NYCU-DCS-2025

HW04

Design: Simplified Direct Memory Access (SDMA)

Data Preparation

1. Extract files from TA's directory:
   % tar xvf ~DCSTA01/HW04.tar
2. The extracted LAB directory contains:
   a. **00_TESTBED**
   b. **01_RTL**
   c. **02_SYN**
   d. **03_GATE**
   e. **09_SUBMIT**

Design Description

Direct Memory Access (DMA) is a hardware feature that allows peripherals to transfer data directly to or from memory without involving the CPU. This improves system efficiency by freeing up the CPU to perform other tasks while data is being moved in the background.

In this assignment, you are required to design a simple **DMA** that communicates with a **Memory (DRAM)** using a simplified **AXI protocol**. The data bus width of the AXI interface is fixed, and the PATTERN accesses data in 8-bit (byte) units. However, the DRAM is organized to be accessed in 32-bit (word) units. Therefore, your DMA must handle **address translation** and **data extraction** to ensure that byte-sized requests from PATTERN are correctly mapped to the word-based structure of the DRAM. Detailed functional requirements will be discussed in the following sections.

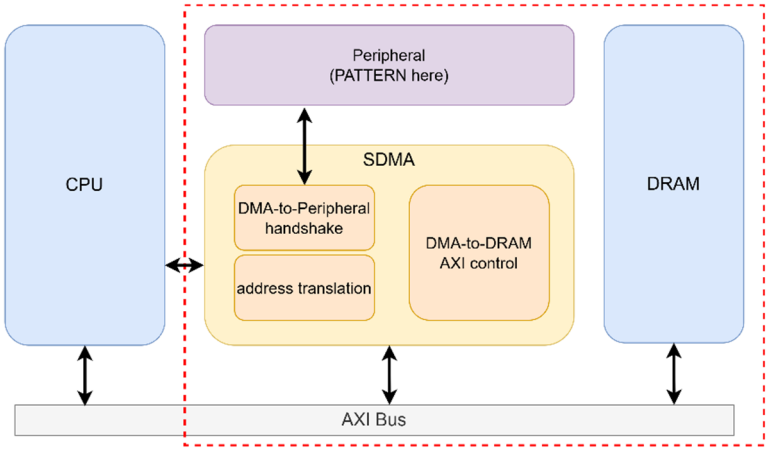


Fig. 1. Direct Memory Access (DMA) in Embedded System

Functional Description

PATTERN and Design (SDMA) Communication

PATTERN-to-SDMA handshake

- (pat_valid, sdma_ready) pair
- For cmd[31:0] transaction

SDMA-to-PATTERN handshake

- (pat_ready, sdma_valid) pair
- For dout[7:0] transaction

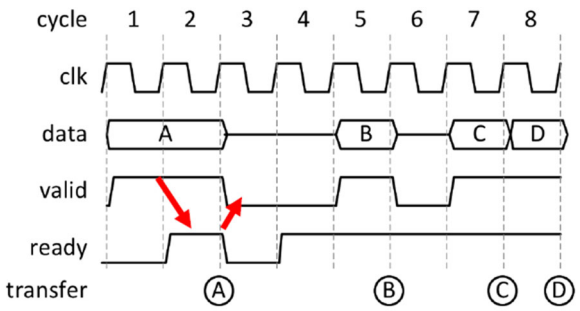


Figure 2. Handshake Process

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

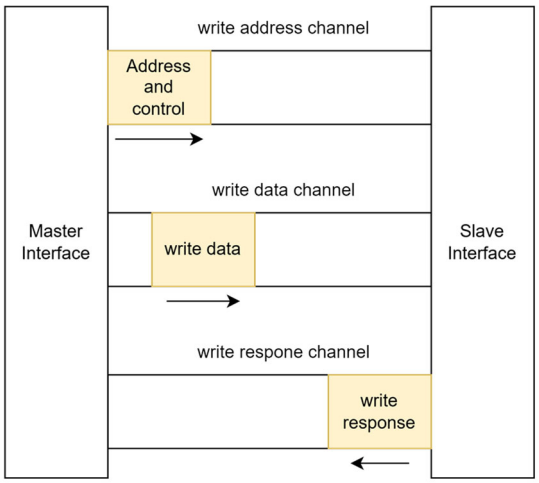


Figure 3. AXI Write Channels

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

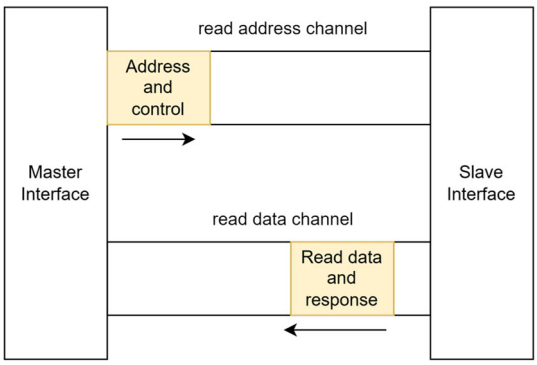


Figure 4. AXI Read Channels

◆ **DRAM Storage and Access**

Each entry in the DRAM stores a 32-bit word, and the memory can hold up to 2^{32} words in total. However, for this assignment, the PATTERN only accesses a limited address range, **from 0x00000800 to 0x00000BFF**, which corresponds to 1024 entries.

Example of data in memory:

Figure 5. DRAM Status

```
// In pseaudo_DRAM
reg [31:0] DRAM [2048 : 3071]; // Address from 0x0800 to 0x0BFF
```

```
@800
af7f06fa 044caa7d 0b8087eb 89ebc9e3
@804
02ebd2aa fd0cec3e 6cdb99ed 50a19903
@808
7714fdb4 5283300b 06ad92b0 1213b6b4
@80c
5f81b55a 981fd219 04a5f091 55a777d1
@810
```

DRAM				
Address	[31:24]	[23:16]	[15:8]	[7:0]
[2048]	af	7f	06	fa
[2049]	04	4c	aa	7d
[2050]	0b	80	87	eb
[2051]	89	eb	c9	e3
[2052]	02	eb	d2	aa
[2053]	fd	0c	ec	3e

Figure 6. Example data Storage

● **DRAM Data Transaction**

Since the data bus width of AXI interface is fixed, each data transaction is always 128-bit (4-word) long. Therefore, even though the required read_data is only 8-bit long, the SDMA receives a 128-bit rdata_m_inf[127:0] after a read data handshake.

Read data example:

Access address : 0x0808

Receive rdata_m_inf [127:0] : 0x1213 b6b4 06ab 92b0 5283 300b 7714 fdb4

Similarly, although only an 8-bit write_data is required for the DRAM write operation, the SDMA must first retrieve the full 128-bit data block from DRAM. It then replaces the targeted 8 bits with write_data, and finally sends the updated 128-bit value through the wdata_m_inf[127:0] write data bus.

Write data example:

command: Write **0xbe** into the location where currently 0x14 is stored in DRAM[0x0808].

Access address : 0x0808

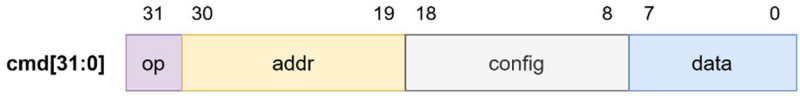
Receive rdata_m_inf [127:0] : 0x1213 b6b4 06ab 92b0 5283 300b 7714 fdb4

Send wdata_m_inf [127:0] : 0x1213 b6b4 06ab 92b0 5283 300b 77**be** fdb4

◆ Command Format

In this assignment, SDMA receives a 32-bit command (cmd[31:0]) containing several fields.

	Field	Description
cmd[31]	op	Indicates whether a **read** or **write** operation should be performed. op = 0 : read op = 1 : write
cmd[30:19]	addr	Specifies the **address** to be accessed. Range from 0 to 4095
cmd[18:8]	config	Defines the **communication configuration** between SDMA and DRAM. (Not used)
cmd[7:0]	data	Carries the **data** to be written into DRAM.



Note that the address extracted from cmd[31:0] ranges from **0 to 4095.** This is due to a misalignment between how the PATTERN accesses data (by byte) and how DRAM stores data (by word). Specifically, each word in DRAM is 4 bytes, but the address in the command is byte-based. This mismatch results in the observed address range. The mapping method is illustrated below.

DRAM address	data	cmd.addr	data
0x0800	0xaf7f06fa	0	0xfa
		1	0x06
		2	0x7f
		3	0xaf
0x0801	0x044caa7d	4	0x7d
		5	0xaa
		6	0x4c
		7	0x04
0x0BFF	0x8e30b926	4092	0x26
		4093	0xb9
		4094	0x30
		4095	0x8e

Input

Input	Bit	Description
clk	1	Clock signal, for positive edge triggered design
rst_n	1	Asynchronous active-low reset
pat_valid	1	Handshake signal High when cmd[31:0] is valid
pat_ready	1	Handshake signal High when PATTERN is ready to receive dout[7:0]
cmd	32	Command to issue a transaction
awready_m_inf	1	AWREADY
wready_m_inf	1	WREADY
bvalid_m_inf	1	BVALID
bresp_m_inf	2	BRESP
arready_m_inf	1	ARREADY
rvalid_m_inf	1	RVALID
rdata_m_inf	128	RDATA

1. cmd is tied to unknown while pat_valid not asserted.
2. pat_ready **wait** for the assertion of sdma_valid.
3. Please refer to "AXI_introduction" for detailed description of input AXI signals.

Output

Output	Bit	Description
sdma_ready	1	Handshake signal High when SDMA is ready to receive cmd[31:0]
sdma_valid	1	Handshake signal. High when dout[7:0] is valid **or a write operation is completed.**
dout	8	Data read from DRAM, don't care if it's a write cmd.
awvalid_m_inf	1	AWVALID
awaddr_m_inf	32	AWADDR
wvalid_m_inf	1	WVALID
wdata_m_inf	128	WDATA
bready_m_inf	1	BREADY
arvalid_m_inf	1	ARVALID
araddr_m_inf	32	ARADDR
rready_m_inf	1	RREADY

1. All output signals (except for sdma_ready) must be **zero/low** with the **asynchronous reset** when rst_n is **low**.
2. **sdma_ready** must be **high** with the **asynchronous reset** when rst_n is **low**.
3. Please refer to "AXI_introduction" for detailed description of output AXI signals.

SPEC

Top module

1. Top module name: SDMA (File name: SDMA.v)
2. The execution latency is limited in **1000 cycles** for single pattern. The latency is the clock cycles between the (pat_valid, sdma_ready) handshake and the (pat_ready, sdma_valid) handshake.

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 sdma_ready) must be **zero/low** with the asynchronous reset when rst_n is low.
6. **sdma_ready** must be **high** with the asynchronous reset when rst_n is low.

Input Signals

7. All input signals are synchronized at **positive edge of the clock**.
8. pat_valid will come after reset.
9. pat_valid remains asserted until a successful handshake with sdma_ready.
10. pat_valid will not be asserted before the previous command is done (except for the first one).
11. cmd remains valid while pat_valid asserted.
12. pat_ready **waits** for the assertion of sdma_valid.

Output Signals

13. All output signals should be synchronized at **positive edge of clock**.
14. The TA's pattern will capture your output for checking at clock negative edge.
15. Once sdma_valid asserted, it should **remain high** until a successful handshake with pat_ready.
16. After a successful handshake with pat_ready, **sdma_valid** must be **pulled down** at the next clock cycle.
17. sdma_valid cannot overlap with pat_valid at any time.

DRAM

18. TA's pattern checks the correctness of **all data** stored in the DRAM after every command is done (at the clock cycle when successful (pat_ready, sdma_valid) handshake occurs).
19. 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

20. You are **not** allowed to modify syn.tcl, except for cycle time (CYCLE).
21. You **can** adjust your clock period, but the maximum period is **10 ns**. The precision of clock period is 0.1, for example, 4.5 is allowed, 4.55 is not allowed.
22. The area is limited in **150,000**.
23. The synthesis time should be less than **2 hours**.
24. 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/).
25. After synthesis, you can check SDMA.timing. The timing report should be **non- negative (MET)**.

Gate-level simulation

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

Supplement

28. In this assignment, you are **NOT** allowed to use DesignWare IP.
29. 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.
30. Don't write Chinese or other language comments in the file you sent.
31. Any error messages during synthesis and simulation, regardless of the result will lead to failure in this lab.
32. 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.
33. During byte-sized **incremental** memory access, redundant read operations may be avoided by optimizing for **consecutive or overlapping accesses** (account for 42% of the test patterns).

Block Diagram

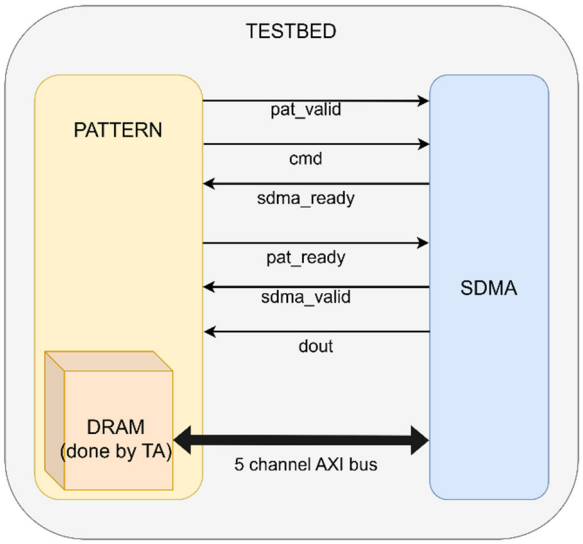


Figure 7. System Block Diagram

Homework Upload

1. Please use the commands we provided to tar whole file under 09_SUBMIT/ and to submit your homework
   - **1st_demo deadline : 2025/5/15 (Thu.) 12:00:00**
   - **2nd_demo deadline : 2025/5/22 (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^2 * Total_Latency * Cycle_Time**

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

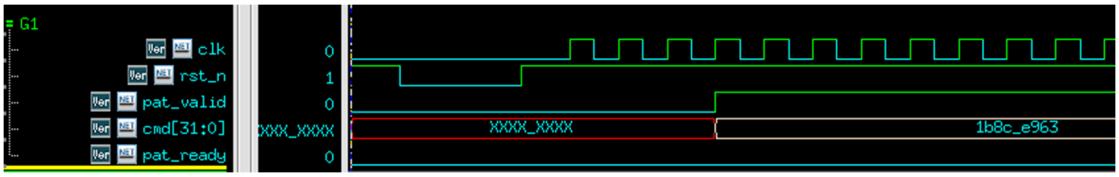


Figure 8. The example of input waveform

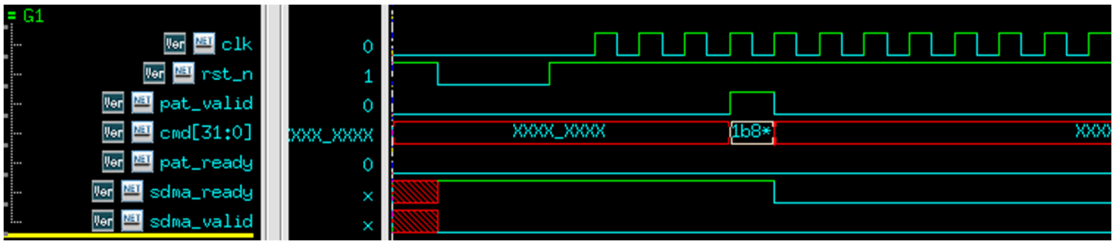


Figure 9. The example of input handshake

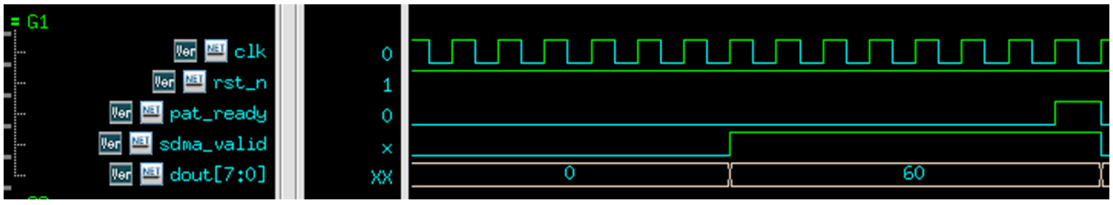


Figure 10. The example of output handshake

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.

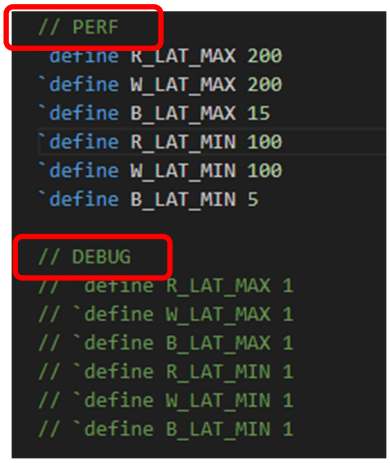


Figure 11. DRAM latency settings