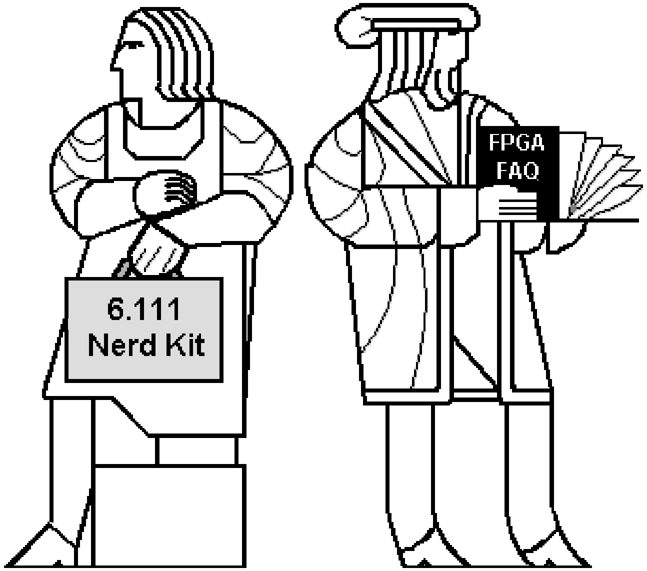
6.111 Lecture 6

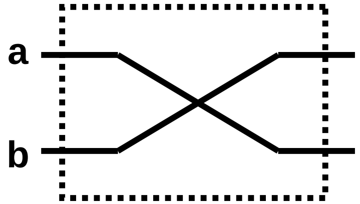
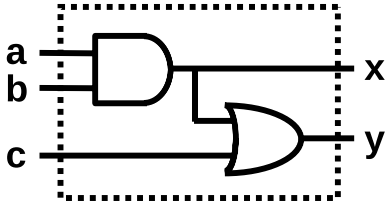
Today:

1. Blocking vs. non-blocking assignments
2. Single clock synchronous circuits
3. Finite State Machines



I. Blocking vs. Nonblocking Assignments

Conceptual need for two kinds of assignment (in always blocks):

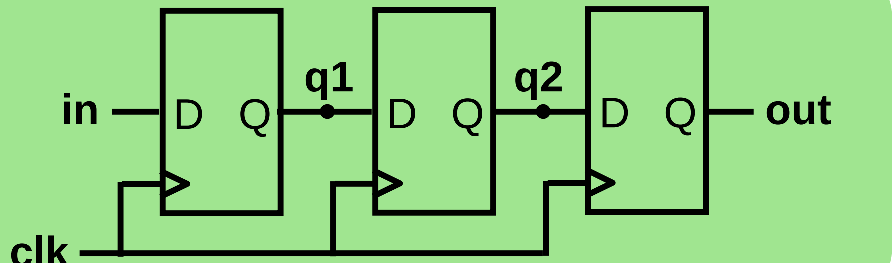
		

Blocking:
Evaluation and assignment are immediate
	~~a = b~~
	~~b = a~~
		x = a & b
		y = x | c

Non-Blocking: Assignment is postponed until all r.h.s. evaluations are done	a <= b b <= a	~~x <= a & b~~ ~~y <= x | c~~
When to use: (only in always blocks!)	Sequential Circuits	Combinatorial Circuits

Assignment Styles for Sequential Logic

Flip-Flop Based Digital Delay Line



- Will nonblocking and blocking assignments both produce the desired result?

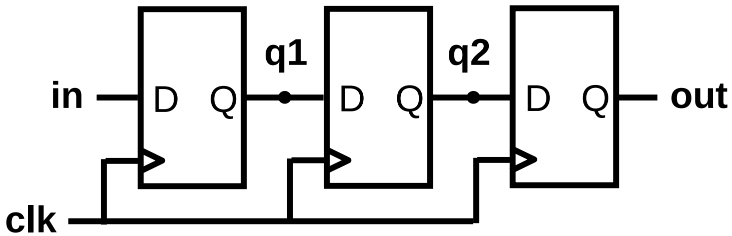
```
module nonblocking(in, clk, out);
  input in, clk;
  output out;
  reg q1, q2, out;
  always @ (posedge clk)
  begin
    q1 <= in;
    q2 <= q1;
    out <= q2;
  end
endmodule
```

```
module blocking(in, clk, out);
  input in, clk;
  output out;
  reg q1, q2, out;
  always @ (posedge clk)
  begin
    q1 = in;
    q2 = q1;
    out = q2;
  end
endmodule
```

Use Nonblocking for Sequential Logic

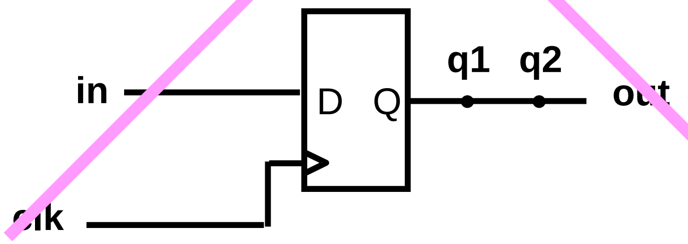
```
always @ (posedge clk)
begin
   q1 <= in;
   q2 <= q1;
   out <= q2;
end
```

“At each rising clock edge, *q1*, *q2*, and *out* simultaneously receive the old values of *in*, *q1*, and *q2*.”



```
always @ (posedge clk)
begin
   q1 = in;
   q2 = q1;
   out = q2;
end
```

“At each rising clock edge, *q1* = *in*. After that, *q2* = *q1* = *in*; After that, *out* = *q2* = *q1* = *in*; Finally *out* = *in*.”



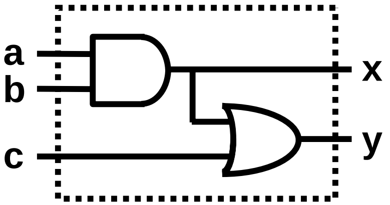
- Blocking assignments do not reflect the intrinsic behavior of multi-stage sequential logic

- **Guideline: use nonblocking assignments for sequential always blocks**

Use Blocking for Combinational Logic

Blocking Behavior	a b c x y
(Given) Initial Condition	1 1 0 1 1
a changes; always block triggered	0 1 0 1 1
`x = a & b;`	0 1 0 0 1
`y = x	c;`	0 1 0 0 0

```
always @ (a or b or c)
begin
   x = a & b;
   y = x | c;
end
```

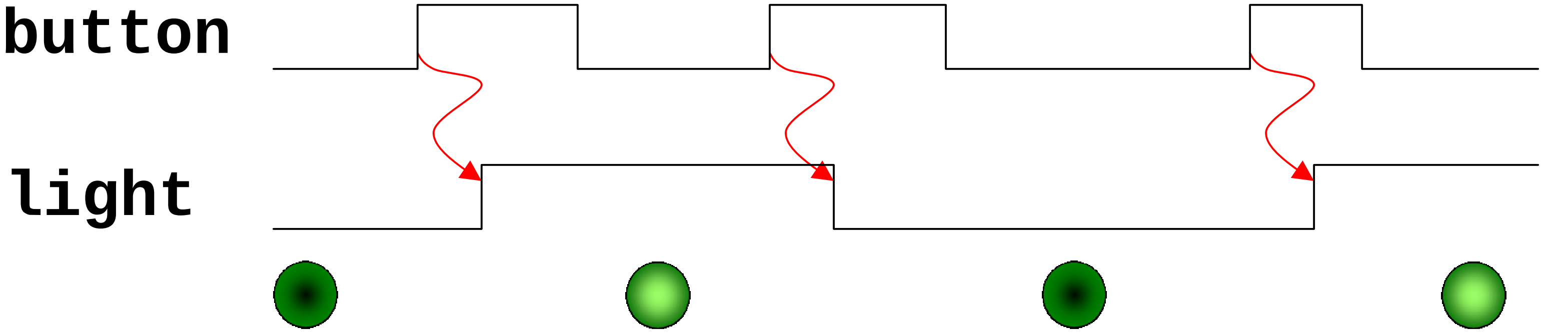


~~*Nonblocking Behavior*~~	~~a b c x y~~	~~*Deferred*~~
~~(Given) Initial Condition~~	~~1 1 0 1 1~~	
~~*a* changes; always block triggered~~	~~0 1 0 1 1~~	
~~`x <= a & b;`~~	~~0 1 0 1 1~~	~~x<=0~~
~~`y <= x	c;`~~	~~0 1 0 1 1~~	~~x<=0, y<=1~~
~~Assignment completion~~	~~0 1 0 0 1~~	

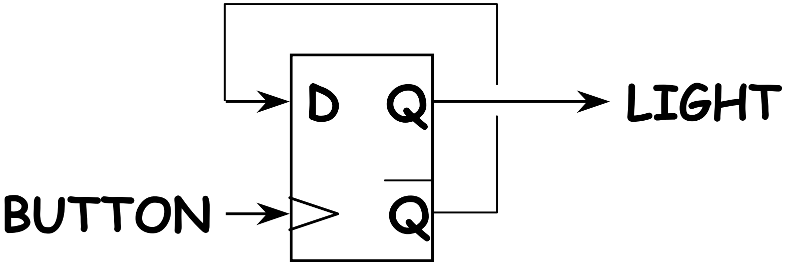
```
always @ (a or b or c)
begin
   x <= a & b;
   y <= x | c;
end
```

- Nonblocking assignments do not reflect the intrinsic behavior of multi-stage combinational logic
- While nonblocking assignments can be hacked to simulate correctly (expand the sensitivity list), it's not elegant
- **Guideline: use blocking assignments for combinational `always` blocks**

Implementation for on/off button



```
module onoff(button,light);
  input button;
  output light;
  reg light;
  always @ (posedge button)
  begin
    light <= ~light;
  end
endmodule
```

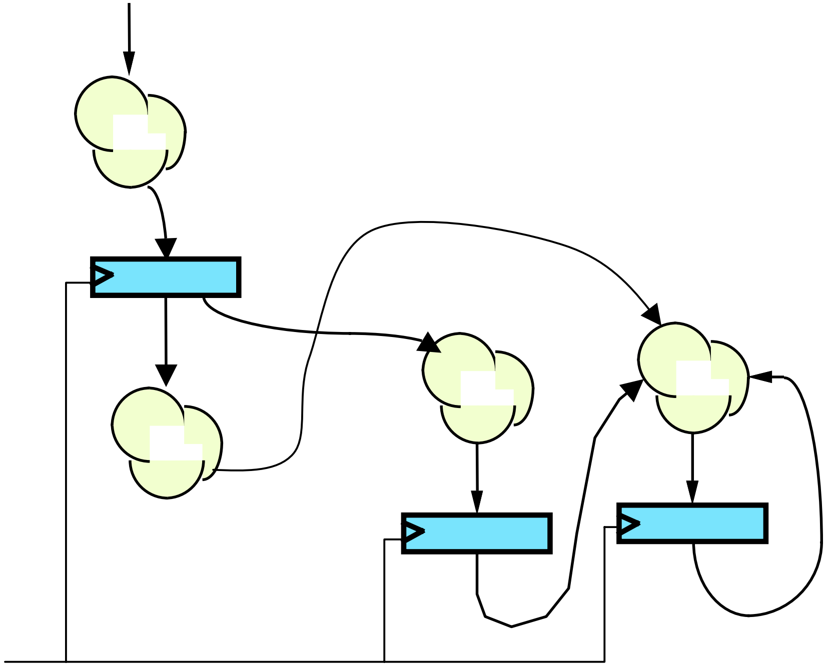


II. Single-clock Synchronous Circuits

We'll use Flip Flops and *Registers* – groups of FFs sharing a clock input – in a highly constrained way to build digital systems.

Single-clock Synchronous Discipline:

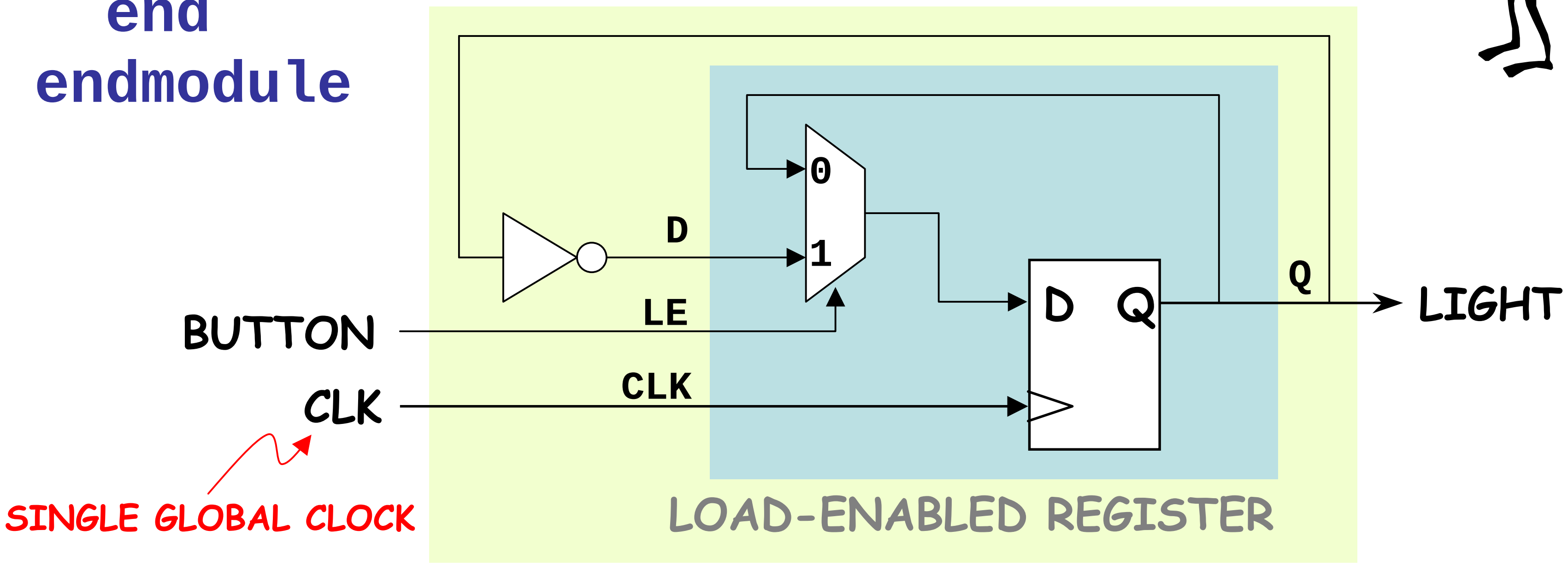
- **No combinational cycles**
- **Single clock signal shared among all clocked devices**
- **Only care about value of combinational circuits just before rising edge of clock**
- **Period greater than every combinational delay**
- **Change saved state after noise-inducing logic transitions have stopped!**



Clocked circuit for on/off button

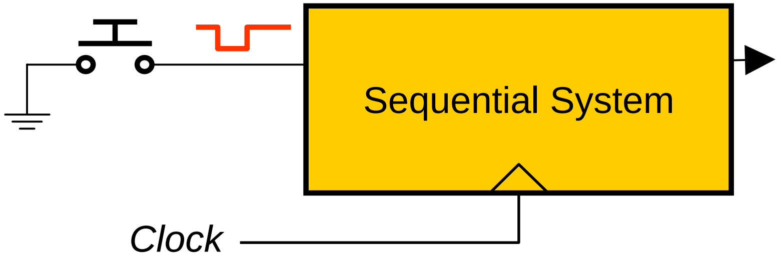
```
module onoff(clk,button,light);
  input clk,button;
  output light;
  reg light;
  always @ (posedge clk)
  begin
    if (button) light <= ~light;
  end
endmodule
```

Does this work with a 1Mhz CLK?



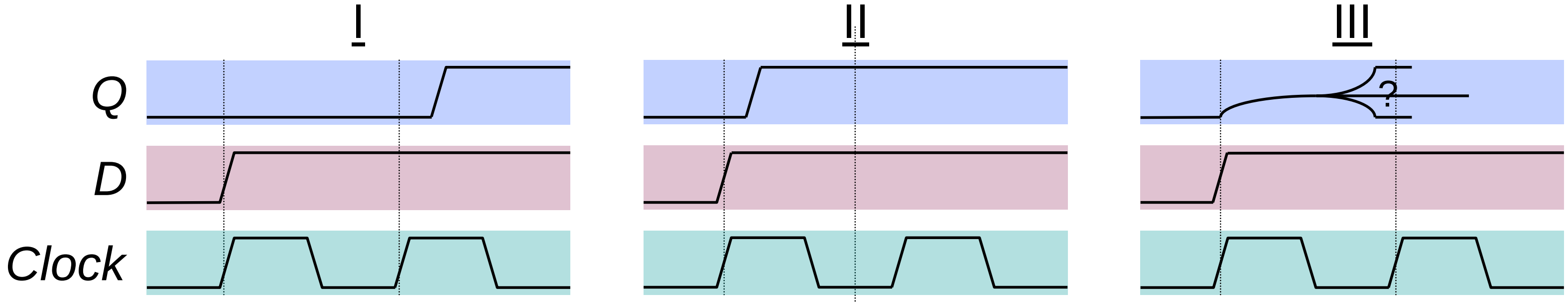
Asynchronous Inputs in Sequential Systems

What about external signals?



Can't guarantee setup and hold times will be met!

When an asynchronous signal causes a setup/hold violation...



Transition is missed on first clock cycle, but caught on next clock cycle.

Transition is caught on first clock cycle.

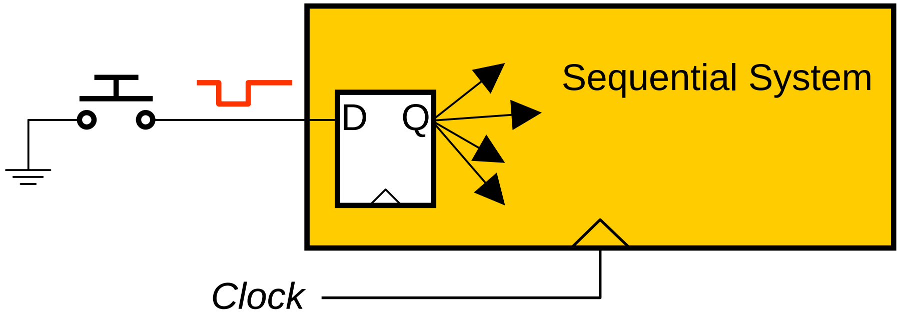
Output is metastable for an indeterminate amount of time.

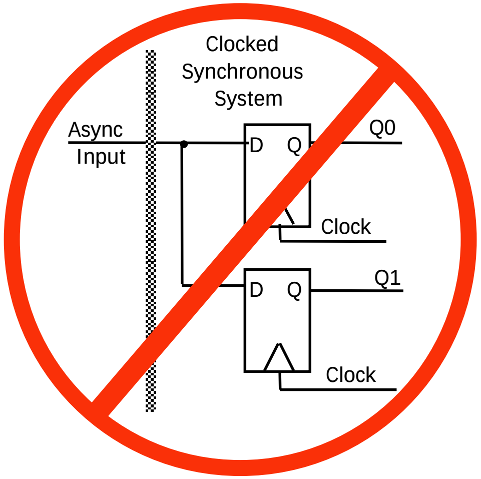
Q: Which cases are problematic?

Asynchronous Inputs in Sequential Systems

All of them can be, if more than one happens simultaneously within the same circuit.

Idea: ensure that external signals directly feed exactly one flip-flop

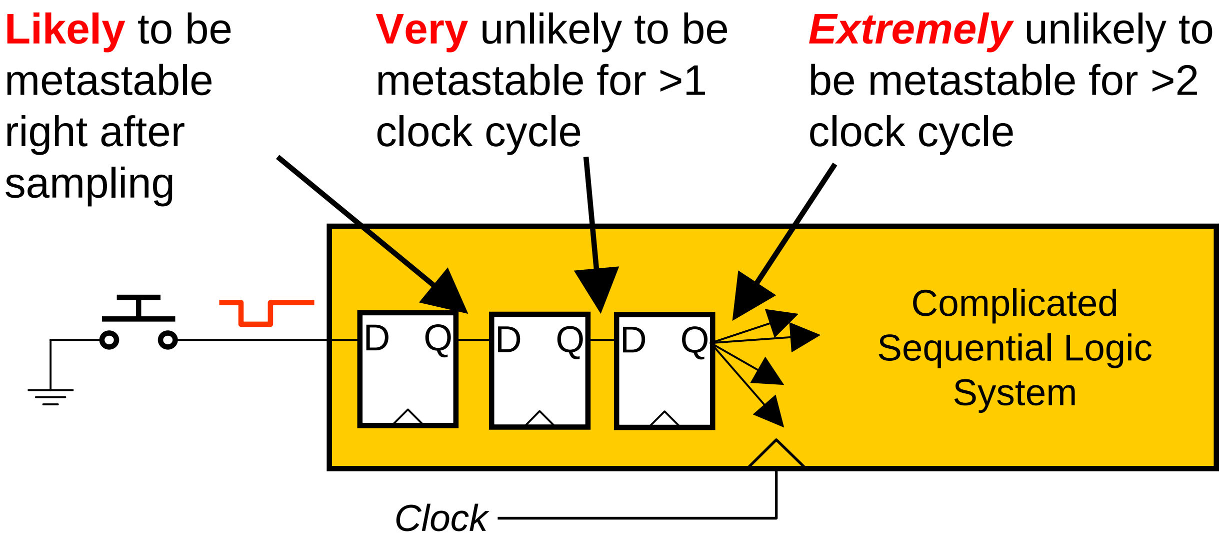




This prevents the possibility of I and II occurring in different places in the circuit, but what about metastability?

Handling Metastability

- Preventing metastability turns out to be an impossible problem
- High gain of digital devices makes it likely that metastable conditions will resolve themselves quickly
- Solution to metastability: allow time for signals to stabilize



Likely to be metastable right after sampling

Very unlikely to be metastable for >1 clock cycle

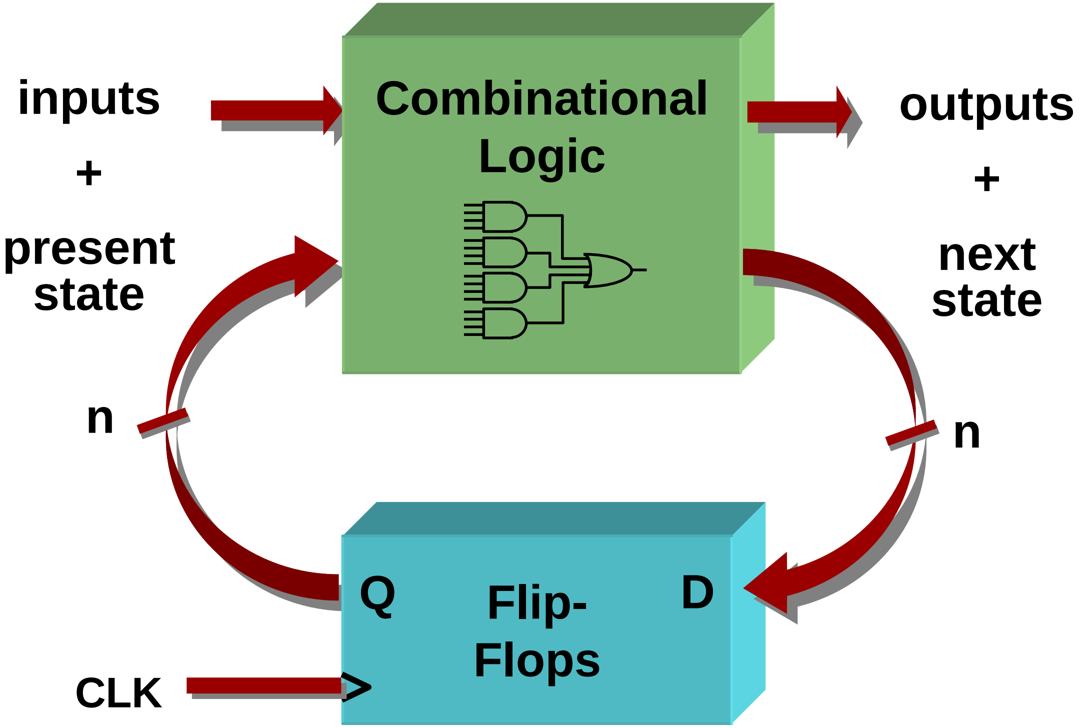
Extremely unlikely to be metastable for >2 clock cycle

How many registers are necessary?

- Depends on many design parameters(clock speed, device speeds, …)
- In 6.111, a pair of synchronization registers is sufficient

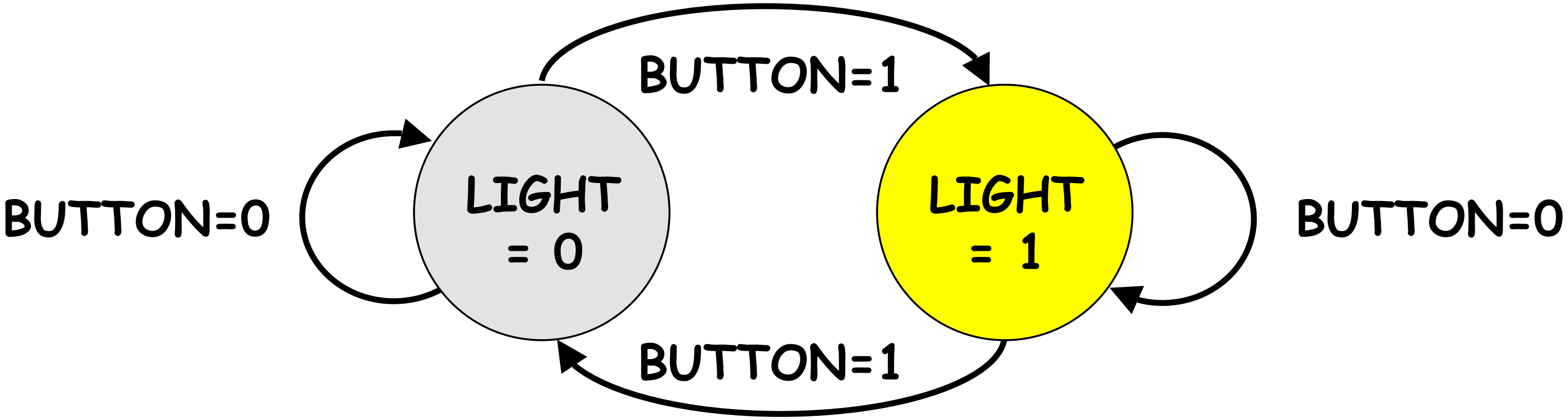
III. Finite State Machines

- Finite State Machines (FSMs) are a useful abstraction for sequential circuits with centralized "states" of operation
- At each clock edge, combinational logic computes *outputs* and *next state* as a function of *inputs* and *present state*

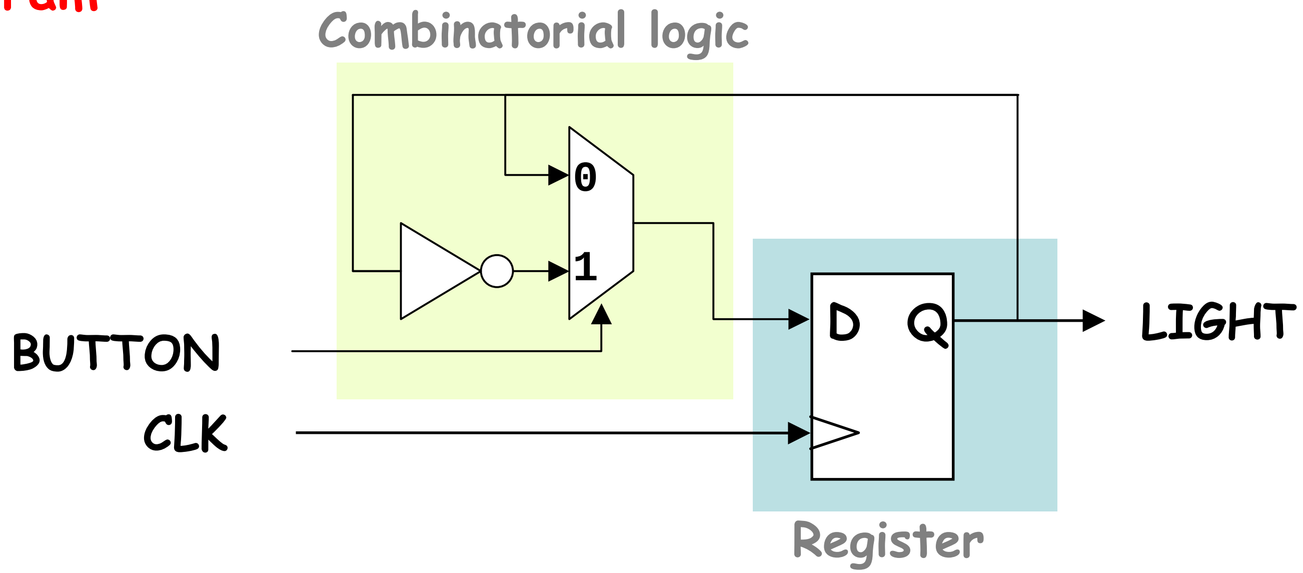


Example 1: Light Switch

- State transition diagram



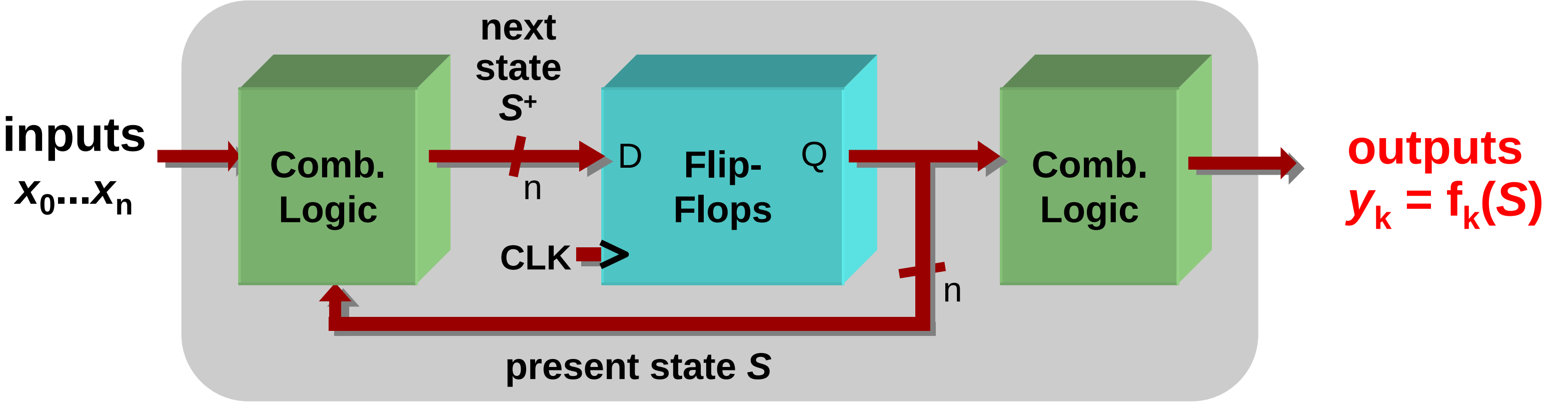
- Logic diagram



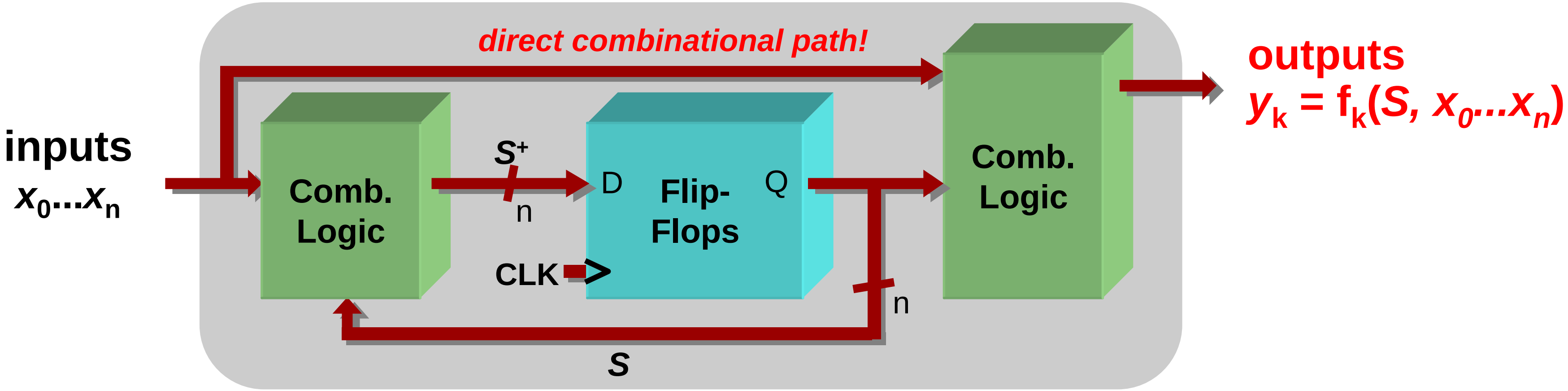
Two Types of FSMs

Moore and Mealy FSMs : different output generation

- **Moore FSM:**

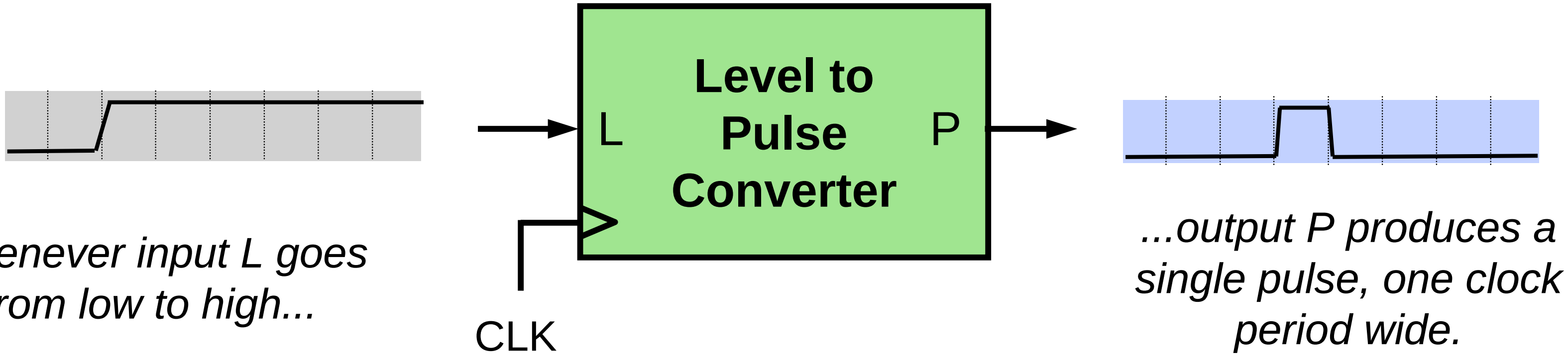


- **Mealy FSM:**



Design Example: Level-to-Pulse

- A level-to-pulse converter produces a single-cycle pulse each time its input goes high.
- It's a synchronous rising-edge detector.
- Sample uses:
  - Buttons and switches pressed by humans for arbitrary periods of time
  - Single-cycle enable signals for counters

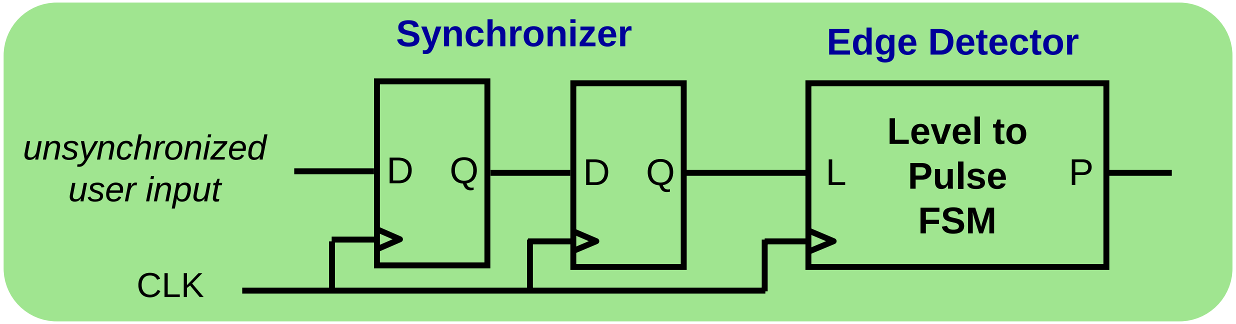


Whenever input L goes from low to high...

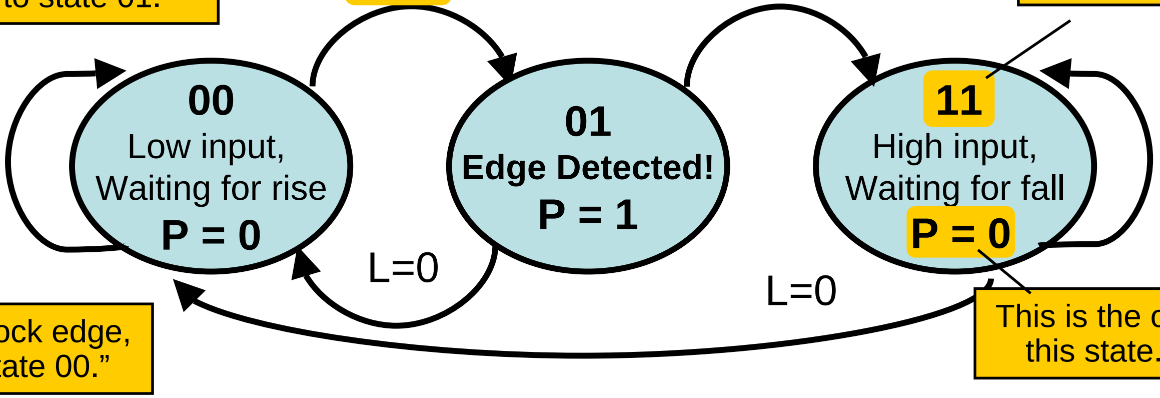
...output P produces a single pulse, one clock period wide.

Step 1: State Transition Diagram

- Block diagram of desired system:



- **State transition diagram** is a useful FSM representation and design aid:



"if L=1 at the clock edge, then jump to state 01."

Binary values of states

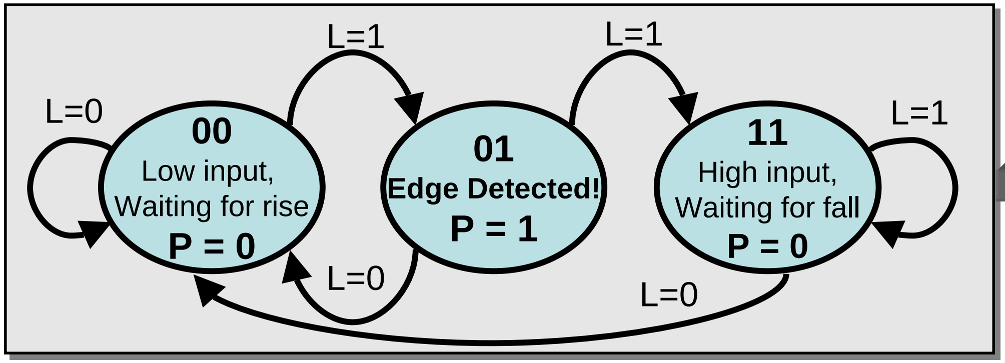
L=0

"if L=0 at the clock edge, then stay in state 00."

This is the output that results from this state. **(Moore or Mealy?)**

Step 2: Logic Derivation

Transition diagram is readily converted to a state transition table (just a truth table)



Current State		In	Next State		Out
S_1	S_0	L	S_1^+	S_0^+	P
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	1
1	1	0	0	0	0
1	1	1	1	1	0

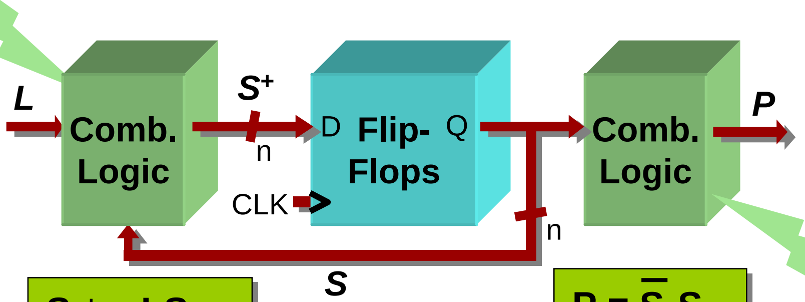
- Combinational logic may be derived using Karnaugh maps

for S_1^+:

$L \backslash S_1S_0$	00	01	11	10
0	0	0	0	X
1	0	1	1	X

for S_0^+:

$L \backslash S_1S_0$	00	01	11	10
0	0	0	0	X
1	1	1	1	X



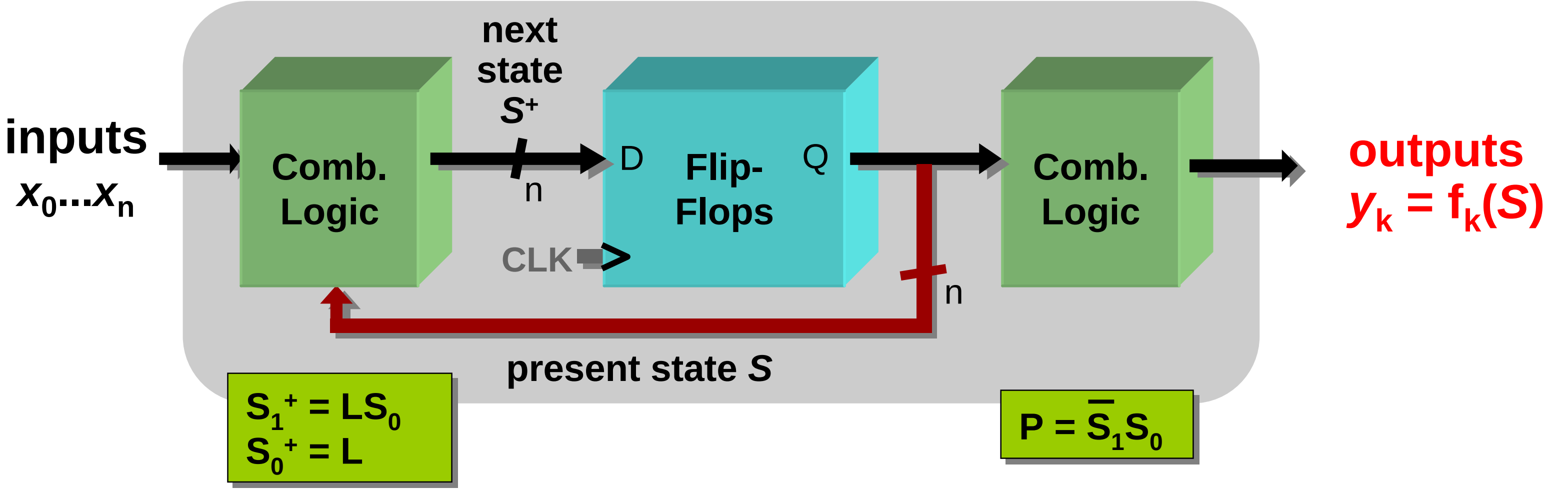
$S_1^+ = LS_0$
$S_0^+ = L$

$P = \overline{S_1}S_0$

for P:

$S_0 \backslash S_1$	0	1
0	0	X
1	1	0

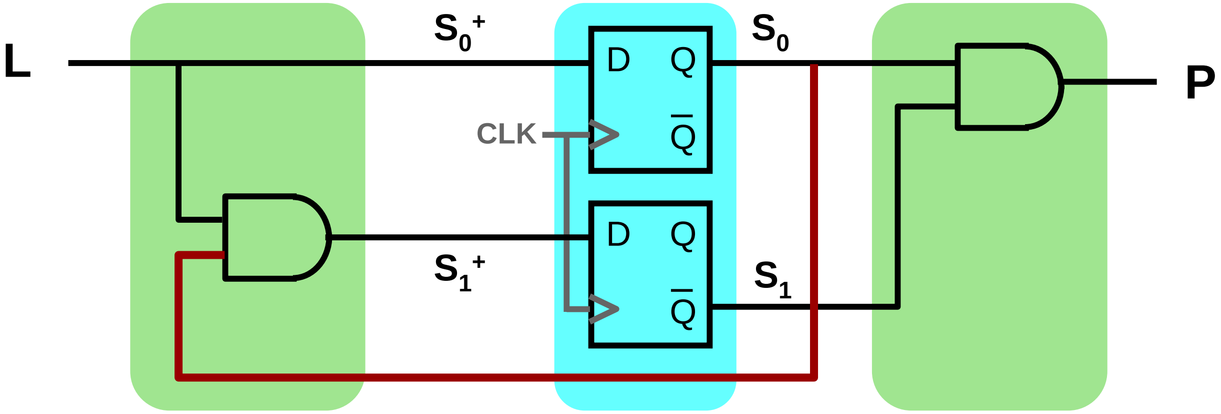
Moore Level-to-Pulse Converter



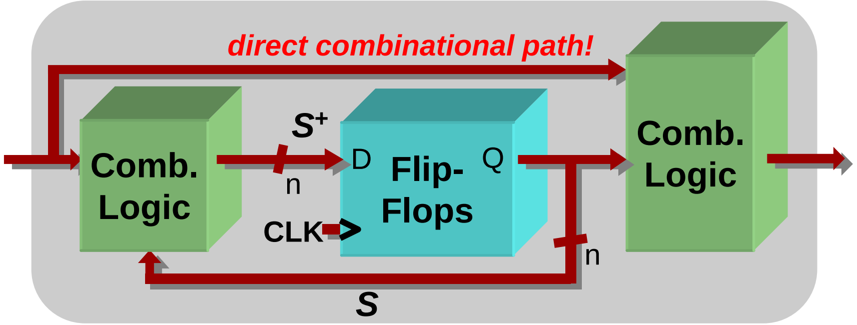
$S_1^+ = LS_0$
$S_0^+ = L$

$P = \overline{S_1}S_0$

Moore FSM circuit implementation of level-to-pulse converter:

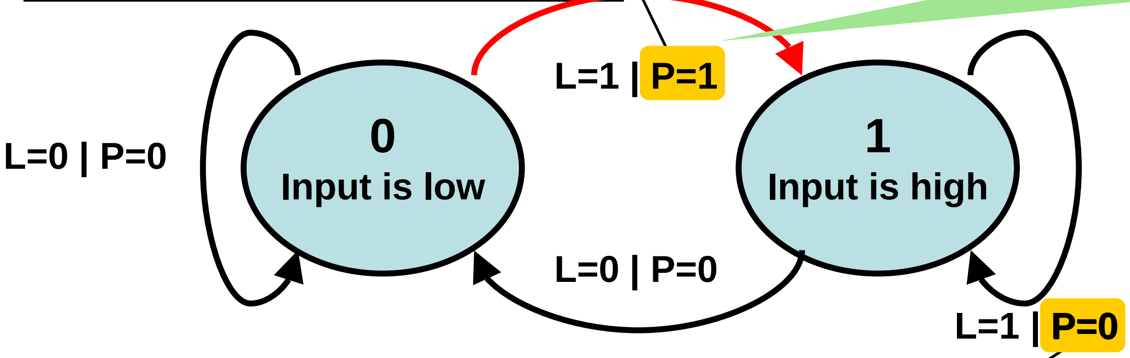


Design of a Mealy Level-to-Pulse

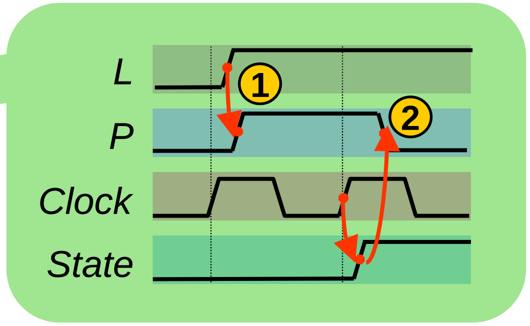


- Since outputs are determined by state *and* inputs, Mealy FSMs may need fewer states than Moore FSM implementations

1. When *L*=1 and *S*=0, this output is asserted **immediately** and **until** the state transition occurs (or *L* changes).

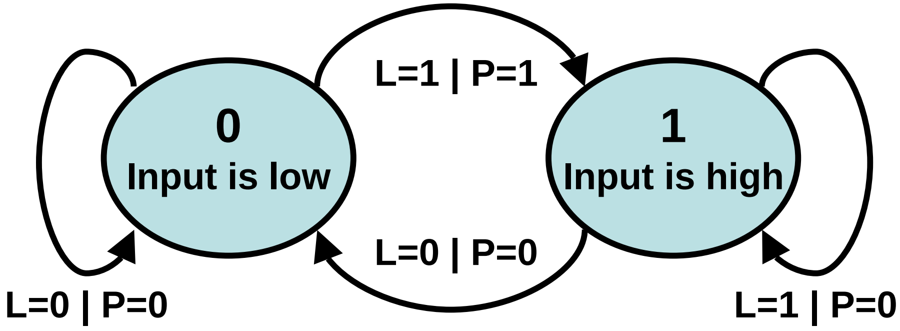


2. **After** the transition to S=1 and as long as *L* remains at *1*, *this* output is asserted.



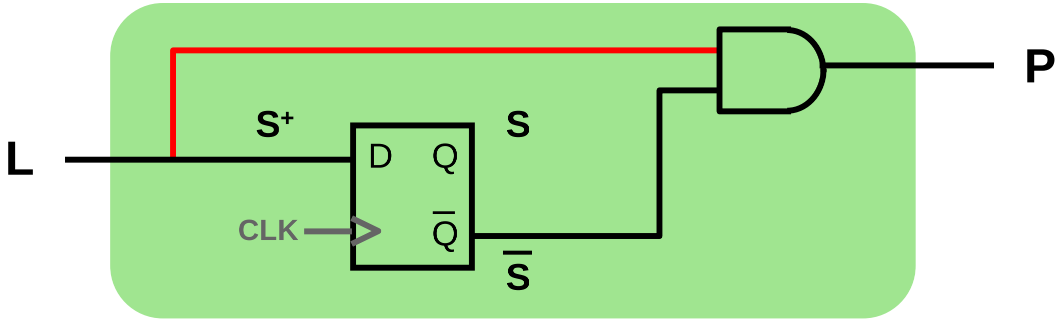
Output transitions immediately.
State transitions at the clock edge.

Mealy Level-to-Pulse Converter



Pres. State	In	Next State	Out
S	*L*	S^+	*P*
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	0

Mealy FSM circuit implementation of level-to-pulse converter:

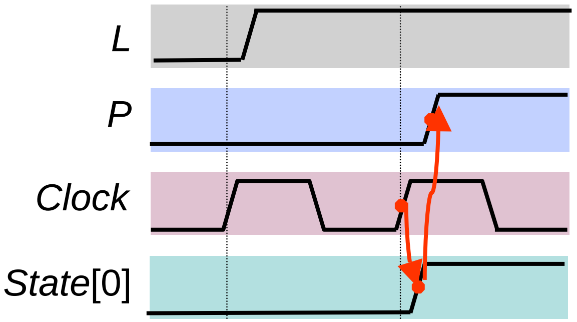


- FSM's state simply remembers the previous value of L
- Circuit benefits from the Mealy FSM's implicit single-cycle assertion of outputs during state transitions

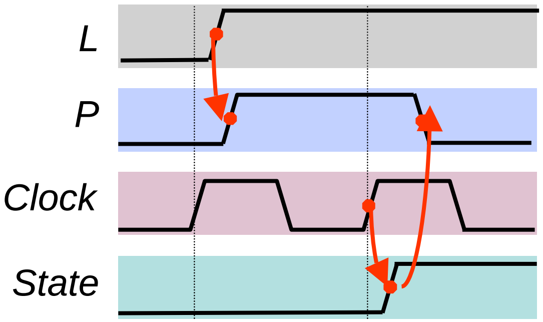
Moore/Mealy Trade-Offs

- How are they different?
  - Moore: outputs = f(state) only
  - Mealy outputs = f(state *and input*)
  - Mealy outputs generally occur one cycle earlier than a Moore:

Moore: delayed assertion of P



Mealy: immediate assertion of P

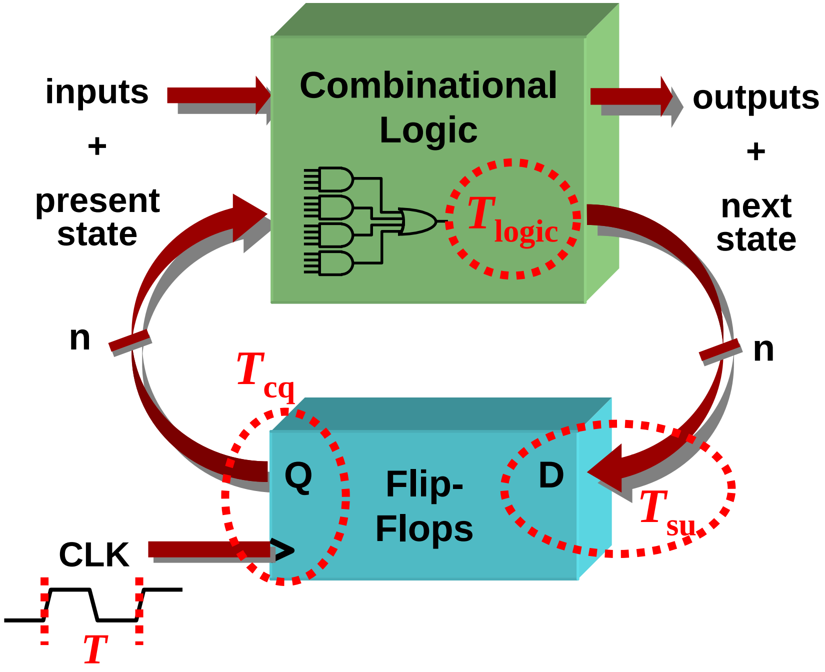


- Compared to a Moore FSM, a Mealy FSM might...
  - Be more difficult to conceptualize and design
  - Have fewer states

FSM Timing Requirements

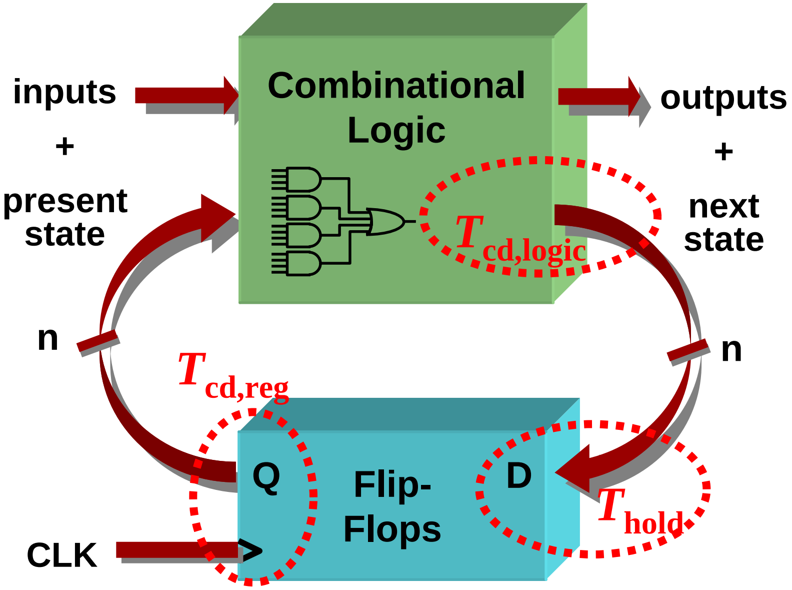
- Timing requirements for FSM are identical to any generic sequential system with feedback

Minimum Clock Period



$$T > T_{cq} + T_{logic} + T_{su}$$

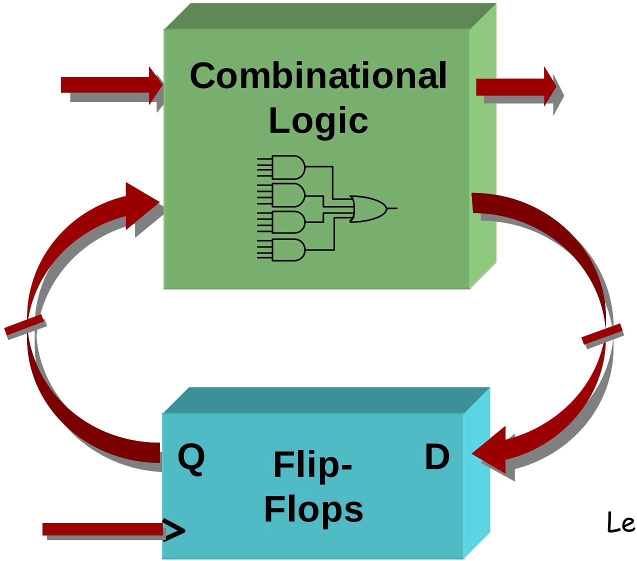
Minimum Delay



$$T_{cd,reg} + T_{cd,logic} > T_{hold}$$

Summary

- Assignments in always blocks:
  - blocking ("=") for combinational logic
  - non-blocking ("<=") for sequential logic
- Single-clock Synchronous discipline:
  - Reliable digital circuits / systems
  - Global clock to edge-triggered registers
- Finite state machines:
  - Programmable systems
  - Moore & Mealy


Combinational Logic
Q
Flip-Flops
D