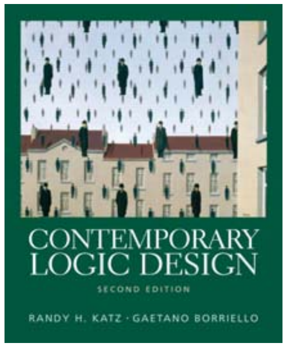
CSE 370 Spring 2006
Introduction to Digital Design

Lecture 16: Clock Skew, Asynchronous Inputs, Registers



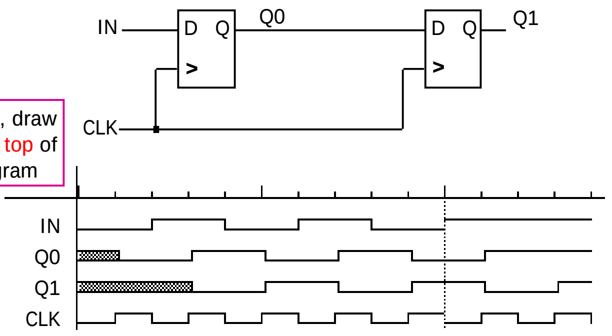
Last Lecture
- Timing Methodology
- Sequential Verilog

Today
- Clock Skew
- Asynchronous Inputs
- Registers

Administrivia

Cascading flip-flops

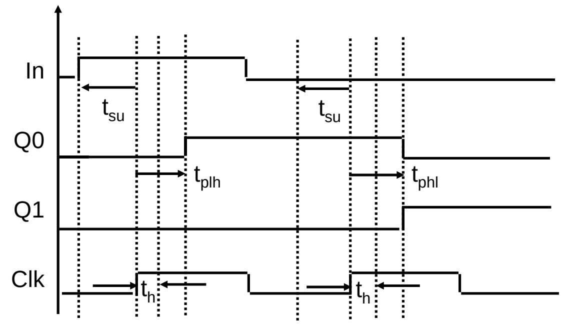
- Example: Shift registers
  - First FF acquires IN at rising clock edge
  - Second FF acquires Q0 at rising clock edge



Unlike this figure, draw your clock at the top of the timing diagram

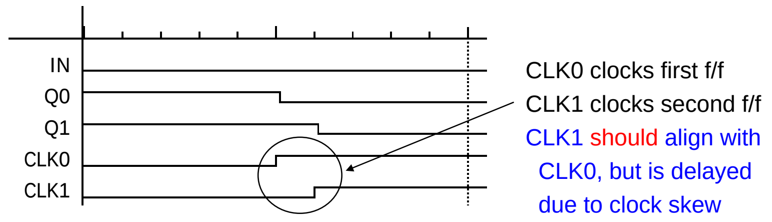
Cascading flip-flops (con't)

- Flip-flop propagation delays exceed hold times
  - Second stage latches its input before input changes



Clock skew

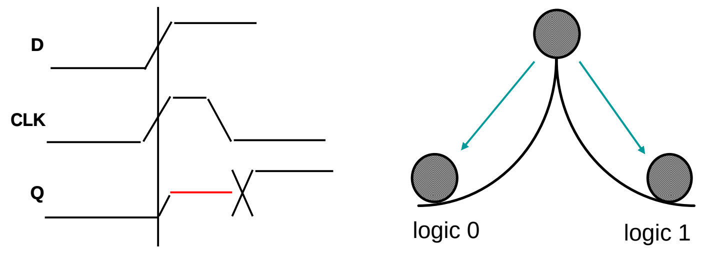
- Goal: Clock all flip-flops at the same time
  - Difficult to achieve in high-speed systems
    - Clock delays (wire, buffers) are comparable to logic delays
  - Problem is called clock skew


IN
Q0
Q1
CLK0
CLK1
CLK0 clocks first f/f
CLK1 clocks second f/f
CLK1 should align with CLK0, but is delayed due to clock skew

Original state: IN = 0, Q0 = 1, Q1 = 1
Next state: Q0 = 0, Q1 = 0 (should be Q1 = 1)

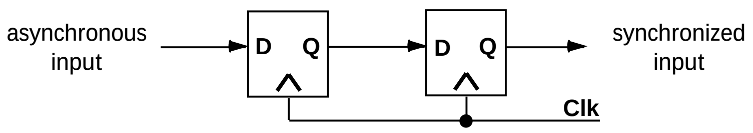
Synchronizer failure

- Occurs when FF input changes near clock edge
  - Input is neither 1 or 0 when clock goes high
  - Output may be neither 0 or 1
    - May stay undefined for a long time
  - Undefined state is called metastability



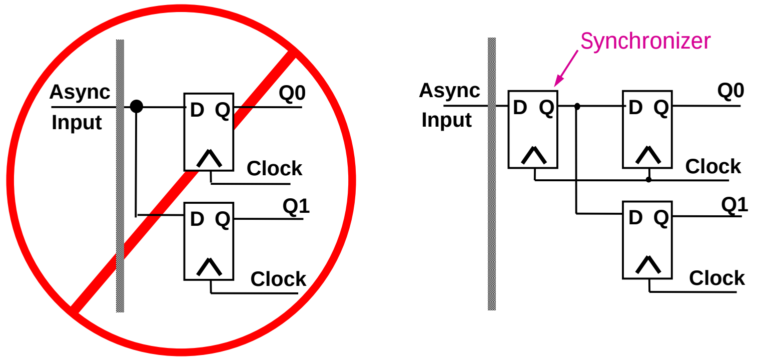
Minimizing synchronizer failures

- Failure probability can never be 0
  - Can make it small
  - Cascade two (or more) flip-flops
    - Effectively synchronizes twice
    - Both would have to fail for system to fail



Handling asynchronous inputs

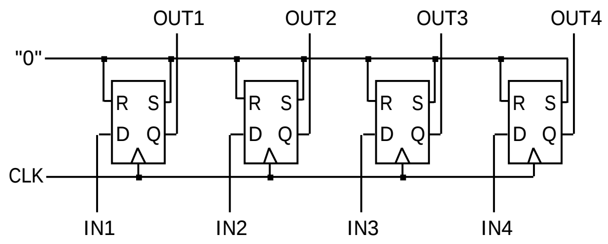
- Never fan-out asynchronous inputs
  - Synchronize at circuit boundary
  - Fan-out synchronized signal



Registers

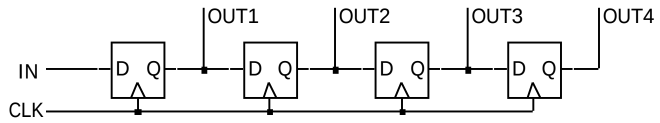
- Collection of flip-flops with common control
  - Store related values (e.g. a binary word)
  - Share clock, reset, and set lines
  - Examples
    - Storage registers, shift registers, counters

Example:
A 4-bit
storage
register



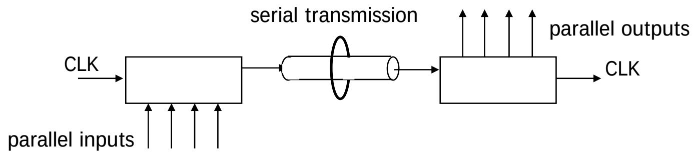
Shift registers

- Hold successively sampled input values
  - Delays values in time
  - Example: 4-bit shift register
    - Stores 4 input values in sequence

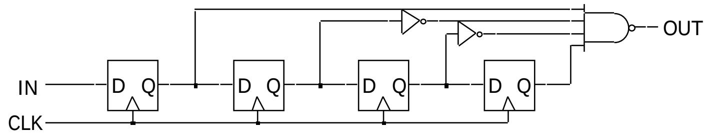


Shift-register applications

- Parallel-to-serial conversion for signal transmission

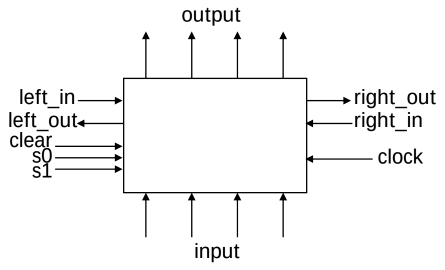


- Pattern recognition (circuit recognizes 1001)



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

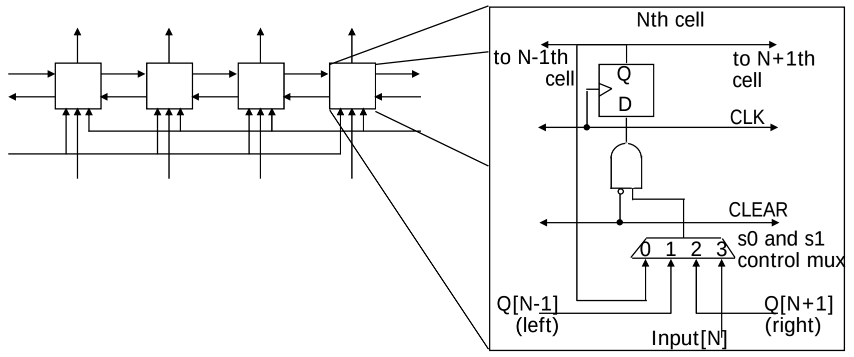
s1 and s0 determine the shift function

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

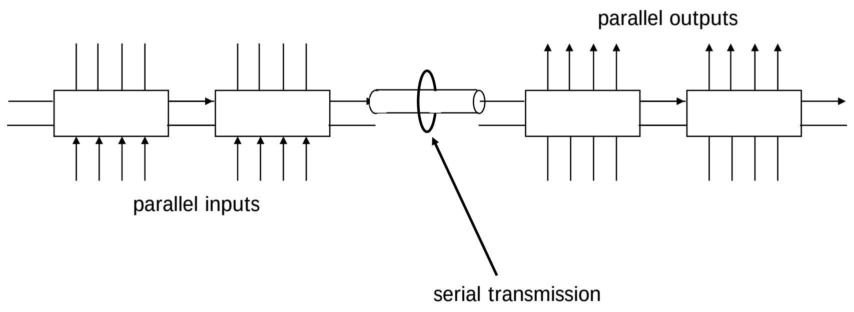
- Consider one of the four flip-flops
  - new value at next clock cycle:

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



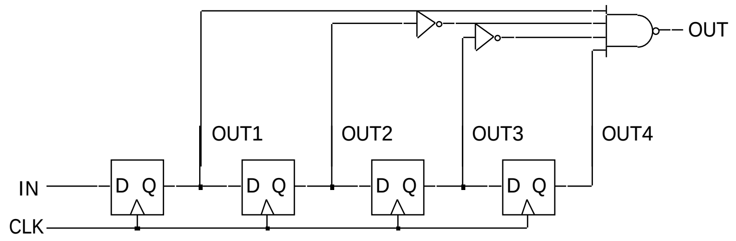
Shift register application

- Parallel-to-serial conversion for serial transmission



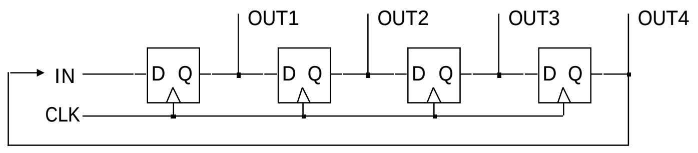
Pattern recognizer

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



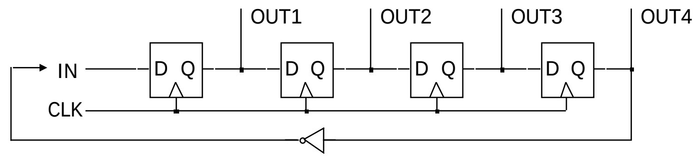
Counters

- 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)



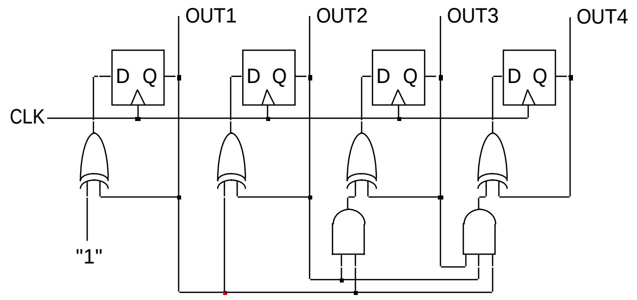
Activity

- How does this counter work?



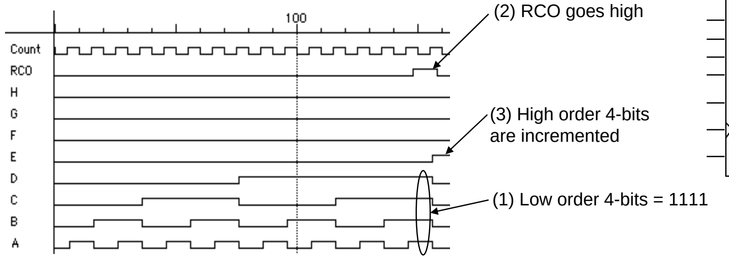
Binary counter

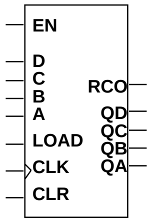
- Logic between registers (not just multiplexer)
  - XOR decides when bit should be toggled
  - always for low-order bit, only when first bit is true for second bit, and so on



Four-bit binary synchronous up-counter

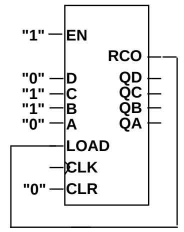
- Standard component with many applications
  - positive edge-triggered FFs w/ synchronous load and clear inputs
  - parallel load data from D, C, B, A
  - enable inputs: must be asserted to enable counting
  - RCO: ripple-carry out used for cascading counters
    - high when counter is in its highest state 1111
    - implemented using an AND gate



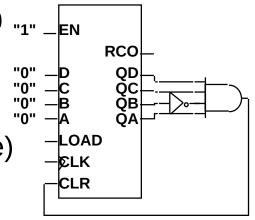


Offset counters

- Starting offset counters – use of synchronous load
  - e.g., 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1111, 0110, . . .



- Ending offset counter – comparator for ending value
  - e.g., 0000, 0001, 0010, ..., 1100, 1101, 0000



- Combinations of the above (start and stop value)