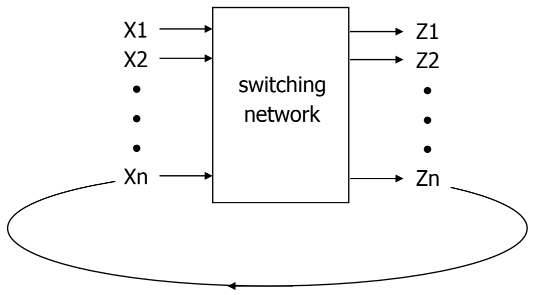
Sequential logic

- Sequential circuits
    - simple circuits with feedback
    - latches
    - edge-triggered flip-flops
- Timing methodologies
    - cascading flip-flops for proper operation
    - clock skew
- Basic registers
    - shift registers
    - simple counters
- Hardware description languages and sequential logic

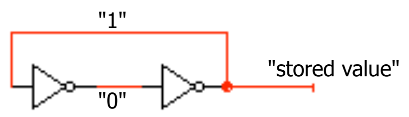
Circuits with feedback

- How to control feedback?
    - what stops values from cycling around endlessly

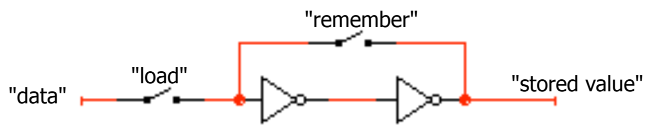


Simplest circuits with feedback

- Two inverters form a static memory cell
  - will hold value as long as it has power applied

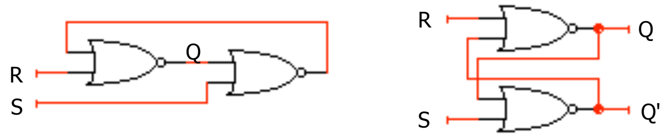


- How to get a new value into the memory cell?
  - selectively break feedback path
  - load new value into cell

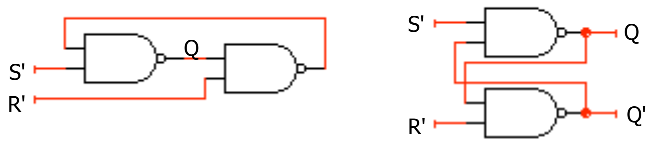


Memory with cross-coupled gates

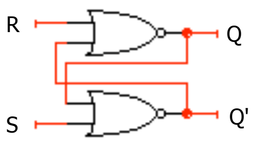
- Cross-coupled NOR gates
  - similar to inverter pair, with capability to force output to 0 (reset=1) or 1 (set=1)

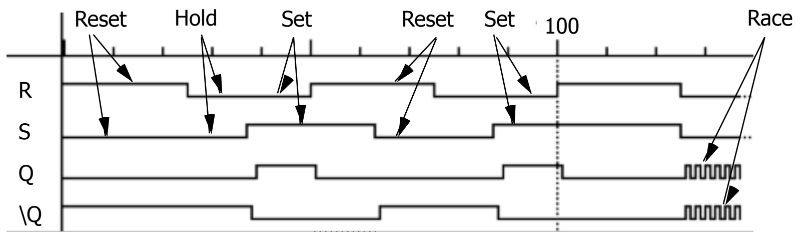


- Cross-coupled NAND gates
  - similar to inverter pair, with capability to force output to 0 (reset=0) or 1 (set=0)

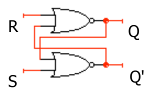


Timing behavior



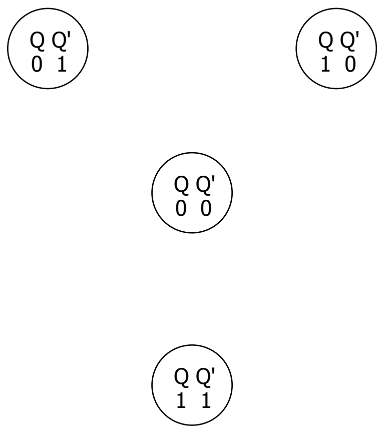


State behavior or R-S latch

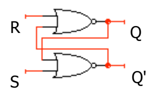


- Truth table of R-S latch behavior

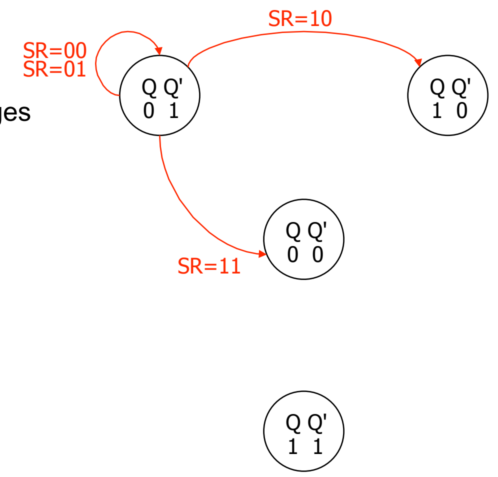
S	R	Q
0	0	hold
0	1	0
1	0	1
1	1	unstable



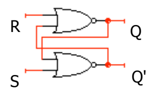
Theoretical R-S latch behavior



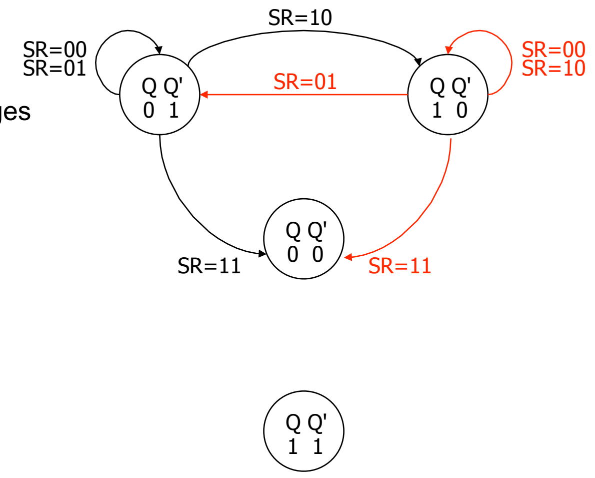
- State diagram
  - states: possible values of outputs
  - transitions: changes based on inputs



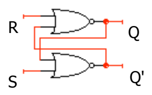
Theoretical R-S latch behavior



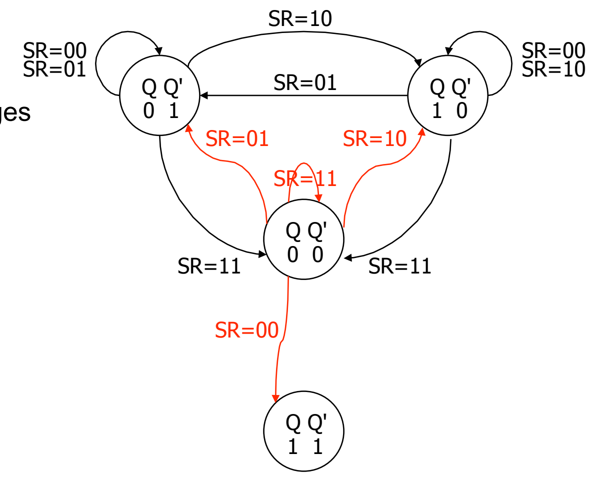
- State diagram
  - states: possible values of outputs
  - transitions: changes based on inputs



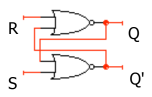
Theoretical R-S latch behavior



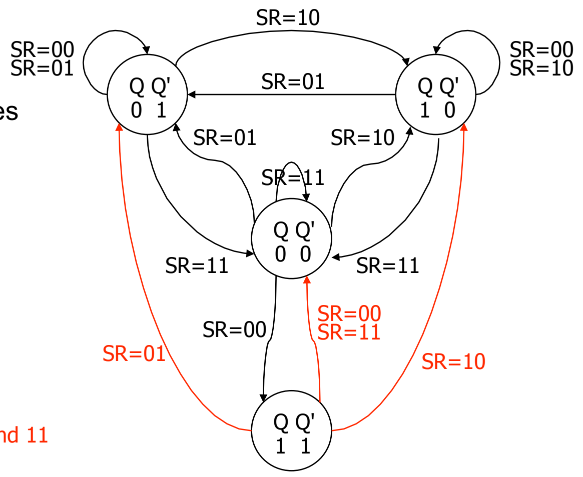
- State diagram
  - states: possible values of outputs
  - transitions: changes based on inputs



Theoretical R-S latch behavior

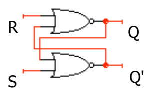


- State diagram
  - states: possible values of outputs
  - transitions: changes based on inputs

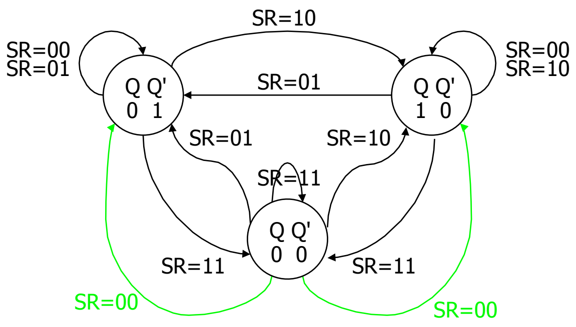


possible oscillation between states 00 and 11

Observed R-S latch behavior

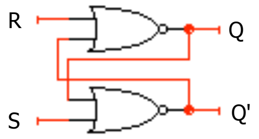


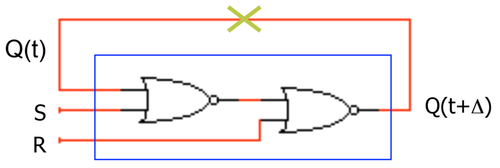
- Very difficult to observe R-S latch in the 1-1 state
  - one of R or S usually changes first
- Ambiguously returns to state 0-1 or 1-0
  - a so-called "race condition"
  - or non-deterministic transition



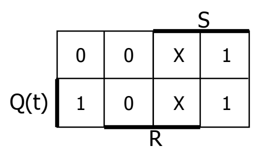
R-S latch analysis

- Break feedback path





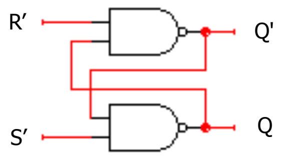
S	R	Q(t)	Q(t+Δ)	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	not allowed
1	1	1	X	

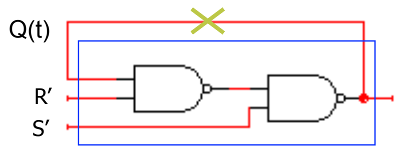


			S	S
	0	0	X	1
Q(t)	1	0	X	1
		R	R	

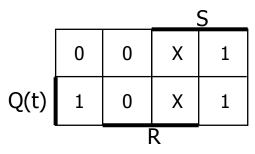
characteristic equation
$Q(t+\Delta) = S + R' Q(t)$

R-S latch using NAND gates





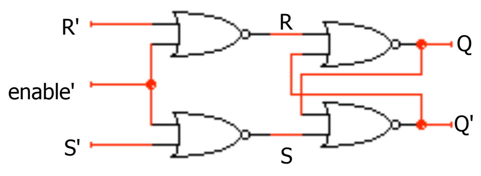
S	R	S′	R′	Q(t)	Q(t+Δ)	
0	0	1	1	0	0	hold
0	0	1	1	1	1	
0	1	1	0	0	0	reset
0	1	1	0	1	0	
1	0	0	1	0	1	set
1	0	0	1	1	1	
1	1	0	0	0	X	not allowed
1	1	0	0	1	X	

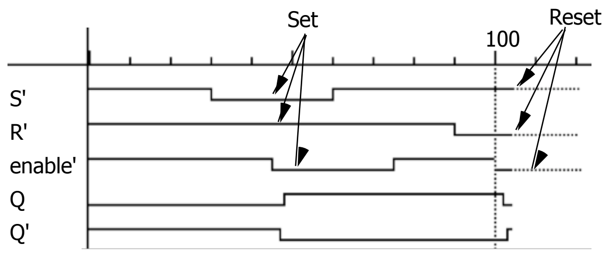


characteristic equation
$Q(t+\Delta) = S + R'\,Q(t)$

Gated R-S latch

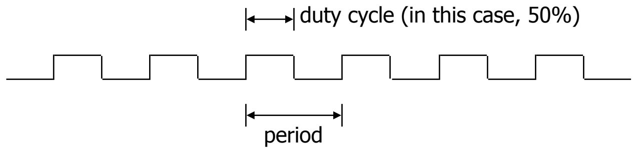
- Control when R and S inputs matter
  - otherwise, the slightest glitch on R or S while enable is low could cause change in stored value





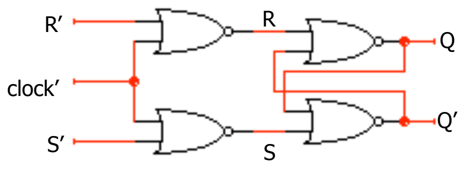
Clocks

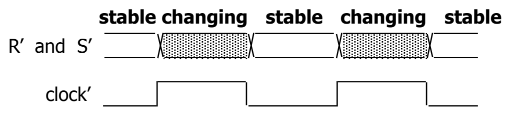
- Used to keep time
  - wait long enough for inputs (R' and S') to settle
  - then allow to have effect on value stored
- Clocks are regular periodic signals
  - period (time between clock ticks)
  - duty-cycle (clock pulse width - expressed as % of period)



Clocks (cont’d)

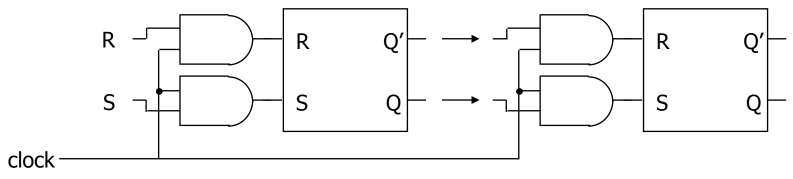
- Controlling an R-S latch with a clock
  - R and S can’t change while clock is active
  - only have half of clock period for signal changes to propagate
  - signals must be stable for the other half of clock period





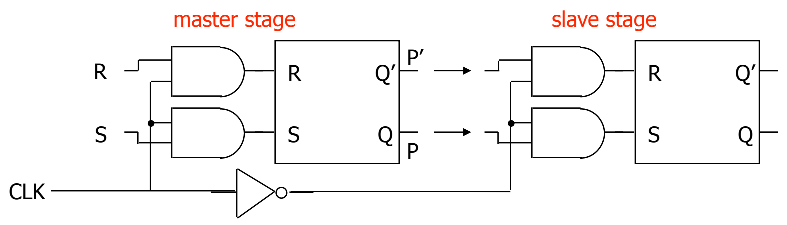
Cascading latches

- Connect output of one latch to input of another
- How to stop changes from racing through chain?
  - need to be able to control flow of data from one latch to the next
  - move one latch per clock period
  - have to worry about logic between latches (arrows) that is too fast



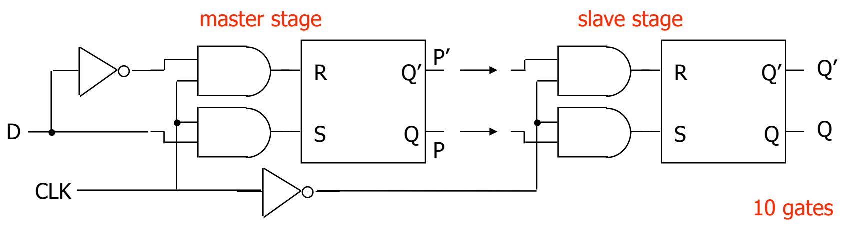
Master-slave structure

- Break flow by alternating clocks (like an air-lock)
  - use positive clock to latch inputs into one R-S latch
  - use negative clock to change outputs with another R-S latch
- View pair as one basic unit
  - master-slave flip-flop
  - twice as much logic
  - output changes a few gate delays after the falling edge of clock but does not affect any cascaded flip-flops



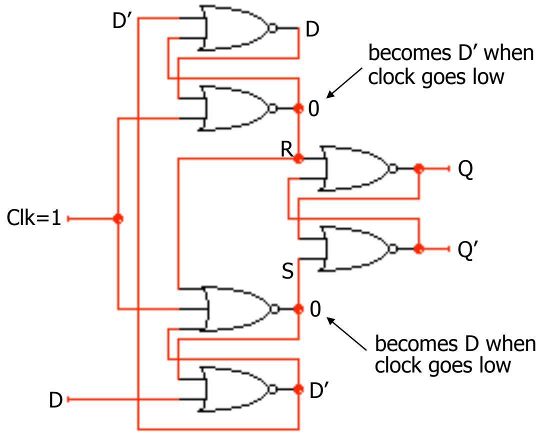
D flip-flop

- Make S and R complements of each other
  - eliminates 1s catching problem
  - can't just hold previous value
    - must have new value ready every clock period
  - value of D just before clock goes low is what is stored in flip-flop
  - can make R-S flip-flop by adding logic to make D = S + R' Q



Edge-triggered flip-flops using gates

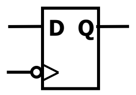
- Only 6 gates
  - sensitive to inputs only near edge of clock signal (not while high)



negative edge-triggered D flip-flop (D-FF)

4-5 gate delays

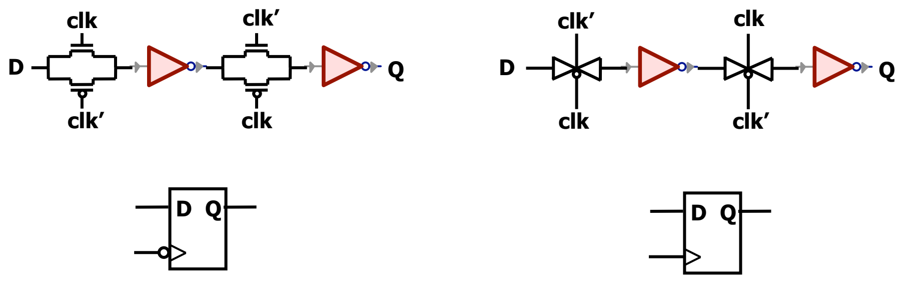
must respect setup and hold time constraints to successfully capture input



characteristic equation
Q(t+1) = D

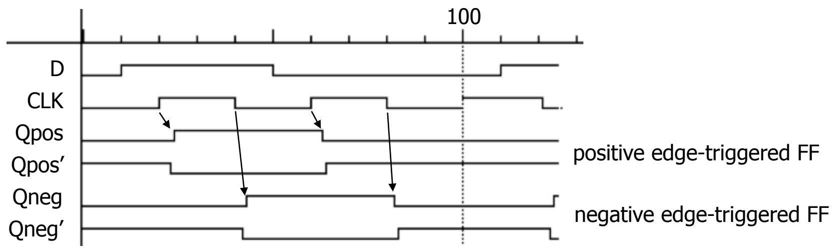
Edge-triggered flip-flops using transistors

- Only 8 transistors



Edge-triggered flip-flops (cont'd)

- Positive edge-triggered
  - inputs sampled on rising edge; outputs change after rising edge
- Negative edge-triggered flip-flops
  - inputs sampled on falling edge; outputs change after falling edge

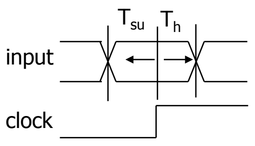


Timing methodologies

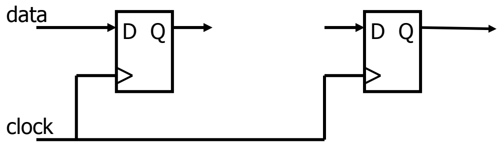
- Rules for interconnecting components and clocks
  - guarantee proper operation of system when strictly followed
- Approach depends on building blocks used for memory elements
  - we'll focus on systems with edge-triggered flip-flops
    - found in programmable logic devices such as our FPGA
  - many custom integrated circuits focus on level-sensitive latches
    - they are much smaller (but need to be careful about timing)
- Basic rules for correct timing:
  - (1) correct inputs, with respect to clock, are provided to the flip-flops
  - (2) no flip-flop changes state more than once per clocking event

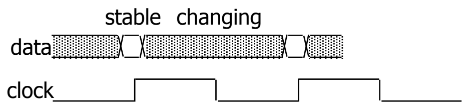
Timing methodologies (cont'd)

- Definition of terms
  - Clock: periodic event, causes state of memory element to change
    can be rising edge or falling edge or high level or low level
  - Setup time: minimum time before the clocking event by which the
    input must be stable (T_{setup} or T_{su})
  - Hold time: minimum time after the clocking event until which the
    input must remain stable (T_{hold} or T_h)

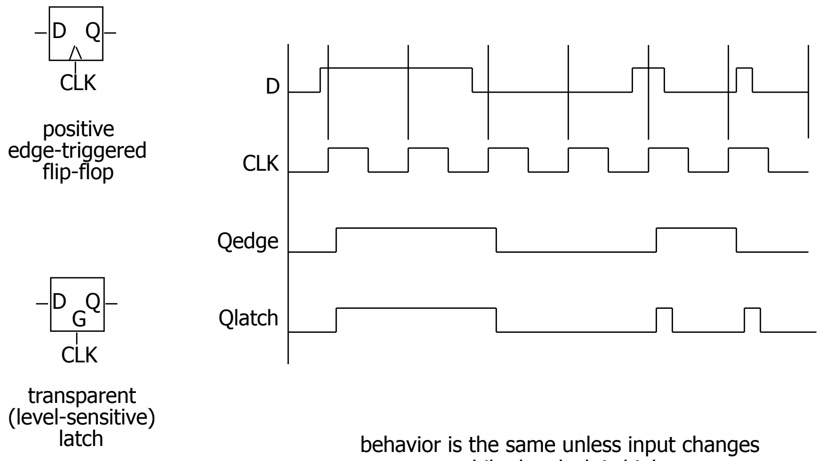


there is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognized





Comparison of latches and flip-flops


D
Q
CLK
positive edge-triggered flip-flop
D
Q
G
CLK
transparent (level-sensitive) latch
D
CLK
Qedge
Qlatch

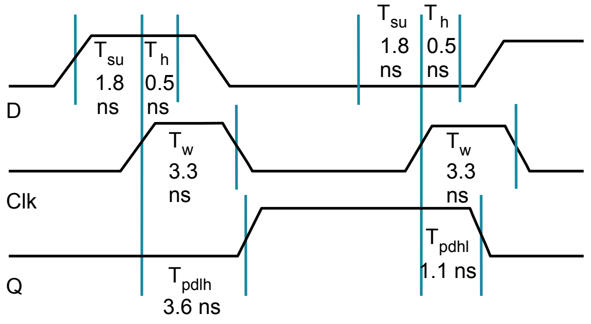
behavior is the same unless input changes while the clock is high

Comparison of latches and flip-flops (cont'd)

Type	When inputs are sampled	When output is valid
unclocked latch	always	propagation delay from input change
level-sensitive latch	clock high (Tsu/Th around falling edge of clock)	propagation delay from input change or clock edge (whichever is later)
master-slave flip-flop	clock hi-to-lo transition (Tsu/Th around falling edge of clock)	propagation delay from falling edge of clock
negative edge-triggered flip-flop	clock hi-to-lo transition (Tsu/Th around falling edge of clock)	propagation delay from falling edge of clock

Typical timing specifications

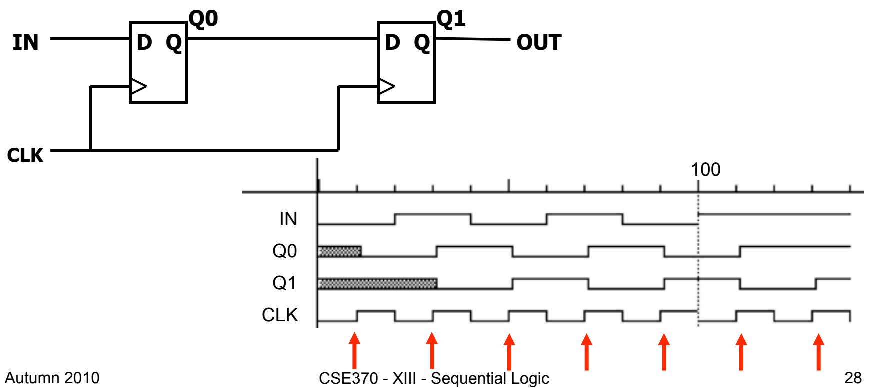
- Positive edge-triggered D flip-flop
  - setup and hold times
  - minimum clock width
  - propagation delays (low to high, high to low, max and typical)



all measurements are made from the clocking event (the rising edge of the clock)

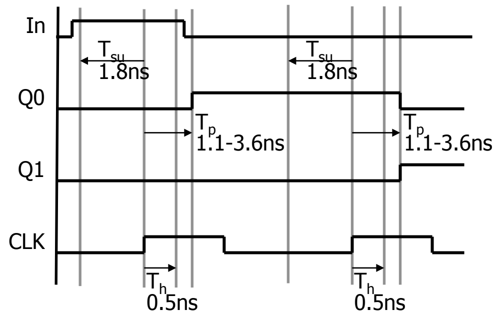
Cascading edge-triggered flip-flops

- Shift register
  - new value goes into first stage
  - while previous value of first stage goes into second stage
  - consider setup/hold/propagation delays (prop must be > hold)



Cascading edge-triggered flip-flops (cont'd)

- Why this works
  - propagation delays exceed hold times
  - clock width constraint exceeds setup time
  - this guarantees following stage will latch current value before it changes to new value

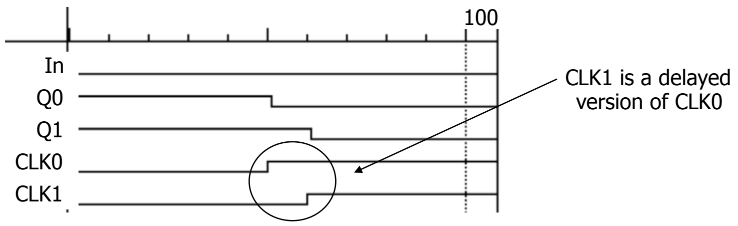


timing constraints guarantee proper operation of cascaded components

assumes infinitely fast distribution of the clock

Clock skew

- When it doesn't work
  - correct behavior assumes next state of all storage elements determined by all storage elements at the same time
  - this is difficult in high-performance systems because time for clock to arrive at flip-flop is comparable to delays through logic
  - effect of skew on cascaded flip-flops:



original state: IN = 0, Q0 = 1, Q1 = 1 expected next state: Q0 = 0, Q1 = 1
due to skew, next state becomes: Q0 = 0, Q1 = 0 (0 races through two FFs instead of one)

Summary of latches and flip-flops

- Development of D-FF
  - level-sensitive used in custom integrated circuits
    - can be made with 8 switches
  - edge-triggered used in modern programmable logic devices
  - good choice for data storage register
- Historically, JK-FF was popular but now never used
  - similar to RS but with 1-1 being used to toggle output
  - JK = 00 hold, 01 reset, 10 set, 11 toggle (complement state)
  - good in days of SSI (more complex input function: D = J Q’ + K’ Q)
  - can always be implemented using D-FF, if needed
- Preset and clear inputs are highly desirable on flip-flops
  - used at start-up or to reset system to a known state

Flip-flop features

- Reset (set state to 0) – R
  - synchronous: D_{new} = R' • D (when next clock edge arrives)
  - asynchronous: doesn't wait for clock, quick but dangerous
- Preset or set (set state to 1) – S (or sometimes P)
  - synchronous: D_{new} = D + S (when next clock edge arrives)
  - asynchronous: doesn't wait for clock, quick but dangerous
- Both reset and preset
  - D_{new} = R' • D + S (set-dominant)
  - D_{new} = R' • D + R'S (reset-dominant)
- Selective input capability (input enable or load) – LD or EN
  - multiplexor at input: D_{new} = LD' • Q + LD • D
  - load may or may not override reset/set (usually R/S have priority)
- Complementary outputs – Q and Q'