Lab Session 5
Design of FSM and Memory

Instructor: Lih-Yih Chiou

TA: Michael

Date: 2021/03/24

Outline

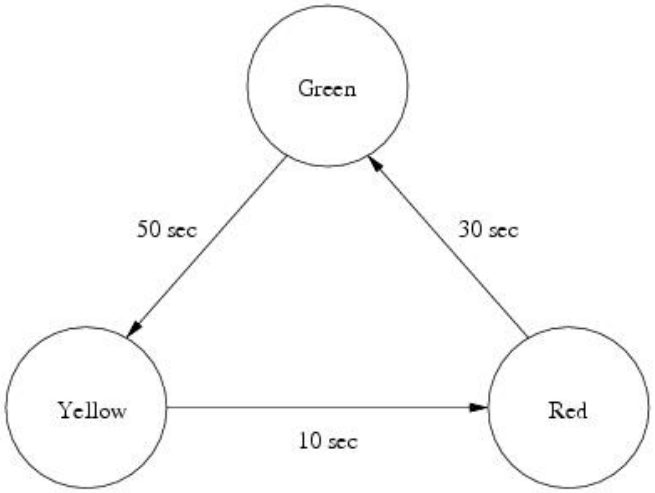
- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

Outline

- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

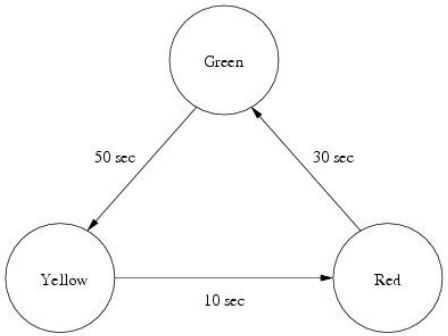
FSM Introduction (1/2)

- A Finite State Machine (FSM) is generic sequential system that consists both combinational networks and memory elements.
- Storage elements hold the machine's state
- The machine's inputs and outputs are called primary inputs and primary outputs



FSM Introduction (2/2)

- General FSM design procedure:
  1) Determine the inputs and outputs of the system
  2) System control pins (clk, rst …)
  3) Determine number of possible operating states of the machine
  4) Construct the state diagram
  5) State reductions if possible
  6) Derive the Boolean Equations for each state and the output in terms of the inputs and control signals.
  7) Implement the circuit

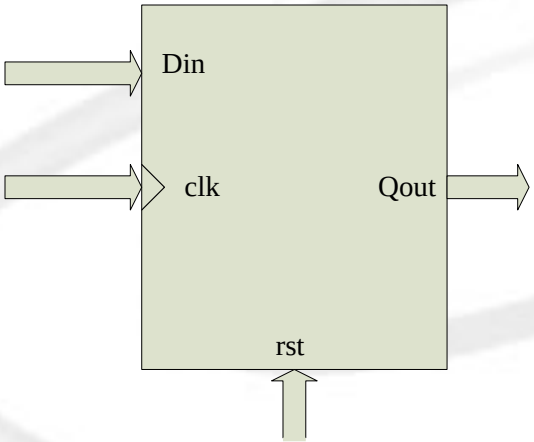


Outline

- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

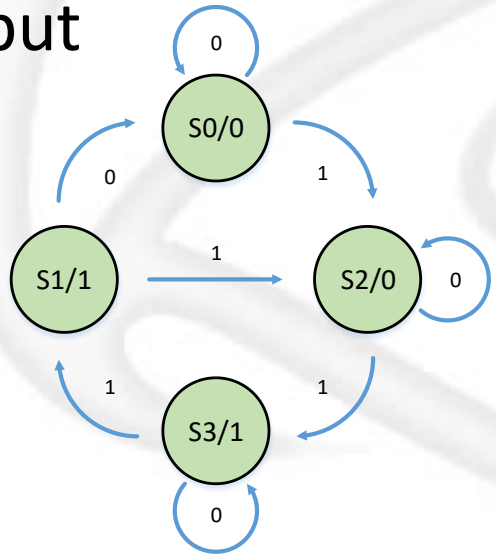
Moore Machine (1/2)

- The primary outputs depend only on the state
- State diagram and state table are used to describe a Finite State Machine
- Example of a system with the following configuration



Moore Machine (2/2)

- Use binary numbers to represent each state
  - ➔ parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
- Reading data input and changes state for every clock cycles
- Upon rst, machine goes back to state S0
- Next state will be determined by current state and input



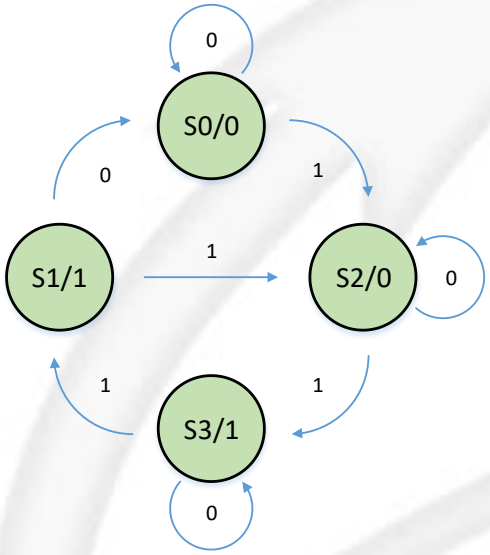
Current State	Next State		qout
	din=0	din=1	
S0=00	S0	S2	0
S1=01	S0	S2	1
S2=10	S2	S3	0
S3=11	S3	S1	1

Verilog Code & Testbench

```
1  module moore (clk, rst, din, qout);
2
3  output qout;
4  input clk, rst, din;
5
6  reg qout;
7  reg [1:0] cs, ns;
8
9  parameter s0 = 2'b00,
10           s1 = 2'b01,
11           s2 = 2'b10,
12           s3 = 2'b11;
13
14 always @(posedge clk or posedge rst) begin
15   if (rst)
16     cs <= s0;
17   else
18     cs <= ns;
19 end
20
21 always @(cs or din) begin
22   case (cs)
23     s0: ns = ...;
24     s1: ns = ...;
25     s2: ns = ...;
26     s3: ns = ...;
27   endcase
28 end
29
30 always @(cs) begin
31   case (cs)
32     s0: qout = ...;
33     s1: qout = ...;
34     s2: qout = ...;
35     s3: qout = ...;
36   endcase
37 end
38
39 endmodule
```

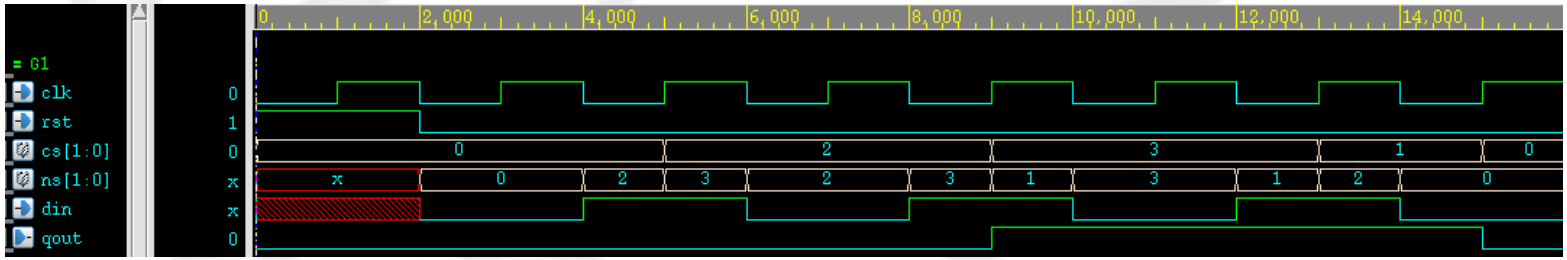
```
1  `timescale 1ns/10ps
2  `include "moore.v"
3
4  module moore_tb;
5
6  reg clk, rst, din;    //inputs
7  wire qout;            //outputs
8
9  moore m0 (.qout(qout), .clk(clk), .rst(rst), .din(din));
10
11 initial $monitor($time, " clk=%d, rst=%d, din=%d, qout=%d",
12   clk, rst, din, qout);
13
14 initial clk = 1'b0;
15 always #10 clk = ~clk;
16
17 initial begin
18       rst=1;
19   #20 rst=0; din=0;
20   #20 din=1;
21   #20 din=0;
22   #20 din=1;
23   #20 din=0;
24   #20 din=1;
25   #20 din=0;
26   #20 $finish;
27 end
28
29 initial begin
30   `ifdef FSDB
31   $fsdbDumpfile("moore.fsdb");
32   $fsdbDumpvars;
33   `endif
34   #10000 $finish;
35 end
36
37 endmodule
```

Simulation Results




```
*Verdi* Loading libsscore_ius152.so
ncsim> source /usr/cad/cadence/INCISIV/cur/tools/inca/files/ncsimrc
ncsim> run
                   0 clk=0, rst=1, din=x, qout=0
                  10 clk=1, rst=1, din=x, qout=0
                  20 clk=0, rst=0, din=0, qout=0
                  30 clk=1, rst=0, din=0, qout=0
                  40 clk=0, rst=0, din=1, qout=0
                  50 clk=1, rst=0, din=1, qout=0
                  60 clk=0, rst=0, din=0, qout=0
                  70 clk=1, rst=0, din=0, qout=0
                  80 clk=0, rst=0, din=1, qout=0
                  90 clk=1, rst=0, din=1, qout=1
                 100 clk=0, rst=0, din=0, qout=1
                 110 clk=1, rst=0, din=0, qout=1
                 120 clk=0, rst=0, din=1, qout=1
                 130 clk=1, rst=0, din=1, qout=1
                 140 clk=0, rst=0, din=0, qout=1
                 150 clk=1, rst=0, din=0, qout=0
Simulation complete via $finish(1) at time 160 NS + 0
./moore_tb.v:25         #20 $finish;

ncsim> exit
```

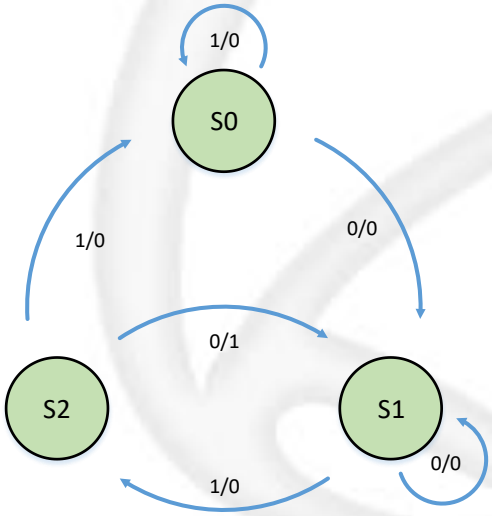


Outline

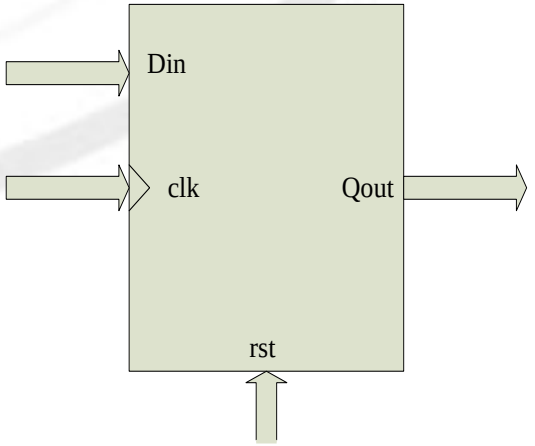
- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

Mealy Machine (1/2)

- The primary outputs are a function of both the primary inputs and states
- System detecting sequence of 010 from input data
- Construct the state diagram and state table as below :



Current State	Next State, output	
	din=0	din=1
S0=00	S1,0	S0,0
S1=01	S1,0	S2,0
S2=11	S1,1	S0,0



Mealy Machine (2/2)

- The Verilog Code for the Mealy machine is as shown below

```
1  module mealy (clk, rst, din, qout);
2
3  output qout;
4  input clk, rst, din;
5
6  reg qout;
7  reg [1:0] cs, ns;
8
9  parameter s0 = 2'b00,
10           s1 = 2'b01,
11           s2 = 2'b10;
12
13 always @(posedge clk or posedge rst) begin
14   if (rst)
15     cs <= s0;
16   else
17     cs <= ns;
18 end
19
```

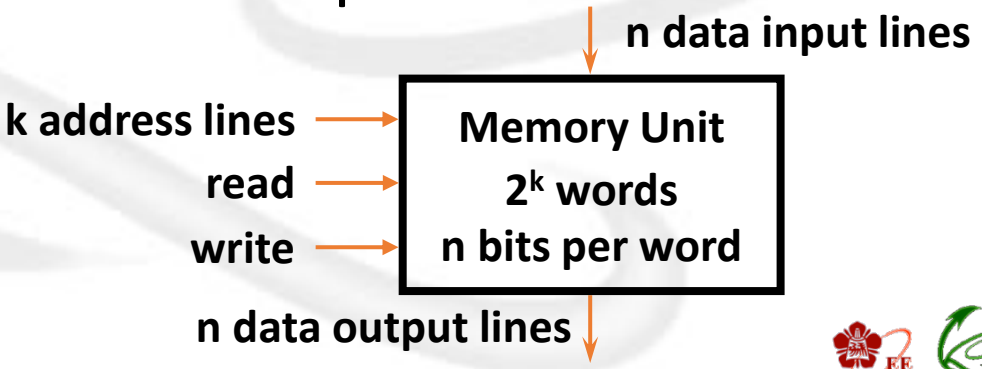
```
20 always @(cs or din) begin
21   case (cs)
22     s0: ns = ...;
23     s1: ns = ...;
24     s2: ns = ...;
25     default: ...;
26   endcase
27 end
28
29 always @(cs or din) begin
30   case (cs)
31     s0: qout = ...;
32     s1: qout = ...;
33     s2: qout = ...;
34     default: ...;
35   endcase
36 end
37
38 endmodule
```

Outline

- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

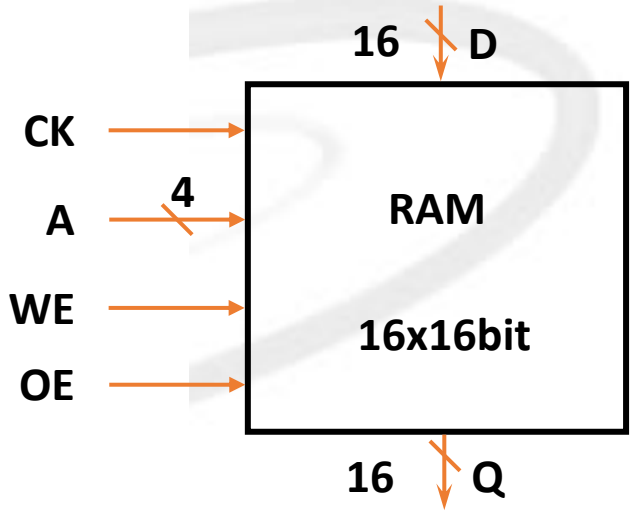
Memory Elements

- Memory is a collection of binary storage cells together with associated circuits needed to transfer information
- During read operation, a specific word is selected by applying k-bit address and the selected word will be put on data output
- During write operation, the input address selects the memory location to be written with the data appear at the data input



Random-Access Memory

```
1  `timescale 1ns/10ps
2
3  module RAM (CK, A, WE, OE, D, Q);
4
5  /*Please rewrite this example code according to the assignment*/
6
7    input         CK;
8    input  [3:0]  A;
9    input         WE;
10   input         OE;
11   input  [15:0] D;
12   output [15:0] Q;
13
14   reg    [15:0] Q;
15   reg    [3:0]  latched_A;
16   reg    [15:0] memory [0:15];
17
18   always @(posedge CK) begin
19     if (WE) begin
20       memory[A] <= D;
21     end
22     latched_A <= A;
23
24   end
25
26   always @(*) begin
27     if (OE) begin
28       Q = memory[latched_A];
29     end
30     else begin
31       Q = 16'hz;
32     end
33   end
34
35 endmodule
```

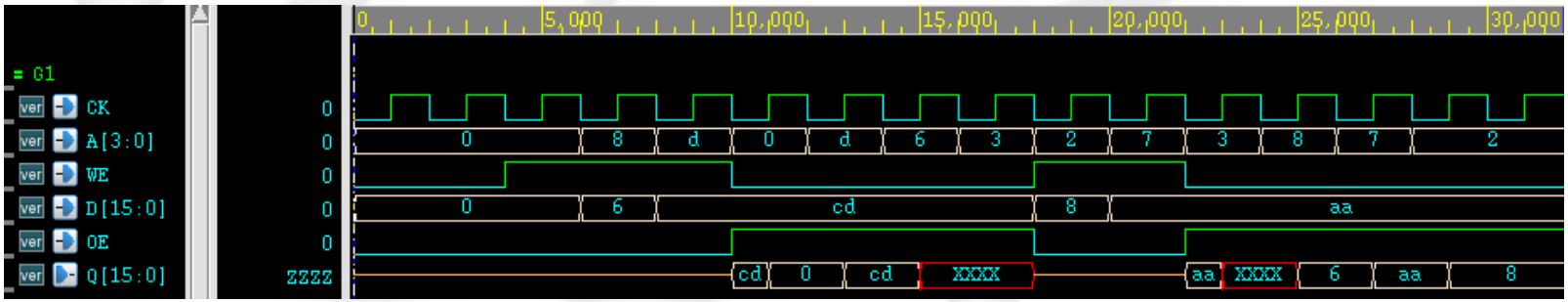


Testbench

```
1  `timescale 1ns/10ps
2  `include "RAM.v"
3  module RAM_tb;
4    reg         clk, read_enable, write_enable;
5    reg [15:0] data_in;
6    reg [3:0]  address;
7    wire[15:0] data_out;
8    integer    i;
9    RAM ram1 (clk, address, write_enable, read_enable, data_in, data_out);
10   initial clk=1'b0;
11   always #10 clk=~clk;
12   initial begin
13         read_enable=0;   write_enable=0;
14         address=4'd0;    data_in=16'd0;
15     #20 read_enable=0;   write_enable=0;
16     #20 write_enable=1;
17         address = 4'd0;  data_in=16'h0;
18     // Please add some test pattern to verify your module
19
20     // Display result
21     #20 for(i=0;i<16;i=i+1)
22         $display($time, " RAM[%d]=%h, ", i, ram1.memory[i]);
23     #20 $finish;
24   end
25   initial begin
26     `ifdef FSDB
27       $fsdbDumpfile("RAM.fsdb");
28       $fsdbDumpvars;
29     `endif
30     #10000 $finish;
31   end
32 endmodule
```

Simulation Result

```
300 RAM[          0]=0000,
300 RAM[          1]=xxxx,
300 RAM[          2]=0008,
300 RAM[          3]=xxxx,
300 RAM[          4]=xxxx,
300 RAM[          5]=xxxx,
300 RAM[          6]=xxxx,
300 RAM[          7]=00aa,
300 RAM[          8]=0006,
300 RAM[          9]=xxxx,
300 RAM[         10]=xxxx,
300 RAM[         11]=xxxx,
300 RAM[         12]=xxxx,
300 RAM[         13]=00cd,
300 RAM[         14]=xxxx,
300 RAM[         15]=xxxx,
Simulation complete via $finish(1) at time 320 NS + 0
./RAM_tb.v:41     #20 $finish;
ncsim> exit
```

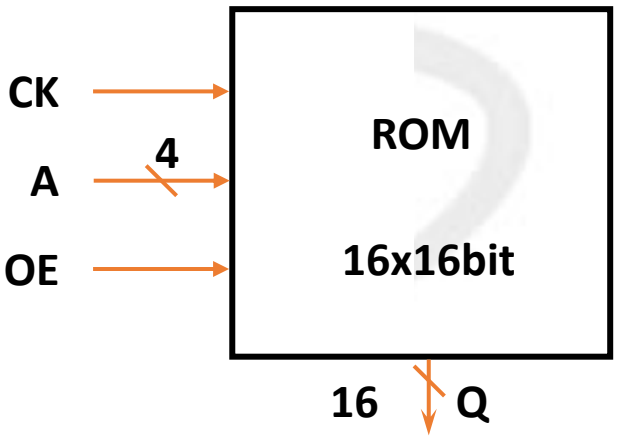


Outline

- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

Read-Only Memory

```
1  `timescale 1ns/10ps
2
3  module ROM (CK, A, OE, Q);
4
5  /*Please rewrite this example code according to the assignment*/
6
7    input          CK;
8    input  [3:0]   A;
9    input          OE;
10   output [15:0]  Q;
11
12   reg    [15:0]  Q;
13   reg    [3:0]   latched_A;
14   reg    [15:0]  memory [0:15];
15
16   always @(posedge CK) begin
17     latched_A <= A;
18   end
19
20   always @(*) begin
21     if (OE) begin
22       Q = memory[latched_A];
23     end
24     else begin
25       Q = 16'hz;
26     end
27   end
28 endmodule
```



Load Data into Memory

- Test program “ROM_data.dat” contains the values that are stored in the memory.
- Added the following block into the testbench to load “ROM_data.dat” into ROM.

```
initial begin
   $readmemh("ROM_data.dat",rom1.memory);
end
```

File name

Hierarchical name

- The system task $readmemb (read in binary) and $readmemh (read in hex) will read data in specified file.
- It takes 2 arguments, the file name and the memory register’s hierarchical name. It reads the data into the memory register.

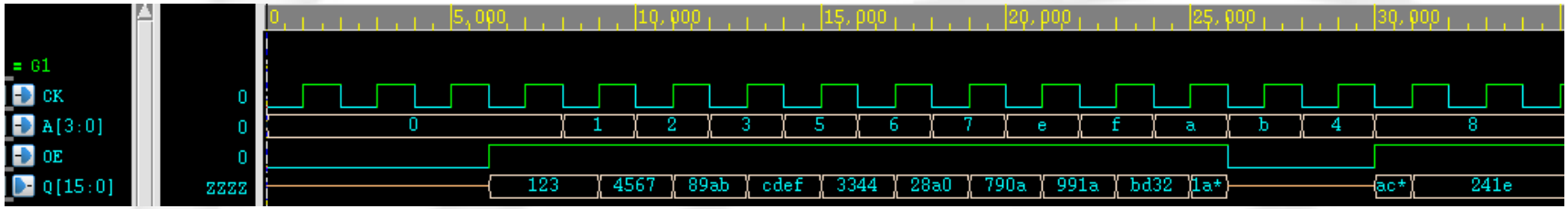
Testbench

```
1  `timescale 1ns/10ps
2  `include "ROM.v"
3  module ROM_tb;
4    reg         clk;
5    reg         rst;
6    reg         read_enable;
7    reg  [3:0]  address;
8    wire [15:0] data_out;
9
10   ROM rom1 (clk, address, read_enable, data_out);
11
12   initial clk=1'b0;
13   always #10  clk=~clk;
14
15   initial begin
16         clk = 0; rst = 0;
17         read_enable = 0; address = 4'd0;
18     #20 rst = 1;
19     #40 rst = 0; read_enable = 1;
20     // Please add some test pattern to verify your module
21
22   end
23
24   initial begin
25     $readmemh("ROM_data.dat",rom1.memory);
26   end
27
28   initial begin
29     `ifdef FSDB
30     $fsdbDumpfile("ROM.fsdb");
31     $fsdbDumpvars();
32     #1000 $finish;
33     `endif
34   end
35 endmodule
```

ROM_data.dat

```
1  0123
2  4567
3  89ab
4  cdef
5  ac87
6  3344
7  28a0
8  790a
9  241e
10 5398
11 1ae5
12 d123
13 ff3c
14 62d0
15 991a
16 bd32
```

Simulation Result



Outline

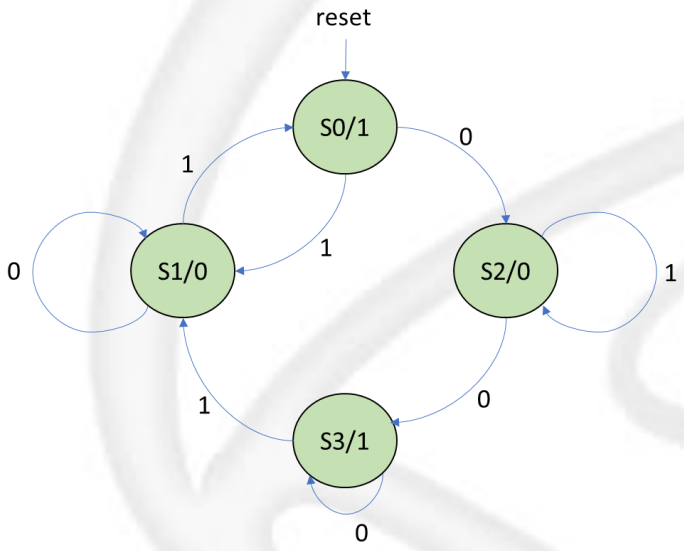
- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

Homework

- Due day : **2021-04-14, Wed, 15:00**
- Prob A :
  - ➔ Design a "Moore Machine" according to given specifications and verify its' functionality.
- Prob B :
  - ➔ Design a "Mealy Machine" according to given specifications and verify its' functionality.
- Prob C :
  - ➔ Design the "Memory" according to given specifications and verify its' functionality.
- Prob D:
  - ➔ Design a "MAC with shift register".
- Prob E :
  - ➔ Combining the "Input_memory", "Output_memory", "Controller", "Grayscale" to form a simple system.
- Attention
  - ➔ Make sure all your verilog code can be compiled on the environment in SoC Lab.

Prob A : Moore Machine

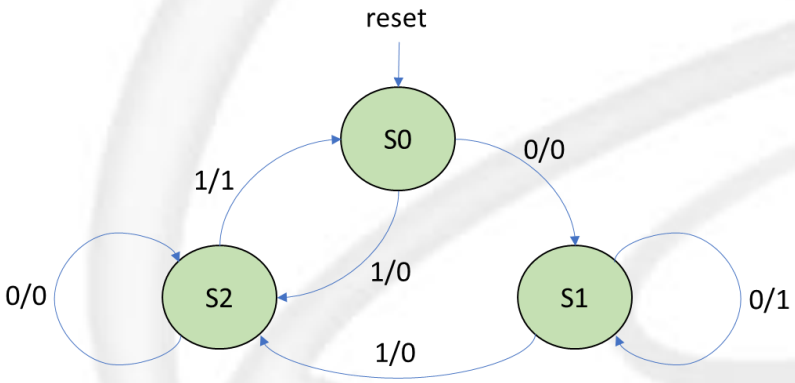
- Upon rst, machine goes back to state S0
- Next state will be determined by current state and input



Current State	Next State		qout
	din=0	din=1	
S0=00	S2	S1	1
S1=01	S1	S0	0
S2=10	S3	S2	0
S3=11	S3	S1	1

Prob B : Mealy Machine

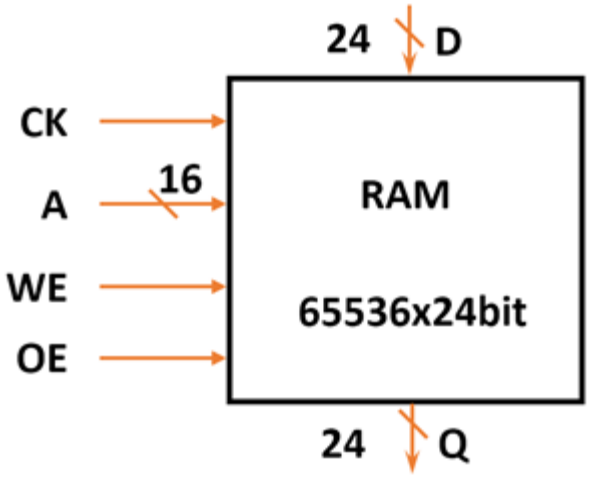
- Upon rst, machine goes back to state S0
- The primary outputs are a function of both the primary inputs and states

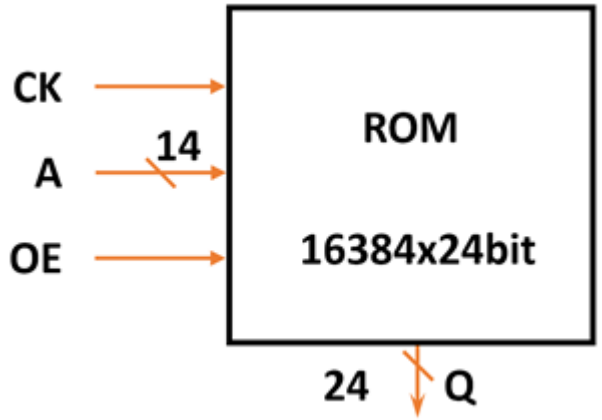


Current State	Next State, output	
	din=0	din=1
S0=00	S1,0	S2,0
S1=01	S1,1	S2,0
S2=11	S2,0	S0,1

Prob C : Memory

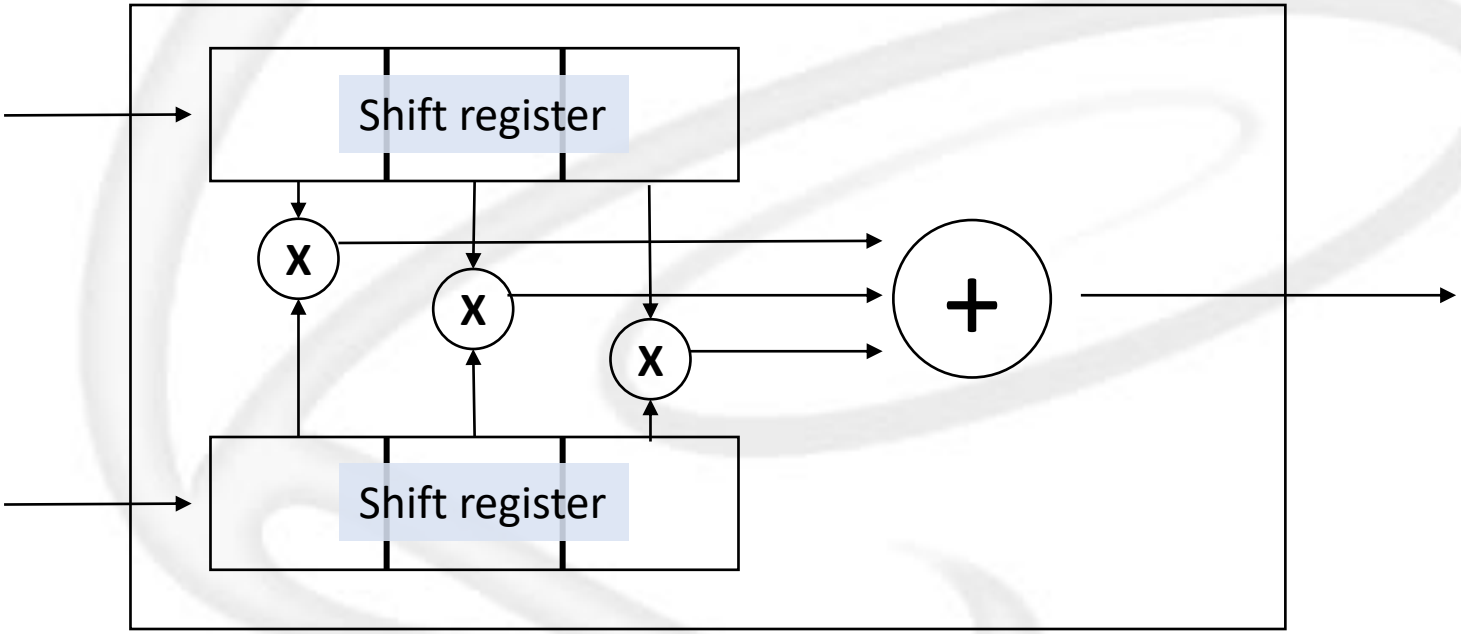
- C-1:
  - Design a 65536x24 bits random-access memory
- C-2:
  - Design a 16384x24 bits read-only memory





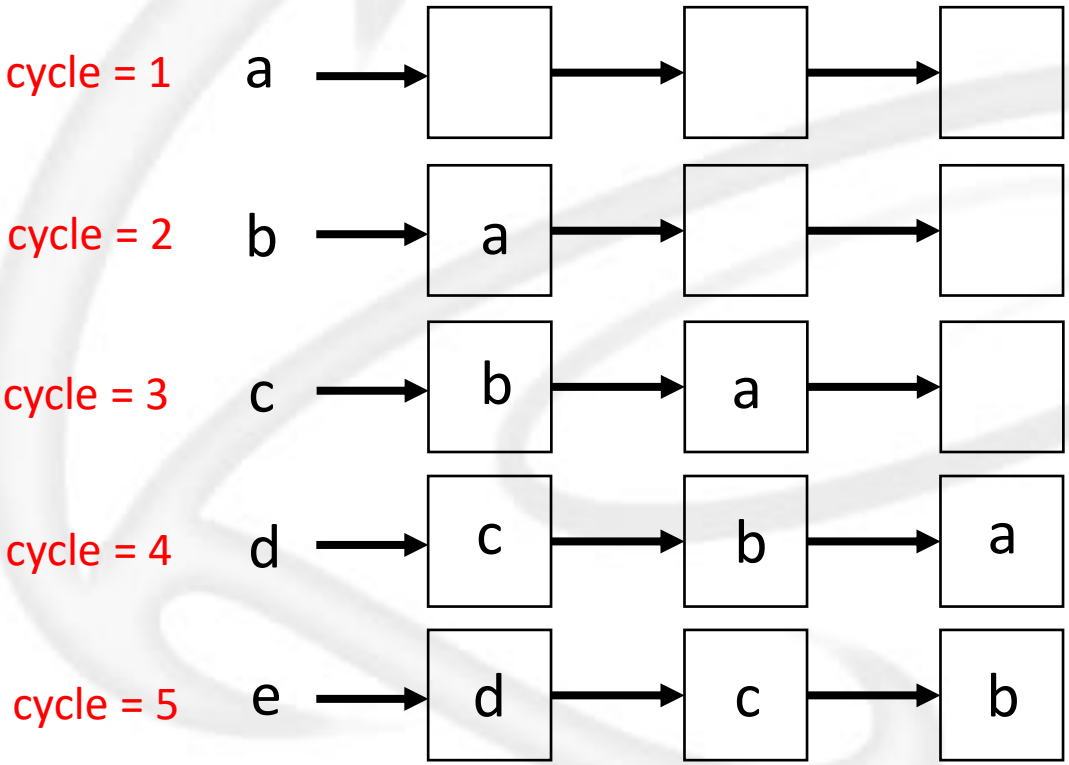
Prob D : MAC using shift register

- Last lab, we need many ports to input data.
- We can design a MAC with shift register to reuse the resource.



Prob D : MAC using shift register

- Shift register is a cascade of flip flops.
- The output of each flip-flop is connected to the input of the next flip-flop.

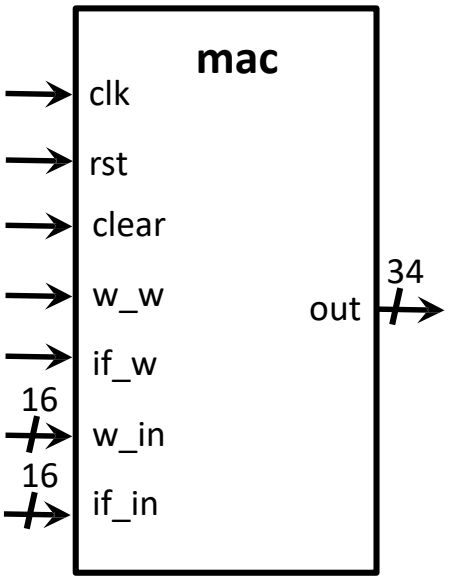


Prob D : MAC using shift register

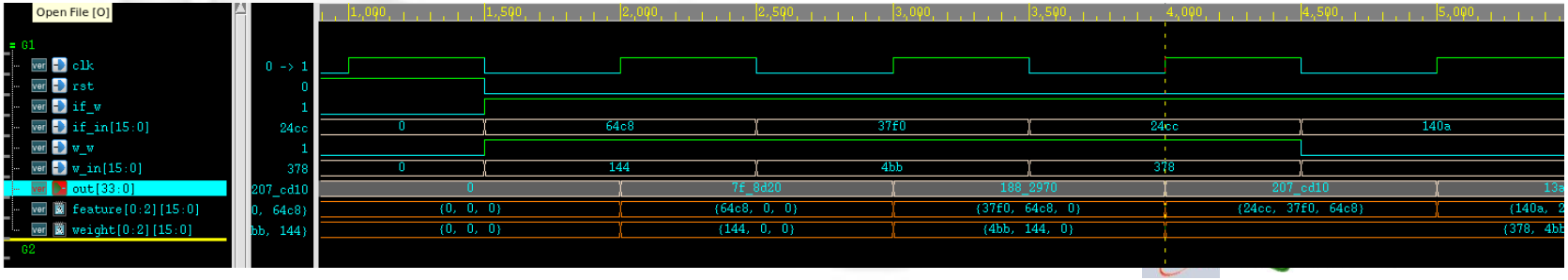
Port list

mac

Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
clear	input	1	Set all register to 0
w_w	input	1	Write weight enable. When w_w is high, write w_in.
if_w	input	1	Write input feature map enable. When if_w is high, write if_in.
w_in	input	16	Input weight data
if_in	input	16	Input feature map data
out	output	34	Output data

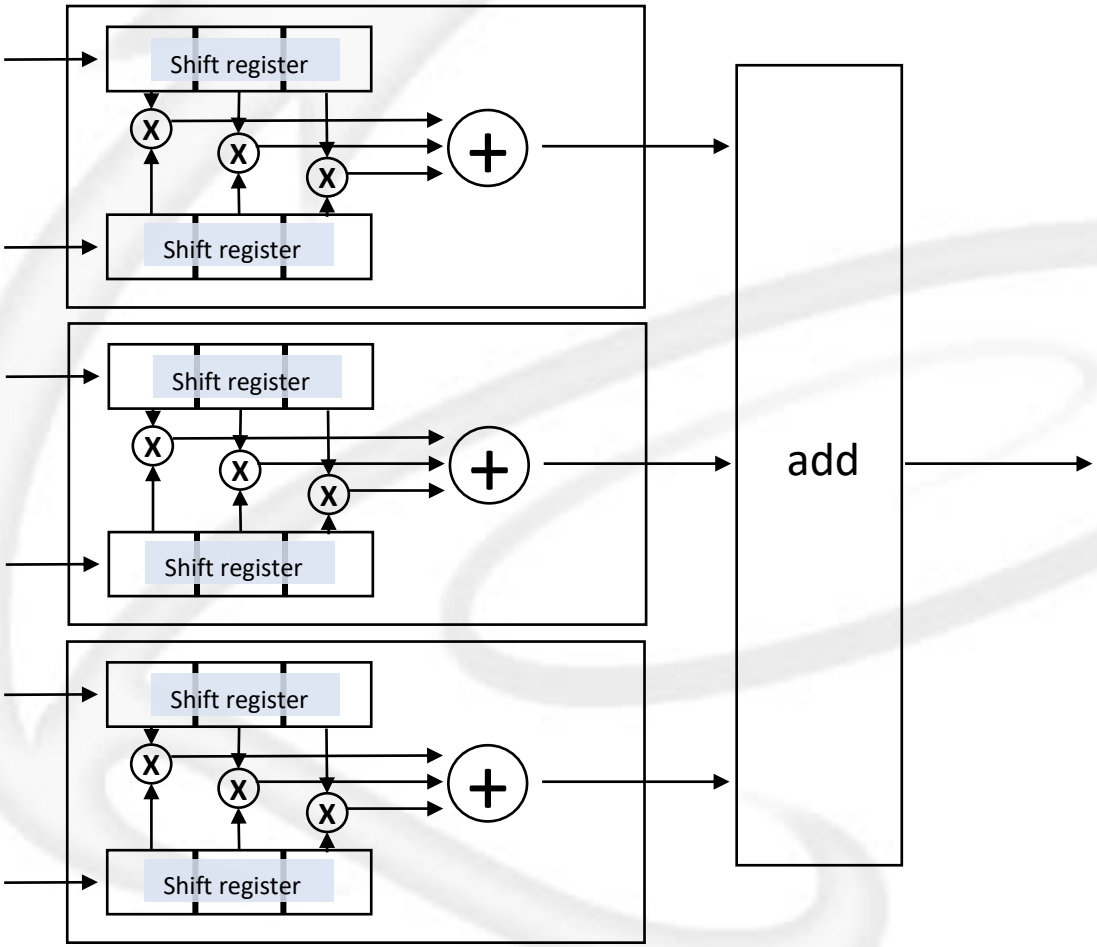


Waveform



Prob D : MAC using shift register

- How complete a 3*3 convolution?



Prob E : A Simple system

- Purpose: Change RGB picture to gray scale picture.

Image Format

- RGB (Red, Green, Blue)
  - Each pixel can be represented in the computer memory or interface as binary values for the red, green, and blue color components.
- Current typical display adapters use 24 bits of information for each pixel.
  - Each color has 8 bits (0-255)
  - Represent as (255, 0, 0)
  - In hexadecimal #FF0000
- Total color
  - **256 * 256 * 256 = 16,777,216**

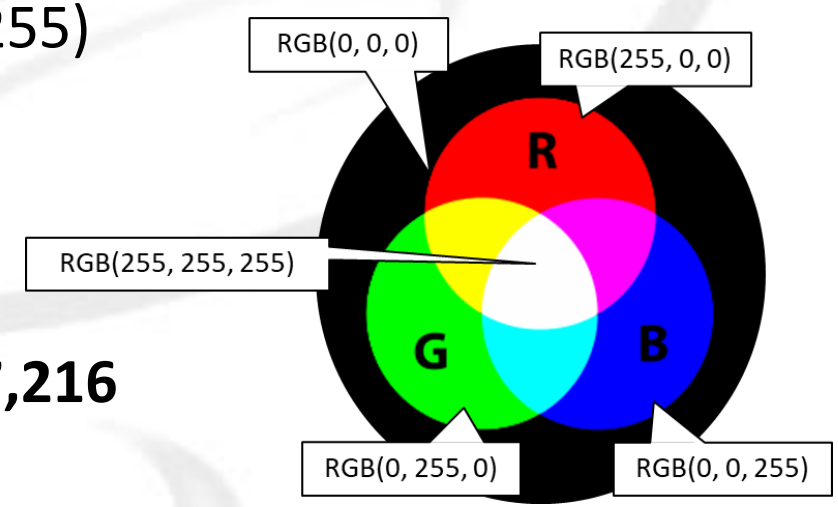


Image Format

- Image decomposed into red, green and blue component.

Red component

Green component

Blue component

Image Format

Bitmap image file (.bmp)

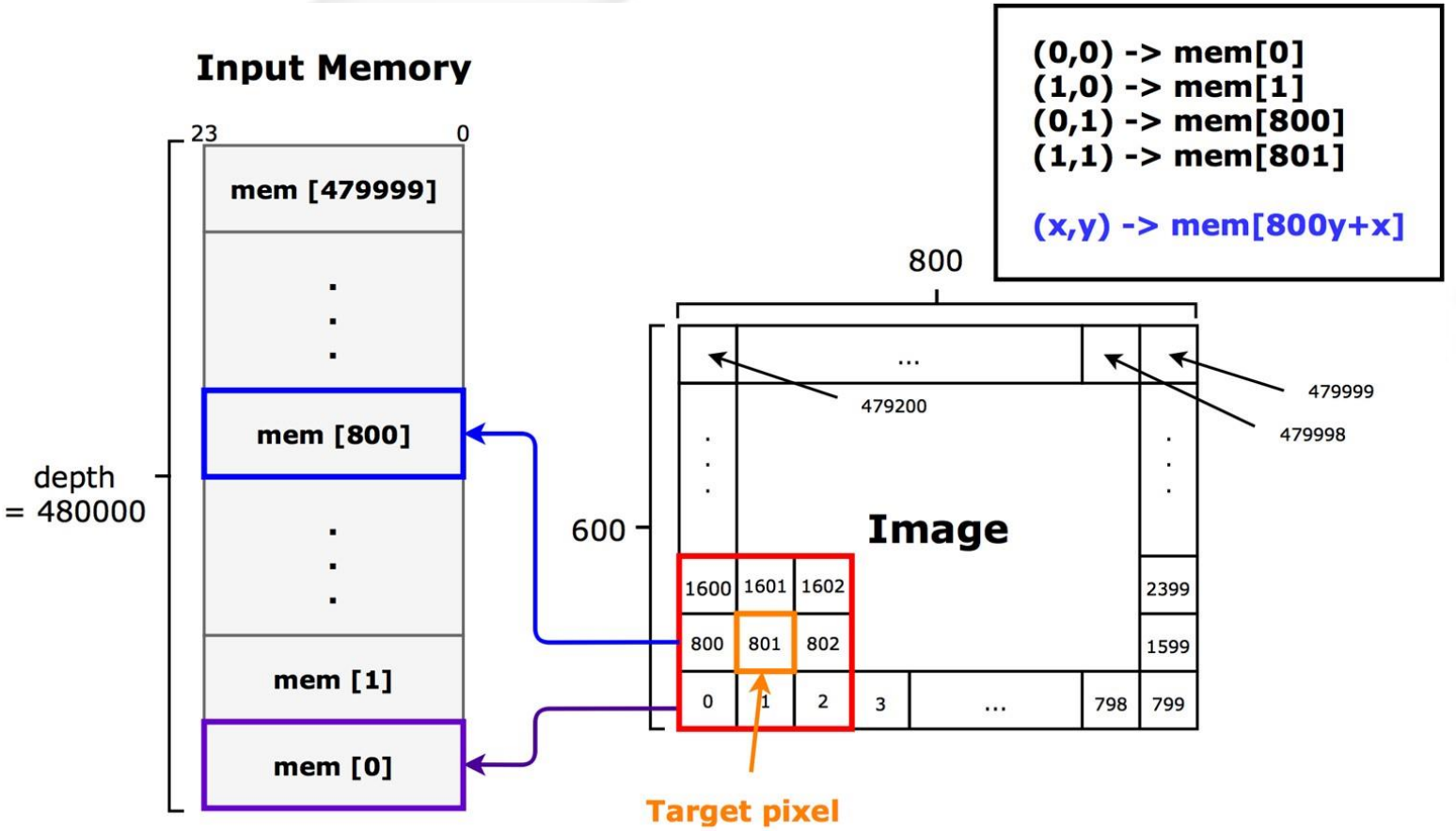
B M Size of BMP file (byte)

The number of bits per pixel

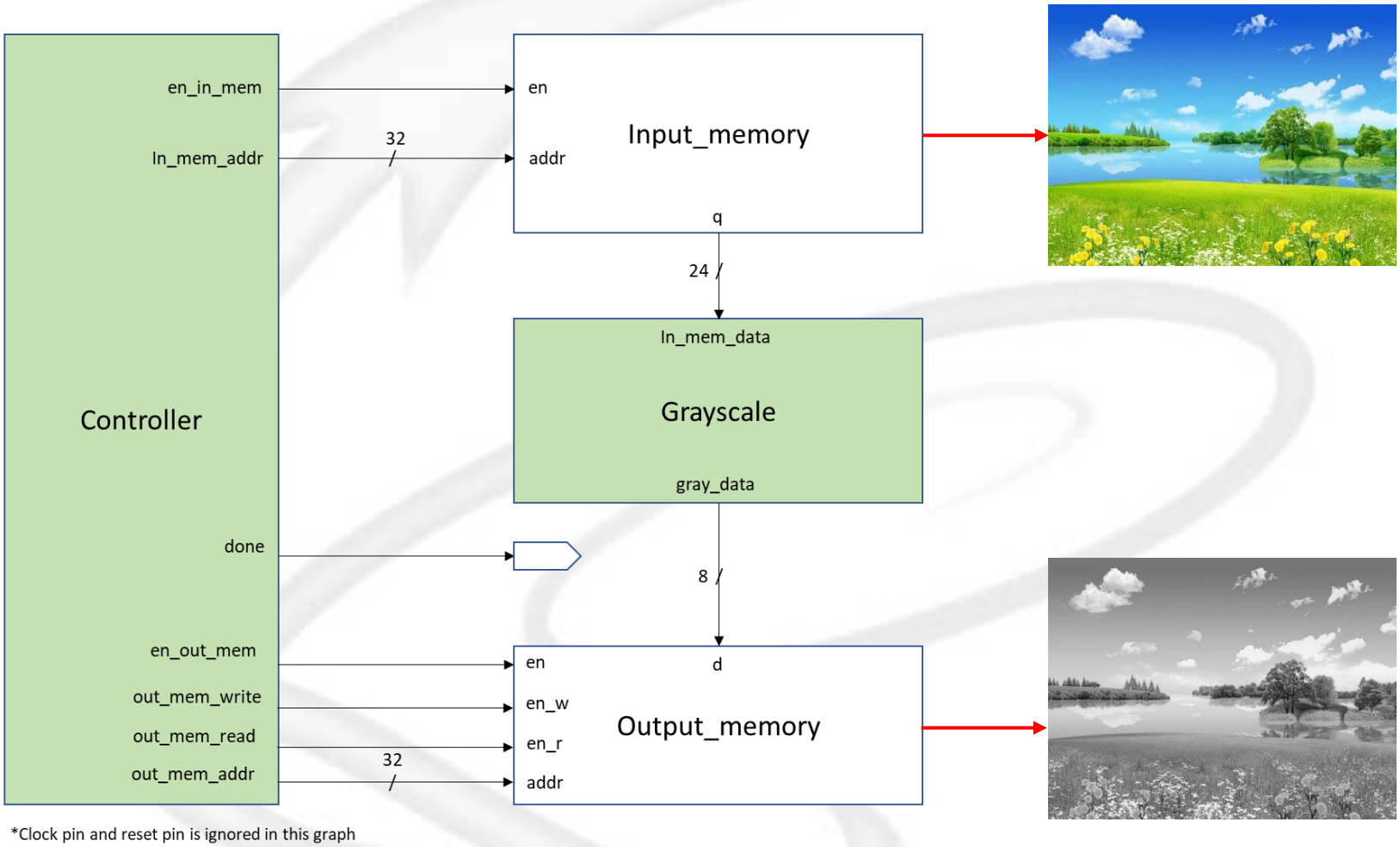
Address	0	1	2	3	4	5	6	7	8	9	a	b	Dump
00000000	42	4d	36	00	24	00	00	00	00	00	36	00	BM6.$.....6.
0000000c	00	00	28	00	00	00	00	04	00	00	00	03	..(.........
00000018	00	00	01	00	18	00	00	00	00	00	00	00	....
00000024	24	00	c4	0e	00	00	c4	0e	00	00	00	00	$.?..?....
00000030	00	00	00	00	00	00	25	1f	12	25	1f	12	......%..%..
0000003c	25	1f	12	25	1f	12	25	1f	12	25	1f	12	%..%..%..%..
00000048	25	1f	12	25	1f	12	25	1f	12	25	1f	12	%..%..%..%..
00000054	25	1f	12	25	1f	12	25	1f	12	25	1f	12	%..%..%..%..
00000060	25	1f	12	25	1f	12	25	1f	12	25	1f	12	%..%..%..%..
0000006c	25	1f	12	25	1f	12	25	1f	12	25	1f	12	%..%..%..%..
00000078	25	1f	12	25	1f	12	25	1f	12	25	1f	12	%..%..%..%..

Image Format

- Image(Here we take 800 * 600 picture)



Architecture



*Clock pin and reset pin is ignored in this graph

Components

- Controller:
  - Control the address, enable signal of memories
  - If the process is finished, make done=1 to terminate simulation. Make sure that your data has store in Output_memory.
- Grayscale:
  - Same as Lab3
  - Change RGB(24bits) to Grayscale(8bits)
- Input Memory:
  - Store pixels of the original image
- Output Memory:
  - Store pixels of the processed image

Controller

Port List

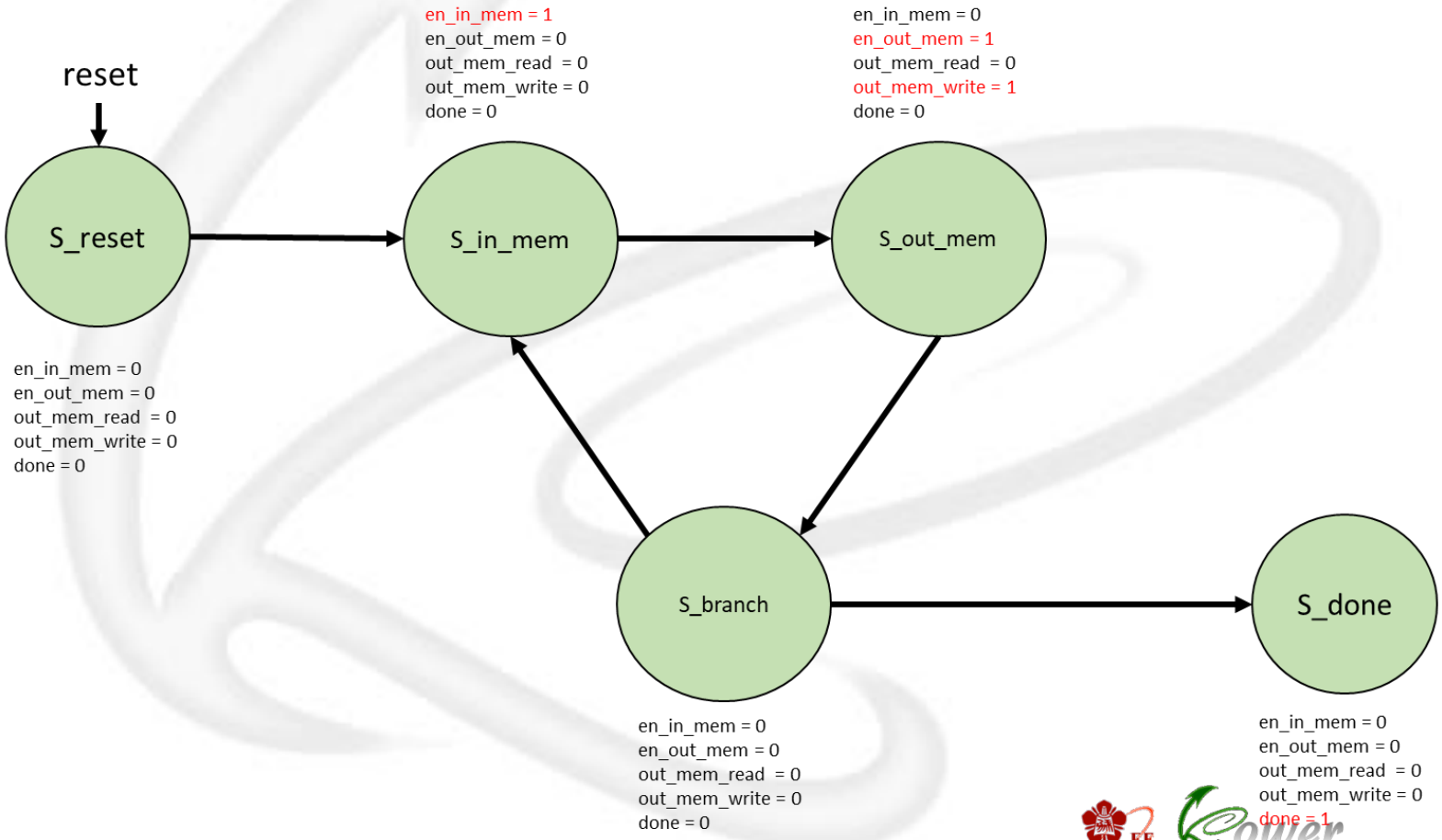
Signal	Type	Bits	Description
clk	input	1	clock
rst	input	1	reset
en_in_mem	output	1	0 → off 1 → on
in_mem_addr	output	32	input memory address
en_out_mem	output	1	0 → off 1 → on
out_mem_read	output	1	0 → off 1 → on
out_mem_write	output	1	0 → off 1 → on
out_mem_addr	output	32	output memory address
done	output	1	0 → uncompleted 1 → completed

※
Output memory read : en_out_mem & out_mem_read are both 1'b1
Output memory write : en_out_mem & out_mem_write are both 1'b1

Controller

State Diagram

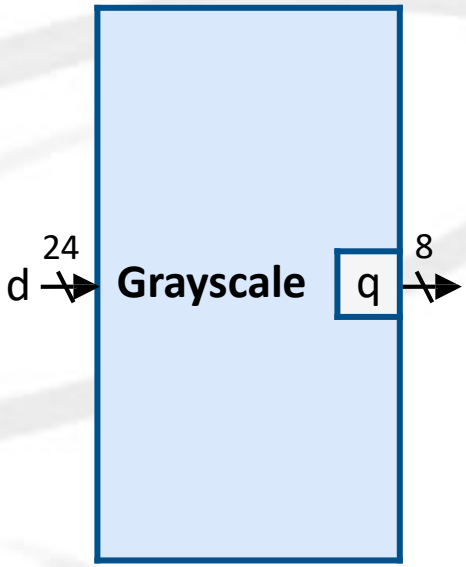
** You can design your own FSM in this lab if you want **



Grayscale

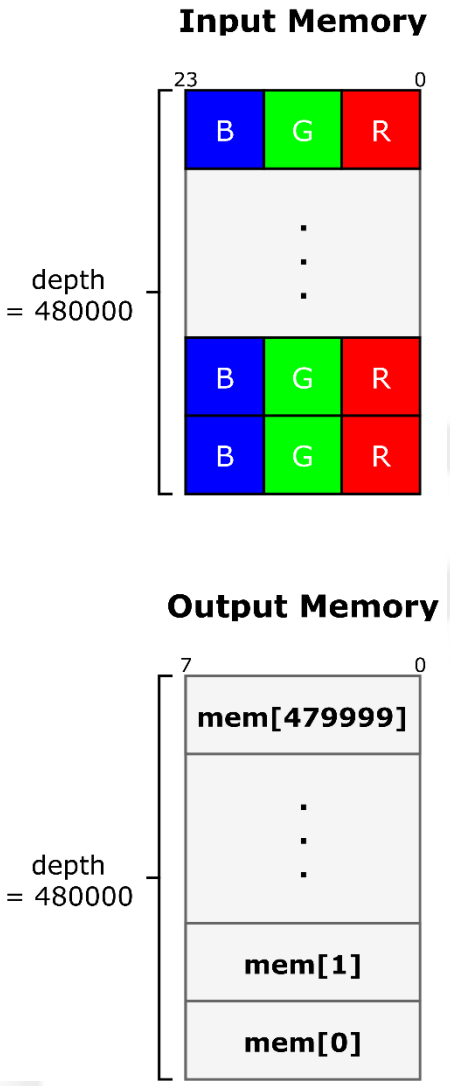
Grayscale

- The grayscale operation y = 0.3125r + 0.5625g + 0.125b (0-255)
- 24-bit input for pixel RGB value
- 8-bit output for pixel grayscale value



Input / Output Memory

- Input Memory
  - → Store pixels of the original image
  - → Memory depth : 800x600=480000
  - → Size per entry : 24-bit (B,G,R)
- Output Memory
  - → Store pixels of the processed image
  - → Memory depth : 800x600=480000
  - → Size per entry : 8-bit



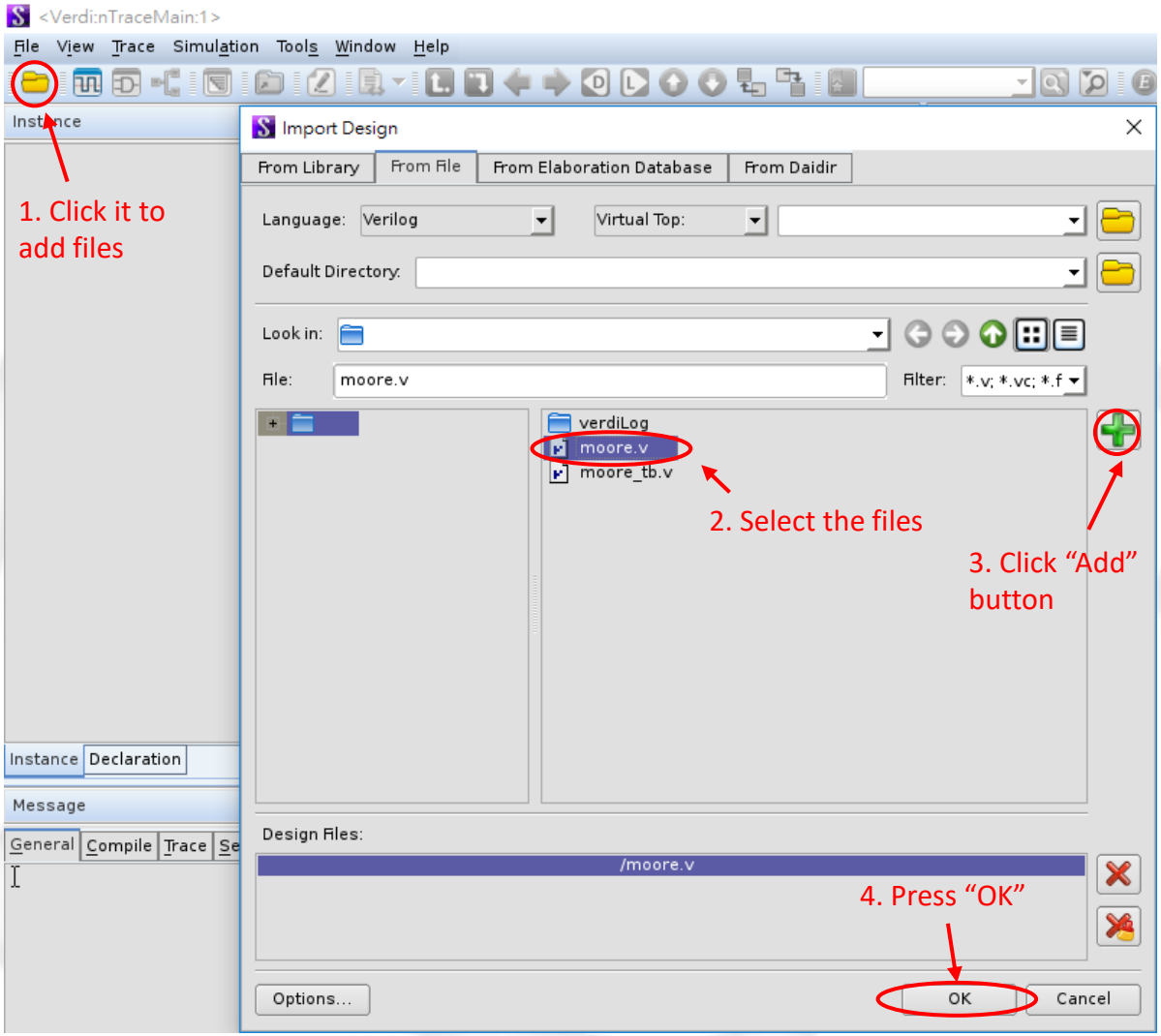
Simulation command

Problem	Command
LabA	ncverilog +access+r moore_tb.v +define+FSDB
LabB	ncverilog +access+r mealy_tb.v +define+FSDB
LabC-1	ncverilog +access+r RAM_tb.v +define+FSDB
LabC-2	ncverilog +access+r ROM_tb.v +define+FSDB
LabD-RTL	ncverilog +access+r mac_tb.v +define+FSDB
LabD-SYN	ncverilog +access+r mac_tb.v +define+FSDB+syn
LabE-RTL	ncverilog +access+r top_tb.v +define+FSDB+picX(X=1,2,3)
LabE-SYN	ncverilog +access+r top_tb.v +define+FSDB+syn+picX(X=1,2,3)

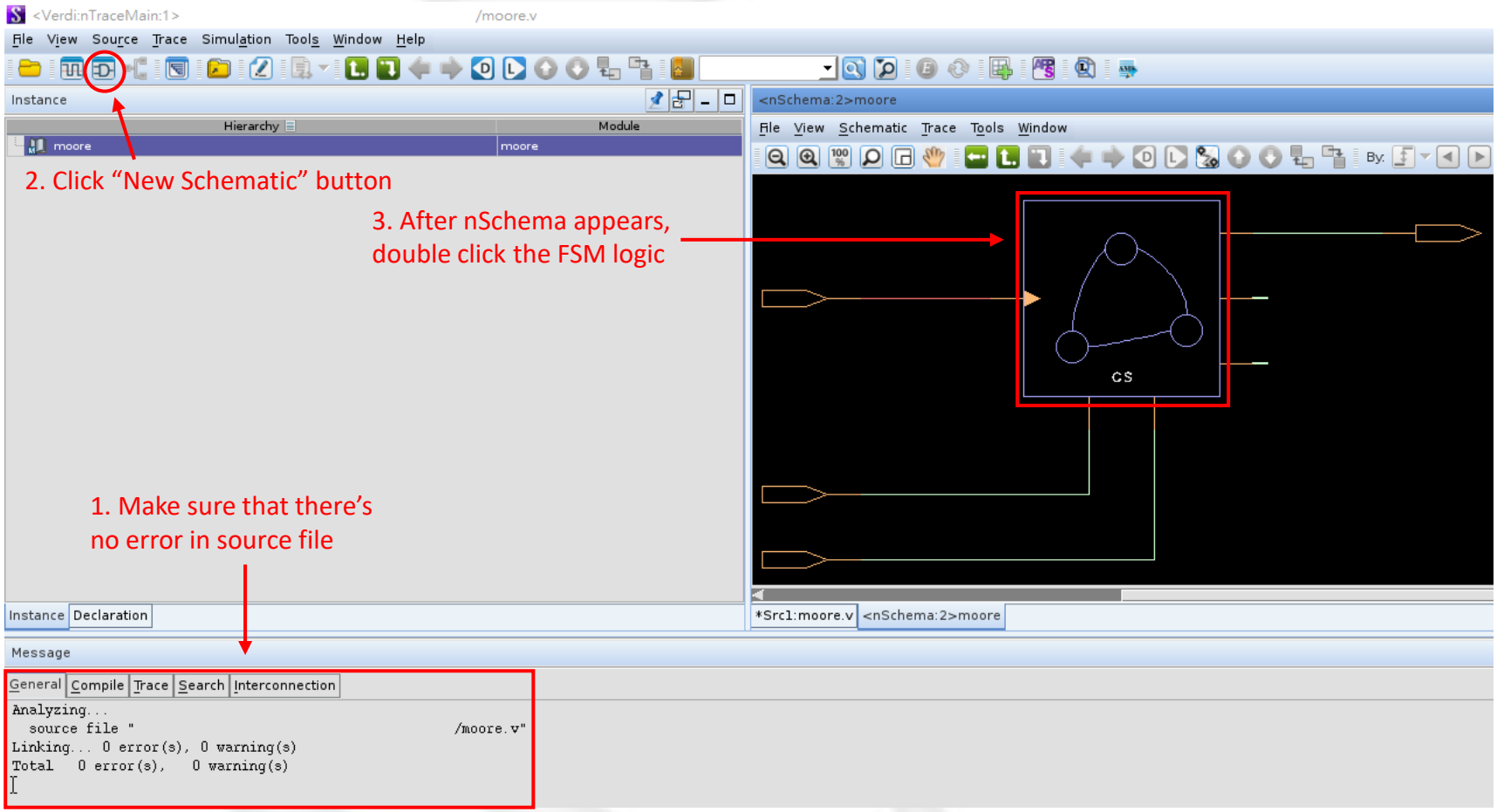
Outline

- FSM Introduction
- Moore Machine
- Mealy Machine
- Random-access Memory
- Read-only Memory
- Homework
- Memo: Use Verdi to See FSM

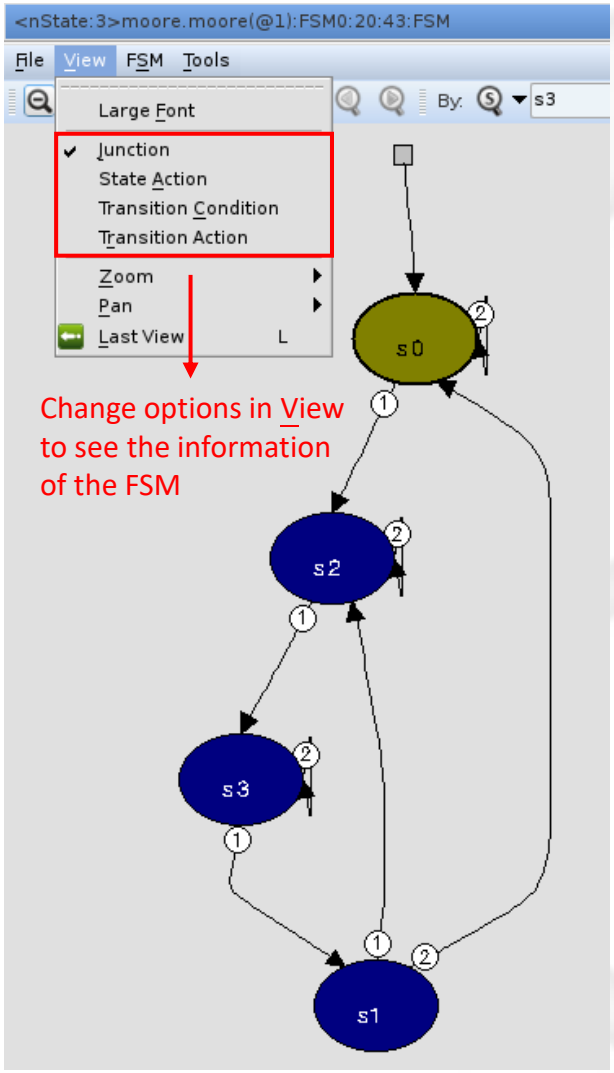
Memo: Use Verdi to See FSM (1/3)

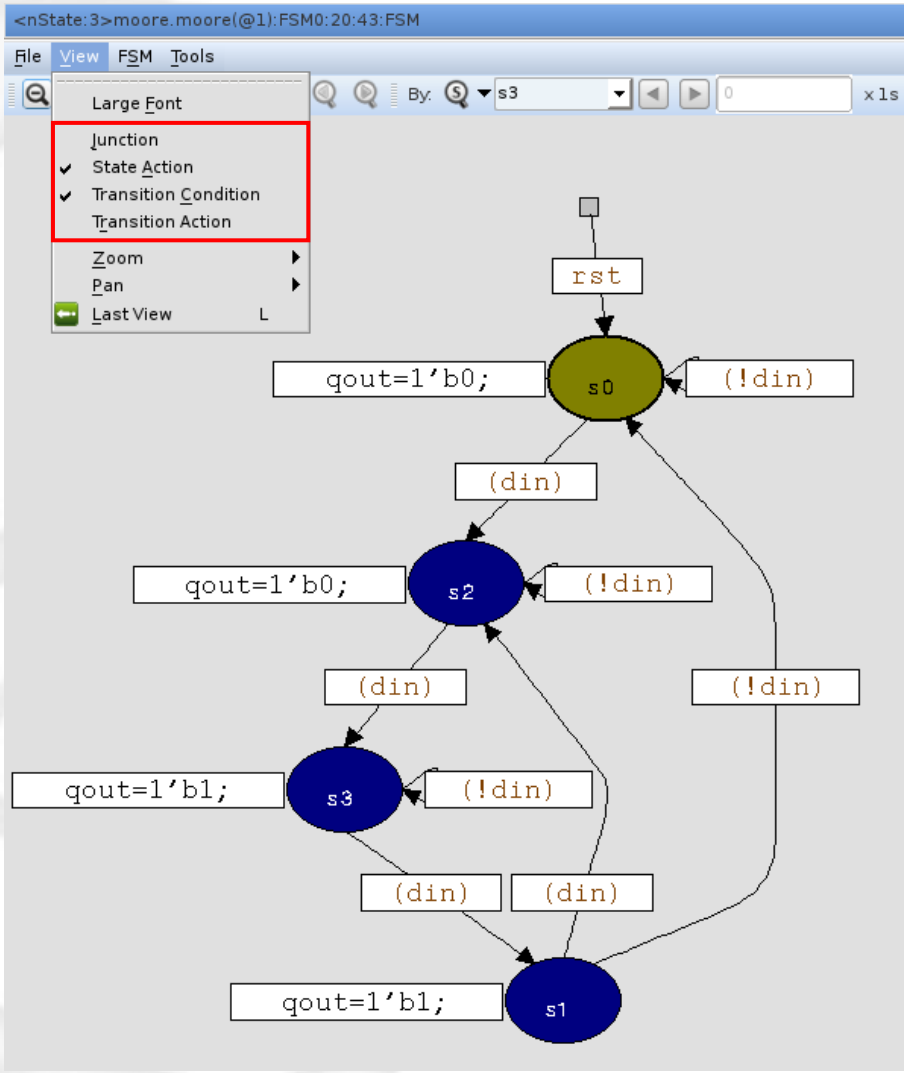


Memo: Use Verdi to See FSM (2/3)



Memo: Use Verdi to See FSM (3/3)





Appendix- Blocking & Non-Blocking

- Recommend :
  - Combinational -> Blocking Assignment
  - Sequential -> Non-Blocking Assignment

```
always @ (*) begin

    A = B;//Blocking assignment
    B = A;//Values of A and B are equal.

end


always @ (posedge clk) begin

    A <= B;//Non-Blocking assignment
    B <= A;//A and B swap with each other.

end
```

Appendix- Full Case

- Use “default case” to fulfill all situations.

```
reg [1:0] cs;//2bit states

always @ (cs) begin

    case (cs)

        2'b00 :...;
        2'b01 :...;
        2'b10 :...;
        2'b11 :...;

    endcase

end
```

```
reg [1:0] cs;//2bit states

always @ (cs) begin

    case (cs)

        2'b00 :...;
        2'b01 :...;

        default:...;

    endcase

end
```