BROWN
School of Engineering

DIGITAL ELECTRONICS SYSTEM DESIGN

FALL 2019

PROFS. IRIS BAHAR & ROD BERESFORD

OCTOBER 23, 2019

LECTURE 14: STATE MACHINE DESIGN

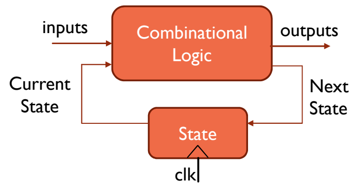
VERILOG TUTORIAL (PART 2)

- McKenna will again give a Verilog tutorial during his lab hours this Wednesday from 4:30-6:30
- He will go over state machine design with Verilog as well as procedures to set up a simulation with Verilog

MIDTERM STUDY SESSION

- Jiwon will give a help session on Monday, Oct. 28 from 7-9pm in B&H190 to prepare for the midterm exam next week.
- Come with questions. She will also solve some problems (from the problem sets) on the board.
- Solutions to practice problem set #1 will be available later tonight
- She will hold office hours in B&H 196 this week and next:
  - Saturday, Oct. 26 : 10am-noon
  - Sunday, Oct. 27: 7-9pm
  - Tuesday, Oct 29: 7-9pm

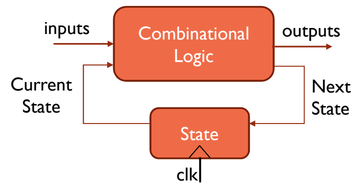
FINITE STATE MACHINE



- A Finite State Machine (FSM) is an abstract representation of a sequential circuit
  - The state embodies the condition of the system at this particular time
  - The combinational logic determines the output and next state values
  - The output values may depend only on the current state value, or on the current state and input values

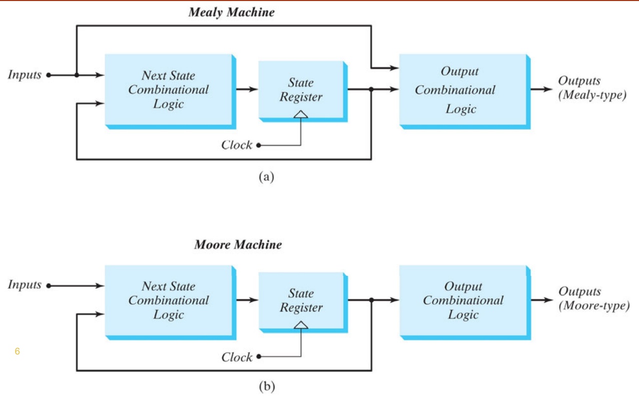
ELEMENTS OF AN FSM

- 4 properties of an FSM
  1. A finite number of inputs
  2. A finite number of outputs
  3. A finite number of states
  4. A specification of all state transitions



- Can be described by a state diagram

BLOCK DIAGRAMS OF MEALY AND MOORE STATE MACHINES

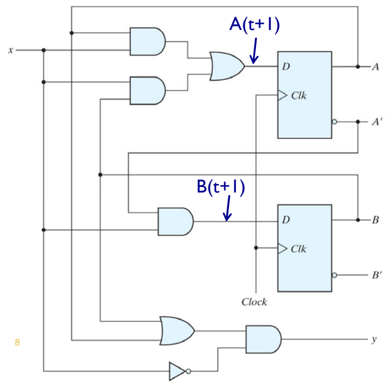


6

STATE MACHINE ANALYSIS PROCEDURE

1. Given a circuit diagram for a sequential circuit
2. Derive expressions for FF inputs (or state equations for each FF)
3. Derive an equation for each output as a function of the present state (and the inputs - Mealy only)
4. Set up a state table
   - Present state, inputs, next state, output
   - Optional intermediate columns for FF inputs
5. Draw the state diagram

EXAMPLE OF SEQUENTIAL CIRCUIT



Is this a Moore or Mealy machine?

FF input equations?
Output equation?

8

STATE TABLE FOR CIRCUIT

State equations
Output equation

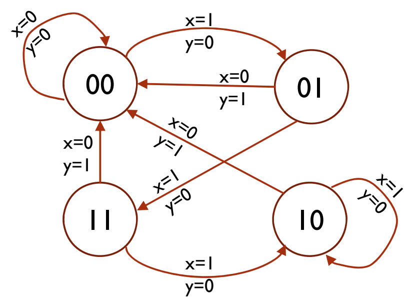
Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

ALTERNATIVE FORM FOR THE STATE TABLE

State diagram?

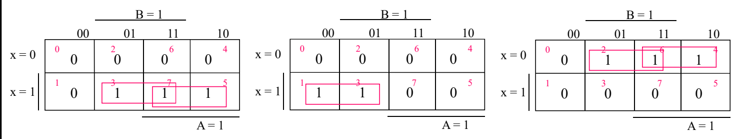
Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	*B*	*A*	*B*	*A*	*B*	*y*	*y*
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

STATE DIAGRAM



- A, B are encoded into 4 states

BOOLEAN EXPRESSION FOR FSM



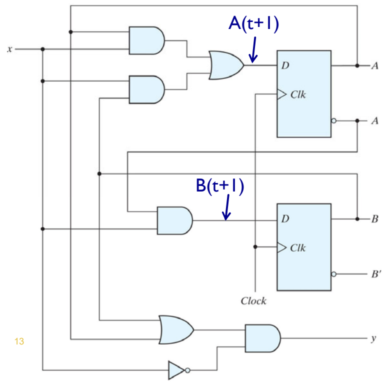
A_{next} = xB+xA
= x(B + A)

B_{next} = A'x

y = x'B+x'A
= x'(B + A)

- Express outputs from truth table

EXAMPLE OF SEQUENTIAL CIRCUIT



Is this a Moore or Mealy machine?

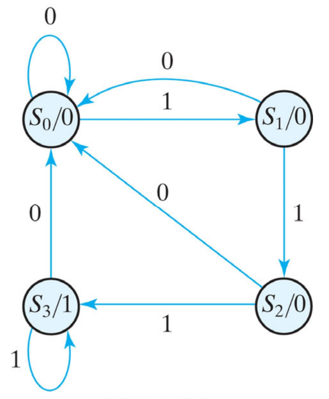
FF input equations? Output equation?

13

STATE MACHINE DESIGN PROCEDURE

1. Define the task in words (Mealy or Moore?)
2. Draw a state diagram
3. Assign state values to the states (number the states)
4. Minimize the number of states in the state table/diagram
5. Set up a state table
6. Select a flip-flop type and set up an excitation table
7. Use the excitation table to generate columns in the state table for the FF inputs
8. Design the combinational circuits
9. Draw the logic diagram and build the circuit

EXAMPLE 2: STATE DIAGRAM FOR SEQUENCE DETECTOR



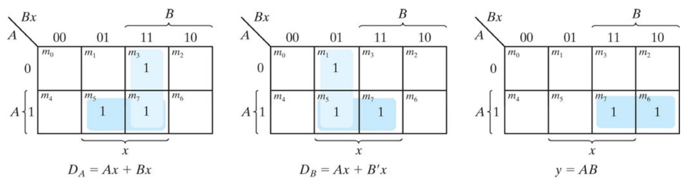
- Make a machine that sets an output signal to 1 when the input signal is 1 for 3 or more times in a row
- State diagram to detect 3 ones in a row
- *Is this a Mealy or Moore machine?*

STATE TABLE FOR SEQUENCE DETECTOR: MOORE MACHINE

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

16

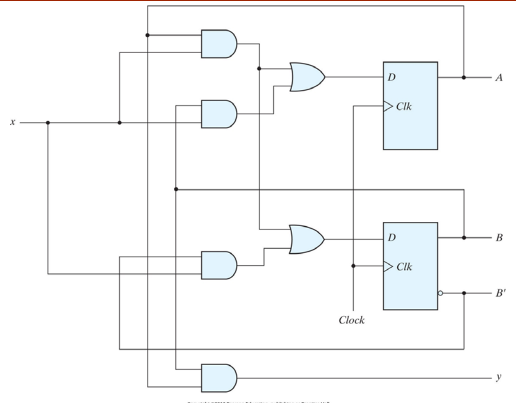
K-MAPS FOR SEQUENCE DETECTOR USING D-FFS



$D_A = Ax + Bx$ $D_B = Ax + B'x$ $y = AB$

- Each output is represented with a separate Karnaugh map

LOGIC DIAGRAM OF A MOORE-TYPE SEQUENCE DETECTOR



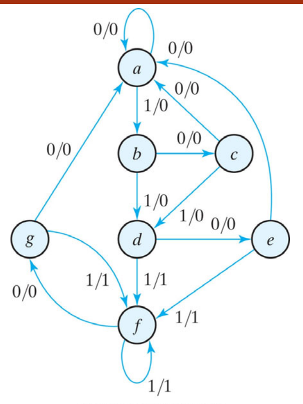
STATE ASSIGNMENT

- *How many states do I need?*
- *How many bits do I need to represent each state?*

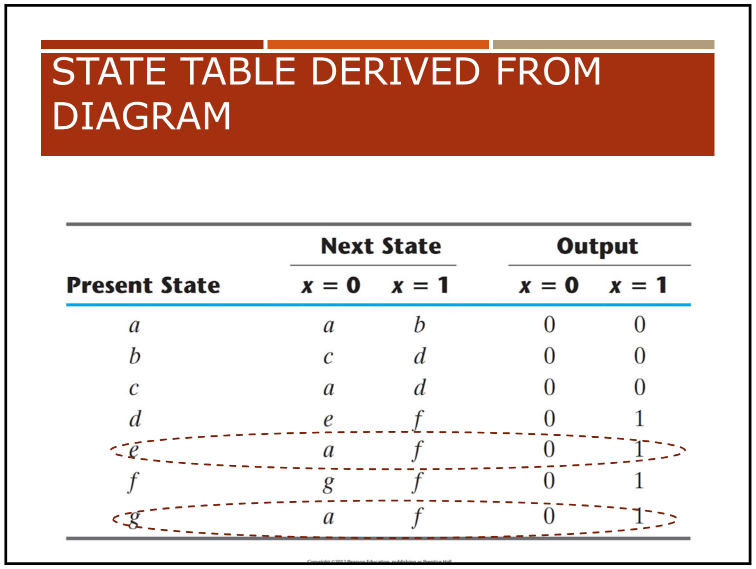
STATE REDUCTION

- Two states are the same if:
  1. They produce the same outputs for the same inputs
  2. They go to the same (or equivalent) next states for all inputs

EXAMPLE 3: STATE DIAGRAM

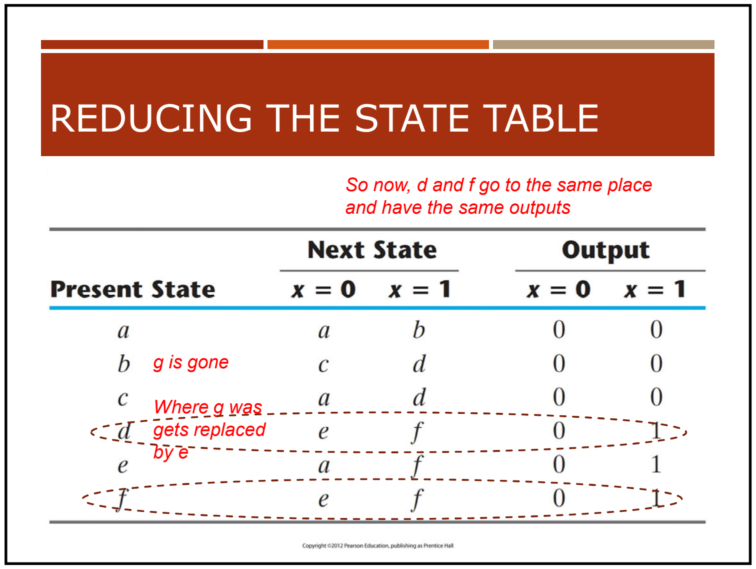


21



STATE TABLE DERIVED FROM DIAGRAM

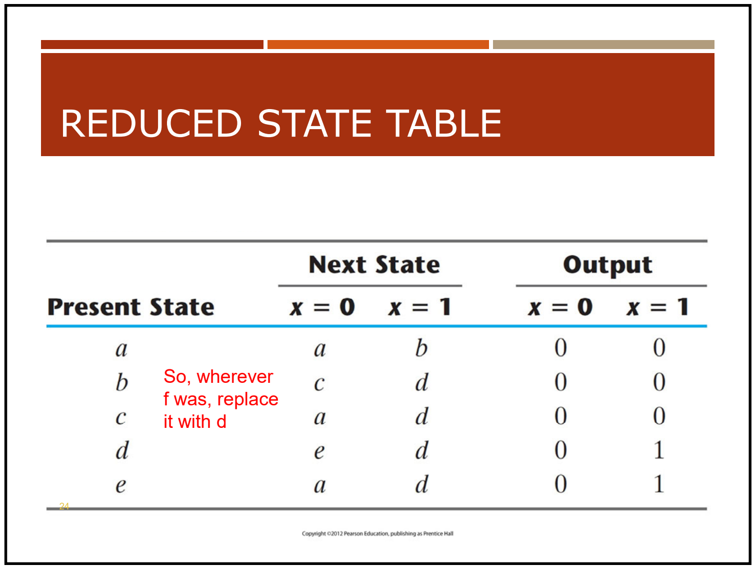
	Next State		Output	
Present State	**x = 0**	**x = 1**	**x = 0**	**x = 1**
a	*a*	*b*	0	0
b	*c*	*d*	0	0
c	*a*	*d*	0	0
d	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*g*	*f*	0	1
g	*a*	*f*	0	1



REDUCING THE STATE TABLE

So now, d and f go to the same place and have the same outputs

	Next State		Output	
Present State	**x = 0**	**x = 1**	**x = 0**	**x = 1**
a	*a*	*b*	0	0
b g is gone	*c*	*d*	0	0
c Where g was	*a*	*d*	0	0
d gets replaced by e	*e*	*f*	0	1
e	*a*	*f*	0	1
f	*e*	*f*	0	1

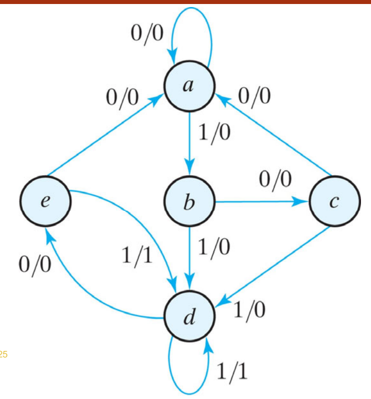


REDUCED STATE TABLE

	Next State		Output	
Present State	**x = 0**	**x = 1**	**x = 0**	**x = 1**
a	*a*	*b*	0	0
b So, wherever	*c*	*d*	0	0
c f was, replace it with d	*a*	*d*	0	0
d	*e*	*d*	0	1
e	*a*	*d*	0	1

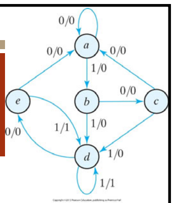
24

REDUCED STATE DIAGRAM



Now, you're stuck. It still takes 3 bits in binary to get beyond four states

THREE POSSIBLE BINARY STATE ASSIGNMENTS



State	Assignment 1, Binary	Assignment 2, Gray Code	Assignment 3, One-Hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000
	3 bits	3 bits	5 bits

REDUCED STATE TABLE WITH BINARY ASSIGNMENT 1

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

K-MAP FOR OUTPUTS (S_2, S_1, S_0, Y)

S_1 S_2

x S_0

	00	01	11	10
00	0	0	-	0
01	0	0	-	0
11	0	0	-	-
10	0	1	-	-

3 state encodings are not specified so we can set them to don't care values
(we can never get to these states)

$S_{2\text{-NEXT}} = S_1 S_0 x'$

K-MAP FOR OUTPUTS (S_2, S_1, S_0,Y)

S_1 S_2

x S_0

	00	01	11	10
00	0	0	-	0
01	0	1	-	1
11	1	1	-	-
10	1	0	-	-

$S_{1\text{-NEXT}} = S_1x + S_0x + S_2S_1'x + S_1'S_0$

K-MAP FOR OUTPUTS (S_2, S_1, S_0,Y)

S_1 S_2

x S_0

	00	01	11	10
00	0	0	-	0
01	1	1	-	1
11	1	1	-	-
10	0	0	-	-

$S_{0\text{-NEXT}} = x$

K-MAP FOR OUTPUTS (S_2, S_1, S_0,Y)

S_1 S_2

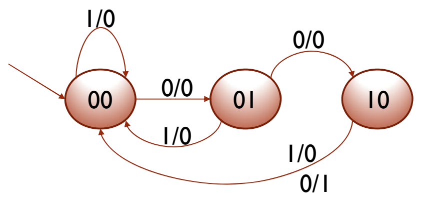
x S_0

	00	01	11	10
00	0	0	-	0
01	0	0	-	1
11	0	1	-	-
10	0	0	-	-

$Y = S_2x + S_1S_0x$

MEALY STATE MACHINE EXAMPLE

State machine outputs 1 for one clock cycle when three consecutive 0s are received as input with no overlap between sequences



$$\begin{tikzcd}
\arrow[r] & 00 \arrow[loop above, "1/0"] \arrow[r, "0/0"] & 01 \arrow[l, bend left, "1/0"] \arrow[r, bend left, "0/0"] & 10 \arrow[ll, bend left=40, "1/0\ 0/1"]
\end{tikzcd}$$

VERILOG PROCEDURAL BLOCKS - REVIEW

```
always @(A or B or C)
begin
  Q = D;  // Q must be of type reg
...
end


always @(posedge clock or negedge reset)


always @(posedge clock, negedge reset)
```

FSM IN VERILOG

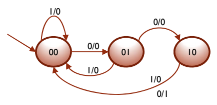
- Suggested coding style for FSMs

<module statement>
<input and output declarations>
<reg declarations>
<parameter and typedef statement>
<*always* block for next state>
<*always* block for output>
<*always* block for state FFs>
endmodule

MEALY STATE MACHINE EXAMPLE

```
module example_state_machine(clock, reset, in, out);
  output out;
  input clock, reset, in;
  reg [1:0] Scurr, Snext;
  parameter S0=2'b00, S1=2'b01, S2=2'b10;

  always @(in, reset, Scurr)
  begin
    if (reset == 1) Snext = S0;
    else
      case (Scurr)
        S0: if (in == 1) Snext = S0; else S1;
        S1: if (in == 1) Snext = S0; else S2;
        default: Snext = S0;
      endcase
  end
  always @(in, Scurr)
    if ((Scurr == S2) && (in == 0)) out = 1; else out = 0;
  always @(posedge clock)
    Scurr <= Snext;
endmodule
```



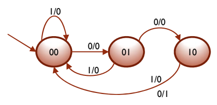
MEALY STATE MACHINE EXAMPLE – VERSION 2

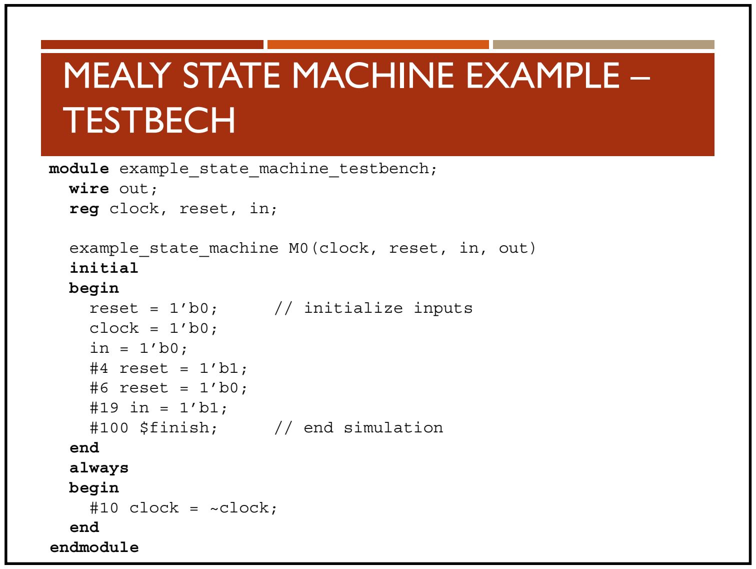
```
module example_state_machine(clock, reset, in, out);
  output out;
  input clock, reset, in;
  reg [1:0] Scurr, Snext;
  parameter S0=2'b00, S1=2'b01, S2=2'b10;

  always @(in, Scurr)
    begin
      case (Scurr)
        S0: if (in == 1) Snext = S0; else S1;
        S1: if (in == 1) Snext = S0; else S2;
        default: Snext = S0;
      endcase
    end
  always @(in, Scurr)
    if ((Scurr == S2) && (in == 0)) out = 1; else out = 0;

  always @(posedge clock)
    if (reset == 1) Scurr <= S0;
    else Scurr <= Snext;
endmodule
```

assumes FF has reset input





MEALY STATE MACHINE EXAMPLE – TESTBECH

```
module example_state_machine_testbench;
  wire out;
  reg clock, reset, in;

  example_state_machine M0(clock, reset, in, out)
  initial
  begin
    reset = 1'b0;      // initialize inputs
    clock = 1'b0;
    in = 1'b0;
    #4 reset = 1'b1;
    #6 reset = 1'b0;
    #19 in = 1'b1;
    #100 $finish;      // end simulation
  end
  always
  begin
    #10 clock = ~clock;
  end
endmodule
```