
Ch 6. Sequential Logic

Sequential logic overview

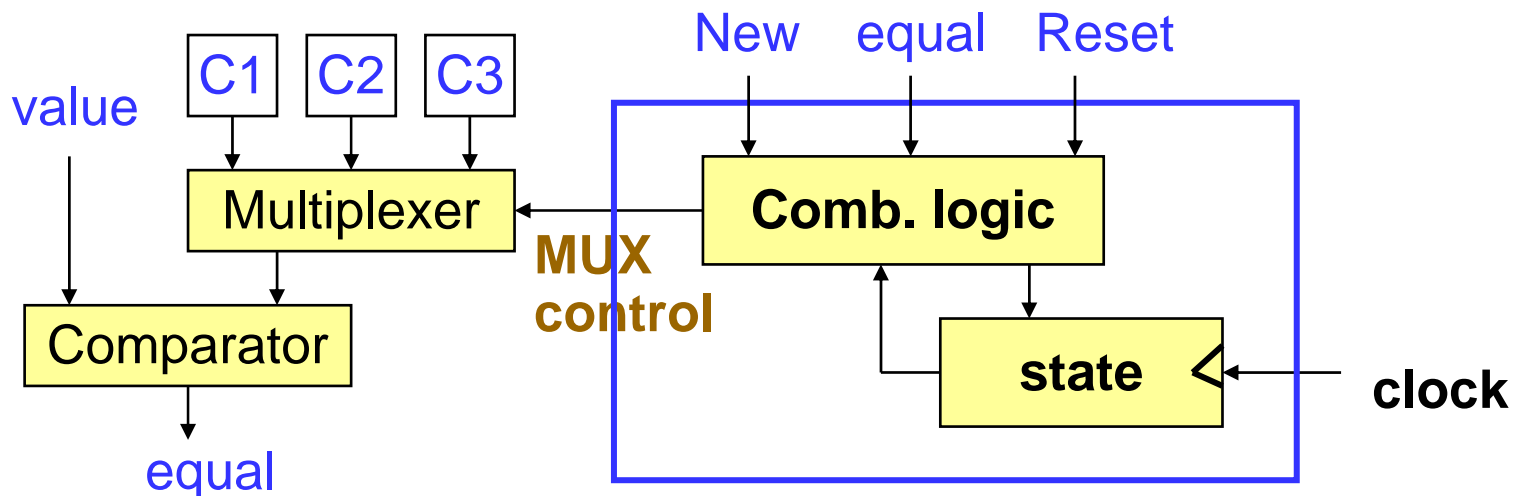
- Sequential circuits
 - simple circuits with feedback
 - latches
 - edge-triggered flip-flops
- Timing methodologies
 - cascading flip-flops for proper operation
 - clock skew
- Asynchronous inputs
 - metastability and synchronization
- Basic registers
 - shift registers
 - simple counters
- Hardware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

Sequential logic

- System has “states”
- New component in sequential logic networks:
 - storage elements to remember the current state
- Outputs depend on inputs and the states
- New state is a function of the inputs and the old state
 - i.e., the state is fed back as inputs!

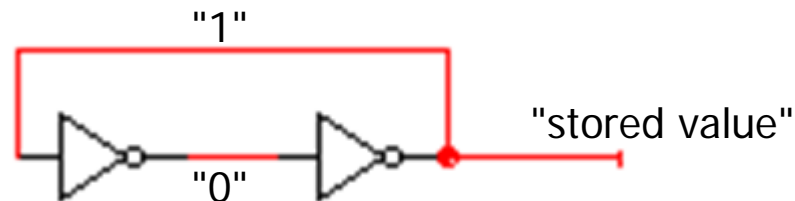
Sequential circuits

- Circuits with feedback
 - outputs = $f(\text{inputs, past inputs, past outputs})$
 - basis for building "memory" into logic circuits
 - door combination lock is an example of a sequential circuit
 - state is memory
 - state is an "output" and an "input" to combinational logic
 - combination storage elements are also memory

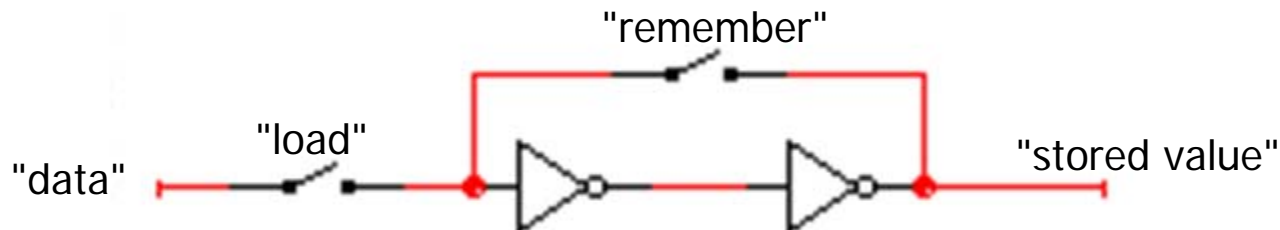


Simplest circuits with feedback

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



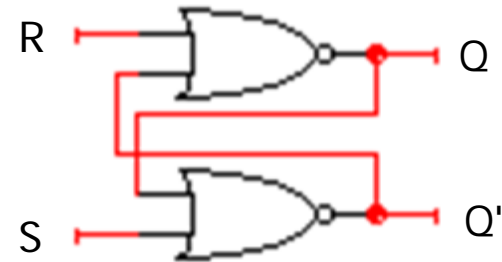
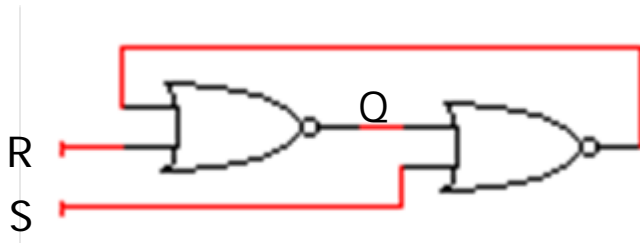
- How to get a new value into the memory cell?
 - selectively break feedback path
 - load new value into cell



Memory with cross-coupled gates

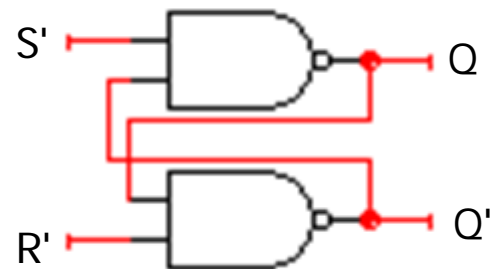
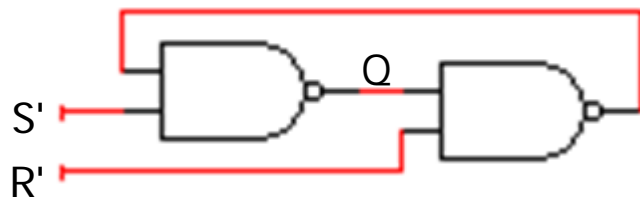
■ Cross-coupled NOR gates

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

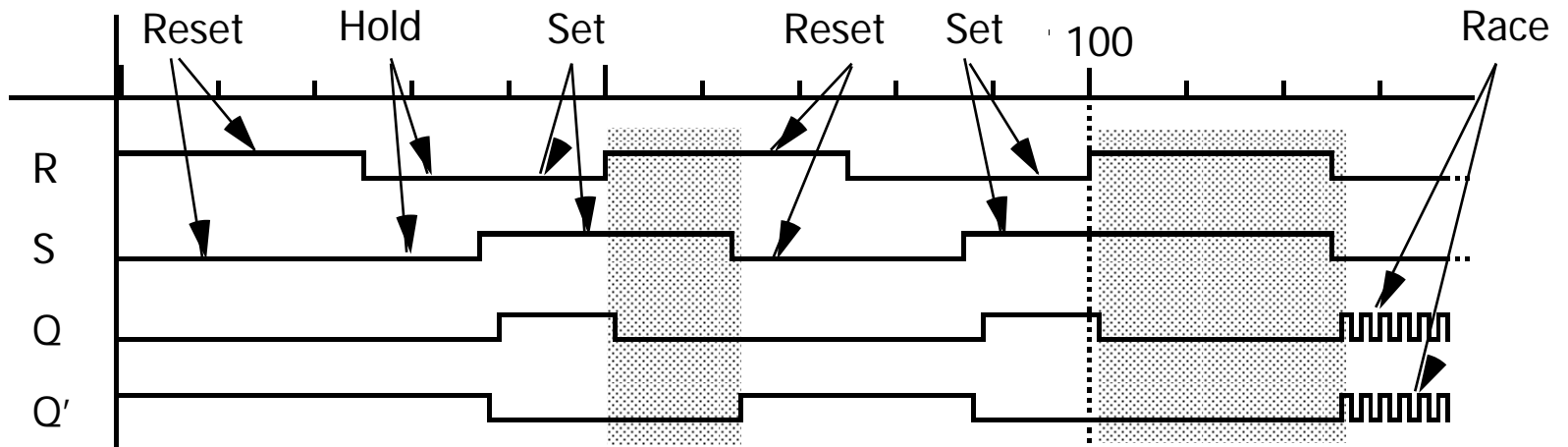
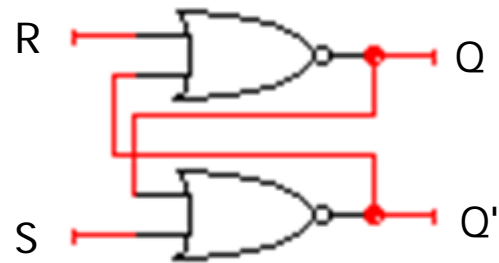


■ Cross-coupled NAND gates

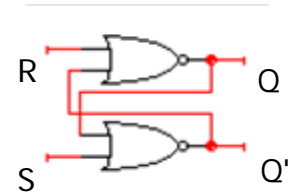
- similar to inverter pair, with capability to force output to 0 (reset=0) or 1 (set=0)



Timing behavior

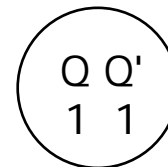
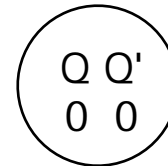
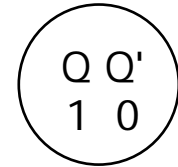
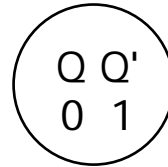


State behavior or R-S latch

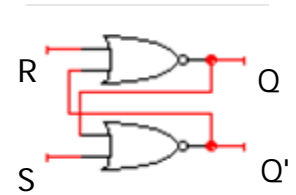


■ Truth table of R-S latch behavior

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

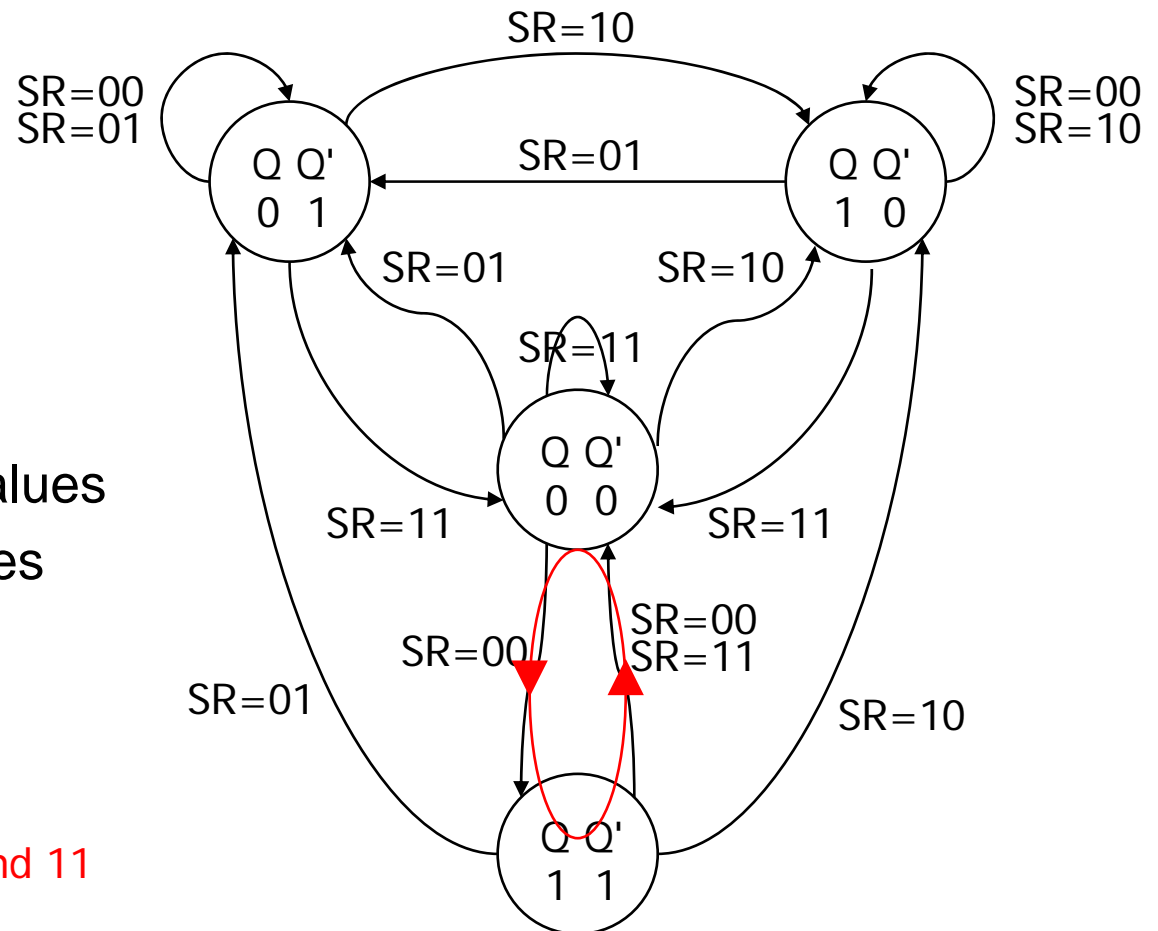


Theoretical R-S latch behavior



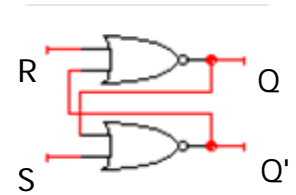
■ State diagram

- states: possible values
- transitions: changes based on inputs

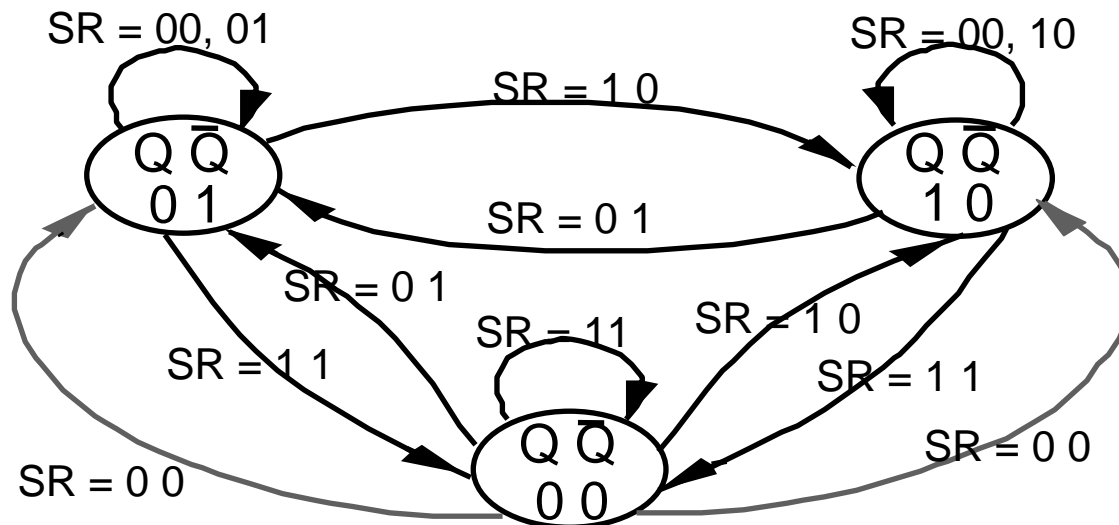


possible oscillation
between states 00 and 11

Observed R-S latch behavior

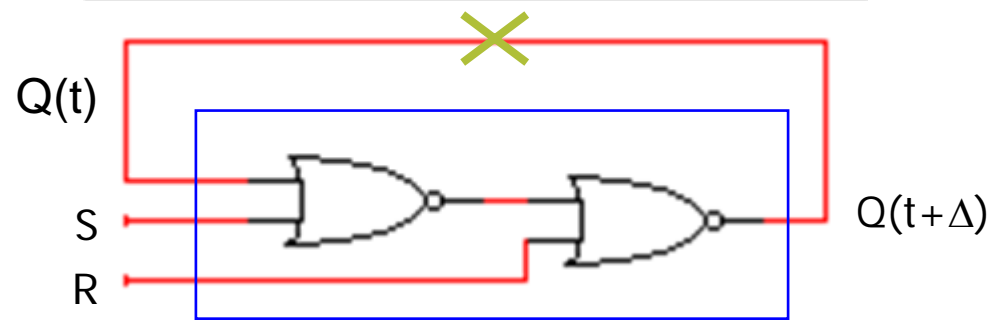
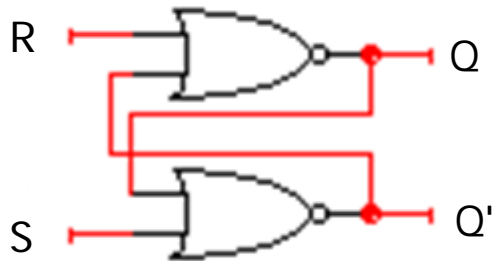


- Very difficult to observe R-S latch in the 1-1 state
 - one of R or S usually changes first
- Ambiguously returns to state 0-1 or 1-0
 - a so-called "race condition"
 - or non-deterministic transition



R-S latch analysis

- Break feedback path



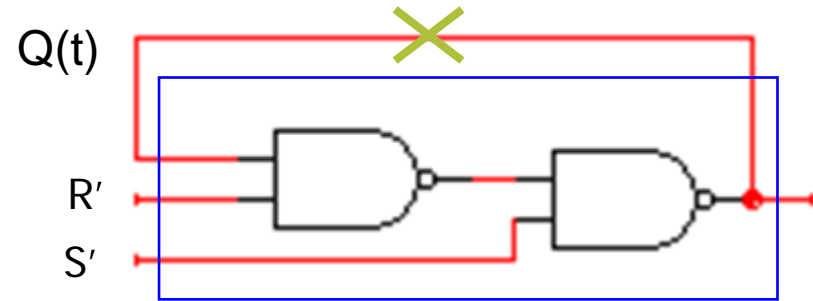
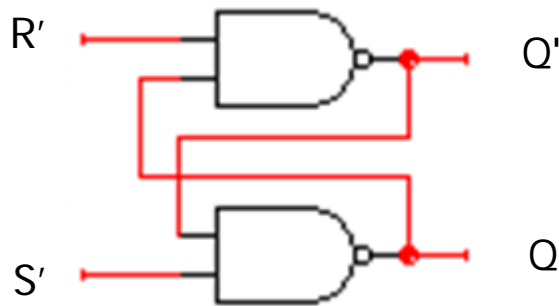
S	R	Q(t)	Q(t+Δ)	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	not allowed
1	1	1	X	

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

characteristic equation

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

Activity: R-S latch using NAND gates



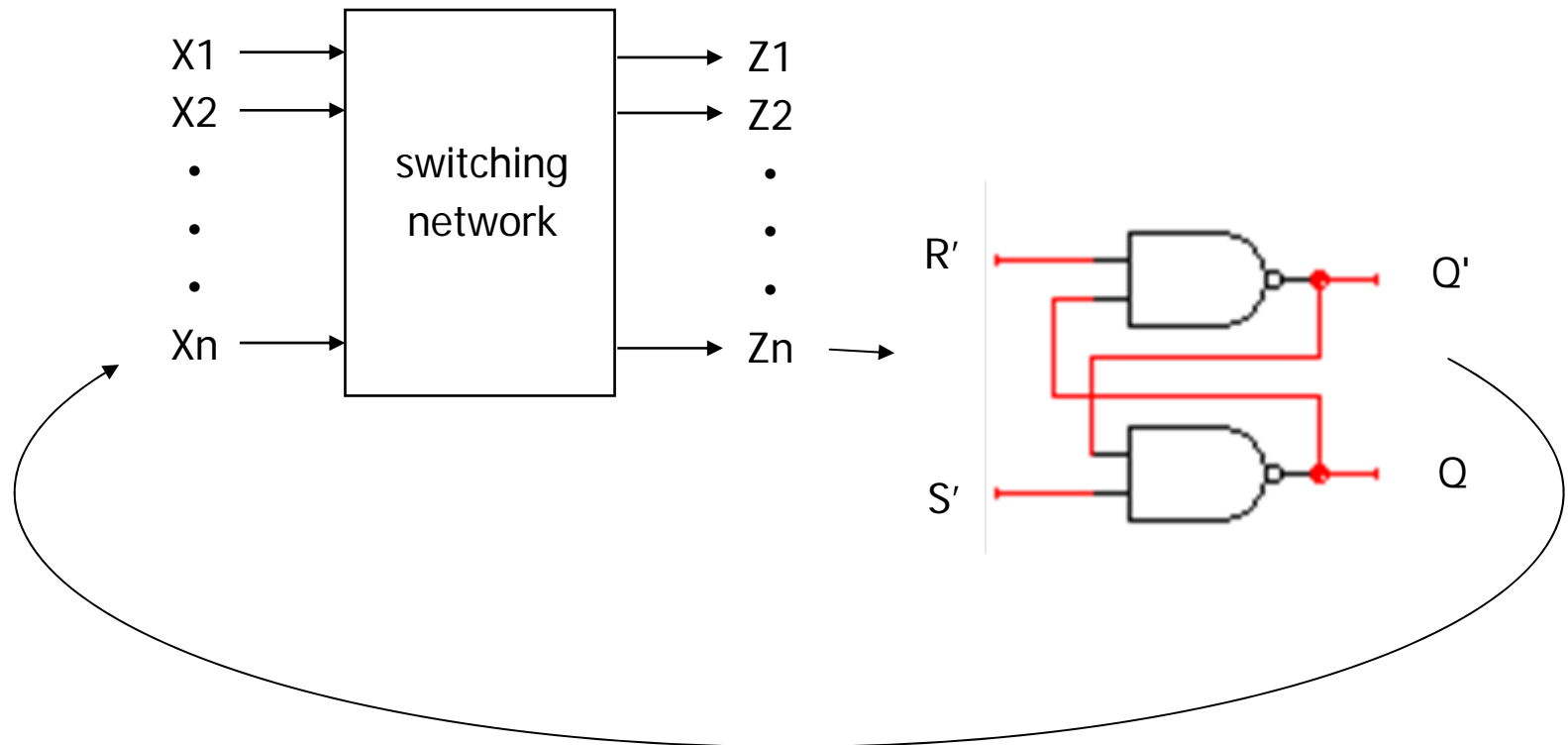
S	R	S'	R'	Q(t)	Q(t+Δ)	
0	0	1	1	0	0	hold
0	0	1	1	1	1	
0	1	1	0	0	0	reset
0	1	1	0	1	0	
1	0	0	1	0	1	set
1	0	0	1	1	1	
1	1	0	0	0	X	not allowed
1	1	0	0	1	X	

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

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

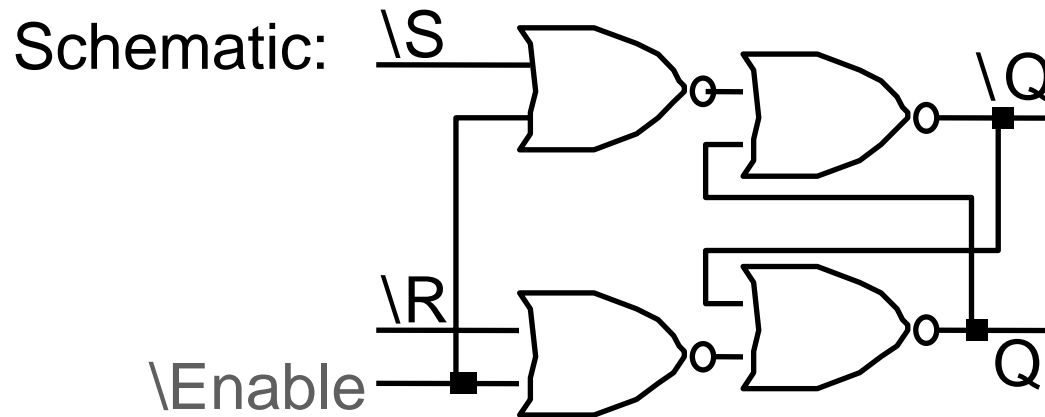
Circuits with feedback

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

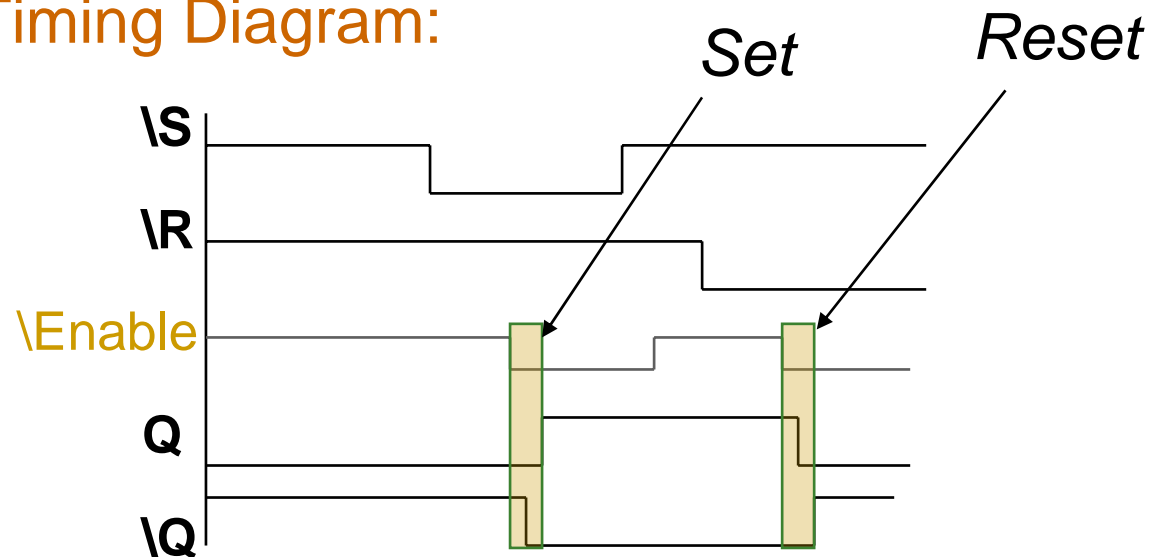


Gated or Level-Sensitive Latch

- Latch is allowed to change state when the enable signal is asserted.
- Propagation delay from enable signal to output changes is shown in the diagram.

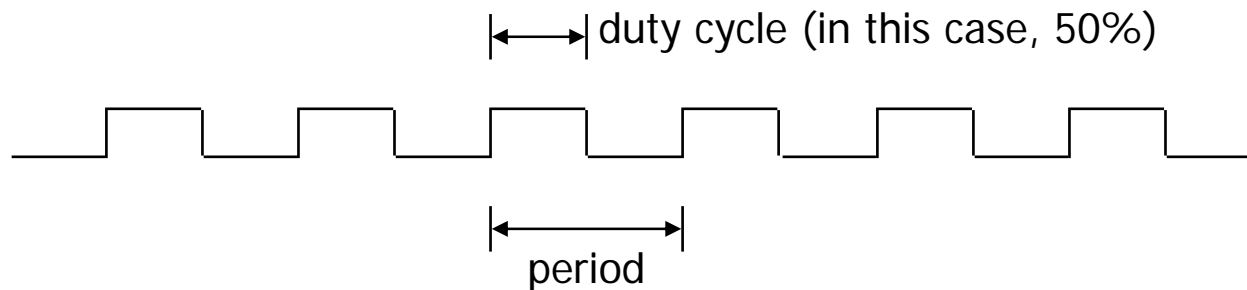


Timing Diagram:



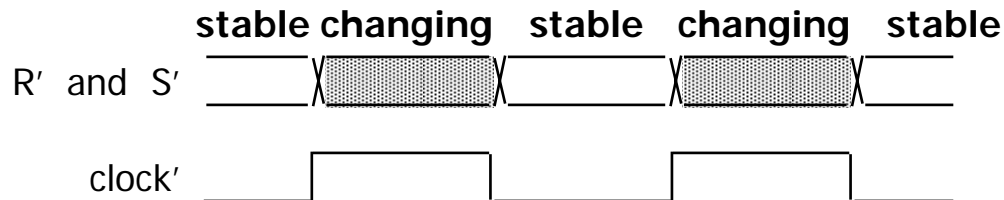
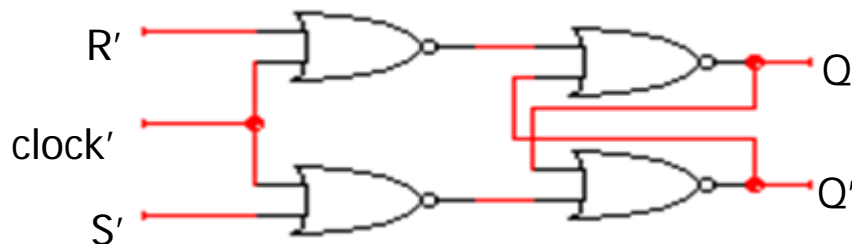
Clocks

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



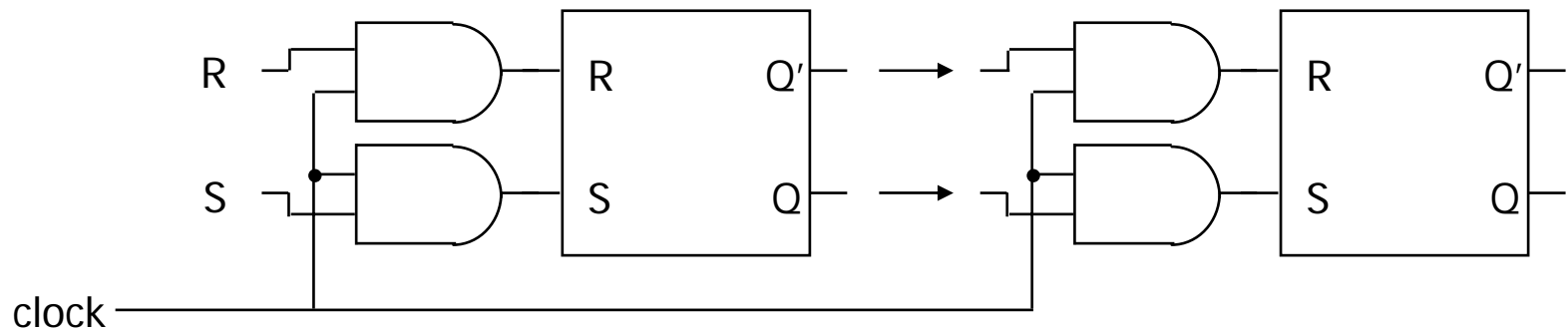
Clocks (cont'd)

- Controlling an R-S latch with a clock
 - can't let R and S change while clock is active (allowing R and S to pass)
 - only have half of clock period for signals to change
 - signals must be stable for the other half of clock period



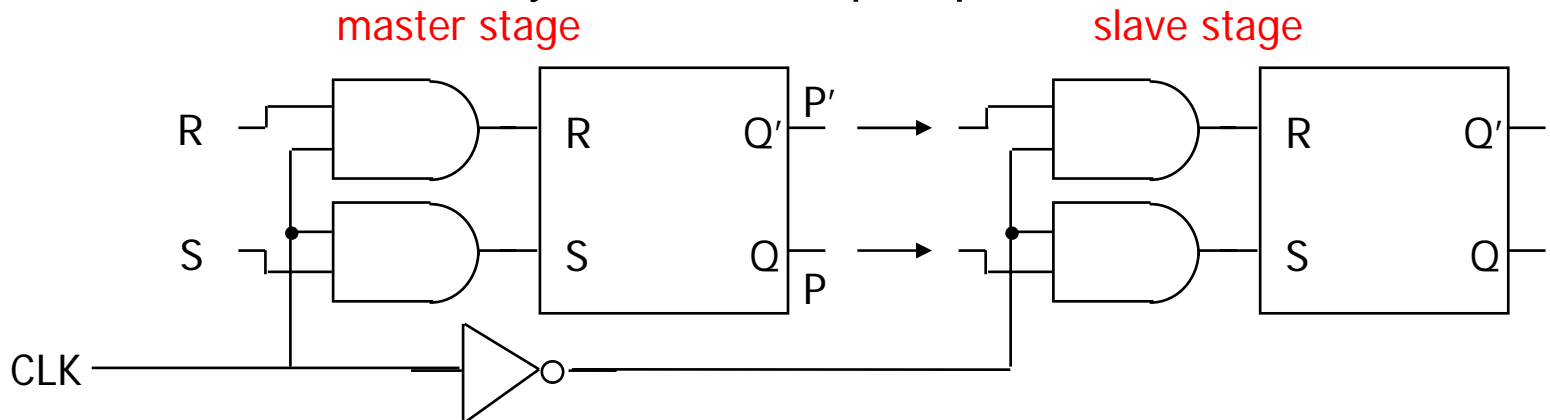
Combining latches

- 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



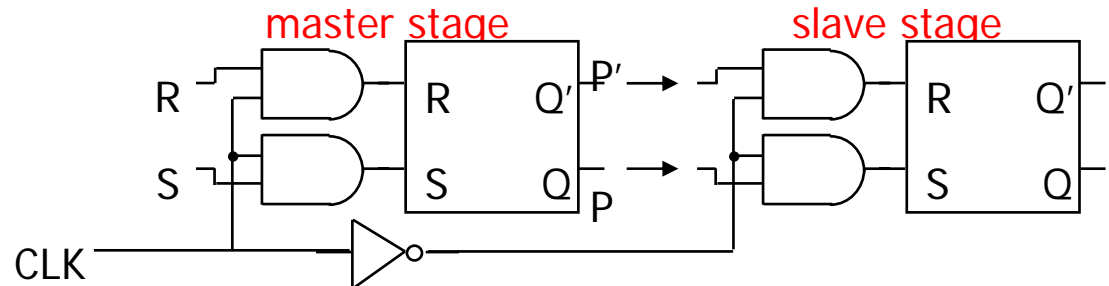
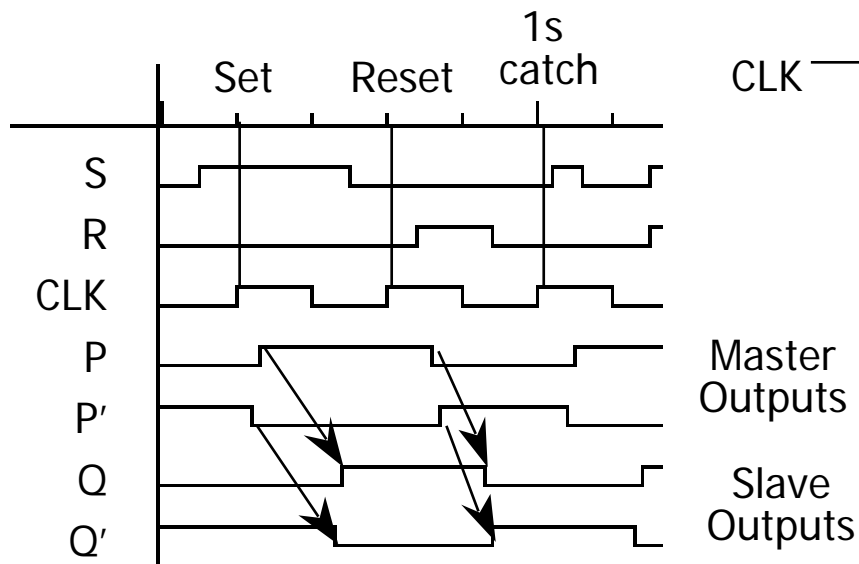
Master-slave structure

- 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



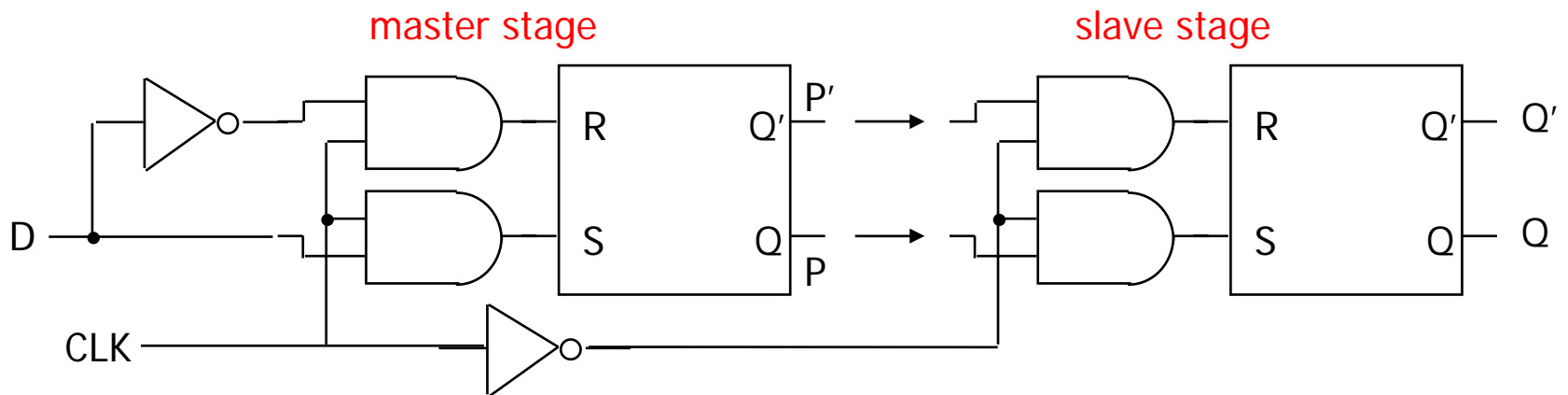
The 1s catching problem

- In first R-S stage of master-slave FF
 - 0-1-0 glitch on R or S while clock is high is "caught" by master stage
 - leads to constraints on logic to be hazard-free



D flip-flop

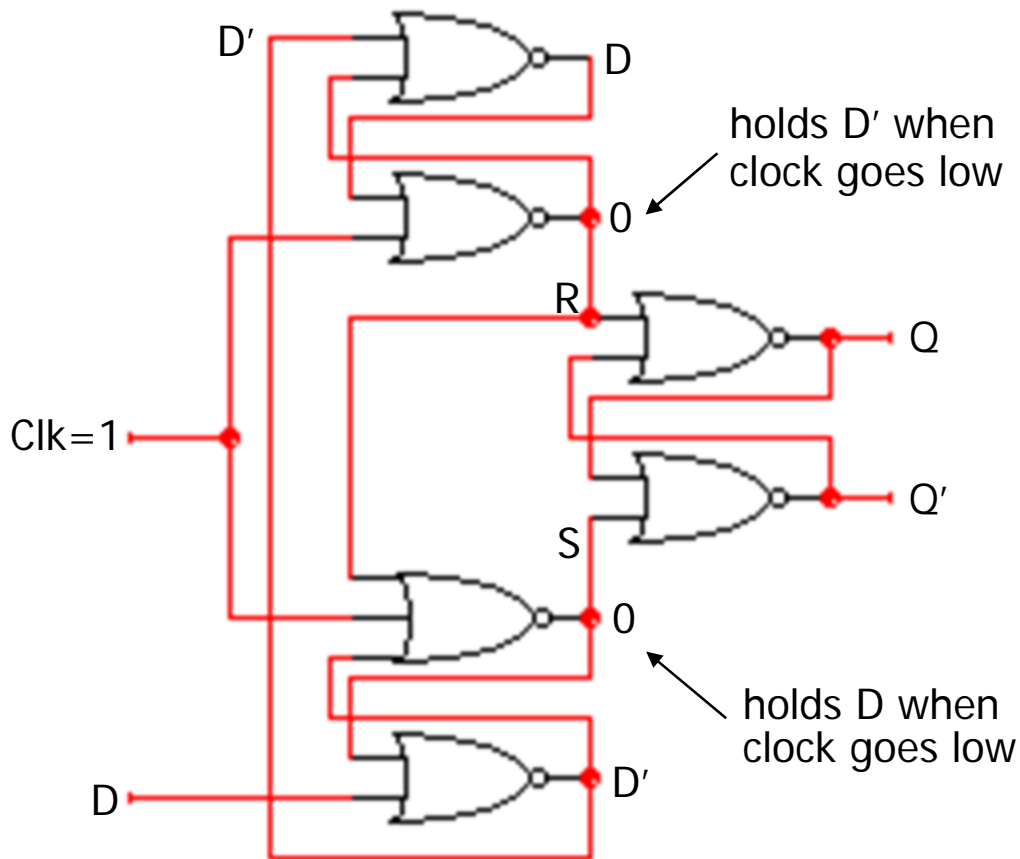
- Make S and R complements of each other
 - eliminates 1s catching problem
 - can't just hold previous value (must have new value ready every clock period)
 - value of D just before clock goes low is what is stored in flip-flop
 - can make R-S flip-flop by adding logic to make $D = S + R' Q$



10 gates

Edge-triggered flip-flops

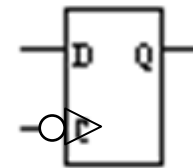
- More efficient solution: only 6 gates
 - sensitive to inputs only near edge of clock signal (not while high)



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

4-5 gate delays

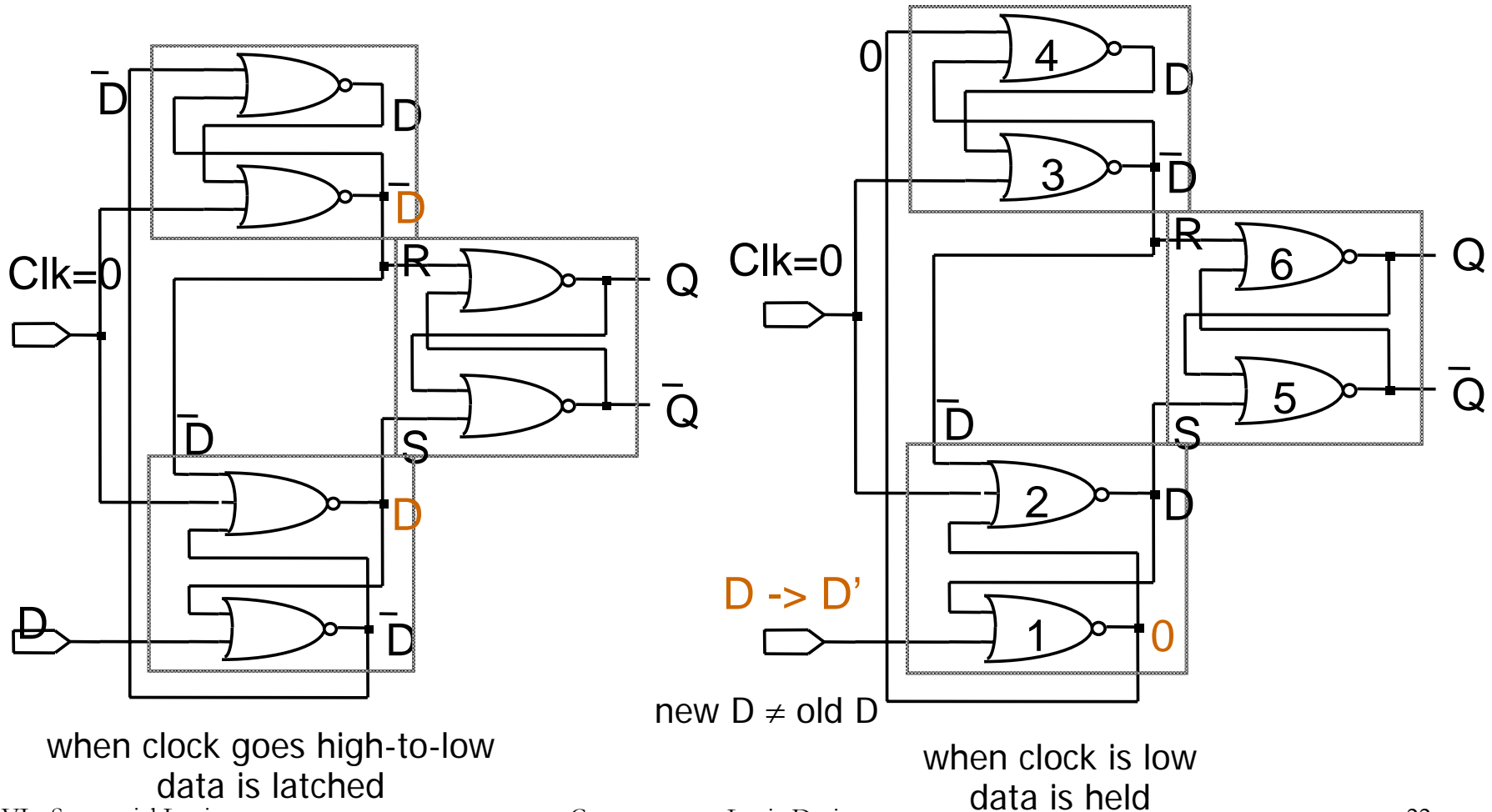
must respect setup and hold time constraints to successfully capture input



characteristic equation
 $Q(t+1) = D$

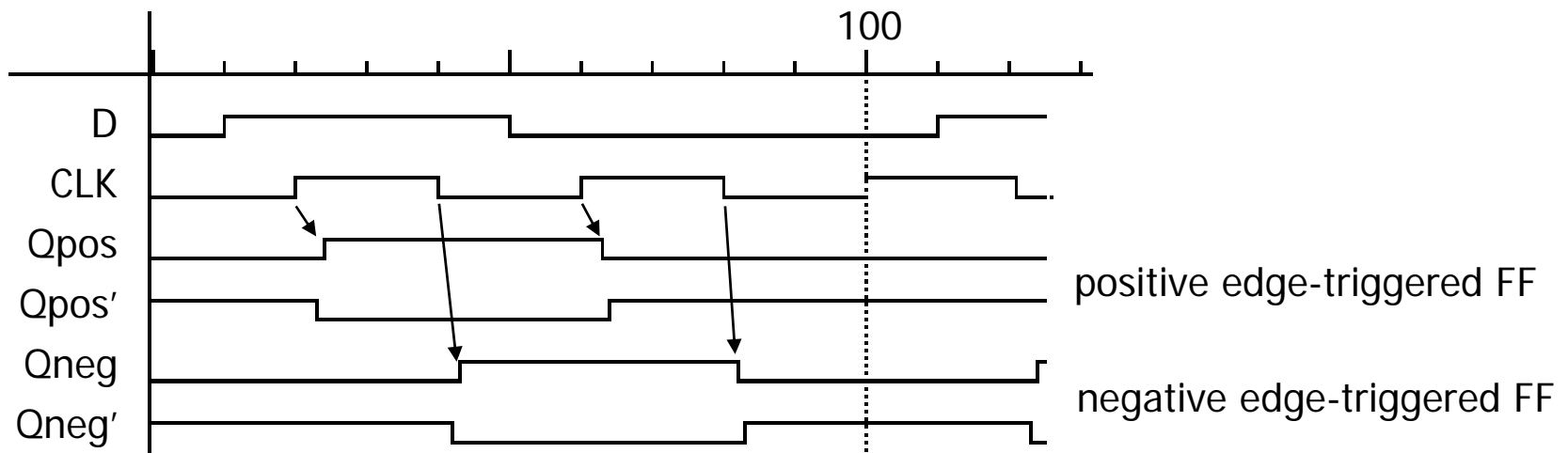
Edge-triggered flip-flops (cont'd)

- Step-by-step analysis

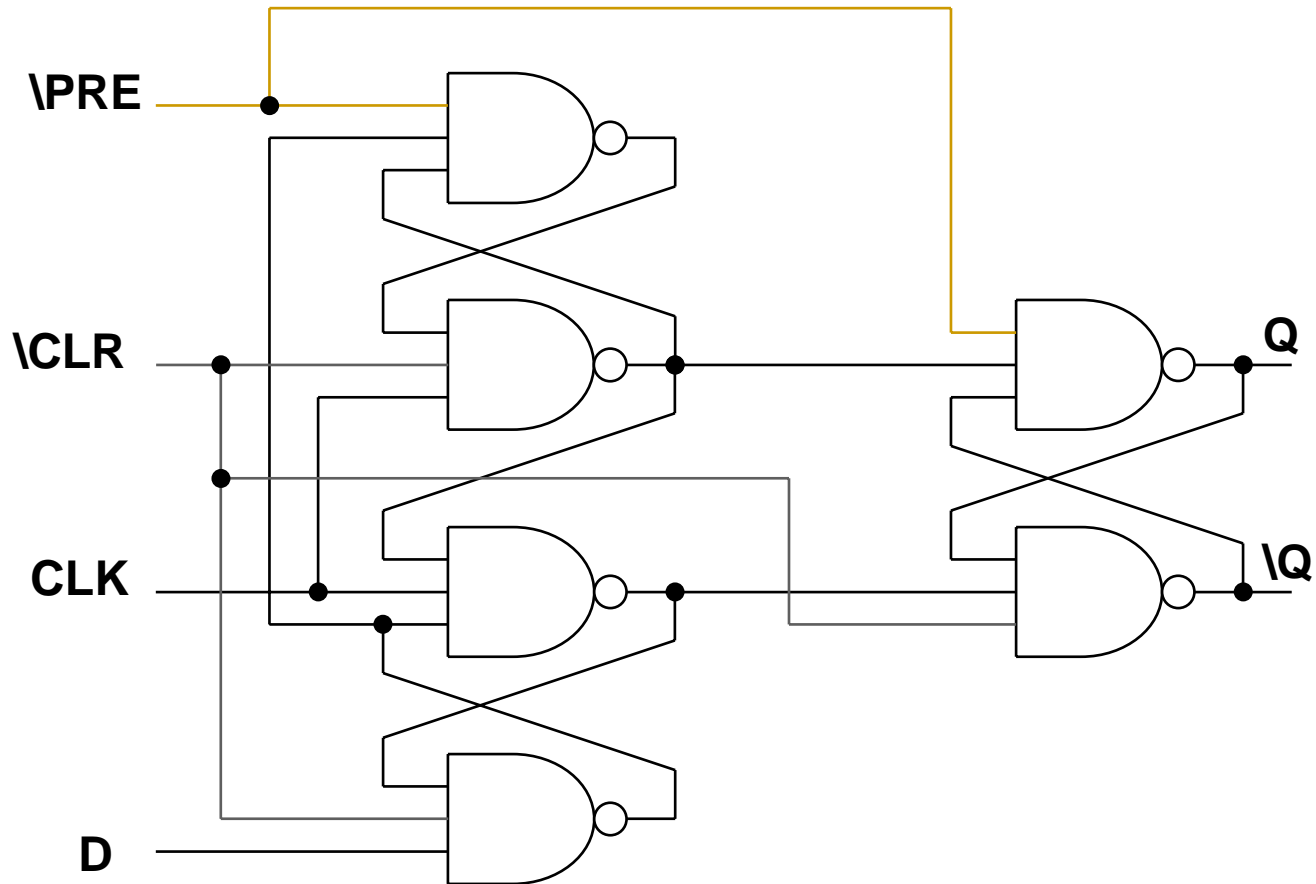


Edge-triggered flip-flops (cont'd)

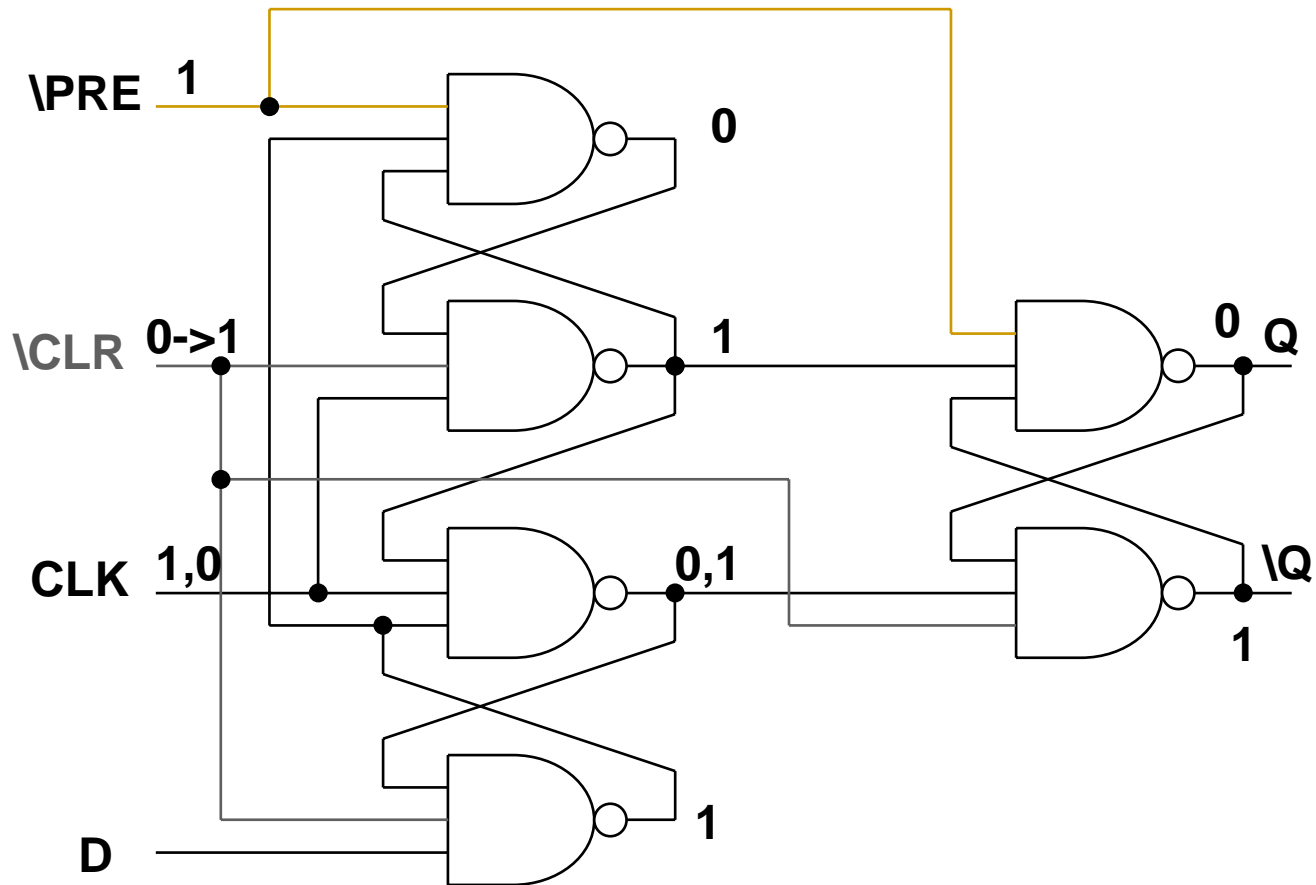
- Positive edge-triggered
 - inputs sampled on rising edge; outputs change after rising edge
- Negative edge-triggered flip-flops
 - inputs sampled on falling edge; outputs change after falling edge



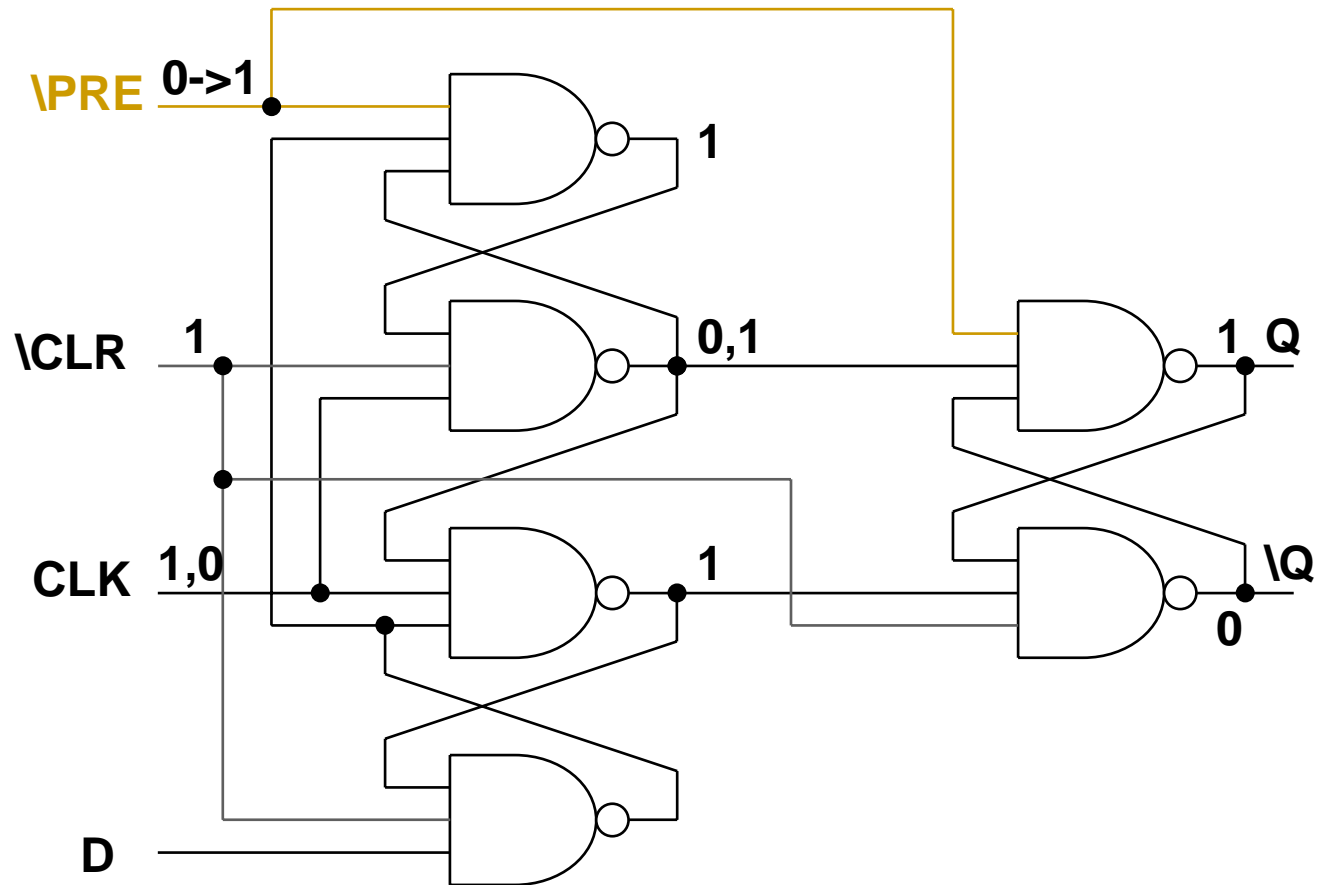
Positive edge-triggered D flip-flop (74LS74)



Positive edge-triggered D flip-flop (74LS74)

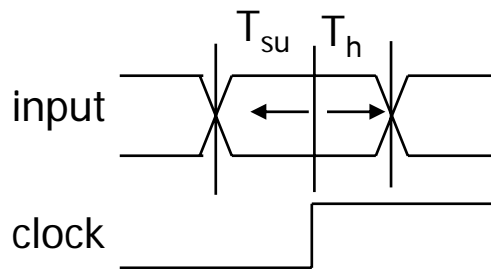


Positive edge-triggered D flip-flop (74LS74)

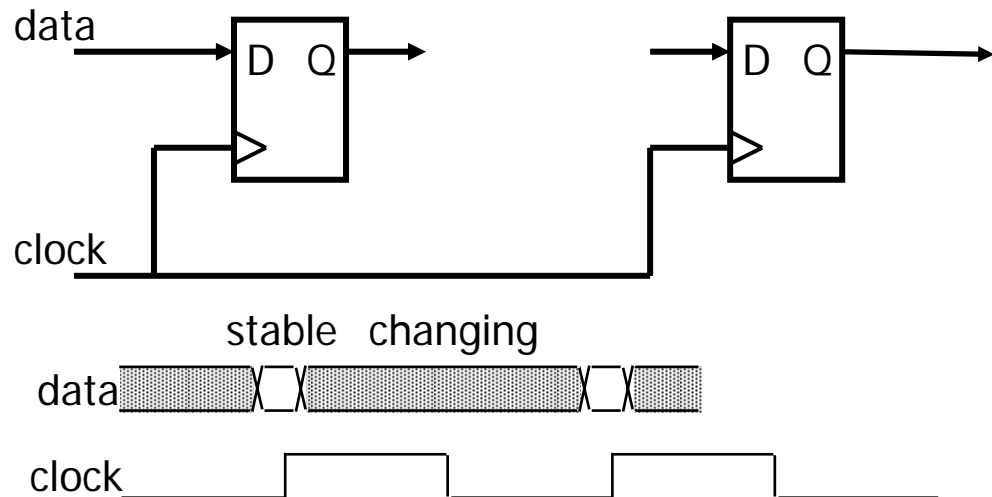


Timing definitions

- ❑ 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_{su})
- ❑ hold time: minimum time after the clocking event until which the input must remain stable (T_h)

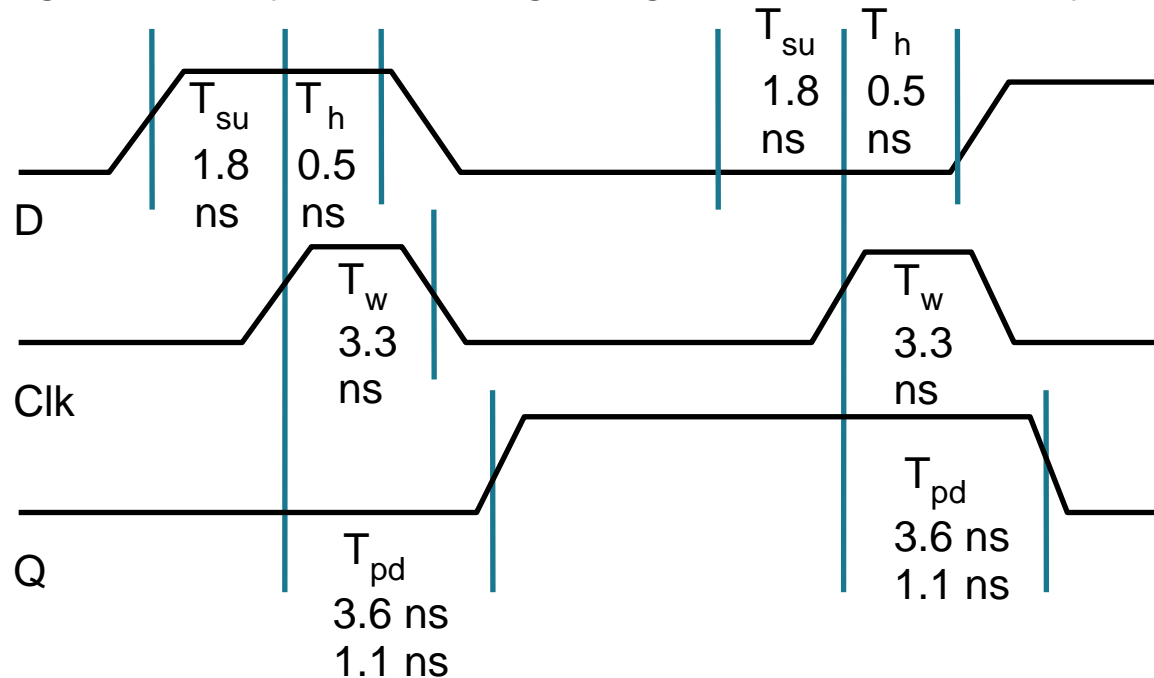


there is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognized



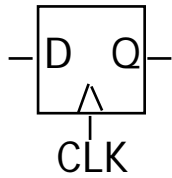
Typical timing specifications

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

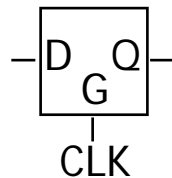


all measurements are made from the clocking event (the rising edge of the clock)

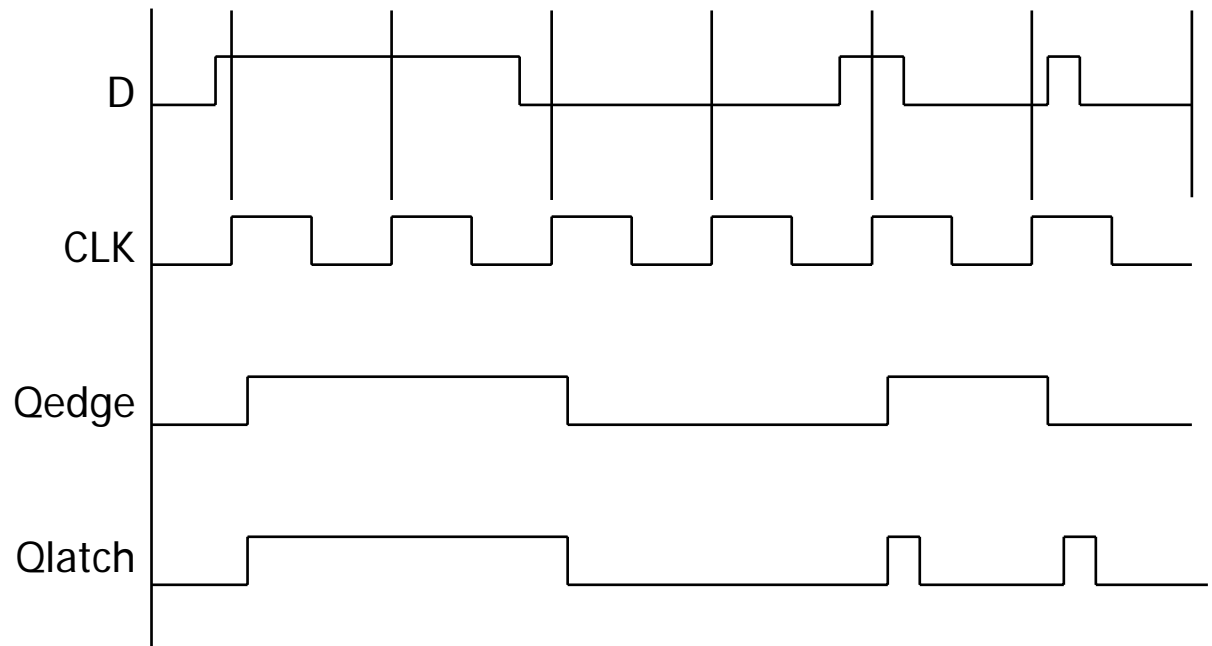
Comparison of latches and flip-flops



positive
edge-triggered
flip-flop



transparent
(level-sensitive)
latch



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

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

<u>Type</u>	<u>When inputs are sampled</u>	<u>When output is valid</u>
unclocked latch	always	propagation delay from input change
level-sensitive latch	clock high (T_{su}/T_h around falling edge of clock)	propagation delay from input change or clock edge
master-slave flip-flop	clock hi-to-lo transition (T_{su}/T_h around falling edge of clock)	propagation delay from falling edge of clock
negative edge-triggered flip-flop	clock hi-to-lo transition (T_{su}/T_h around falling edge of clock)	propagation delay from falling edge of clock

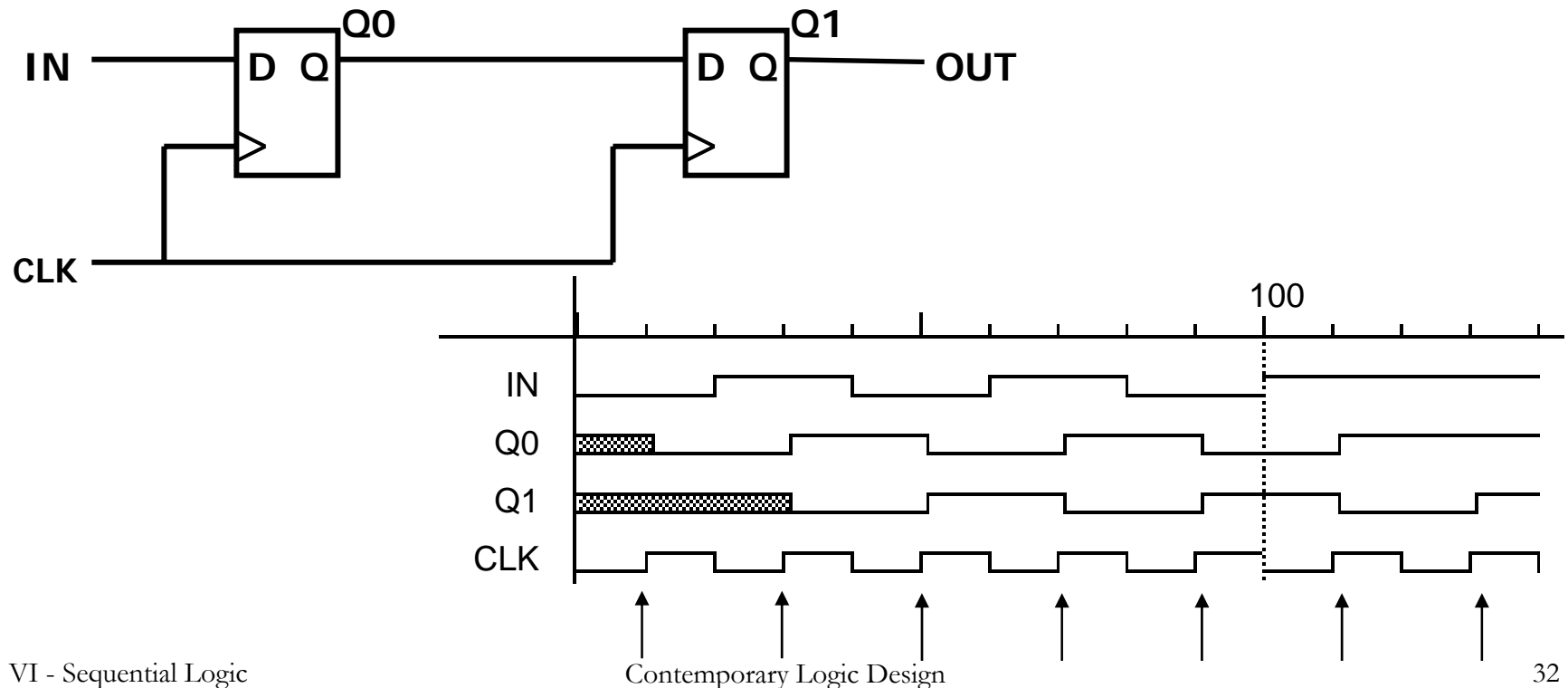
Timing methodologies

- Rules for interconnecting components and clocks
 - guarantee proper operation of system when strictly followed
- Proper operation:
 - (1) correct inputs, with respect to time, are provided to memory elements
 - (2) no memory element changes state more than once per clocking event
- Approach depends on building blocks used for memory elements
 - we'll focus on systems with edge-triggered flip-flops
 - found in programmable logic devices
 - many custom integrated circuits focus on level-sensitive latches

Cascading edge-triggered flip-flops

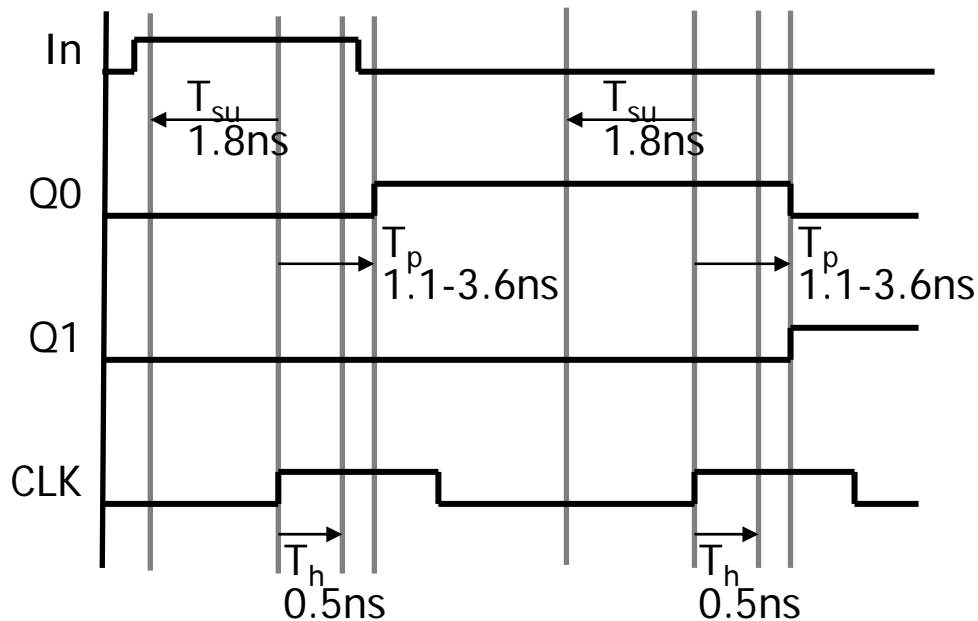
■ Shift register

- 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



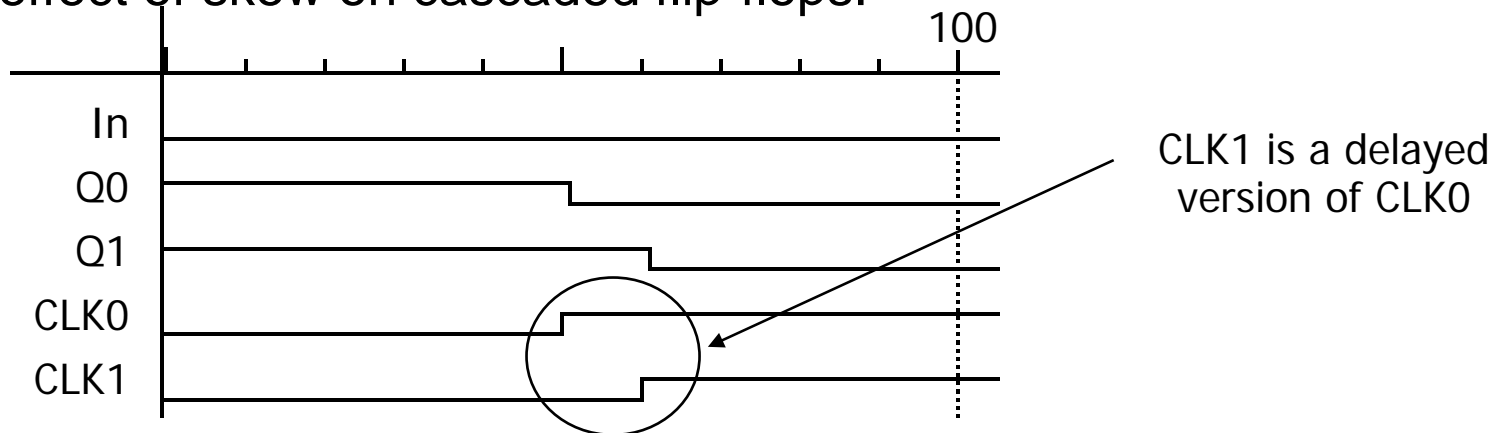
timing constraints
guarantee proper
operation of
cascaded components

assumes infinitely fast
distribution of the clock

Clock skew

■ The problem

- 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: $IN = 0, Q0 = 1, Q1 = 1$

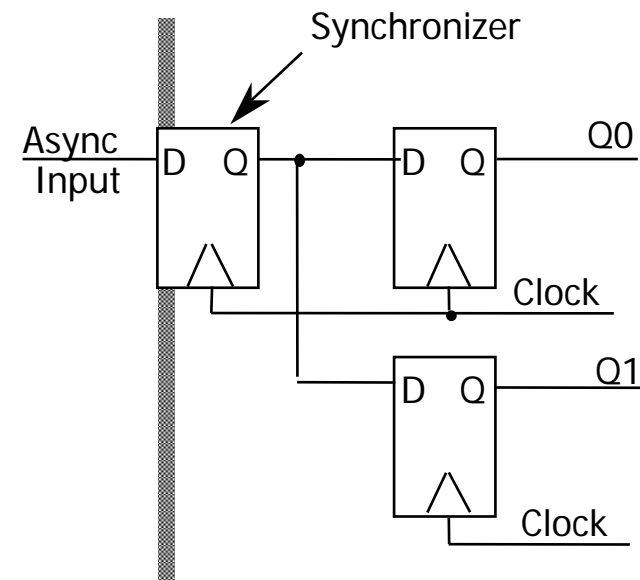
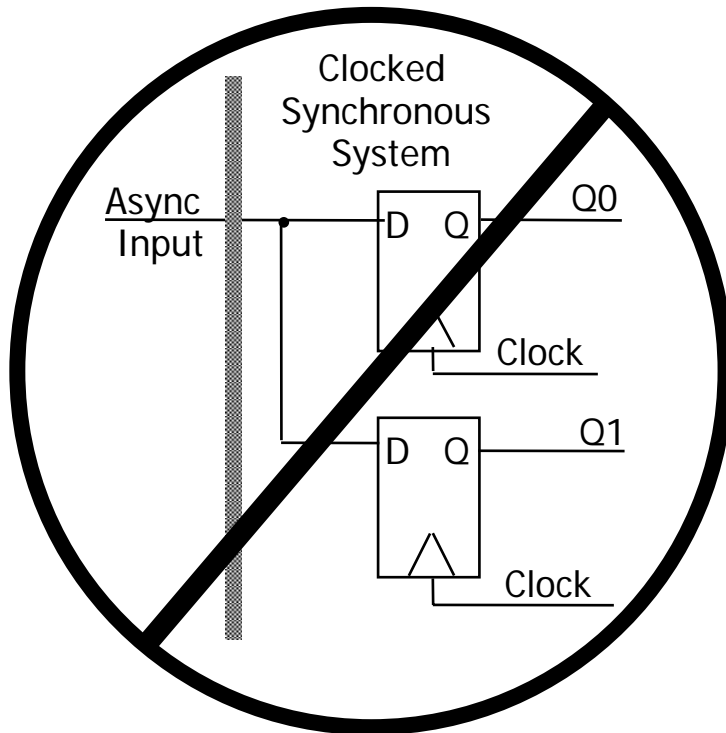
due to skew, next state becomes: $Q0 = 0, Q1 = 0$, and not $Q0 = 0, Q1 = 1$

Asynchronous inputs

- Clocked synchronous circuits
 - inputs, state, and outputs sampled or changed in relation to a common reference signal (called the clock)
 - e.g., master/slave, edge-triggered
- Asynchronous circuits
 - inputs, state, and outputs sampled or changed independently of a common reference signal (glitches/hazards a major concern)
 - e.g., R-S latch
- Asynchronous inputs to synchronous circuits
 - inputs can change at any time, will not meet setup/hold times
 - dangerous, synchronous inputs are greatly preferred
 - cannot be avoided (e.g., reset signal, memory wait, user input)

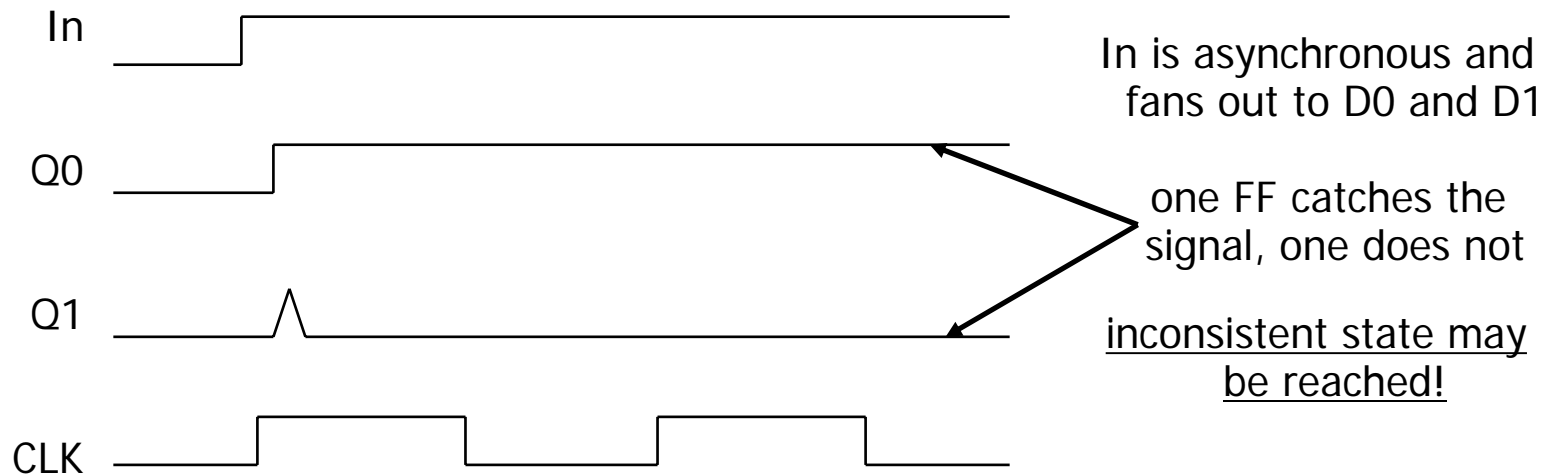
Handling asynchronous inputs

- Never allow asynchronous inputs to fan-out to more than one flip-flop
 - synchronize as soon as possible and then treat as synchronous signal



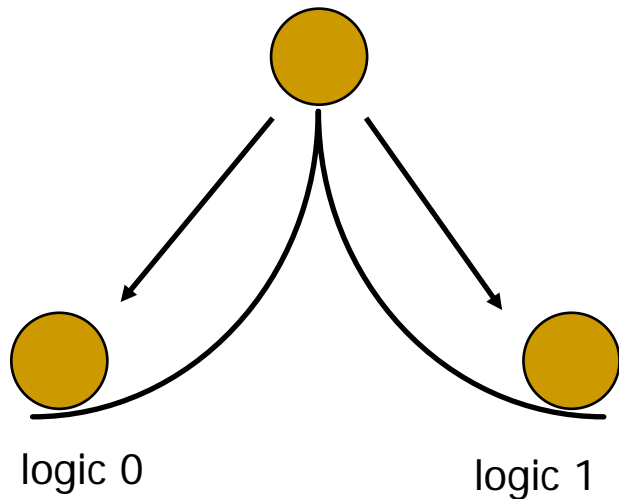
Handling asynchronous inputs (cont'd)

- What can go wrong?
 - input changes too close to clock edge (violating setup time constraint)

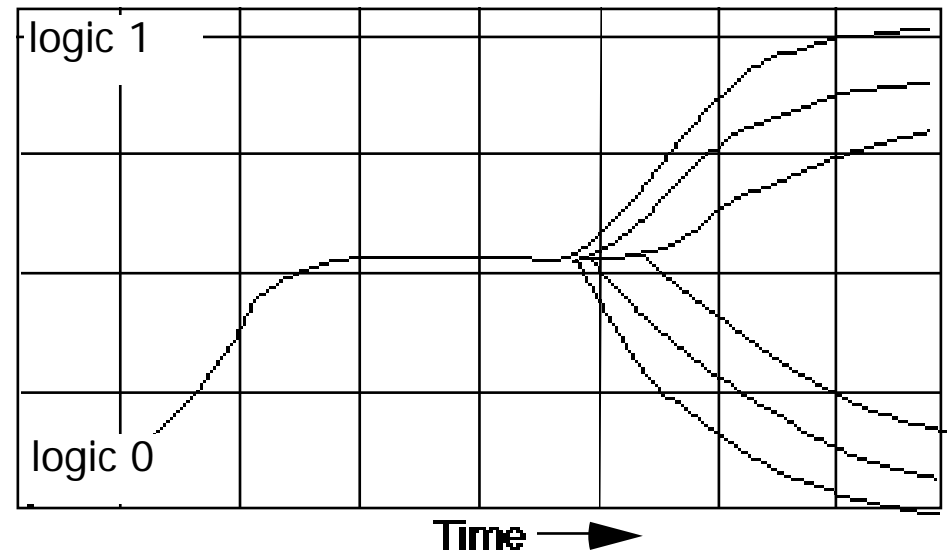


Metastability and synchronization failure

- Occurs when FF input changes close to clock edge
 - the FF may enter a metastable state – neither a logic 0 nor 1 –
 - it may stay in this state an indefinite amount of time
 - this is not likely in practice but has some probability



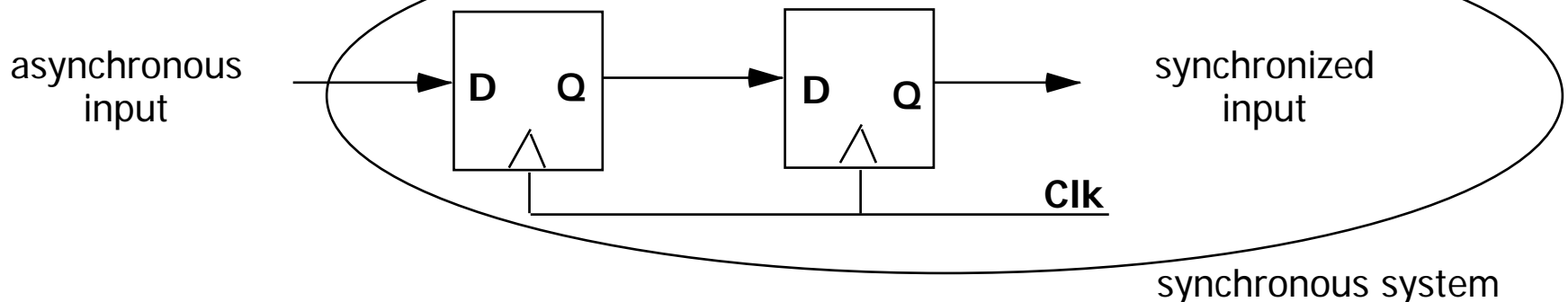
small, but non-zero probability
that the FF output will get stuck
in an in-between state



oscilloscope traces demonstrating
synchronizer failure and eventual
decay to steady state

Dealing with synchronization failure

- Probability of failure can never be reduced to 0, but it can be reduced
 - (1) slow down the system clock
this gives the synchronizer more time to decay into a steady state; synchronizer failure becomes a big problem for very high speed systems
 - (2) cascade two synchronizers
this effectively synchronizes twice (both would have to fail)
 - (3) use fastest possible logic technology in the synchronizer
this makes for a very sharp "peak" upon which to balance

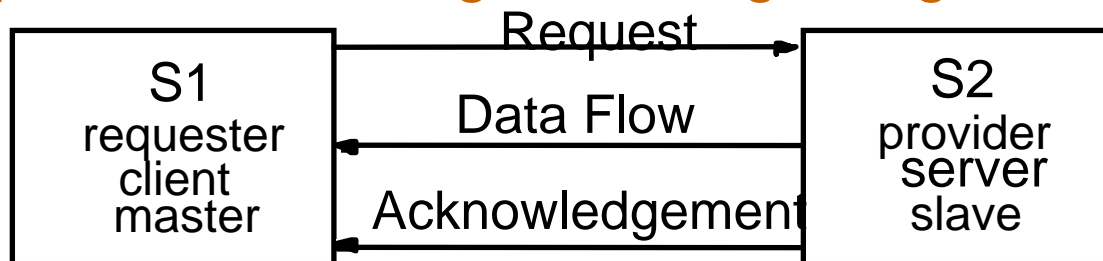


Self-timed and speed independent circuits

- Limits of Synchronous system
 - Fully synchronous not possible for very large systems because of problems of clock skew
 - Partition system into components that are locally clocked
 - These communicate using "speed independent" protocols



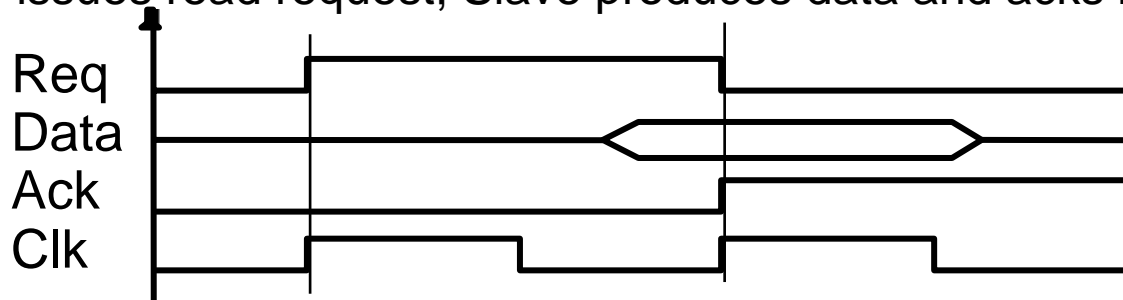
Request/Acknowledgement Signaling



Self-timed and speed independent circuits (cont'd)

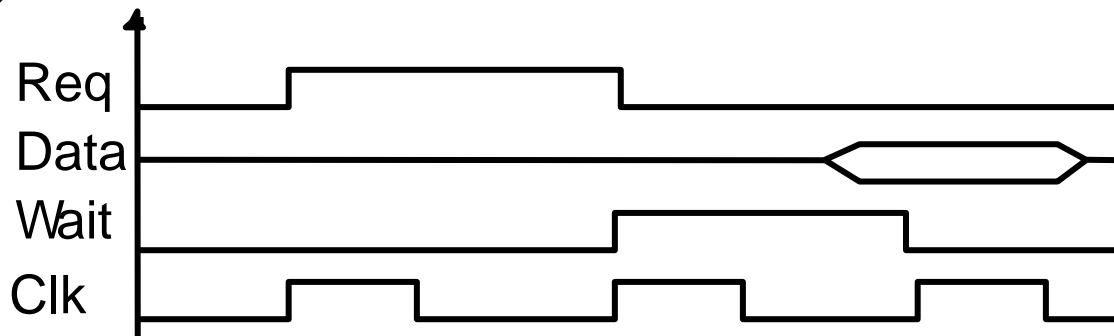
■ Synchronous Signaling

Master issues read request; Slave produces data and acks back



Alternative Synchronous Scheme:

Slave issues **WAIT** signal if it cannot satisfy request in one clock cycle

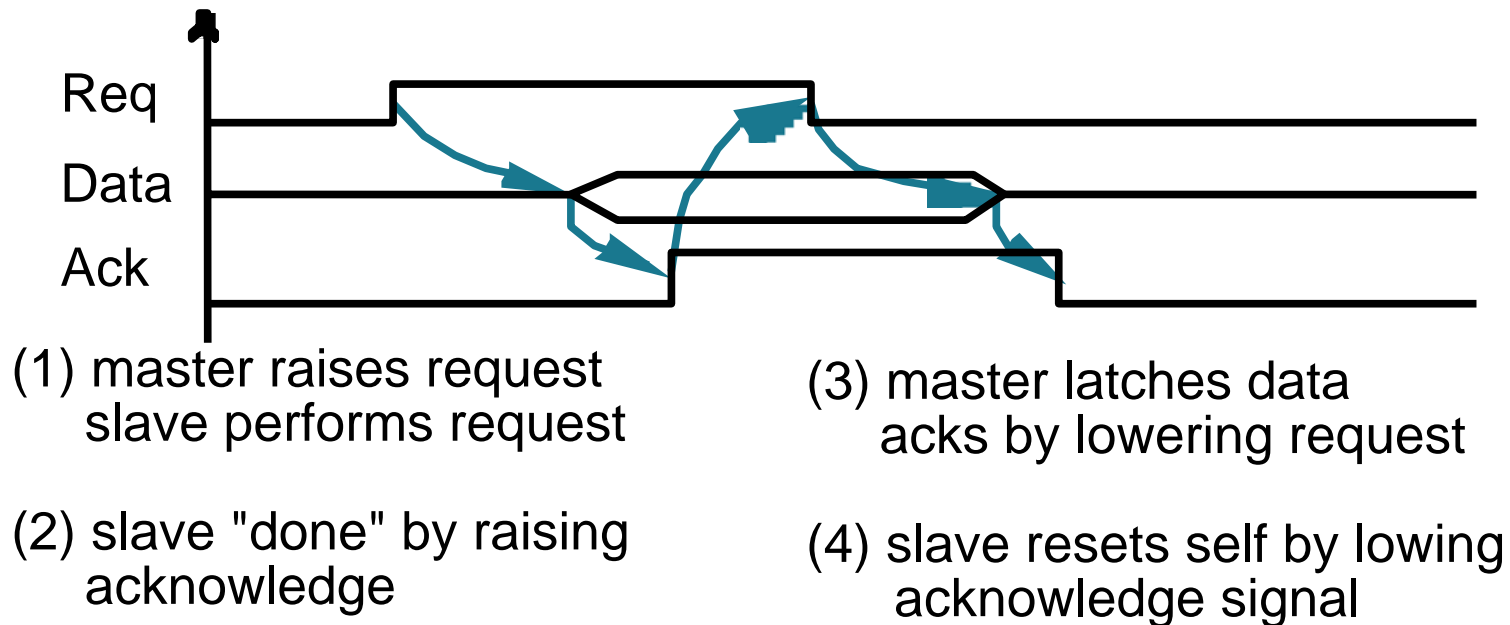


Self-timed and speed independent circuits (cont'd)

■ Asynchronous/Speed Independent Signaling

Communicate information by signal levels rather than edges!
No clock signal

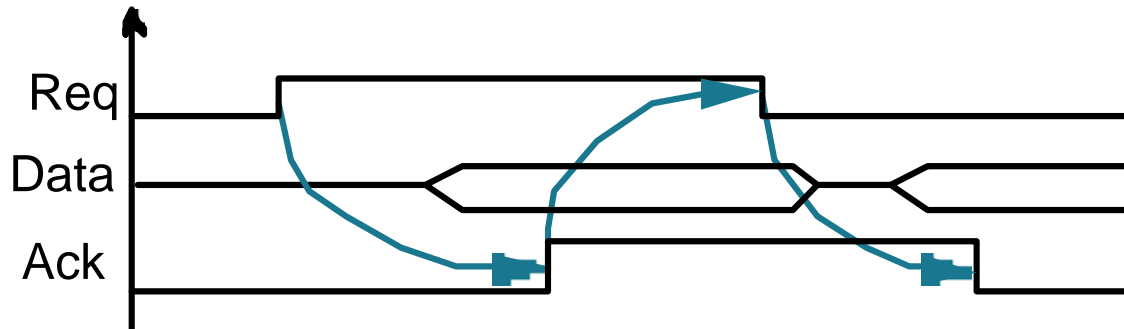
4 Cycle Signaling/Return to Zero Signaling



Self-timed and speed independent circuits (cont'd)

- Alternative: 2 cycle signaling

Non-Return-to-Zero



(1) master raises request, slave services request

(2) slave indicates that it is done by raising acknowledge

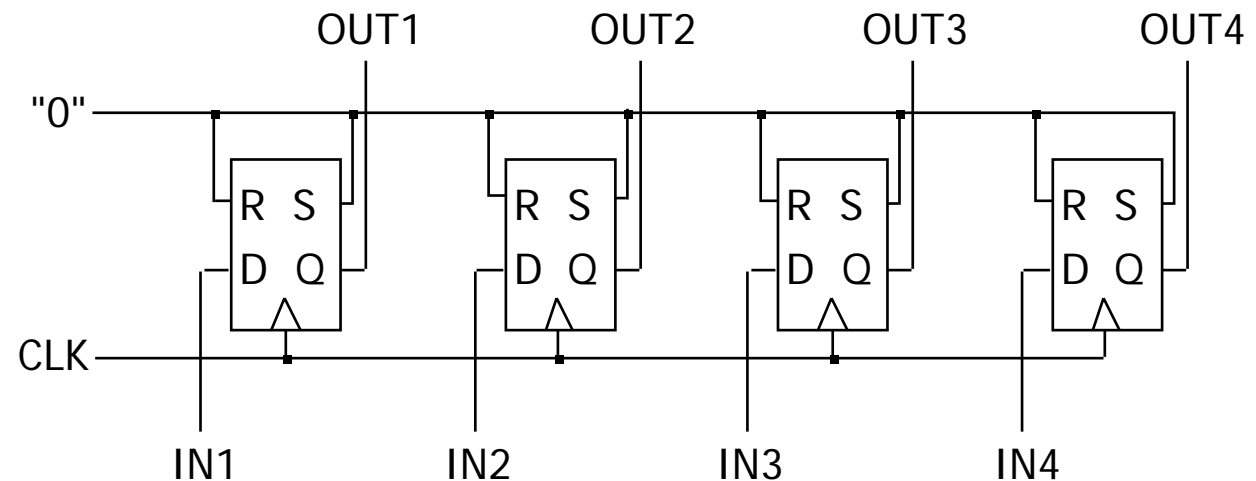
Next request indicated by low level of request

Requires additional state in master and slave
to remember previous setting or request/acknowledge

4 Cycle Signaling is more foolproof

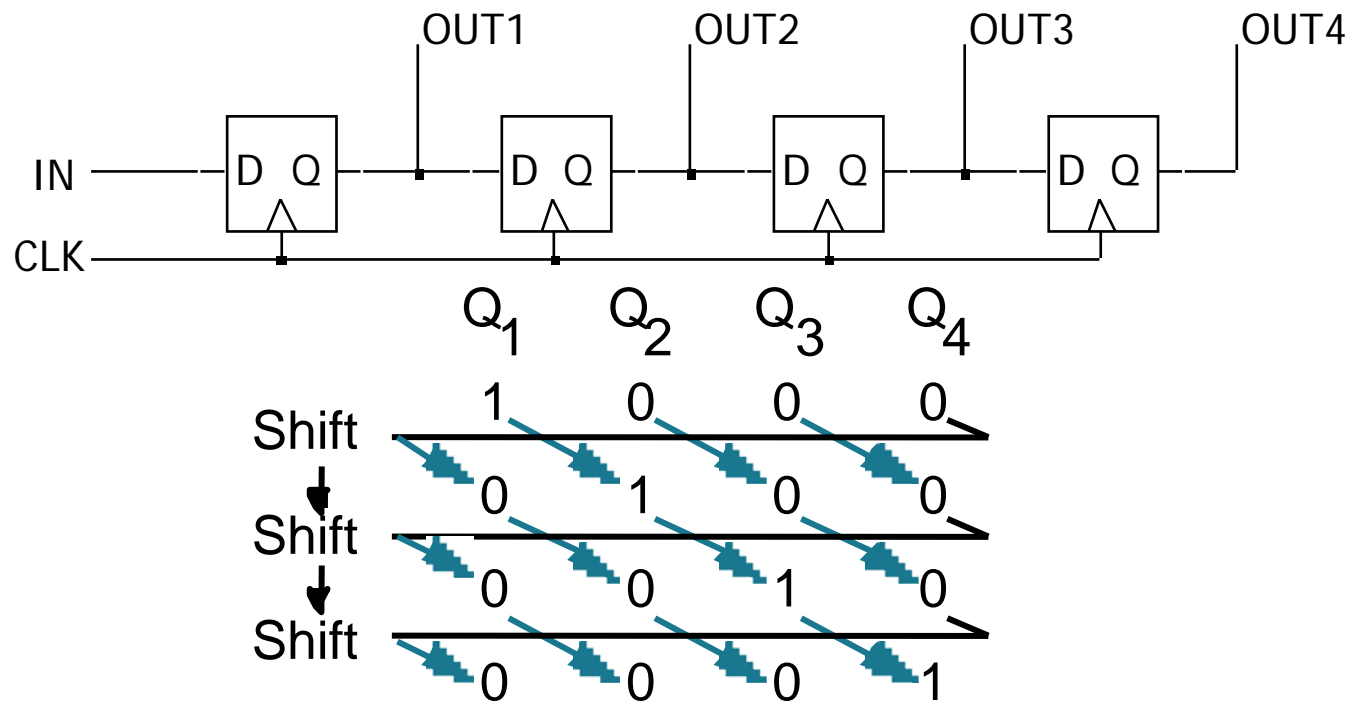
Registers

- Collections of flip-flops with similar controls and logic
 - stored values somehow related (for example, form binary value)
 - share clock, reset, and set lines
 - similar logic at each stage
- Examples
 - shift registers
 - counters



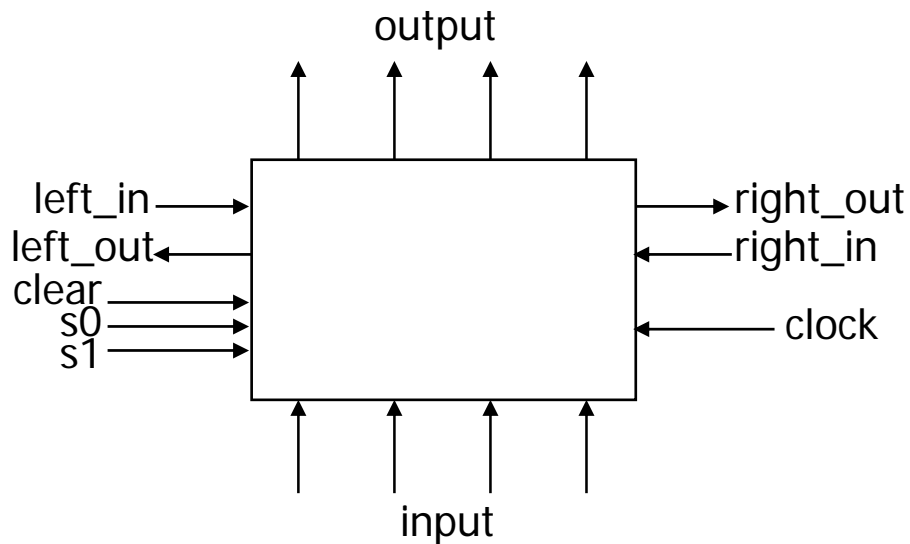
Shift register

- Holds samples of input
 - store last 4 input values in sequence
 - 4-bit shift register:



Universal shift register

- Holds 4 values
 - serial or parallel inputs
 - serial or parallel outputs
 - permits shift left or right
 - shift in new values from left or right



clear sets the register contents and output to 0

s1 and s0 determine the shift function

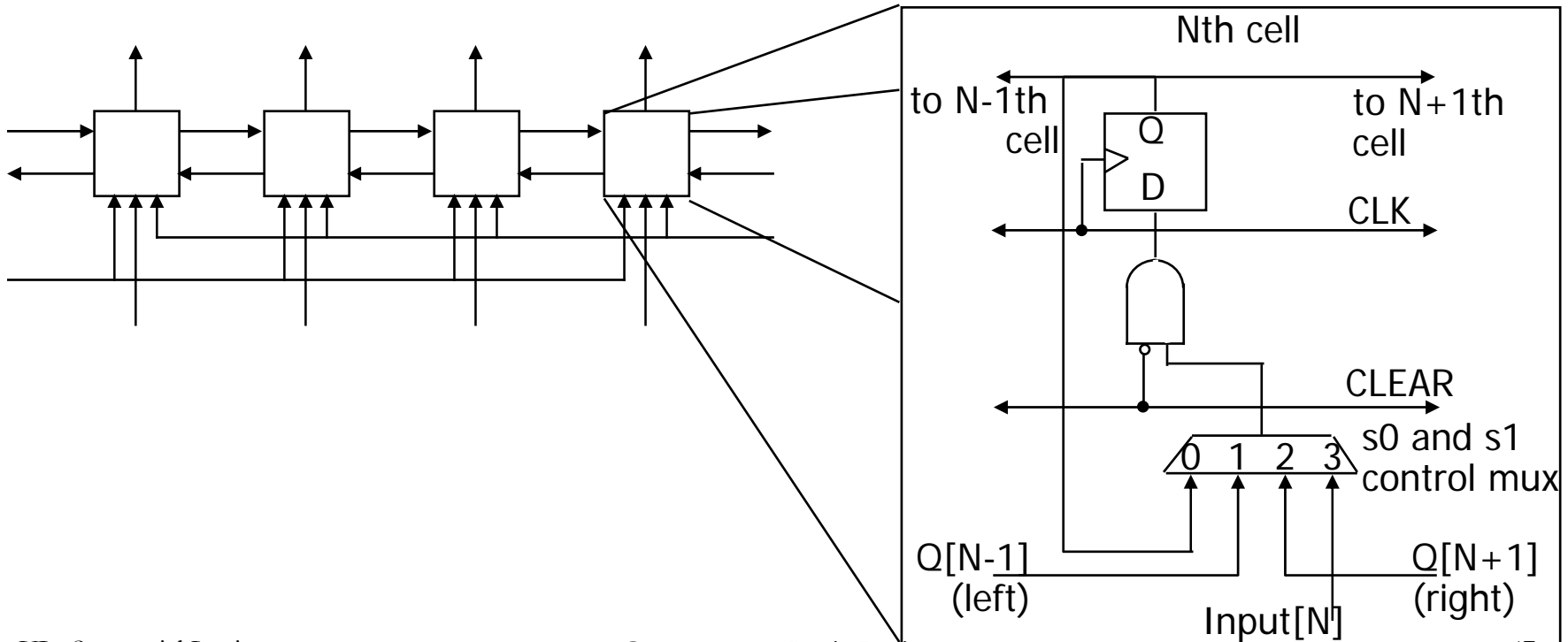
s0	s1	function
0	0	hold state
0	1	shift right
1	0	shift left
1	1	load new input

Design of universal shift register

- Consider one of the four flip-flops

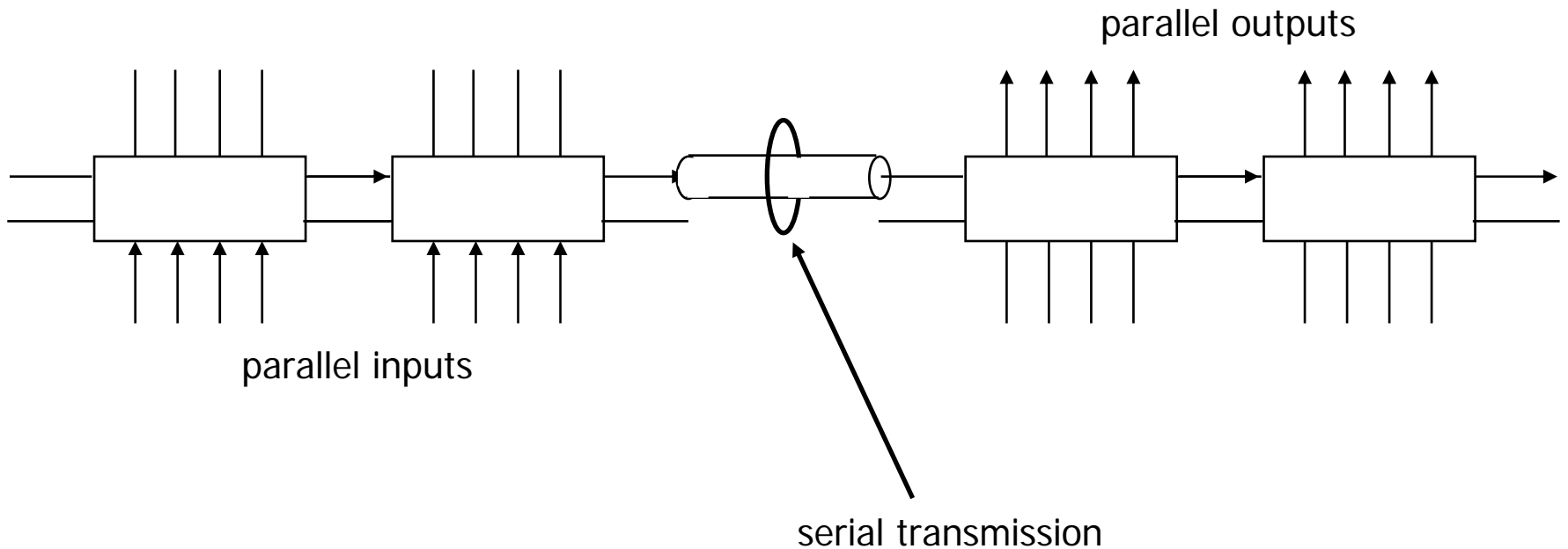
- new value at next clock cycle:

clear	s0	s1	new value
1	-	-	0
0	0	0	output
0	0	1	output value of FF to left (shift right)
0	1	0	output value of FF to right (shift left)
0	1	1	input

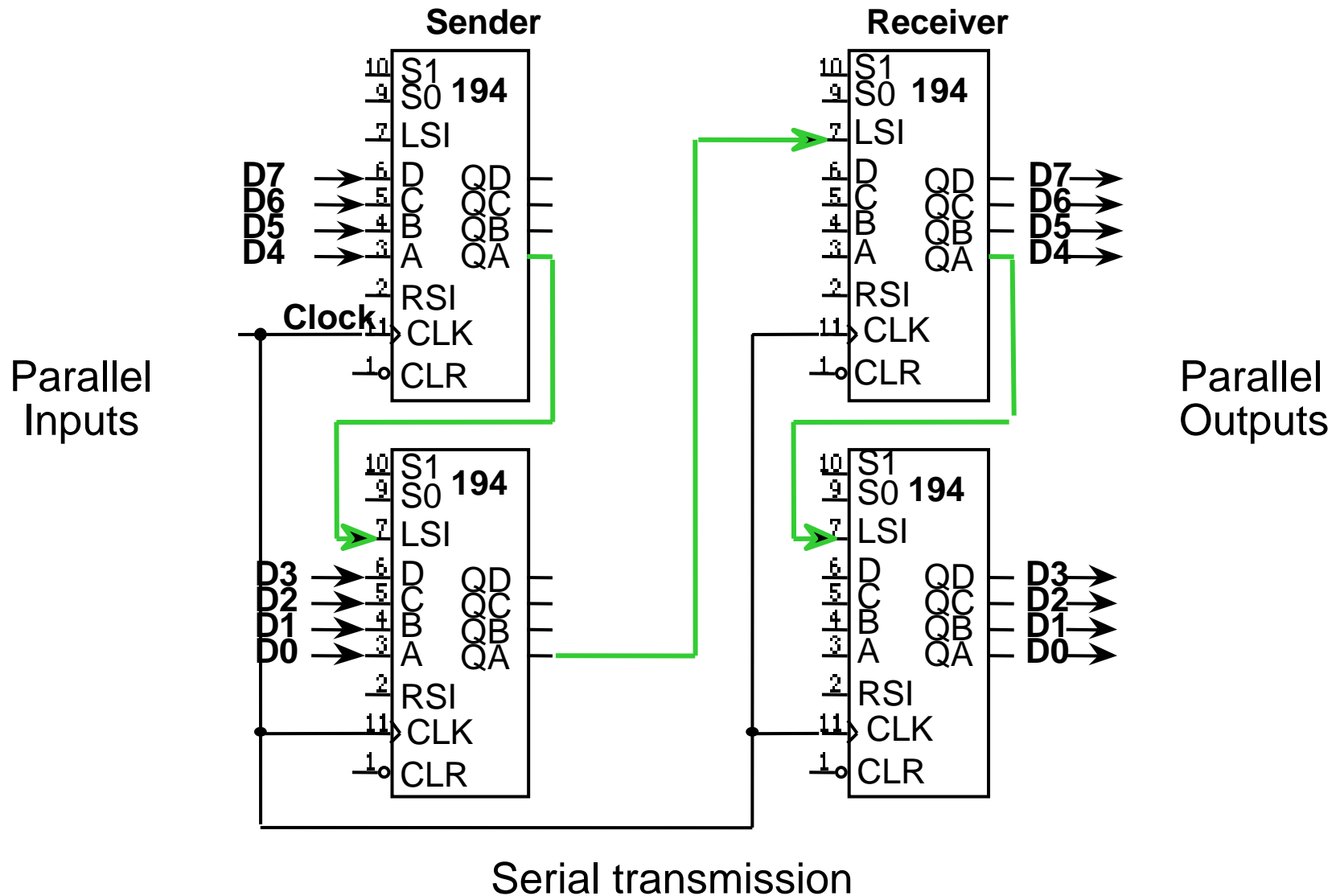


Shift register application

- Parallel-to-serial conversion for serial transmission



Shift register application (cont'd)



Hardware Description Languages and Sequential Logic

- Flip-flops
 - representation of clocks - timing of state changes
 - asynchronous vs. synchronous
- Shift registers
- Simple counters

Flip-flop in Verilog

- Use always block's sensitivity list to wait for clock edge

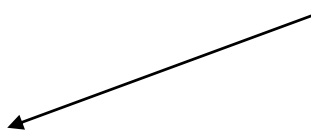
```
module dff (clk, d, q);  
  
    input  clk, d;  
    output q;  
    reg    q;  
  
    always @(posedge clk)  
        q = d;  
  
endmodule
```

Incorrect Flip-flop in Verilog

- Use always block's sensitivity list to wait for clock to change

```
module dff (clk, d, q);  
  
    input  clk, d;  
    output q;  
    reg    q;  
  
    always @(clk)  
        q = d;  
  
endmodule
```

Not correct! Q will change whenever the clock changes, not just on an edge.



More Flip-flops

- Synchronous/asynchronous reset/set
 - single thread that waits for the clock
 - three parallel threads – only one of which waits for the clock

Synchronous

```
module dff (clk, s, r, d, q);
    input  clk, s, r, d;
    output q;
    reg    q;

    always @(posedge clk)
        if (r)      q = 1'b0;
        else if (s) q = 1'b1;
        else        q = d;

endmodule
```

Asynchronous

```
module dff (clk, s, r, d, q);
    input  clk, s, r, d;
    output q;
    reg    q;

    always @(posedge clk or posedge r
            or posedge s)
        if (r)      q = 1'b0;
        else if (s) q = 1'b1;
        else        q = d;

endmodule
```

Blocking and Non-Blocking Assignments

- Blocking assignments ($X=A$)
 - completes the assignment before continuing on to next statement
- Non-blocking assignments ($X<=A$)
 - completes in zero time and doesn't change the value of the target until a blocking point (delay/wait) is encountered
- Example: swap

```
always @(posedge CLK)
begin
    temp = B;
    B = A;
    A = temp;
end
```

```
always @(posedge CLK)
begin
    A <= B;
    B <= A;
end
```

Register-transfer-level (RTL) Assignment

- Non-blocking assignment is also known as an RTL assignment
 - if used in an always block triggered by a clock edge
 - all flip-flops change together

```
// B,C,D all get the value of A
always @(posedge clk)
begin
    B = A;
    C = B;
    D = C;
end
```

```
// implements a shift register too
always @(posedge clk)
begin
    B <= A;
    C <= B;
    D <= C;
end
```

Sequential logic summary

- Fundamental building block of circuits with state
 - latch and flip-flop
 - R-S latch, R-S master/slave, D master/slave, edge-triggered D flip-flop
- Timing methodologies
 - use of clocks
 - cascaded FFs work because propagation delays exceed hold times
 - beware of clock skew
- Asynchronous inputs and their dangers
 - synchronizer failure: what it is and how to minimize its impact
- Basic registers
 - shift registers
 - counters
- Hardware description languages and sequential logic