

Explanation for Homework IV

Speaker: Justin (Cheng-Hong Pai)

Advisor: Lih-Yih Chiou

Date: 2024/12/4

Due day : 2025/1/1 (Wed.) 23:59

Outline

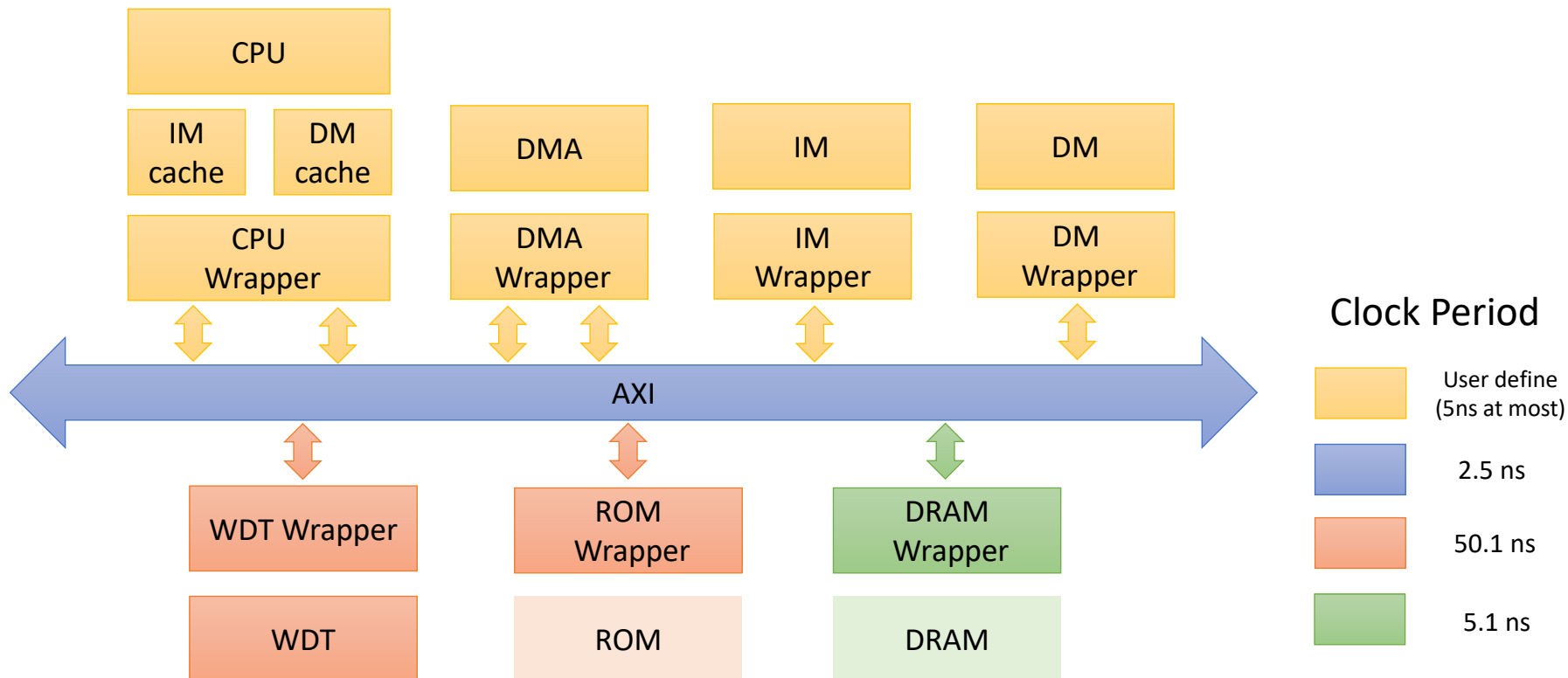
- AXI CDC
- Cache
- IO pad
- System
- Verification
- Simulation Commands
- Submission rule

AXI CDC - System Overview

Why need AXI CDC

- In complex systems, various components may operate at different clock frequencies to optimize their performance and power consumption.

There are total 4 clock domains in this HW4

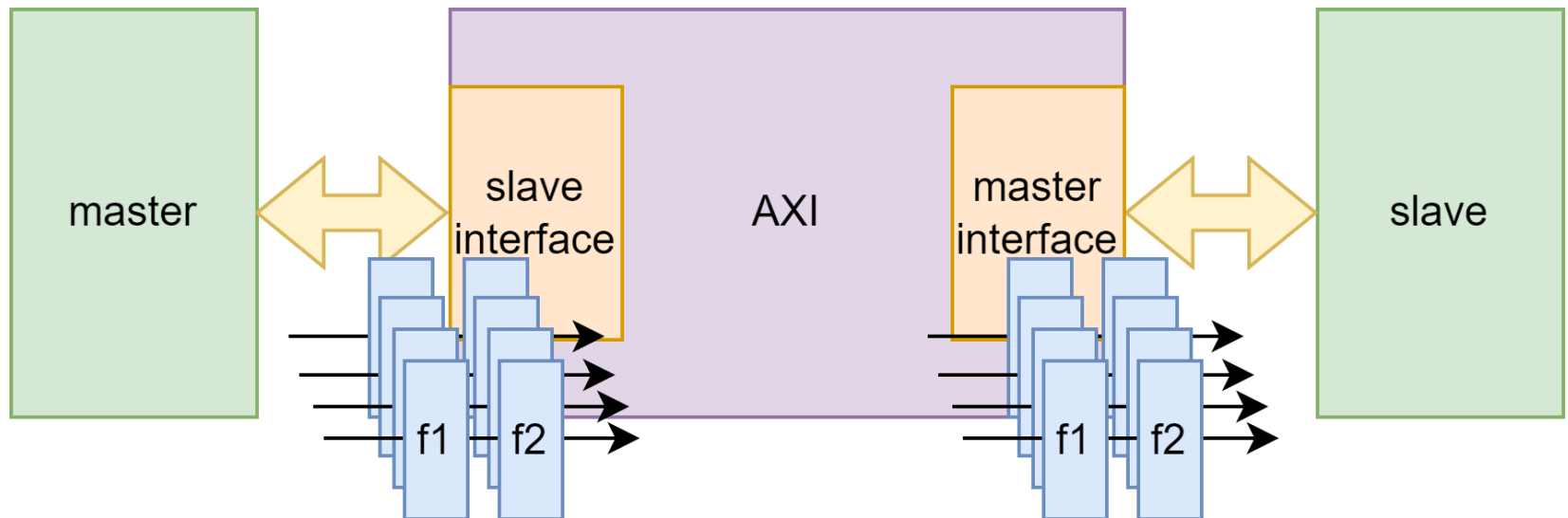


AXI CDC – Implementation tips

Let's take a close look at AXI

What is the problem ?

Ac_unsync02 (multi-bit signal unsynchronized)



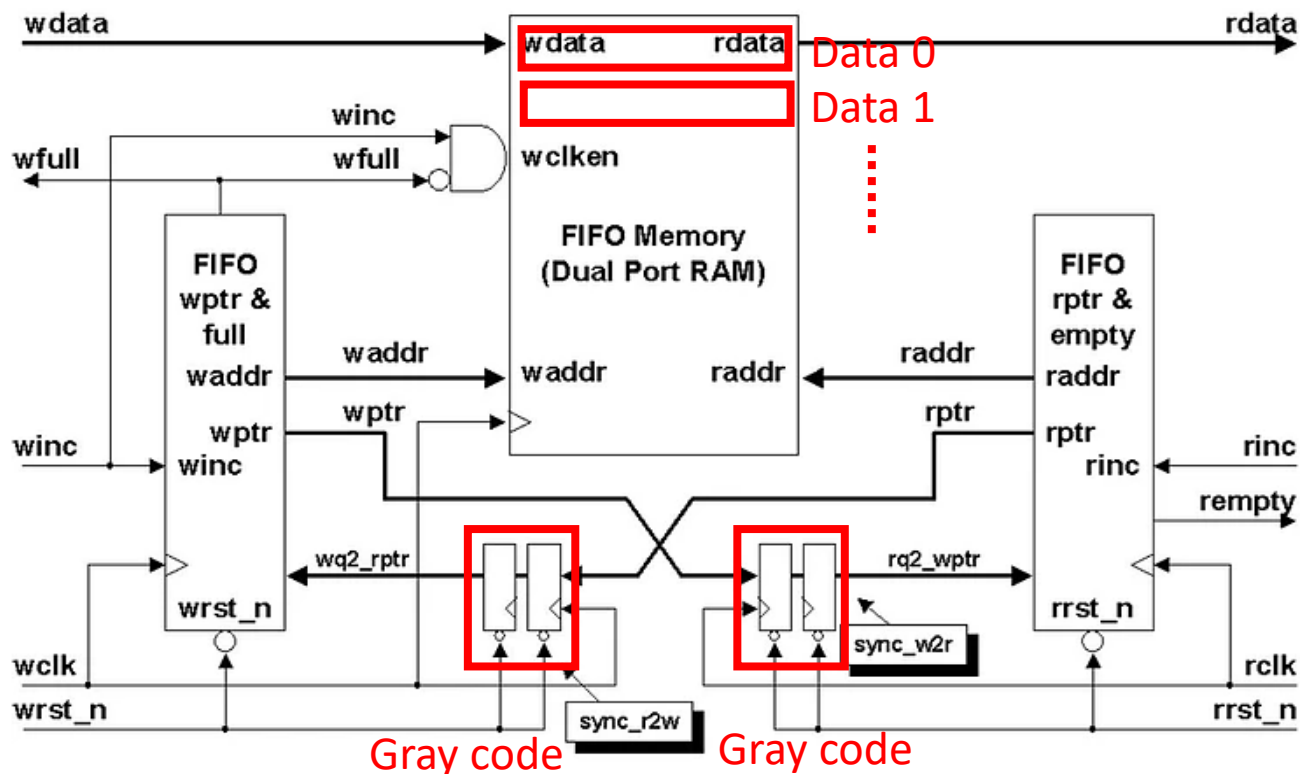
AXI CDC – Implementation tips

Let's take a close look at AXI

How to solved the problem ?

The most straightforward approach for multi-bits clock domain crossing is using a FIFO CDC.

- No synchronization is required for the data.
- The only signal that needs synchronization is the pointer in Gray code.

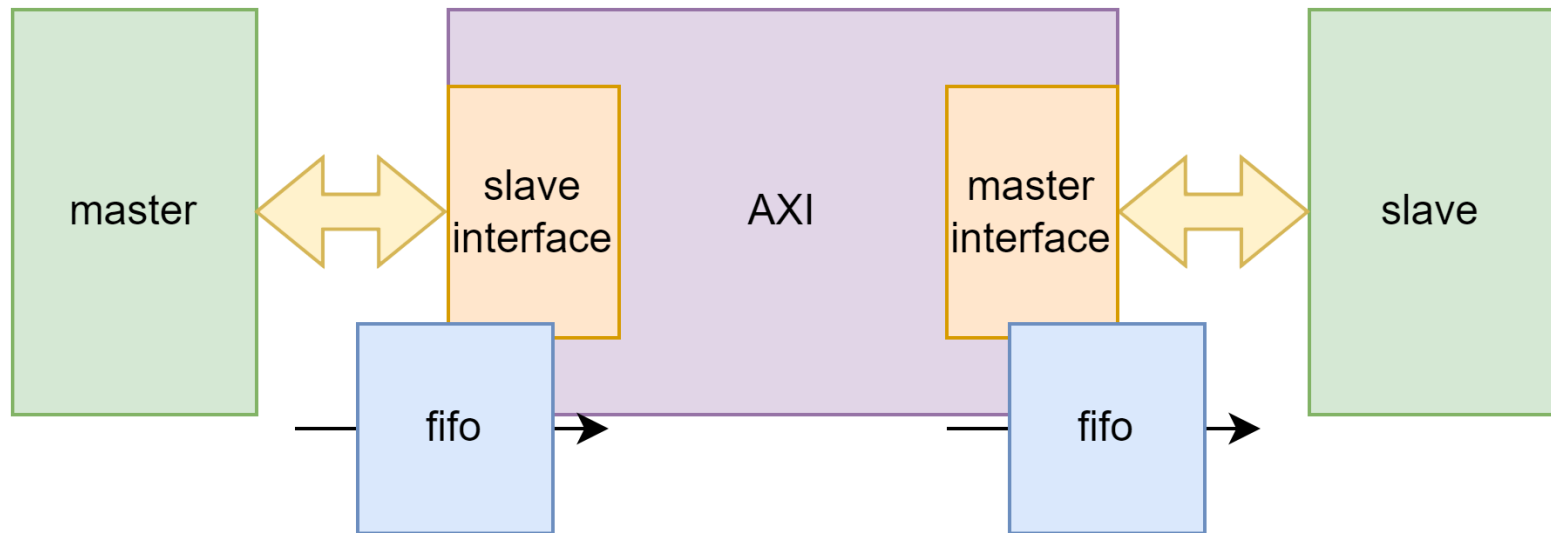


AXI CDC – Implementation tips

▶ Let's take a close look at AXI

▶ How to solved the problem ?

■ The most straightforward approach for clock domain crossing is using a FIFO CDC.



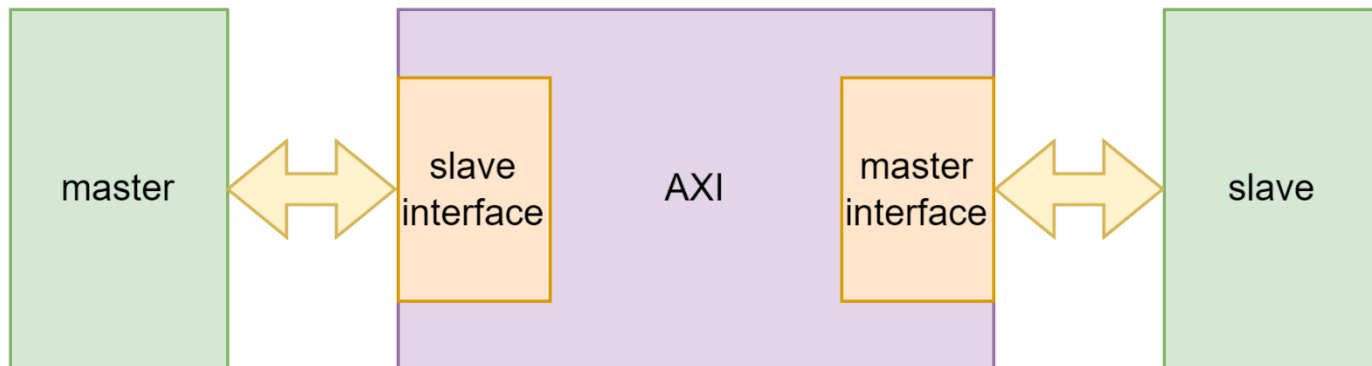
AXI CDC – Implementation tips

Let's take a close look at AXI

Previous work (HW2)

- AXI operates within the same domain as other modules, such as the CPU, DRAM, ROM ...

How is the AXI handshake implemented in HW2?

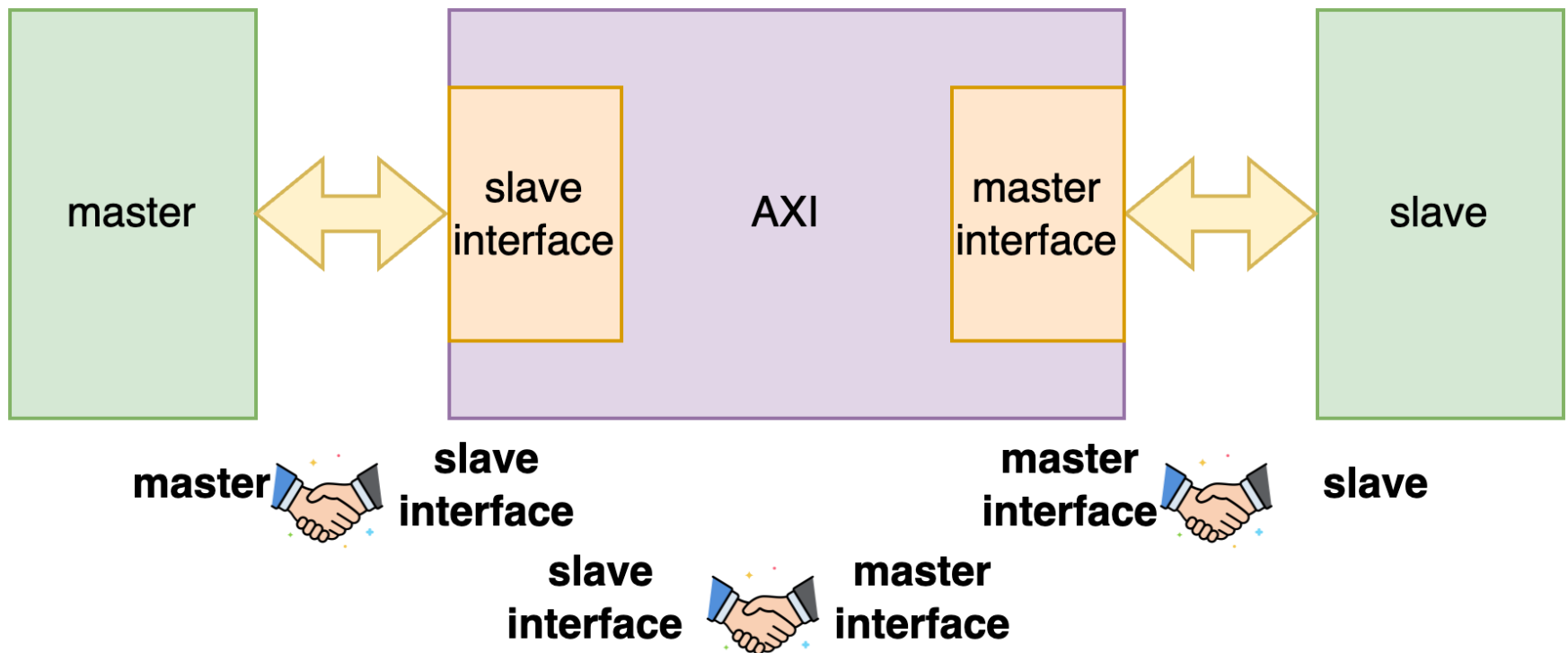


AXI CDC – Implementation tips

Let's take a close look at AXI

In HW4

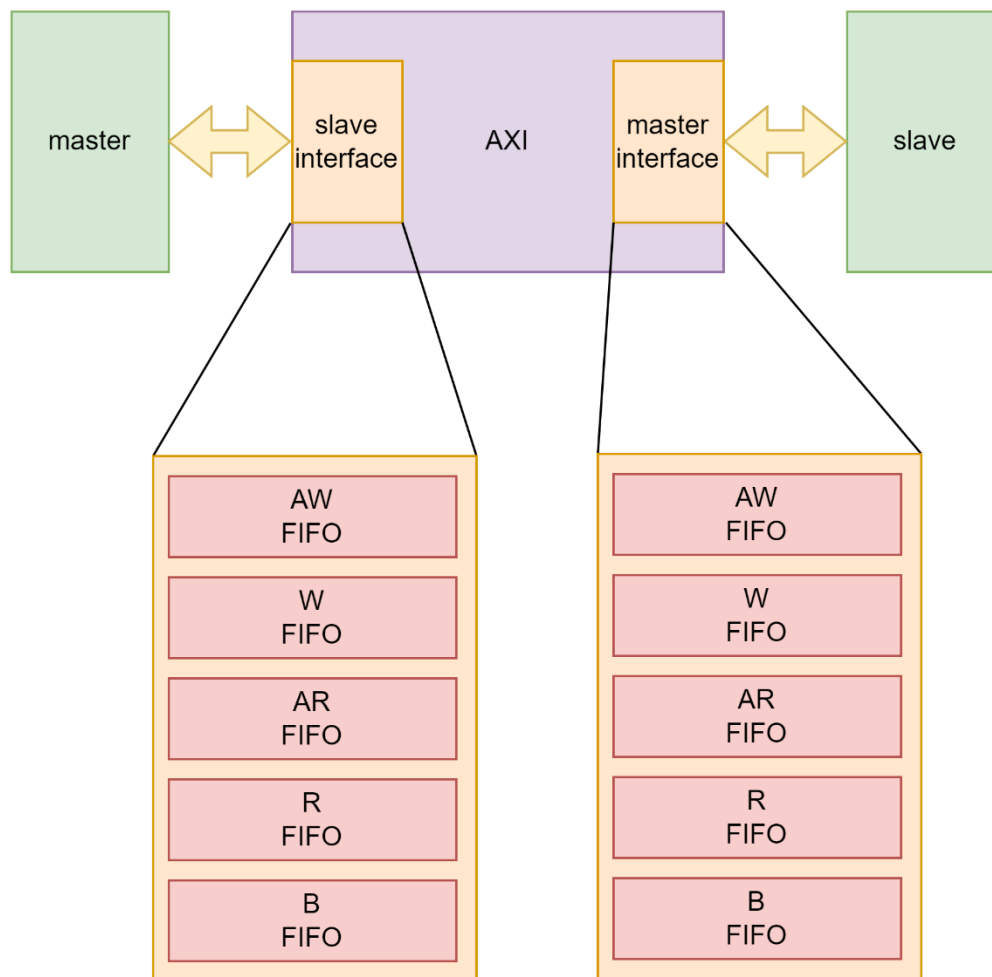
- AXI operates in a clock domain distinct from that of other modules, including the CPU, DRAM, and ROM ...
- How is the AXI handshake implemented in HW4?



AXI CDC – Implementation tips

▶ Let's take a close look at AXI

▶ In HW4



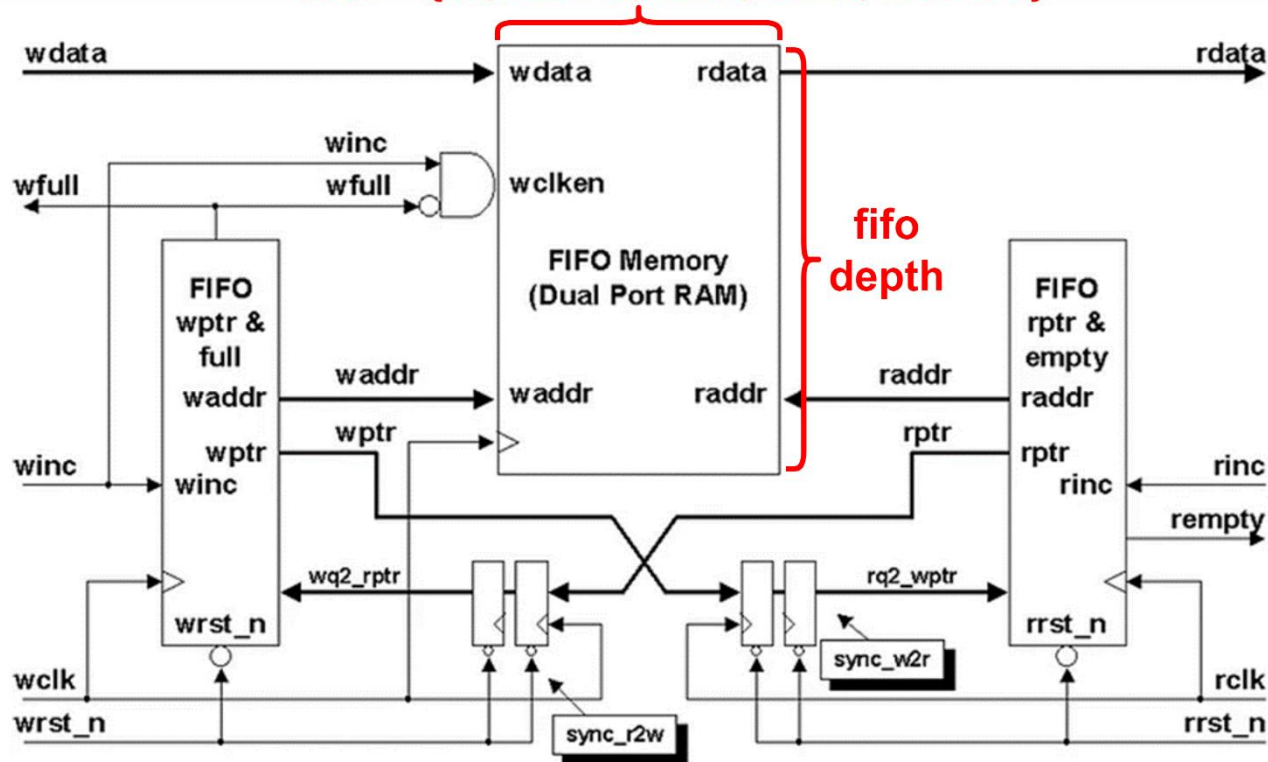
AXI CDC – Implementation tips

Let's take a close look at AXI

➤ In HW4 (AW channel FIFO)

- This is just a suggestion; you can also explore alternative methods.
- Other channel FIFOs are implemented in similar manner.

data = {ID, ADDR, LEN, SIZE, BURST}

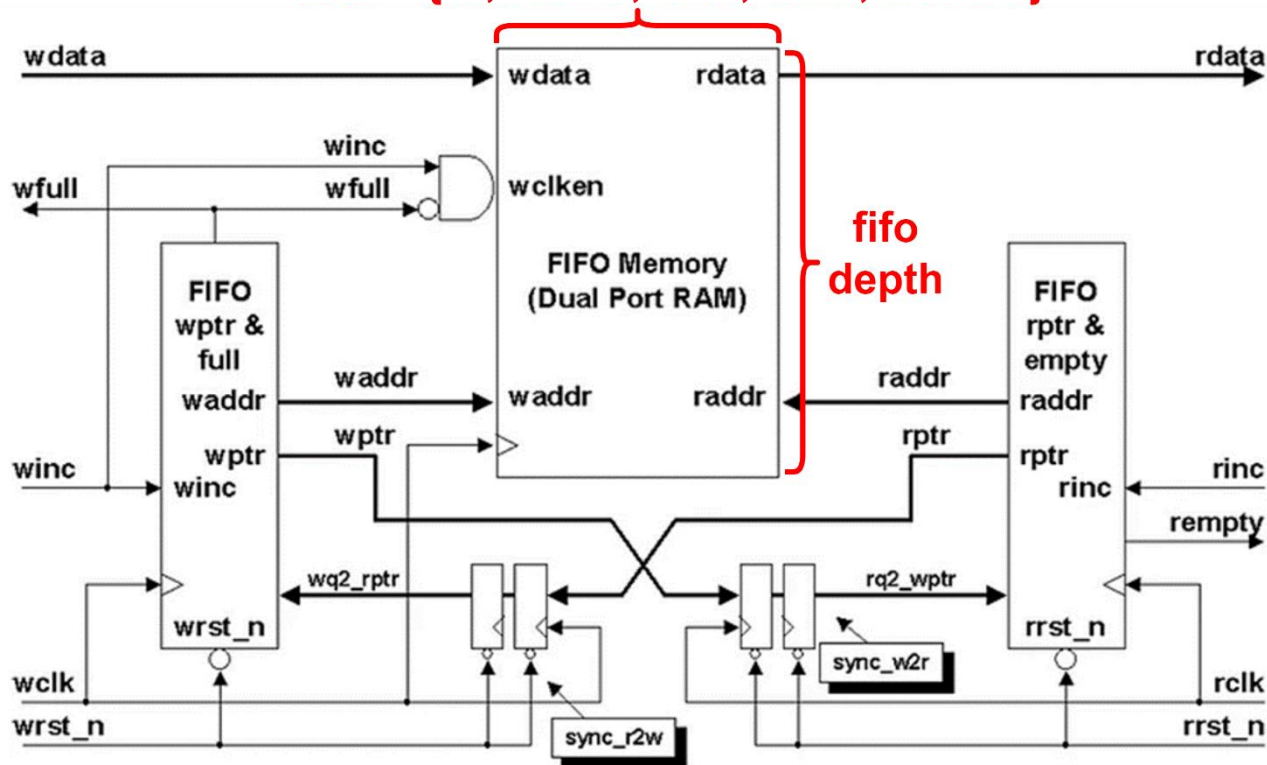


AXI CDC – Implementation tips

Question 1

Is it necessary to synchronize the valid and ready signals ?

data = {ID, ADDR, LEN, SIZE, BURST}

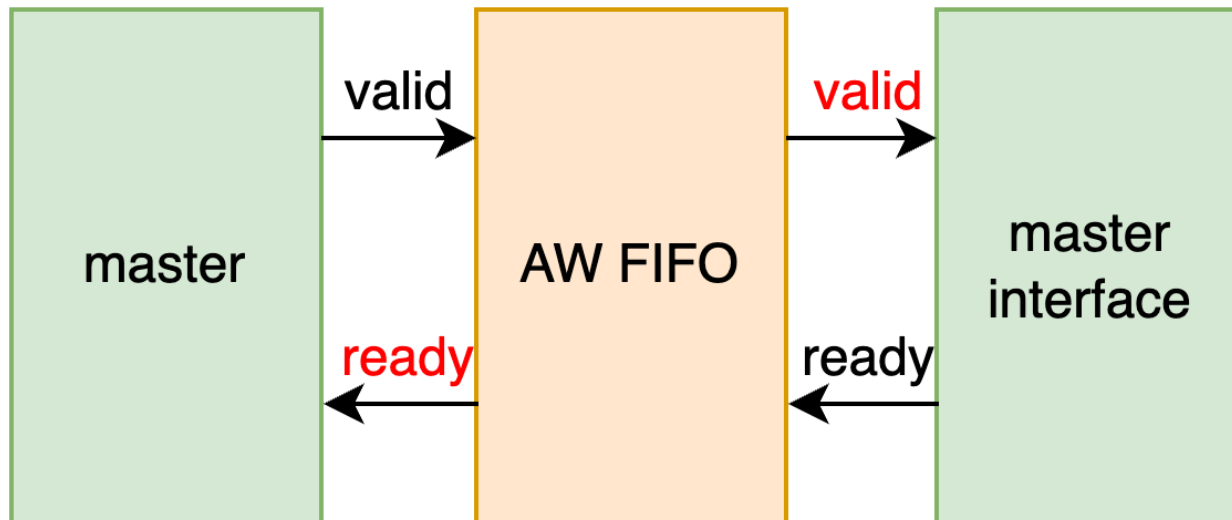


AXI CDC – Implementation tips

Question 2

➤ How to build AW FIFO ready and valid signal ?

■ Hint : AW FIFO depth



AXI CDC – Implementation tips

► FIFO CDC suggested I/O pins

signal	I/O	Bit width	Desc.
wclk	input	1	Write domain clock
wrst	input	1	Write domain reset Active high
wpush	input	1	Data push into FIFO
wdata	input	DATASIZE	Write data
wfull	output	1	FIFO full
rclk	input	1	Read domain clock
rrst	input	1	Read domain reset Active high
rpop	input	1	Data pop from FIFO
rdata	output	DATASIZE	Read data
rempty	output	1	FIFO empty

AXI CDC – Implementation tips

► FIFO CDC assertion (**add in `async_fifo.sv`**)

► Verify the occurrence of invalid operations throughout the simulation.

■ Push data in FIFO when FIFO is full

```
// ASSERTION CHECK
WRITE_FULL_CHECK :
  assert property (
    @(posedge wclk) disable iff (wrst) (!wpush || !wfull))
    else $error("\n *** Assertion failed: Push operation when FIFO is full. *** \n");
```

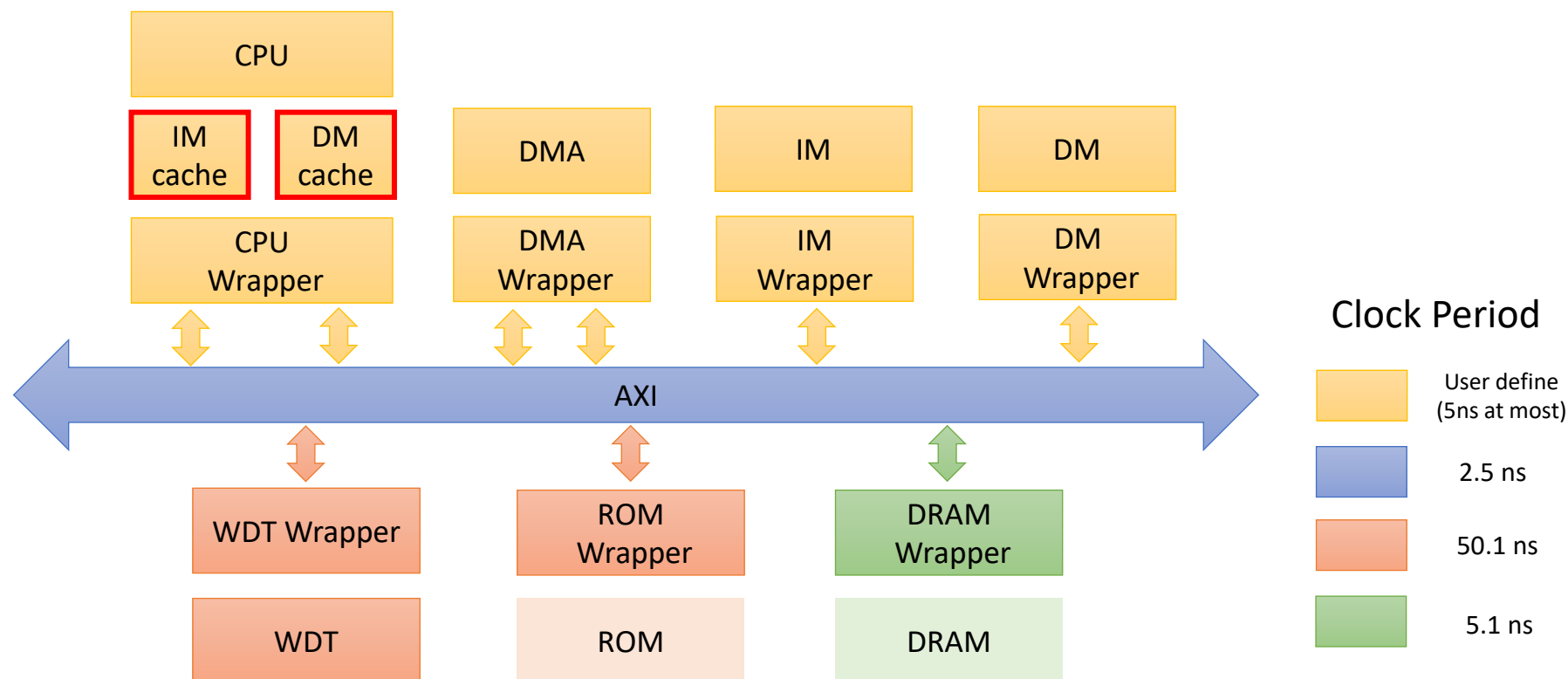
■ Pop data from FIFO when FIFO is empty

```
READ_EMPTY_CHECK :
  assert property (
    @(posedge rclk) disable iff (rrst) (!rpop || !rempty))
    else $error("\n *** Assertion failed: Pop operation when FIFO is empty. *** \n");
```

Cache – basic requirement

Why need the cache

- Accessing data from main memory involves **higher latency** compared to accessing data from a cache. Caches help mitigate the impact of memory latency by providing a faster **intermediate storage for frequently used information**.



Cache – basic requirement

Implementing the connection of two caches beneath the CPU.

Instance

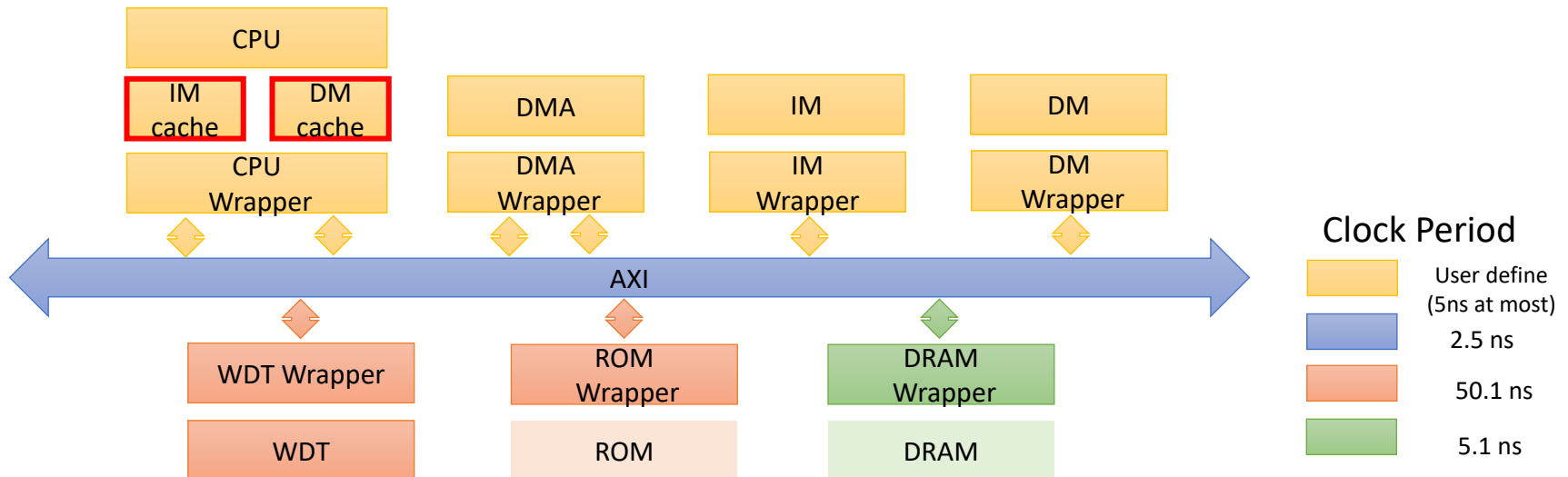
- L1CI (IM cache)
- L1CD (DM cache)

Read Policy

- Read miss : read allocate

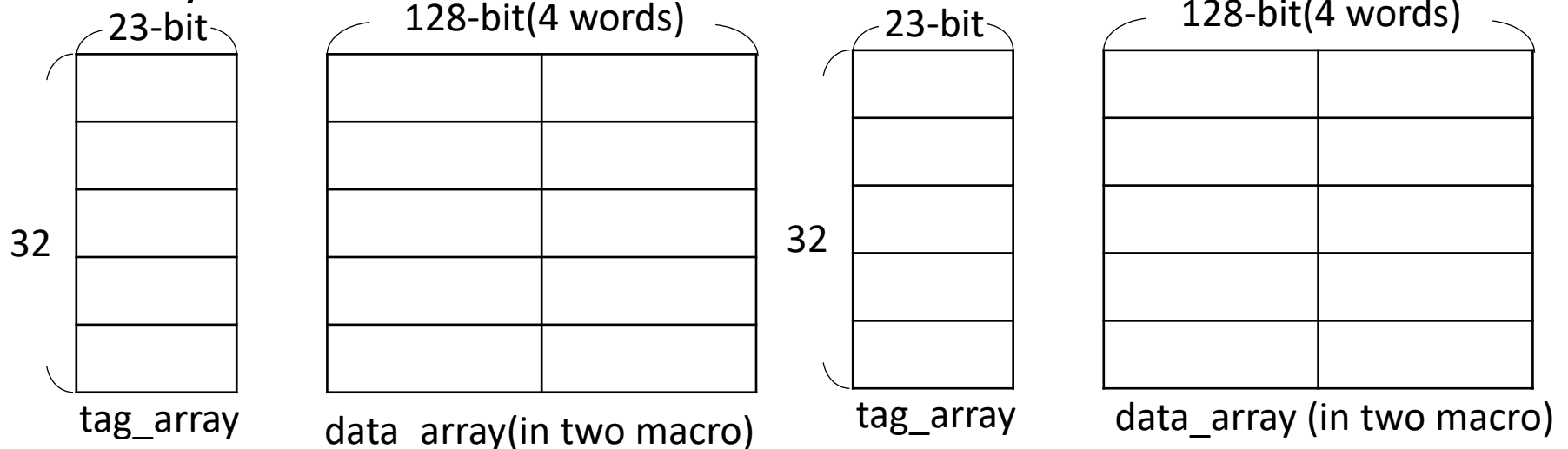
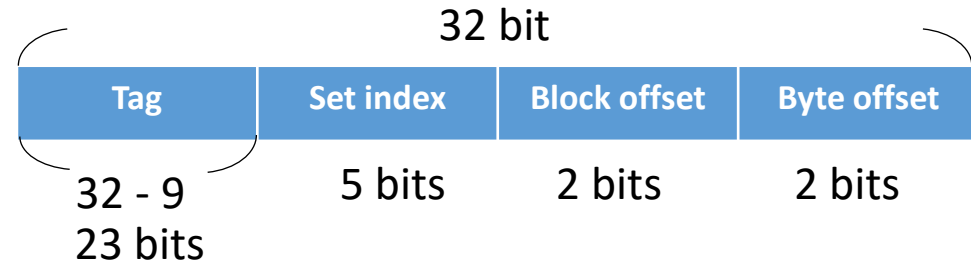
Write Policy

- Write hit : write through
- Write miss : write no allocate



Cache – basic requirements

- Cache size : 1kB
- 2 way set associative
- Cache replacement policies
 - LRU (Least Recently Used Cache)
- Block size (cache line size)
 - 16Byte == 128bit (4 words)
- Entry : 32



Hit = valid && (TA_out equal core_addr[31:10])

Cache – basic requirements

- Show the hit rate of your instruction cache and data cache

$$\text{cache hit rate} = \frac{\text{Number of cache hits}}{\text{Number of cache hits} + \text{Number of cache misses}}$$

Cache – specification L1CI

L1C_inst	clk	input	1	clock
	rst	input	1	reset (active high)
	core_addr	input	32	address from CPU
	core_req	input	1	memory access request from CPU
	core_write	input	1	write signal from CPU
	core_in	input	32	data from CPU
	core_type	input	3	write/read byte, half word, or word (listed in include/def.svh) from CPU
	I_out	input	32	data from CPU wrapper
	I_wait	input	1	wait signal from CPU wrapper
	core_out	output	32	data to CPU
	core_wait	output	1	wait signal to CPU
	I_req	output	1	request to CPU wrapper
	I_addr	output	32	address to CPU wrapper
	I_write	output	1	write signal to CPU wrapper
	I_in	output	32	write data to CPU wrapper
	I_type	output	3	write/read byte, half word, or word (listed in include/def.svh) to CPU wrapper
	valid	logic	64	valid bits of each cache line
	index	logic	6	address to tag array/data array
	TA_write	logic	1	write signal to tag_array
	TA_read	logic	1	read signal to tag_array
	TA_in	logic	22	write data to tag_array
	TA_out	logic	22	read data from tag_array
	DA_write	logic	16	write signal to data_array
	DA_read	logic	1	read signal to data_array
	DA_in	logic	128	write data to data_array
	DA_out	logic	128	read data from data_array

Inputs and outputs of module L1C_data/L1C_inst has declared in src/L1C_data.sv and src/L1C_inst.sv.
You can modify input and output bit width if needed.

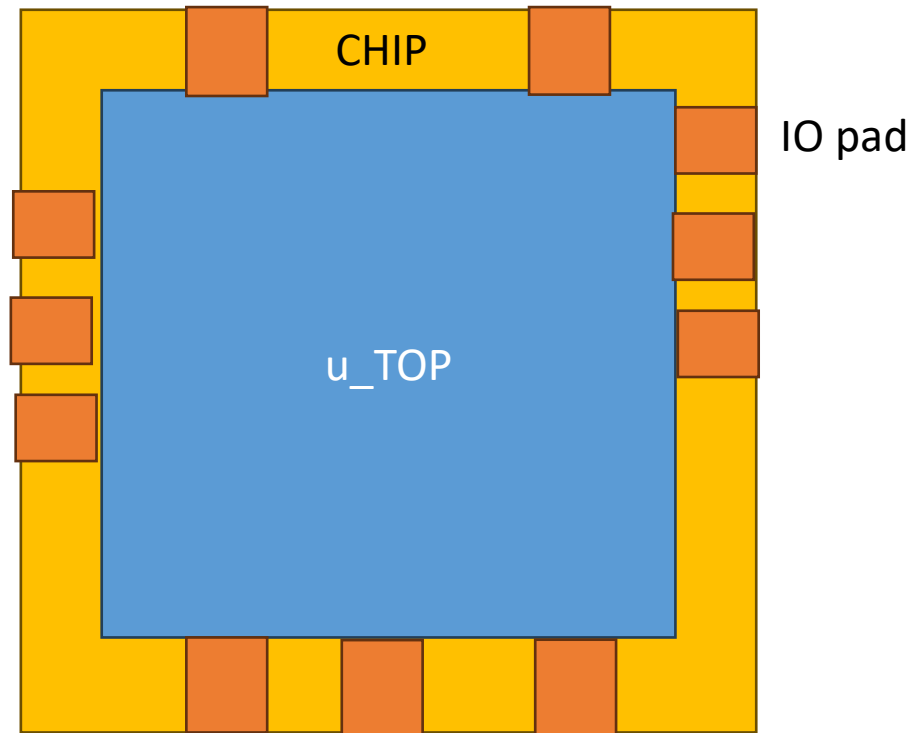
Cache – specification L1CD

L1C_data	clk	input	1	clock
	rst	input	1	reset (active high)
	core_addr	input	32	address from CPU
	core_req	input	1	memory access request from CPU
	core_write	input	1	write signal from CPU
	core_in	input	32	data from CPU
	core_type	input	3	write/read byte, half word, or word (listed in include/def.svh) from CPU
	D_out	input	32	data from CPU wrapper
	D_wait	input	1	wait signal from CPU wrapper
	core_out	output	32	data to CPU
	core_wait	output	1	wait signal to CPU
	D_req	output	1	request to CPU wrapper
	D_addr	output	32	address to CPU wrapper
	D_write	output	1	write signal to CPU wrapper
	D_in	output	32	write data to CPU wrapper
	D_type	output	3	write/read byte, half word, or word (listed in include/def.svh) to CPU wrapper
	valid	logic	64	valid bits of each cache line
	index	logic	6	address to tag array/data array
	TA_write	logic	1	write signal to tag_array
	TA_read	logic	1	read signal to tag_array
	TA_in	logic	22	write data to tag_array
	TA_out	logic	22	read data from tag_array
	DA_write	logic	16	write signal to data_array
	DA_read	logic	1	read signal to data_array
	DA_in	logic	128	write data to data_array
	DA_out	logic	128	read data from data_array

Inputs and outputs of module L1C_data/L1C_inst has declared in src/L1C_data.sv and src/L1C_inst.sv.
You can modify input and output bit width if needed.

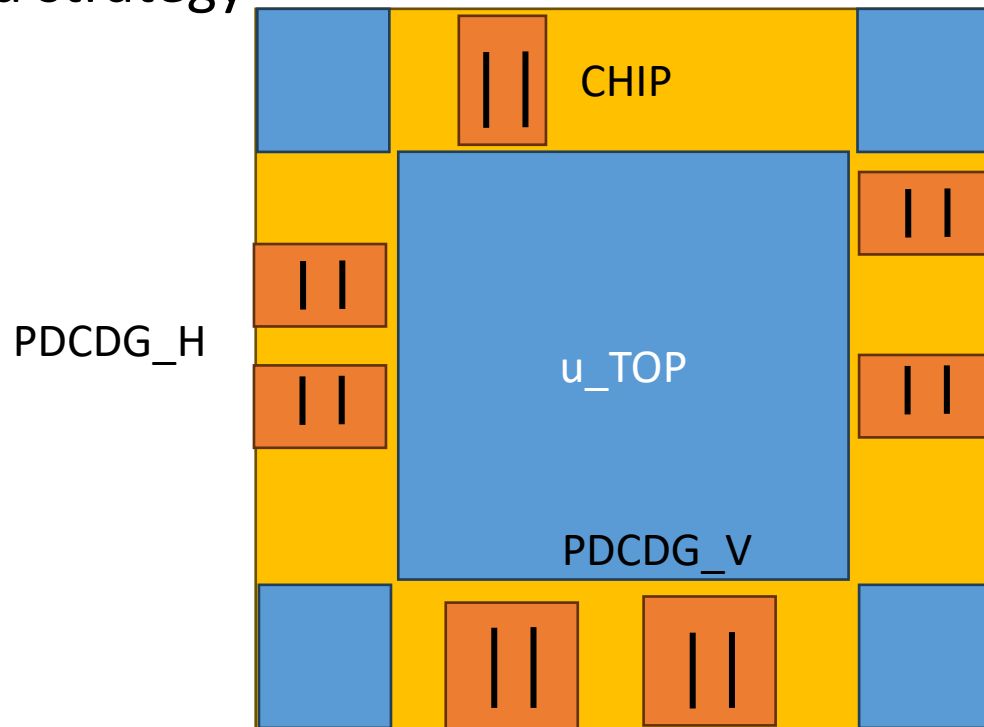
iopad in CHIP.sv

- Add iopad between Testbench & top.sv
- Synthesis and Simulation with iopad in HW4



iopad in CHIP.sv

- ▶ uni-poly direction in ADFP N16
- ▶ There is two kind of iopad : PDCDG_H & PDCDG_V
 - ▶ Top & bottom add PDCDG_V
 - ▶ Left & right add PDCDG_H
 - ▶ You can have your own iopad strategy



iopad in CHIP.sv

➡ Add iopad in CHIP.sv

➤ output : OEN = 0 , IE = 0

➤ input : OEN = 1 , IE = 1

➡ Example :

```
PDCDG_V opad_DRAM_D27 (.OEN(1'b0), .IE(1'b0), .I(DRAM_D_o[27]), .PAD(DRAM_D[27]), .C());
PDCDG_V opad_DRAM_D28 (.OEN(1'b0), .IE(1'b0), .I(DRAM_D_o[28]), .PAD(DRAM_D[28]), .C());
PDCDG_V opad_DRAM_D29 (.OEN(1'b0), .IE(1'b0), .I(DRAM_D_o[29]), .PAD(DRAM_D[29]), .C());
PDCDG_V opad_DRAM_D30 (.OEN(1'b0), .IE(1'b0), .I(DRAM_D_o[30]), .PAD(DRAM_D[30]), .C());
PDCDG_V opad_DRAM_D31 (.OEN(1'b0), .IE(1'b0), .I(DRAM_D_o[31]), .PAD(DRAM_D[31]), .C());
PDCDG_V ipad_cpu_clk (.OEN(1'b1), .IE(1'b1), .I(1'b0), .PAD(cpu_clk), .C(cpu_clk_i)); //cpu_c
PDCDG_V ipad_cpu_rst (.OEN(1'b1), .IE(1'b1), .I(1'b0), .PAD(cpu_rst), .C(cpu_rst_i)); //cpu_r
PDCDG_V ipad_axi_clk (.OEN(1'b1), .IE(1'b1), .I(1'b0), .PAD(axi_clk), .C(axi_clk_i)); //DRAM_
```

System

Slave Configuration

Table 1-5 : Slave configuration

NAME	Number	Start address	End address
ROM	Slave 0	0x0000_0000	0x0000_1FFF
IM	Slave 1	0x0001_0000	0x0001_FFFF
DM	Slave 2	0x0002_0000	0x0002_FFFF
DMA	Slave 3	0x1000_0000	0x1002_0400
WDT	Slave 4	0x1001_0000	0x1001_03FF
DRAM	Slave 5	0x2000_0000	0x201F_FFFF

- You should design all slave wrappers !!

ROM

ROM	System signals			
	CK	input	1	System clock
	Memory ports			
	DO	output	32	ROM data output
	OE	input	1	Output enable (active high)
	CS	input	1	Chip select (active high)
	A	input	12	ROM address input
	Memory Space			
	Memory_byte0	reg	8	Size: [0:4095]
	Memory_byte1	reg	8	Size: [0:4095]
	Memory_byte2	reg	8	Size: [0:4095]
	Memory_byte3	reg	8	Size: [0:4095]

DRAM

DRAM	System signals			
	CK	input	1	System clock
	RST	input	1	System reset (active high)
	Memory ports			
	CSn	input	1	DRAM Chip Select (active low)
	WEn	input	4	DRAM Write Enable (active low)
	RASn	input	1	DRAM Row Access Strobe (active low)
	CASn	input	1	DRAM Column Access Strobe (active low)
	A	input	11	DRAM Address input
	D	input	32	DRAM data input
	Q	output	32	DRAM data output
	VALID	output	1	DRAM data output valid
	Memory space			
	Memory_byte0	reg	8	Size: [0:2097151]
	Memory_byte1	reg	8	Size: [0:2097151]
	Memory_byte2	reg	8	Size: [0:2097151]
	Memory_byte3	reg	8	Size: [0:2097151]

★ Row address is 11-bit and column address is 10-bit

Watchdog Timer

Module	Specifications			
WDT	Name	Signal	Bits	Function explanation
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	clk2	input	1	WDT clock
	rst2	input	1	WDT reset (active high)
	WDEN	input	1	Enable the watchdog timer
	WDLIVE	input	1	Restart the watchdog timer
	WTOCNT	input	32	Watchdog timeout count
	WTO	output	1	watchdog timeout

top.sv I/O ports

signal	I/O	Bit width	Desc.
cpu_clk	input	1	Clock for cpu,sensor ctrl, WDT
axi_clk	input	1	AXI clock
rom_clk	input	1	ROM clock
dram_clk	input	1	DRAM clock
cpu_rst	input	1	Active high
axi_rst	input	1	Active high
rom_rst	input	1	Active high
dram_rst	input	1	Active high

top.sv I/O ports

signal	I/O	Bit width	Desc.
ROM_out	input	32	Data from ROM
ROM_read	output	1	ROM output enable
ROM_enable	output	1	Enable ROM
ROM_address	output	12	Address to ROM
DRAM_Q	input	32	Data from DRAM
DRAM_CS _n	output	1	DRAM Chip Select (active low)
DRAM_WEn	output	4	DRAM Write Enable (active low)
DRAM_RAS _n	output	1	DRAM Row Access Strobe (active low)
DRAM_CAS _n	output	1	DRAM Column Access Strobe (active low)
DRAM_A	output	11	Address to DRAM
DRAM_D	output	32	Data to DRAM
DRAM_valid	input	1	DRAM output data valid

CHIP.sv I/O ports (same as top.sv)

signal	I/O	Bit width	Desc.
cpu_clk	input	1	Clock for cpu,sensor ctrl, WDT
axi_clk	input	1	AXI clock
rom_clk	input	1	ROM clock
dram_clk	input	1	DRAM clock
cpu_rst	input	1	Active high
axi_rst	input	1	Active high
rom_rst	input	1	Active high
dram_rst	input	1	Active high

CHIP.sv I/O ports (same as top.sv)

signal	I/O	Bit width	Desc.
ROM_out	input	32	Data from ROM
ROM_read	output	1	ROM output enable
ROM_enable	output	1	Enable ROM
ROM_address	output	12	Address to ROM
DRAM_Q	input	32	Data from DRAM
DRAM_CS _n	output	1	DRAM Chip Select (active low)
DRAM_WEn	output	4	DRAM Write Enable (active low)
DRAM_RAS _n	output	1	DRAM Row Access Strobe (active low)
DRAM_CAS _n	output	1	DRAM Column Access Strobe (active low)
DRAM_A	output	11	Address to DRAM
DRAM_D	output	32	Data to DRAM
DRAM_valid	input	1	DRAM output data valid

Verification

Verification (1/4)

► Prog0

- Booting + Testing I/M/F extension instructions (Provided by TA)

► Prog1

- Booting + Floating point verification

► Prog2

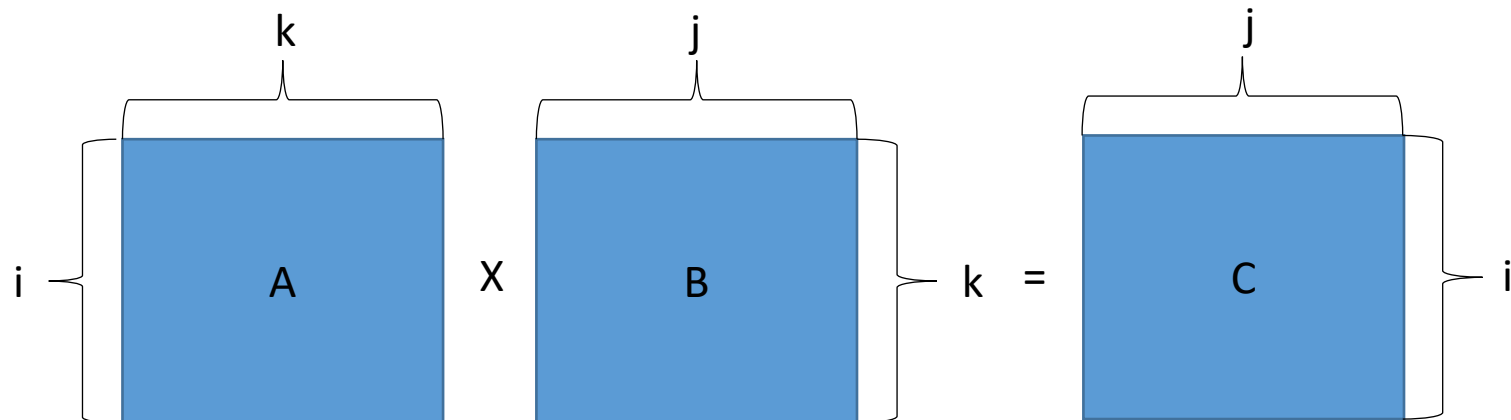
- Booting + Matrix multiplication

► Prog3

- Booting + CPU malfunction + CPU operate normally

Verification(1/2)

Matrix Multiplication



$$\begin{pmatrix} data\cdot1 & data\cdot2 & data\cdot3 \\ data\cdot4 & data\cdot5 & data\cdot6 \end{pmatrix} \cdot \begin{pmatrix} data\cdot7 & data\cdot8 & data\cdot9 & data\cdot10 \\ data\cdot11 & data\cdot12 & data\cdot13 & data\cdot14 \\ data\cdot15 & data\cdot16 & data\cdot17 & data\cdot18 \end{pmatrix} = \begin{pmatrix} result\cdot1 & result\cdot2 & result\cdot3 & result\cdot4 \\ result\cdot5 & result\cdot6 & result\cdot7 & result\cdot8 \end{pmatrix}$$

Verification(2/2)

- Define cpu clock cycle and max cycle count at ./sim/CYCLE_MAX.txt

```
1  `define CPU_CYCLE      10.0 // 100Mhz
2  `define MAX             3000000 // 3000000
```

CYCLE_MAX.txt

rtl_sim.f

- Include every file inside rtl_sim.f
- Delete every `include in your code

```
../src/EXE_MEM_reg.sv
../src/forwarding_unit.sv
../src/hazard_detection_unit.sv
../src/ID_EXE_reg.sv
../src/IF_ID_reg.sv
../src/imm_ext.sv
../src/jb_unit.sv
../src/LlC_data.sv
../src/LlC_inst.sv
../src/ld_filter.sv
../src/MEM_WB_reg.sv
../src/mux2.sv
../src/mux3.sv
../src/mux4.sv
../src/pc.sv
../src/regfile.sv
../src/ROM_wrapper.sv
../src/sensor_ctrl.sv
../src/sensor_ctrl_wrapper.sv
../src/SRAM_wrapper.sv
../src/sync.sv
../src/tag_array_wrapper.sv
../src/top.sv
../src/wdata_shift.sv
../src/WDT.sv
../src/WDT_wrapper.sv
../src/AXI/AXI_interconnect.sv
../src/AXI/AXI.sv
../src/AXI/FIFO_1.sv
/usr/cad/CBDK/Executable_Package/Collaterals/IP/stdio/N16ADFP_StdIO/VERILOG/N16ADFP_StdIO.v
```

Simulation command

Simulation (1/2)

Table B-1: Simulation commands (Partial)

Simulation Level	Command
Problem1	
RTL	make rtl_all
Post-synthesis (optional)	make syn_all
Post-layout Gate-level	make pr_all

Table B-2: Makefile macros (Partial)

Situation	Command	Example
RTL simulation for progX	make rtlX	make rtl0
Post-synthesis simulation for progX	make synX	make syn1
Post-layout simulation for progX	make prX	make pr2
Dump waveform (no array)	make {rtlX,synX,prX} FSDB=1	make rtl2 FSDB=1
Dump waveform (with array)	make {rtlX,synX,prX} FSDB=2	make syn3 FSDB=2
Open nWave without file pollution	make nWave	
Open Superlint without file pollution	make superlint	
Open Spyglass without file pollution	make spyglass	
Open DesignVision without file pollution	make dv	
Synthesize your RTL code (You need write <i>synthesis.tcl</i> in <i>script</i> folder by yourself)	make synthesize	
Delete built files for simulation, synthesis or verification	make clean	
Check correctness of your file structure	make check	
Compress your homework to <i>tar</i> format	make tar	

Simulation (2/2)

Table B-3: Simulation commands

Situation	Command
Open ICC2 gui	<code>icc2_shell -gui</code>

Submission rule

Credit

Report

- 22% design report & lesson learn
- 10% APR & explore(ex: LVS or DRC pass)

Simulation

- 48%

	RTL	SYN	APR
Prog0	3%	4%	5%
Prog1	3%	4%	5%
Prog2	3%	4%	5%
Prog3	3%	4%	5%

Performance & Area

- 20%

Report Requirements

- Report and show screenshots of your prog0 to prog3 simulation time **after APR** and total cell area of your design.
- **20%** homework credit will be given based on your design performance & area
 - The result of PA will be ranked into 5 groups, each group has its credit point.
 - **Only those who pass all SYN simulation will have the chance to get PA credit**

Rank	Credit	
Tier1	20	Top 20% of PA result
Tier2	18	Top 40% of PA result
Tier3	16	Top 60% of PA result
Tier4	14	Top 80% of PA result
Tier5	12	
Tier6	8	SYN Pass, Fail in APR
Tier7	0	SYN Fail

$$\text{Performace(P)} = \sum_{i=0}^{\boxed{2}} \text{simulation time of prog } i$$

$$\text{Area(A)} = \text{total cell area of your design}$$

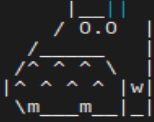
Report Requirements

- ▶ **USE the provided .DOCX document from the TA. (otherwise get 0 credit of this homework)**
- ▶ Proper explanation of your design is required for full credits.
- ▶ **Block diagrams** shall be drawn to depict your designs.
- ▶ Show your **screenshots of the waveforms** and the **simulation results** on the terminal for the different test cases in your report and illustrate the correctness of your results.
- ▶ Show your screenshots of the Spyglass CDC reports and explain why your CDC circuit can work correctly.
- ▶ Show your IM/ DM cache hit rate, and explain
- ▶ Report the number of lines of your RTL code, the results of running Superlint and 3~5 most frequent warning/errors in your code. Describe how you modify your code to comply with Superlint.
- ▶ APR flow explanation
- ▶ Lesson learned

Report Requirements

- Two screenshot needed
- Simulation pass of each prog

```
*****
*
* Congratulations !!
*
* Simulation PASS!!
*
*****
```



```

CYCLE      FREQ
DRAM       : 5.000000 200.000000
ROM        : 50.100000 19.960080
CPU        : 5.000000 200.000000
AXI        : 2.500000 400.000000

$finish called from file "/home/user2/ms112/justin112/AVSD/2023/./sim/top_tb.sv", line 257.
$finish at simulation time 94284500
VCS Simulation Report
Time: 942845000 ps
CPU Time: 9.580 seconds; Data structure size: 16.6Mb
Wed Dec 4 15:53:56 2024
CPU time: 1.367 seconds to compile + .385 seconds to elab + .225 seconds to link + 9.625 seconds in s
superdome2: /home/user2/ms112/justin112/AVSD/2023/

```

Area.log

```

Number of ports: 12689
Number of nets: 43910
Number of cells: 30239
Number of combinational cells: 23568
Number of sequential cells: 6259
Number of macros/black boxes: 151
Number of buf/inv: 3246
Number of references: 5

Combinational area: 7804.045661
Buf/Inv area: 513.786261
Noncombinational area: 5840.398157
Macro/Black Box area: 279595.239868
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 293239.683686
Total area: undefined
1

```

Report Requirements

- Please use the Submission Cover
- Please don't paste your code in the report
 - The code of program is allowed for the explanation
- If two people are in a group, the **contribution** must be written (please put the contribution on the second page)
 - Ex: A(N26071234) 55%, B(N26075678) 45%
 - Total 100%
 - You don't need to write if you own a group

File Upload(1/2)

- 依照檔案結構壓縮成 “.tar” 格式
 - 在Homework主資料夾(N260XXXXX)使用 **make tar** 產生的tar檔即可符合要求
- 檔案結構請依照作業說明
 - 錯誤檔案結構扣**10%**
 - **請勿更改top_tb.sv, top_tb_WDT.sv 以及 Makefile**
- 請勿附上檔案結構內未要求繳交的檔案
 - 在Homework主資料夾(N260XXXXX)使用 **make clean** 即可刪除不必要的檔案
 - 多餘檔案(.v, .fsdb, ...) **一個**檔案扣**2%**
- 請務必確認繳交檔案可以在SoC實驗室的工作站下compile，且功能正常
- 無法compile將直接以0分計算
- 請勿使用generator產生code再修改
- 禁止抄襲

File Upload(2/2)

- 一組只需一個人上傳作業到Moodle
 - 兩人以上都上傳會扣分(5%)
- 壓縮檔、主資料夾名稱、Report名稱、StudentID檔案內的學號都要為上傳者的學號，其他人則在Submission Cover內寫上自己的學號。
 - Ex: A(N26101234)負責上傳，組員為B(N26105678)
 - N26101234.tar (壓縮檔)
 - N26101234 (主資料夾)
 - N26101234.docx (Report，Cover寫上兩者的學號)

Deadline

■ 2023/1/1(三) 23:59前上傳

➤ 不接受遲交，請務必注意時間

➤ Moodle只會留存你最後一次上傳的檔案，檔名只要是「*N260XXXXX.tar*」即可，不需要加上版本號

Thanks for your participation and attendance !!