

# NYCU-DCS-2025

## Final\_Project

### Design: MCU System with CNN Accelerator

#### Data Preparation

1. Extract files from TA's directory:  
**% tar xvf ~DCSTA01/Final\_Project.tar**
2. The extracted LAB directory contains:
  - a. **00\_TESTBED**
  - b. **01\_RTL**
  - c. **02\_SYN**
  - d. **03\_GATE**
  - e. **09\_SUBMIT**

#### Design Description

In modern chip systems, a microcontroller orchestrates the chip's overall behavior, while the AXI protocol serves as the bridge between DRAM and our hardware designs. In previous assignments, we implemented a CNN accelerator and learned how to interface it using DMA over AXI. In this project, you will integrate those individual circuits into a cohesive full-chip architecture, enabling you to grasp the inner workings of contemporary chip systems.

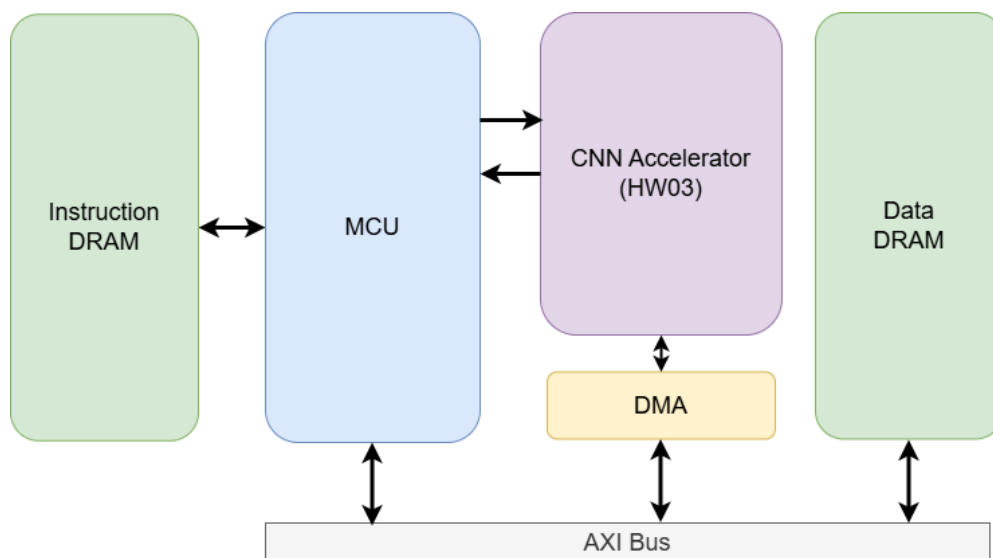


Fig. 1. Overall Microcontroller-based system architecture with CNN accelerator (MCU → Microcontroller Unit)

## Microcontroller Description

The **Central Processing Unit (CPU)** is the core component of any computer system. It is responsible for carrying out the instructions of a program by performing calculations, making decisions, and managing the flow of data.

The CPU has two main parts: the **Control Unit** and the **Datapath**.

- The **Control Unit**, often implemented as a finite state machine (FSM), fetches and decodes instructions from memory. It then generates the necessary control signals to guide operations such as reading/writing registers, performing calculations, or jumping to different instructions based on conditions.
- The **Datapath** includes elements like the Arithmetic Logic Unit (ALU), which handles mathematical computations and logical operations.

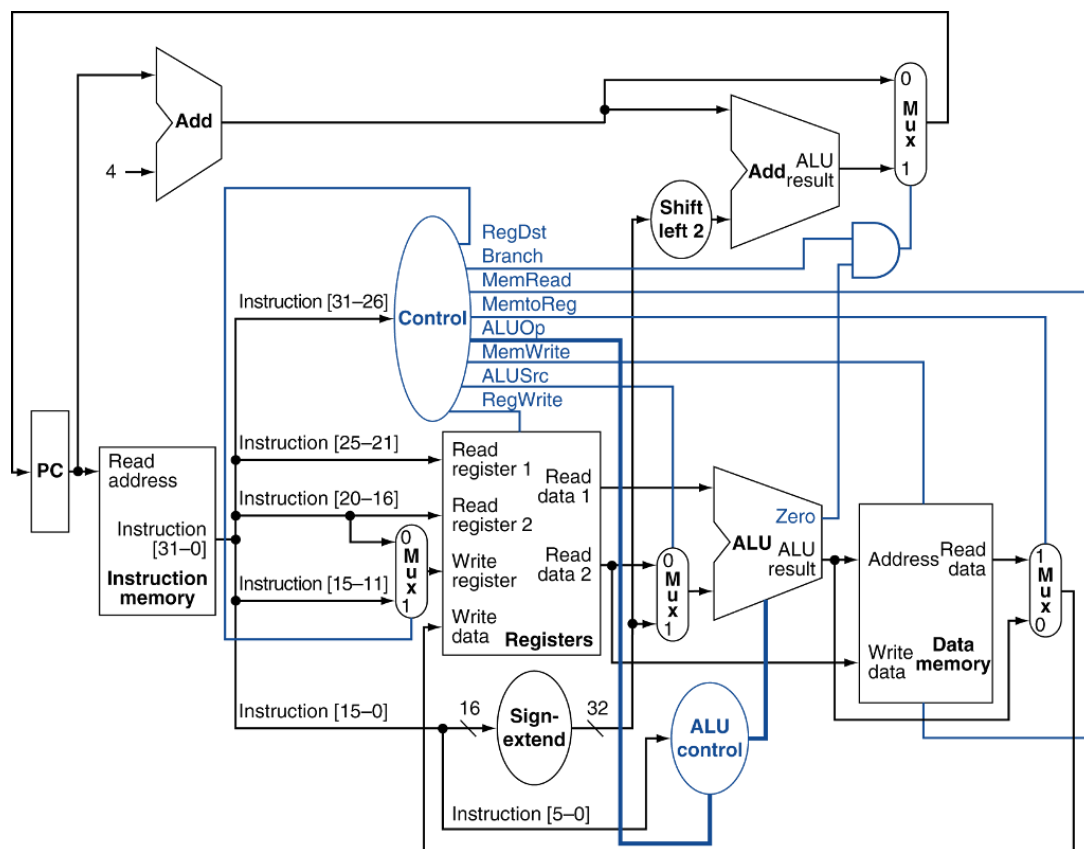


Fig. 2. MIPS CPU Architecture

A typical microcontroller combines the CPU with other essential components on a single chip, including:

- **Processor (CPU)**
  - Executes instructions (e.g., arithmetic operations, storing data, jumping to different parts)
  - Makes logical decisions based on program conditions
- **Memory**
  - **Program memory:** Stores the code or instructions
  - **Data memory:** Temporarily holds variables and data during execution
- **Input/Output (I/O) Ports**
  - Allow the microcontroller to interact with external devices like buttons, sensors, displays, LEDs, and motors
- **Timers, Counters, and Peripherals**
  - Support features such as time tracking and communication with other devices using protocols like UART, SPI, and I<sup>2</sup>C

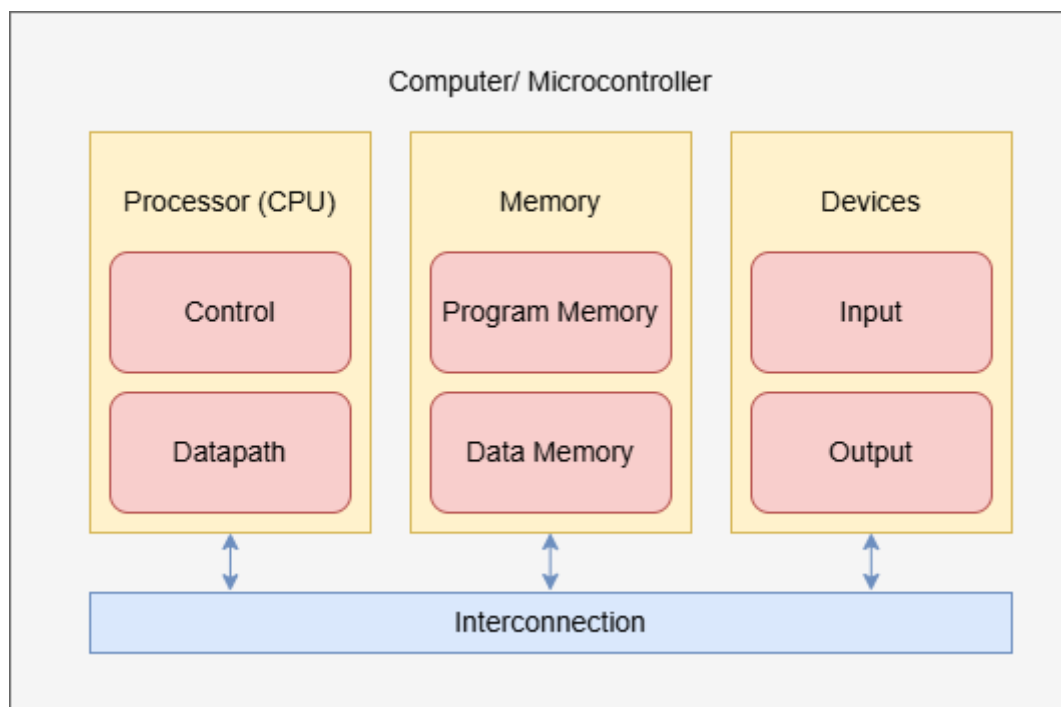


Fig.3 Microcontroller Architecture

## Functional Description

The following are the four formats for the core instruction set

Type	16-bit						
	MSB					LSB	
R	op(3-bits)	rs(4-bits)		rt(4-bits)	rd(4-bits)	func(1-bit)	
I	op(3-bits)	rs(4-bits)		rt(4-bits)	Imm(5-bits)		
J	op(3-bits)	Address(13-bits)					
E	op(3)	ImgA(3)	ImgB(3)	rd(4)	K(1)	W(1)	Mode(1)

Register s(rs), Register t(rt) and Register d(rd) represent the address of registers.

Since the instruction takes 4 bits to store the address, it means we have 16 registers, from r0 to r15. Each register reserve 8 bits to store data, e.g. rs = 4'b0000 means one of operands is "r0", rt = 4'b0101 means one of operands is "r5", and so on.

Detailed instructions are shown below

Function name	Meaning		Type	Inst. Encode
Load	reg[rt] = DRAM[signed(reg[rs] * imm + imm)]		I	010-rs-rt-imm
Store	DRAM[signed(reg[rs] * imm + imm)] = reg[rt]		I	011-rs-rt-imm
Add	reg [rd] = reg [rs] + reg[rt]		R	000-rs-rt-rd-0
Sub	reg [rd] = reg [rs] - reg[rt]		R	000-rs-rt-rd-1
Mult	reg [rd] = reg [rs] * reg[rt]		R	001-rs-rt-rd-1
Beq	if(reg[rs] == reg[rt])=> pc = pc + 1 + imm; else pc = pc + 1;		I	100-rs-rt-imm
jump	PC = address		J	101-address
CNN	max_hw3_out_idx (0~3)	E	111-img(a)-img(b)-rd-kernal-weight-mode	

**\*CNN instruction description:** In hw3, we design the CNN.v, the CNN module will outputs four 20-bit values: out\_data[0], out\_data[1], out\_data[2], and out\_data[3].

When a CNN instruction is executed, the MCU compares all four outputs and writes the **index (0 to 3) of the maximum value** into the destination register rd. In the case of multiple identical maximum values, the **lowest index** among them is selected.

For example:

out\_data = [5, 4, 1, 5], both index 0 and 3 have the maximum value of 5  
so rd will be assigned 0.

In this project, the image and kernel/weight selection in the CNN accelerator, as well as the computation method, are exactly the same as in HW03.

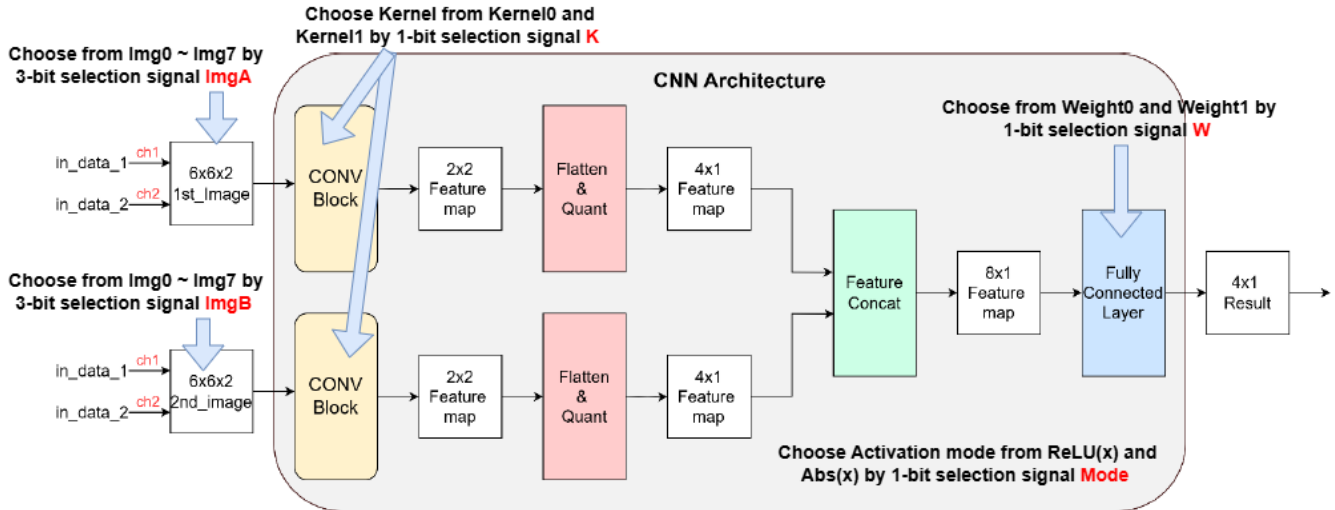


Fig. 4 Selection of input data for CNN

**\*PC: program counter**, meaning the next instruction. For R type, store and load instructions, pc should automatically +1 to get the next instruction

**Design (SDMA) and DRAM Communication (Please refer to AXI\_introduction)**

**SDMA : Master ; DRAM : Slave**

Write address handshake

- (awvalid\_m\_inf, awready\_m\_inf) pair
- For awaddr\_m\_inf [31:0] transaction

Write data handshake

- (wvalid\_m\_inf, wready\_m\_inf) pair
- For wdata\_m\_inf [127:0] transaction

Write response handshake

- (bvalid\_m\_inf, bready\_m\_inf) pair
- For bresp\_m\_inf [1:0] transaction

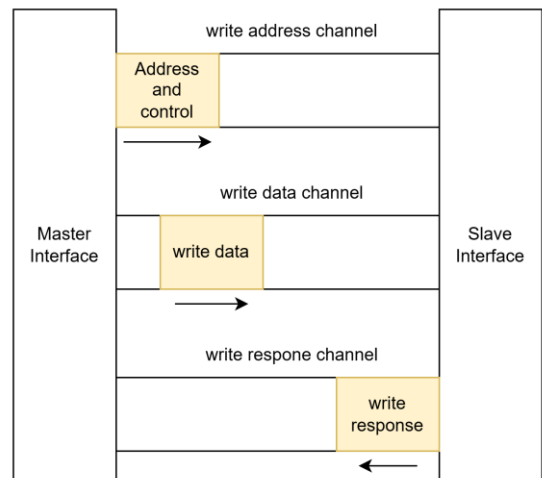


Fig. 5 AXI Write Channels

Read address handshake

- (arvalid\_m\_inf, arready\_m\_inf) pair
- For araddr\_m\_inf [31:0] transaction

Read data handshake

- (rvalid\_m\_inf, rready\_m\_inf) pair
- For rdata\_m\_inf [127:0] transaction

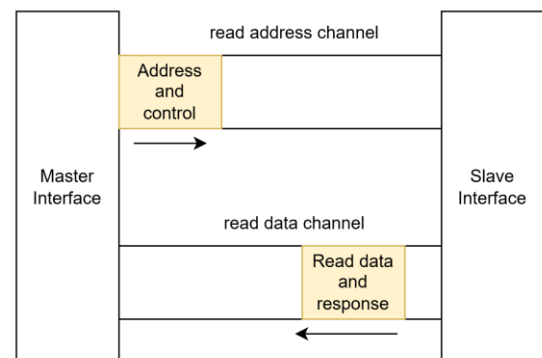


Fig. 6 AXI Read Channels

## Input && output

I/O	Input Signal	Bit Width	Description
Input	Clk	1	Positive edge trigger clock
Input	rst_n	1	Asynchronous active-low
Output	IO_stall	1	Pull high when MC is busy. It should be low for one cycle whenever you finished an instruction. TA will check the values of Data DRAM in each 10 instructions when your IO_stall is low.

## AXI 4 Interface For Data Dram

**TABLE I: AXI Signal for Write Address Channel**

Signal	Width	From	Description
awaddr_m_inf_data	32	TOP.v	Write address. Address start from 32'h1000
awvalid_m_inf_data	1	TOP.v	Write address valid. 1 = address and control information available; 0 = address and control information not available.
awlen_m_inf_data	8	TOP.v	Burst length. The burst length gives the exact number of transfers in a burst.
awready_m_inf_data	1	pseudo_data_DRAM.v	Write address ready. 1 = slave ready; 0 = slave not ready.

**TABLE II: AXI Signal for Write Data Channel**

Signal	Width	From	Description
wdata_m_inf_data	8	TOP.v	Write data.
wlast_m_inf_data	1	TOP.v	Write last. This signal indicates the last transfer in a write burst.
wvalid_m_inf_data	1	TOP.v	Write valid. 1 = write data and strobes available; 0 = write data and strobes not available.
wready_m_inf_data	1	pseudo_data_DRAM.v	Write ready. 1 = slave ready; 0 = slave not ready.

**TABLE III: AXI Signal for Write Response Channel**

<b>Signal</b>	<b>Width</b>	<b>From</b>	<b>Description</b>
bresp_m_inf_data	2	pseudo_data_DRAM.v	Write response. In this project we only issue OKAY(2'b00)
bvalid_m_inf_data	1	pseudo_data_DRAM.v	Write response valid. 1 = write response available; 0 = write response not available.
bready_m_inf_data	1	TOP.v	Response ready. 1 = master ready; 0 = master not ready.

**TABLE IV: AXI Signal for READ Address Channel**

<b>Signal</b>	<b>Width</b>	<b>From</b>	<b>Description</b>
araddr_m_inf_data	32	TOP.v	Read address. Address start from 32'h1000
arvalid_m_inf_data	1	TOP.v	Read address valid. 1 = address and control information available; 0 = address and control information not available.
arlen_m_inf_data	8	TOP.v	Burst length. The burst length gives the exact number of transfers in a burst.
arready_m_inf_data	1	pseudo_data_DRAM.v	Read address ready. 1 = slave ready; 0 = slave not ready.

**TABLE V: AXI Signal For Read Data Channel**

<b>Signal</b>	<b>Width</b>	<b>From</b>	<b>Description</b>
rdata_m_inf_data	8	TOP.v	Read data.
rlast_m_inf_data	1	TOP.v	Read last. This signal indicates the last transfer in a read burst.
rvalid_m_inf_data	1	TOP.v	Read valid. 1 = read data available; 0 = read data not available.
rready_m_inf_data	1	pseudo_data_DRAM.v	Read ready. 1 = master ready; 0 = master not ready.

**TABLE I: AXI Signal for READ Address Channel**

Signal	Width	From	Description
araddr_m_inf_inst	32	TOP.v	Read address. Address start from 32'h0000
arvalid_m_inf_inst	1	TOP.v	Read address valid. 1 = address and control information available; 0 = address and control information not available.
arlen_m_inf_inst	8	TOP.v	Burst length. The burst length gives the exact number of transfers in a burst.
arready_m_inf_inst	1	pseudo_data_DRAM.v	Read address ready. 1 = slave ready; 0 = slave not ready.

**TABLE II: AXI Signal For Read Data Channel**

Signal	Width	From	Description
rdata_m_inf_inst	16	TOP.v	Read data.
rlast_m_inf_inst	1	TOP.v	Read last. This signal indicates the last transfer in a read burst.
rvalid_m_inf_inst	1	TOP.v	Read valid. 1 = read data available; 0 = read data not available.
rready_m_inf_inst	1	pseudo_data_DRAM.v	Read ready. 1 = master ready; 0 = master not ready.



### ◆ Data DRAM Storage and Access

Each entry in the Data DRAM stores a 8-bit word, and the memory can hold up to 6144 words in total. However, for this assignment, the PATTERN only accesses a limited address range, **from 0x1000 to 0x17FF**, which corresponds to 2048 entries.

For images and kernels, the data is stored in raster scan order.

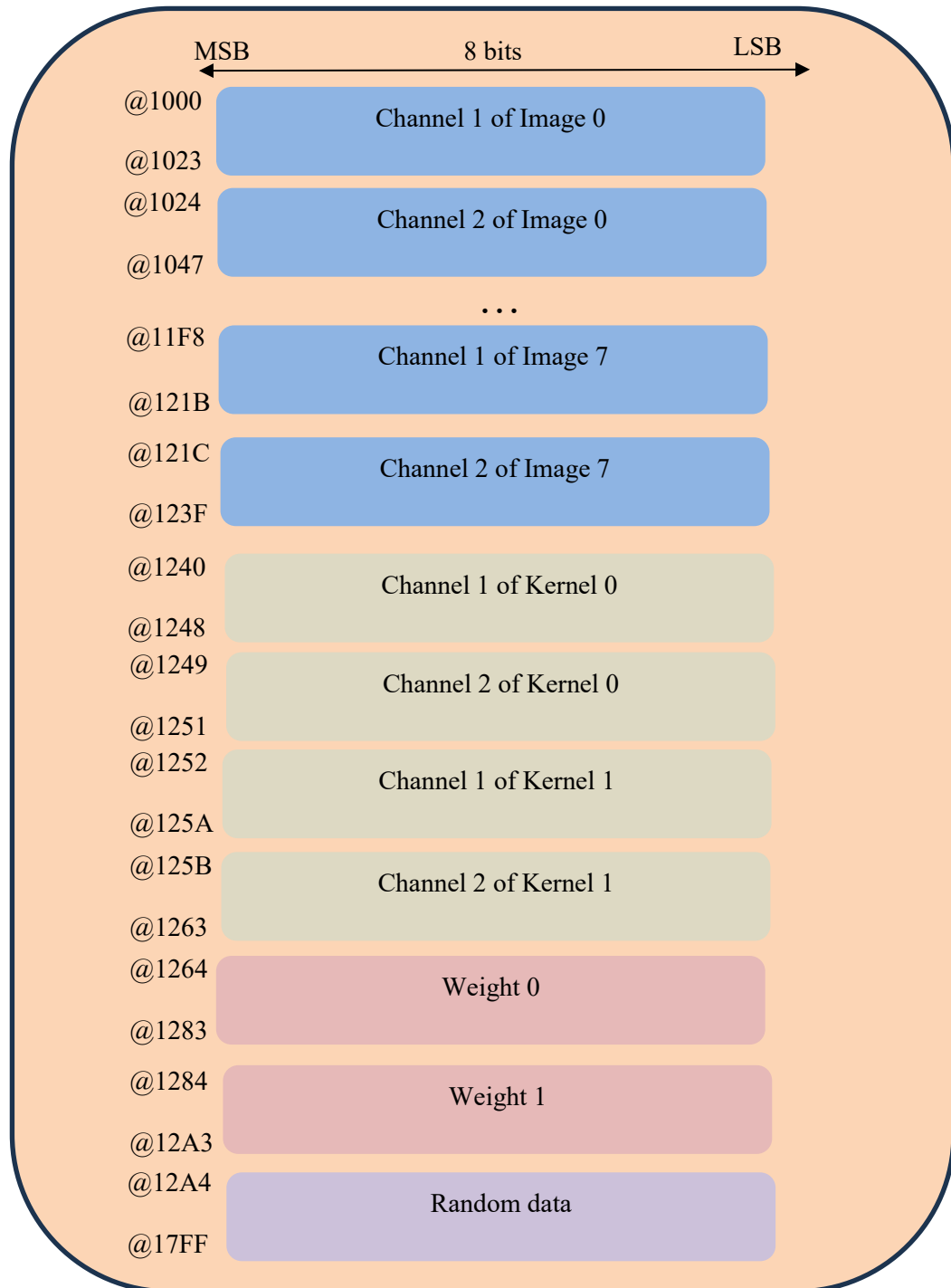


Fig. 7 Example of data in memory

## SPEC

---

### Top module

1. Top module name: Top (File name: Top.v)
2. The total execution latency is limited in 2,000,000 cycles for all instructions.

### 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 (except for IO\_stall) must be **zero/low** with the asynchronous reset when rst\_n is low.
6. IO\_stall must be high after initial reset.
7. All 16 registers in the Register File must be reset to zero, or your execution result may be incorrect.

### Input Signals

8. All input signals are synchronized at **positive edge of the clock**.
9. All the data are in signed format in all registers.

### 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.

### DRAM

12. TA's pattern checks the correctness of **all data** stored in the Data DRAM in each 10 instructions when your IO\_stall is low.

### Synthesis

13. You are **not** allowed to modify syn.tcl, except for cycle time (CYCLE).
14. 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.
15. The area is limited in **6,000,000**.
16. The synthesis time should be less than **2 hours**.
17. 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/).
18. After synthesis, you can check SDMA.timing. The timing report should be **non-negative (MET)**.

### Gate-level simulation

19. The gate-level simulation **cannot** include any **timing violations**.

20. The cycle time used in Gate-Level Simulation must be the **same** as the cycle time used in synthesis.

### Supplement

21. In this assignment, you are **NOT** allowed to use DesignWare IP.
22. 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.
23. Don't write Chinese or other language comments in the file you sent.
24. Any error messages during synthesis and simulation, regardless of the result will lead to failure in this lab.
25. 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.
26. After a conv (CNN) instruction, there are over 3000 instructions that do not depend on the CNN output register (Rd). This means that **Out-of-Order (OOO) execution** is possible and could improve performance if managed properly. However, the PATTERN assumes instructions execute In-Order, and the current DRAM check mechanism relies on this assumption. So, OOO execution can still be used, but you must carefully control when and how it happens to avoid breaking data correctness or timing expectations.
27. There are only 3,710 instructions in total; as long as the execution process is identical, the DRAM results will be the same.

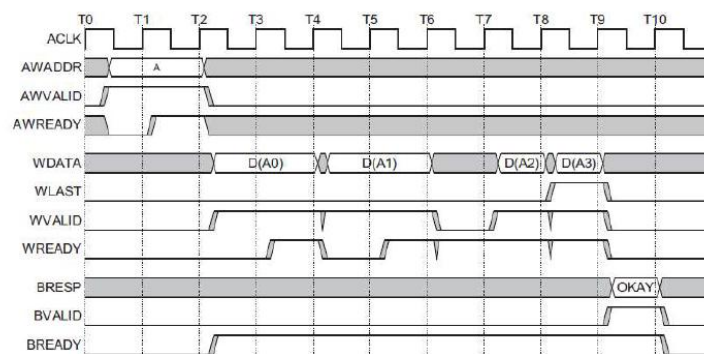


Figure 1-6 Write burst

Fig. 8 The example of using Burst length

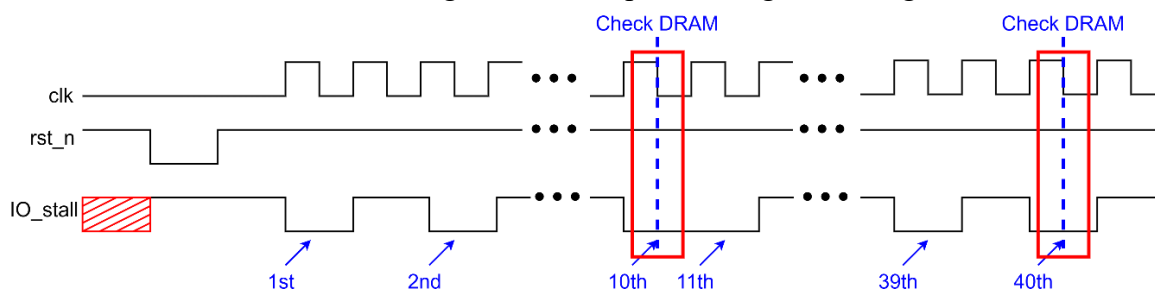


Fig. 9 Check DRAM data every 10 cycles

## Block Diagram

---

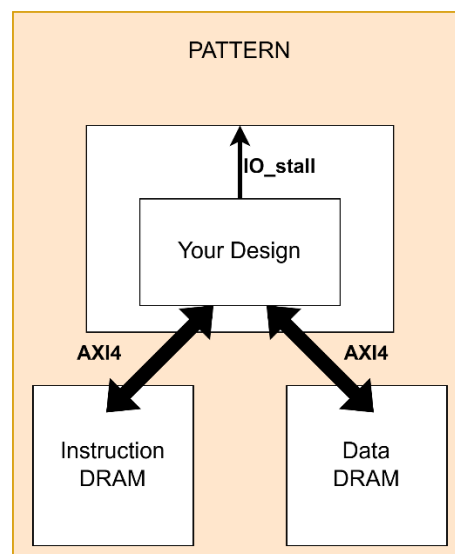


Fig. 10 System Block Diagram

## Homework Upload

---

1. Please use the commands we provided to tar whole file under 09\_SUBMIT/ and to submit your homework
  - **1st\_demo deadline : 2025/6/20 (Fri.) 23:59:59**
  - **2nd\_demo deadline : 2025/6/25 (Wed.) 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
  - ◆ Provided Test Data (60%)
  - ◆ Hidden Test Data (40%)
2. Report: 30%
  - ◆ Block diagram, FSM, AXI control, .....
  - ◆ If Out-Of-Order execution is applied to well exploit the latency of CNN, you will get higher score in this section.

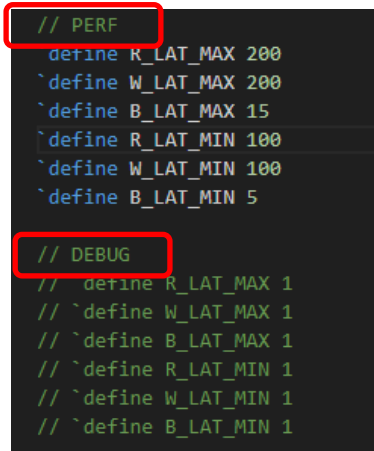
## Command List

---

- ❖ 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.

## 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
```

Fig. 11 DRAM latency settings

**Please note that in this project the test patterns are provided in plaintext. Whenever an instruction’s behavior is unclear, consulting the pattern directly will be more efficient. Additionally, during debugging you may invoke the pattern’s waveform for further analysis.**