# NYCU-EE IC LAB – Fall 2023

Lab03 Exercise

## Design: AXI-SPI DataBridge

### Data Preparation

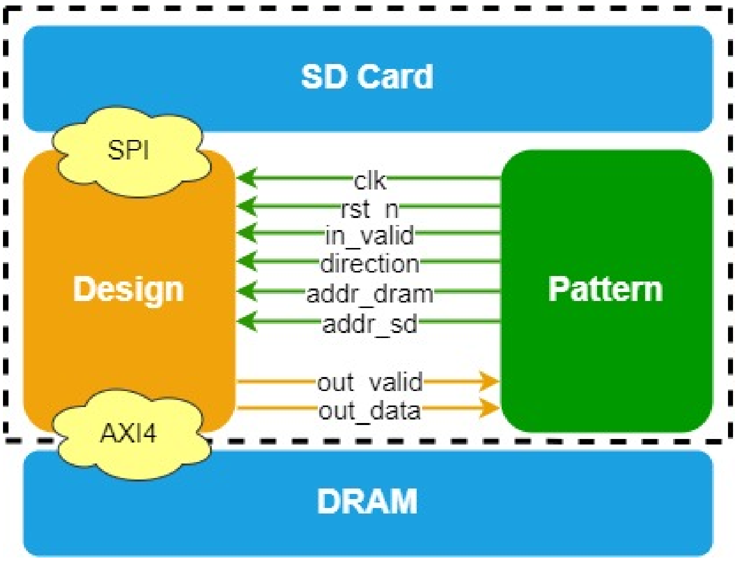
1. Extract files from TA's directory:

   **% openssl des3 -d -k MApocYIQNgU= -salt -in ~iclabTA01/Lab03.tar | tar xvf -**

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

### Introduction

In today's digital era, the seamless transfer of data between various storage mediums is pivotal to the efficiency and reliability of electronic systems. The task at hand pertains to the design of a mechanism that enables data transfer between Dynamic Random Access Memory (DRAM) using the AXI4-Lite protocol and an SD card using the Serial Peripheral Interface (SPI) protocol. The system architecture is shown below.




In this lab, your task is to finalize the **PATTERN** and develop a **pseudo SD card** by referencing the provided design specifications and the SD card protocol. We will supply both encrypted versions of correct and incorrect designs to guide and assist you in this process. Additionally, we offer a pseudo DRAM that not only simulates the behavior of actual DRAM but also verifies its specifications. Once the PATTERN and pseudo SD card are complete, your next step is to design a **bridge** connecting the

DRAM and the SD card.

## Problem Description

Read Pattern

In this lab, you need to read the patterns from the file Input.txt. The format of Input.txt is outlined below:

**2** (2 patterns in this file)

**0 11 22** (Reading data from the DRAM starting at address 11 and writing it to the SD card located at address 22)

**1 33 44** (Reading data from the SD card starting at address 44 and writing it to the DRAM located at address 33)

The first number at the beginning of the file is the number of patterns, and each subsequent line corresponds to a pattern. For each pattern:

1. The first number signifies the **direction** of the data transfer.
   (0) DRAM → SD card, out_data
   (1) SD card → DRAM, out_data
2. The second number indicates the starting **DRAM address**. (Legal range: 0~8191)
3. The third number indicates to the **SD card address**. (Legal range: 0~65535)

Please note that you should calculate the golden answer in the PATTERN. The golden answer from external files is NOT allowed.

Read Initial Data

The initial data sets are been saved as DRAM_init.dat for DRAM and SD_init.dat for the SD card. The format can be readable by “readmemh” function from Verilog.

The legal address range for DRAM and SD card is from **0 to 8191** and for the SD card, it is from **0 to 65535**. For each address of DRAM and SD card contains **64** bits data.

Simulation and Final Data

If the design pass the simulation, capture the concluding data states of both the DRAM and SD card. Save them as DRAM_final.dat and SD_final.dat respectively. The format must be readable by “readmemh” function from Verilog. You can use “writememh” function to write the files. We will check the correctness of final state of DRAM and SD card according to the Input.txt, DRAM_init.dat and SD_init.dat.

All the above files are stored in 00_TESTBED.

## AXI4-Lite Protocol (pseudo_DRAM.v)

For simplicity, we use AXI4-Lite protocol to access DRAM. You can get familiar with the basic operation of AXI4-Lite protocol through this lab first, and you will encounter the complete AXI4 protocol in the following lab.

The key difference between AXI4 and AXI4-Lite is that while AXI4 supports burst operations, AXI4-Lite does not.

Global Signals

| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **clk** | 1 | Clock source | Global clock signal. All signals are sampled on the **Positive** edge of the global clock. |
| **rst_n** | 1 | Reset source | Global reset signal. This signal is active LOW. |

Write Address Channel

| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **AW_ADDR** | 32 | Master | Write address. |
| **AW_VALID** | 1 | Master | Write address valid. This signal indicates that valid write address is available:<br>1 = address available<br>0 = address not available.<br>The address remain stable until the address acknowledge signal, **AW_READY**, goes HIGH. |
| **AW_READY** | 1 | Slave | Write address ready. This signal indicates that the slave is ready to accept an address signals:<br>1 = slave ready<br>0 = slave not ready. |

Write Data Channel

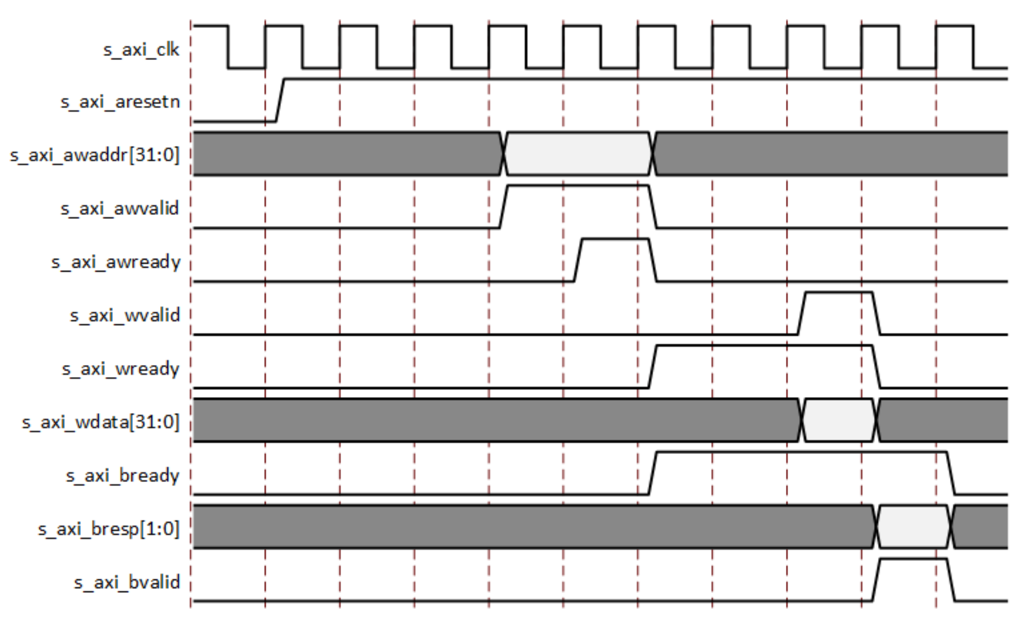
| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **W_DATA** | 64 | Master | Write data bus. |
| **W_VALID** | 1 | Master | Write valid. This signal indicates that valid write data is available:<br>1 = write data available<br>0 = write data not available. |

| | | | |
|---|---|---|---|
| **W_READY** | 1 | Slave | Write ready. This signal indicates that the slave can accept the write data:<br>1 = slave ready<br>0 = slave not ready. |

Write Response Channel

| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **B_RESP** | 2 | Slave | Write response. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (In this lab we only issue OKAY(2'b00)) |
| **B_VALID** | 1 | Slave | Write response valid. This signal indicates that a valid write response is available:<br>1 = write response available.<br>0 = write response not available. |
| **B_READY** | 1 | Master | Response ready. This signal indicates that the master can accept the response information.<br>1 = master ready.<br>0 = master not ready. |

Waveform of Write Transaction




(AMD Adaptive Computing Documentation Portal)

Spec about Write Operation

1. After **AW_VALID** is high, **AW_READY** should be pulled high in the next 1~50 cycles.
2. **AW_VALID** and **AW_ADDR** should remain stable until **AW_READY** goes high.
3. **AW_ADDR** should be within the legal range (0~8191).
4. **AW_ADDR** should be valid when **AW_VALID** is high.
5. **AW_ADDR** should be reset when **AW_VALID** is low.
6. After write address channel communicate, **W_VALID** and **W_READY** should be pulled high in the next 1~100 cycles.
7. **W_DATA** should be valid when **W_VALID** is high.
8. **W_DATA** should be reset when **W_VALID** is low.
9. **W_VALID** and **W_DATA** should be remained stable until **W_READY** goes high.
10. After write data channel communicate, **B_VALID** should be pulled high in the next 1~100 cycles.
11. **B_RESP** should be valid when **B_VALID** is high.
12. **B_RESP** should be reset when **B_VALID** is low.
13. **B_VALID** and **B_RESP** should be remained stable until **B_READY** goes high.
14. **B_READY** should be asserted before **B_VALID** goes high by at least 100 cycles. **B_READY** can be asserted even before **B_VALID** goes high.

Read Address Channel

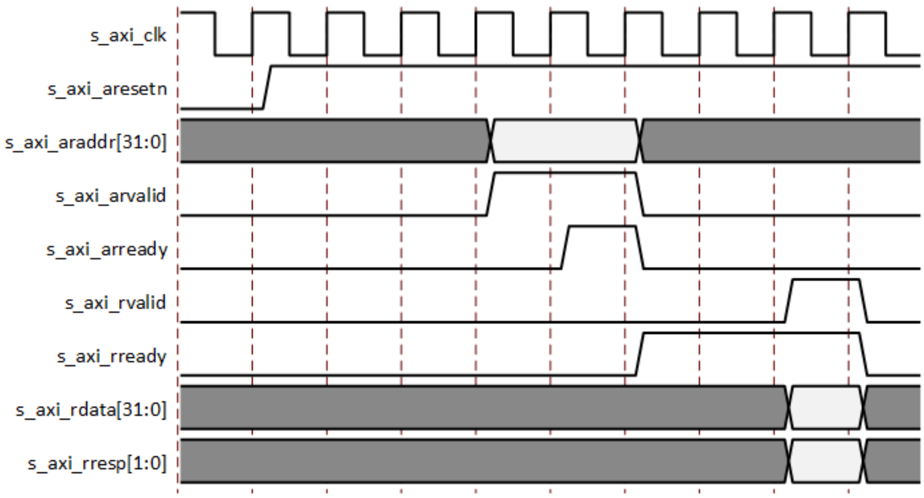
| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **AR_ADDR** | 32 | Master | Read address. |
| **AR_VALID** | 1 | Master | Read address valid. This signal indicates, when HIGH, that the read address is valid and will remain stable until the address acknowledge signal, **AR_READY**, is high.<br>1 = address and control information valid<br>0 = address and control information not valid. |
| **AR_READY** | 1 | Slave | Read address ready. This signal indicates that the slave is ready to accept an address signal:<br>1 = slave ready<br>0 = slave not ready. |

Read Data Channel

| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **R_DATA** | 64 | Slave | Read data. |

| | | | |
|---|---|---|---|
| **R_RESP** | 2 | Slave | Read response. This signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. (In this project we only issue OKAY) |
| **R_VALID** | 1 | Slave | Read valid. This signal indicates that the required read data is available, and the read transfer can complete:<br>1 = read data available<br>0 = read data not available. |
| **R_READY** | 1 | Master | Read ready. This signal indicates that the master can accept the read data and response information:<br>1= master ready<br>0 = master not ready. |

Waveform of Read Transaction




(AMD Adaptive Computing Documentation Portal)

Spec about Read Operation

1. After **AR_VALID** is high, **AR_READY** should be pulled high in the next 1~50 cycles.
2. **AR_VALID** and **AR_ADDR** should remain stable until **AR_READY** goes high.
3. **AR_ADDR** should be within the legal range (0~8191).
4. **AR_ADDR** should be valid when **AR_VALID** is high.
5. **AR_ADDR** should be reset, when **AR_VALID** is low.
6. After read address channel communicate, **R_VALID** and **R_READY** should be pulled high in the next 1~100 cycles.
7. **R_DATA** should be valid when **R_VALID** is high.
8. **R_DATA** should be reset when **R_VALID** is low.

9. **R_VALID** and **R_DATA** should be remained stable until **R_READY** goes high.

## SPI Protocol (pseudo_SD.v)

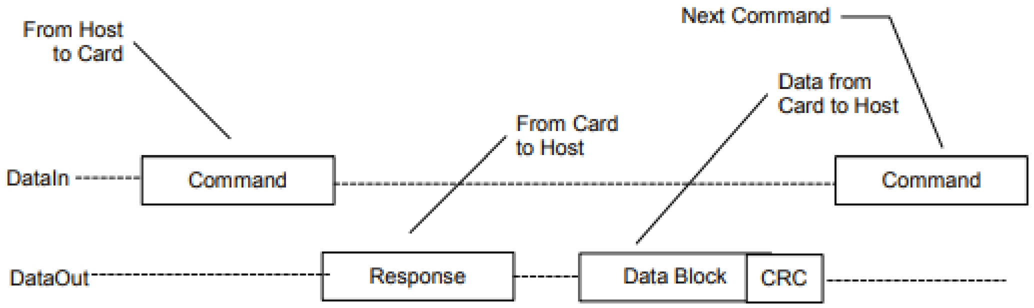
For simplicity, we remove some unimportant initial stage and some complex rule so that we can focus on the read / write operation. Please keep in mind that what is described below is not the full version of the SPI protocol to access SD card and with some modifications.

| Signal | Bit Width | Source | Description |
|---|---|---|---|
| **clk** | 1 | Clock source | Global clock signal. All signals are sampled on the **Positive** edge of the global clock. |
| CS_n | 1 | Master | Chip Select, active LOW |
| **MISO** | 1 | Slave | Master Input Slave Output. When the data is not transfer, keep HIGH. Serial In Serial Out (SISO) transmission. |
| **MOSI** | 1 | Master | Master Output Slave Input. When the data is not transfer, keep HIGH. Serial In Serial Out (SISO) transmission. |

Command Format

| Byte 1 | | | Bytes 2—5 | Byte 6 | |
|---|---|---|---|---|---|
| 7 | 6 | 5 0 | 31 0 | 7 | 0 |
| 0 | 1 | Command | Command Argument | CRC | 1 |

Read Operation




**Command** (from host)

Start bit + transmission bit = 2'b01

Command: CMD17 = 6'd17

Argument: 32 bits address

CRC: CRC-7 ({Start bit, Transmission bit, Command, Argument}) // 40 bits input

End bit: 1'b1

(wait 0~8 units, units = 8 cycles)

**Response** (from SD card)

Response: 0x00 (8 bits)

(wait 1~32 units, units = 8 cycles)

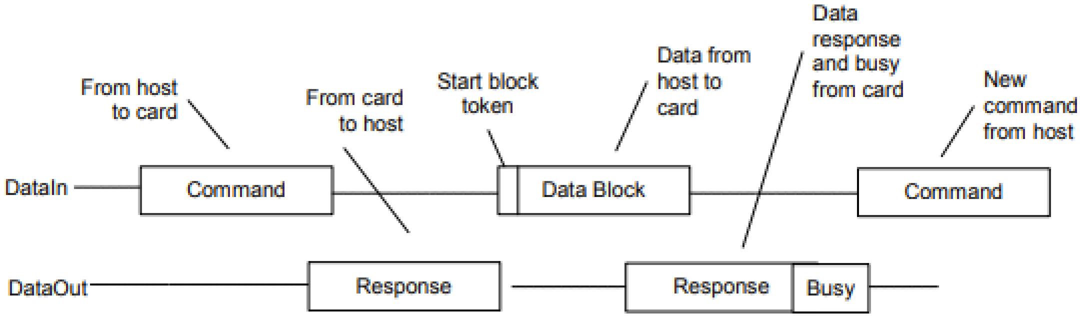
**Data** (from SD card)

Start token: 0xFE (8 bits)

Data block: 64 bits (differ from the original protocol)

CRC: CRC-16-CCITT (Data block) // 64 bits input

Write Operation




**Command** (from host)

Start bit + transmission bit = 2’b01

Command: CMD24 = 6'd24

Argument: 32 bits address

CRC: CRC-7 ({Start bit, Transmission bit, Command, Argument}) // 40 bits input

End bit: 1’b1

(wait 0~8 units, units = 8 cycles)

**Response** (from SD card)

Response: 0x00

(wait 1~32 units, units = 8 cycles)

**Data** (from host)

Start token: 0xFE

Data block: 64 bits (differ from the original protocol)

CRC: CRC-16-CCITT (Data block) // 64 bits input

(wait 0 units, units = 8 cycles)

**Data response** (from SD card)

Data_response: 8’b00000101

Busy: keep low until finish write. (wait 0~32 units, units = 8 cycles)

Spec about SD card

1. Command format should be correct, other command is not allowed.
2. The address should be within the legal range.
3. CRC (CRC-7, CRC-16-CCITT) check should be correct.
4. Time between each transmission should be correct. Please notice that the **time unit should be integer**. For example, 10 cycles = 1.25 units is not allowed.
5. All transfers are from MSB to LSB.

## Inputs / Outputs (BRIDGE.v)

■ The following are the definition of input signals

| Input Signals | Bit Width | Definition |
|---|---|---|
| **clk** | 1 | Clock |
| **rst_n** | 1 | Asynchronous active-low reset |
| **in_valid** | 1 | High when all the input signals (direction, addr_dram, addr_sd) are valid |
| **direction** | 1 | Input pattern from Input.txt, if the signal is not valid, the data should be all zero. |
| **addr_dram** | 13 | |
| **addr_sd** | 16 | |

■ The following are the definition of output signals

| Output Signals | Bit Width | Definition |
|---|---|---|
| **out_valid** | 1 | High when all read and write operation finish and out_data is valid. |
| **out_data** | 8 | Output the data that the design reads or writes, starting from the MSB. Each pattern should have 8 cycles, producing a total of 64 bits of data. |

1. All inputs will be changed at the clock negative edge.
2. All input signals are synchronized at the **negative edge** of the clock.
3. For each pattern, **in_valid** should keep high for **1** cycle and **out_valid** should keep high for **8** cycles.
4. The next input pattern will come in **2~4 negative edge of the clock** after your **out_valid** is pulled down.

## Specifications

1. Top module name: BRIDGE (design file name: BRIDGE.v)
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. For simplify the design, DRAM, SD card and design use a common clock and the clock period is **40 ns**.
4. The **reset signal (rst_n)** would be given only once at the beginning of simulation. All output signals except MOSI (SD card signal) should be reset and MOSI should be set after the reset signal is asserted. The pattern will check the output signal 100ns after the reset signal is pulled low.
5. The **out_data** should be reset when your **out_valid** is low.
6. The execution latency is limited in **10000** cycles**.** The latency is the time of the clock cycles between the **falling edge of the in_valid and the rising edge of the out_valid**.
7. The **out_valid** and **out_data** must be asserted in **8 cycles**.
8. The **out_data** should be correct when **out_valid** is high.
9. The data in the DRAM and SD card should be correct when **out_valid** is high.
10. The input delay is set to **0.5*(clock period)**.
11. The output delay is set to **0.5*(clock period)**, and the output loading is set to **0.05**.
12. The synthesis result of the data type **cannot** include any **latches**.
13. Gate-level simulation **cannot** include any **timing violations** without the notimingcheck command.
14. After synthesis, The slack at the end of the timing report should be **non-negative (MET)**.
15. Any words with "error", or "congratulation" can't be used as variable name.

## Grading Policy

1. **Test Bench**: 60% (Demo: Use your pattern.v and pseudo_SD.v, use the BRIDGE_encrypted.v, Input.txt, DRAM_init.dat and SD_init.dat we provided.)
   - (15%) Pattern correctness. (Complete the simulation when design is correct.)
   - (2.5%) SPEC MAIN-1: All output signals should be reset after the reset signal is asserted.
   - (2.5%) SPEC MAIN-2: The **out_data** should be reset when your **out_valid** is low.
   - (2.5%) SPEC MAIN-3: The execution latency is limited in 10000 cycles.
   - (2.5%) SPEC MAIN-4: The **out_valid** and **out_data** must be asserted in 8 cycles.
   - (2.5%) SPEC MAIN-5: The **out_data** should be correct when **out_valid** is high.
   - (2.5%) SPEC MAIN-6: The data in the DRAM and SD card should be correct when **out_valid** is high.
   - (5%) The final states of DRAM and SD card (DRAM_final.dat and SD_final.dat) should be all correct (for correct design).
   - (5%) SPEC SD-1: Command format should be correct.
   - (5%) SPEC SD-2: The address should be within the legal range (0~65535).

- (2.5%) SPEC SD-3: CRC-7 check should be correct.
- (2.5%) SPEC SD-4: CRC-16-CCITT check should be correct.
- (10%) SPEC SD-5: Time between each transmission should be correct. (Only integer time units is allowed).

2. **Design Functionality**: 40% (Demo: Use our pattern.v and pseudo_SD.v, use your BRIDGE.v, and use Input.txt, DRAM_init.dat and SD_init.dat we provided.)
3. The grade of the 2nd demo would be **30% off**.
4. NO design performance score.
5. The design we provided will also check the pattern correctness. If the simulation cannot complete because of the error of pattern, the pattern demo will fail.
6. The specification of DRAM will check in pseudo_DRAM.v which we provide.

## Check the result

Check Test Bench

1. Include BRIDGE_encrypted.v in your TESTBED.v.
2. Check each specification in terminal.

| Specification | Command | Check the result on screen |
|---|---|---|
| **Pattern correctness** | *./01_run_vcs_rtl CORRECT* | Congratulations |
| **SPEC MAIN-1** | *./01_run_vcs_rtl SPEC_MAIN_1_{1~11}* | SPEC MAIN-1 FAIL |
| **SPEC MAIN-2** | *./01_run_vcs_rtl SPEC_MAIN_2_{1~2}* | SPEC MAIN-2 FAIL |
| **SPEC MAIN-3** | *./01_run_vcs_rtl SPEC_MAIN_3_1* | SPEC MAIN-3 FAIL |
| **SPEC MAIN-4** | *./01_run_vcs_rtl SPEC_MAIN_4_{1~2}* | SPEC MAIN-4 FAIL |
| **SPEC MAIN-5** | *./01_run_vcs_rtl SPEC_MAIN_5_{1~2}* | SPEC MAIN-5 FAIL |
| **SPEC MAIN-6** | *./01_run_vcs_rtl SPEC_MAIN_6_{1~5}* | SPEC MAIN-6 FAIL |
| **The final states of DRAM and SD card** | Write your own script to check it. | |
| **SPEC SD-1** | *./01_run_vcs_rtl SPEC_SD_1_{1~3}* | SPEC SD-1 FAIL |
| **SPEC SD-2** | *./01_run_vcs_rtl SPEC_SD_2_1* | SPEC SD-2 FAIL |
| **SPEC SD-3** | *./01_run_vcs_rtl SPEC_SD_3_1* | SPEC SD-3 FAIL |
| **SPEC SD-4** | *./01_run_vcs_rtl SPEC_SD_4_1* | SPEC SD-4 FAIL |
| **SPEC SD-5** | *./01_run_vcs_rtl SPEC_SD_5_{1~5}* | SPEC SD-5 FAIL |

- You can run *./07_check_pattern* to check all above specifications.
- Make sure the keyword printed on the screen is EXACTLY THE SAME as the one listed in the table above. You can print out other information according to your preference, but

only ONE keyword from the table above is permissible and it is indispensable.

Check Design

1. Include BRIDGE.v in your TESTBED.v.
2. Check each specification in terminal.

| Specification | Command | Check the result on screen |
|---|---|---|
| **Pattern correctness** | *./01_run_vcs_rtl* | Congratulations |

- You can run *./08_check_design* to check the design functionality.

## Note

1. Please submit PATTERN.v, pseudo_SD.v, BRIDGE.v in **Lab03/EXERCISE/09_SUBMIT**
    a. 1st_demo deadline: 2023/10/09(Mon.) 12:00:00
    b. 2nd_demo deadline: 2023/10/11(Wed.) 12:00:00
2. **Please upload the following file under 09_SUBMIT:**
    - PATTERN.v, pseudo_SD.v, BRIDGE.v
    - If your file **violates the naming rule**, you will **lose 5 points**.
    - Encryption of any uploaded files is prohibited or the demo will fail.
3. We provide the BRIDGE_encrypted.v for you. You only need to complete the pattern and check the correctness of each design we provided.
4. We provide you with pseudo_DRAM.v and check all the specification related to AXI4-Lite protocol.
5. You need to implement the BRIDGE.v. We only check the functionality using our pattern.
6. No hidden design and hidden pattern.
7. Directly access DRAM and SD card data in pattern. Use [instance name].[variable name] to access. The variable name in the provided pseudo_DRAM.v is “**DRAM**”. The default variable name in the provided pseudo_SD.v is “**SD**”.

```verilog
pseudo_DRAM u_DRAM (
    .clk(clk),
    .rst_n(rst_n),
    // write address channel
    .AW_ADDR(AW_ADDR),
    .AW_VALID(AW_VALID),
    .AW_READY(AW_READY),
    // write data channel
    .W_VALID(W_VALID),
    .W_DATA(W_DATA),
    .W_READY(W_READY),
    // write response channel
    .B_VALID(B_VALID),
    .B_RESP(B_RESP),
    .B_READY(B_READY),
    // read address channel
    .AR_ADDR(AR_ADDR),
    .AR_VALID(AR_VALID),
    .AR_READY(AR_READY),
    // read data channel
    .R_DATA(R_DATA),
    .R_VALID(R_VALID),
    .R_RESP(R_RESP),
    .R_READY(R_READY)
);
```

Variable name in pseudo_DRAM.v

```verilog
reg [63:0] DRAM[0:8191];
```

Access submodule element.

```verilog
u_DRAM.DRAM[my_addr_dram]
```

8. CRC function (C and Verilog)

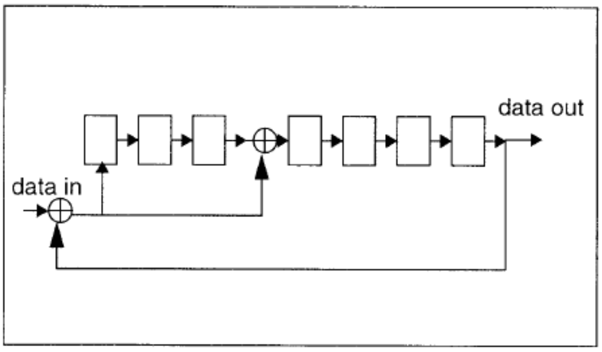



**Figure 4-10. CRC7 Generator/Checker**

```verilog
function automatic [6:0] CRC7; // Return 7-bit result
    input [39:0] data;  // 40-bit data input
    reg [6:0] crc;
    integer i;
    reg data_in, data_out;
    parameter polynomial = 7'h9;  // x^7 + x^3 + 1

    begin
        crc = 7'd0;
        for (i = 0; i < 40; i = i + 1) begin
            data_in = data[39-i];
            data_out = crc[6];
            crc = crc << 1;  // Shift the CRC
            if (data_in ^ data_out) begin
                crc = crc ^ polynomial;
            end
        end
        CRC7 = crc;
    end
endfunction
```

Try to implement CRC-16-CCITT function by yourself.

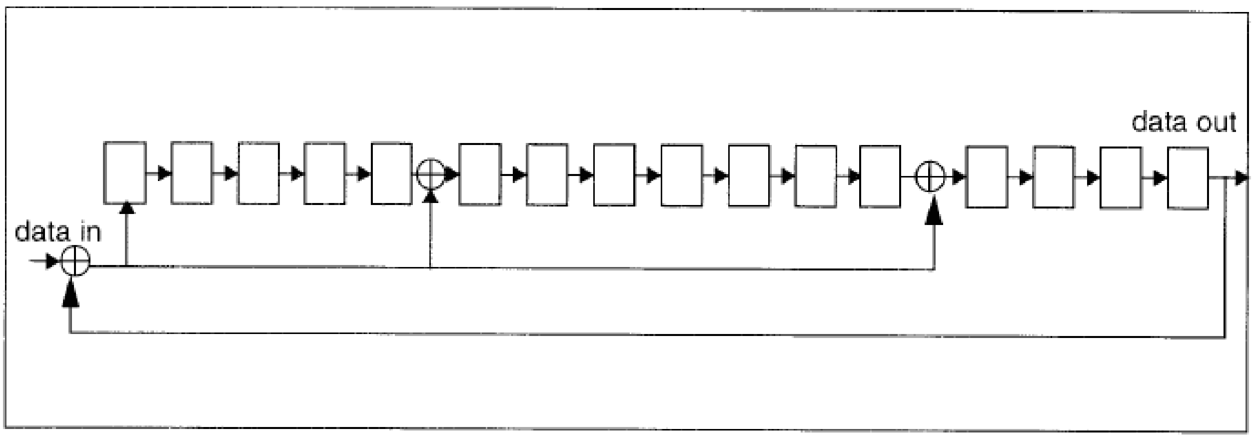
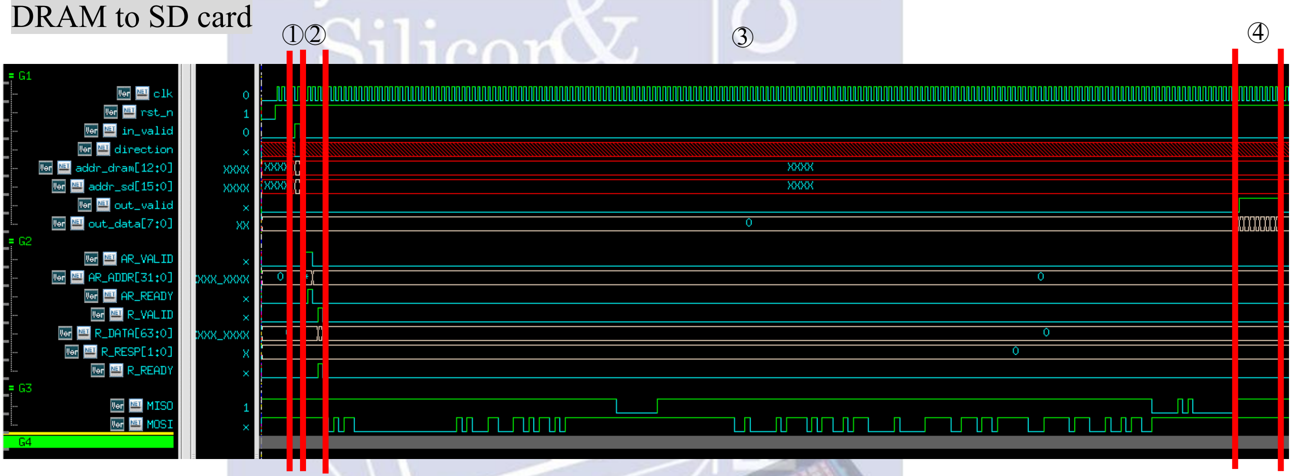



**Figure 4-11. CRC16 Generator/Checker**

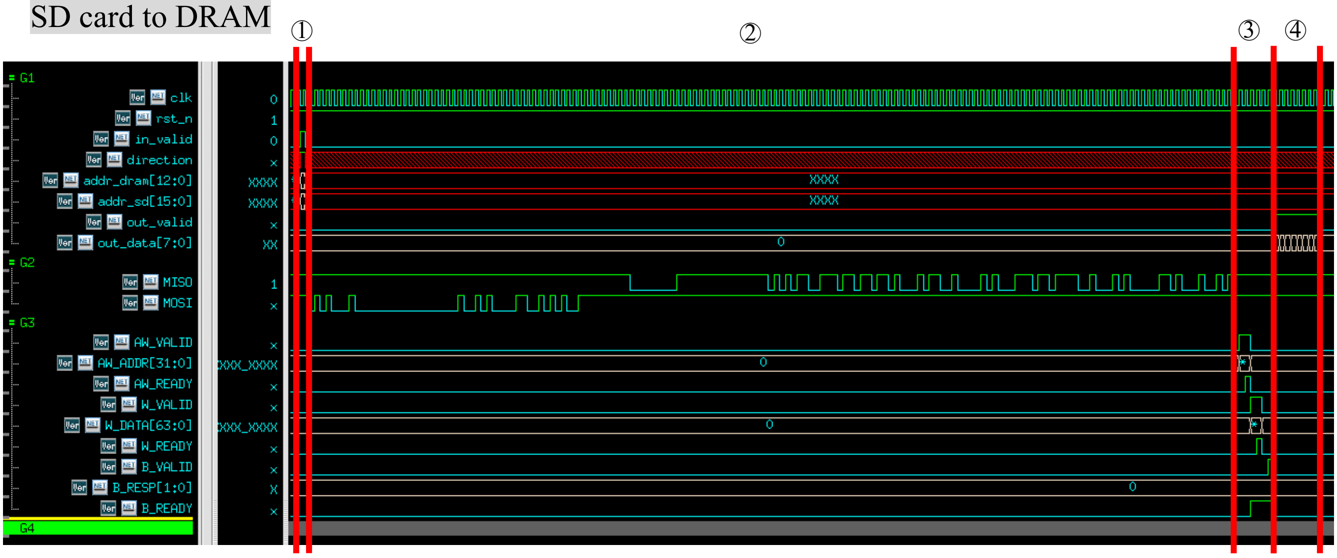
## Example Waveform

DRAM to SD card



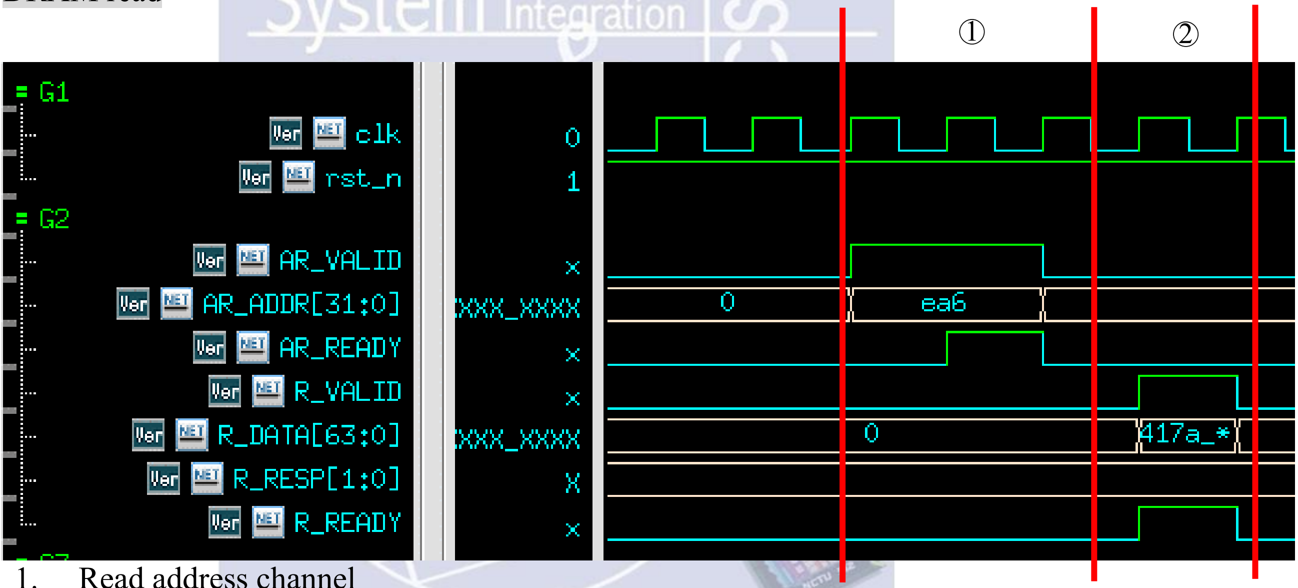

1. Input
2. DRAM read
3. SD card write
4. Output

SD card to DRAM



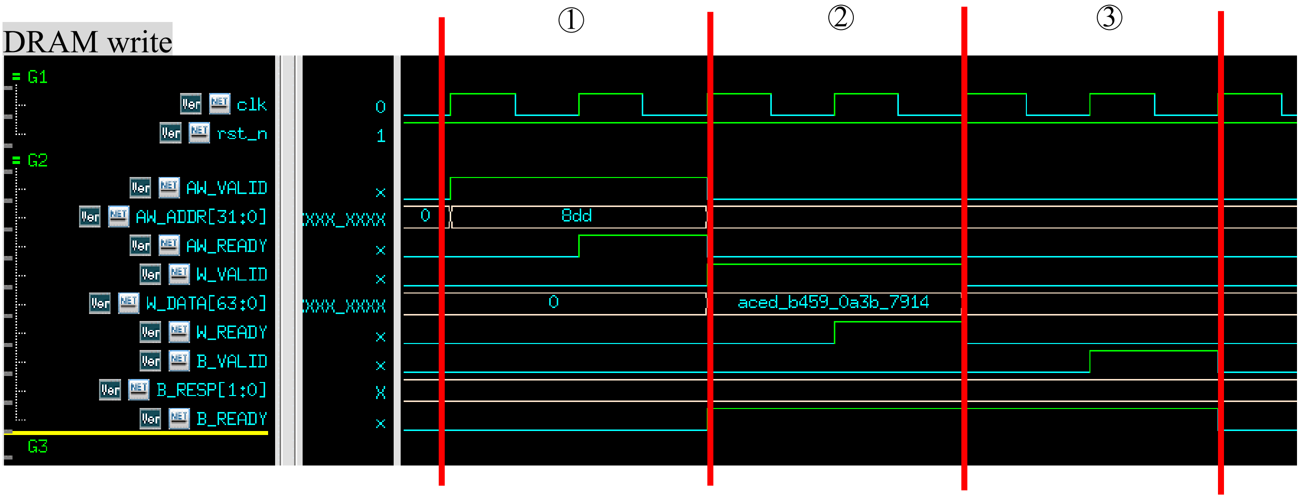

1. Input
2. SD card read
3. DRAM write
4. Output

DRAM read



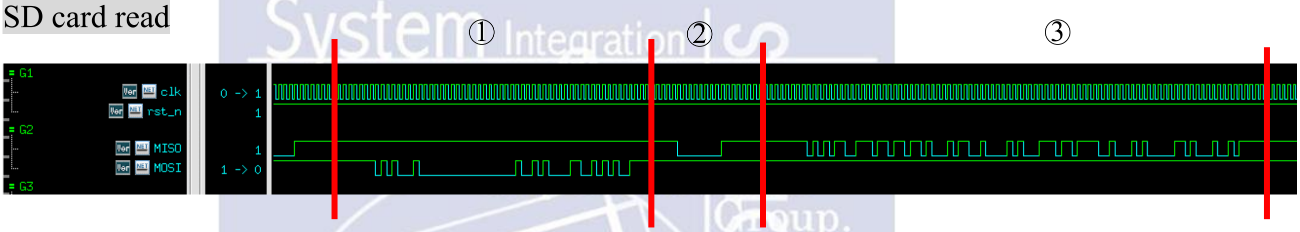

1. Read address channel
2. Read data channel

DRAM write



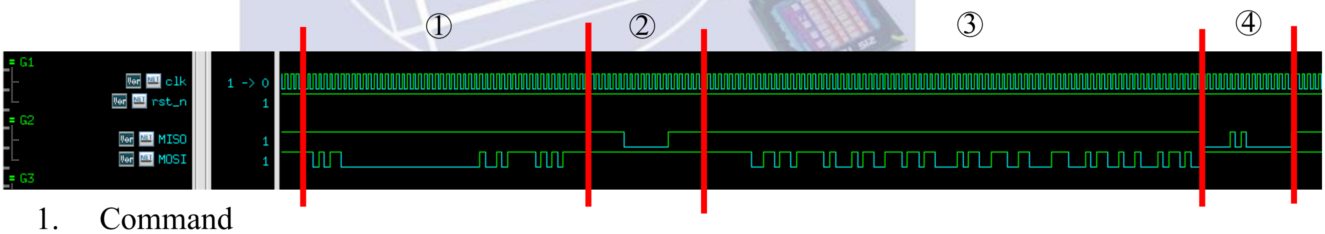

1. Write address channel
2. Write data channel
3. Write response channel

SD card read




1. Command
2. Response
3. Data

SD card write




1. Command
2. Response
3. Data
4. Data response