# Clocked Circuits

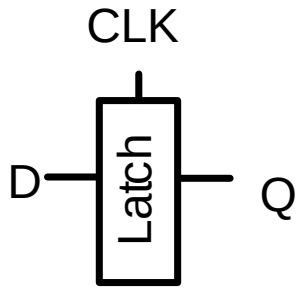
## Additional Slides

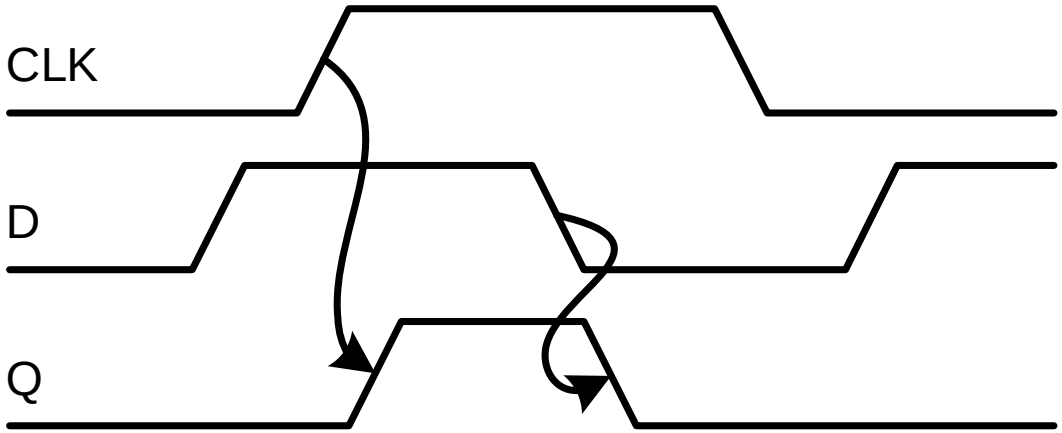
Vishal Saxena
ECE, Boise State University

Nov 30, 2010

# D Latch

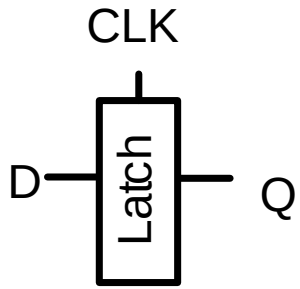
- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D

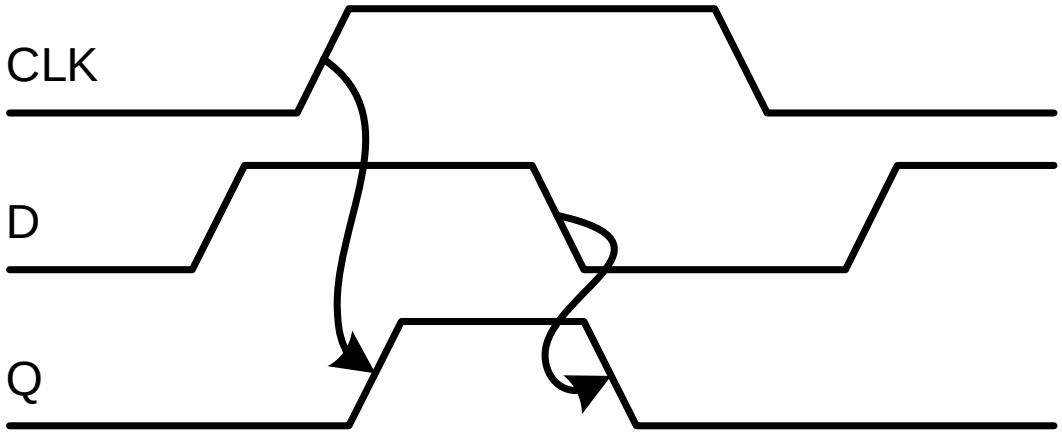


# D Latch

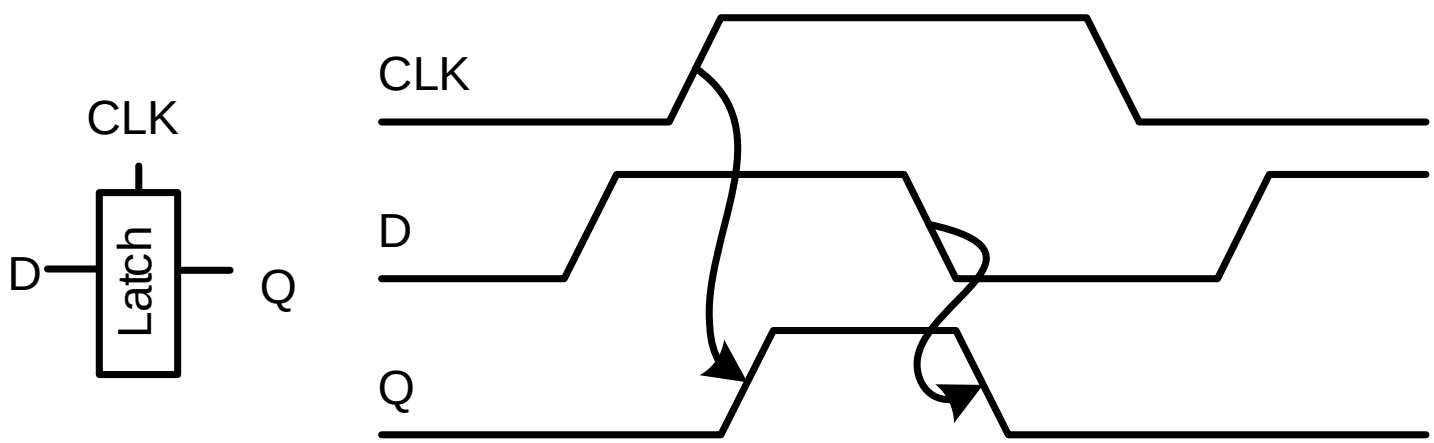
- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D



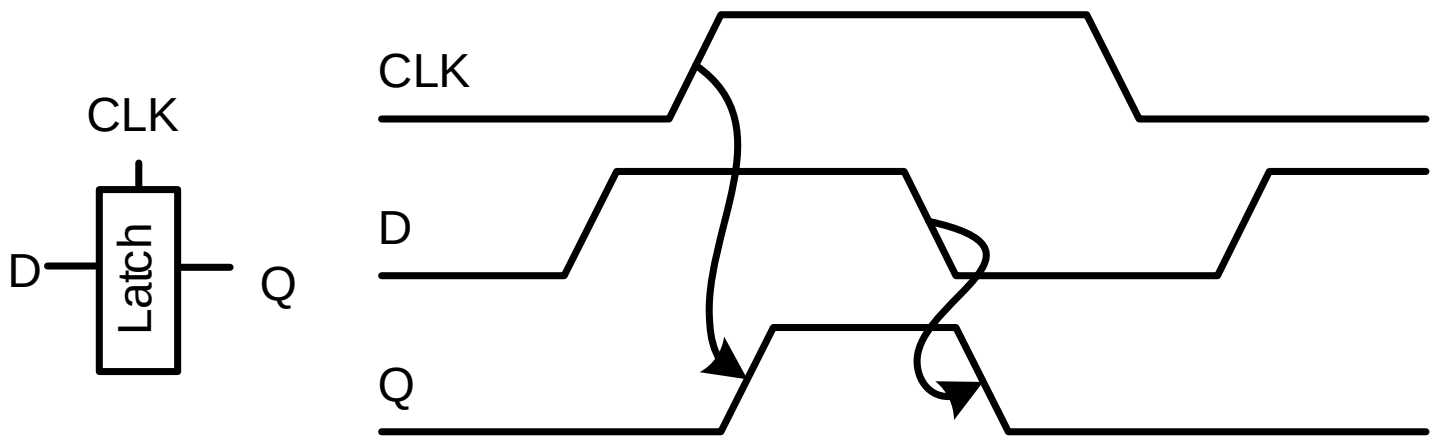
# D Latch

- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D



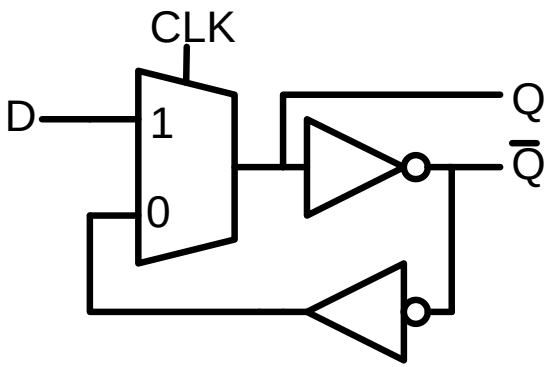
# D Latch

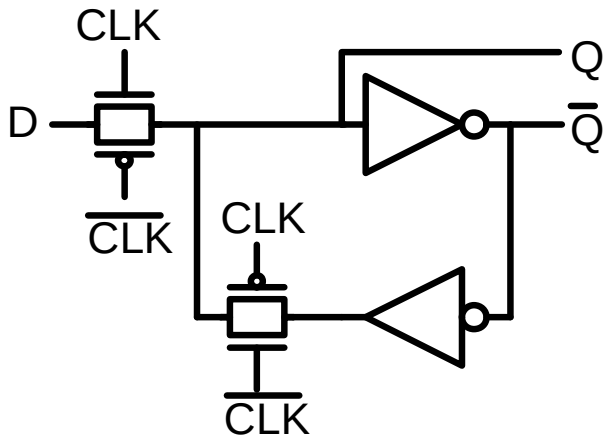
- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D



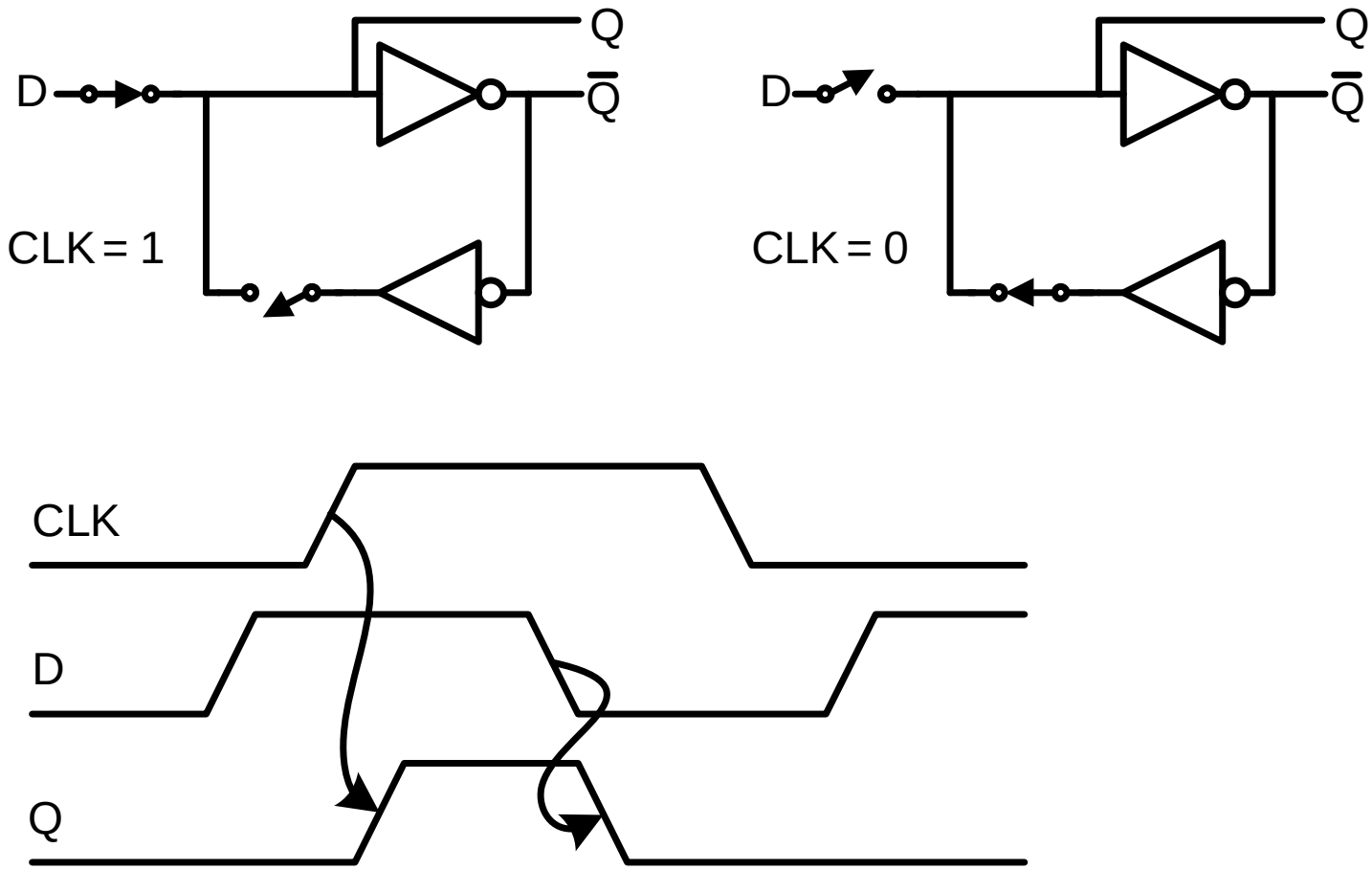
# D Latch Design

- Multiplexer chooses D or old Q



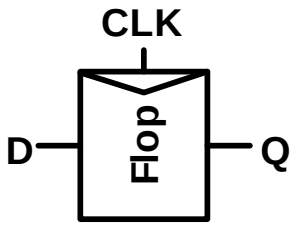



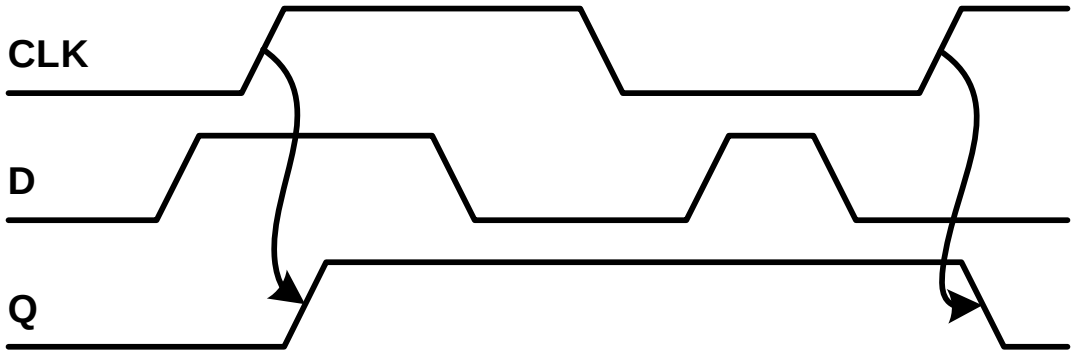
# D Latch Operation



# D Flip-flop

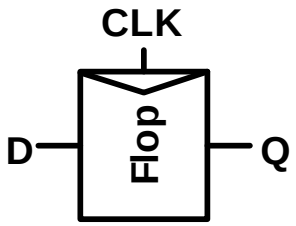
- When CLK rises, D is copied to Q
  - At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop, master-slave flip-flop*

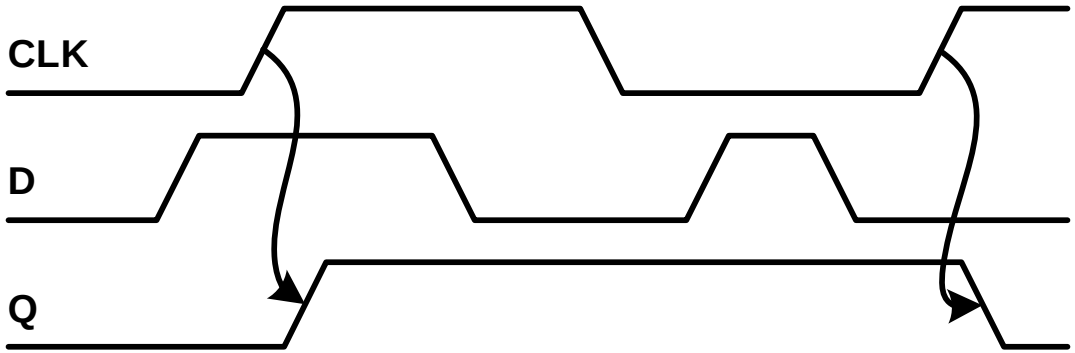


# D Flip-flop

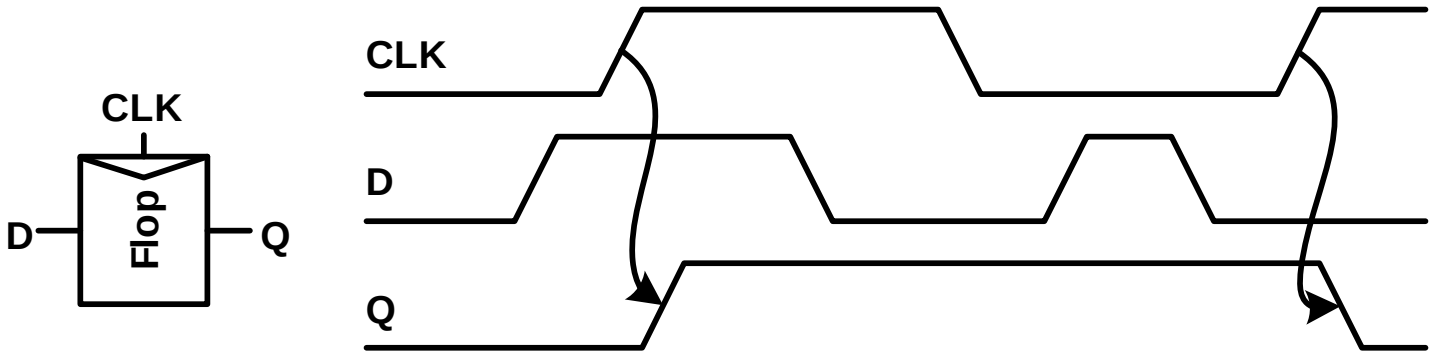
- When CLK rises, D is copied to Q
  - At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop, master-slave flip-flop*



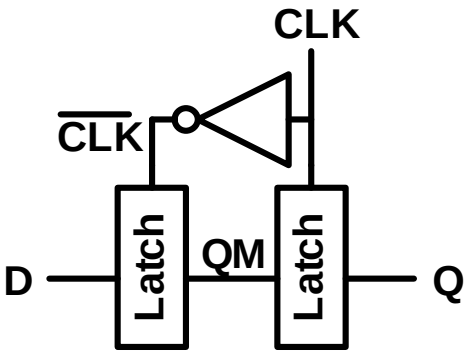
# D Flip-flop

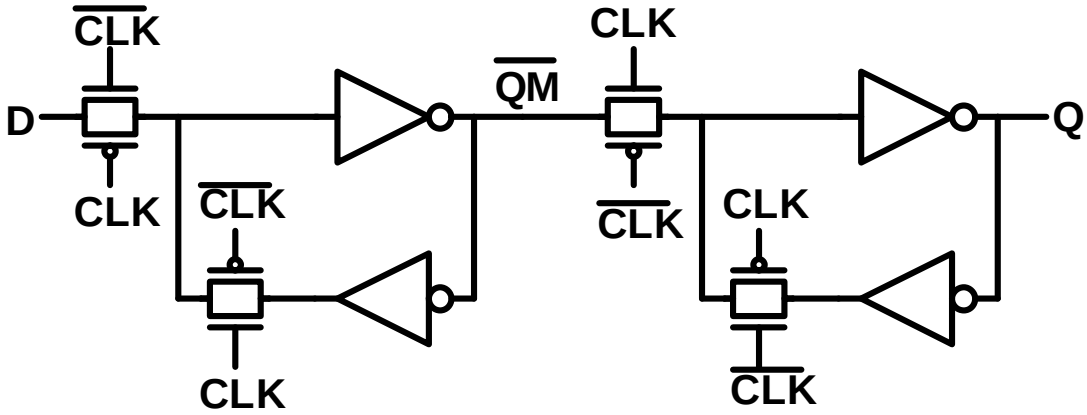
- When CLK rises, D is copied to Q
  - At all other times, Q holds its value
- a.k.a. *positive edge-triggered flip-flop, master-slave flip-flop*



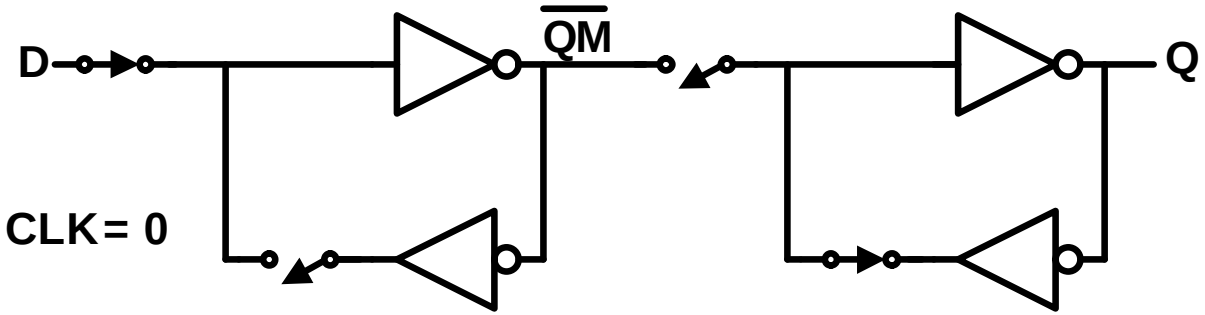
# D Flip-flop Design

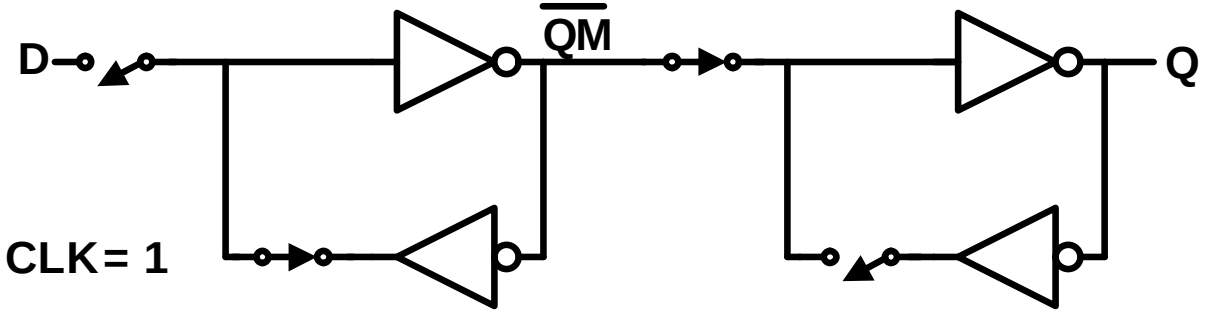
- Built from master and slave D latches

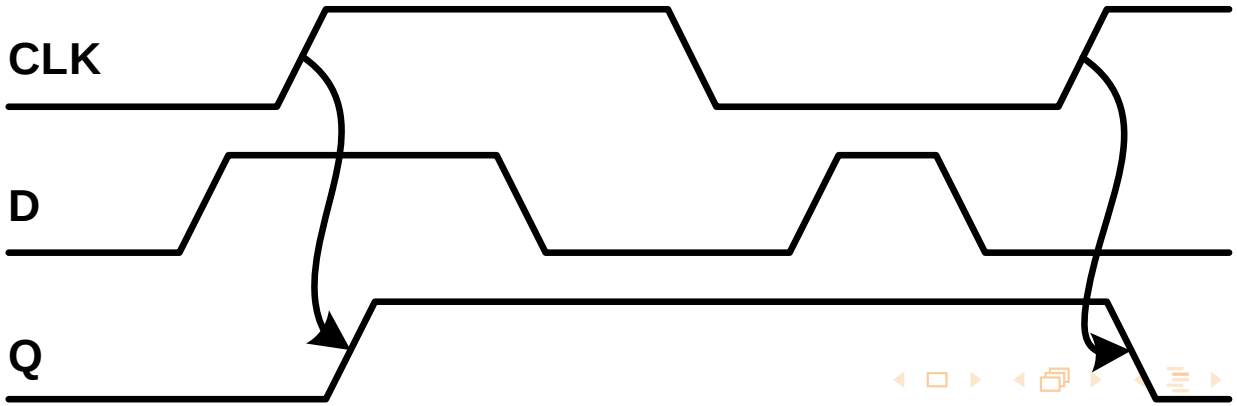


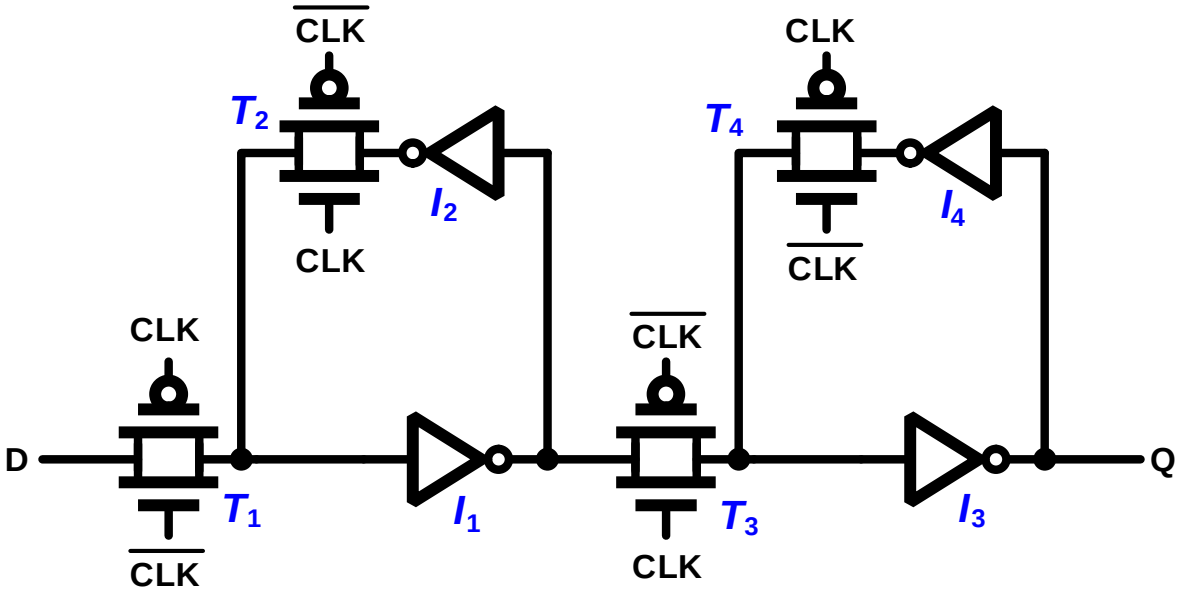
# D Flip-flop Operation



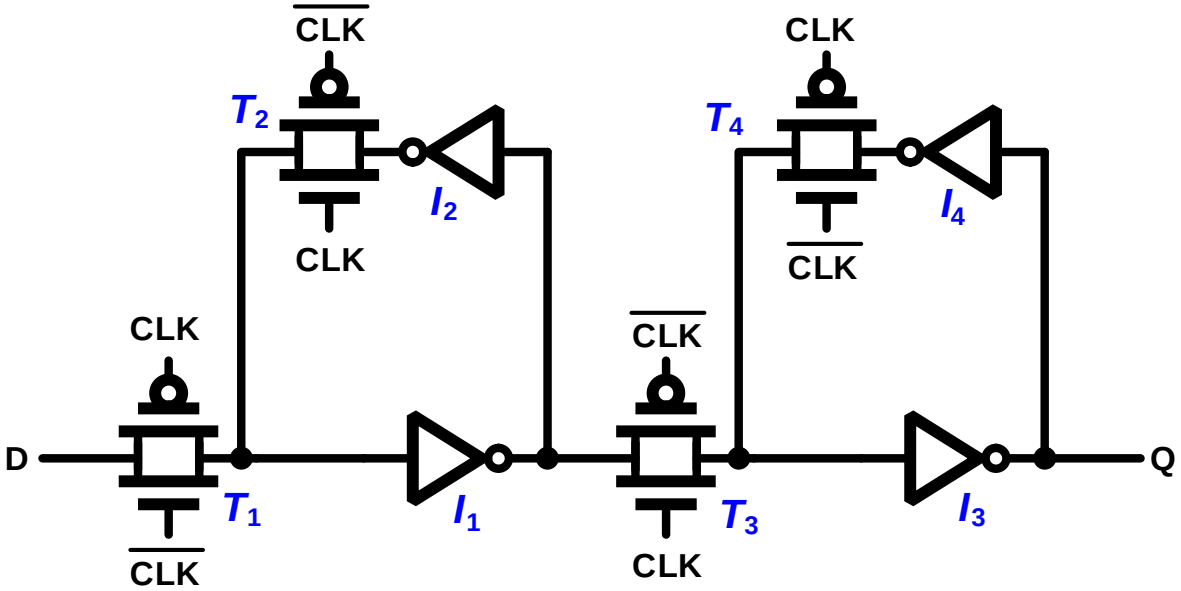



# DFF Example 1



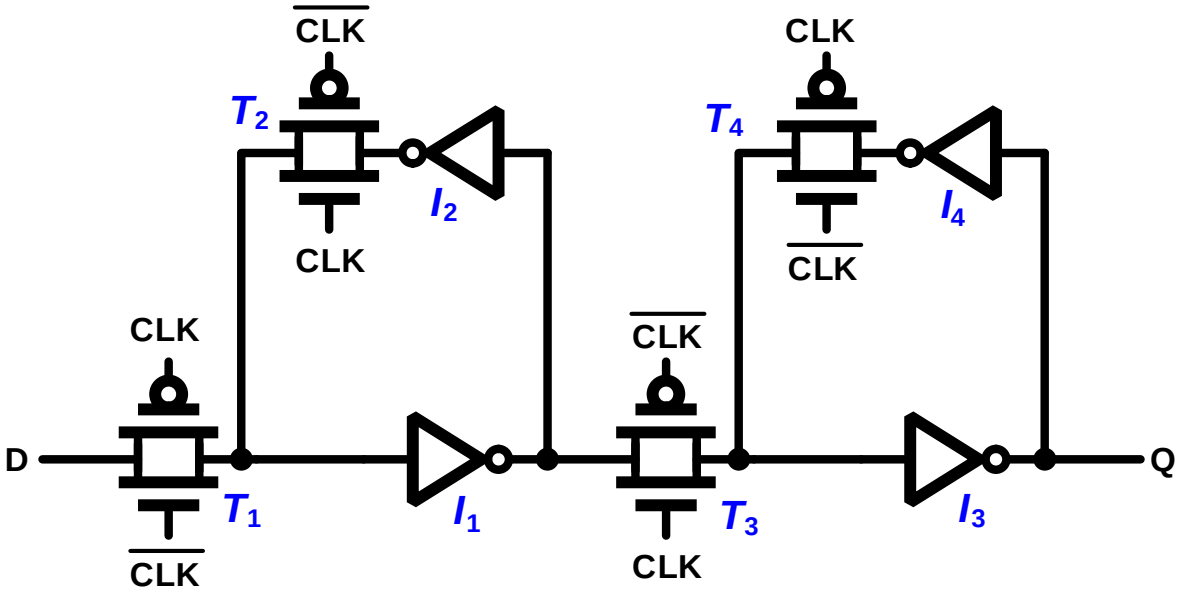

- $t_{setup} = t_{T_1} + t_{I_1}$
- $t_{hold} = -(t_{T_1} + t_{I_1})$
- $t_{pcq} = t_{T_3} + t_{I_3}$

# DFF Example 1



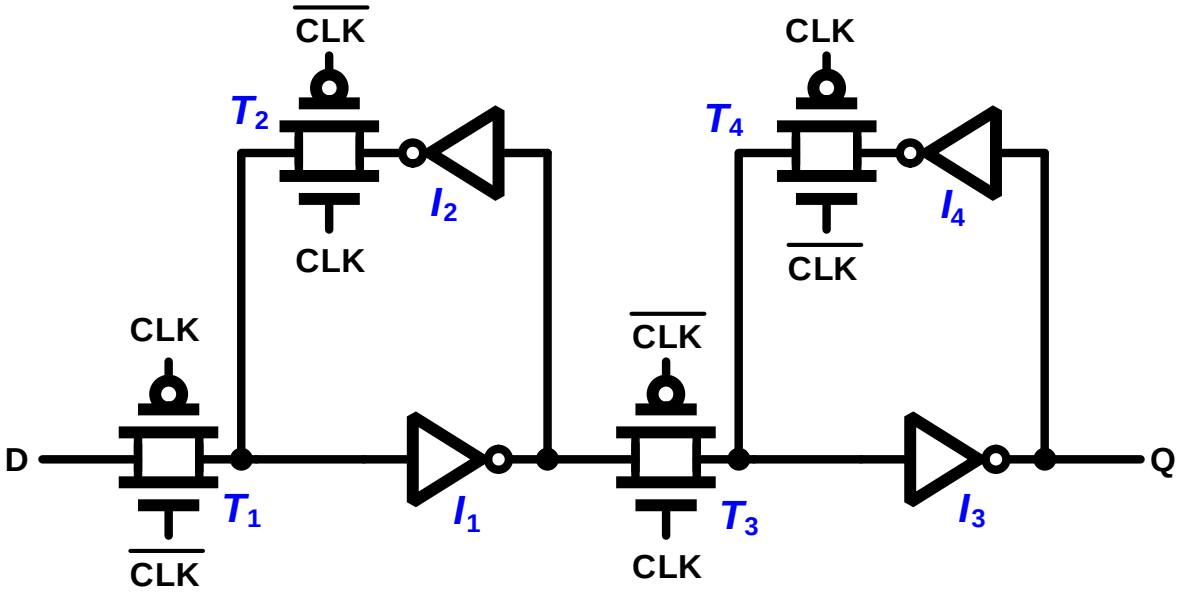

- $t_{setup} = t_{T_1} + t_{I_1}$
- $t_{hold} = -(t_{T_1} + t_{I_1})$
- $t_{pcq} = t_{T_3} + t_{I_3}$

# DFF Example 1



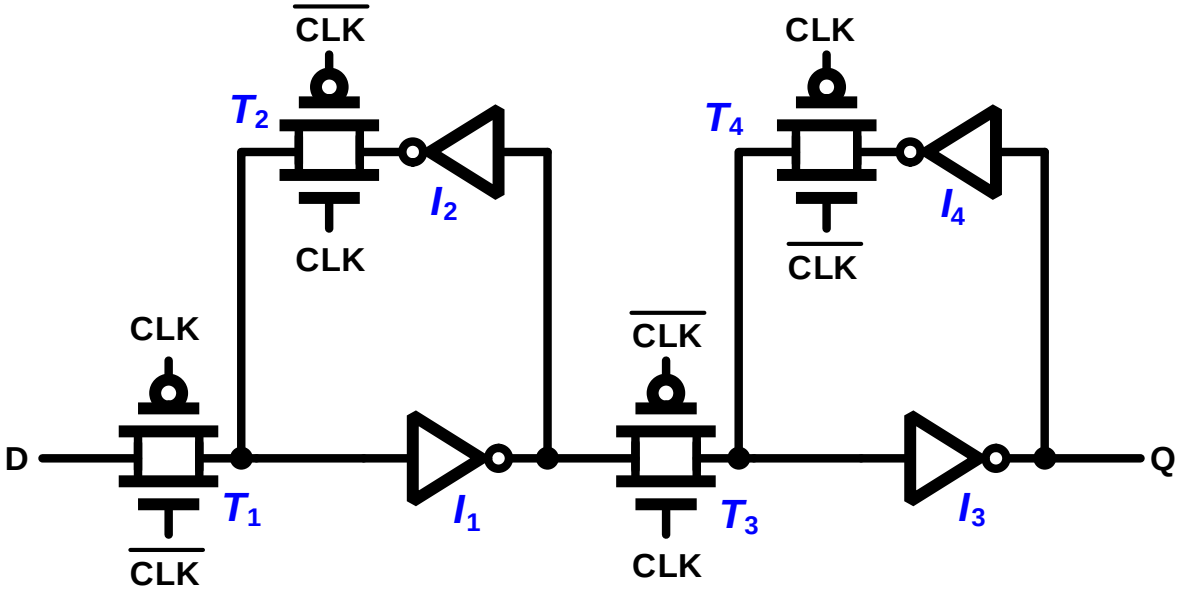

- $t_{setup} = t_{T_1} + t_{I_1}$
- $t_{hold} = -(t_{T_1} + t_{I_1})$
- $t_{pcq} = t_{T_3} + t_{I_3}$

# DFF Example 1



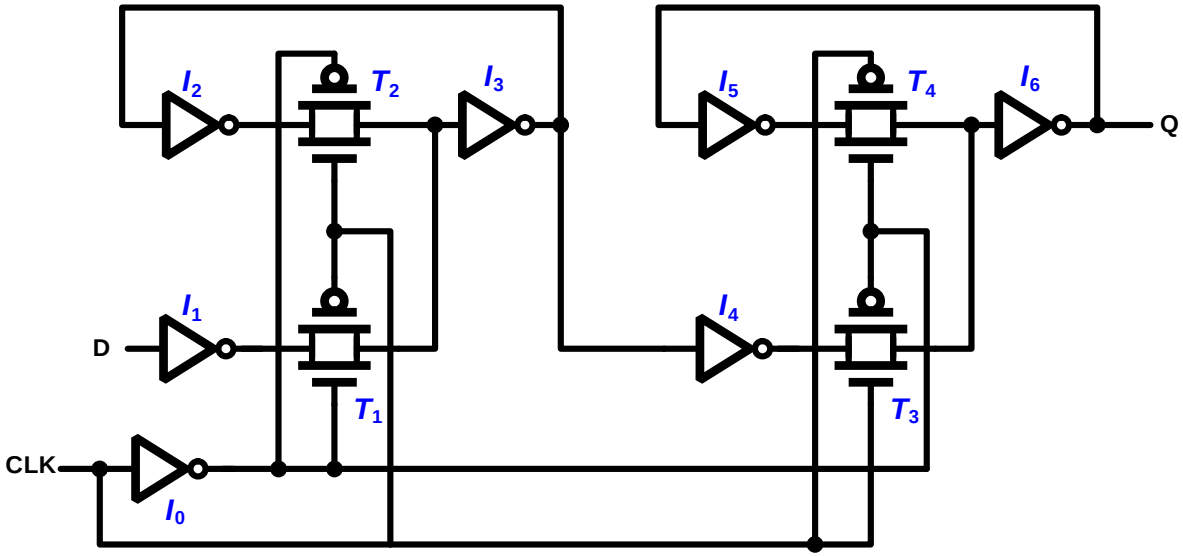

- $t_{setup} = t_{T_1} + t_{I_1}$
- $t_{hold} = -(t_{T_1} + t_{I_1})$
- $t_{pcq} = t_{T_3} + t_{I_3}$

# DFF Example 1



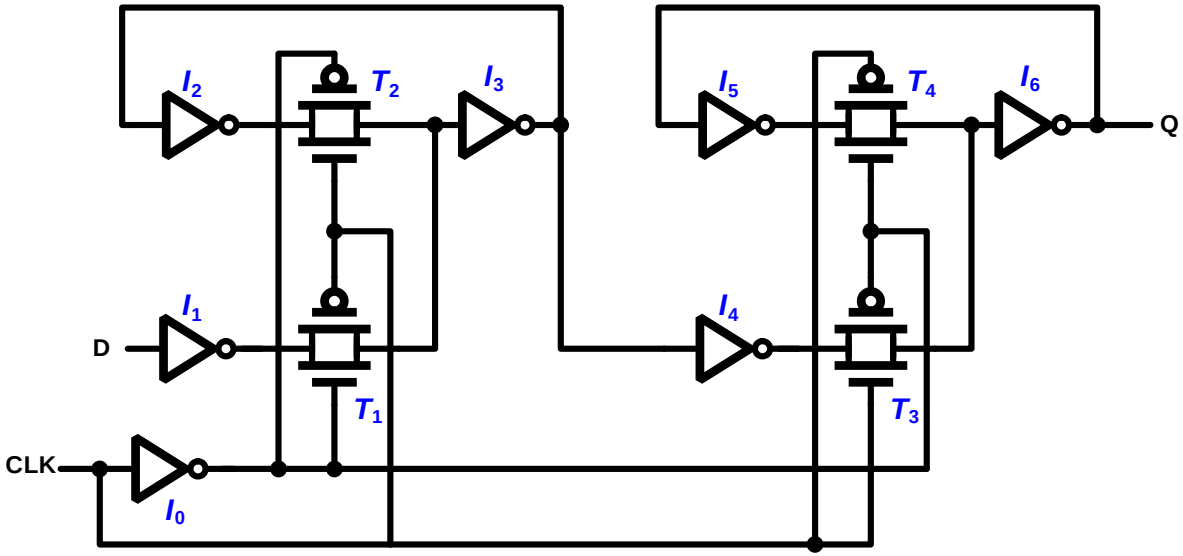

- $t_{setup} = t_{T_1} + t_{I_1}$
- $t_{hold} = -(t_{T_1} + t_{I_1})$
- $t_{pcq} = t_{T_3} + t_{I_3}$

# DFF Example 2



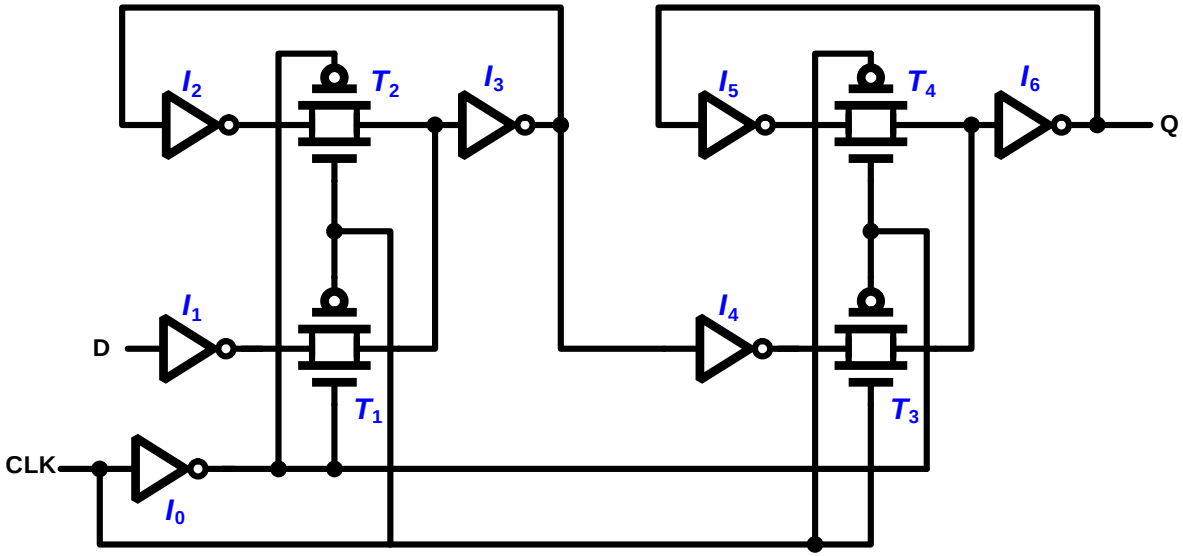

- $t_{setup} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4}$
- $t_{hold} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3})$(Note that the $\overline{CLK}$ and $CLK$ have an overlap equal to an inverter delay ($t_{I_0}$))
  - Need to hold input (D), otherwise the master will latch new D value.
- $t_{pcq} = t_{T_3} + t_{I_6}$

# DFF Example 2



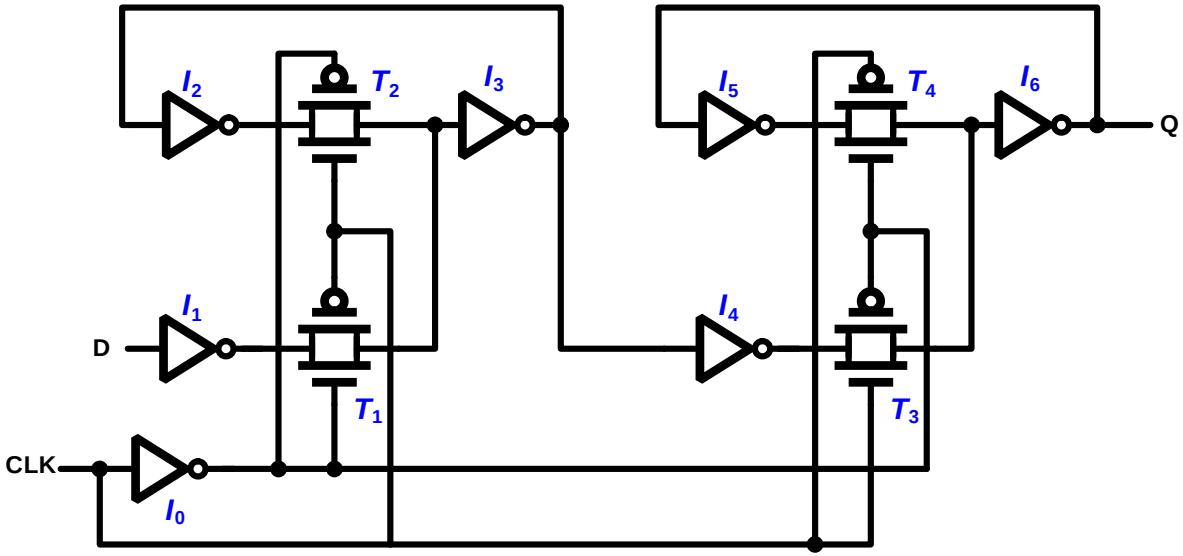

- $t_{setup} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4}$
- $t_{hold} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3})$(Note that the $\overline{CLK}$ and $CLK$ have an overlap equal to an inverter delay ($t_{I_0}$))
  - Need to hold input (D), otherwise the master will latch new D value.
- $t_{pcq} = t_{T_3} + t_{I_6}$

# DFF Example 2



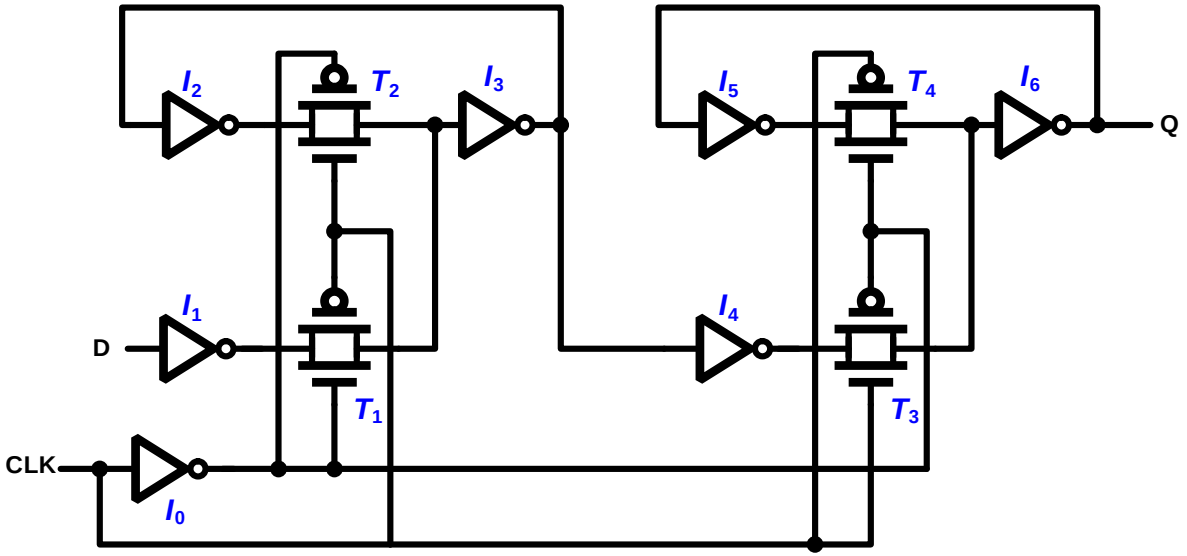

- $t_{setup} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4}$
- $t_{hold} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3})$(Note that the $\overline{CLK}$ and $CLK$ have an overlap equal to an inverter delay $(t_{I_0})$)
  - Need to hold input (D), otherwise the master will latch new D value.
- $t_{pcq} = t_{T_3} + t_{I_6}$

# DFF Example 2




- $t_{setup} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4}$
- $t_{hold} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3})$(Note that the $\overline{CLK}$ and $CLK$ have an overlap equal to an inverter delay $(t_{I_0})$)
  - Need to hold input (D), otherwise the master will latch new D value.
- $t_{pcq} = t_{T_3} + t_{I_6}$

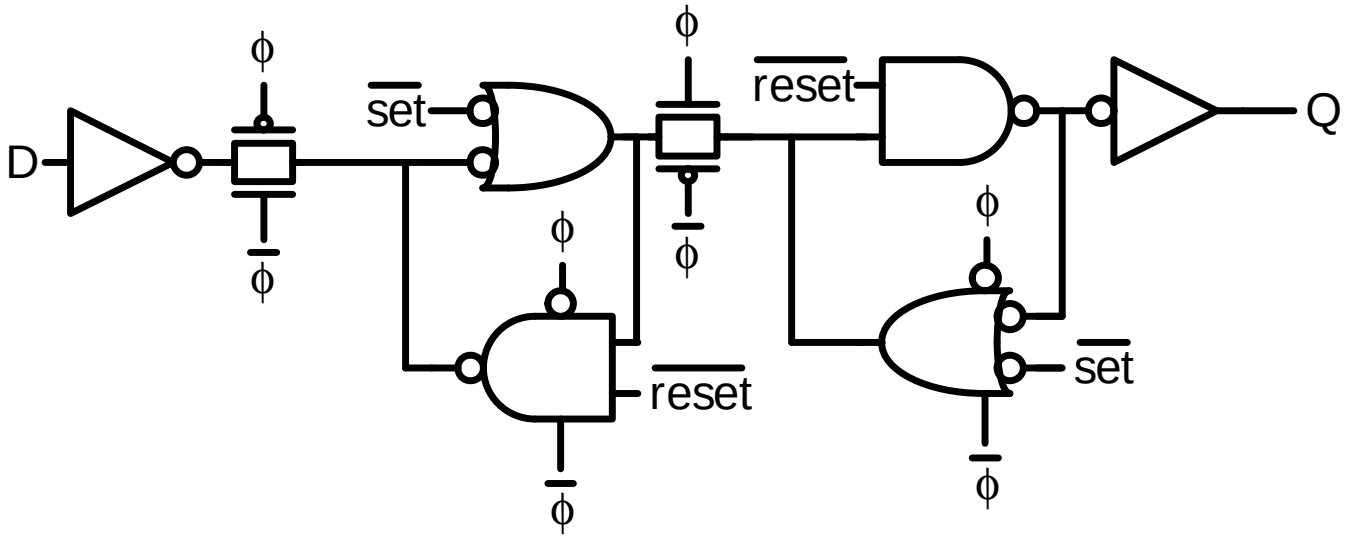
# DFF Example 2




- $t_{setup} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4}$
- $t_{hold} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3})$(Note that the $\overline{CLK}$ and $CLK$ have an overlap equal to an inverter delay ($t_{I_0}$))
  - Need to hold input (D), otherwise the master will latch new D value.
- $t_{pcq} = t_{T_3} + t_{I_6}$

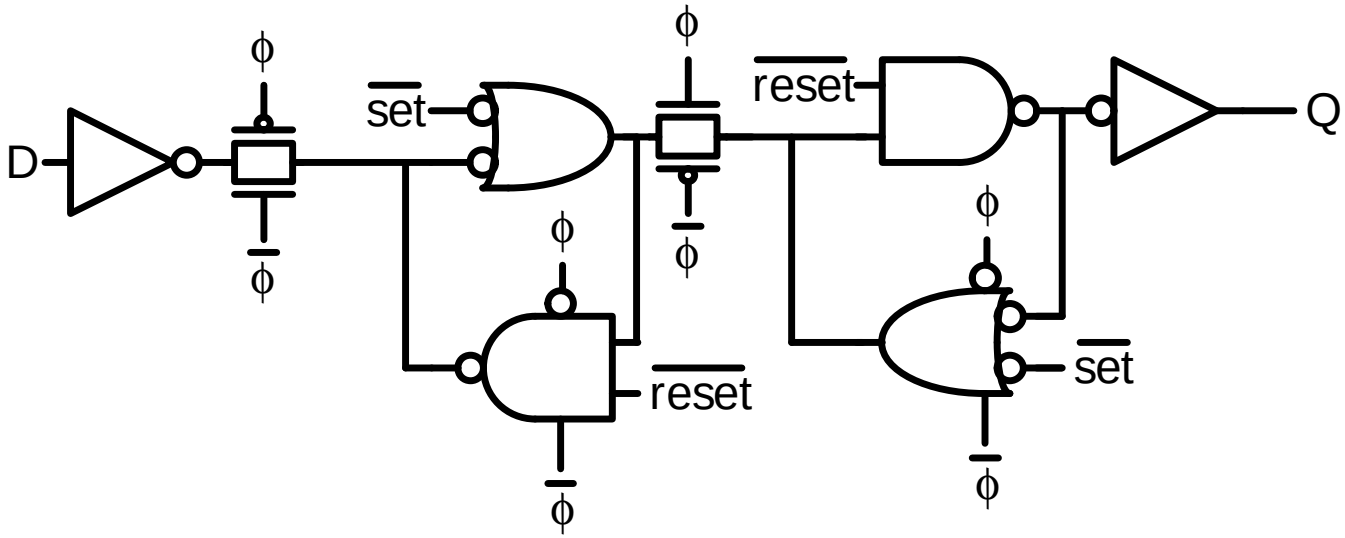
# Set / Reset

- Set forces output high when enabled
- Flip-flop with asynchronous set and reset



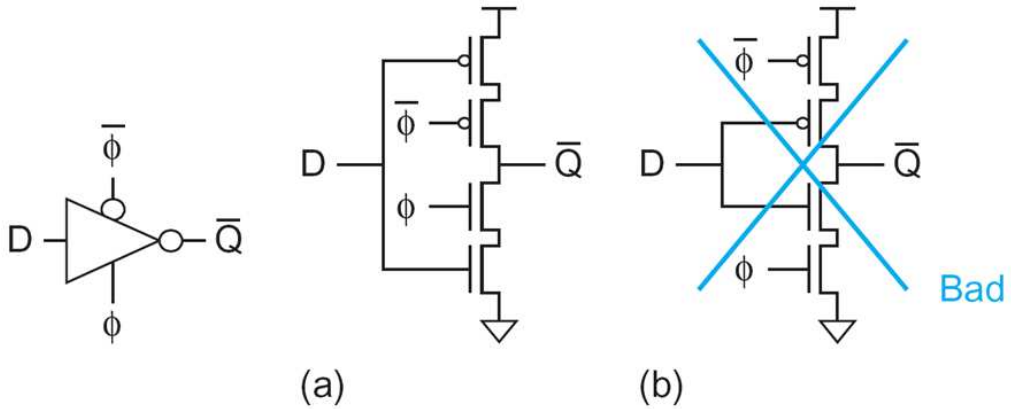
# Set / Reset

- Set forces output high when enabled
- Flip-flop with asynchronous set and reset



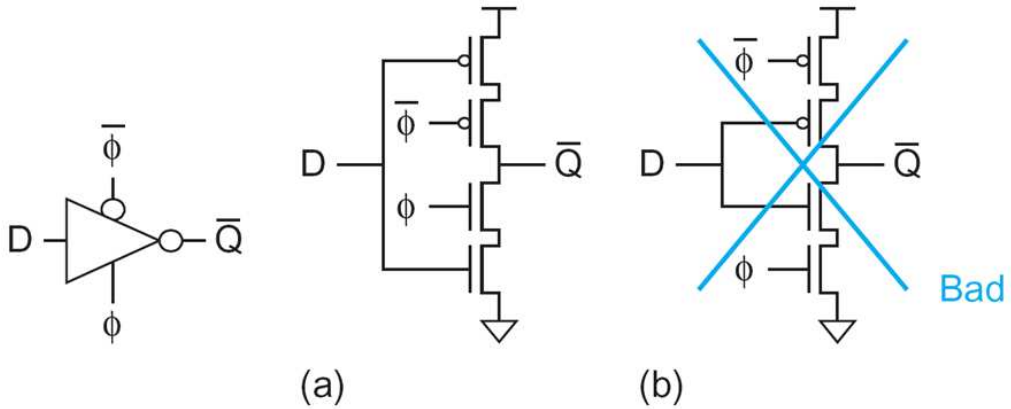
# Dynamic Latch

- A dynamic latch
  - sometimes called clocked CMOS ($C^2MOS$) latch
  - second design is inferior as the input noise passed into the output when the latch is opaque



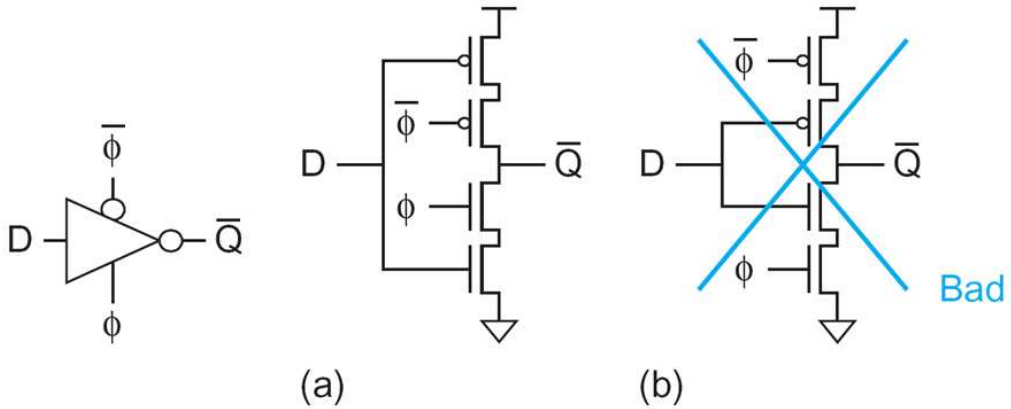
# Dynamic Latch

- A dynamic latch
  - sometimes called clocked CMOS ($C^2MOS$) latch
  - second design is inferior as the input noise passed into the output when the latch is opaque



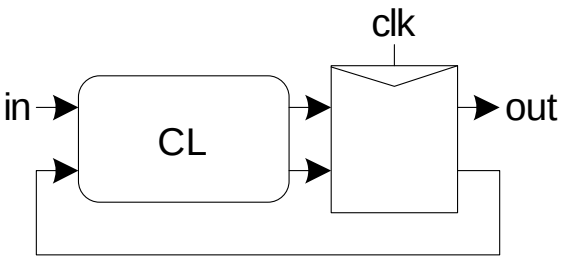
# Dynamic Latch

- A dynamic latch
  - sometimes called clocked CMOS ($C^2MOS$) latch
  - second design is inferior as the input noise passed into the output when the latch is opaque

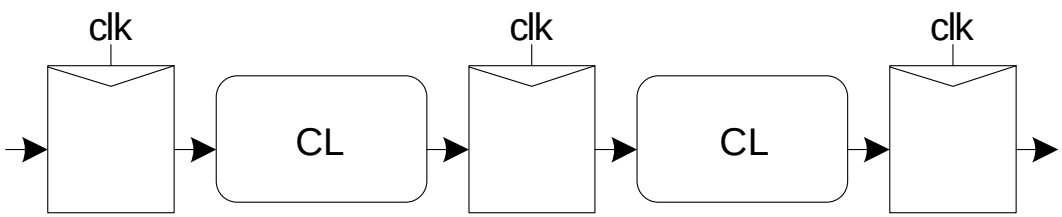


# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline



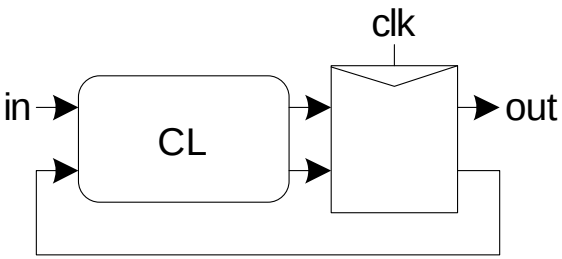

Finite State Machine



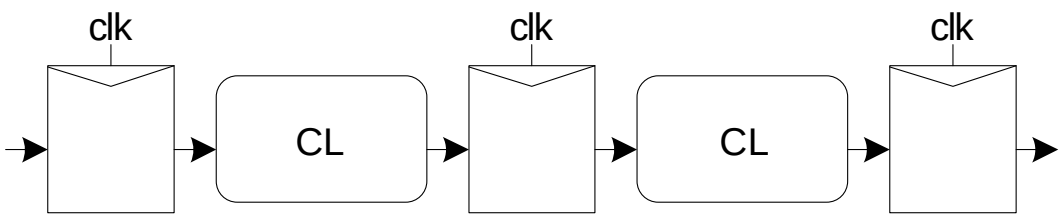

Pipeline

# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline



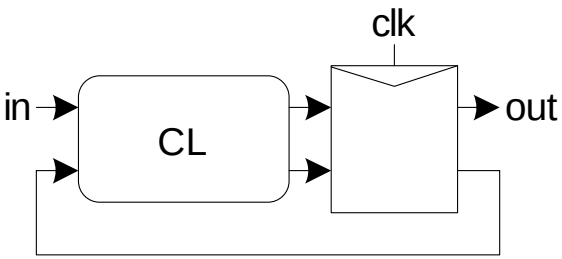

Finite State Machine



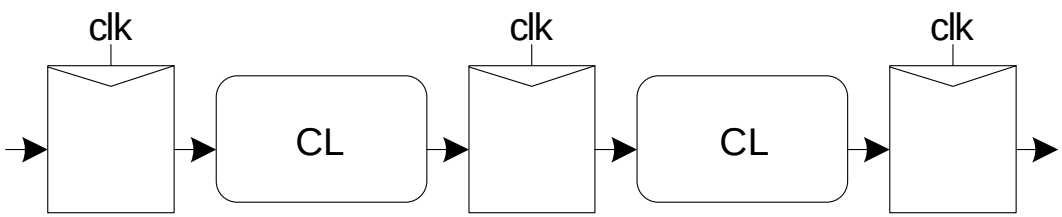

Pipeline

# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline



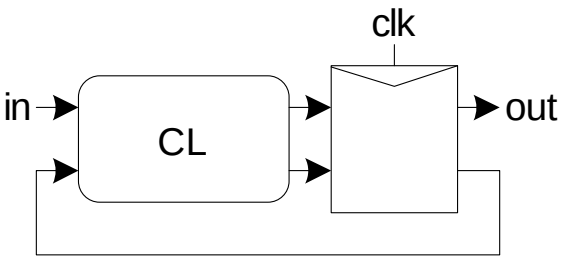

Finite State Machine



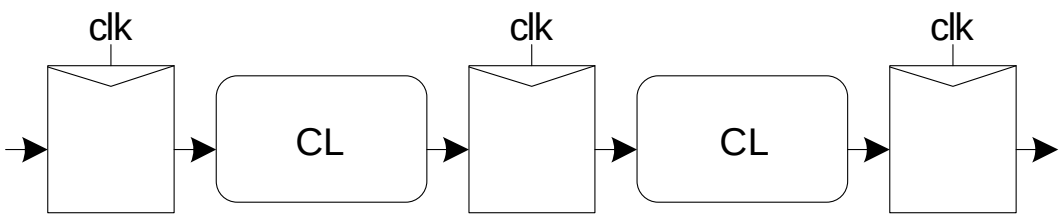

Pipeline

# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline



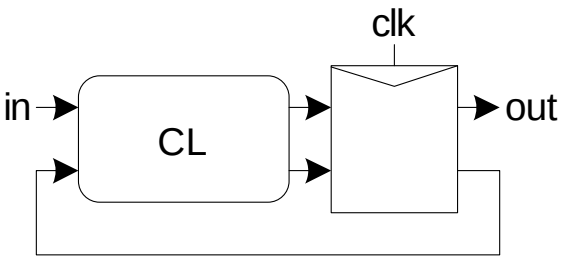

Finite State Machine



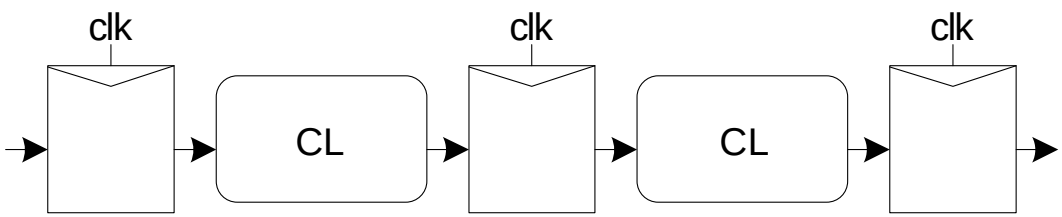

Pipeline

# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline



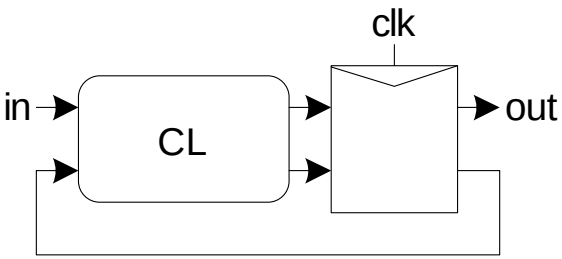

Finite State Machine



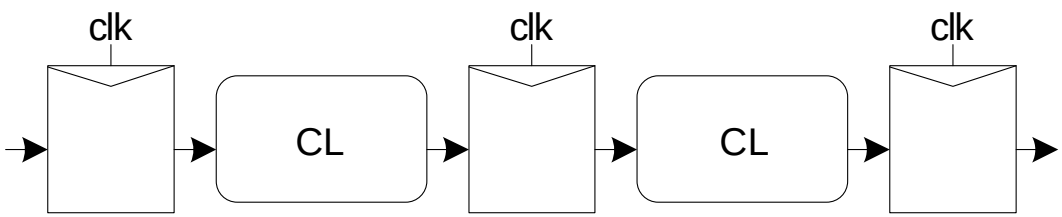

Pipeline

# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline



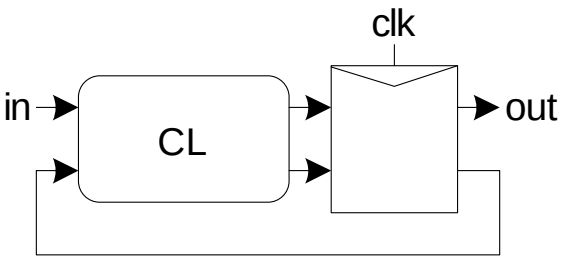

Finite State Machine



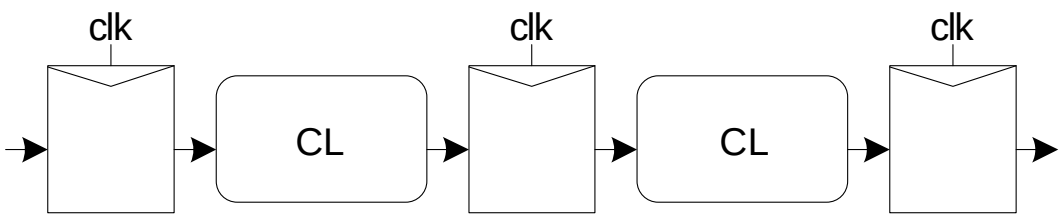

Pipeline

# Sequencing

- Combinational logic
  - output depends on current inputs
- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline




Finite State Machine




Pipeline

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing contd.

- If tokens moved through pipeline at constant speed, no sequencing elements would be necessary
- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses
- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don't catch slow ones.

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Overhead

- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence

# Sequencing Elements

- Latch: Level sensitive
  - a.k.a. transparent latch, D latch
- Flip-flop: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register

# Sequencing Elements

- Latch: Level sensitive
  - a.k.a. transparent latch, D latch
- Flip-flop: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register

# Sequencing Elements

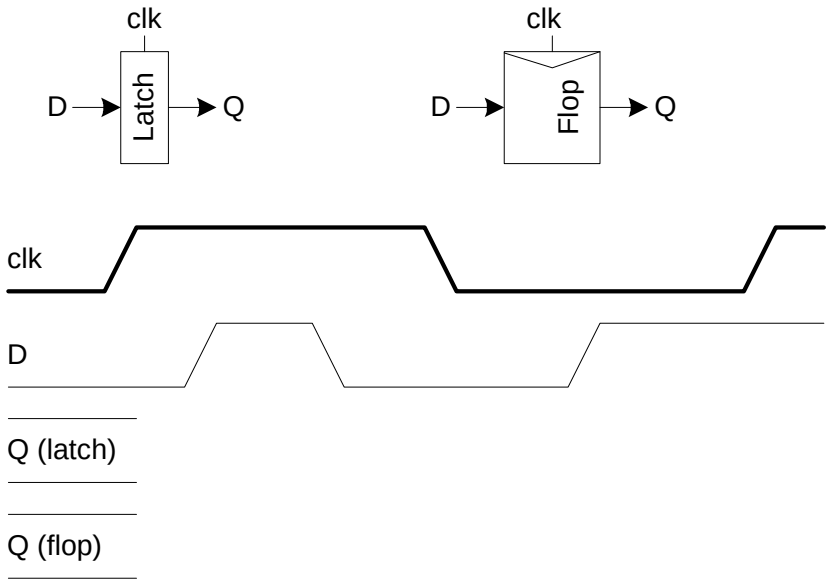
- Latch: Level sensitive
  - a.k.a. transparent latch, D latch
- Flip-flop: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register

# Sequencing Elements

- Latch: Level sensitive
  - a.k.a. transparent latch, D latch
- Flip-flop: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register

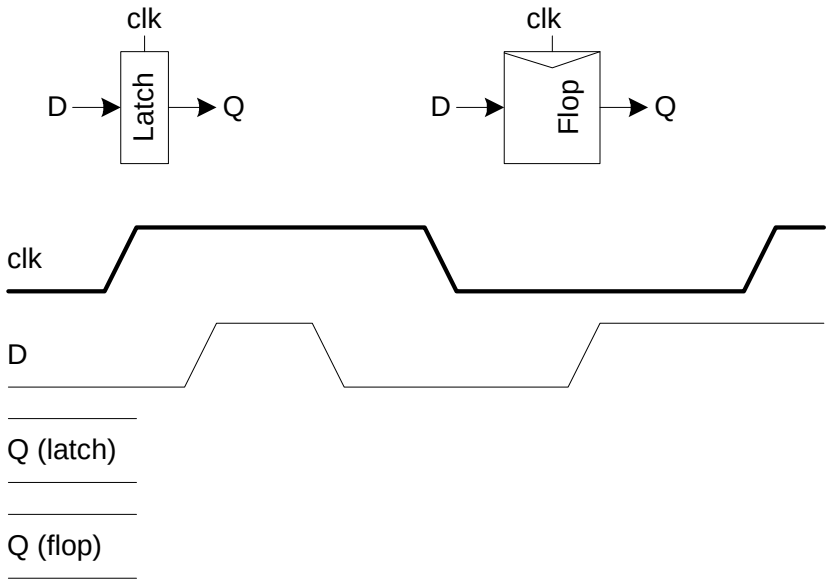
# Timing Diagrams

- Transparent
- Opaque
- Edge-trigger



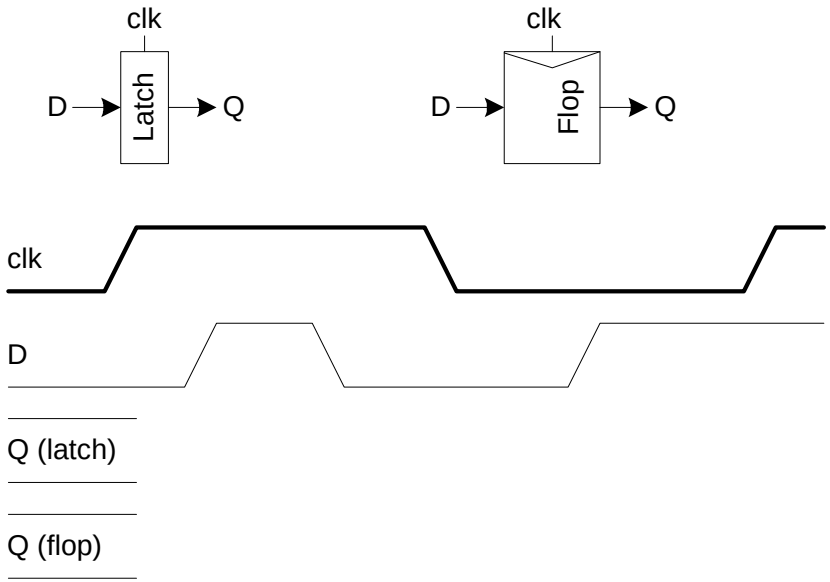
# Timing Diagrams

- Transparent
- Opaque
- Edge-trigger



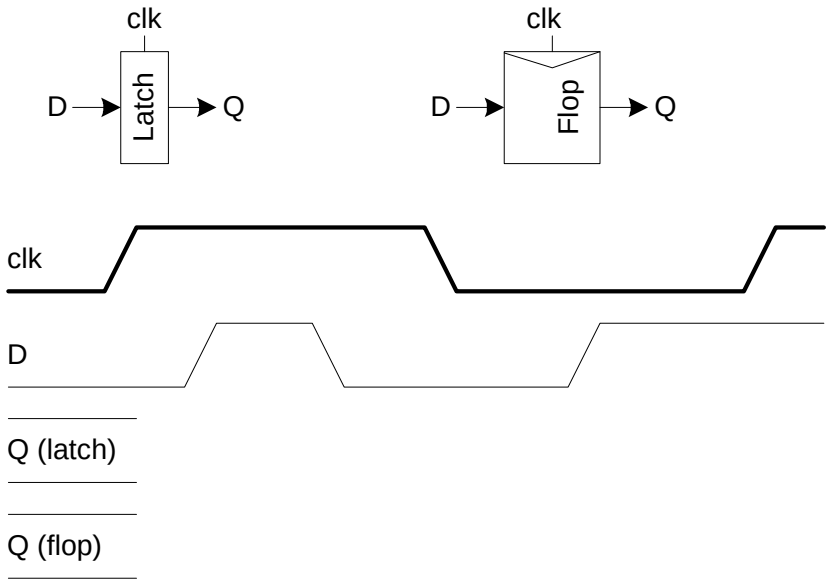
# Timing Diagrams

- Transparent
- Opaque
- Edge-trigger



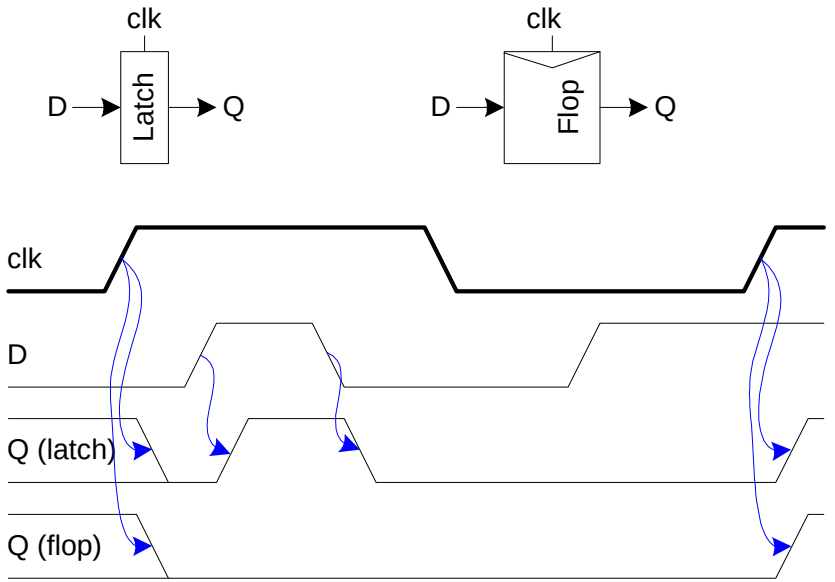
# Timing Diagrams

- Transparent
- Opaque
- Edge-trigger



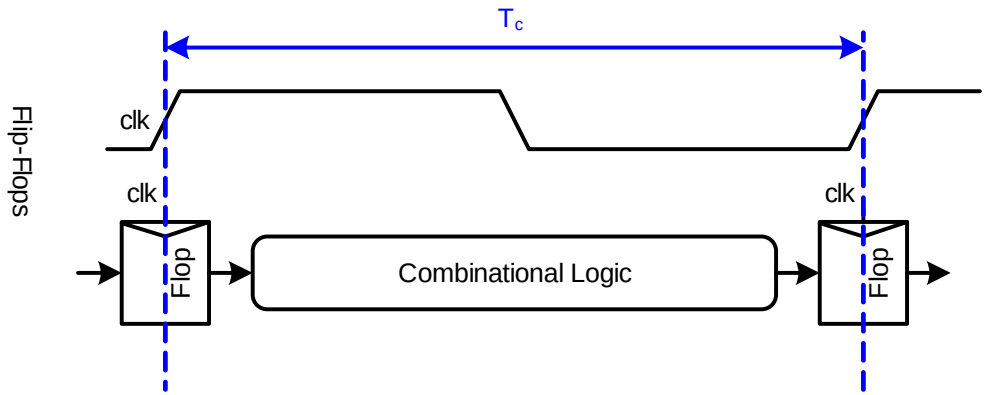
# Timing Diagrams

- Transparent
- Opaque
- Edge-trigger



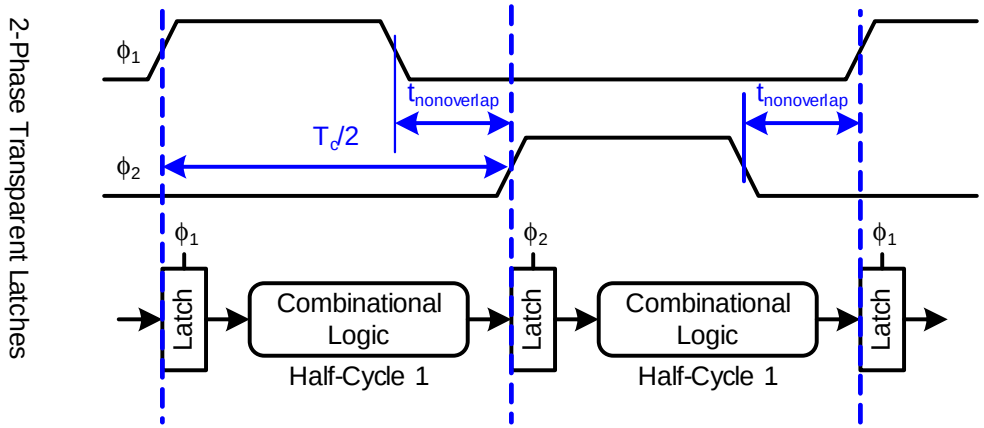
# Sequencing Methods

- Flip-flops
- 2-phase latches
- Pulsed Latches



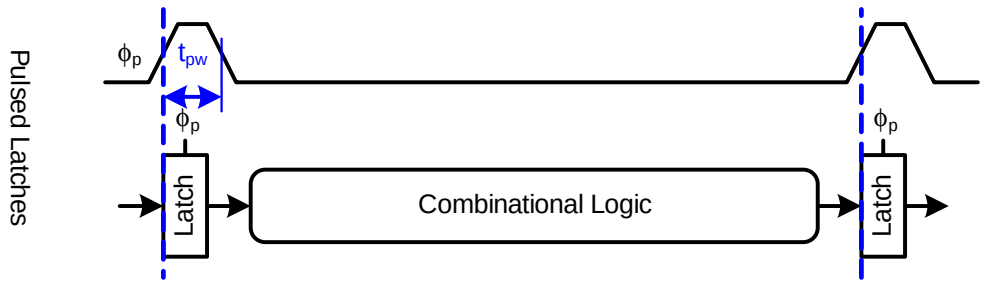
# Sequencing Methods

- Flip-flops
- 2-phase latches
- Pulsed Latches



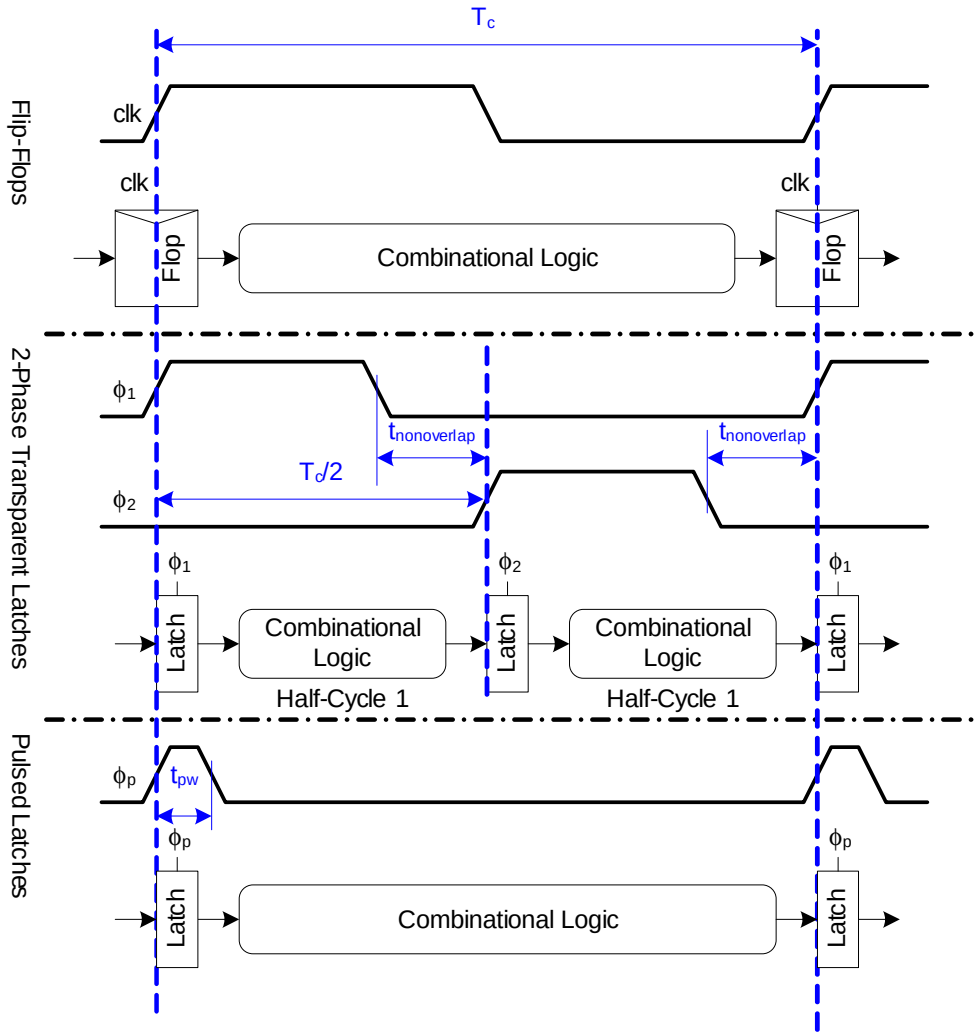
# Sequencing Methods

- Flip-flops
- 2-phase latches
- Pulsed Latches



# Sequencing Methods

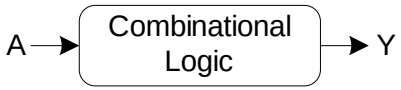
- Flip-flops
- 2-phase latches
- Pulsed Latches

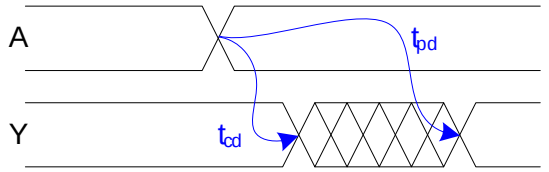


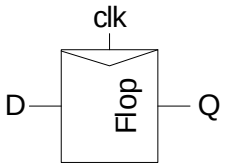
# Timing Diagrams

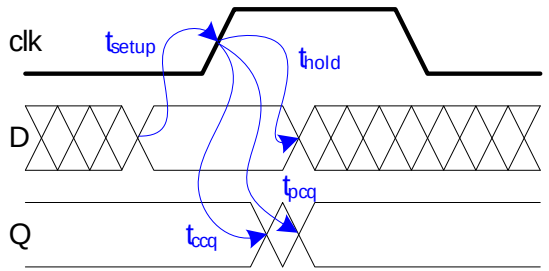
| | |
|---|---|
| $t_{pd}$ | Logic Prop. Delay |
| $t_{cd}$ | Logic Cont. Delay |
| $t_{pcq}$ | Latch/Flop Clk->Q Prop. Delay |
| $t_{ccq}$ | Latch/Flop Clk->Q Cont. Delay |
| $t_{pdq}$ | Latch D->Q Prop. Delay |
| $t_{cdq}$ | Latch D->Q Cont. Delay |
| $t_{setup}$ | Latch/Flop Setup Time |
| $t_{hold}$ | Latch/Flop Hold Time |

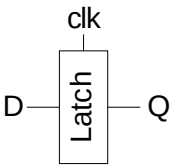
Table: Contamination and Propagation Delays

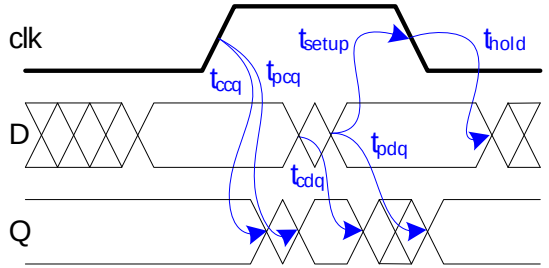


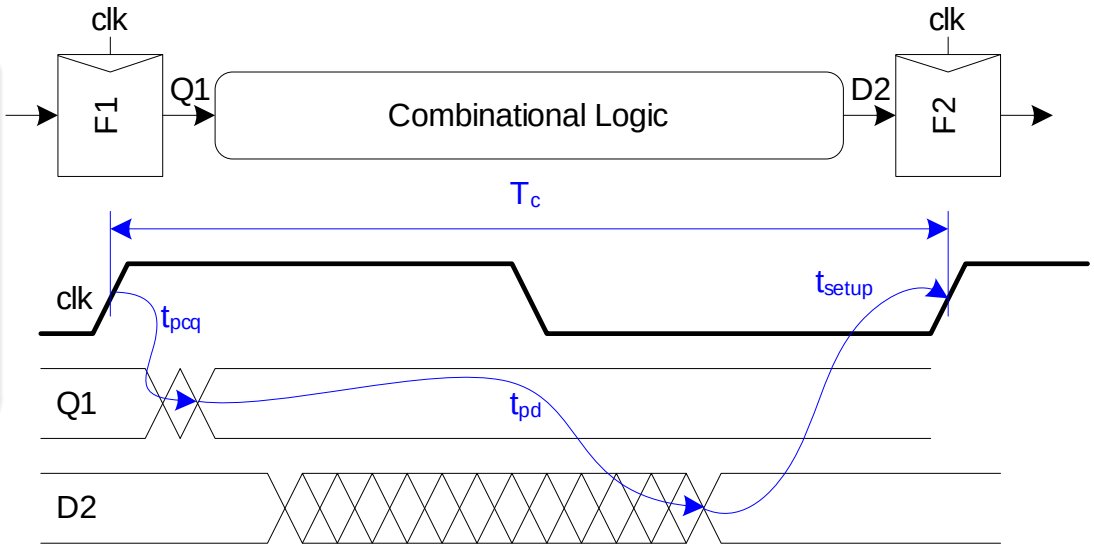



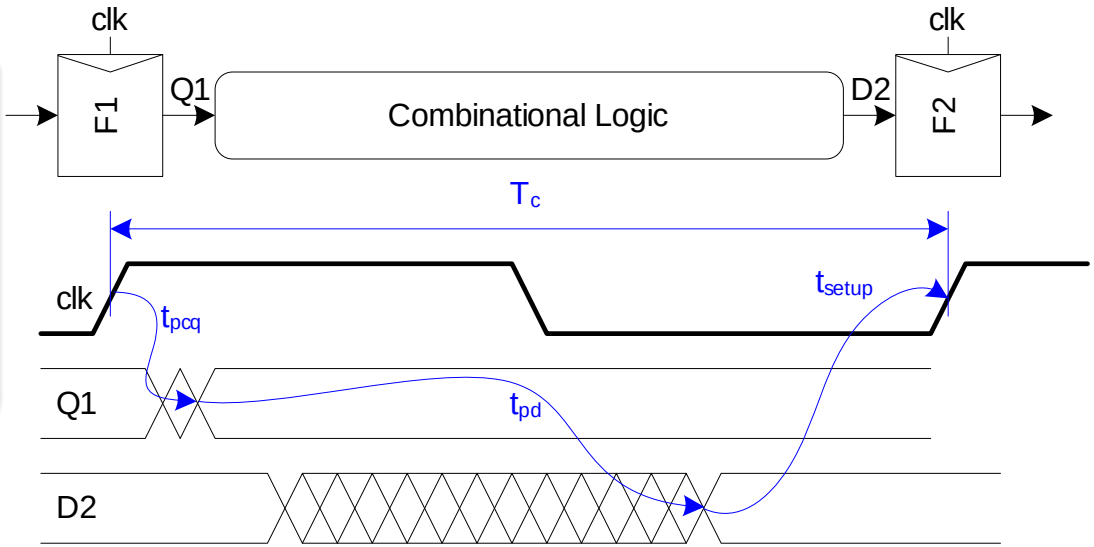
# Max-Delay:Flip-Flops

- $t_{pd} \leq T_c - (?)$
- $t_{pd} \leq T_c - (t_{setup} + t_{pcq})$
- $(t_{setup} + t_{pcq})$ is the sequencing overhead
- setup check



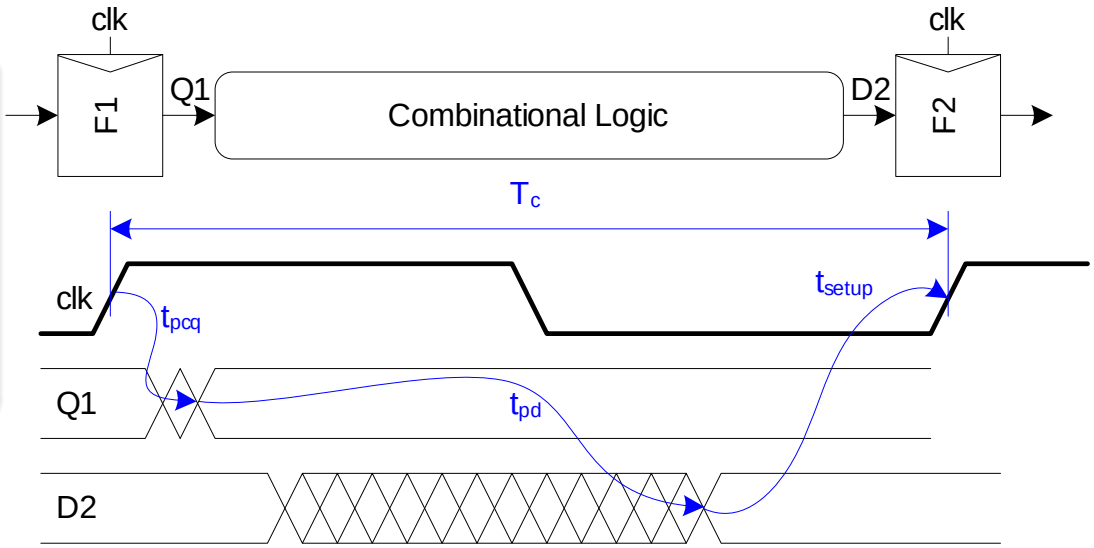
# Max-Delay:Flip-Flops

- $t_{pd} \leq T_c - (?)$
- $t_{pd} \leq T_c - (t_{setup} + t_{pcq})$
- $(t_{setup} + t_{pcq})$ is the sequencing overhead
- setup check



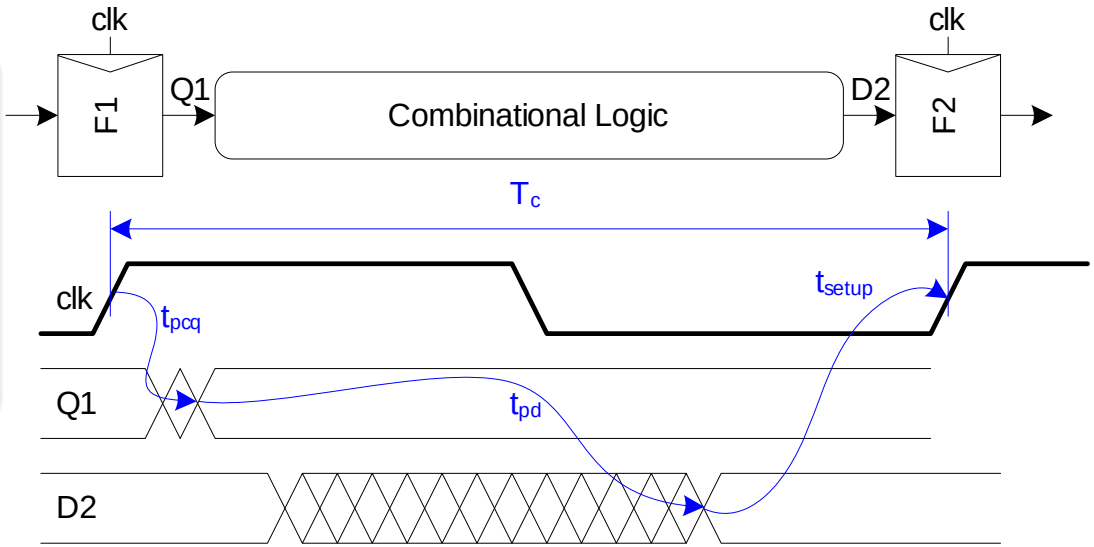
# Max-Delay:Flip-Flops

- $t_{pd} \leq T_c - (?)$
- $t_{pd} \leq T_c - (t_{setup} + t_{pcq})$
- $(t_{setup} + t_{pcq})$ is the sequencing overhead
- setup check



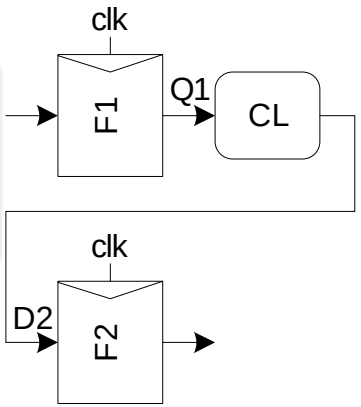
# Max-Delay:Flip-Flops

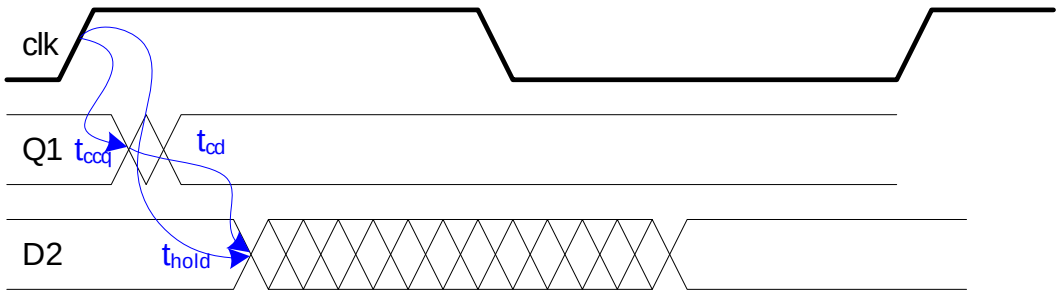
- $t_{pd} \leq T_c - (?)$
- $t_{pd} \leq T_c - (t_{setup} + t_{pcq})$
- $(t_{setup} + t_{pcq})$ is the sequencing overhead
- setup check



# Min-Delay:Flip-Flops

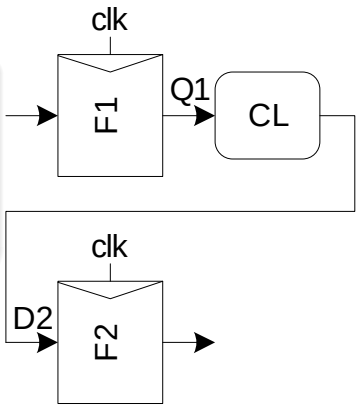
- $t_{cd} \geq ?$
- $t_{cd} \geq t_{hold} - t_{ccq}$
- hold check

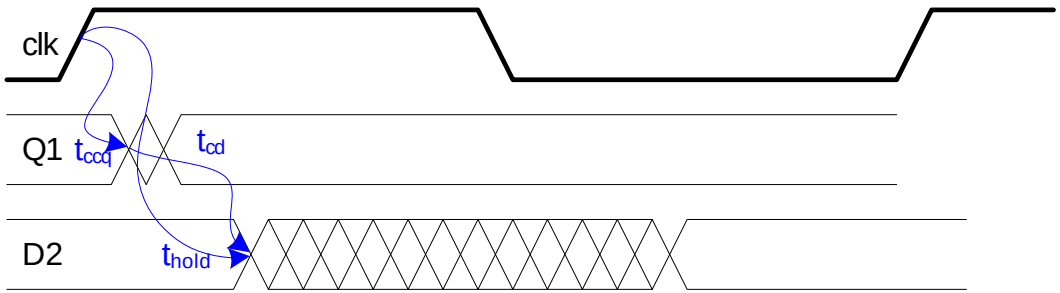


# Min-Delay:Flip-Flops

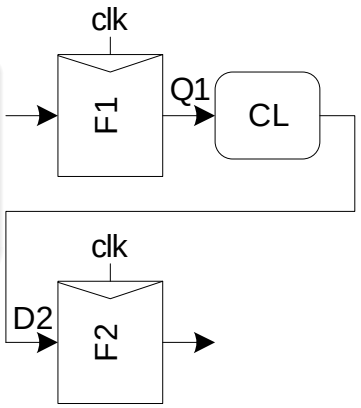
- $t_{cd} \geq ?$
- $t_{cd} \geq t_{hold} - t_{ccq}$
- hold check

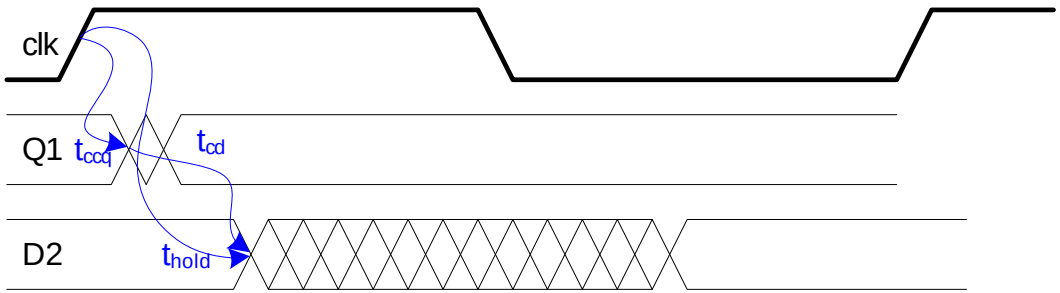


# Min-Delay:Flip-Flops

- $t_{cd} \geq ?$
- $t_{cd} \geq t_{hold} - t_{ccq}$
- hold check



# References I



N. Weste and D. Harris, CMOS VLSI Design: A circuits and systems perspective, 4th Ed., Addison-Wesley, 2010.


R. J. Baker, CMOS Circuit Design, Layout and Simulation, revised 2nd Edition, Wiley-IEEE, 2008.