Overview

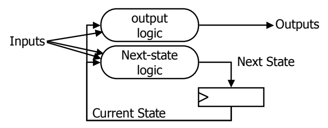
- ◆ Last lecture
  - Finished counter design
    - Design example
    - Self-starting counters
- ◆ Today
  - Introduction to finite-state machines
    - Moore versus Mealy machines
    - Synchronous Mealy machines
    - Example: A parity checker

Finite state machines

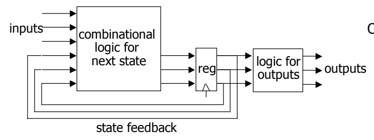
- ◆ FSM: A system that visits a finite number of logically distinct states
- ◆ Counters are simple FSMs
  - Outputs and states are identical
  - Visit states in a fixed sequence
- ◆ FSMs are more complex than counting
  - Outputs can depend on current state and on inputs
  - State sequencing depends on current state and on inputs

Generalized FSM model

- ◆ State variables (state vector) holds circuit state
  - Stored in registers
- ◆ Combinational logic computes next state and outputs
  - Next state is a function of current state and inputs
  - Outputs are functions of
    - Current state (Moore machine)
    - Current state and inputs (Mealy machine)

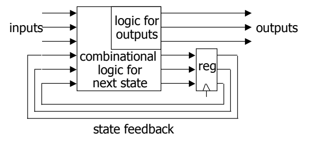


Moore versus Mealy machines



Moore machine
Outputs are a function of current state

Outputs change synchronously with state changes

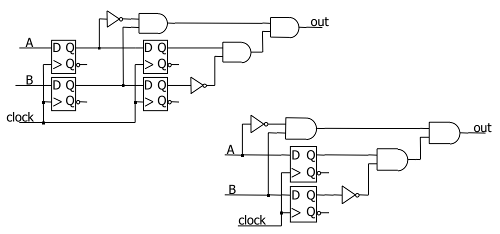


Mealy machine
Outputs depend on state and on inputs

Input changes can cause immediate output changes **(asynchronous)**

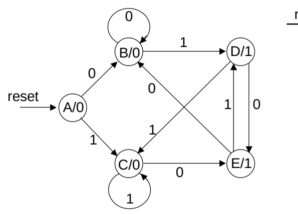
Example: Moore versus Mealy

- ◆ Circuits recognize AB=10 followed by AB=01
  - What kinds of machines are they?



Specifying outputs for a Moore machine

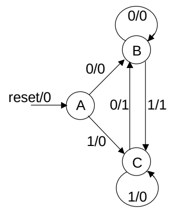
- ◆ Output is a function of state only
  - Specify output in the state bubble
  - Example: Detector for 01 or 10



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

Specifying outputs for a Mealy machine

- Output is a function of state and inputs
  - Specify outputs on transition arcs
  - Example: Detector for 01 or 10



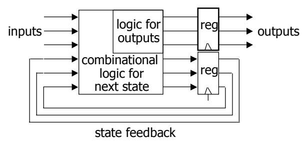
reset	input	current state	next state	current output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

Comparing Moore and Mealy machines

- Moore machines
  - + Safer do use because outputs change at clock edge
  - – May take additional logic to decode state into outputs
- Mealy machines
  - + Typically have fewer states
  - + React faster to inputs — don't wait for clock
  - – Asynchronous outputs can be dangerous
- We will often design synchronous Mealy machines
  - Design a Mealy machine
  - Then register the outputs

Synchronous (registered) Mealy machine

- Registered state and registered outputs
  - No glitches on outputs
  - No race conditions between communicating machines



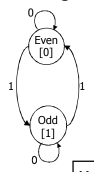
FSM design

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

Example: A parity checker

- Serial input string
  - OUT=1 if odd # of 1s in input
  - OUT=0 if even # of 1s in input

1. State diagram and state-transition table



Present State	Input	Next State	Present Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

Moore-machine state diagram

Parity checker (con't)

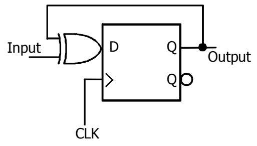
2. State minimization: Already minimized
   - Need both states (even and odd)
   - Use one flip-flop
3. State assignment (or state encoding)

Present State	Input	Next State	Present Output
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Assignment
Even → 0
Odd → 1

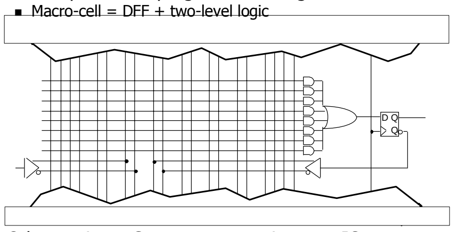
Parity checker (con't)

4. Next-state logic minimization
   - Assume D flip-flops
   - Next state = (present state) XOR (present input)
   - Present output = present state
5. Implement the design



Implementation

- Can map FSMs to programmable logic devices
  - Macro-cell = DFF + two-level logic



- Other options: Gate arrays, semicustom ICs, etc.

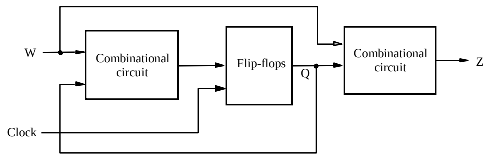


Figure 8.1. The general form of a sequential circuit.

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Figure 8.2. Sequences of input and output signals.

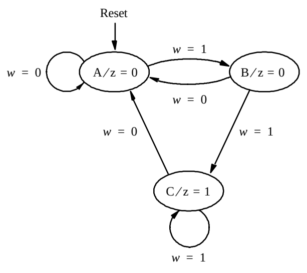


Figure 8.3. State diagram of a simple sequential circuit.

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	C	1

Figure 8.4. State table.

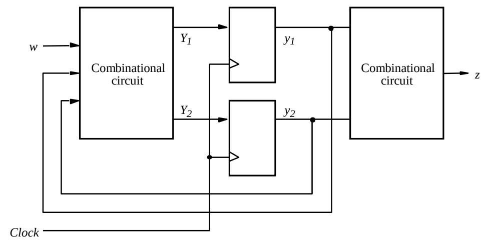
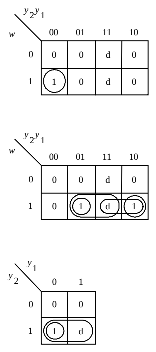


Figure 8.5. A general sequential circuit.

	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	y_2y_1	Y_2Y_1	Y_2Y_1	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Figure 8.6. A state-assigned table.



Ignoring don't cares

$Y_1 = w\bar{y}_1\bar{y}_2$

$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$

$z = \bar{y}_1y_2$

Using don't cares

$Y_1 = w\bar{y}_1\bar{y}_2$

$$\begin{aligned} Y_2 &= wy_1 + wy_2 \\ &= w(y_1 + y_2) \end{aligned}$$

$z = y_2$

Figure 8.7. Derivation of logic expressions.

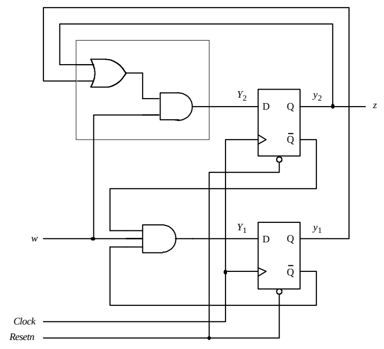


Figure 8.8. Sequential circuit derived in Figure 8.7.

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y, Y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    // Define the next state combinational circuit
    always @(w or y)
        case (y)
            A: if (w)   Y = B;
               else    Y = A;
            B: if (w)   Y = C;
               else    Y = A;
            C: if (w)   Y = C;
               else    Y = A;
            default:    Y = 2'bxx;
        endcase

    // Define the sequential block
    always @(negedge Resetn or posedge Clock)
        if (Resetn == 0)y <= A;
        elsey <= Y;

    // Define output
    assign z = (y == C);

endmodule
```

Figure 8.29. Verilog code for the FSM in Figure 8.3.

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] state, NextState;
    parameter [2:1] stateA = 2'b00, stateB = 2'b01, stateC = 2'b10;

    // Define the next state combinational circuit
    always @(w or state)
        case (state)
            stateA : if (w)  NextState = stateB;
              else NextState = stateA;
            stateB : if (w)  NextState = stateC;
              else NextState = stateA;
            stateC : if (w)  NextState = stateC;
              else NextState = stateA;
            default:    NextState = 2'bxx;
        endcase

    // Define the sequential block
    always @(negedge Resetn or posedge Clock)
        if (Resetn == 0) state <= stateA;
        else state <= NextState;

    // Define output
    assign z = (state == stateC);

endmodule
```

Figure 8.29. Re-written Verilog code for the FSM in Figure 8.3.