BROWN
School of Engineering

DIGITAL ELECTRONICS SYSTEM DESIGN

FALL 2019

PROF. IRIS BAHAR

OCTOBER 28, 2019

LECTURE 15: MORE STATE MACHINE DESIGN

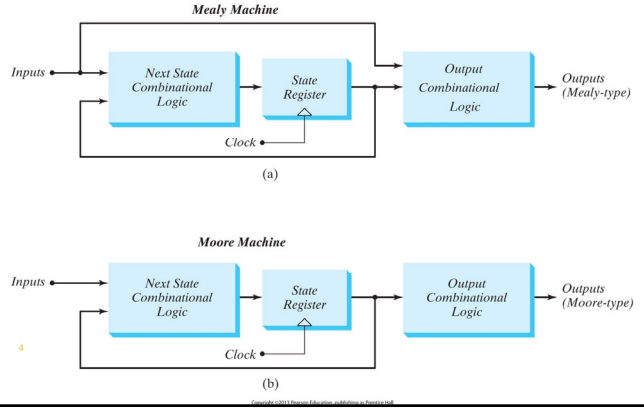
UPDATED LAB TA HOURS

- Pratishtha will be away attending a conference this week
  - She is canceling her Tuesday hours from 10am-noon
  - She will hold normal hours on Friday from 1-4pm

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.
- She will hold office hours in B&H 196 this Tuesday, Oct. 29 from 7-9pm
- Practice Problem Set #2 has been posted (as well as solutions)

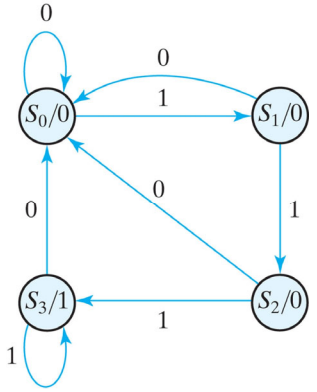
BLOCK DIAGRAMS OF MEALY AND MOORE STATE MACHINES



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

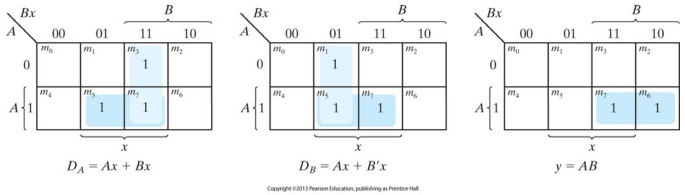


- Make a machine that sets an output signal to 1 when the input signal is 1 for 3 or more times in a row
- State diagram to detect 3 ones in a row
- *Is this a Mealy or Moore machine?*

STATE TABLE FOR SEQUENCE DETECTOR: MOORE MACHINE

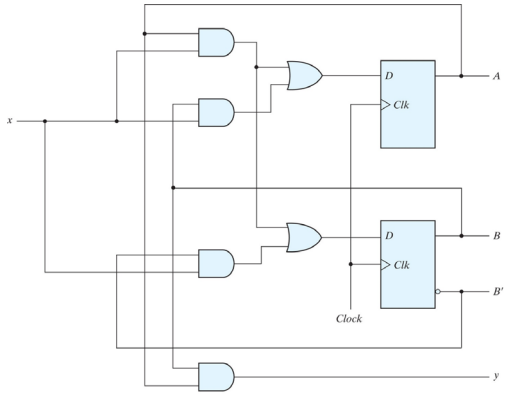
Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

K-MAPS FOR SEQUENCE DETECTOR USING D-FFS



- Each output is represented with a separate Karnaugh map

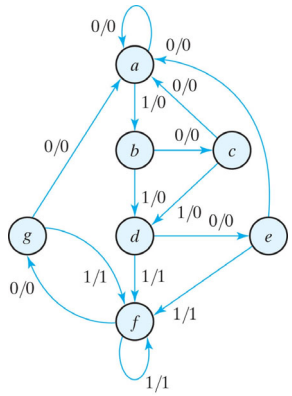
LOGIC DIAGRAM OF A MOORE-TYPE SEQUENCE DETECTOR



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

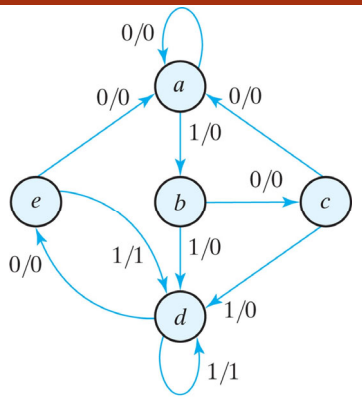


Is this a Mealy or Moore machine?

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

REDUCED STATE DIAGRAM



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

REDUCED STATE TABLE

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
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)

x \ S_1S_2	00	01	11	10
00			--	
01			--	
11			--	--
10			--	--

3 state encodings (101, 110, 111) are not specified so we can set their next state values to don't care
(we can never get to these states)

$S_{1\text{-NEXT}} = ?$

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

x \ S_1S_2	00	01	11	10
00	0	0	-	0
01	0	0	-	0
11	0	0	-	-
10	0	1	-	-

3 state encodings (101, 110, 111) are not specified so we can set their next state values to don't care
(we can never get to these states)

$S_{2\text{-NEXT}} = S_1S_0x'$

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

$S_{1\text{-NEXT}}$

S_1 \ S_2	00	01	11	10
00	0	0	-	0
01	0	1	-	1
11	1	1	-	-
10	1	0	-	-

(x / S_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	00	01	11	10
00	0	0	-	0
01	1	1	-	1
11	1	1	-	-
10	0	0	-	-

(x / S_0)

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

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

S_1 \ S_2	00	01	11	10
00	0	0	-	0
01	0	0	-	1
11	0	1	-	-
10	0	0	-	-

(x / S_0)

$Y = S_2x + S_1S_0x$

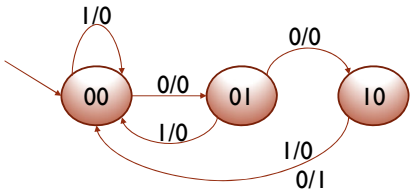
FSM IN VERILOG

- 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

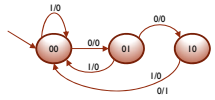
State machine outputs 1 for one clock cycle when three consecutive 0s are received as input with no overlap between sequences



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
```

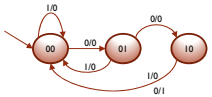


MEALY STATE MACHINE EXAMPLE – VERSION 2

```
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
```

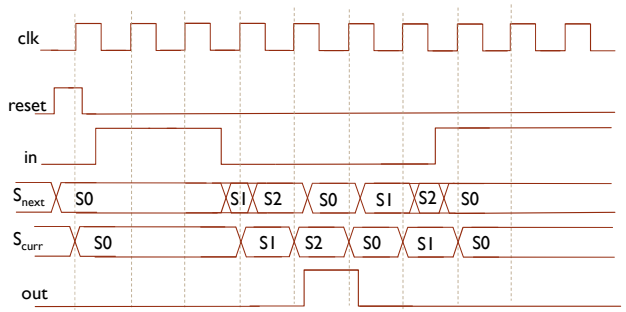


assumes FF has reset input

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;
    #12 reset = 1'b0;
    #19 in = 1'b1;
    #65 in = 1'b0;
    #141 in = 1'b1;
    #200 $finish;       // end simulation
  end
  always
  begin
    #10 clock = ~clock;
  end
endmodule
```

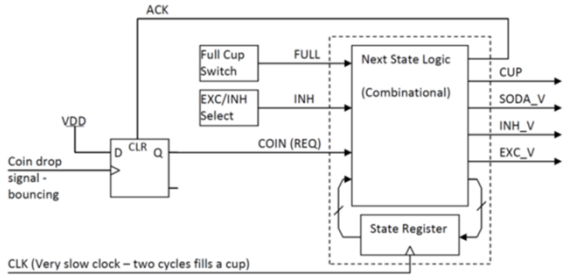
SIMULATION OUTPUT



THE DIABOLICAL COKE MACHINE

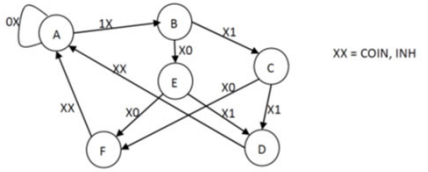
DESIGN FROM PROF. BILL PATTERSON
(FOUND ON HANDOUT PAGE ON COURSE WEBSITE)

DIABOLICAL COKE MACHINE



- Machine dispenses cup when coin is inserted and fills cup with soda.
- Shut off soda stream when FULL or after 2 clock cycles.
- Counting is handled through INH signal (but still counts 2 cycles regardless of value of INH)

STATE DIAGRAM



- Next state only determined by current state, coin drop, and INH signal
- Outputs (INH_V, EXC_V, CUP, SODA) determined by current state and FULL

STATE TABLE

State	State Name	Function	Style 1 (Binary)	Style 2 (*ad hoc*)	Style 3 (one-hot)
A	waiting	Wait for money	000	0000	000000
B	release	Drop a cup	001	1000	000011
C	soda_inh1	Dispense soda with inh 1 cycle	010	0001	000101
D	soda_inh2	Dispense soda with inh 2 cycle	011	0011	001001
E	soda_exc1	Dispense soda with exc 1 cycle	100	0101	010001
F	soda_exc2	Dispense soda with exc 2 cycle	101	0111	100001

- Six state can be encoded in different ways

STATE TRANSITION TABLE

COIN	INH	Pres. State	Q[2:0]	Next State	D[2:0]
0	X	A	000	A	000
1	X	A	000	B	001
X	0	B	001	E	100
X	1	B	001	C	010
X	0	C	010	F	101
X	1	C	010	D	011
X	0	E	100	F	101
X	1	E	100	D	011
X	X	D	011	A	000
X	X	F	101	A	000

COIN	INH	Pres. State	Q[3:0]	Next State	D[3:0]
0	X	A	0000	A	0000
1	X	A	0000	B	1000
X	0	B	1000	E	0101
X	1	B	1000	C	0001
X	0	C	0001	F	0111
X	1	C	0001	D	0011
X	0	E	0101	F	0111
X	1	E	0101	D	0011
X	X	D	0011	A	0000
X	X	F	0111	A	0000

- State transitions are the same, just different encodings
- What about the output signals (CUP, INH_V, EXC_V, SODA)?

OUTPUT TABLE

Full	Pres. State	Cup	Soda	INH_V	EXC_V
X	A	0	0	0	0
X	B	1	0	0	0
0	C,D	0	1	1	0
1	C,D	0	0	0	0
0	E,F	0	1	0	1
1	E,F	0	0	0	0

- 2 different ways to count
- Logic for these outputs will be different depending on state encoding

VERILOG

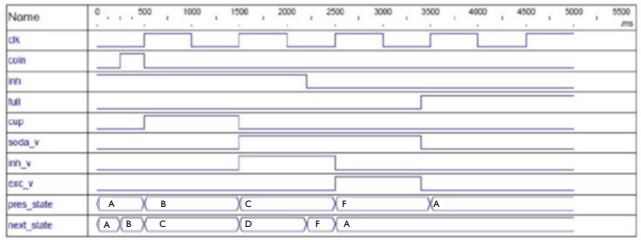
```
module coke_style1 (input full ,inh ,coin ,clk , output reg cup ,inh_v ,exc_v ,soda_v );
reg [2:0] pres_state, next_state; // State variables

// State bit assignments
parameter [2:0] waiting = 3'b000, cup_drop = 3'b001, dispense_i1 = 3'b010,
dispense_i2 = 3'b011, dispense_e1 = 3'b100, dispense_e2 = 3'b101;
// Next state logic
always @ (pres_state, coin, inh) begin
 case (pres_state)
   waiting: if (coin == 1'b1) next_state = cup_drop;  else next_state = waiting;
   cup_drop: if (inh == 1) next_state = dispense_i1;  else next_state = dispense_e1;
   dispense_i1: if (inh == 1) next_state = dispense_i2;  else next_state = dispense_e2;
   dispense_e1: if (inh == 1) next_state = dispense_i2;  else next_state = dispense_e2;
   dispense_i2: next_state = waiting;
   dispense_e2: next_state = waiting;
   default next_state = waiting;
 endcase
end
```

VERILOG (CONT.)

```
// Output logic - note that outputs are reg type even though not ff outputs
always @ (pres_state, full) begin
  if (pres_state == cup_drop) cup = 1;    else cup = 0;
  if (pres_state == dispense_i1) | (pres_state == dispense_i2)
    begin
      inh_v = ~full;
      soda_v = ~full;
    end
    if (pres_state == dispense_e1 | (pres_state == dispense_e2)
      begin
        exc_v = ~full;
        soda_v = ~full;
      end
  end
// State register
always @ (posedge clk) pres_state <= next_state;
endmodule
```

TIMING SIMULATION



- Notice the Coke machine starts filling the cup with soda with *inh*=1, then switches counting with *inh_v* to *exc_v* when inh=0.
- The *full* input signal goes high once soda has been dispensed for 2 full cycles.