Lab 2
Finite State Machine and CPU Controller

Fall 2008

低功率高效能VLSI設計實驗室
NCKU, Low power, high performance VLSI design lab

Lab Overview

- Target Design
  - A controller for the datapath design in Lab 1 (3), which later could be modified to be the CPU controller
- Finite state machines are briefly reviewed and demonstrated ways to describe in Verilog.
- You then run simulation to test if the given code run as prescribed.
- Using experiences learned in previous steps, design a controller and verify it.
- Detailed description for each step are in Lab2_handout.

OUTLINE

- Introduction
- Moore Machine
- Verilog code
- Testbench
- Mealy Machine
- CPU Controller

Introduction (1/2)

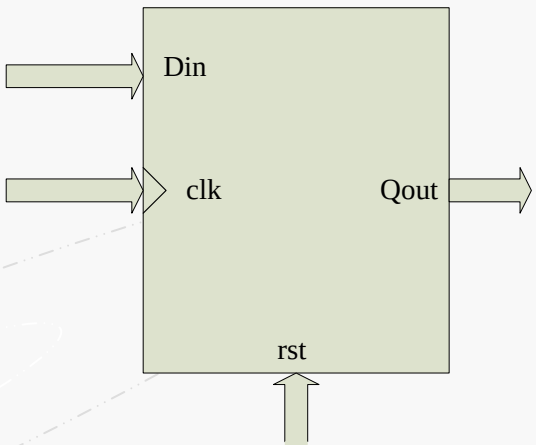
- A finite state machine is generic sequential system that consists both combinational networks and memory elements.
- Memory elements hold the machine's state
- The machine's inputs and outputs are called **primary inputs** and **primary outputs**
- Mealy machine and Moore Machine

Introduction (2/2)

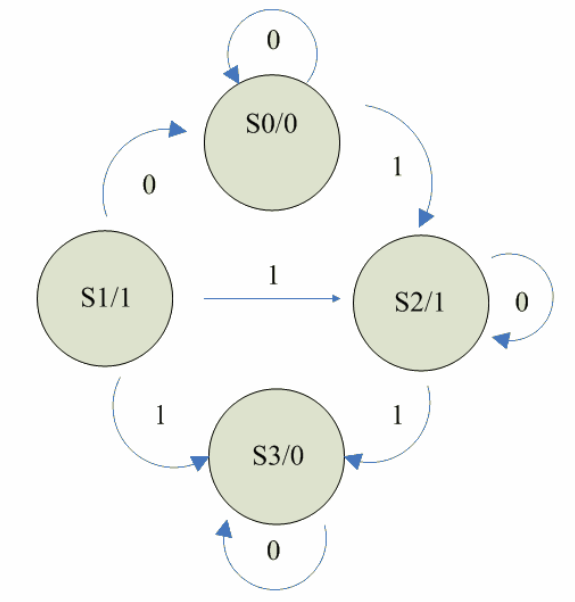
- General FSM design procedure:
  1) Determine the inputs and outputs of the system
  2) System control pins (clk, rst …)
  3) Determine number of possible operating states of the machine
  4) Construct the state diagram
  5) State reductions if possible
  6) Derive the Boolean Equations for each state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

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)



Current State	Next State		Qout
	Din=0	Din=1	
S0=00	S0	S2	0
S1=01	S0	S2	1
S2=10	S2	S3	1
S3=11	S3	S1	0

- Use binary numbers to represent each state

```
parameter [1:0] S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
```

- Reading data input and changes state for every clock cycles
- Upon reset, machine goes back to state S0
- Next state will be determined by current state and input

Verilog Code (1/3)

```
`timescale 1ns/10ps
module moore (Qout, clk, rst, Din);
output Qout;
input clk, rst, Din;
parameter [1:0] S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
reg Qout;
reg [1:0] CS, NS;

always @ (posedge clk or posedge rst)
begin
if (rst==1'b1) CS=S0;
else CS = NS;
end
```

Verilog Code (2/3)

```
always @ (CS or Din)
begin
case (CS)
S0:     begin
        Qout=1'b0;
        if (Din==1'b0) NS=S0;
        else NS = S2;
        end
S1:     begin
        Qout=1'b1;
        if (Din==1'b0) NS=S0;
        else NS = S2;
        end
```

Verilog Code (3/3)

```
S2:         begin
            Qout=1'b1;
            if (Din==1'b0) NS=S2;
            else NS = S3;
            end
S3:         begin
            Qout=1'b0;
            if (Din==1'b0) NS=S3;
            else NS = S1;
            end
endcase
end
endmodule
```

Testbench

```
`timescale 1ns/10ps
module moore_tb;
reg clk, rst, Din;    //inputs
wire Qout; //outputs
moore m0 (.Qout(Qout), .clk(clk), .rst(rst), .Din(Din));
initial $monitor($time, " clk=%d, rst=%d, Din=%d, Qout=%d", clk, rst, Din, Qout);
initial clk=1'b0;
always #10 clk=~clk;
initial begin     rst=1;
#20       rst=0; Din=0;
#20       Din=1;
#20       Din=0;
#20       Din=1;
#20       Din=0;
#20       Din=1;
#20       Din=0;
end
initial begin
$dumpfile("moore.vcd");
$dumpvars;
#200 $finish;
end
endmodule
```

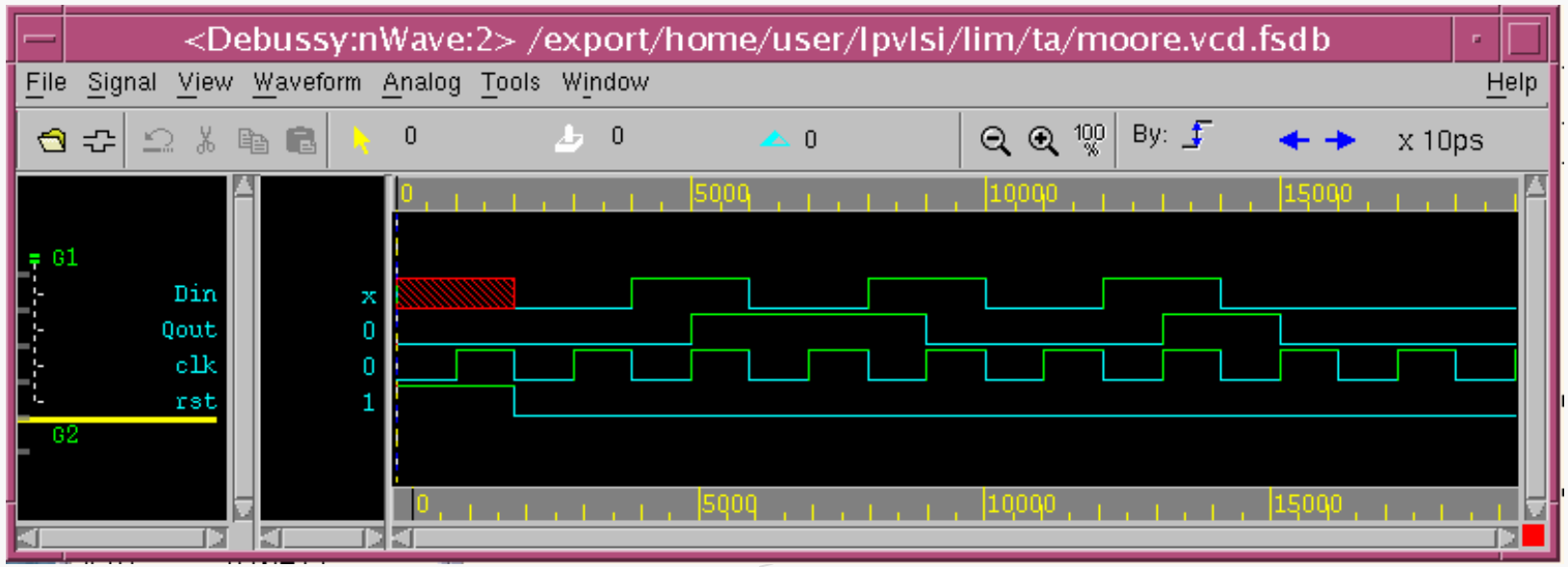
Simulation Results (1/2)

Terminal

Window Edit Options Help

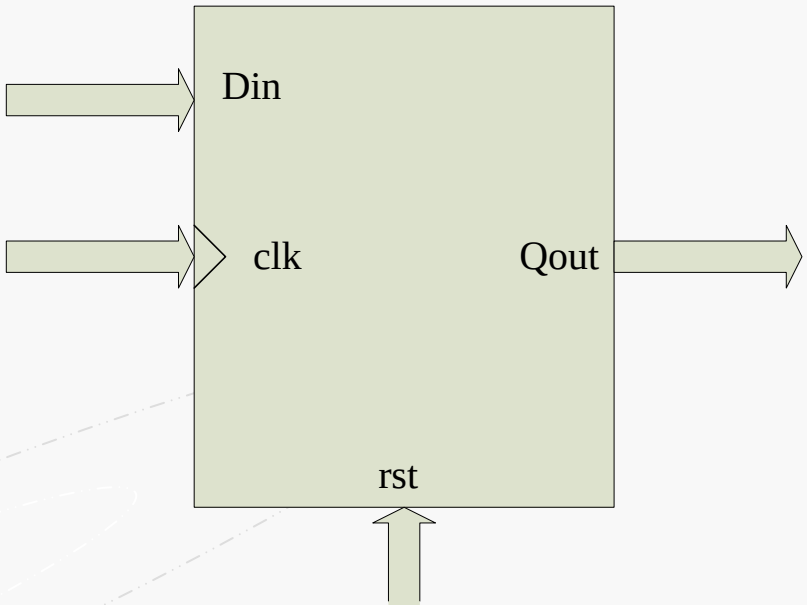
```
CADENCE DESIGN SYSTEMS, INC.
RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to
restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause at DFARS 252.227-7013 or
subparagraphs (c)(1) and (2) of Commercial Computer Software -- Restricte
d
Rights at 48 CFR 52.227-19, as applicable.

                Cadence Design Systems, Inc.
                555 River Oaks Parkway
                San Jose, California  95134

For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com

For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com

Compiling source file "moore.v"
Compiling source file "tb_moore.v"
Highest level modules:
moore_tb

                  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=1
                 60 clk=0, rst=0, Din=0, Qout=1
                 70 clk=1, rst=0, Din=0, Qout=1
                 80 clk=0, rst=0, Din=1, Qout=1
                 90 clk=1, rst=0, Din=1, Qout=0
                100 clk=0, rst=0, Din=0, Qout=0
                110 clk=1, rst=0, Din=0, Qout=0
                120 clk=0, rst=0, Din=1, Qout=0
                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
                160 clk=0, rst=0, Din=0, Qout=0
                170 clk=1, rst=0, Din=0, Qout=0
                180 clk=0, rst=0, Din=0, Qout=0
                190 clk=1, rst=0, Din=0, Qout=0
L25 "tb_moore.v": $finish at simulation time 20000
0 simulation events (use +profile or +listcounts option to count)
CPU time: 0.0 secs to compile + 0.0 secs to link + 0.0 secs in simulation
End of Tool:    VERILOG-XL      05.30.004-s   Apr 10, 2007  21:57:59
/export/home/user/lpvlsi/lim//ta>
```

Simulation Results (2/2)



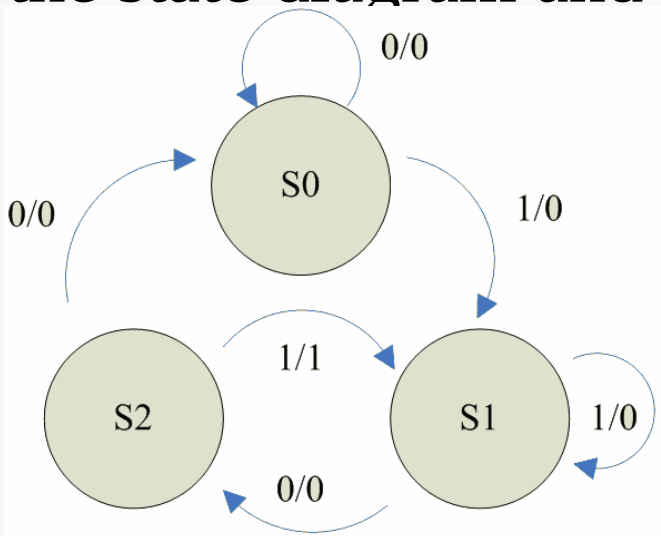
Mealy Machine (1/3)

- The primary outputs are a function of both the primary inputs and state
- System detecting sequence of 101 from input data



Mealy Machine (2/3)

- Construct the state diagram and state table as below:



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

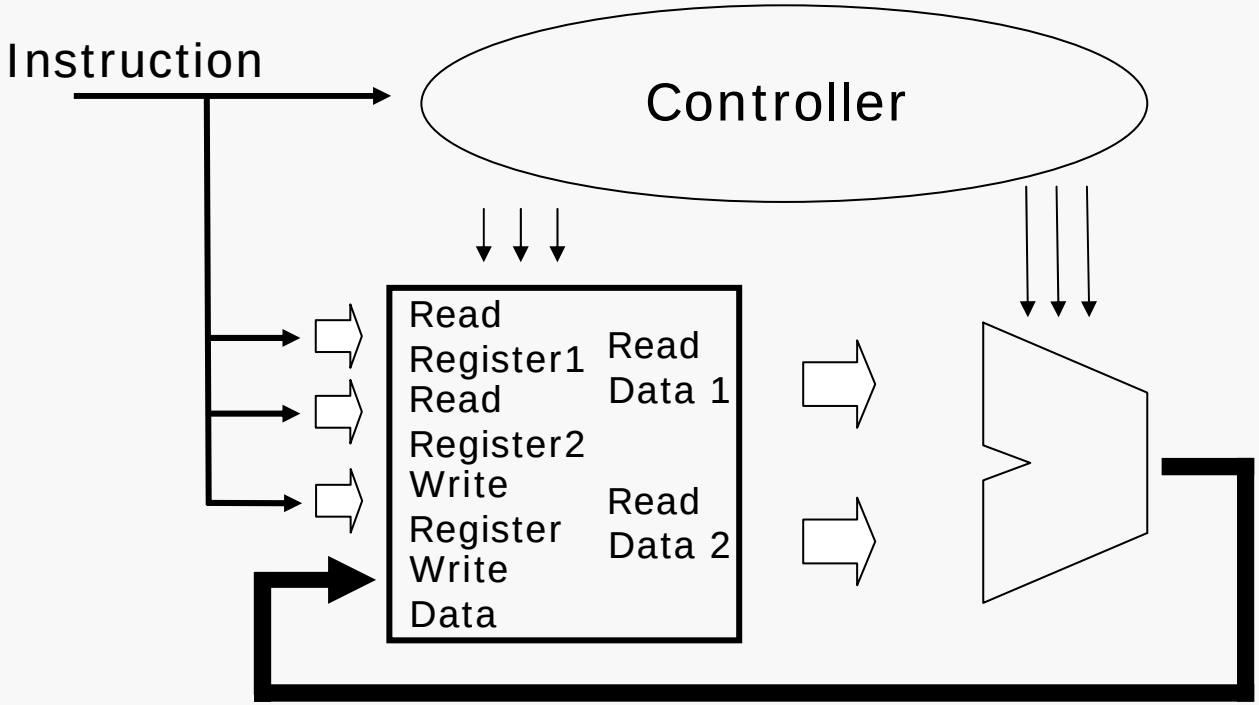
Mealy Machine (3/3)

- The Verilog Code for the Mealy machine is as shown below
- Please complete the Verilog Code in Lab Session

```
`timescale 1ns/10ps
module mealy (Qout, clk, rst, Din);
output Qout;
input clk, rst, Din;
parameter [1:0] S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;
reg Qout;
reg [1:0] CS, NS;

always @ (posedge clk or posedge rst)
begin
if (rst==1'b1) CS=S0;
else CS = NS;
end

always @ (CS or Din)
begin
case (CS)
S0:                     begin
                        if (Din==1'b0)
                        begin
                        NS=S0;
                        Qout=1'b0;
                        end
                        else
                        begin
                        NS = S2;
                         Qout=1'b0;
                        end
```

```
S1:     begin
          .
          .
          .
        end
S2:     begin
          .
          .
          .
        end
S3:     begin
          .
          .
          .
        end
endcase
end
endmodule
```

CPU Controller (1/5)



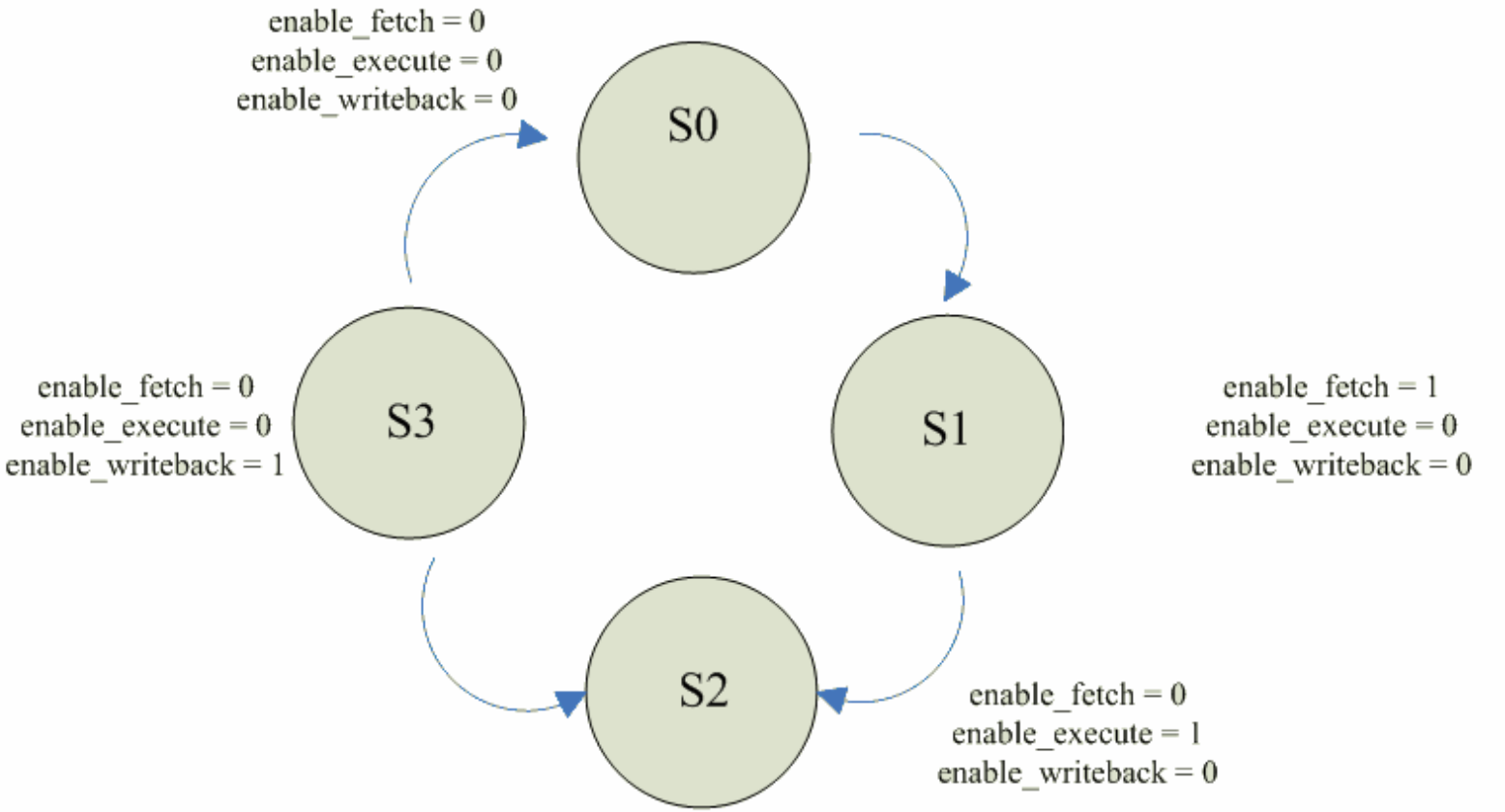
- A controller unit provides signals that activate the various micro-operations within the datapath to perform the specified processing tasks
- Determines the sequence in which the various actions are performed
- Sequential circuit with states that dictates the control signals for system

CPU Controller (2/5)

- All operations initiated at the positive edge trigger
- Control signals:
  - enable_fetch: To fetch and store the designated data from instruction memory into the register file and prepare them to be ready for execution
  - enable_execute: To order ALU to execute the operation
  - enable_writeback: To store the ALU result back to the register file or data memory
- 4 different operating states for the controller, i.e. stop operation, fetch, execute and write-back

CPU Controller (3/5)

- State diagram of the CPU controller



CPU Controller (5/5)

```
module ctrl_state(clk, reset, enable_fetch,
enable_execute, enable_writeback, ir, PC_adjust, pc);
`define OPCODE  ir[31:28]
`define SRCTYPE ir[27]    //1= imm; 0 = reg
`define DSTTYPE ir[26]    //1 = Data Mem; 0 = reg
`define SRC     ir[25:13]
`define DST     ir[12:0]
input [31:0] ir;
input [4:0]  pc;
input clk,reset;
output enable_fetch, enable_execute;
output enable_writeback, PC_adjust;
reg  enable_fetch, enable_execute;
reg  enable_writeback, PC_adjust;
reg  [1:0] current_state, next_state;
reg  [31:0] pre_opcode;

parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;

always @(posedge clk)
begin
  if(reset) begin
    current_state=S0;
  end
  else begin
    current_state=next_state;
  end
end
```

```
always @(current_state)
begin
  case(current_state)
    2'b00: begin
          next_state=S1;
          enable_fetch=0;
          enable_execute=0;
          enable_writeback=0;
        end
              .
              .
              .
    default: next_state=S0; //avoid unknown state
  endcase
end

always @(posedge enable_fetch)
begin
  if(pc==0) pre_opcode=0;
  else pre_opcode=ir;
end
```

Program Counter (1/2)

- Controller contains a PC and the associated decision logic to interpret the instruction in order to execute it
- Program Counter (PC) is a register storing the memory address of the instruction to be executed.
- PC controls the sequence of operations using decision based on status information from the datapath using it’s counting capability

Program Counter (2/2)

```
module PCounter ( clk, reset, pc);
input clk, reset;
output [4:0]pc;
reg [4:0] pc;

always @(posedge clk)
begin
   if (reset) pc=0;
   else
     begin
        pc = pc + 1 ;
        if (pc==5'b11111) pc=0;
     end
end
endmodule
```