.
7. The type defined in Usertype_PKG.sv by TA **should not be modified, but you are encouraged to define new datatype if needed**.
8. C_in_valid can only be high for one cycle, and can't be pulled high again before C_out_valid
9. Out_valid can only be high for exactly one cycle.

***Synthesis***

10. The input delay and output delay are **"0.5*clock period"**.
11. The output load should be set to **0.05**
12. The synthesis result of data type cannot include any **LATCH.**
13. Total area (bridge_area + pokemon_area) should be **less than 150,000**

***Gate level simulation***

14. The gate level simulation cannot include any timing violations without the *notimingcheck* command.

***Supplement***

15. 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.
16. Don't write Chinese comments or other language comments in the file you turned in.

17. 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**.
18. 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.
19. 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 synthesizable**.

## Grading

- **Function: 70%**
- **Performance: ( bridge_Area + pokemon_Area ) x Latency**

## Note

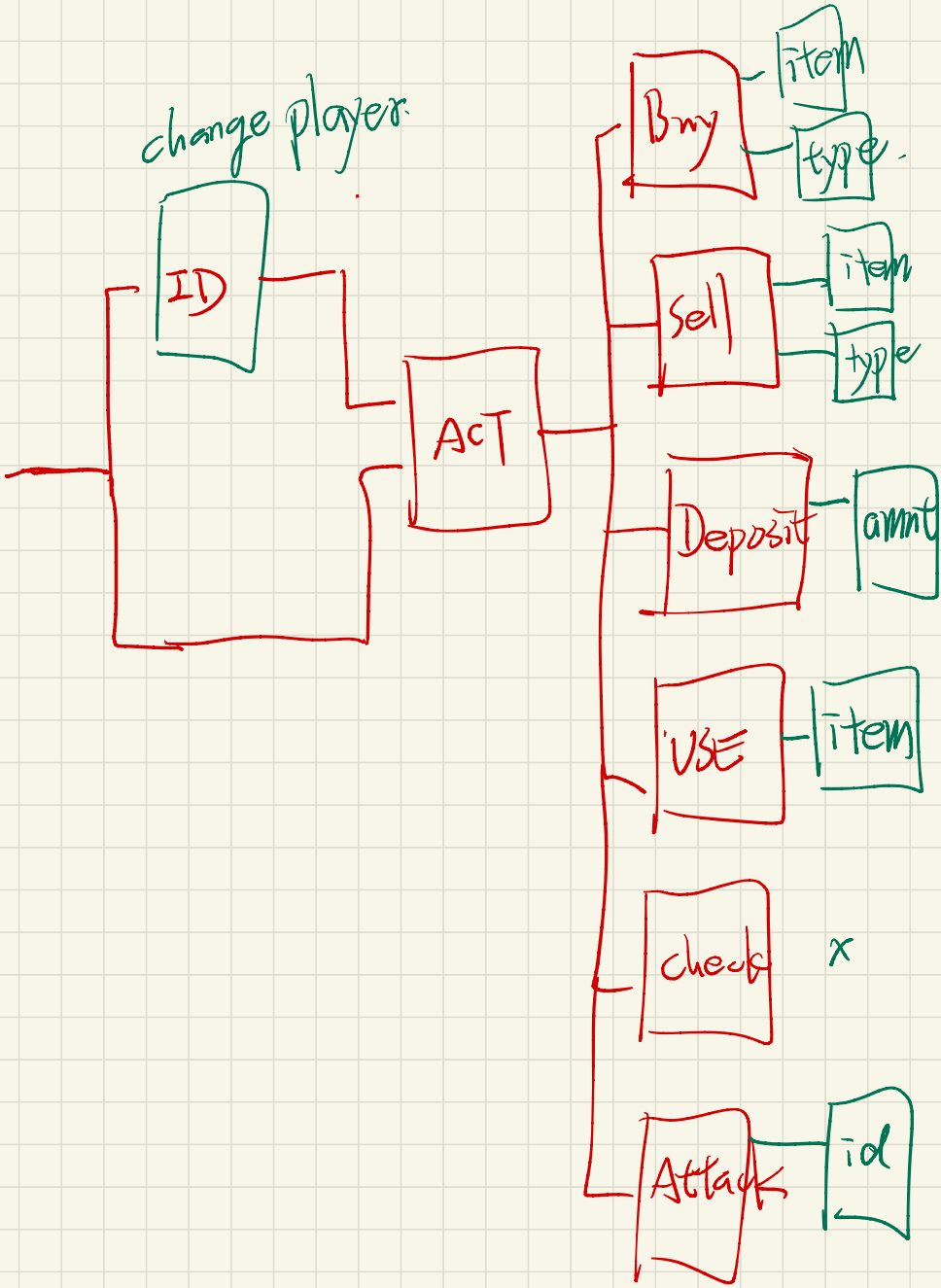
- Upload your design on e3 before 23:59 on 5/08
- Upload your design and package:
  Usertype_PKG_iclabXXX.sv (ex: Usertype_PKG_iclab999.sv)
  clock_period_iclabXXX.txt (ex: 8.0_iclab999.txt)
  pokemon_iclabXXX.sv (ex: pokemon_iclab999.sv)
  bridge_iclabXXX.sv (ex: bridge_iclab999.sv)
- If the uploaded files violating the naming rule, you will get 5 deduct points.

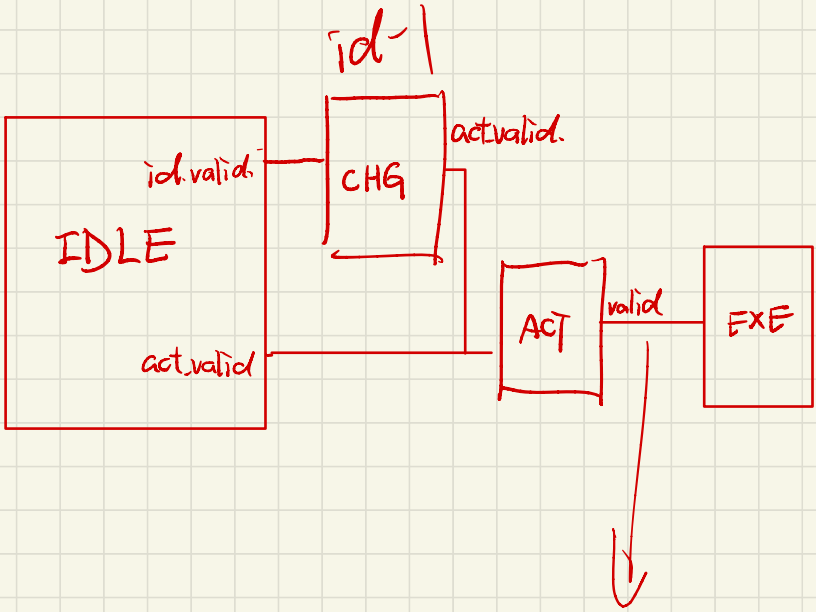
## Reference Waveform

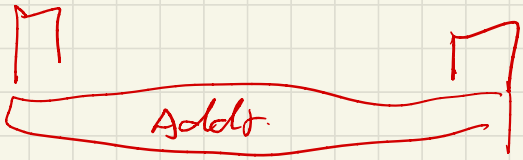
Please refer to Lab08_exercise_note.pdf

※ **Since Lab09 is about pattern with SystemVerilog, so if you need to share your pattern with others, please 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
```


change player
ID
ACT
Buy
item
type
Sell
item
type
Deposit
amnt
USE
item
check
x
Attack
id




$$3 \times (14835 + 67150) = 245955$$

$$5 \times (14825 + 48078) = 318015$$

$$26 \times (14889 + 72991) = 228440$$