

SEQUENTIAL LOGIC



- Combinational logic:
 - Compute a function all at one time
 - Fast/expensive
 - e.g. combinational multiplier
- Sequential logic:
 - Compute a function in a series of steps
 - Slower/more efficient
 - e.g. shift/add multiplier
- Key to sequential logic: circuit has feedback
 - Use the result of one step as an input to the next

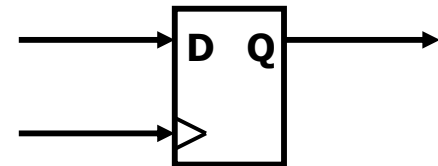
WHAT WE NEED:



- When inputs change...
- Wait until combinational logic has finished and result is stable...
- Then sample the output value and save...
- Feed the saved output back to the input of the combinational logic
 - Make sure the saved output can't change
- Key idea: we sample the result at the right time, i.e. when it is ready
 - Only then do we update the stored value
- *How do we know when to sample?*
- *How do we know when the inputs changed?*
- *How do we know how long to wait?*

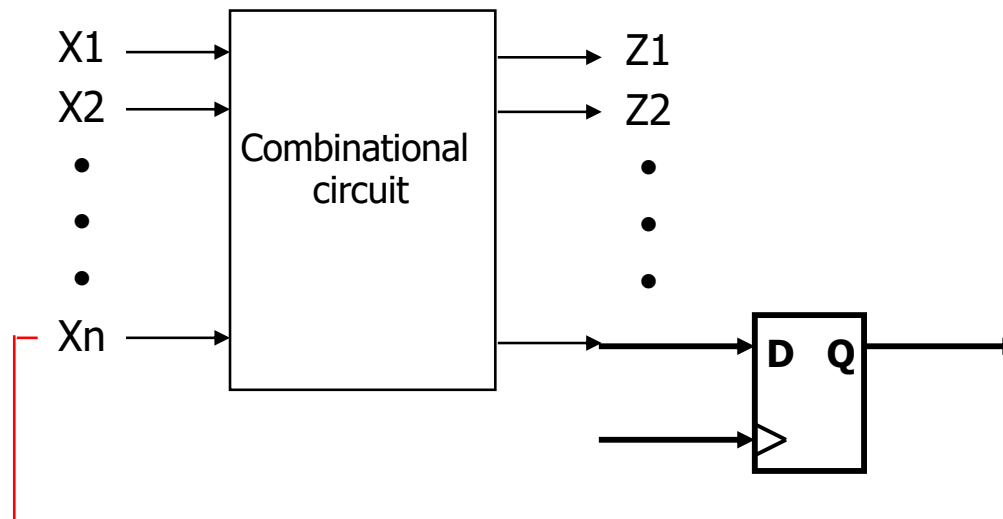
WHAT WE NEED:

- A circuit that can *sample* a value
- A signal that says *when* to sample
- Edge-triggered D flip-flop (register)
 - Samples on positive edge of clock
 - Holds value until next positive edge
 - Most common storage element
- Clock
 - Periodic signal, each rising edge signals D flip-flops to sample
 - All registers sample at the same time



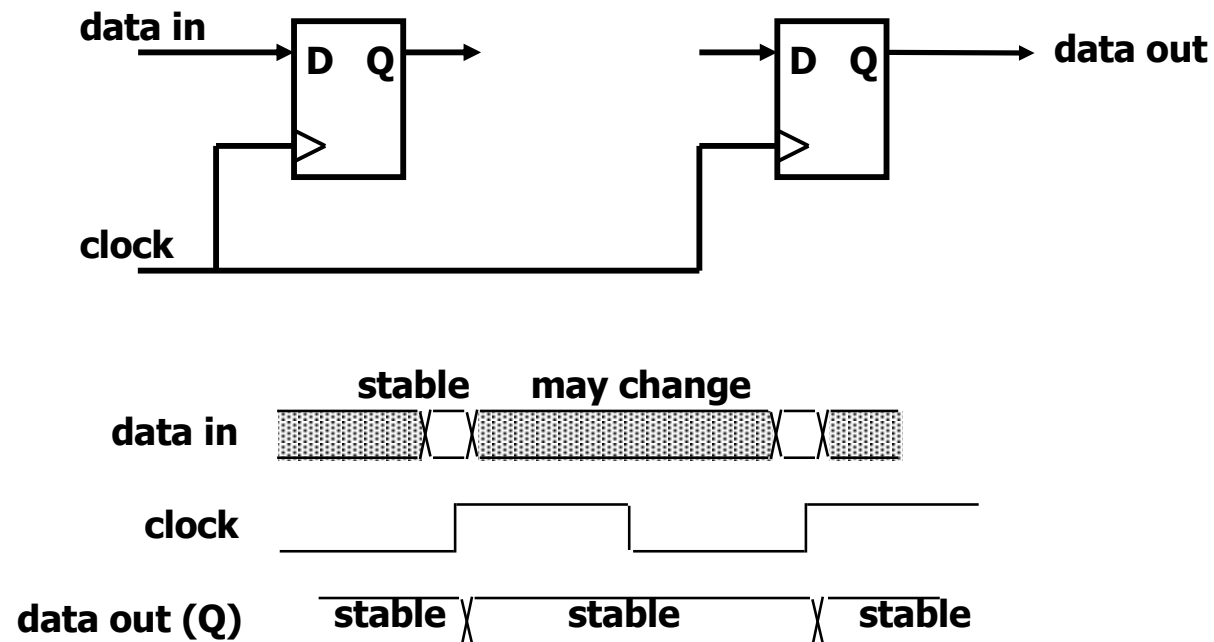
LET'S USE THIS D FLIP-FLOP

- Does this work?
- What do we need to say about the inputs $X_1, X_2, \dots$?
- This is a synchronous sequential circuit



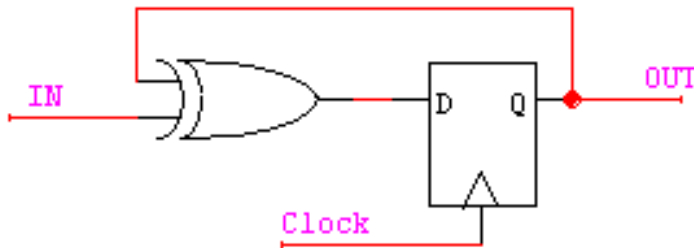
REGISTERS

- Sample data using clock
- Hold data between clock cycles
- Computation (and delay) occurs between registers



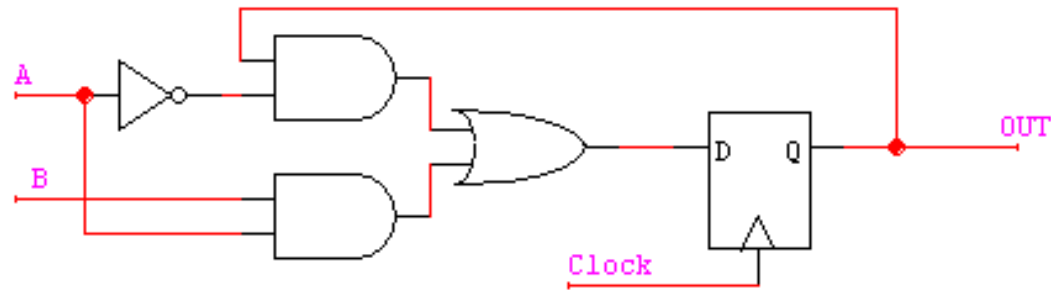
EXAMPLE - CIRCUIT WITH FEEDBACK

- Output is a function of arbitrarily many past inputs



EXAMPLES (CONT'D)

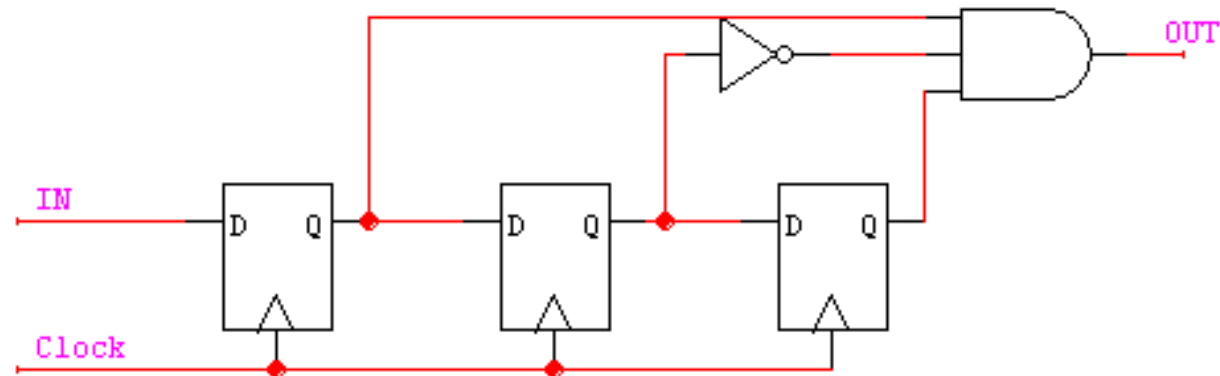
- Output is a function of inputs and the previous state of the circuit



B	A	out ^t	out ^{t+1}
—	0	0	0
—	0	1	1
0	1	—	0
1	1	—	1

EXAMPLE - CIRCUIT WITHOUT FEEDBACK

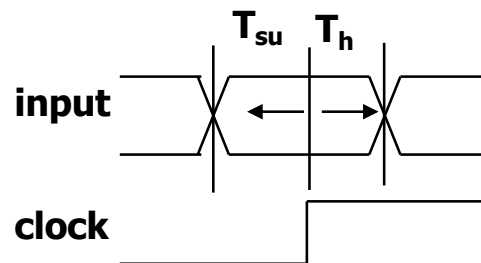
- Output is a function of the input sampled at three different points in time



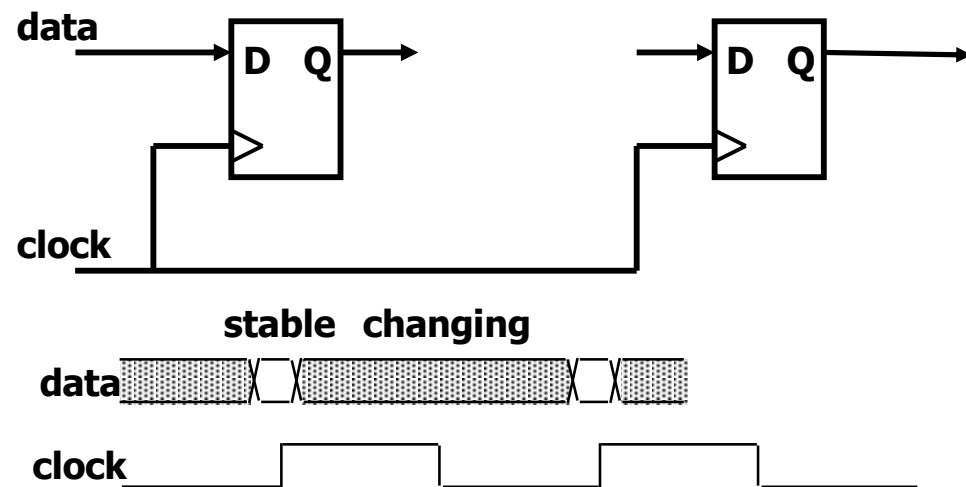
TIMING METHODOLOGIES (CONT'D)

■ Definition of terms

- 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)

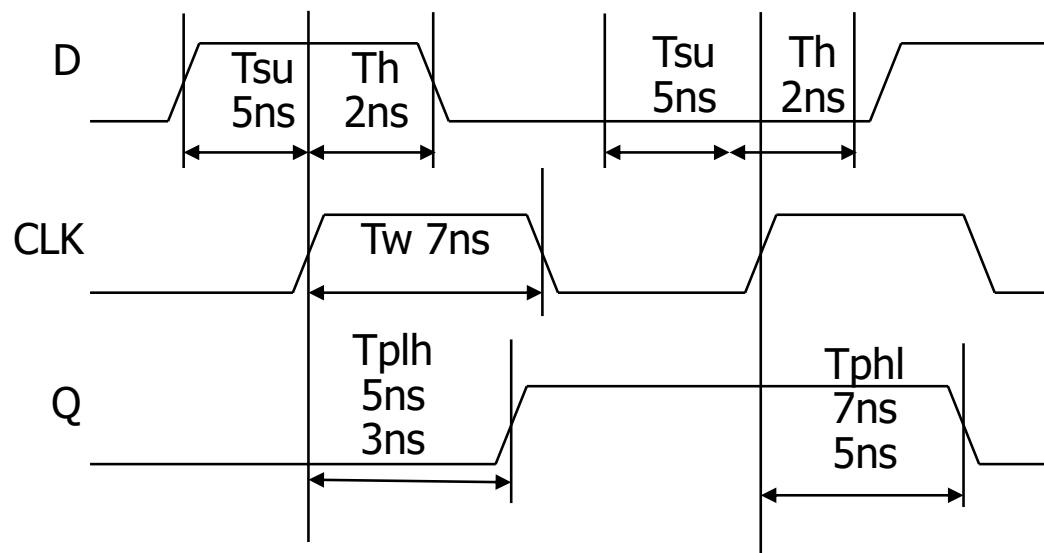


there is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognized



TYPICAL TIMING SPECIFICATIONS

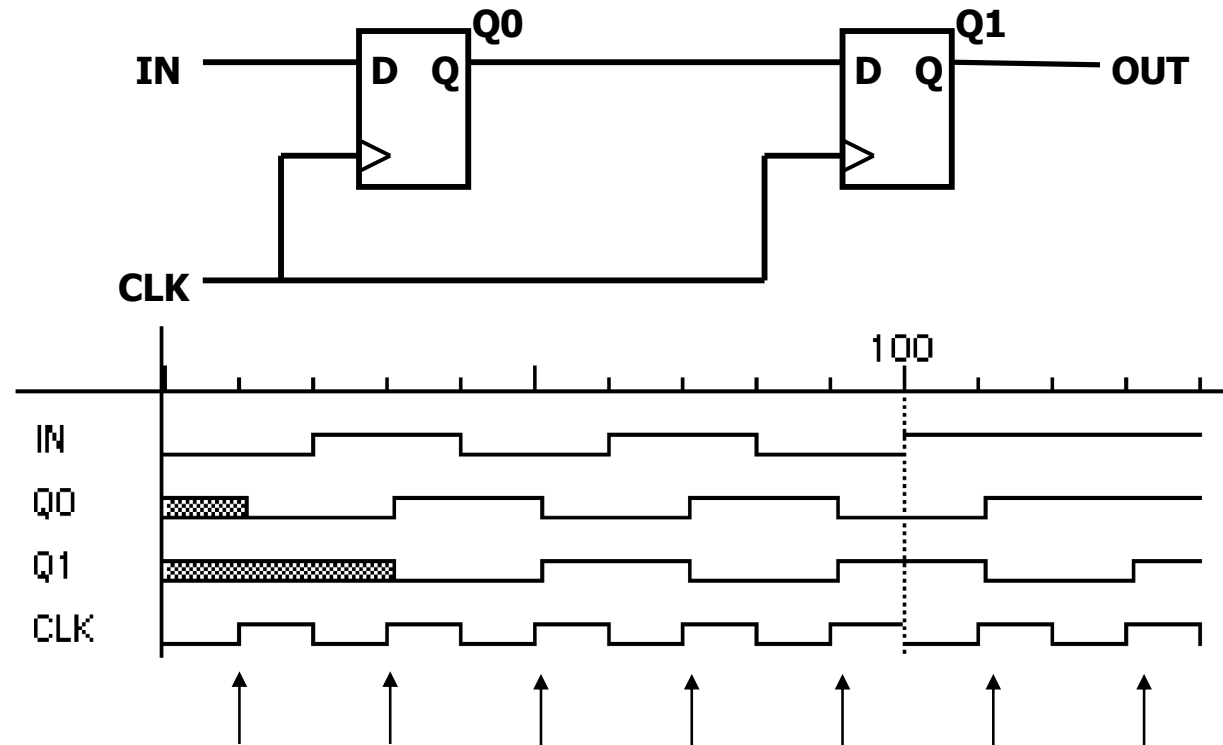
- Positive edge-triggered D flip-flop
 - setup and hold times
 - minimum clock width
 - propagation delays (low to high, high to low, max and typical)



all measurements are made from the clocking event that is,
the rising edge of the clock

CASCADED FLIP-FLOPS

- Shift register: new value to first stage while second state obtains current value of first stage
- Outputs change only on clock tick

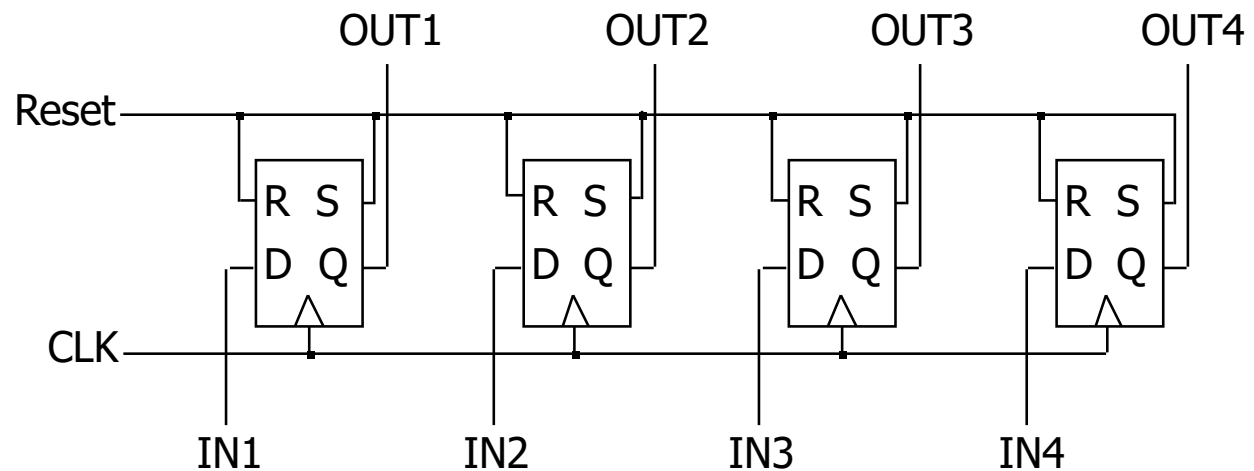


FLIP-FLOP EXTRAS

- Reset (set state to 0) - R
 - synchronous: $D_{\text{new}} = R' \cdot D_{\text{old}}$ (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: $D_{\text{new}} = D_{\text{old}} + S$ (when next clock edge arrives)
 - asynchronous: doesn't wait for clock, quick but dangerous
- Both reset and preset
 - $D_{\text{new}} = R' \cdot D_{\text{old}} + S$ (set-dominant)
 - $D_{\text{new}} = R' \cdot D_{\text{old}} + R'S$ (reset-dominant)
- Selective input capability (input enable or load) - LD or EN
 - multiplexor at input: $D_{\text{new}} = LD' \cdot Q + LD \cdot D_{\text{old}}$
 - load may or may not override reset/set
- Complementary outputs - Q and Q'
- Output enable - tristate output

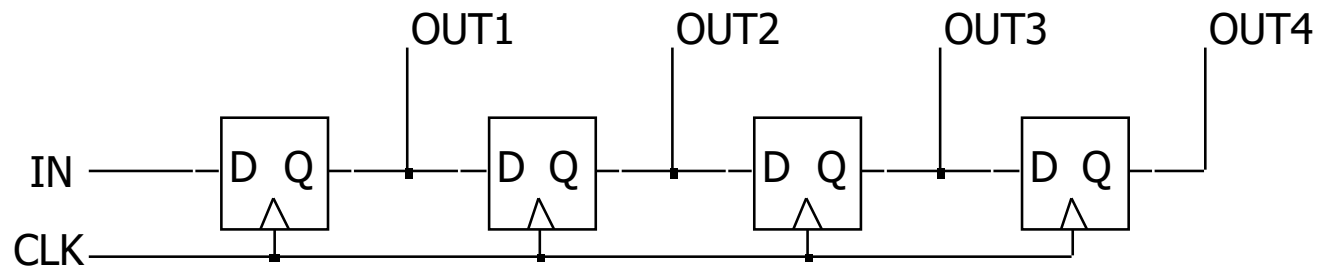
REGISTERS

- Collection of flip-flops with same control
 - stored values somehow related (for example, form binary value)
- Examples
 - shift registers
 - counters



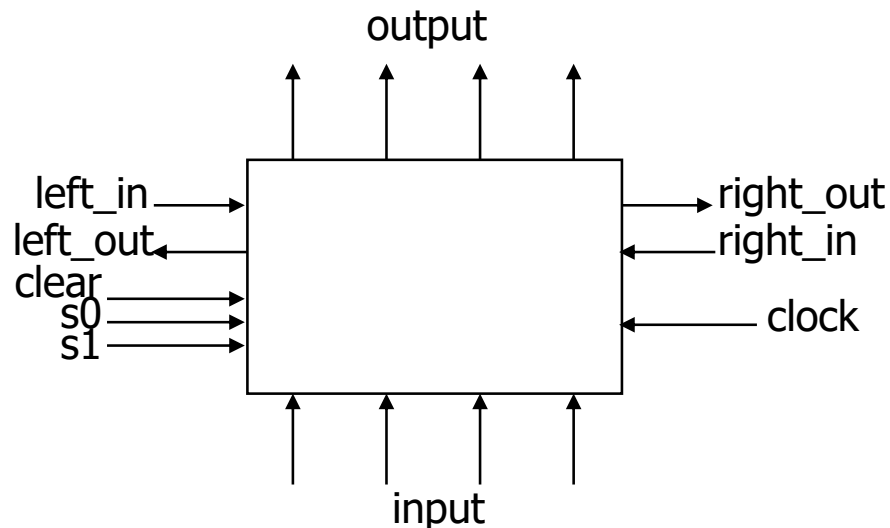
SHIFT REGISTER

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



4-BIT 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

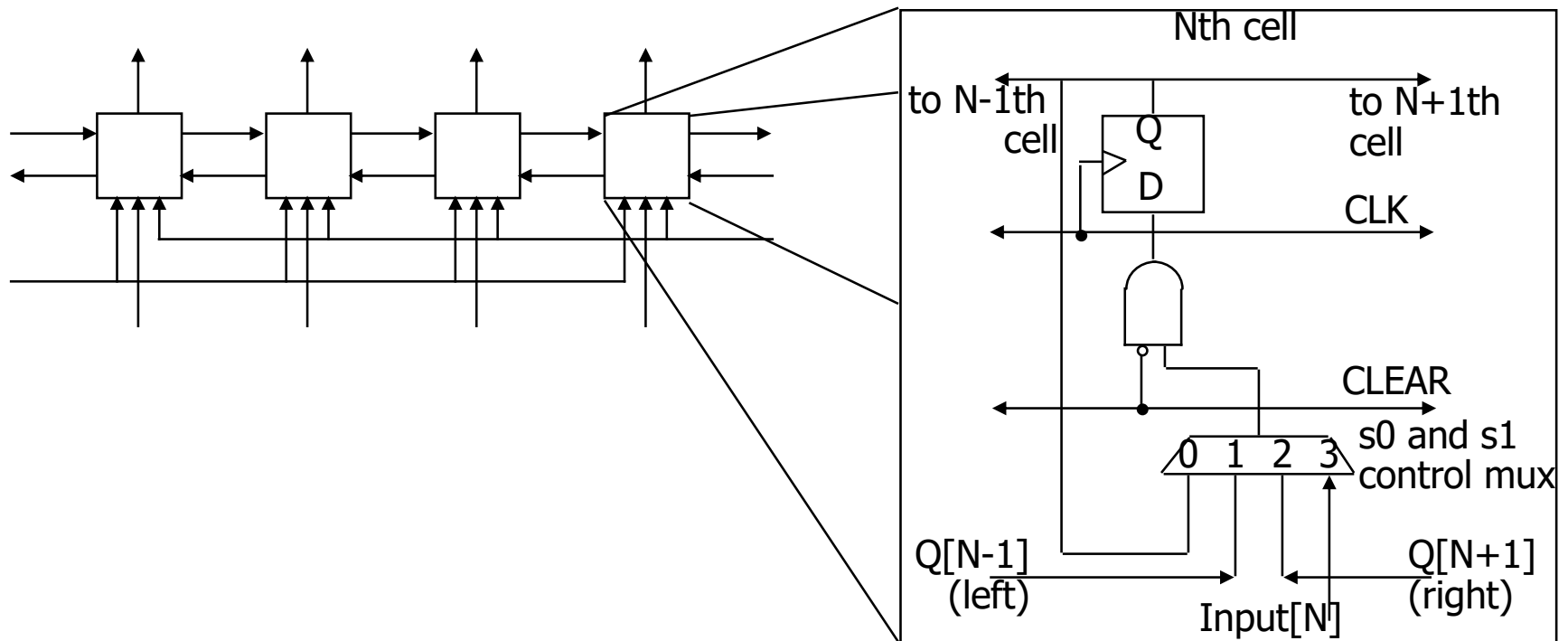
s0	s1	function
0	0	hold state
0	1	shift right
1	0	shift left
1	1	load new input

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

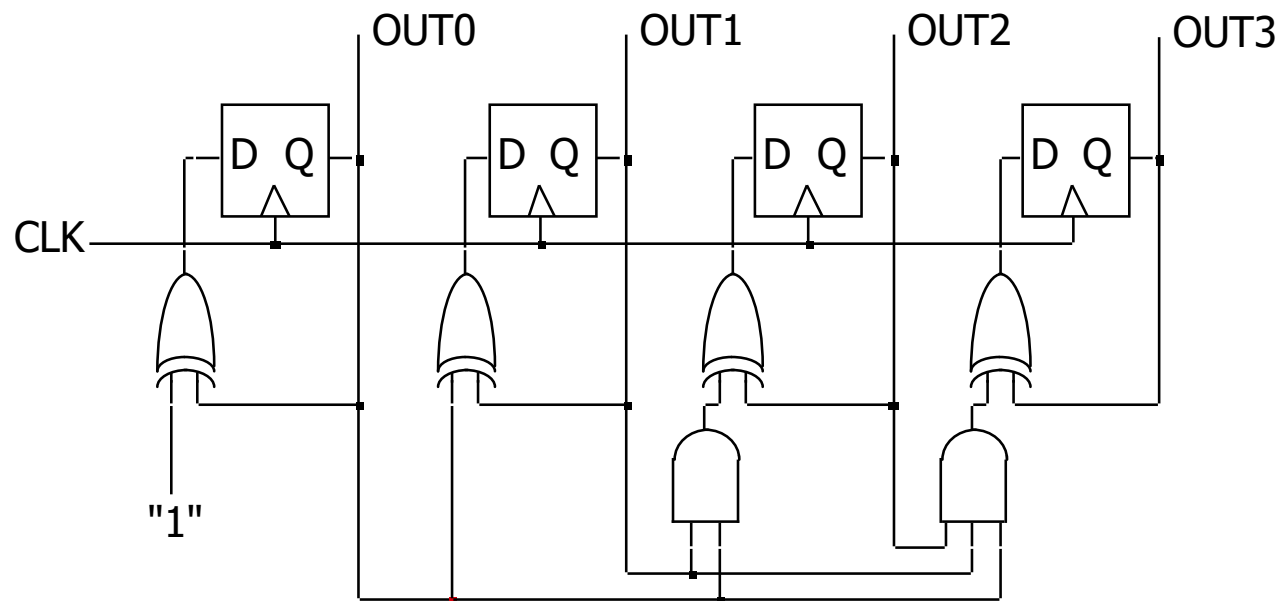


Sequential Logic

16

BINARY COUNTER

- Next state function for bit i
 - XOR acts like a "programmable" inverter
 - if bits $0:i-1$ are 1, then toggle bit i
 - requires an i -input AND for bit i
- Synchronous: outputs all change when clock ticks

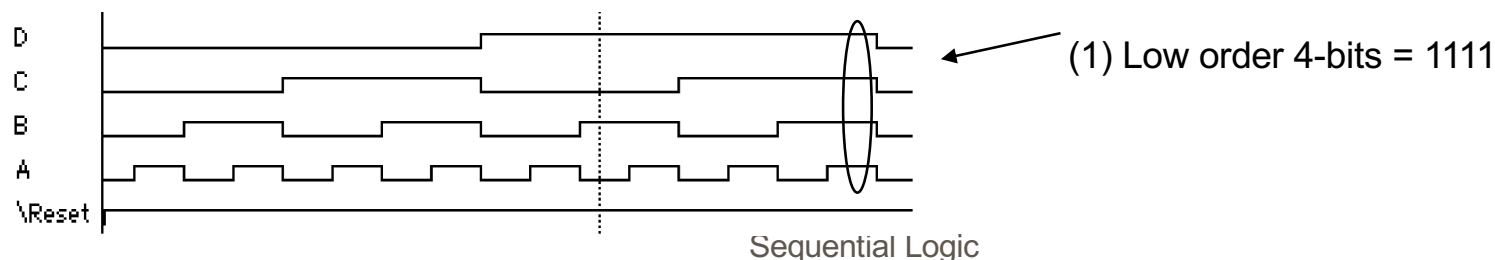
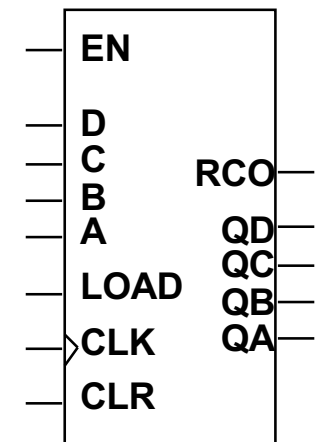
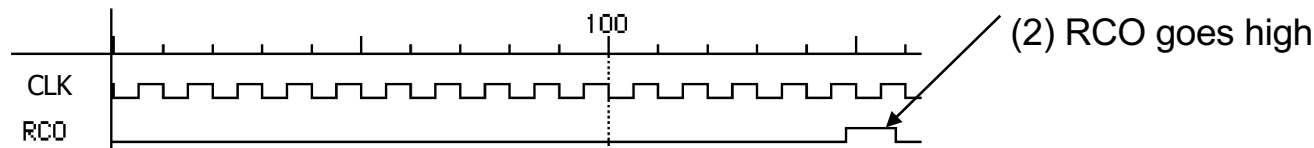


Sequential Logic

17

EXAMPLE: 4-BIT BINARY SYNCHRONOUS COUNTER

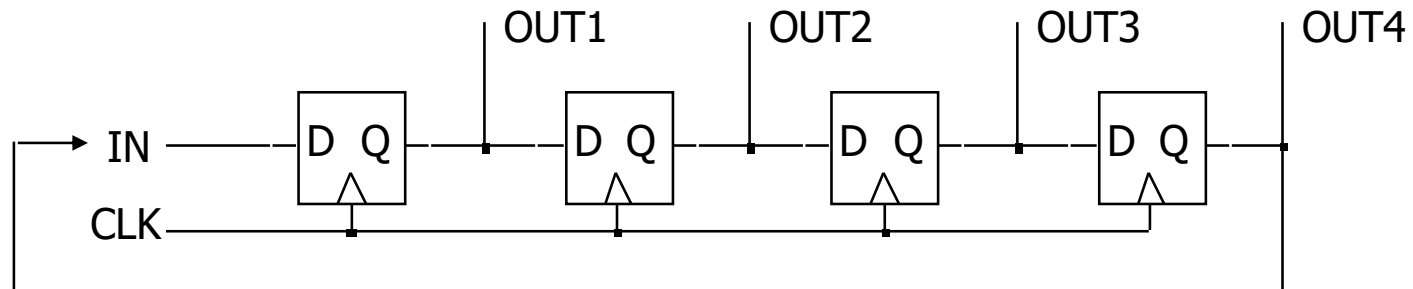
- Typical library component
 - positive edge-triggered FFs w/ synchronous load and clear inputs
 - parallel load data from D, C, B, A
 - enable input: assert to enable counting
 - RCO: "ripple-carry out" used for cascading counters
 - high when counter is in its highest state 1111
 - implemented using an AND gate



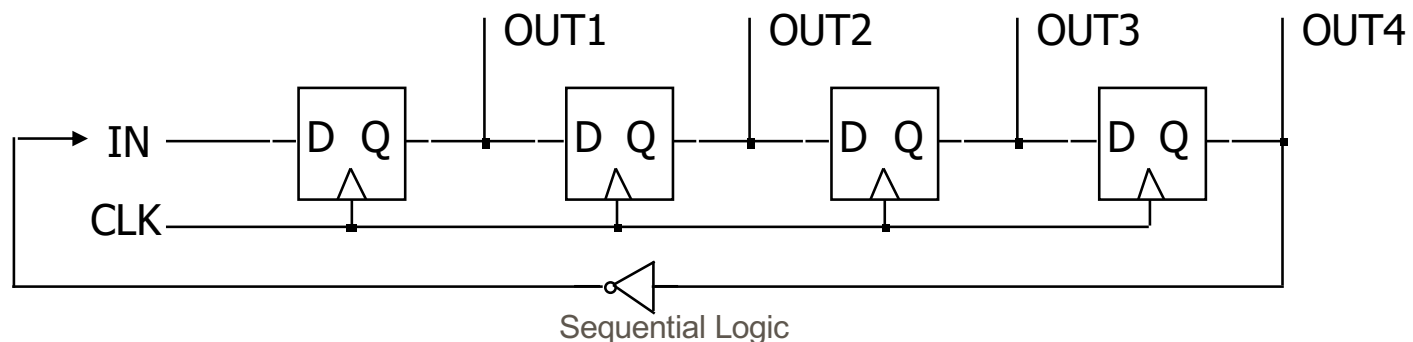
Sequential Logic

OTHER COUNTERS: CHEAPER/FASTER

- Sequences through a fixed set of patterns
 - in this case, 1000, 0100, 0010, 0001
 - if one of the patterns is its initial state (by loading or set/reset)

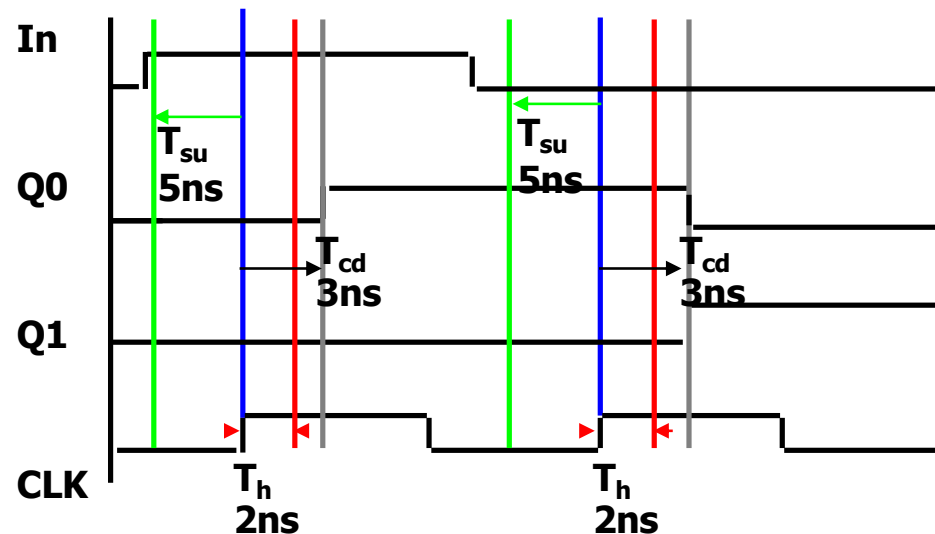


- Mobius (or Johnson) counter
 - in this case, 1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000



CASCADED FLIP-FLOPS (CONT'D)

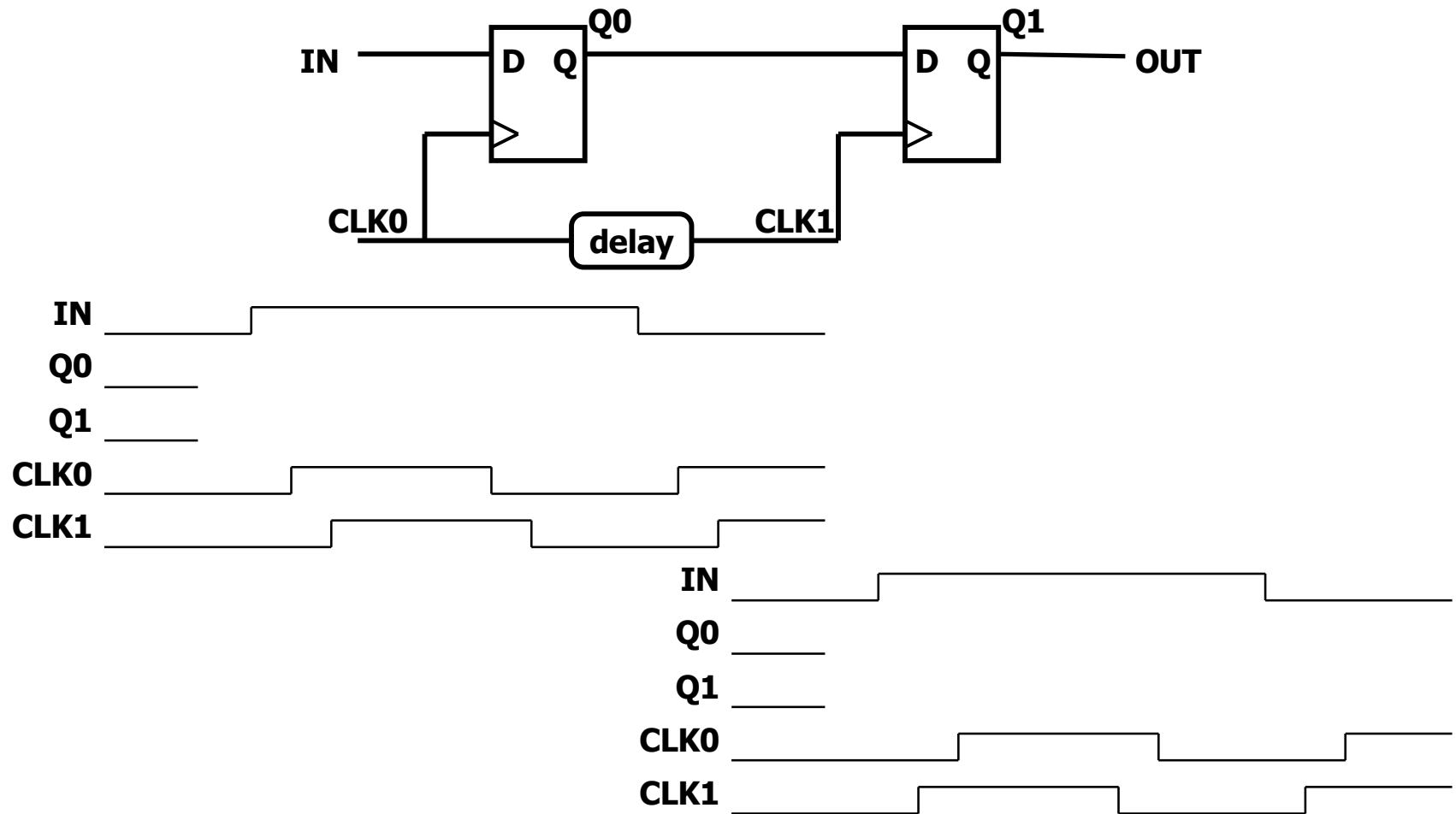
- Contamination delay > hold time
 - next stage latches current value before it is replaced by new value
- Clock period > propagation delays + setup time
 - new value must arrive early enough to be seen at next clock event
- Timing constraints guarantee proper operation of cascaded components
- Assumes infinitely fast distribution of clock signal



Sequential Logic

20

EFFECT OF CLOCK SKEW



CLOCK SKEW

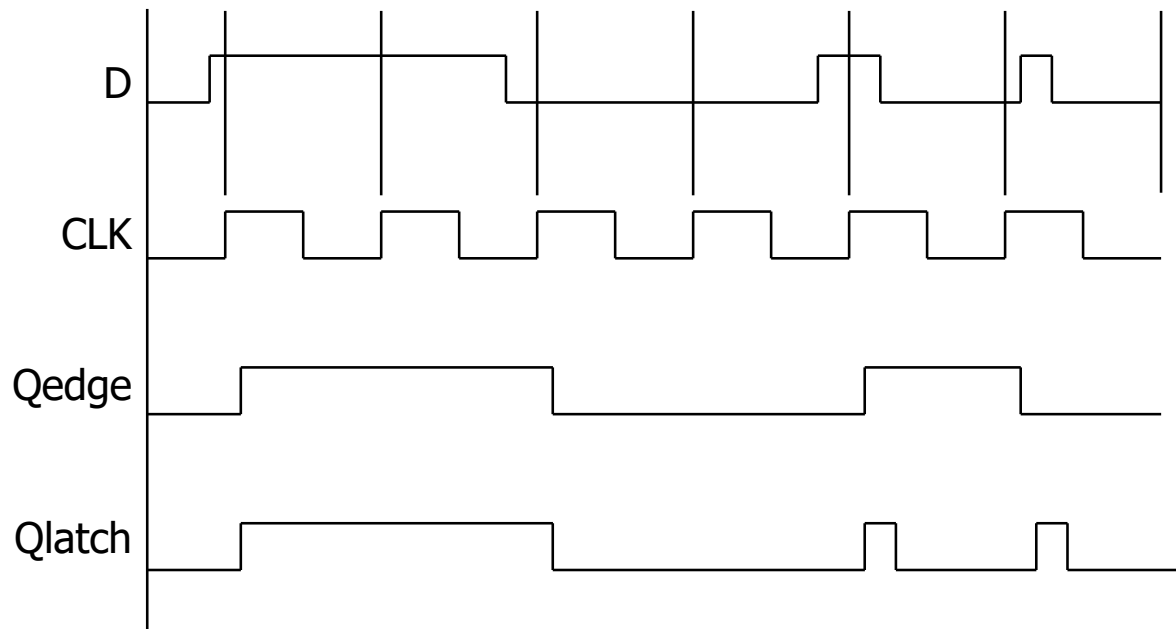
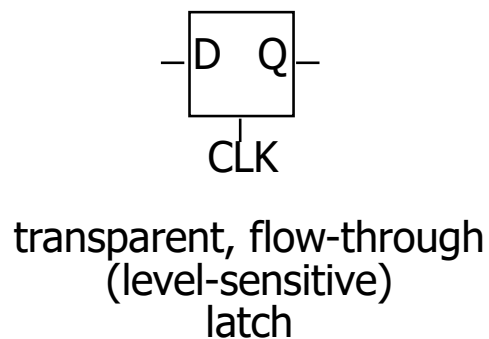
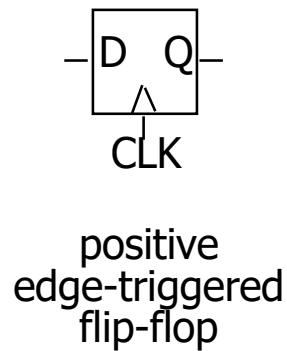


- Correct behavior assumes that all storage elements sample at exactly the same time
- Not possible in real systems:
 - clock driven from some central location
 - different wire delay to different points in the circuit
- Problems arise if skew is of the same order as FF contamination delay
- Gets worse as systems get faster (wires don't improve as fast)
 - 1) distribute clock signals in general direction of data flow
 - 2) wire carrying the clock between two communicating components should be as short as possible
 - 3) try to make all wires from the clock generator be the same length
 - clock tree

OTHER TYPES OF LATCHES AND FLIP-FLOPS

- Best choice is D-FF
 - simplest design technique, minimizes number of wires
 - preferred in PLDs and FPGAs
 - good choice for data storage register
 - edge-triggered has most straightforward timing constraints
- Historically J-K FF was popular
 - versatile building block, usually requires least amount of logic to implement function
 - two inputs require more wiring and logic (e.g., two two-level logic blocks in PLDs)
 - good in days of TTL/SSI, not a good choice for PLDs and FPGAs
 - can always be implemented using D-FF
- Level-sensitive latches in special circumstances
 - popular in VLSI because they can be made very small (4 transistors)
 - fundamental building block of all other flip-flop types
 - two latches make a D-FF
- Preset and clear inputs are highly desirable

COMPARISON OF LATCHES AND FLIP-FLOPS



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

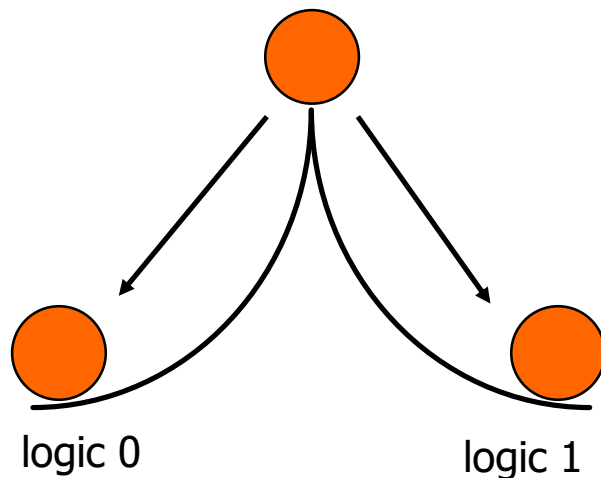
WHAT ABOUT EXTERNAL INPUTS?



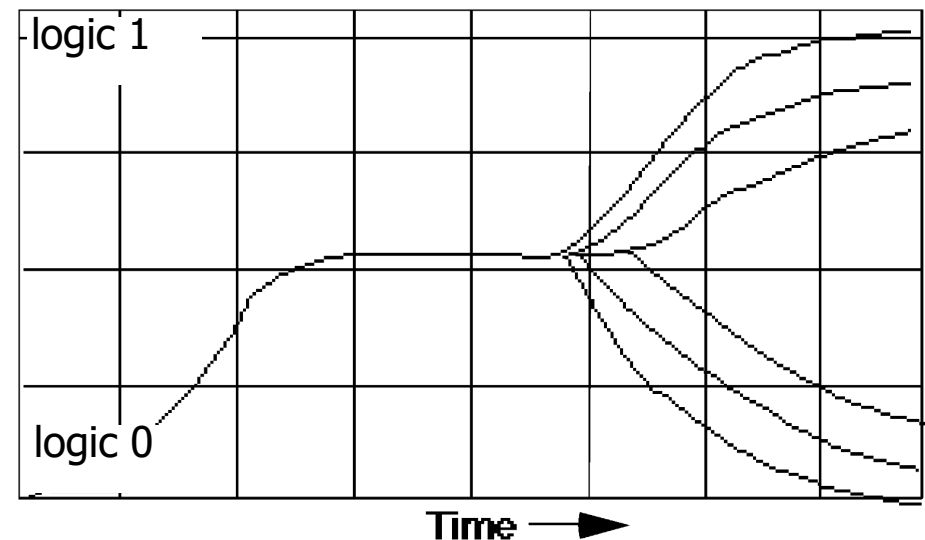
- Internal signals are OK
 - Can only change when clock changes
- External signals can change at any time
 - Asynchronous inputs
 - Truly asynchronous
 - Produced by a different clock
- This means register may sample a signal that is changing
 - Violates setup/hold time
 - What happens?

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

CALCULATING PROBABILITY OF FAILURE

- For a single synchronizer

$$\text{Mean-Time Between Failure (MTBF)} = \exp (t_r / \tau) / (T_0 \times f \times a)$$

where a failure occurs if metastability persists beyond time t_r after a clock edge

- t_r is the resolution time - extra time in clock period for settling
 - $T_{clk} - (t_{pd} + T_{CL} + t_{setup})$
- f is the frequency of the FF clock
- a is the number of asynchronous input changes per second applied to the FF
- T_0 and τ are constants that depend on the FF's electrical characteristics (e.g., gain or steepness of curve)
 - typical values are $T_0 = .4s$ and $\tau = 1.5ns$ (sensitive to temperature, voltage, cosmic rays, etc.).
- Must add probabilities from all synchronizers in system
 $1/MTBF_{system} = \Sigma 1/MTBF_{synch}$

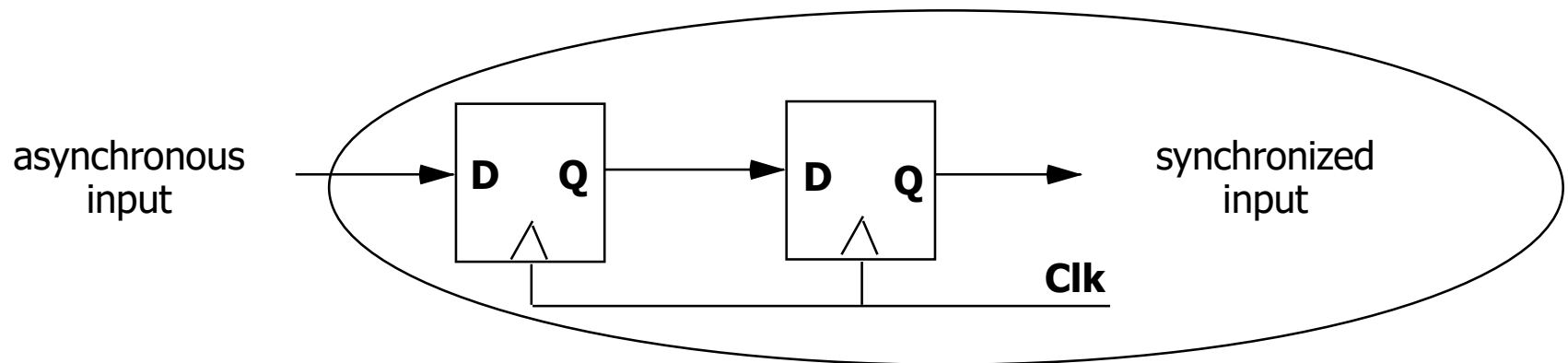
these are just averages

METASTABILITY

- Example
 - input changes at 1 MHz
 - system clock of 10MHz, flipflop ($t_{pd} + t_{setup}$) = 5ns
$$MTBF = \exp(95ns / 1.5ns) / (.4s \times 10^7 \times 10^6) = 25 \text{ million years}$$
 - if we go to 20MHz then:
$$MTBF = \exp(45ns / 1.5ns) / (.4s \times 2 \times 10^7 \times 10^6) = 1.33 \text{ seconds!}$$
- Must do the calculations and allow enough time for synchronization

GUARDING AGAINST SYNCHRONIZATION FAILURE

- Give the register time to decide
 - Probability of failure cannot be reduced to 0, but it can be reduced
- *Slow down the system clock?*
- *Use very fast technology for synchronizer -> quicker decision?*
- *Cascade two synchronizers?*



ANOTHER PROBLEM WITH ASYNCHRONOUS INPUTS

- *What goes wrong here? (Hint: it's not a metastability thing)*
- *What is the fix?*

