ECE 2300
Digital Logic & Computer Organization

Spring 2025

More Verilog
Finite State Machines

Cornell University

Announcements

- Lab 2 released
- HW 3 due tomorrow
- Prelim 1
  - Thursday Feb 27th, 1:25-2:40pm in class
    - closed book, closed notes, closed Internet
  - Coverage: Lectures 1~6, first part of Lecture 7
    - Binary number, Boolean algebra, CMOS, combinational logic, sequential logic
  - A sample prelim exam will be posted tomorrow
  - A TA-led review session will be scheduled (& recorded) on Feb 24

Recap: Continuous Assignments in Verilog

- **Continuous assignments apply to combinational logic only**

- **Multiple continuous assignments happen in parallel; the order does not matter**

```
wire a, b, c, d;
assign c = a & d; // Uses d, even though d is assigned later
assign d = ~b;
```

Exercise: Verilog Circuit Modeling

- **Which of the following Verilog code snippets infer sequential logic**

```
input clk, d;
output q;
assign q = clk & d;
```

(a)

```
input clk, d;
reg q;
always @ (clk, d)
begin
  q = clk & d;
end
```

(b)

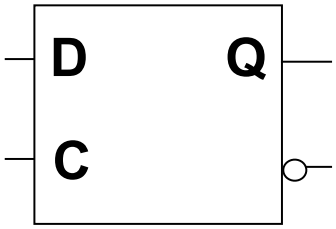
```
input clk, d;
reg q;
always @ (clk)
begin
  if ( clk )
    q = d;   // D latch?
end
```

(c)

Recap: Sequential Logic with Always Blocks

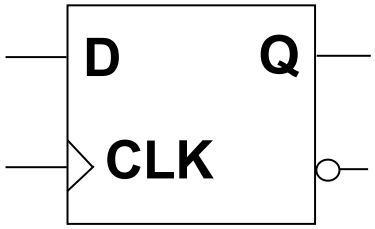
Sequential logic can ONLY be modeled using always blocks

```
reg Q;

always @(clk, D)
begin
  if ( clk )
      Q = D;
end
```



D latch

```
reg Q;

always @(posedge clk)
begin
  Q = D;
end
```



DFF

Q is declared as a "reg" since it appears on the left-hand side of a procedural assignment

Procedural Assignments in Always Blocks

- ***Always blocks*** **contain a set of procedural assignments (blocking or nonblocking)**

Simulation behavior:

- **Blocking assignments (=) execute RHS sequentially within the always block, completing each assignment to LHS before moving to the next**

- **Non-blocking assignments (<=) execute RHS in parallel and the assignments are scheduled to update LHS at the end of the always block**

Blocking Assignments

- **Left-hand side (LHS) = Right-hand side (RHS)**

```
input A, B;
reg Y, Z;
always @ (posedge clk)
begin
  Y = A & B;
  Z = Y;
end
```

Y and Z are inferred as FFs here, since the always block is sensitive to the clock edge

---

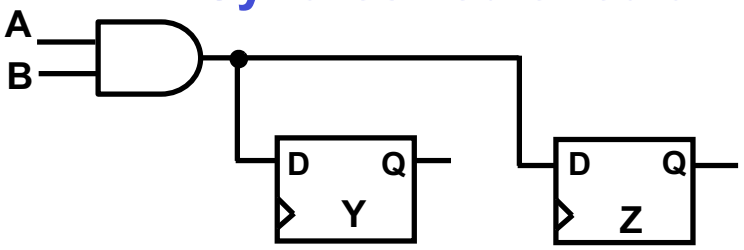
Simulation behavior

$Y_{next} \leftarrow A \& B$

$Z_{next} \leftarrow (Y_{next} = A \& B)$ // use "new" Y

- RHS evaluated *sequentially*
- Assignment to LHS is *immediate*

Synthesized circuit



When a reg (Y here) is assigned in a *blocking* assignment (Y=A&B), *employ its* ***D input*** (i.e., A&B) *for connection* in RHS of a subsequent assignment (Z=Y)

Nonblocking Assignments

- **Left-hand side (LHS) <= Right-hand side (RHS)**

```
input A, B;
reg Y, Z;
always @ (posedge clk)
begin
  Y <= A & B;
  Z <= Y;
end
```

Y and Z are inferred as FFs here, since the always block is sensitive to the clock edge

---

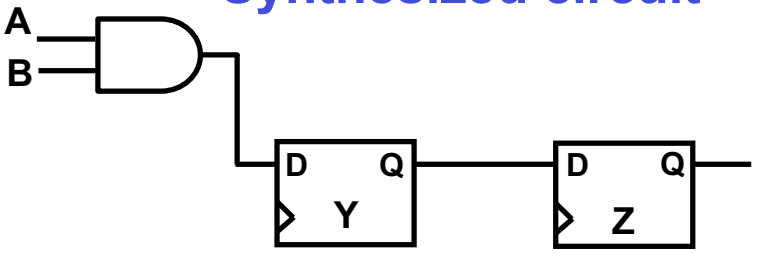
Simulation behavior

$Z_{next} \leftarrow Y$ // use "old" Y

$Y_{next} \leftarrow A \& B$

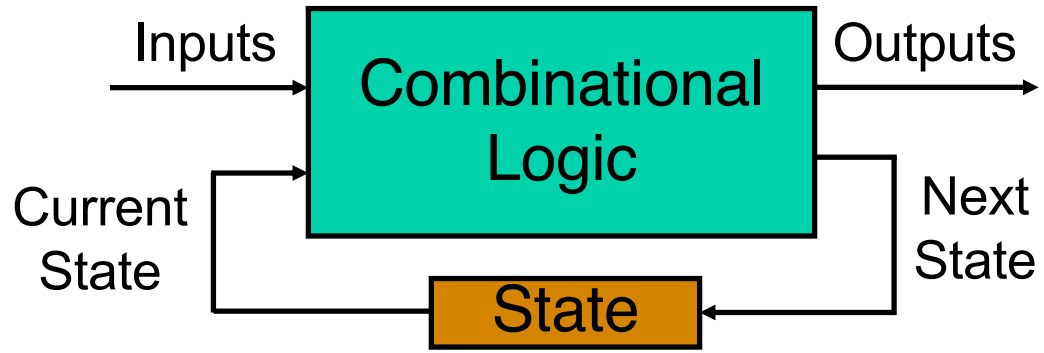
- RHS evaluated *in parallel* (order doesn't matter)
- Assignment to LHS is *delayed* until the end of the always block

Synthesized circuit



When a reg (Y here) is assigned in a *nonblocking* assignment (Y<=A&B), *employ its Q **output** for connection* in RHS of another assignment (Z<=Y)

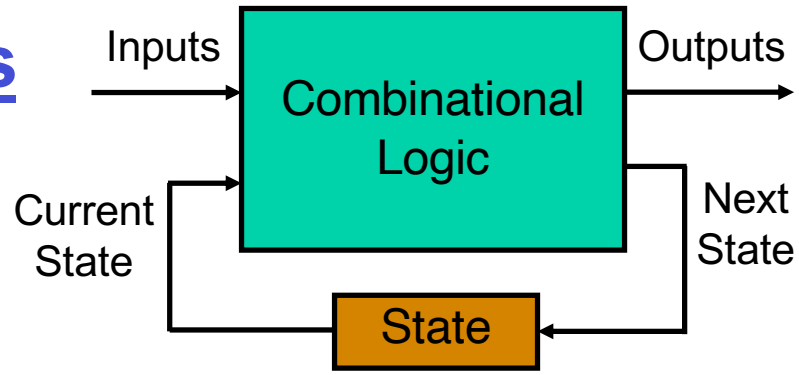
Finite State Machine



- **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 (Moore), or on the current state and input values (Mealy)**

Elements 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



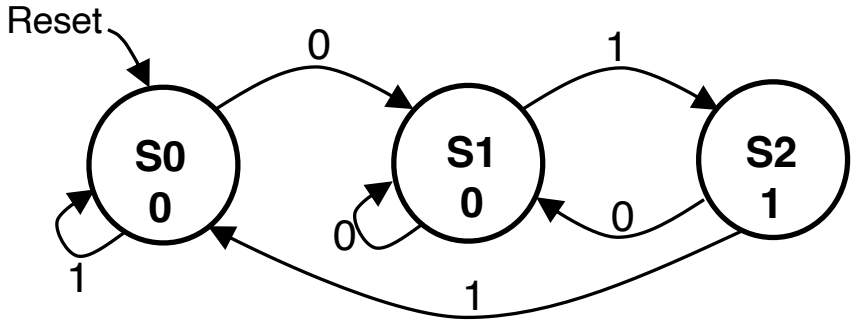
Can be described by a state diagram

- Inputs and current state determine state transitions
- Output changes determined by changes in
  - Current state (More FSM), or
  - Current state + inputs (Mealy FSM)

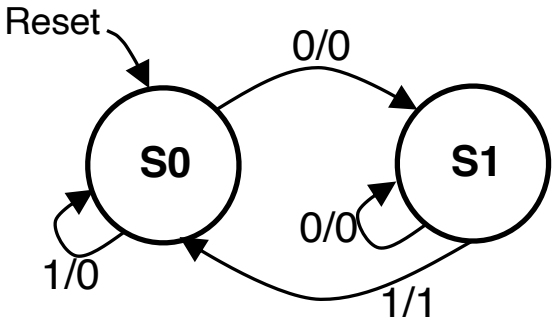
State Diagram

- **Visual specification of an FSM**
  - **Bubble for every state**
  - **Arcs showing state transitions**
  - **Input values shown on the arcs**
  - **Output values shown within the bubbles (Moore) or on the arcs (Mealy)**
  - **Clock input implicit (always present, triggering state transitions)**

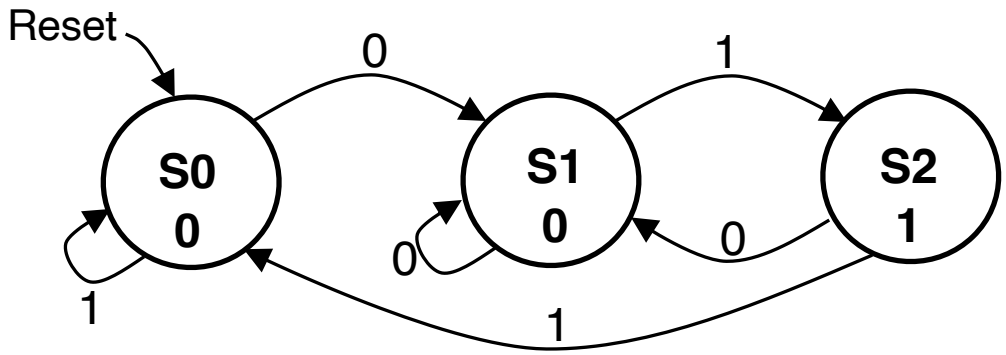
Moore FSM Example



Mealy FSM Example



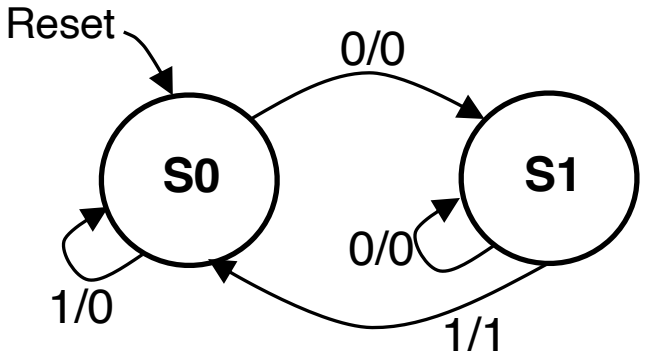
Example: Moore State Diagram



Moore FSM Example

- 1 input (1 bit), 1 output (1 bit), 3 states
- Bubble for each state
- State transitions (arcs) for each input value
- Input values on the arcs
- Output values within the bubbles
- Starts at S0 when Reset asserted

Example: Mealy State Diagram



Mealy FSM Example

- 1 input (1 bit), 1 output (1 bit), 2 states
- Bubble for each state
- State transitions (arcs) for each input value
- Input values on the arcs (first number)
- Output values on the arcs (second number)
- Starts at S0 when Reset asserted

FSM Design Procedure

(1) Understand the problem statement and determine inputs and outputs

(2) Identify states and create a *state diagram*

This lecture

(3) Determine the number of required FFs

(4) Implement combinational logic for outputs and next state

(5) Simulate the circuit to test its operation

Example FSM: Pattern Detector

- Monitors the input, and outputs a 1 whenever a specified input pattern is detected

- Example: Output a 1 whenever 111 is detected on the input over 3 consecutive clock cycles
  - Overlapping patterns also detected (1111...)

- Input *In* (one bit)
- Output *Out* (one bit)
- *Reset* causes FSM to start in initial state
- *Clock* input not shown (always present)

111 Pattern Detector: Moore State Diagram

- Output a 1 whenever 111 is detected on the input over 3 consecutive clock cycles (overlapping pattern also detected)

111 Pattern Detector: Mealy State Diagram

- Output a 1 whenever 111 is detected on the input over 3 consecutive clock cycles (overlapping pattern also detected)

Example FSM: Pushbutton Lock

- Two pushbutton inputs, X1 and X2
- One output, UL ("Unlock")

- UL = 1: the lock is unlocked, when X1 is pushed, followed by X2 being pushed twice (X1, X2, X2)

- Represent X1 and X2 as two-bit input
  - 00: neither button pushed
  - 10: X1 pushed
  - 01: X2 pushed
  - 11: both pushed, reset the lock (to the locked state)

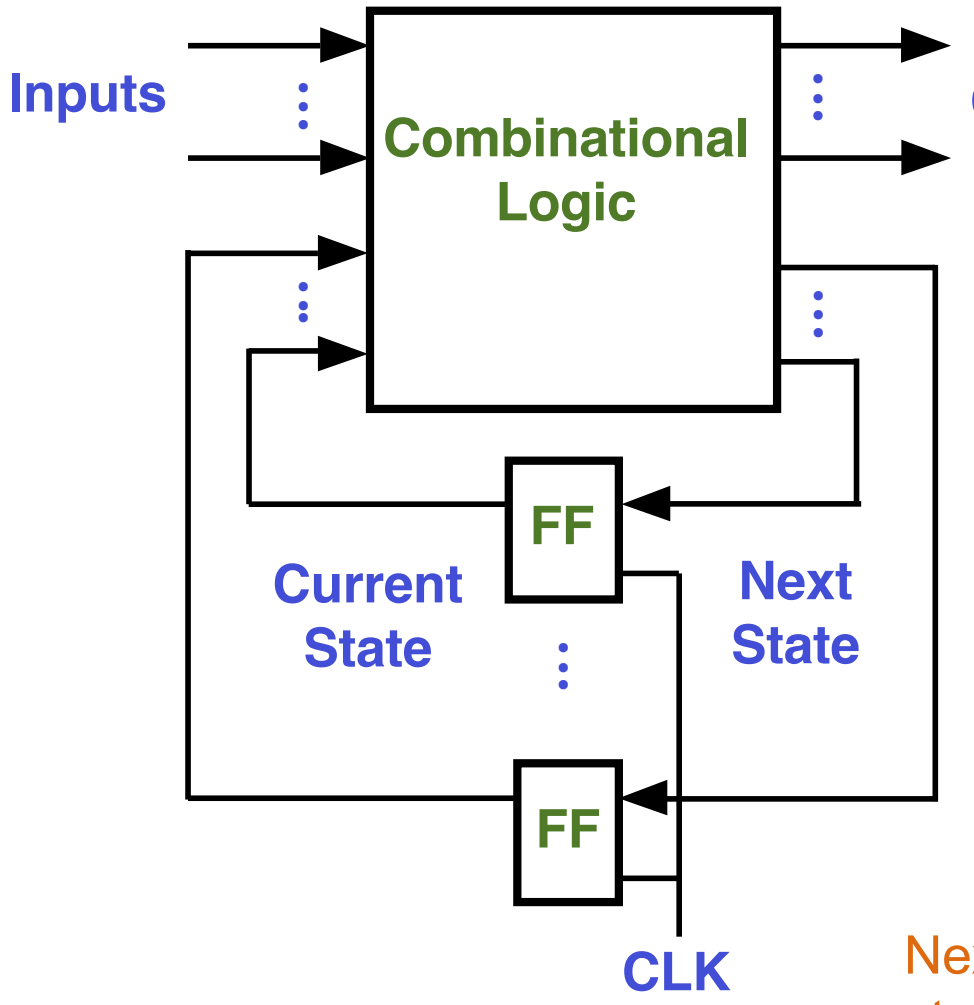
Pushbutton Lock: Moore State Diagram

- Output: UL=1 with (X1, X2, X2)
- Input: 00 (neither), 10 (X1), 01 (X2), 11 (reset)

Pushbutton Lock: Mealy State Diagram

- Output: UL=1 with (X1, X2, X2)
- Input: 00 (neither), 10 (X1), 01 (X2), 11 (reset)

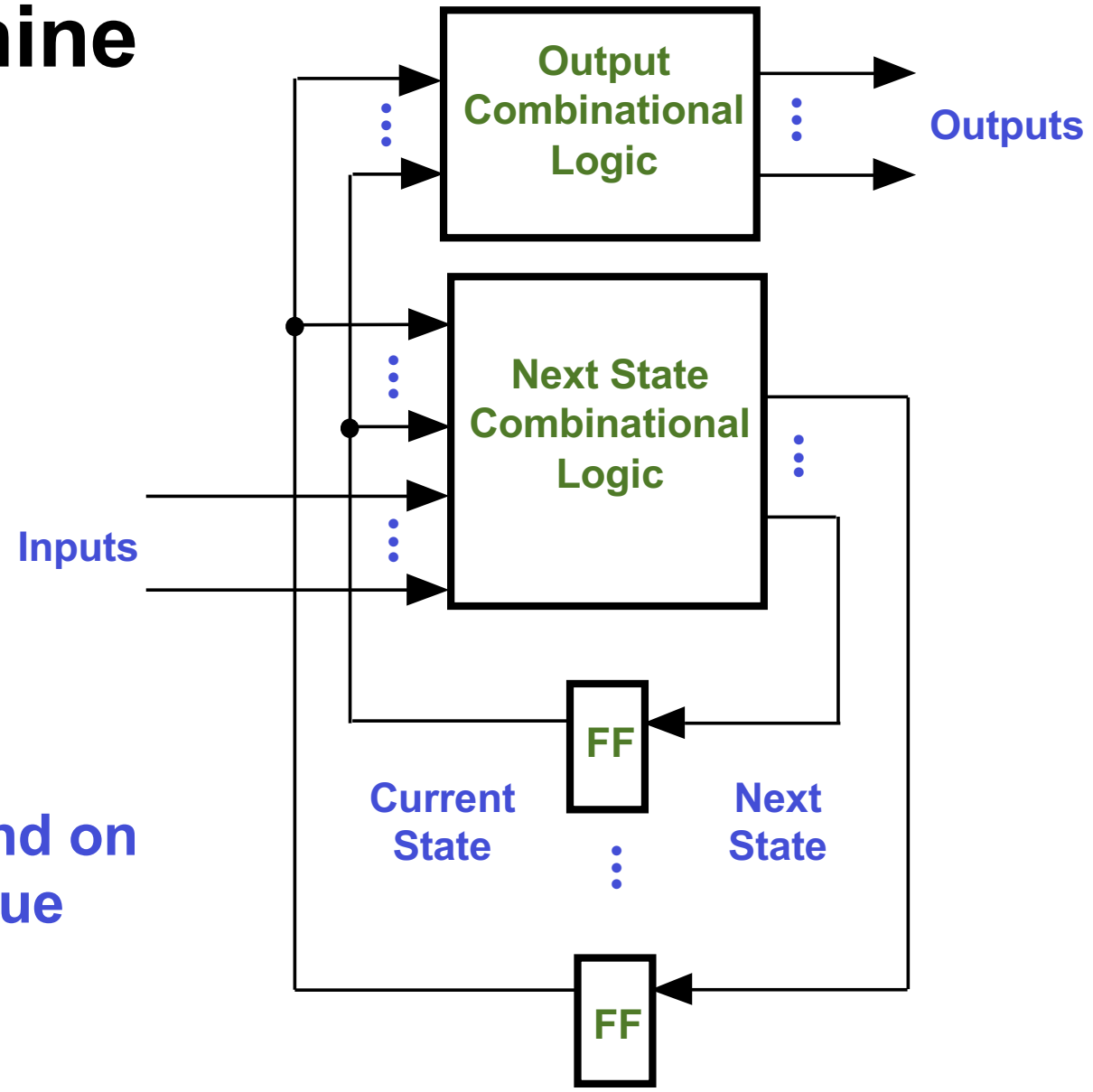
FSM: General Circuit Form



- **Inputs and current state determine state transitions**
- **Output changes determined by changes in**
  - **Current state (Moore), or**
  - **Current state + inputs (Mealy)**

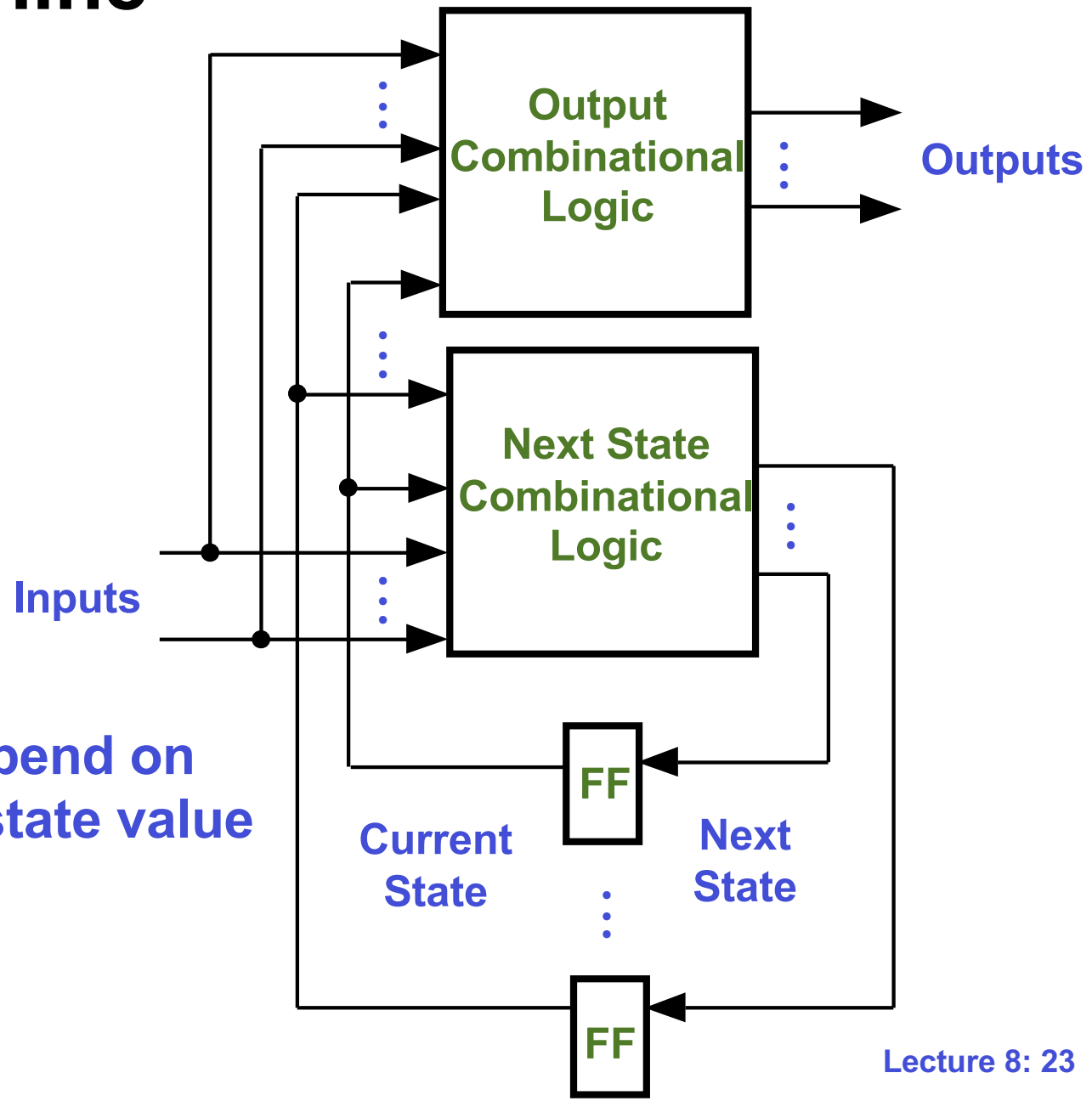
Next lecture will cover how to convert state diagrams into circuits

Moore Machine



Outputs only depend on current state value

Mealy Machine



Outputs only depend on input and current state value

Next Class

More Finite State Machines
(H&H 4.6, 4.9)