NYCU-DCS-2025

HW05

Design: AXI Bus and Interconnect (AXI_inter)

Data Preparation

1. Extract files from TA's directory:

   % tar xvf ~DCSTA01/HW05.tar

2. The extracted LAB directory contains:
   a. **00_TESTBED**
   b. **01_RTL**
   c. **02_SYN**
   d. **03_GATE**
   e. **09_SUBMIT**

Design Description

The **AXI Bus** is part of ARM's AMBA specification, designed for high-performance, high-frequency SoC communication between masters (like CPUs) and slaves (like memory). The **AXI Interconnect** serves as the central hub that connects multiple masters and slaves, handling address decoding, data routing, and arbitration to ensure efficient and correct data transfer across the system.

In this design, **two masters (Master1 and Master2)** communicate with **a shared slave (DRAM)** through the AXI Interconnect. Each master can independently initiate read and write operations, but the interconnect ensures proper arbitration and prioritization—granting Master1 higher priority when both request access simultaneously. The interconnect manages signal handshakes and routes transactions to the slave, enabling concurrent yet controlled access to shared resources while maintaining AXI protocol compliance.

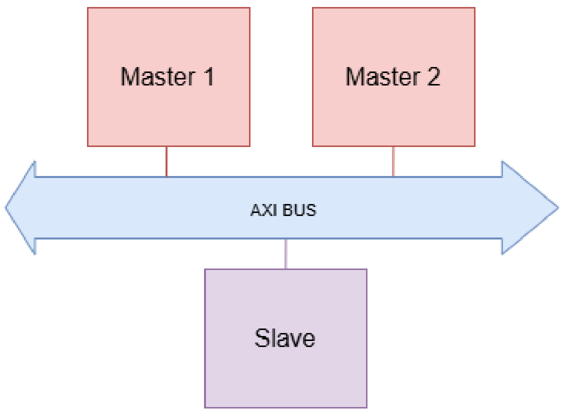


Fig. 1. Overall System

Functional Description

Actions:

- Index Check **(Read DRAM)**
    - Input D:
        1. Action is Index Check
        2. Choose the type of formula.
        3. Select the No. of data in DRAM.
        4. New indices data
            1) Index A
            2) Index B
            3) Index C
            4) Index D
- Update **(Read & Write DRAM)**
    - Input D:
        1. Action is Update
        2. Select the No. of data in DRAM.
        3. Variations of the indices data.
            1) Variation of Index A
            2) Variation of Index B
            3) Variation of Index C
            4) Variation of Index D

Formulas:

Table 1. Formulas

Formula	Input	Content
A	3'd0	$R = \left\lfloor (\frac{I(A)+I(B)+I(C)+I(D)}{4}) \right\rfloor$
B	3'd1	$R = max(I(A), I(B), I(C), I(D)) - \min(I(A), I(B), I(C), I(D))$
C	3'd2	$R = \min(I(A), I(B), I(C), I(D))$
D	3'd3	$R = 1_{\{I(A) \geq 2047\}} + 1_{\{I(B) \geq 2047)\}} + 1_{\{I(C) \geq 2047\}} + 1_{\{I(D) \geq 2047\}}$ $Ex: I(A) = I(B) = 0, I(C) = I(D) = 4095, R = 2$
E	3'd4	$R = 1_{\{I(A) \geq TI(A)\}} + 1_{\{I(B) \geq TI(B)\}} + 1_{\{I(C) \geq TI(C)\}} + 1_{\{I(D) \geq TI(D)\}}$
F	3'd5	$n = argmax_{n \in \{A,B,C,D\}}(G(n)),$ $R = floor(\frac{1}{3}(\sum_{i \neq n, i \in \{A,B,C,D\}} G(i)))$ $Ex: G(A) = G(B) = G(C) = 3, G(D) = 5,$ $R = \frac{G(A)+G(B)+G(C)}{3} = 3$
G	3'd6	$N_0, N_1, N_2, N_3 = ascended.sort(G(A), G(B), G(C), G(D)),$ $R = \left\lfloor (\frac{N_0}{2}) \right\rfloor + \left\lfloor (\frac{N_1}{4}) \right\rfloor + \left\lfloor (\frac{N_2}{4}) \right\rfloor$ $Ex: G(A) = 5, G(B) = G(C) = 4, G(D) = 2$ $N_0, N_1, N_2, N_3 = 2,4,4,5 \;\; R = 3$
H	3'd7	$R = \left\lfloor \frac{G(A)+G(B)+G(C)+G(D)}{4} \right\rfloor$

- R means result.
- I(A), I(B), I(C), I(D) are indices A, B, C, D from dram respectively.
- TI(A), TI(B), TI(C), TI(D) are indices A, B, C, D from input respectively.
- G(A), G(B), G(C), G(D) are |I(A) − TI(A)|, |I(B) − TI(B)|, |I(C) − TI(C)|, |I(D) − TI(D)|

DRAM Storage:

DRAM stores 8-bit words, from address 0x10000 to 0x107FF, but each access must be 64 bits, so addresses must be aligned to 8-byte boundaries. To access data, calculate the address as:

Address = 0x10000 + (Data Count × 8)

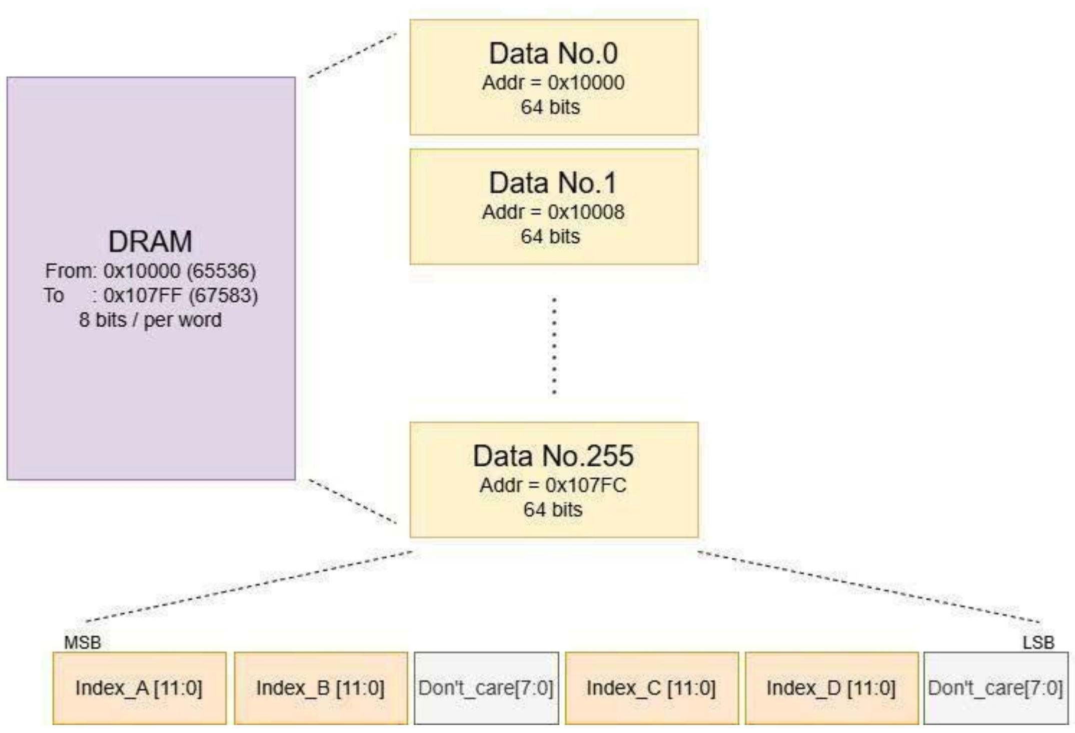
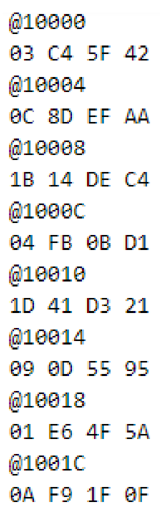


Figure 2. DRAM Data Structure



```
@10000
03 C4 5F 42
@10004
0C 8D EF AA
@10008
1B 14 DE C4
@1000C
04 FB 0B D1
@10010
1D 41 D3 21
@10014
09 0D 55 95
@10018
01 E6 4F 5A
@1001C
0A F9 1F 0F
```

Figure 3. DRAM data

Address	Data (8 bit)
0x10000	0x03
0x10001	0xC4
0x10002	0x5F
0x10003	0x42
0x10004	0x0C
0x10005	0x8D
0x10006	0xEF
0x10007	0xAA
0x10008	0x1B
0x10009	0x14

Category	Value (12 bit)
Index_A	0xAAE
Index_B	0xF8D
Index_C	0x425
Index_D	0xFC4

Table 2. Data storage explanation

Operation Rules Overview:

- There is a delay of 1 to 4 clock cycles between each input transaction.
- The signals **in_valid1** and **in_valid2** are asserted first to indicate that Master1 or Master2 is initiating an operation.
- Following that, a sequence of valid signals will be asserted to indicate what the value in D1 or D2 represents (e.g., action, formula, DRAM number, or index).
- Inputs from Master1 and Master2 may arrive simultaneously. However, **Master1 has higher priority than Master2,** so its results must be output before Master2's.
- Use **out_valid1** or **out_valid2** to indicate which master the **result** belongs to.
- Signal Groups:
    - Master1 signal: in_valid1, action_valid1, formula_valid1, dram_no_valid1, index_valid1, D1, out_valid1.
    - Master2 signal: in_valid2, action_valid2, formula_valid2, dram_no_valid2, index_valid2, D2, out_valid2.
    - Others: result, AXI signals

Index Check

- Trigger Condition: **action_valid** is high and **D[0] = 0**.
- Input Order for **D**:
    - i. Action (1 bit) → D[0]
    - ii. Formula Type (3 bits) → D[2:0]
    - iii. DRAM Data Count (8 bits) → D[7:0]
    - iv. Index A Value (12 bits) → D[11:0]
    - v. Index B Value (12 bits) → D[11:0]
    - vi. Index C Value (12 bits) → D[11:0]
    - vii. Index D Value (12 bits) → D[11:0]
- When **index_valid** is high, the input **D** values represent indices A–D **(range: 0–4095).**
- Based on the specified formula, compute a result using indices A–D from inputs and DRAM, then output the result on the shared result signal.

Update

- Trigger Condition: **action_valid** is high and **D[0] = 1**.
- Input Order for **D**:
    - i. Action (1 bit) → D[0]
    - ii. DRAM Data Count (8 bits) → D[7:0]
    - iii. Index A Value (12 bits) → D[11:0]
    - iv. Index B Value (12 bits) → D[11:0]
    - v. Index C Value (12 bits) → D[11:0]
    - vi. Index D Value (12 bits) → D[11:0]
- When the **index_valid** is high, D represents variations to be added to indices A–D **(range: -2048 to 2047)**.
- Each updated index is **clipped between 0 and 4095** if it exceeds bounds.
- Set bits [3:0] of the **result** signal as follows to indicate out-of-bound updates:
    - i. Bit 3 = 1 → Index A exceeded limits
    - ii. Bit 2 = 1 → Index B exceeded limits
    - iii. Bit 1 = 1 → Index C exceeded limits
    - iv. Bit 0 = 1 → Index D exceeded limits
- All indices out of bounds, result = 14'b1111. All in-bound updates, result = 14'd0

Input

Input	Bit	Description
clk	1	Clock signal, for positive edge triggered design
rst_n	1	Asynchronous active-low reset
in_valid1	1	Indicates Master1 begins sending an operation
in_valid2	1	Indicates Master2 begins sending an operation
action_valid1	1	High when input means select the action for Master1.
action_valid2	1	High when input means select the action for Master2.
formula_valid1	1	High when input means type of formula for Master1.
formula_valid2	1	High when input means type of formula for Master2.
dram_no_valid1	1	High when input means the number of data for Master1.
dram_no_valid2	1	High when input means the number of data for Master2.
index_valid1	1	High when input means index or variation for Master1. (Will pull HIGH 4 times for index A, B, C, D)
index_valid2	1	High when input means index or variation for Master2. (Will pull HIGH 4 times for index A, B, C, D)

D1[11:0]	12	D1 = {11'bX, Action1}
		= {9'bX, Formula_Type1}
		= {4'bX, Data No.1}
		= {Index1 or variation in Index1}

D2[11:0]	12	D2 = {11'bX, Action2}
		= {9'bX, Formula_Type2}
		= {4'bX, Data No.2}
		= {Index2 or variation in Index2}

AR_READY	1	AXI Lite signal
R_VALID	1	AXI Lite signal
R_DATA	64	AXI Lite signal
AW_READY	1	AXI Lite signal
W_READY	1	AXI Lite signal
B_VALID	1	AXI Lite signal

Output

Output	Bit	Description
out_valid1	1	Should set to high when your output for Master1 is ready. **out_valid1** will be high for **only one** cycle.
out_valid2	1	Should set to high when your output for Master2 is ready. **out_valid2** will be high for **only one** cycle.
result	12	Output value according to different action.
AR_VALID	1	AXI Lite signal
AR_ADDR	17	AXI Lite signal
R_READY	1	AXI Lite signal
AW_VALID	1	AXI Lite signal
AW_ADDR	17	AXI Lite signal
W_VALID	1	AXI Lite signal
W_DATA	64	AXI Lite signal
B_READY	1	AXI Lite signal

SPEC

Top module

1. Top module name: **AXI_inter** (File name: **AXI_inter.v**)
2. The execution latency is limited to **1000 cycles** for a single pattern. The latency is the clock cycles between the falling edge of the fourth index_valid to the last rising edge of out_valid.

Reset

3. It is **asynchronous reset** and **active-low** architecture. If you use synchronous reset (considering reset after clock starting), you may fail to reset signals.
4. The reset signal (**rst_n**) would be given only once at the beginning of simulation.
5. All output signals must be **zero/low** with the asynchronous reset when rst_n is low.

Input Signals

6. Each master will use **5 valid signals + 1 data signal**
7. The 5 valid input signals will not overlap with each other.
8. The next input valid signal will be valid for 1~4 cycles after a input valid signal or out_valid signal fall.
9. All input signals are synchronized at **negative edge of the clock**.

Output Signals

10. All output signals should be synchronized at **positive edge of clock**.
11. The TA's pattern will capture your output for checking at clock negative edge.
12. Each master will use **1 valid signals + 1 shared data signal**
13. Raise **out_valid** only when all input_valid signals are transmitted and your design has done current input operation (out_valid cannot overlap with the all input valid).
14. out_valid should be high for **exactly one** cycle
15. out_valid1 **cannot overlap** with out_valid2 at any time

DRAM

16. Please refer to "AXI_introduction" in HW04 for detailed description of AXI signals.
17. TA's pattern will not check data stored in the DRAM.
18. You may set DRAM latency to 1 clock cycle to simplify the design process. However, the **performance** will be calculated under provided DRAM settings

Synthesis

19. You are **not** allowed to modify syn.tcl, except for cycle time (CYCLE).
20. You **can** adjust your clock period, but the maximum period is **20 ns**. The precision of clock period is 0.1, for example, 4.5 is allowed, 4.55 is not allowed.
21. The area is limited in **100,000**.
22. The synthesis time should be less than **2 hours**.
23. The synthesis result of data type **cannot** include any **latches** (using ctrl+F to find the term of "**Latch**" or using the command ./08_check in 02_SYN/).
24. After synthesis, you can check AXI_inter.timing. The timing report should be **non- negative (MET)**.

Gate-level simulation

25. The gate-level simulation **cannot** include any **timing violations**.
26. The cycle time used in Gate-Level Simulation must be the **same** as the cycle time used in synthesis.

Supplement

27. In this assignment, you are **NOT** allowed to use DesignWare IP.
28. Please check whether there is any wire/reg/submodule being named as "error", "fail", "pass", "congratulations", "latch". All letters used in the formats of uppercase or lowercase is prohibited. If there is, you will **fail** this homework.
29. Don’t write Chinese or other language comments in the file you sent.
30. Any error messages during synthesis and simulation, regardless of the result will lead to failure in this lab.
31. The score is only based on the final version you submit to 09_SUBMIT in your last upload. Please ensure that you successfully upload the latest version.

Block Diagram

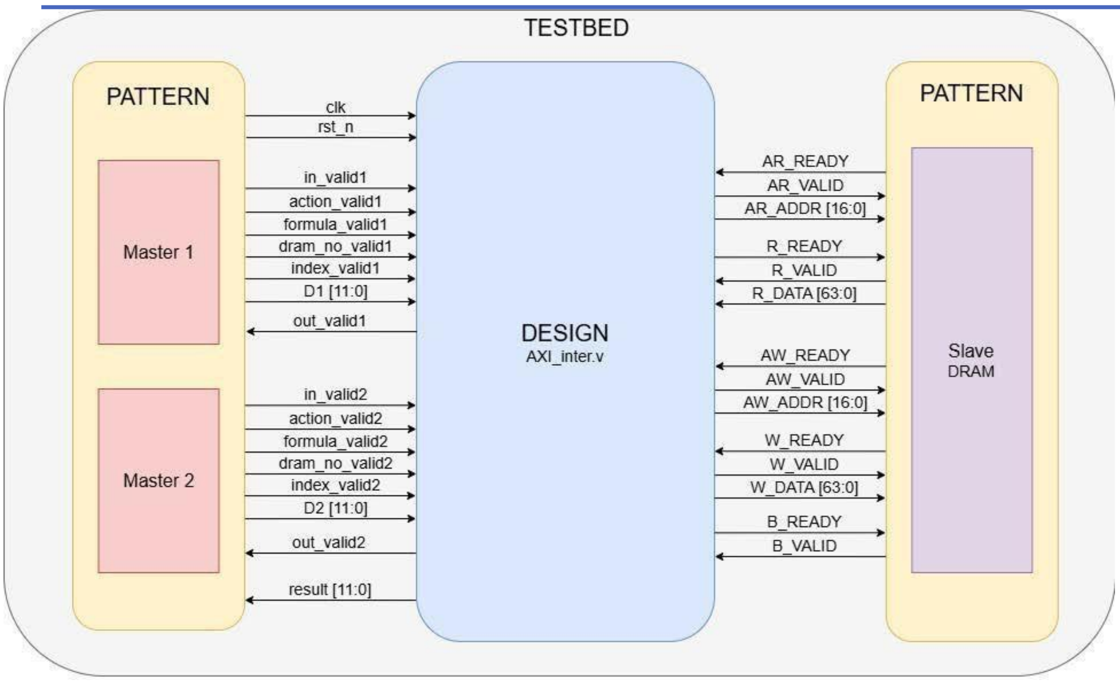


Figure 4. System Block Diagram

Homework Upload

1. Please use the commands we provided to tar whole file under 09_SUBMIT/ and to submit your homework
   - **1st_demo deadline : 2025/5/22 (Thu.) 12:00:00**
   - **2nd_demo deadline : 2025/5/29 (Thu.) 12:00:00**
2. Please check the homework file again after you upload it. If you upload the wrong file, you will **fail** this homework.
3. When TA demos your design, a hidden dram.dat will be included.

Grading Policy

The performance is determined by the area, cycle time and **gate-level** simulation latency of your design. The less cost your design has, the higher grade you get.

The score is only based on the demo result you submit with 09_SUBMIT. Please ensure that you successfully upload the latest version.

1. Function Validity: 70%
   - PASS 01_RTL, 02_SYN, and 03_GATE

2. Performance: 30%
   - **Performance = Area *(Execution_Latency * Cycle_Time)** ^{2}

Command List

- ❖ Verilog RTL simulation (01_RTL/):
  - ◆ **./01_run_vcs_rtl**
- ❖ Synthesis (02_SYN/):
  - ◆ **./01_run_dc_shell**
  - ◆ **./08_check**
- ❖ Gate level simulation (03_GATE/):
  - ◆ **./01_run_vcs_gate**
- ❖ Submit your files (09_SUBMIT/):
  - ◆ **./00_tar 10.0**
    - ● 10.0 should be replaced with the cycle time you use.
  - ◆ **./01_submit**
    - ● You must type '**y**' when you are ready to hand in your homework.
  - ◆ **./02_check**
- ❖ Waveform for debug:
  - ◆ **nWave &**
  - ◆ find *.fsdb
  - ◆ shift+L for reloading the *.fsdb file after you simulate your design again.

Reference Waveform

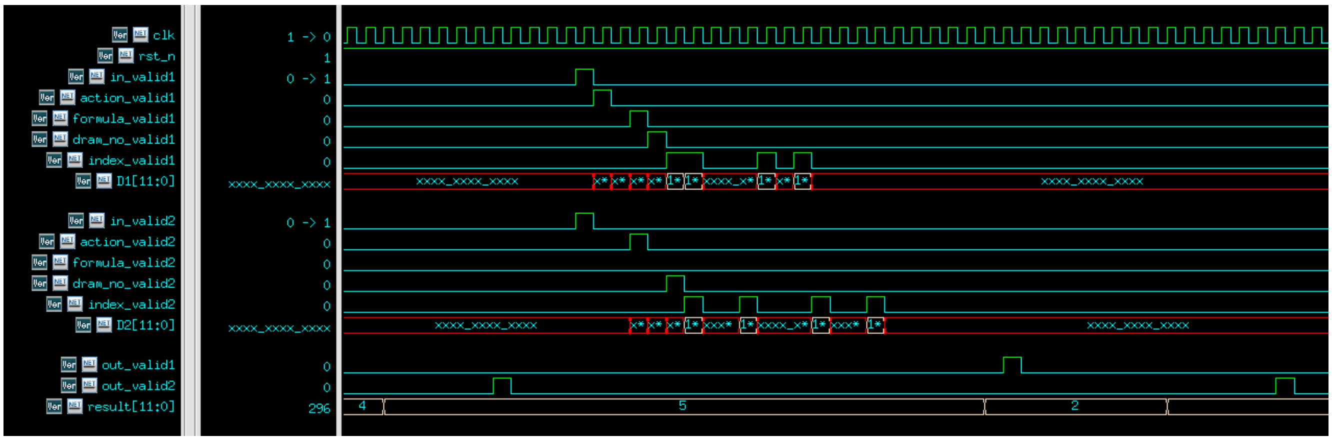
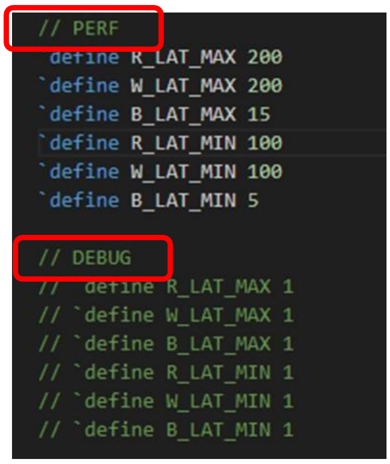


Figure 5. Example waveform

Debug

You can set the DRAM latency to 1 cycle in "00_TESTBED/pseudo_DRAM.vp" to make the waveform easier to read and better observe the handshake behavior.



```
// PERF
`define R_LAT_MAX 200
`define W_LAT_MAX 200
`define B_LAT_MAX 15
`define R_LAT_MIN 100
`define W_LAT_MIN 100
`define B_LAT_MIN 5

// DEBUG
// `define R_LAT_MAX 1
// `define W_LAT_MAX 1
// `define B_LAT_MAX 1
// `define R_LAT_MIN 1
// `define W_LAT_MIN 1
// `define B_LAT_MIN 1
```

Figure 6. DRAM latency settings