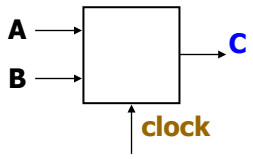
Sequential Logic

- 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

Sequential versus combinational



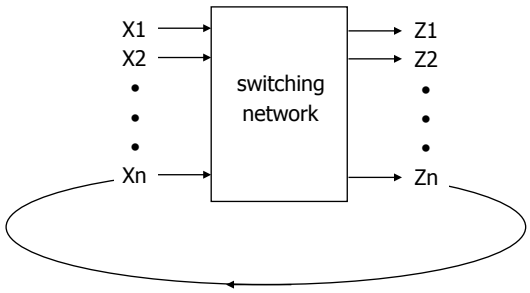
Apply fixed inputs A, B
Wait for clock edge and then observe C
Wait for another clock edge and observe C again

Combinational: C will always be the same
Sequential: C may change

Why a clock signal? What is a clock signal?

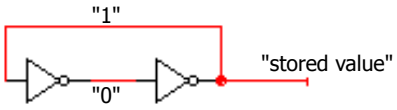
Sequential logic: circuits with feedback

- Why feedback?
  - How else do you remember something?
  - But what stops values from cycling around endlessly?

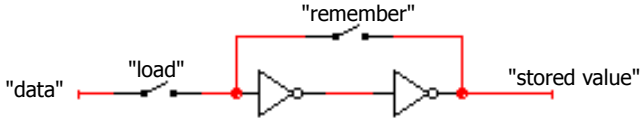


Simplest circuits with feedback

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

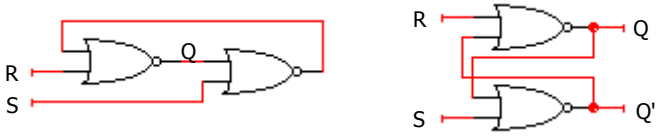


- How to get a new value into the memory cell?
  - selectively break feedback path
  - load new value into cell

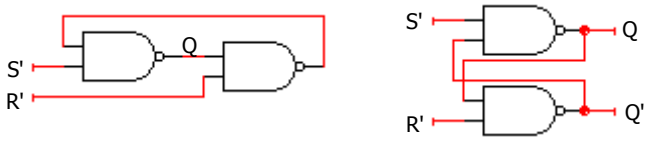


Memory with cross-coupled gates

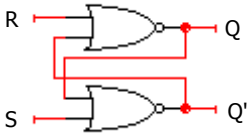
- Set-Reset register – no clock
- Cross-coupled NOR gates
  - similar to inverter pair, with capability to force output to 0 (reset=1) or 1 (set=1)

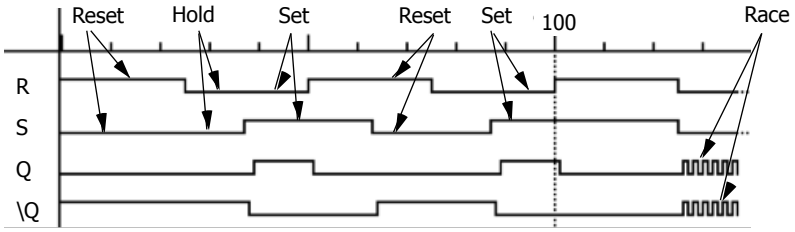


- Cross-coupled NAND gates
  - similar to inverter pair, with capability to force output to 0 (reset=0) or 1 (set=0)



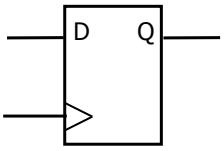
Timing behavior





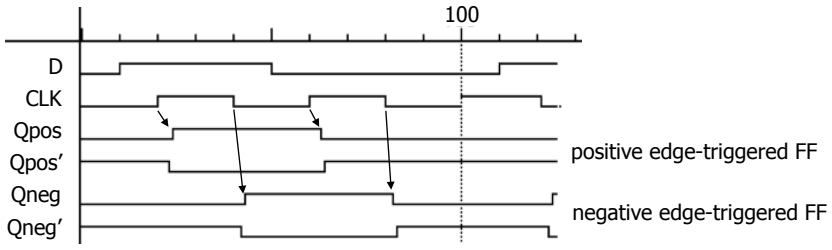
Clocked Edge-Triggered D-FF

- Basic storage element – sample and hold
  - Samples D on rising clock edge
  - Outputs this value on Q until the next sample
- We won't worry about how the D flip-flop is implemented
  - Lots of ways, mostly magic these days



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

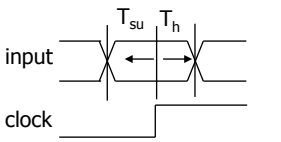


Timing methodologies

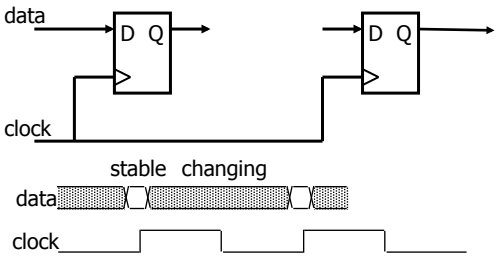
- Rules for interconnecting components and clocks
  - guarantee proper operation of system when strictly followed
- Approach depends on building blocks used for memory elements
  - we'll focus on systems with edge-triggered flip-flops
    - found in programmable logic devices such as our FPGA
  - many custom integrated circuits focus on level-sensitive latches
    - smaller, faster a bit more complicated to work with (CSE467)
- 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)

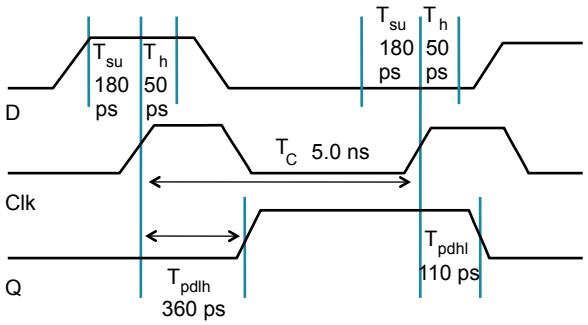


there is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognized



Typical timing specifications

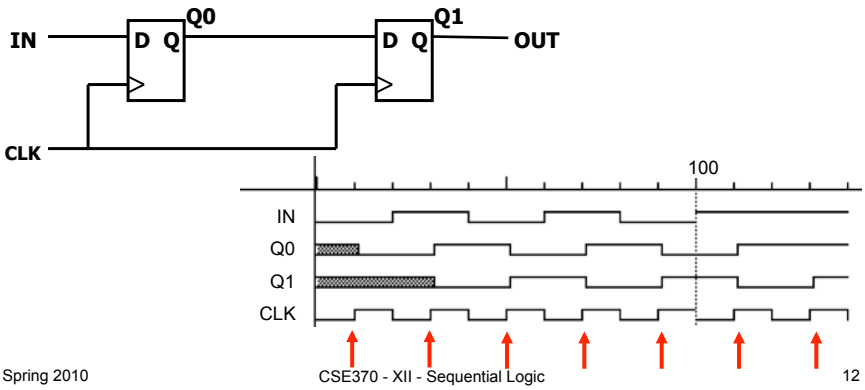
- Positive edge-triggered D flip-flop
  - setup and hold times
  - minimum clock width
  - propagation delays (low to high, high to low, max and typical)



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

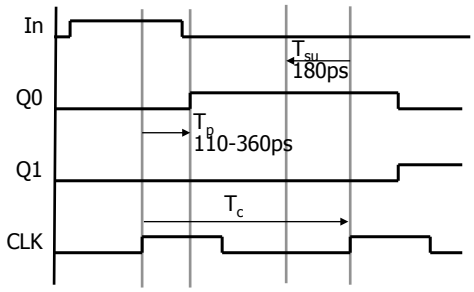
Cascading edge-triggered flip-flops

- All registers sample at exactly the same time
- Shift register
  - new value goes into first stage
  - while previous value of first stage goes into second stage



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

- Long path constraint
  - Tsu + Tp <= Tc

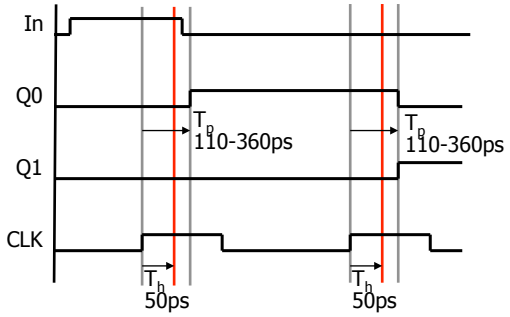


timing constraints guarantee proper operation of cascaded components

assumes infinitely fast distribution of the clock

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

- Short path constraint
  - Tp > Th
  - propagation delays exceed hold times
  - this guarantees following stage will latch current value before it changes to new value

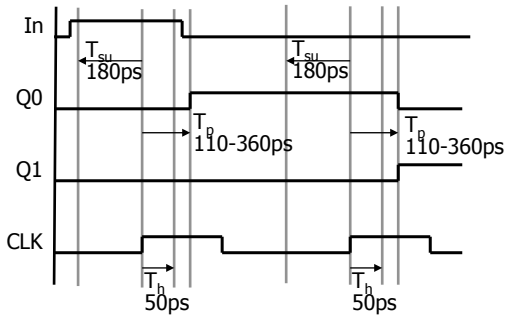


timing constraints guarantee proper operation of cascaded components

assumes infinitely fast distribution of the clock

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

- Why this works
  - propagation delays exceed hold times
  - this guarantees following stage will latch current value before it changes to new value

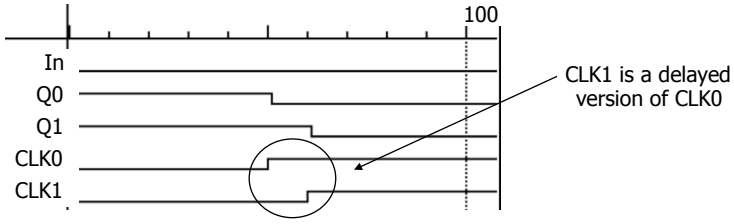


timing constraints guarantee proper operation of cascaded components

assumes infinitely fast distribution of the clock

Clock skew

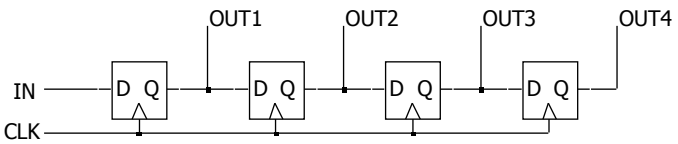
- When it doesn't work
  - correct behavior assumes next state of all storage elements determined by all storage elements at the same time
  - this is difficult in high-performance systems because time for clock to arrive at flip-flop is comparable to delays through logic
  - effect of skew on cascaded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

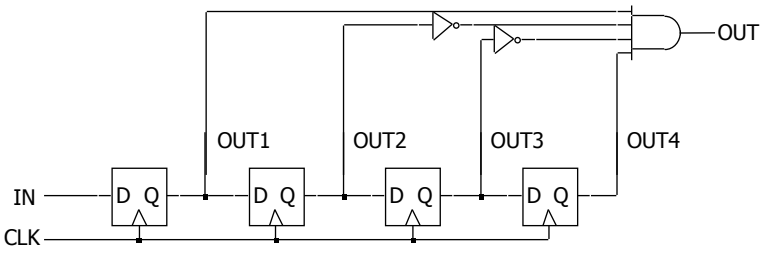
Shift register

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



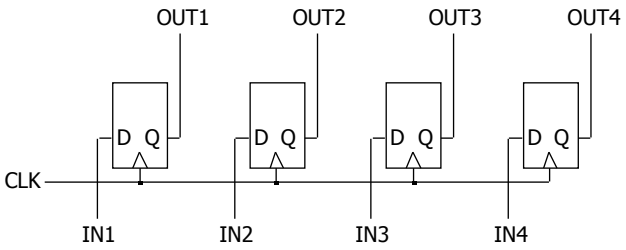
Pattern recognizer

- Combinational function of input samples
  - in this case, recognizing the pattern 1001 on the single input signal



Multi-Bit Registers

- Collection of 1-bit registers that share the same clock and controls
  - store a binary value
  - share clock, reset, and set lines

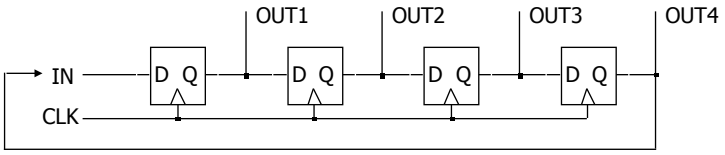


Register Control

- Reset
  - Clear the register to 0
  - Synchronous – happens on clock edge
  - Asynchronous – happens immediately
    - Dangerous
- Set
  - Set the register to 1
  - Synchronous or Asynchronous
- Enable
  - Register samples input only if enable = 1
  - aka “load”

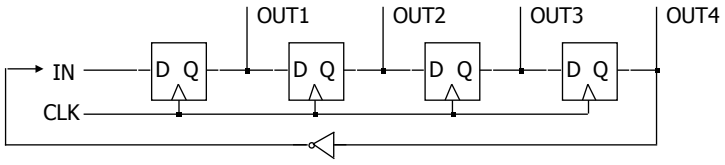
Simple "Counter"

- 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)



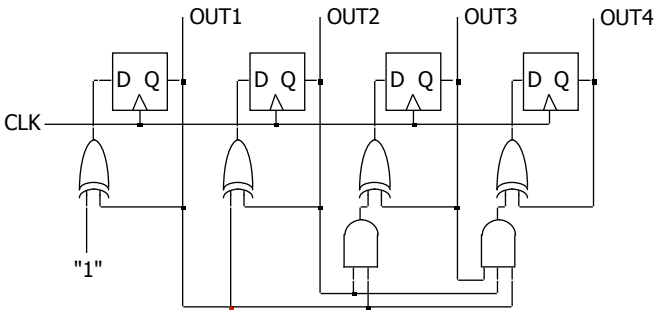
Activity

- How does this counter work (assuming it starts in state 0000)?



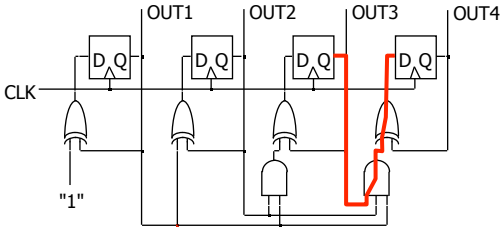
Binary counter

- Logic between registers (not just multiplexer)
  - XOR decides when bit should be toggled
  - always for low-order bit,
    only when first bit is true for second bit,
    and so on



How fast is our counter?

- $Tc > T_p + T_{pCL} + T_{su}$



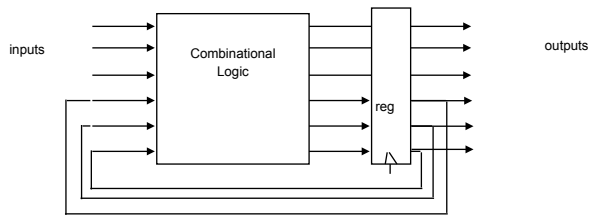
- Frequency < 1/Tc

Counter Controls

- Reset – return counter to 0
  - Usually Synchronous
- Load – start counter at a given value
- Enable – counter counts only when enable = 1
- Up/Down – count up when 1, count down when 0
- etc.
- How do you design these?

General Synchronous Design

- Combinational logic for the “next value” function
  - Can be a function of inputs, register value or both



Example: Versatile Shift Register

- Clear
- Load – parallel input
- Shift Left – serial input/output
- Shift Right – serial input/output
- Parallel output

Shift register application

- Parallel-to-serial conversion for serial transmission