Overview

- Last lecture
  - Introduction to finite-state machines
    - Moore versus Mealy machines
    - Synchronous Mealy machines
    - Example: A parity checker
- Today
  - Example: A sequence detector FSM
  - Example: A vending machine FSM
  - FSMs in Verilog

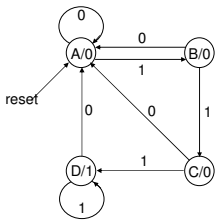
FSM design

- FSM-design procedure
  1. State diagram and state-transition table
  2. State minimization
  3. State assignment (or state encoding)
  4. Minimize next-state logic
  5. Implement the design

Example: Sequence detector

- Design a circuit to detect 3 or more 1's in a bit string
  - Assume Moore machine
  - Assume D flip-flops
  - Assume flip-flops have a reset

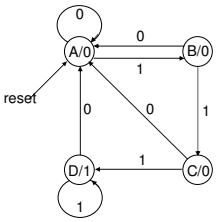
1. State diagram and state-transition table



reset	current state	input	next state	current output
1	–	–	A	0
0	A	0	A	0
0	A	1	B	0
0	B	0	A	0
0	B	1	C	0
0	C	0	A	0
0	C	1	D	0
0	D	0	A	1
0	D	1	D	1

2. State minimization & 3. State encoding

- State diagram is already minimized
- Try a binary encoding



reset	current state	input	next state	current output
1	–	–	00	0
0	00	0	00	0
0	00	1	01	0
0	01	0	00	0
0	01	1	10	0
0	10	0	00	0
0	10	1	11	0
0	11	0	00	1
0	11	1	11	1

4. Minimize next-state logic

MSB+

In \ M	0	0	0	0
0	0	1	1	1

L

MSB+ = LIn + MIn

LSB+

In \ M	0	0	0	0
1	0	1	1	1

L

LSB+ = L'In + MIn

OUT+

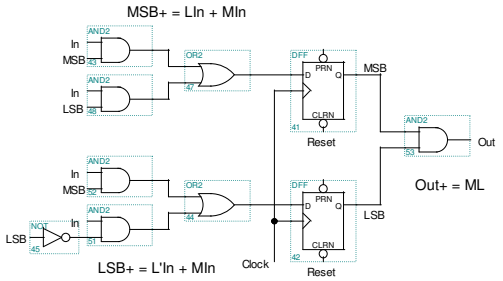
In \ M	0	0	1	0
0	0	0	1	0

L

Out+ = ML

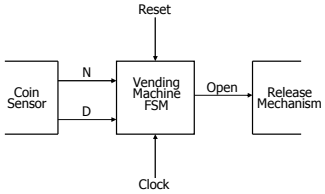
Notation
M := MSB
L := LSB
In := Input

5. Implement the design



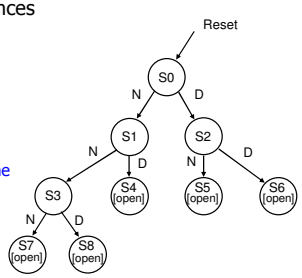
Design example: A vending machine

- Release item after receiving 15 cents
  - Single coin slot for dimes and nickels
    - Sensor specifies coin type
  - Machine does not give change



1a. State diagram

- Consider input sequences
  - 3 nickels
  - 2 nickels, dime
  - nickel, dime
  - dime, nickel
  - two dimes
- Draw state diagram
  - Assume Moore machine
  - Inputs: N, D, reset
  - Output: Open

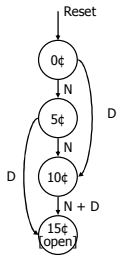


1b. Symbolic state-transition table

present state	inputs D	N	next state	present output
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	–	–
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	–	–
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	–	–
15¢	–	–	15¢	1

2. State minimization

- Reuse states where possible
  - Notice we can use the deposited coin values for states
  - State is the same if we input 2 nickels or 1 dime



3. State encoding

- Encode states uniquely
  - 4 states:
    - 2 bits minimum
    - 4 bits maximum
  - Look for optimal encoding
  - Assume D flip-flops

present state Q0	Q1	inputs D	N	next state P1	P0	present output
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	–	–	–
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	–	–	–
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	–	–	–
1	1	–	–	1	1	1

4. Minimize the logic

K-map for P1

DN \ Q1Q0	00	01	11	10
00	0	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	1	1

N, D, Q0

K-map for P0

DN \ Q1Q0	00	01	11	10
00	0	1	1	0
01	1	0	1	1
11	X	X	X	X
10	0	1	1	1

N, D, Q0

K-map for Open

DN \ Q1Q0	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	X	X	X	X
10	0	0	1	0

N, D, Q0

$P_1 = Q_1 + D + Q_0N$

$P_0 = Q_0'N + Q_0N' + Q_1N + Q_1D$

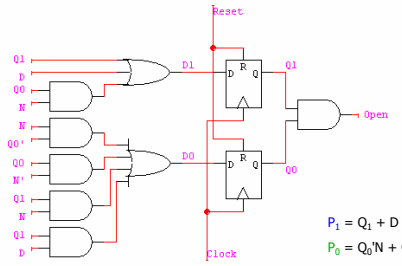
$OPEN = Q_1Q_0$

if FFs do not have a reset pin then

$P_1 = reset'(Q_1 + D + Q_0N)$

$P_0 = reset'(Q_0'N + Q_0N' + Q_1N + Q_1D)$

5. Implement the design



$P_1 = Q_1 + D + Q_0N$

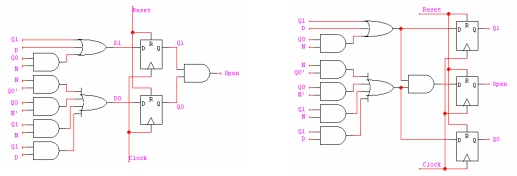
$P_0 = Q_0'N + Q_0N' + Q_1N + Q_1D$

$OPEN = Q_1Q_0$

Retime design

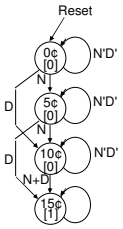
- OPEN is delayed by AND gate after Q_1 and Q_0
  - Can remove this delay by retiming
    - Move output logic (AND gate) to eliminate delay
  - OPEN $= Q_1Q_0 = (Q_1+D+Q_0N)(Q_0'N+Q_0N'+Q_1N+Q_1D)$

Original Design — Retimed Design

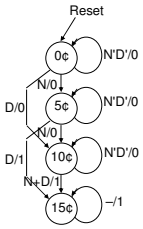


Moore versus Mealy vending machine

Moore machine



Mealy machine



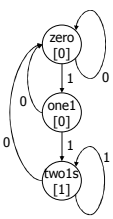
Hardware Description Languages and Sequential Logic

- Flip-flops
  - representation of clocks - timing of state changes
  - asynchronous vs. synchronous
- FSMs
  - structural view (FFs separate from combinational logic)
  - behavioral view (synthesis of sequencers – not in this course)
- Data-paths = data computation (e.g., ALUs, comparators) + registers
  - use of arithmetic/logical operators
  - control of storage elements

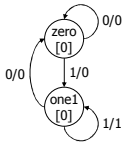
Example: reduce-1-string-by-1

- Remove one 1 from every string of 1s on the input

Moore



Mealy

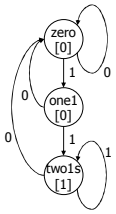


Verilog FSM - Reduce 1s example

- Moore machine

```
`define zero  0
`define one1  1
`define twols 2

module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg [2:1] state;       // state variables
  reg [2:1] next_state;

  always @(posedge clk)
    if (reset) state = `zero;
    else       state = next_state;
```

state assignment



Moore Verilog FSM (cont'd)

```
always @(in or state)

  case (state)
    `zero:
   // last input was a zero
    begin
      if (in) next_state = `one1;
      else    next_state = `zero;
    end
     `one1:
   // we've seen one 1
    begin
      if (in) next_state = `twols;
      else    next_state = `zero;
    end
     `twols:
   // we've seen at least 2 ones
    begin
      if (in) next_state = `twols;
      else    next_state = `zero;
    end
  endcase
```

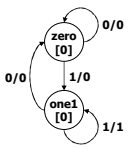
crucial to include all signals that are input to state determination

note that output depends only on state

```
always @(state)
  case (state)
     `zero: out = 0;
     `one1: out = 0;
    `twols: out = 1;
  endcase

endmodule
```


Mealy Verilog FSM

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state;   // state variables
  reg next_state;

  always @(posedge clk)
    if (reset) state = `zero;
    else       state = next_state;

  always @(in or state)
    case (state)
      `zero:               // last input was a zero
      begin
        out = 0;
        if (in) next_state = `one1;
        else    next_state = `zero;
      end
      `one1:               // we've seen one 1
      if (in) begin
          next_state = `one1; out = 1;
      end else begin
          next_state = `zero; out = 0;
      end
    endcase
endmodule
```



Synchronous Mealy Machine

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg state; // state variables

  always @(posedge clk)
    if (reset) state = `zero;
    else
     case (state)
      `zero:     // last input was a zero
      begin
        out = 0;
        if (in) state = `one1;
        else    state = `zero;
      end
       `one1:     // we've seen one 1
      if (in) begin
         state = `one1; out = 1;
      end else begin
         state = `zero; out = 0;
      end
    endcase
endmodule
```