Digital Circuit and System
Lab04

20240328

TA: 王英育 Ying-Yu (Inyi) Wang
Institute of Electronics,
National Yang Ming Chiao Tung University

Behavioral Modeling

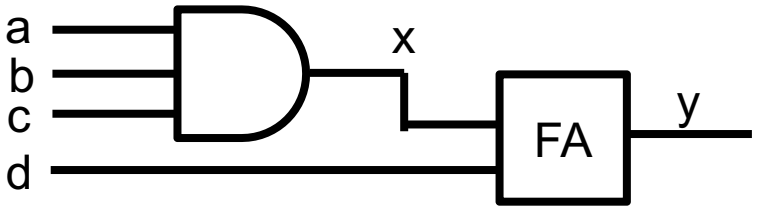
- **Using always construct (Procedure assignment)**
  - always @(a or b or c or d) begin

    x = a & b & c;

    y = x + d;

    end

- **Using assign construct (Continuous assignment)**
  - assign x = a & b & c;
  - assign y = x + d;



Behavioral Modeling

- Blocking assignment is used in always construct
  - The always block runs once when the signals in the **Sensitivity List** change value.

```
always @(a or b or c) begin
    x = a & b & c;
    y = x + d;
end
```

Simulation – Synthesize Mismatch

```
always @(a or b or c or d) begin
    x = a & b & c;
    y = x + d;
end
```

Correct

```
always @(a or b or c or d) begin
    y = x + d;
    x = a & b & c;
end
```

It's not in topological order

Simulation – Synthesize Mismatch

```
always @(*) begin
    x = a & b & c;
    y = x + d;
end
```

Use this !

Wire and Reg

- Wire (default = Z, high impedance)
  - Wire can't store any value.
  - It is often used in the combination circuit

1. Use it for port connection between devices

```
wire a, b, c;
FA u_FA (.a(a), .b(b), .cin(c));
```

2. Don't use it in procedure assignment

```
wire c;
always @(*) begin
    c = a + b;
end
```

3. Use it in continuous assignment

```
wire c;
assign c = a + b;
```

Wire and Reg

- Reg (default = X, unknown)
  - Reg represents a data storage element.
  - Reg must store their value until explicitly assigned in procedure assignment.

1. Use it in procedure assignment.
2. It may not imply a physical register.

- reg c;

```
always @(*) begin
    c = a + b;
end
```

An abstract register

- reg c;

```
always @(posedge clk) begin
    C <= a + b;
end
```

A physical register

Wire and Reg

- Wire and Reg can be declared as a vector and an array.
  - Vector:
    - Single element with multiple bits
      - wire [7:0] verctor1;
      - reg [3:0] vector2;
  - Array:
    - Multiple element with multiple bits
      - wire [7:0] array1 [0:31];
      - reg [3:0] array2 [0:127] [0:127];

Conditional Description

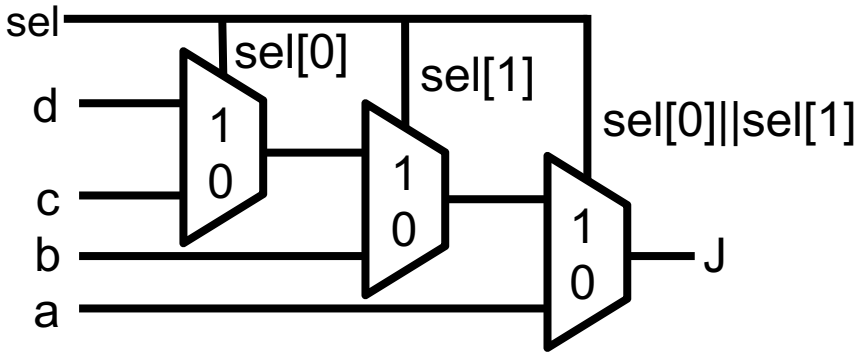
- Conditional assignment
  - It is used in cont. assignment.
  - It is the same as if-else statement.
  - ? : => c = (condition_a) ? a : b;
- If-else statement
  - It is used in proc. assignment.
  - It often infers a cascaded encoder.
  - It inputs signals with different arrival time.
  - It is involved in priority.
- Case
  - It is used in proc. assignment.
  - It is better if there is no issue for priority.
  - It is generally simulated faster then if-else statement.

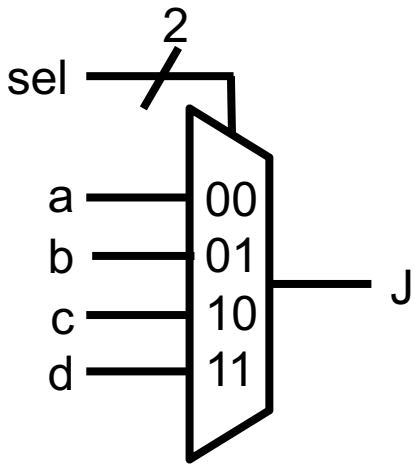
Conditional Description

```
assign J = (sel == 2'b00) ? a :
           ((sel == 2'b01) ? b :
           ((sel == 2'b10) ? c :
           ((sel == 2'b11) ? d)));
```

```
always @(*) begin
   if (sel == 2'b00) J = a;
   else if (sel == 2'b01) J = b;
   else if (sel == 2'b10) J = c;
   else J = d;
end
```

```
always @(*) begin
   case (sel)
      2'b00: J = a;
      2'b01: J = b;
      2'b10: J = c;
      2'b11: J = d;
   endcase
end
```





Latch

- You need to avoid latches in the combinational circuit.
  - Avoid combinational feedback.
  - Avoid incomplete if-else statement.
  - Avoid incomplete case statement.

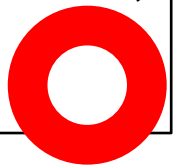
```
always @(*) begin
    A = B;
    B = A;
end
```

```
always @(*) begin
    if (Q == 2'd1) A = B;
    else if (Q == 2'd2) A = C;
end
```

```
always @(*) begin
    case (Q)
        2'd1: A = B;
        2'd2: A = C;
    endcase
end
```

```
always @(*) begin
    out = out + 1;
end
```

```
always @(*) begin
    if (Q == 2'd1) A = B;
    else if (Q == 2'd2) A = C;
    else A = D;
end
```



```
always @(*) begin
    case (Q)
        2'd1: A = B;
        2'd2: A = C;
        default: A = D;
    endcase
end
```

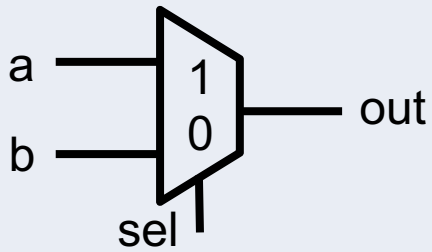
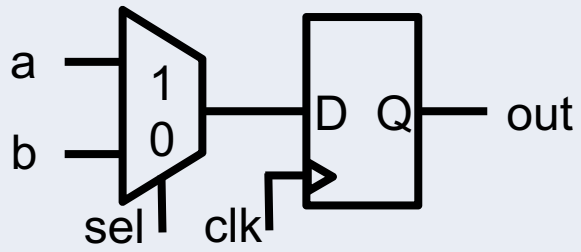
Blocking and Non-Blocking

- Blocking assignment
  - Evaluations and assignments are updated **immediate and in order**.
  - Syntax: =
    - J = a;
  - Combinational blocks must use blocking assignments!
- Non-Blocking
  - Evaluations and assignments are carried out **at the same time considering any specific sequence or interdependency between them**.
  - Syntax: <=
    - J <= a;
  - Sequential logic must use non-blocking assignments!

Blocking and Non-Blocking

Blocking	Non-Blocking
Combination	Sequential

always @(*) begin
  if (sel) out = a;
  else out = b;
end
	always @(posedge clk) begin
	  if (sel) out <= a;
	  else out <= b;
	end

=	<=
	
不會和Non-Blocking一起出現	不會和Blocking一起出現

Reset

- Register must be reset in sequential logic!
  - Synchronous reset
    - always @(posedge clk) begin
      if (reset) a <= 0;
      else a <= b;
      end

    – Avoid metastability (glitch)
  - Asynchronous reset
    - always @(posedge clk or posedge reset) begin
      if (reset) a <= 0;
      else a <= b;
      end

    – Independent of clock signal
    – immediate
    – Less area

Reset

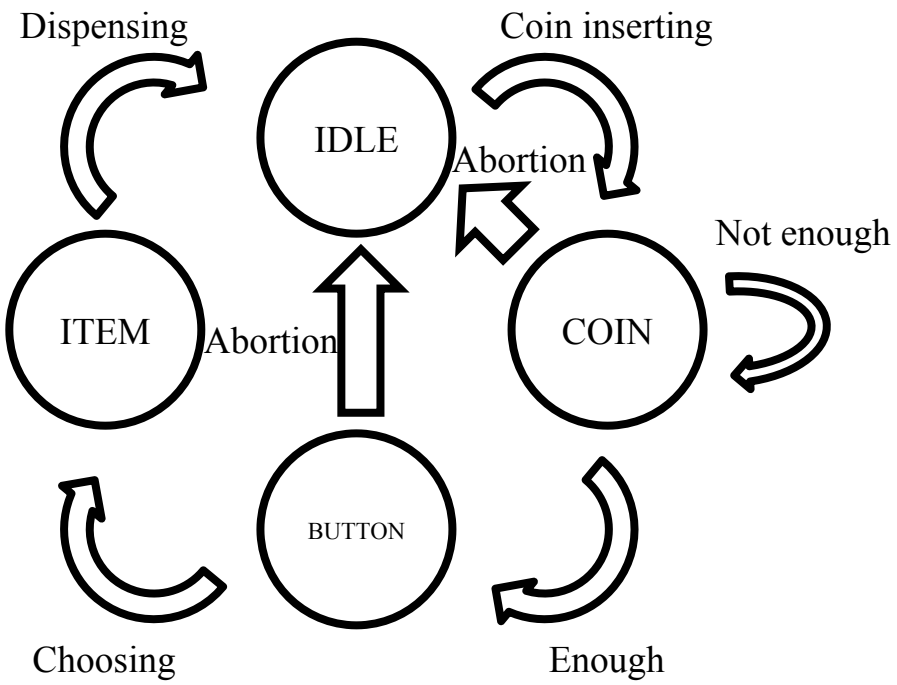
- Register must be reset in sequential logic!
  - What is a's value?

```
reg [31:0] a;
always @(posedge clk or posedge reset) begin
    a <= (a + 86) * 36;
end
```

  - If you know, tell me because I don't know.

Finite State Machine (FSM)

- Vending machine



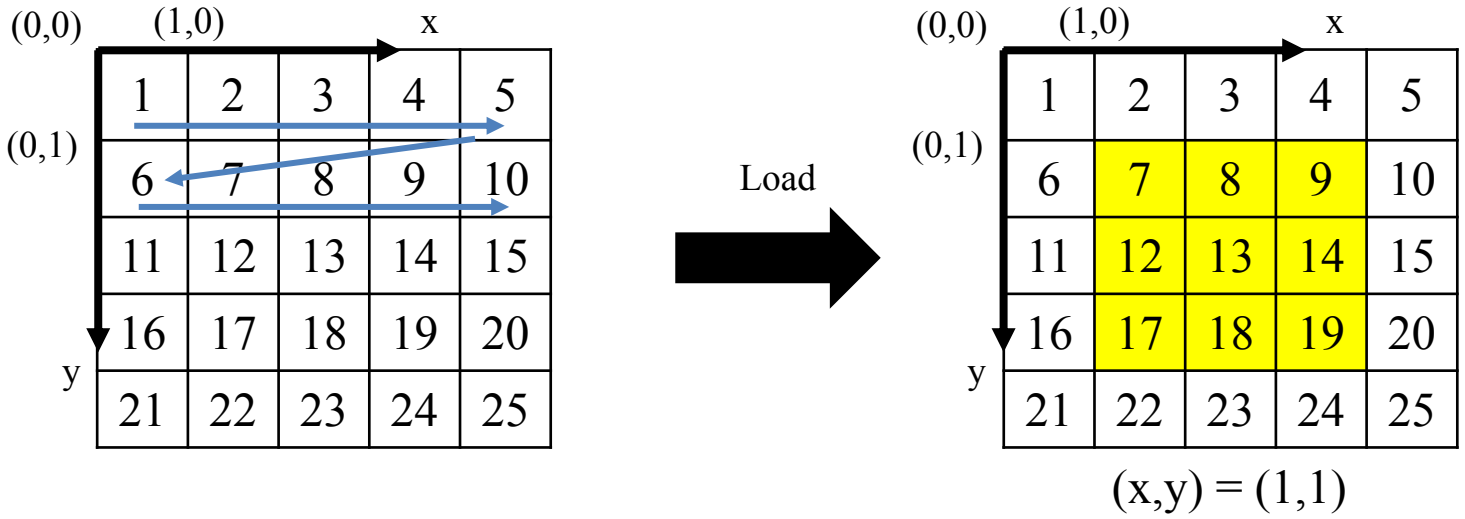
Lab

- Image display control:

Signal	I/O	Bit Width	Description
clk	I	1	Clock signal (Posedge design)
reset	I	1	Reset signal for active high asynchronout
cmd	I	3	Command signal (Only be valid when cmd_valid is high and busy is low)
cmd_valid	I	1	Valid command signal (Be high for command is valid)
datain	I	8	Input port for 8-bits image data
dataout	O	8	Output port for 8-bits image data
output_valid	O	1	Valid output signal (Be high for output is valid)
busy	O	1	Busy signal (Be high for blocking other new input commands)

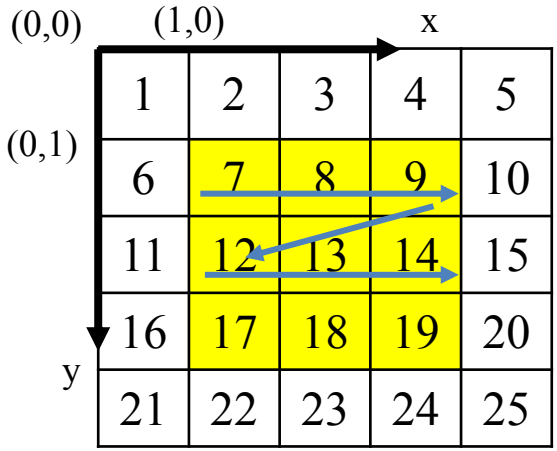
Lab

- Image display control:
  - Input image is 5x5: 25 elements of image input in order.
  - There are five commands in this design: load, shift right, shift left, shift up, shift down.
    - The first command must be load.



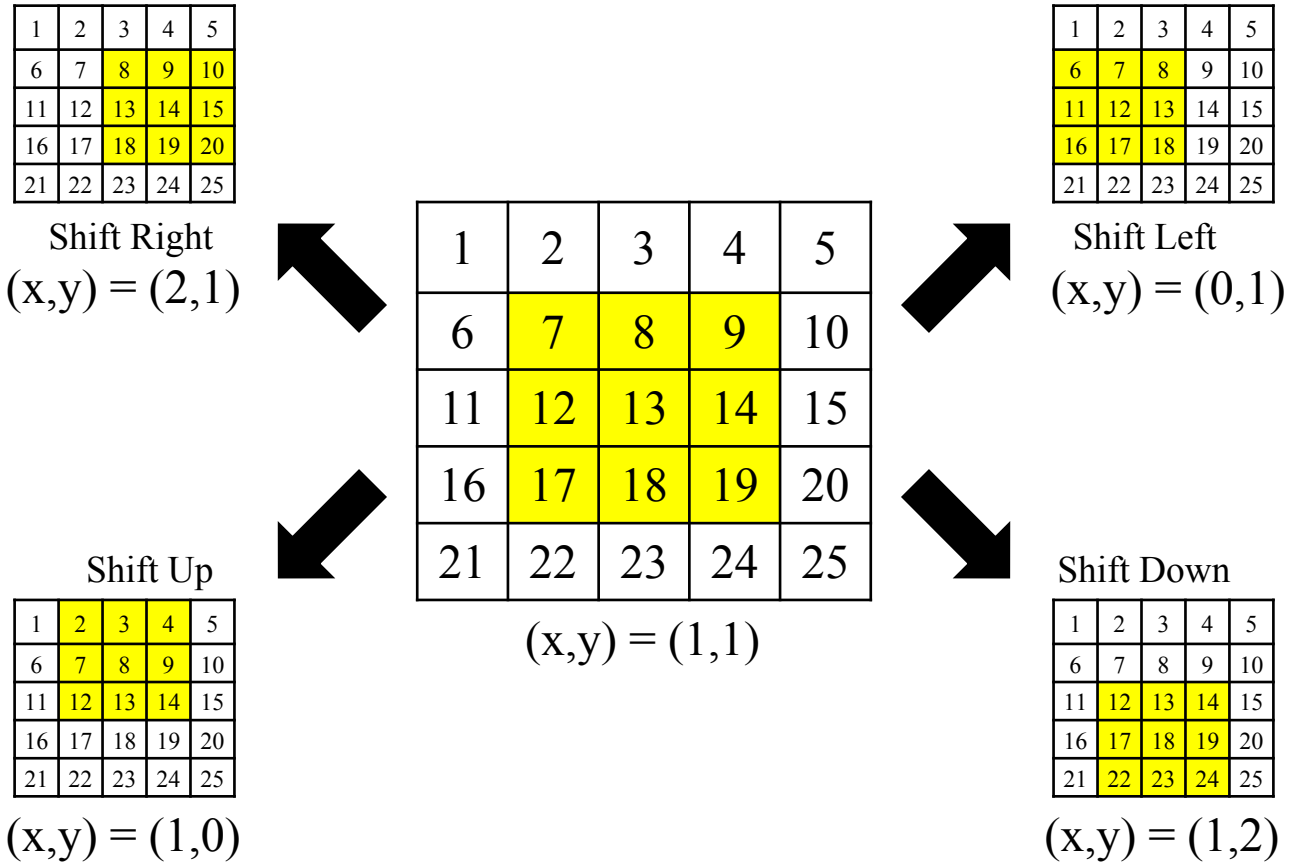
Lab

- Image display control:
  - Output image is 3x3: 9 elements of image output in order.
  - When data output, the signal of out_valid must be high.
  - Busy must be high during the processing and become low for new command after finishing the output.



Lab

- There are four directions for shifting:



Lab

- If the output image on the boundary and the next command makes the output image exceed the image range, the output results will remain the previous results.

(x,y) = (2,0)

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Shift Right

(x,y) = (2,0)

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Shift UP

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

(x,y) = (1,0)

(x,y) = (1,0)

Lab

- The Verilog file is lcd.ctrl.v

```
module LCD_CTRL(clk, reset, datain, cmd, cmd_valid, dataout, output_valid, busy);
input           clk;
input           reset;
input   [7:0]   datain;
input   [2:0]   cmd;
input           cmd_valid;
output reg [7:0] dataout;
output reg       output_valid;
output reg       busy;

///// Reg /////
reg [1:0] cs, ns;
reg [2:0] x_cnt, y_cnt;
reg [7:0] image_buffer [0:4][0:4];
reg [2:0] x, y;
reg [3:0] OUT_cnt;
///////////////

///// Parameter /////
parameter IDLE = 2'd0;
parameter READ = 2'd1;
parameter EXE  = 2'd2;
parameter OUT  = 2'd3;
/////////////////////
```

Lab

- FSM is completed in this Lab; however, if you want to design by yourself, do it!

```
///// FSM /////
always @(posedge clk or posedge reset) begin···
end

always @(*) begin
    case (cs)
        IDLE: begin···
        end

        READ: begin···
        end

        EXE: begin···
        end

        OUT: begin···
        end
    endcase
end
///////////////
```

Lab

- Please fill out the part of busy setting!
  - Busy signal is high for blocking other new input commands.
  - Busy must be high during the processing!

```
///// Busy Setting /////

//////////////////////////
```

Lab

- This part is incomplete pointer.
  - The part for input is complete in *if (cs == READ)* statement.

- Please finish the pointer part for output!

```
///// Pointer /////
always @(posedge clk or posedge reset) begin
    if (reset) begin
        x_cnt <= 3'd0;
        y_cnt <= 3'd0;
    end else if (cs == READ) begin
        if (x_cnt == 3'd4) begin
            if (y_cnt == 3'd4) begin
                x_cnt <= 3'd0;
                y_cnt <= 3'd0;
            end else begin
                x_cnt <= 3'd0;
                y_cnt <= y_cnt + 3'd1;
            end
        end else begin
            x_cnt <= x_cnt + 3'd1;
            y_cnt <= y_cnt;
        end
    end
end
//////////////////
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

7	8	9
12	13	14
17	18	19

Lab

- The part for image input is complete; however, if you change the FSM part, you may modify this part.

```
///// Image Input /////
integer i, j;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        for (i = 0; i < 5; i = i + 1) begin
            for (j = 0; j < 5; j = j + 1) begin
                image_buffer[i][j] <= 8'd0;
            end
        end
    end else if (cs == READ) begin
        image_buffer[x_cnt][y_cnt] <= datain;
    end
end
//////////////////////
```

Be careful of using for loop

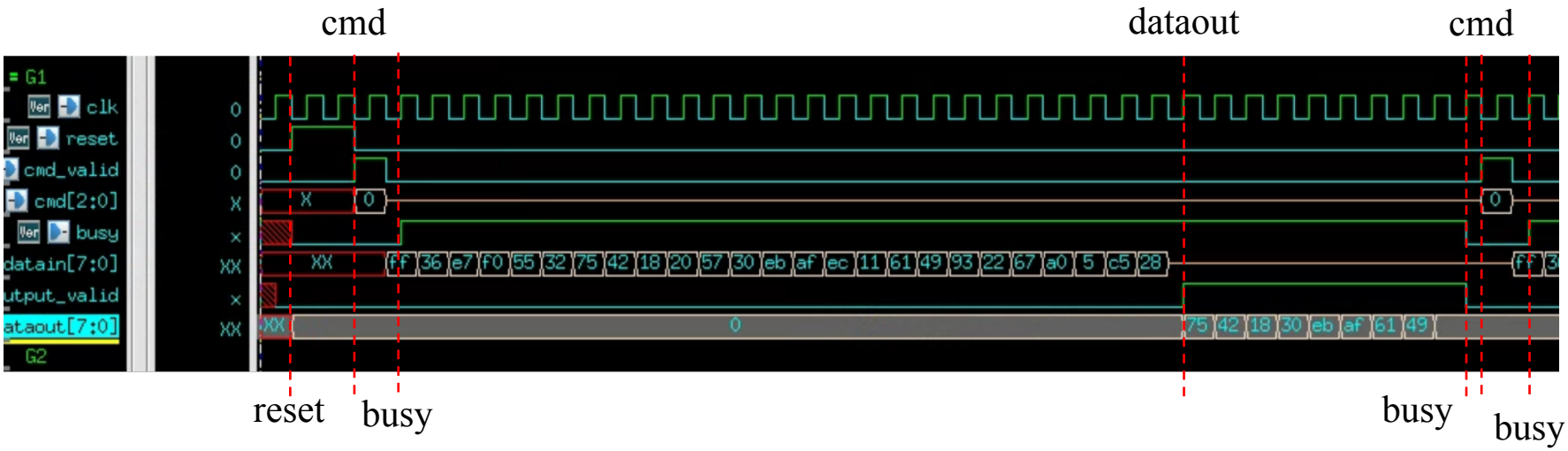
Lab

- Please fill out the part of execute, output setting, and image output.
  - Execute: how to do when a command inputs?
  - Output Setting: when does the output_valid set as high?
  - Image Output: how to correctly output the dataout in order?

```
///// Execute /////

////////////////////

///// Output Setting /////

//////////////////////////

///// Image Output /////

//////////////////////
```

Time Diagram

- Asynchronous reset
- cmd_valid and cmd
- 25 cycles for datain
- 9 cycles for dataout
- output_valid and output
- Busy for new command



Result

- Your design must pass all pattern.
- If it doesn't pass all pattern, find which point that makes wrong.
  - Report will show when your design goes wrong and what the golden output is.

```
================================================================================
                                Congratulations!
                            You have passed this pattern!!
  Please go to 09_SUBMMIT folder demo and submit your code (No need to submmit to E3 platform)
================================================================================


Your execution cycles :         406 cycles

-------------------------------------------

$finish called from file "PATTERN.v", line 149.
$finish at simulation time                4090000
         V C S   S i m u l a t i o n   R e p o r t
```

```
 ============================================================
                  PATTERN   72 FAIL
          Your answer =    17 Correct answer =  87
 ============================================================
$finish called from file "PATTERN.v", line 109.
$finish at simulation time                  143500
           V C S   S i m u l a t i o n   R e p o r t
Time: 1435000 ps
```

Score

- Finish this Lab:
  - Today: 100 points
    - On Friday: 80 points
    - On Saturday: 60 points
    - On Sunday: 40 points
    - On Monday: 20 points
    - After Monday: 0 points

Command List

- Extract files from TA's directory (home directory): (It will be valid at 16:30 p.m.)
  - **tar -xvf ~DCSTA01/Lab04.tar**
- Verilog RTL simulation (01_RTL/):
  - **./01_run_vcs_rtl**
- Observe waveform to debug
  - **nWave &**
  - find *.fsdb
  - shift + L for reload fsdb file
- Synthesis (02_SYN/) (optional):
  - **./01_run_dc_shell**
- Gate_level simulation (03_GATE/) (optional):
  - **./01_run_vcs_gate**

Command List

- Submit your Lab:
  - **cd 09_SUBMIT**
  - **./00_tar 10**
    - 10 means the cycle time in this design. Don't change in this Lab.
  - **./01_submit**
  - **./02_check**
    - We only score your RTL design, so you don't need to care your results of SYN and GATE if they fails.