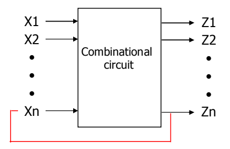
Sequential Logic

- 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

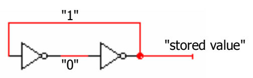
Circuits with feedback

- How to control feedback?
  - what stops values from cycling around endlessly?
  - this is an *asynchronous* sequential circuit

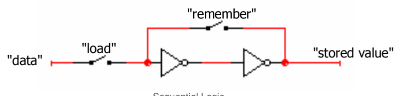


Simplest circuits with feedback: latch

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

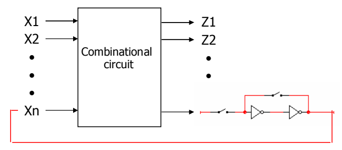


- How to get a new value into the memory cell?
  - selectively break feedback path
  - load new value into cell



Let's Use This Latch

- What happens?

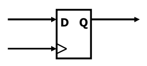


What We Need:

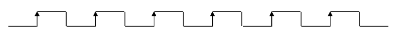
- When inputs change...
- Wait until combinational logic has finished and result it stable...
- Then sample the output value and save...
- Feed the saved output back to the input of the combinational logic
  - Make sure the saved output can't change
- Key idea: we sample the result at the right time, 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

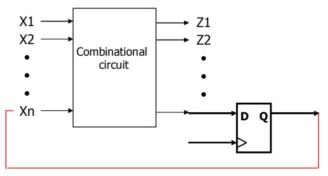


- Clock
  - Periodic signal, each rising edge signals D flip-flops to sample
  - All registers sample at the same time



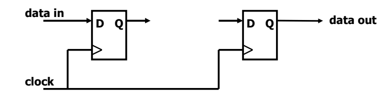
Let's Use This D flip-flop

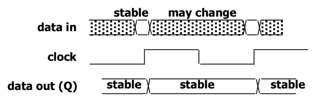
- Does this work?
- What do we need to say about the inputs X1, X2, ...?
- This is a synchronous sequential circuit



Registers

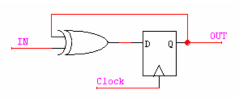
- Sample data using clock
- Hold data between clock cycles
- Computation (and delay) occurs between registers





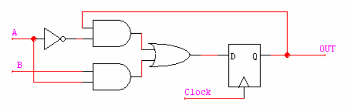
Example - Circuit with Feedback

- Output is a function of arbitrarily many past inputs



Examples (cont'd)

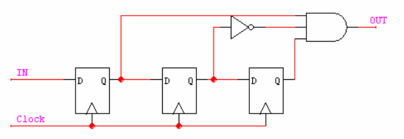
- Output is a function of inputs and the previous <u>state</u> of the circuit



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

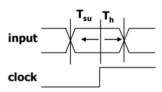
Example - Circuit without Feedback

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

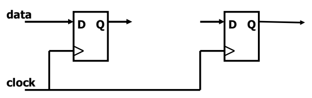


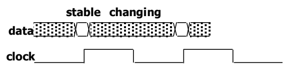
Timing Methodologies (cont'd)

- 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)



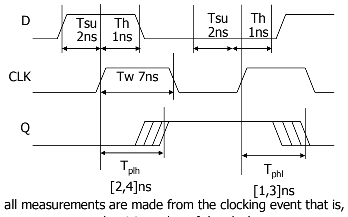
there is a timing "window" around the clocking event during which the input must remain stable and unchanged in order to be recognized





Typical timing specifications

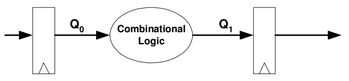
- Positive edge-triggered D flip-flop
  - setup and hold times
  - minimum clock width
  - propagation delays (low to high, high to low, max and typical)



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

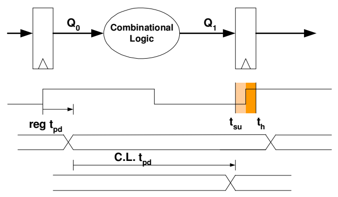
Synchronous System Model

- Register-to-register operation
- Perform operations during transfer
- Many transfers/operations occur simultaneously



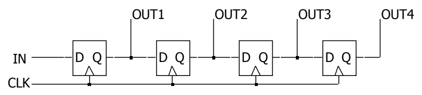
System Clock Frequency

- Register transfer must fit into one clock cycle
  - reg t_{pd} + C.L. t_{pd} + reg t_{su} < T_{clk}
  - Use maximum delays
  - Find the “critical path”
    - Longest register-register delay



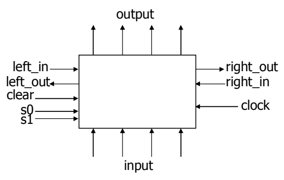
Shift register

- 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

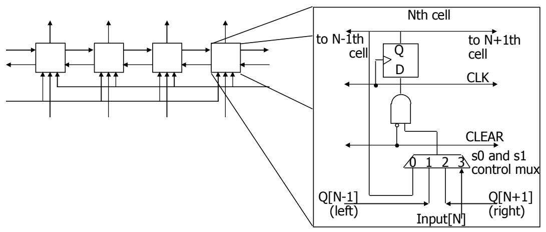
s1 and s0 determine the shift function

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

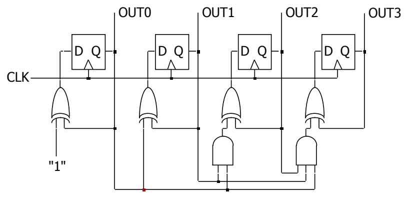
- Consider one of the four flip-flops
  - new value at next clock cycle:

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



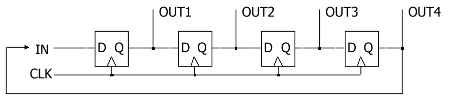
Binary counter

- 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

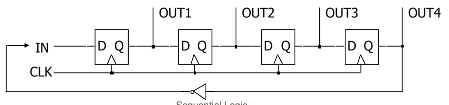


Other Counters: cheaper/faster

- 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


OUT1
OUT2
OUT3
OUT4
IN
CLK
D Q