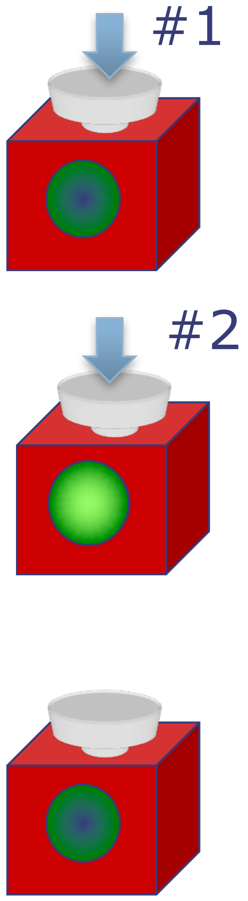
Sequential Circuits

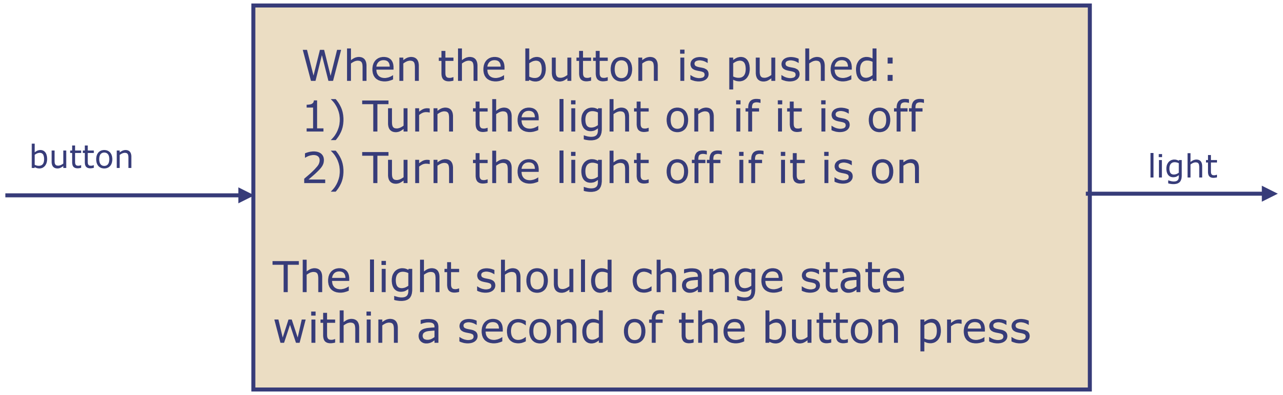
And Finite State Machines

Lab 1 checkoff due tomorrow (9/27)
Lab 2 due on Thu (9/28)
Quiz 1 happening on 10/5

Something We Cannot Build (Yet)

What if you were given the following design specification:





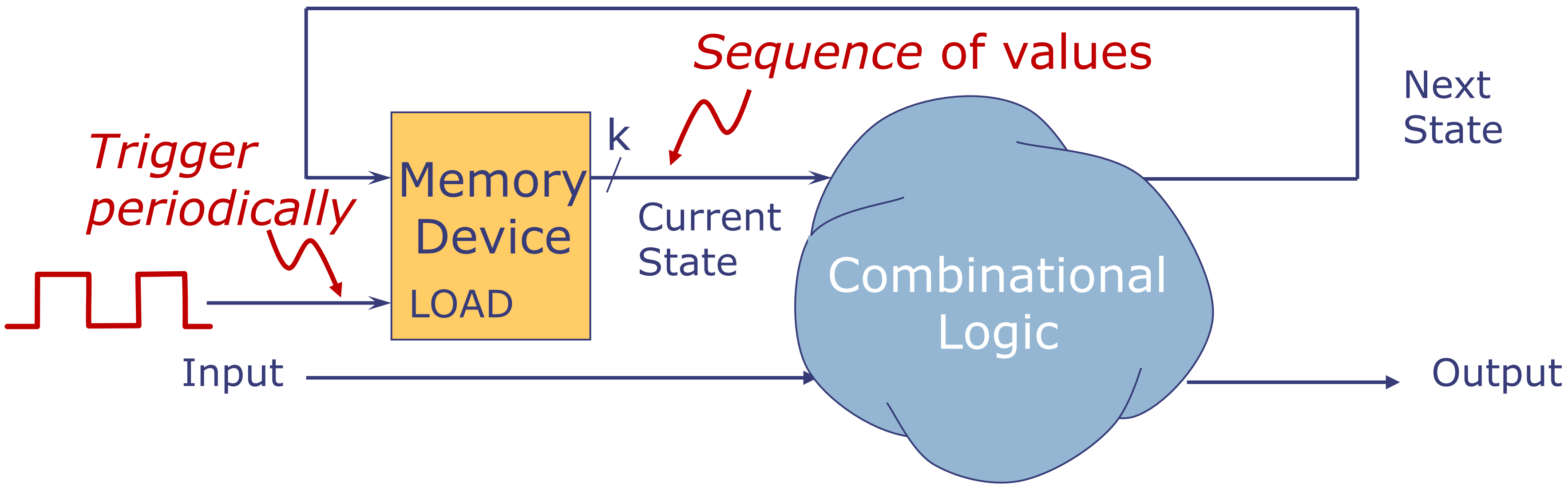
When the button is pushed:
1) Turn the light on if it is off
2) Turn the light off if it is on

The light should change state within a second of the button press

What makes this device different from those we've discussed before?

State – i.e., the device has memory

Sequential Circuits



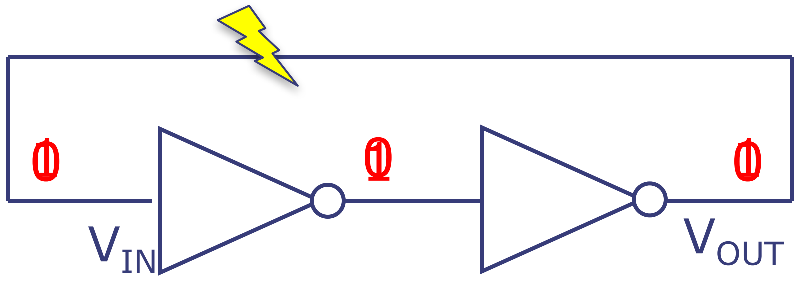
Sequential circuits contain digital memory and combinational logic

- Memory stores current state
- Combinational logic computes:
    - Next state (from input + current state)
    - Output bits (from input + current state)
- State changes on LOAD control input

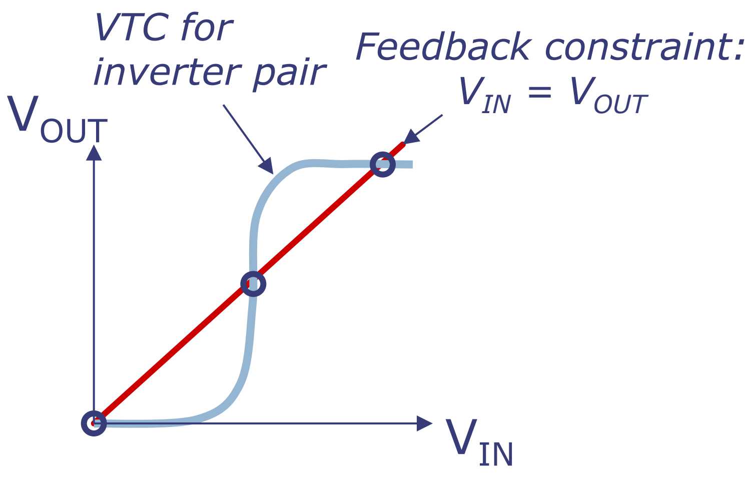
Need loadable memory

Memory: Using Feedback

Idea: use feedback to maintain storage indefinitely.
Our logic gates are built to restore marginal signal levels, so noise shouldn't be a problem!



Result: a bistable storage element



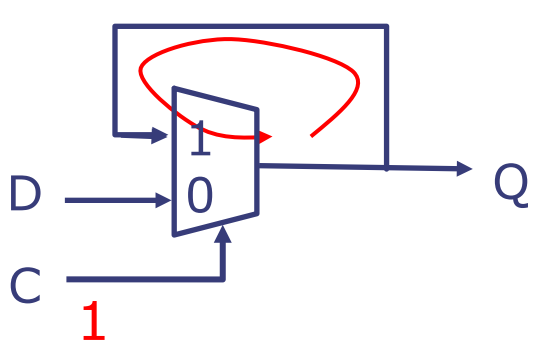
Not affected by noise

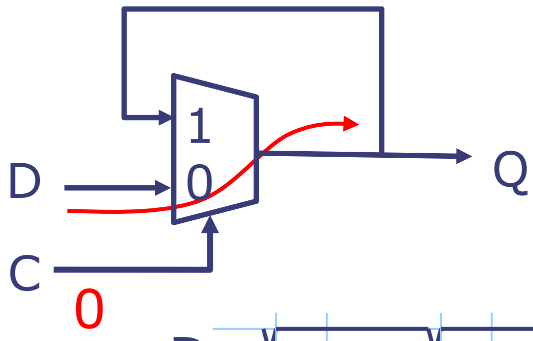
Three solutions:
- two end-points are stable
- middle point is metastable

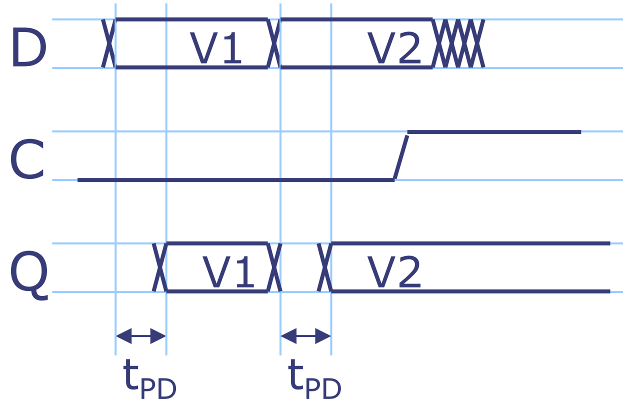
We'll get back to this!

D Latch

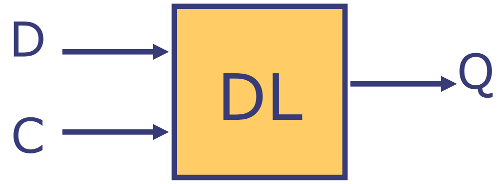
A simple circuit that can hold state







if C=1, the value of Q *holds*
if C=0, the value of D *passes* to Q

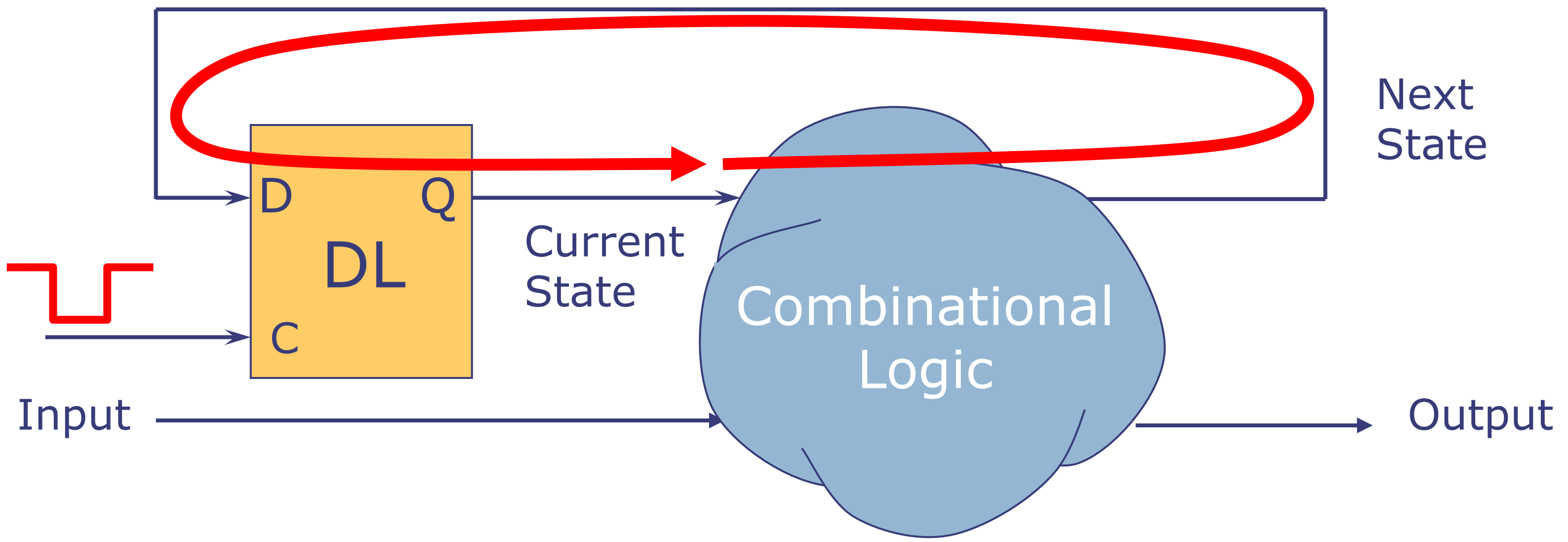


C	D	Q^{t-1}	Q^t	
0	0	X	0	pass
0	1	X	1	
1	X	0	0	hold
1	X	1	1	

Q^{t-1} represents the value previously held in DL;
Q^t represents the current value.

Sequential circuits using D latches?

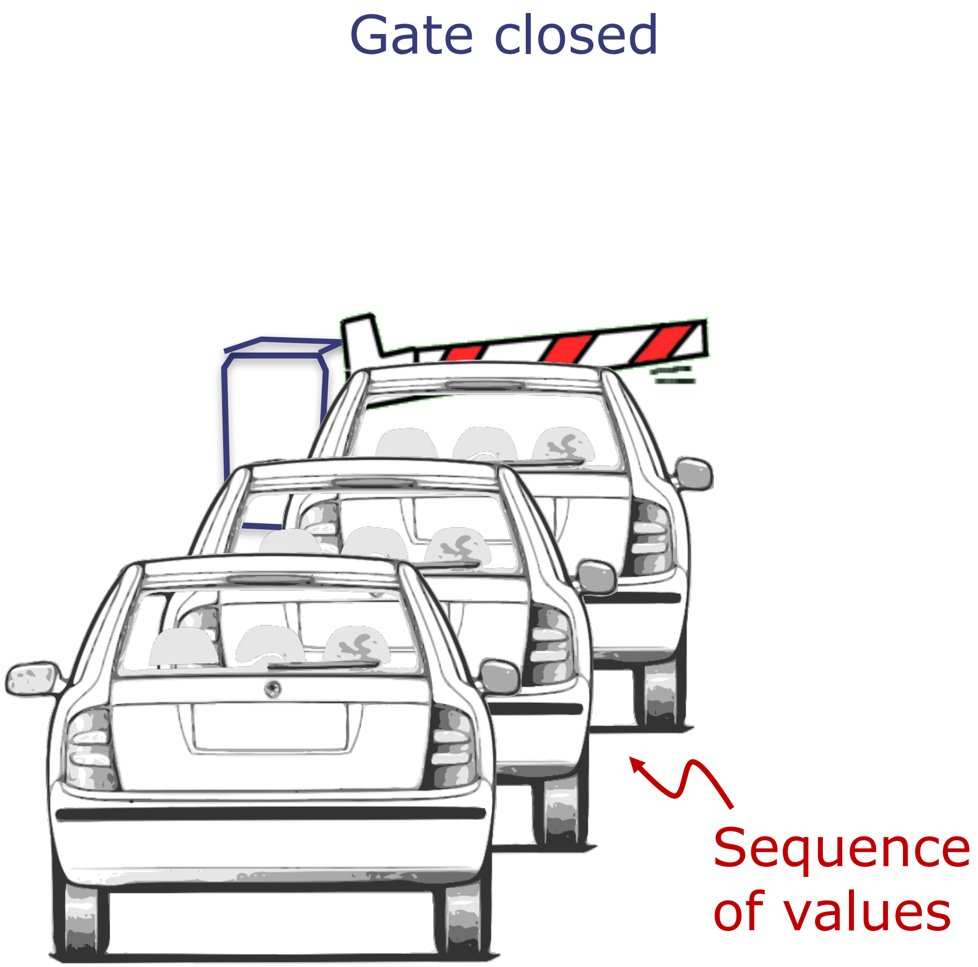
A terrible idea



- When C=0, D latch passes input to output…
  - Creates a cycle from Q to D!
  - Our combinational logic stops being combinational ☹
- In practice, very hard to get right: Needs tricky timing constraints on C=0 pulse + comb logic

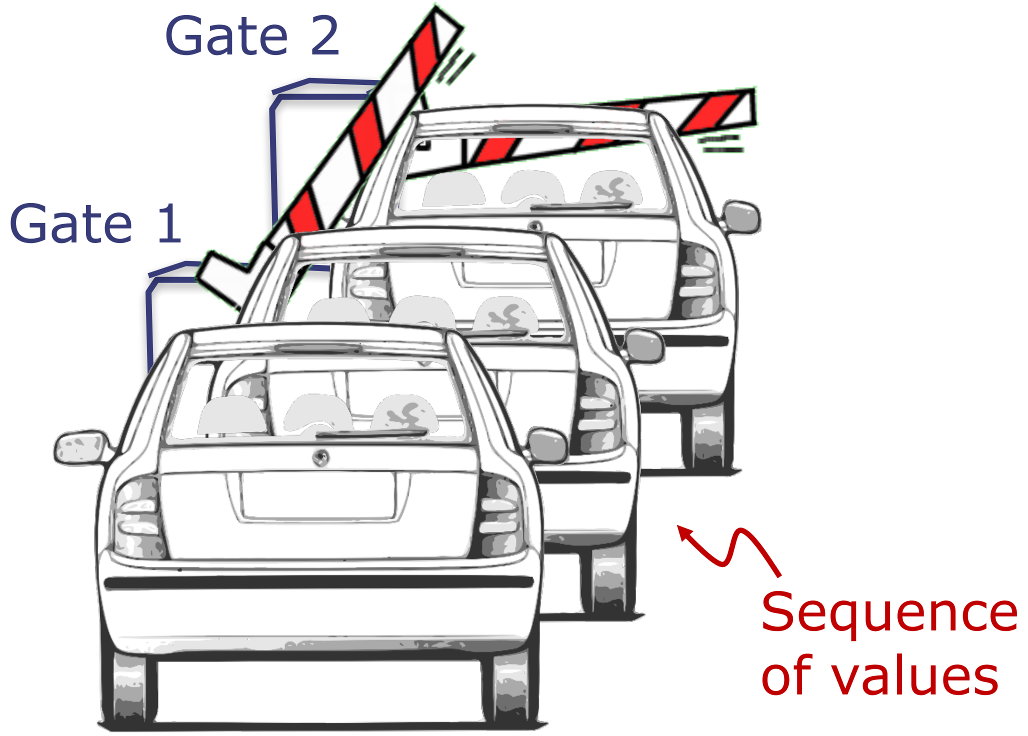
Memory should sample an *instant*, not an *interval*

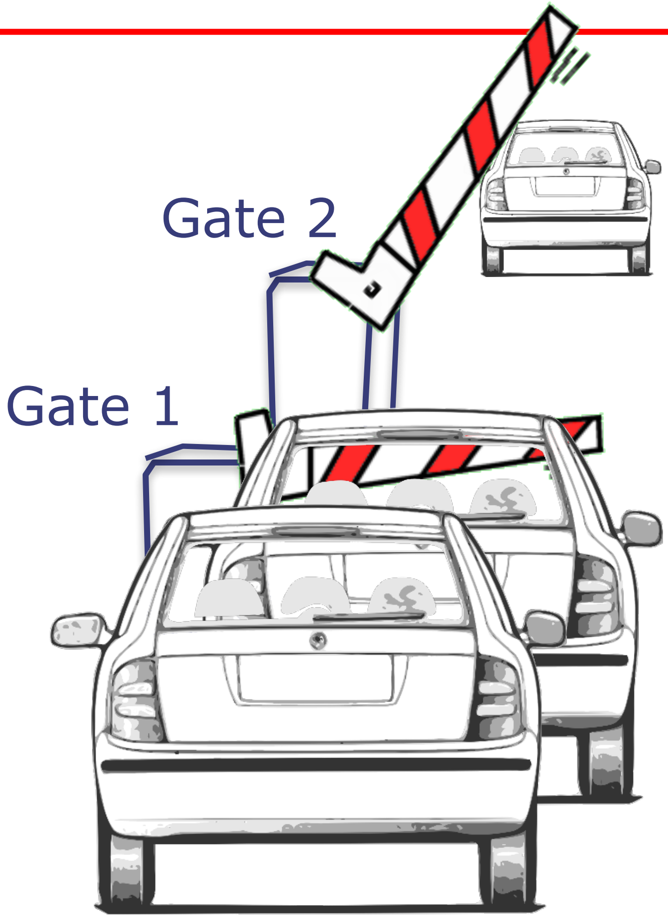
A Similar Problem…



How can we ensure only one car gets through?

Solution: Use two gates!



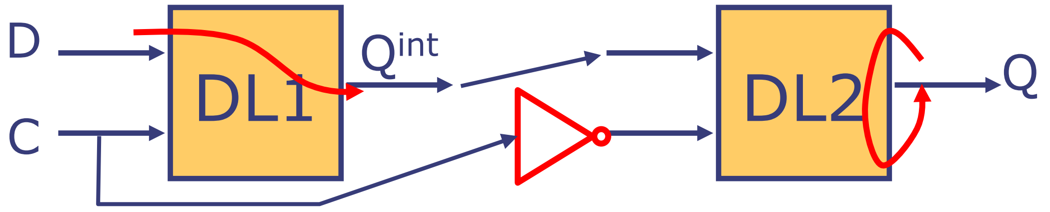


Gate 1: open
Gate 2: closed

Gate 1: closed
Gate 2: open

D Flip-Flop

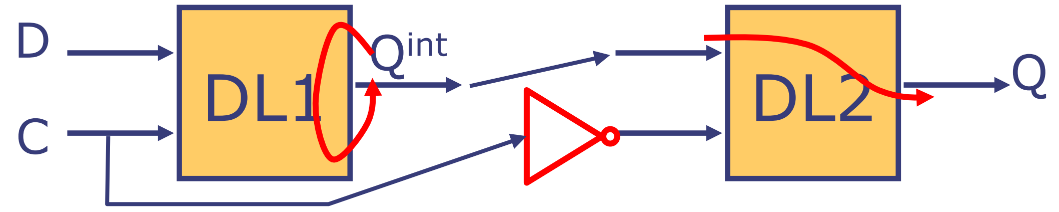
C=0



- Two latches driven by inverted C signals, one is always holding, and one is always passing
- How does this circuit behave?
  - C = 0: Q^{int} follows the input D, but Q holds its old value

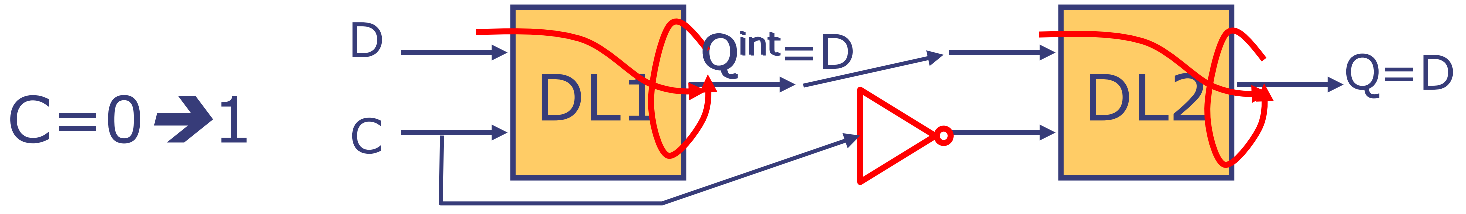
D Flip-Flop

C=1



- Two latches driven by inverted C signals, one is always holding, and one is always passing
- How does this circuit behave?
  - C = 0: Q^{int} follows the input D, but Q holds its old value
  - C = 1: Q^{int} holds its old value, but Q follows Q^{int}
  - Q doesn't change when C=0 or C=1
  - It changes when C transitions from 0 to 1 (a *rising-edge* of C)

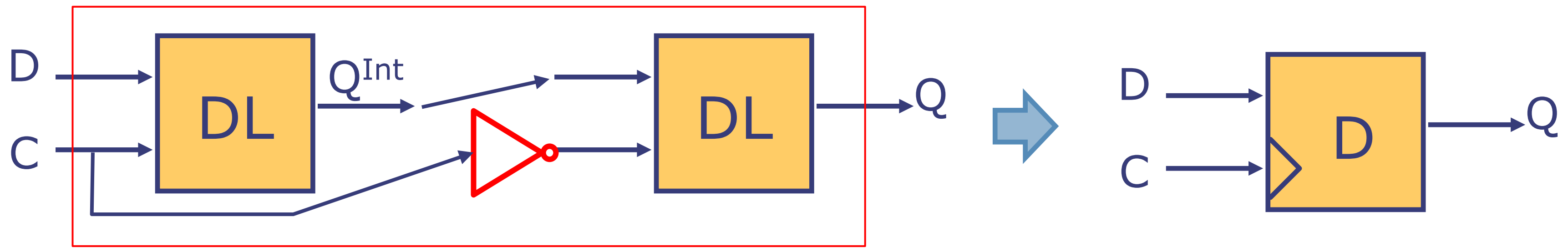
D Flip-Flop



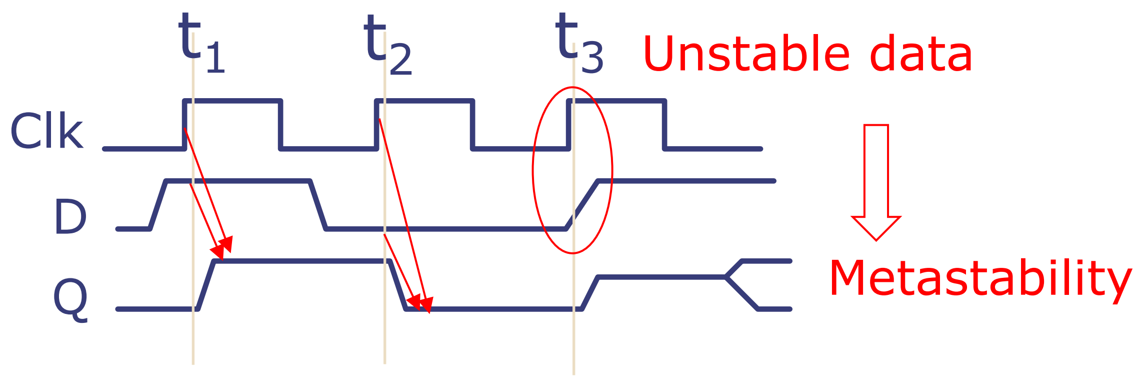
- Two latches driven by inverted C signals, one is always holding, and one is always passing
- How does this circuit behave?
  - C = 0: Q^{int} follows the input D, but Q holds its old value
  - C = 1: Q^{int} holds its old value, but Q follows Q^{int}
  - Q doesn't change when C=0 or C=1
  - It changes when C transitions from 0 to 1 (a *rising-edge* of C)

What happens on a falling edge (C: 1 ➔ 0)?

D Flip-Flop
An *edge-triggered* storage element

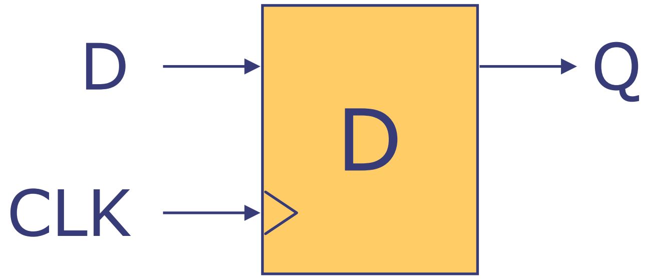


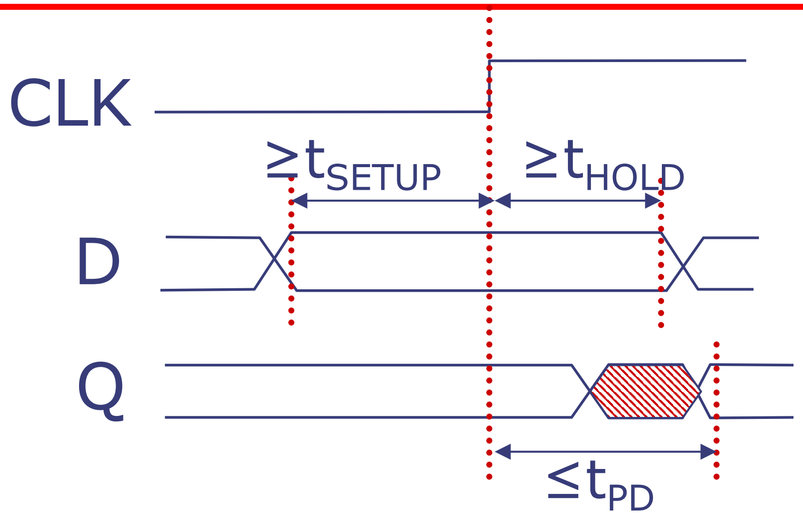
C changes periodically (a *Clock* signal)



Data is sampled at the rising edge of the clock and must be ***stable*** *at that time*

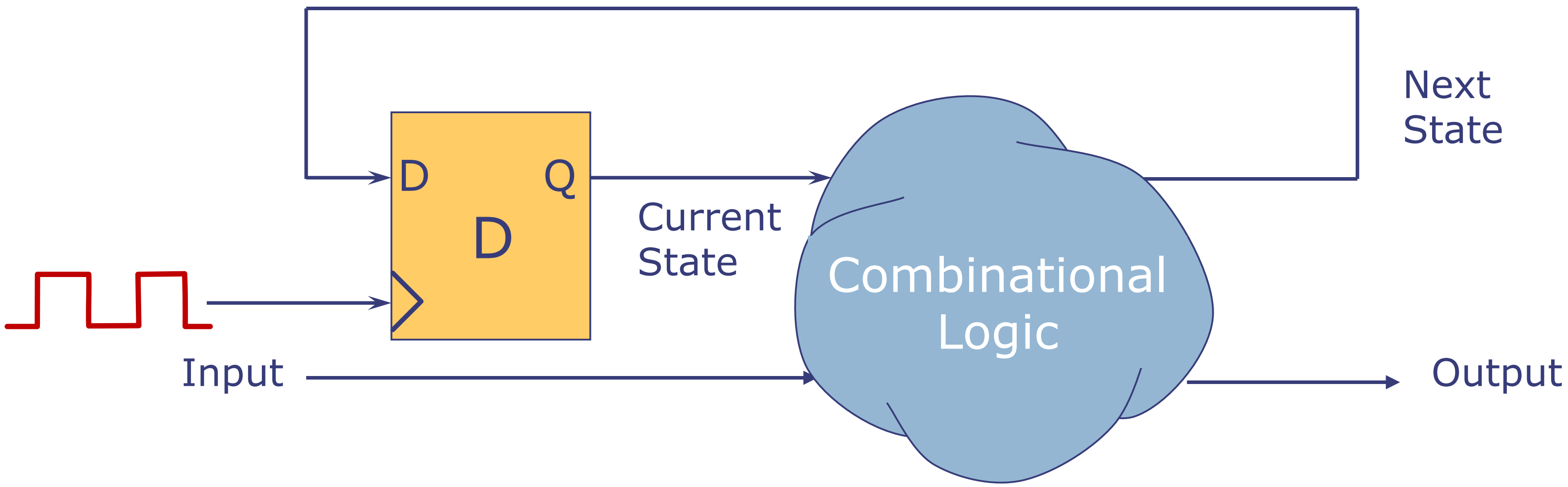
D Flip-Flop Timing





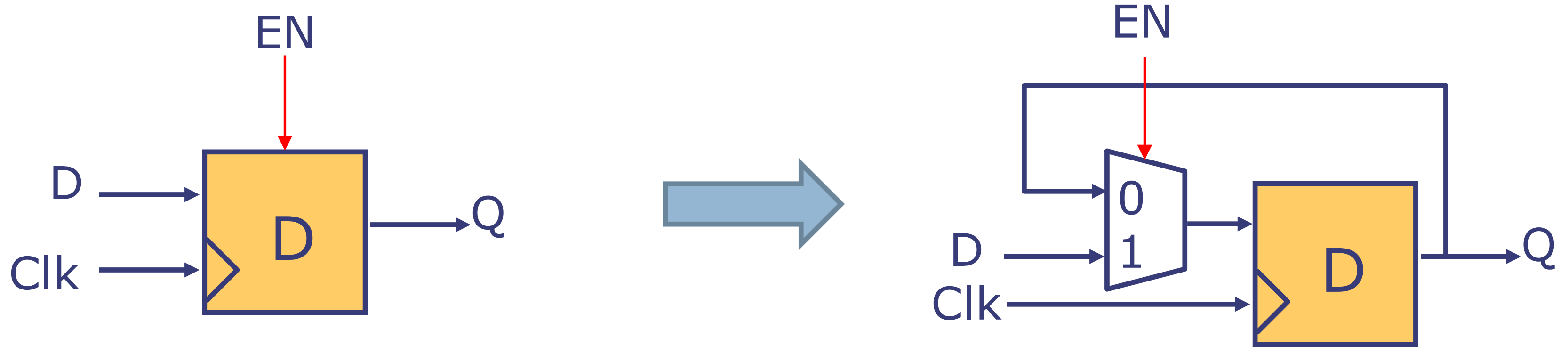
- Flip-flop input D should not change around the rising edge of the clock to avoid *metastability*
- Formally, D should be a stable and valid digital value:
  - For at least t_{SETUP} before the rising edge of the clock
  - For at least t_{HOLD} after the rising edge of the clock
- Flip-flop propagation delay t_{PD} is measured from rising edge of the clock to valid output (CLK→Q)

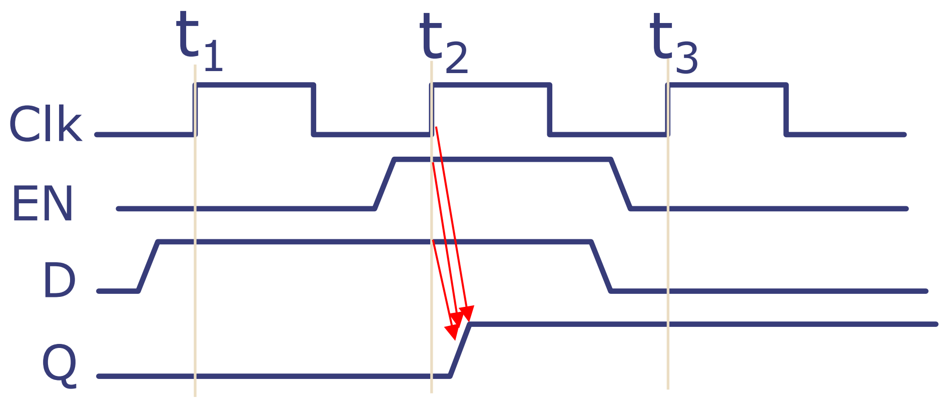
Sequential Circuits using D Flip-Flops



- There is never a combinational cycle between D and Q!
- Works correctly, as long as we meet t_{SETUP} and t_{HOLD}
  - More on this later

D Flip-Flop with Write Enable



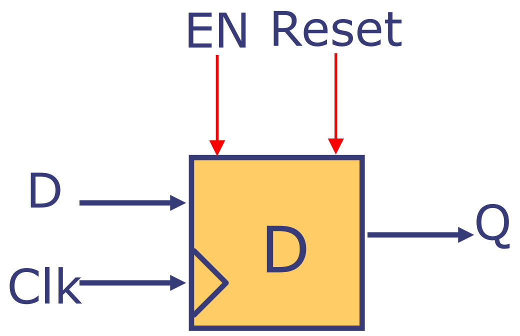


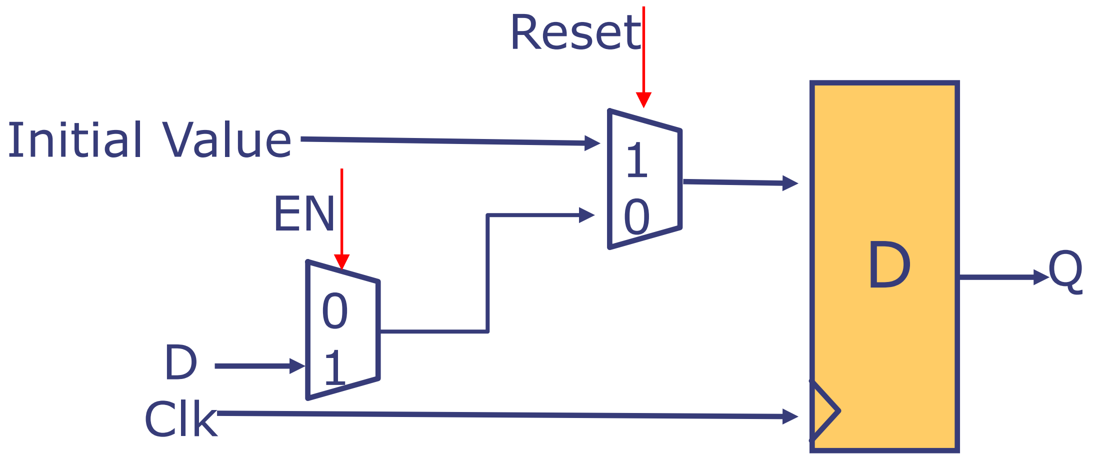
EN	D	Q^t	Q^{t+1}	
0	X	0	0	hold
0	X	1	1	
1	0	X	0	copy input
1	1	X	1	

Data is captured only if EN is on

Transitions happen at rising edge of the clock
No need to specify the clock explicitly

How Are Flip-Flops Initialized?





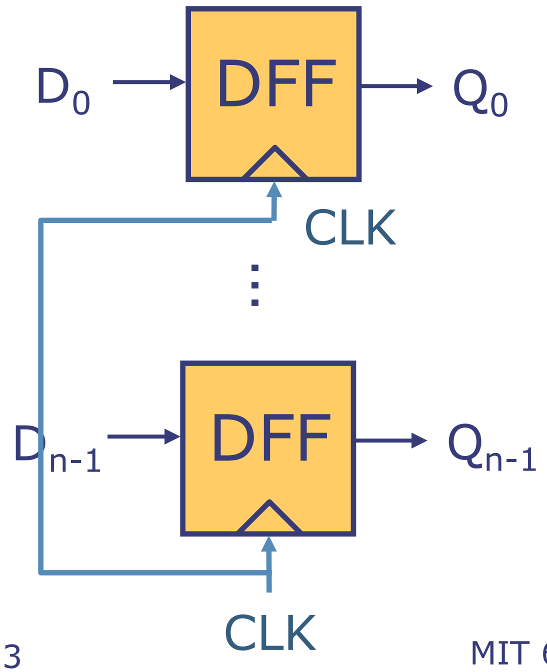
- When Reset = 1, flip flop is set to initial value regardless of value of EN

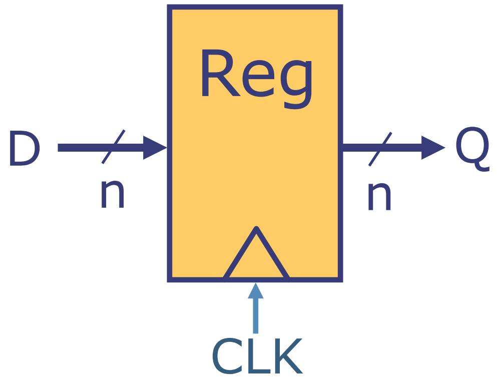
- When Reset = 0, then it behaves like a D flip-flop with enable

Using D Flip-Flops to Build Sequential Circuits

Synchronous Sequential Circuits

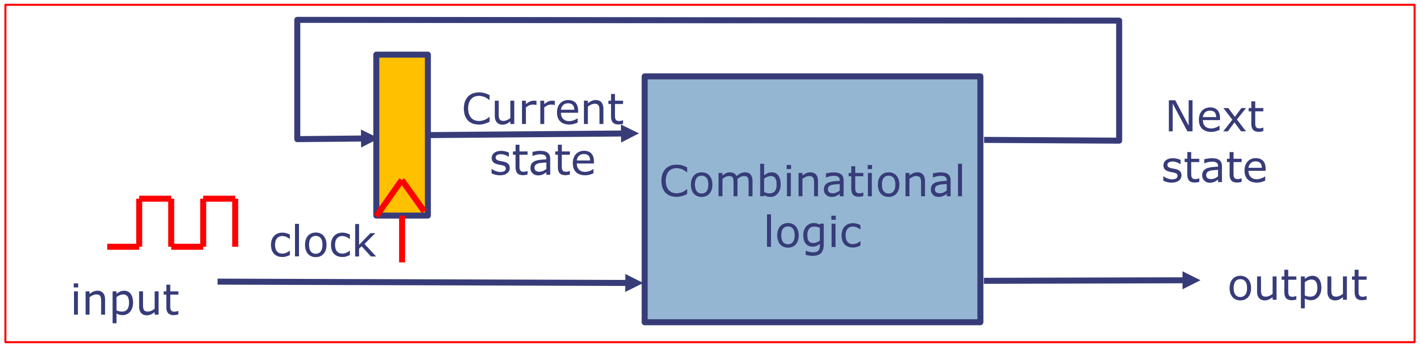
- All the D Flip-Flops use the same periodic clock signal
- Register: Group of DFFs
  - Stores multi-bit values
- Registers update their contents simultaneously, at the rising edge of the clock





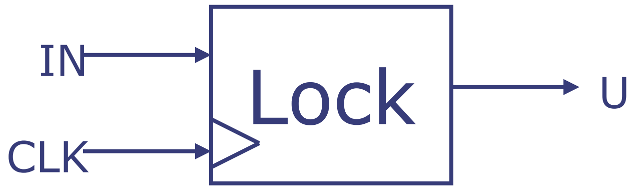
Synchronous Sequential Circuits

- Synchronous sequential circuits: All state kept in registers driven by the same clock
- This allows discretizing time into cycles and abstracting sequential circuits as finite state machines (FSMs)
- FSMs can be described with state-transition diagrams or truth tables



A Simple Sequential Circuit

Let's make a digital binary *Combination Lock:*

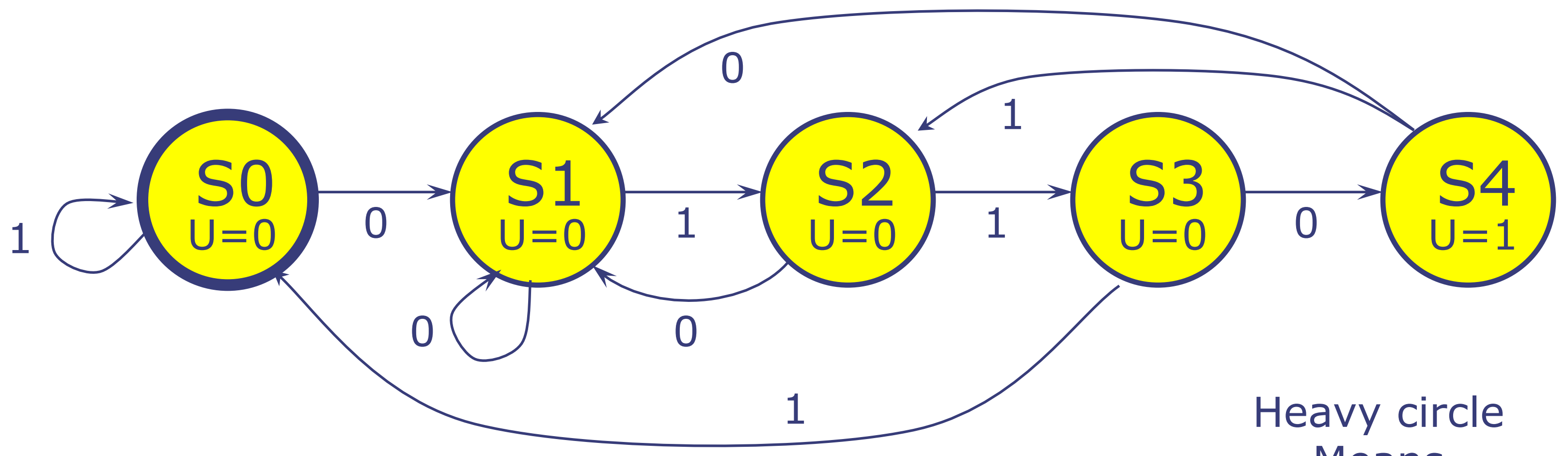


Specification:

- A 1-bit input ("0" or "1")
- A 1-bit output ("Unlock" signal)
- UNLOCK is 1 if and only if:
  Last 4 inputs were the "combination": 0110

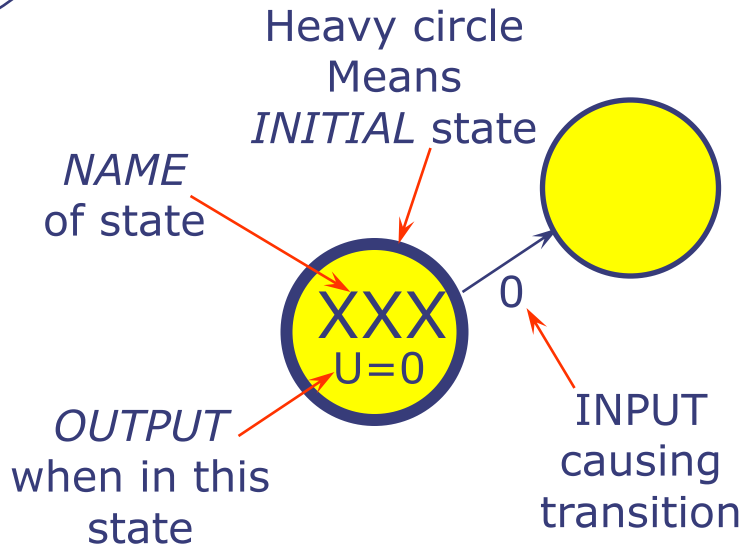
How many states do we need?

State-Transition Diagram



Designing our lock

- Need an initial state; call it S0.
- Must have a separate state for each step of the proper entry sequence
- Must handle other (erroneous) entries


Heavy circle Means *INITIAL* state
NAME of state
XXX
U=0
0
OUTPUT when in this state
INPUT causing transition

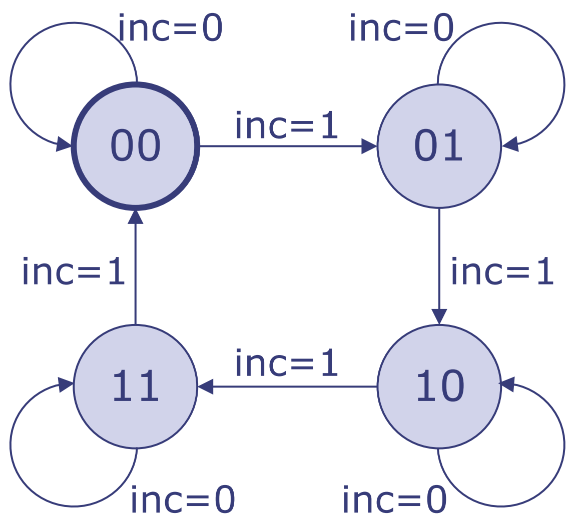
Valid State-Transition Diagrams

- Arcs leaving a state must be:

  (1) mutually exclusive
  - *For a given input value, can't have two choices*

  (2) collectively exhaustive
  - *Every state must specify what happens for each possible input combination*
  - *"Nothing happens" means arc back to itself*

Two-Bit Counter
An Example



State-Transition Diagram

Current State			Next State	
inc	$q1^t$	$q0^t$	$q1^{t+1}$	$q0^{t+1}$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

$$q0^{t+1} = \sim inc \cdot q0^t + inc \cdot \sim q0^t$$
$$= inc \oplus q0^t$$
$$q1^{t+1} = \sim inc \cdot q1^t + inc \cdot \sim q1^t \cdot q0^t + inc \cdot q1^t \cdot \sim q0^t$$
$$= \sim inc \cdot q1^t + inc \cdot (q1^t \oplus q0^t)$$

Output = current state = q1q0

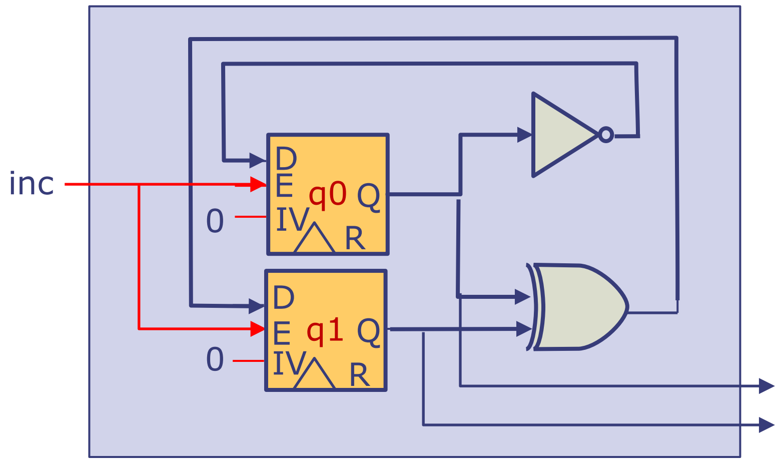
Two-Bit Counter Circuit
Using D flip-flops

$$q0^{t+1} = \sim inc \cdot q0^{t} + inc \cdot \sim q0^{t}$$
$$q1^{t+1} = \sim inc \cdot q1^{t} + inc \cdot (q1^{t} \oplus q0^{t})$$

- Use two D flip-flops with *reset* and *enable* to store q0 and q1

$$\left.\begin{aligned} q0^{t+1} &= \sim q0^{t} \\ q1^{t+1} &= q1^{t} \oplus q0^{t} \end{aligned}\right\} inc=1$$

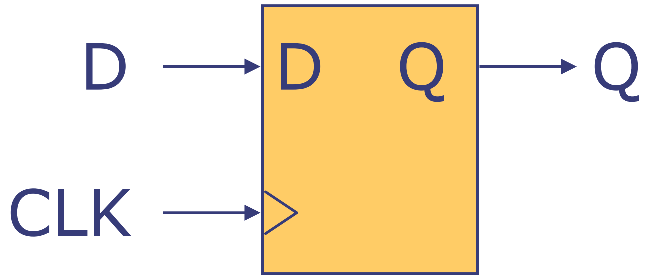
- The state only changes when inc is 1. Let's connect inc to ***EN*** and simplify the equations

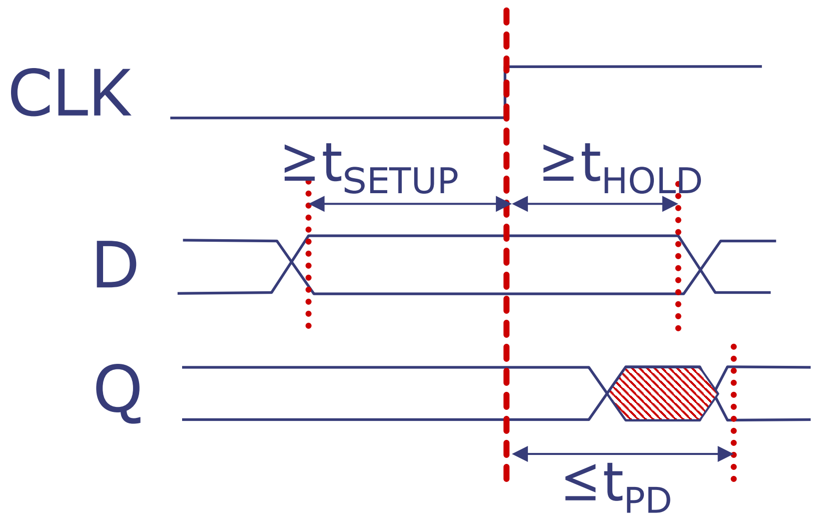


- Output is q1q0
- Set Initial Value of both flip-flops to 0 (initial state: 00)
  - Loaded when Reset = 1

Timing Constraints in Sequential Circuits

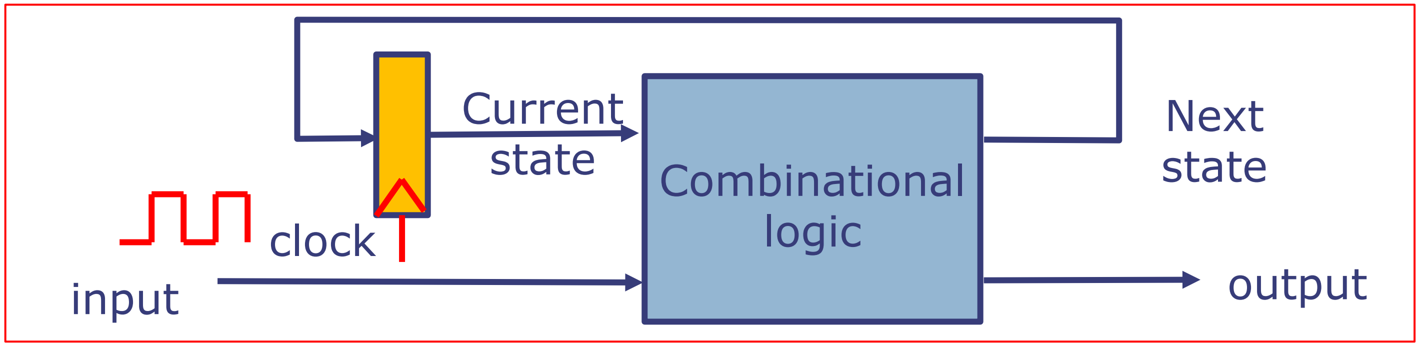
Recap: D Flip-Flop Timing

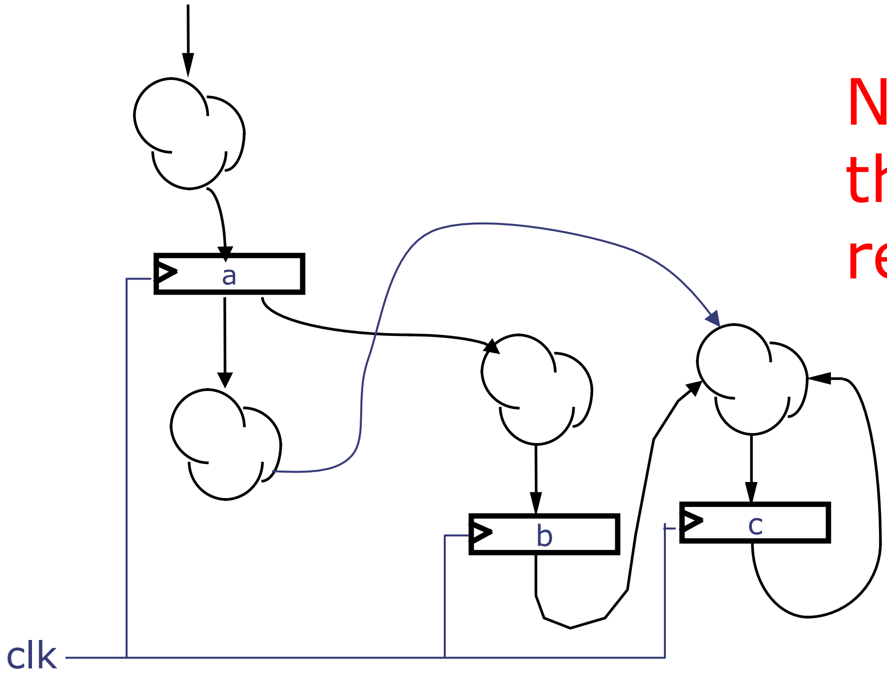




- Flip-flop input D should not change around the rising edge of the clock to avoid *metastability*
- Formally, D should be a stable and valid digital value:
  - For at least t_{SETUP} before the rising edge of the clock
  - For at least t_{HOLD} after the rising edge of the clock
- Flip-flop propagation delay t_{PD} is measured from rising edge of the clock to valid output (CLK→Q)

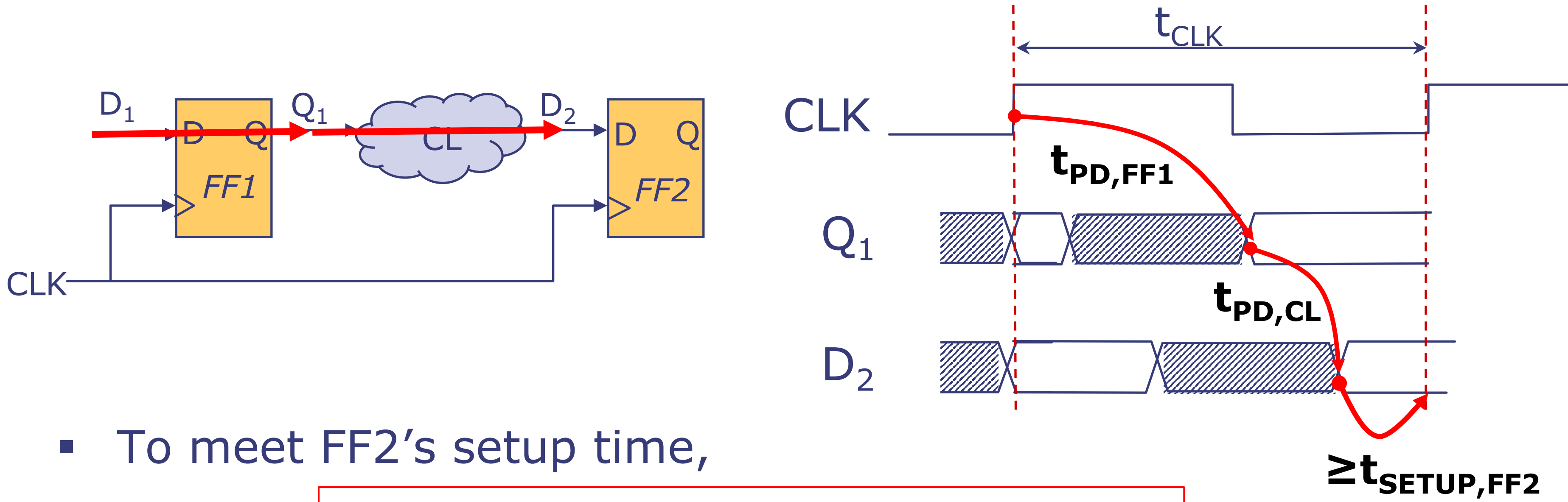
Single-clock Synchronous Circuits





Need to analyze the timing of each register-to-register path

Meeting the Setup-Time Constraint

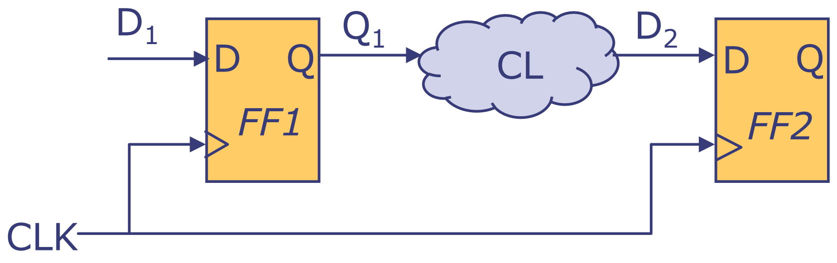


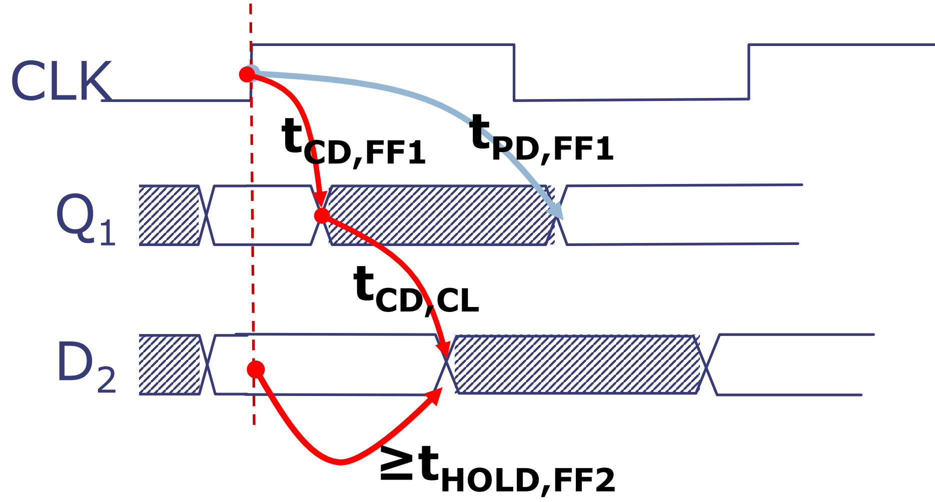
- To meet FF2's setup time,

$$\boxed{t_{CLK} \geq t_{PD,FF1} + t_{PD,CL} + t_{SETUP,FF2}}$$

- The slowest register-to-register path in the system determines the clock;
- Equivalently, a given register technology and clock limit the amount of combinational logic between registers

Meeting the Hold-Time Constraint





- Hold time (t_{HOLD}) constraint of FF2 may be violated if D_2 changes too quickly
- Propagation delay (t_{PD}), the upper bound on time from valid inputs to valid outputs, does not help us analyze hold time!
- *Contamination delay* (t_{CD}) is the lower bound on time from input-to-output transition (invalid input to invalid output)
- To meet FF2's hold-time constraint

$$\boxed{t_{CD,FF1} + t_{CD,CL} \geq t_{HOLD,FF2}}$$

Tools may need to add logic to fast paths to meet t_{HOLD}

Timing Summary

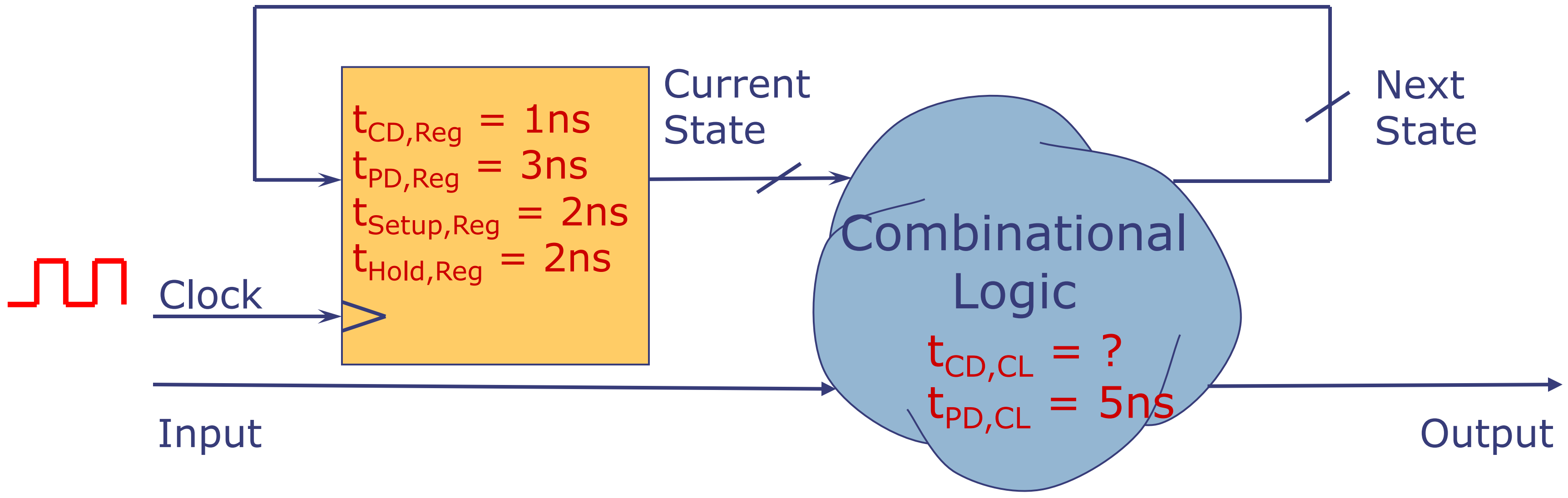
- For a sequential circuit to work properly, we must guarantee that the setup time and hold time constraints of every register will always be satisfied.

- The setup time constraint is affected by both the logic in the circuit *and* the clock period. To fix violations, either:
  - Change the logic to be faster (lower t_{PD})
  - Change the clock to be slower (higher t_{CLK})

- The hold time constraint is affected *only* by the logic in the circuit.
  - Changing the clock period will not fix violations.
  - Sum of contamination delays must be greater than the register hold time, otherwise the circuit won't work.

- If hold time is satisfied, then the fastest clock period can be set as the maximum sum of the propagation delays plus setup time across all register-to-register paths.

Sequential Circuit Timing Example



Questions:

- Constraints on t_{CD} for the logic?

  $t_{CD,Reg}$ (1 ns) + $t_{CD,CL}$(?) $\geq$ $t_{Hold,Reg}$(2 ns)

  $t_{CD,CL} \geq 1$ ns

- Minimum clock period?

  $t_{CLK} \geq t_{PD,Reg} + t_{PD,CL} + t_{Setup,Reg} = 10$ns

Thank you!

Next lecture:

Sequential logic in Minispec