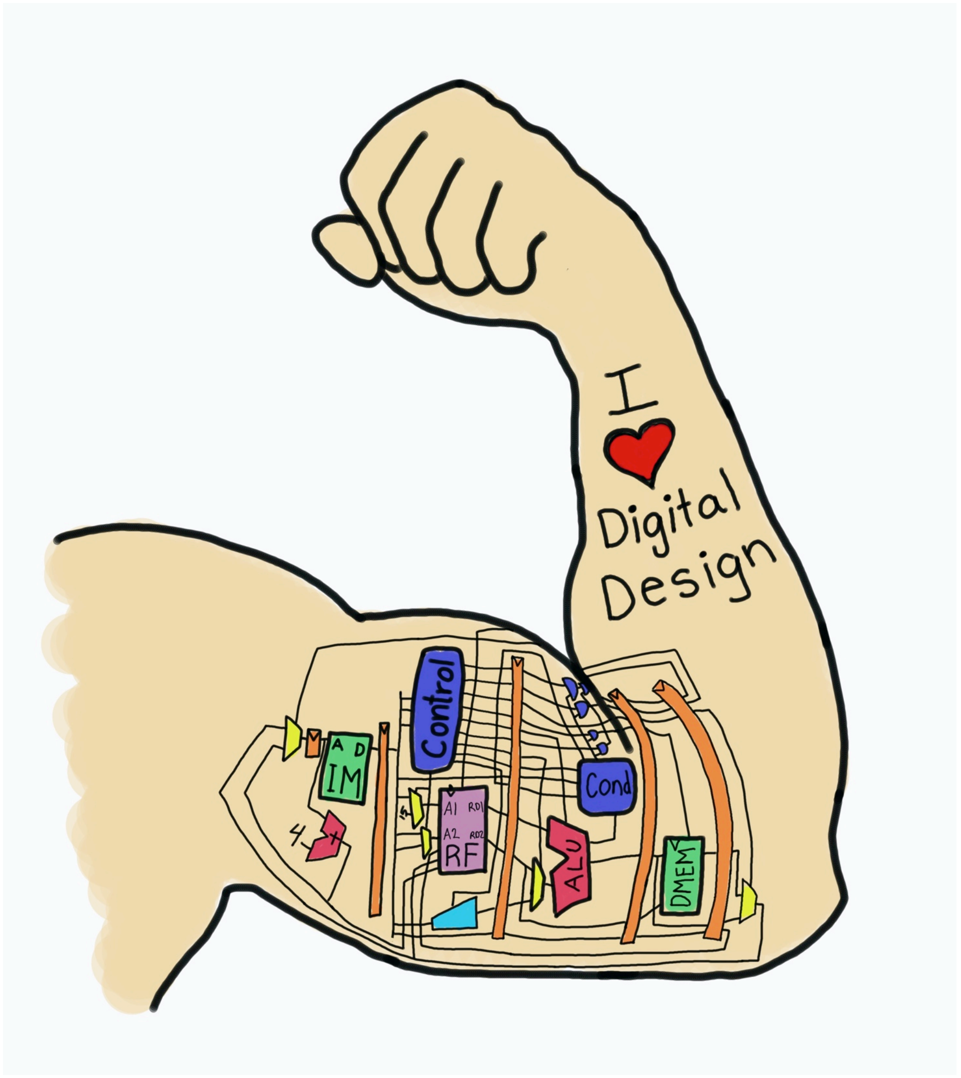
E85 Digital Design & Computer Engineering



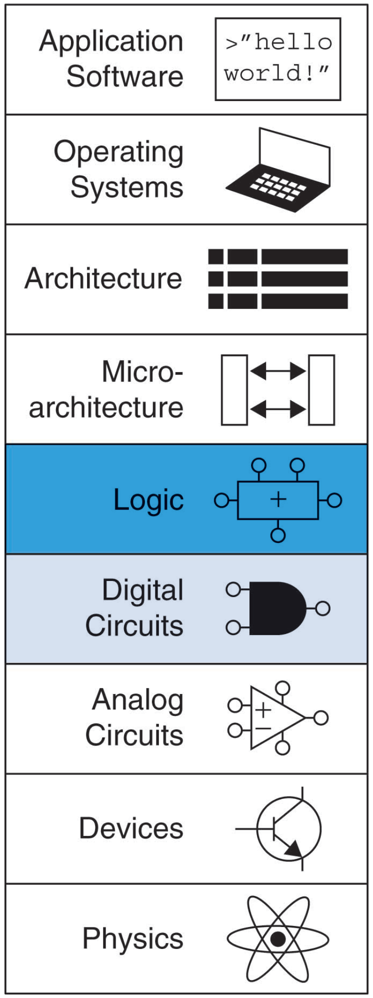
Lecture 3:

Timing & Sequential Circuits

HARVEY MUDD COLLEGE

Lecture 3

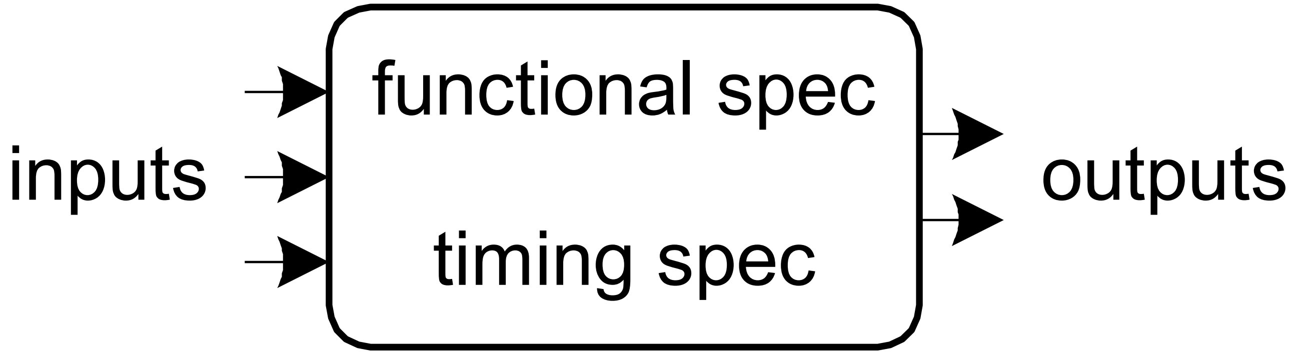
- **Timing**
- **Sequential Circuits**
- **Latches and Flip-Flops**
- **Synchronous Logic Design**



Introduction

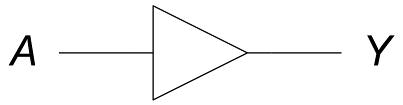
A logic circuit is composed of:

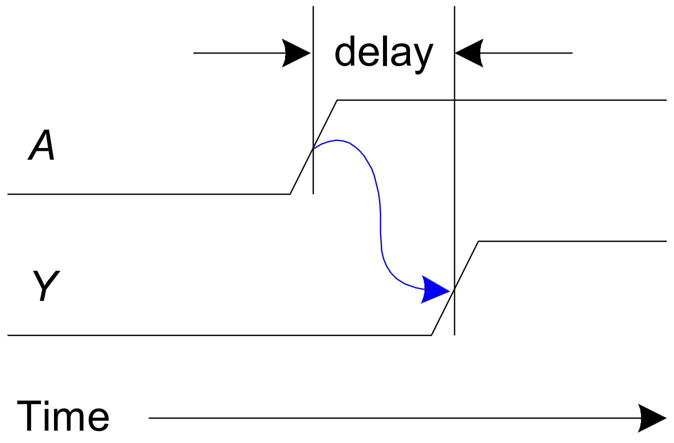
- Inputs
- Outputs
- Functional specification
- Timing specification



Timing

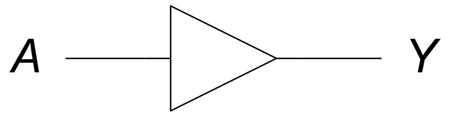
- **Delay:** time between input change and output changing
- How to build fast circuits?

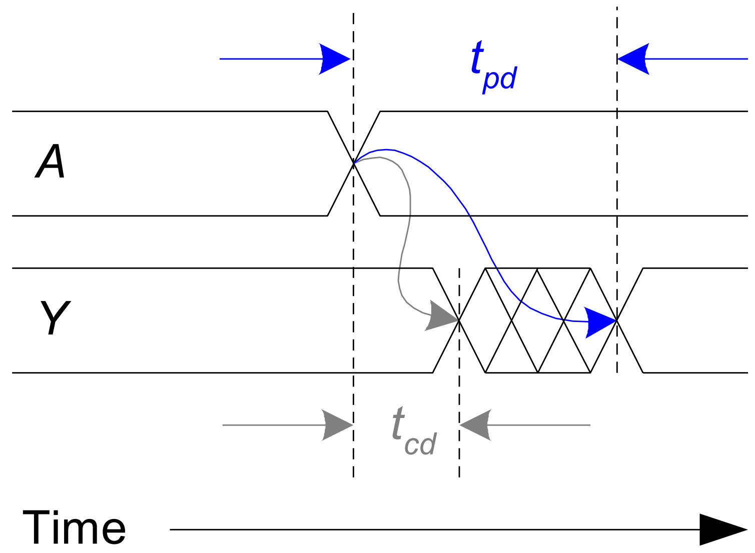




Propagation & Contamination Delay

- **Propagation delay:** t_{pd} = max delay from input to output
- **Contamination delay:** t_{cd} = min delay from input to output

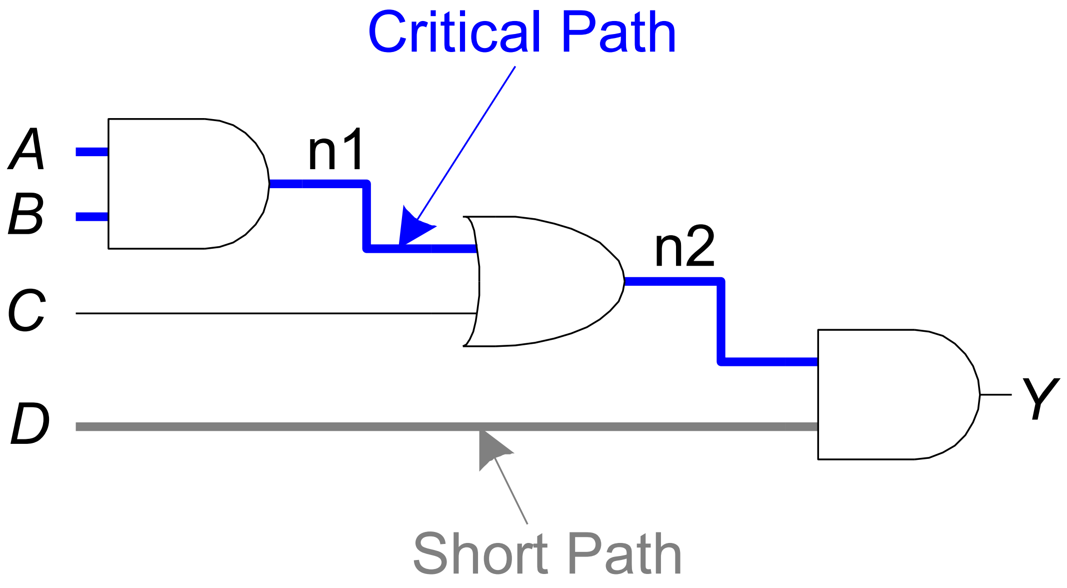




Propagation & Contamination Delay

- Delay is caused by
  - Capacitance and resistance in a circuit
  - Speed of light limitation
- Reasons why t_{pd} and t_{cd} may be different:
  - Different rising and falling delays
  - Multiple inputs and outputs, some of which are faster than others
  - Circuits slow down when hot and speed up when cold

Critical (Long) & Short Paths



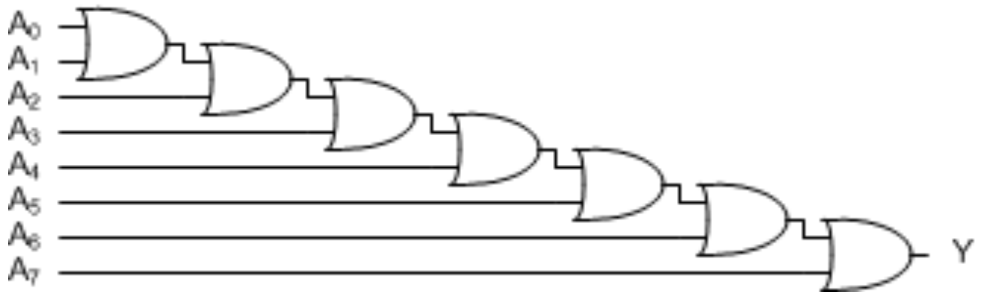
Critical (Long) Path: $t_{pd} = 2t_{pd\_AND} + t_{pd\_OR}$

Short Path: $t_{cd} = t_{cd\_AND}$

Example: 8-input OR Delay

- Find the minimum and maximum delay of this 8-input OR

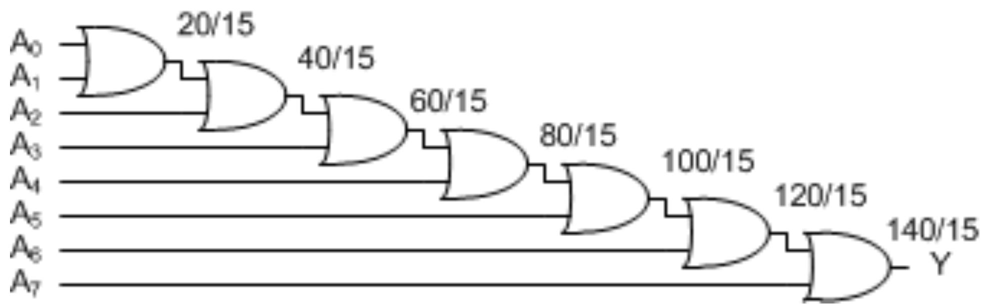
Cell	Propagation Delay (ps)	Contamination Delay (ps)
OR2	20	15
OR3	25	19
OR4	35	28



Solution: 8-input OR Delay

- Find the minimum and maximum delay of this 8-input OR

Cell	Propagation Delay (ps)	Contamination Delay (ps)
OR2	20	15
OR3	25	19
OR4	35	28



- Annotate each node with earliest and latest arrivals
- Min = 15 ps from A_7 to Y.
- Max = 140 ps from A_0 to Y.

Example: Optimized 8-input OR

- Redesign the 8-input OR to be as fast as possible.

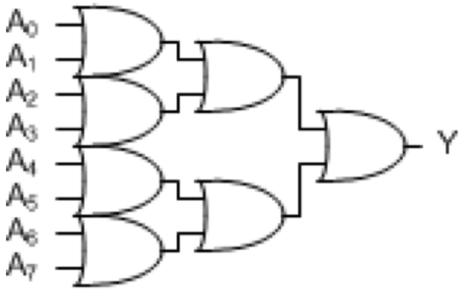
Cell	Propagation Delay (ps)	Contamination Delay (ps)
OR2	20	15
OR3	25	19
OR4	35	28

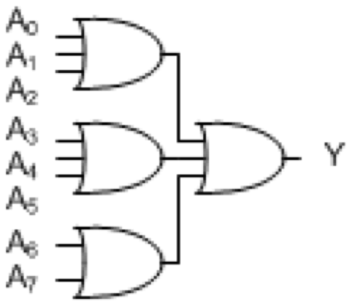
Example: Optimized 8-input OR

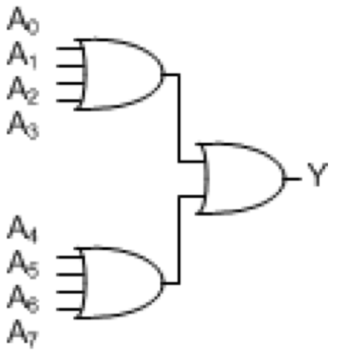
- Redesign the 8-input OR to be as fast as possible.

Cell	Propagation Delay (ps)	Contamination Delay (ps)
OR2	20	15
OR3	25	19
OR4	35	28

- Try various possibilities:





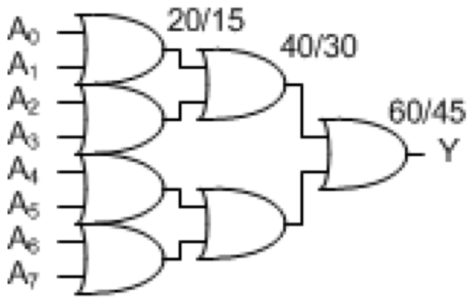


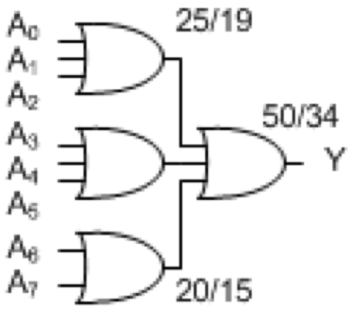
Solution: Optimized 8-input OR

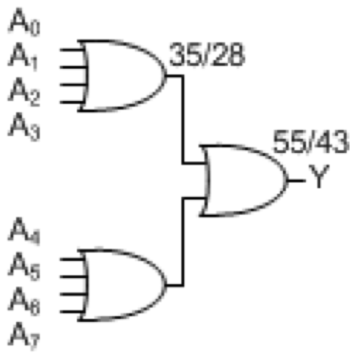
- Redesign the 8-input OR to be as fast as possible.

Cell	Propagation Delay (ps)	Contamination Delay (ps)
OR2	20	15
OR3	25	19
OR4	35	28

- Annotate delays. OR3+OR3 is fastest (50 ps).







Sequential Logic Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has ***memory***.
- Some definitions:
  - **State:** all the information about past inputs necessary to explain its future behavior
  - **Latches and flip-flops:** state elements that store one bit of state
  - **Synchronous sequential circuits:** combinational logic followed by a bank of flip-flops

Sequential Circuits

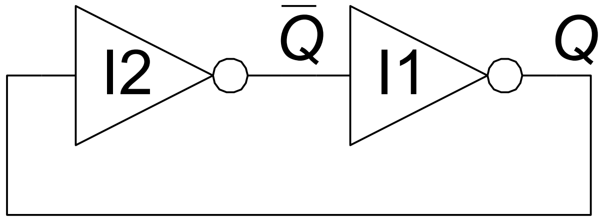
- Give sequence to events
- Have memory (short-term)
- Use feedback from output to input to store information

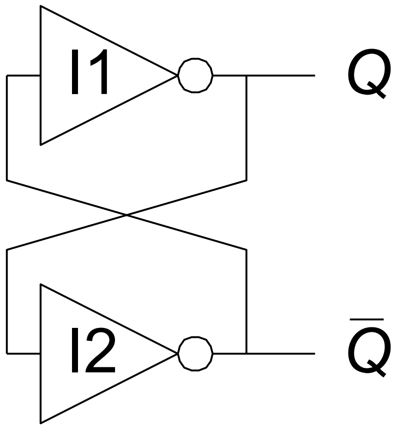
State Elements

- The state of a circuit influences its future behavior
- State elements store state
  - Bistable circuit
  - SR Latch
  - D Latch
  - D Flip-flop

Bistable Circuit

- Fundamental building block of other state elements
- Two outputs: $Q, \overline{Q}$
- No inputs

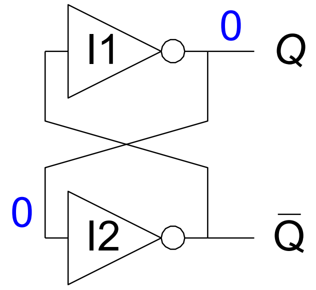




Bistable Circuit Analysis

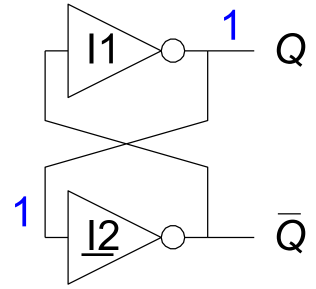
- Consider the two possible cases:
  - ***Q* = 0:**

    then $Q = 0, \overline{Q} =$



  - ***Q* = 1:**

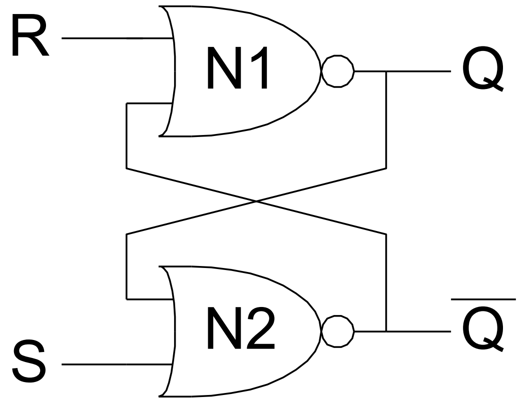
    then $Q = 1, \overline{Q} =$



- Stores 1 bit of state in the state variable, Q (or $\overline{Q}$)
- But there are **no inputs to control the state**

SR (Set/Reset) Latch

- SR Latch

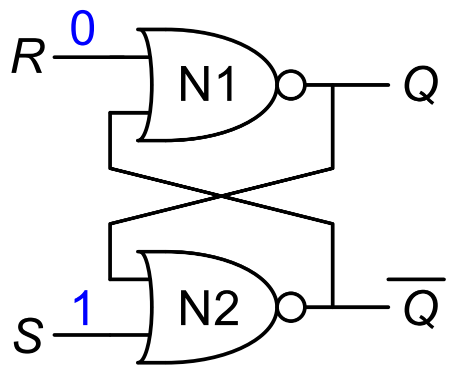


- Consider the four possible cases:
  - *S* = 1, *R* = 0
  - *S* = 0, *R* = 1
  - *S* = 0, *R* = 0
  - *S* = 1, *R* = 1

SR Latch Analysis

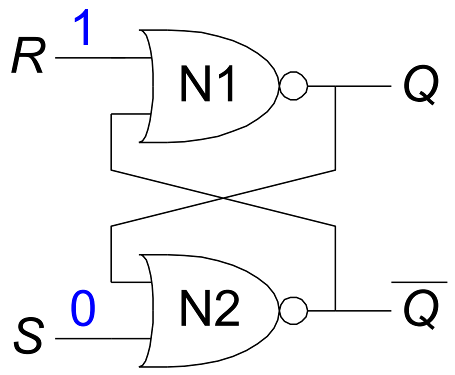
- **$S = 1, R = 0$:**

  then $Q =$ and $\overline{Q} =$



- **$S = 0, R = 1$:**

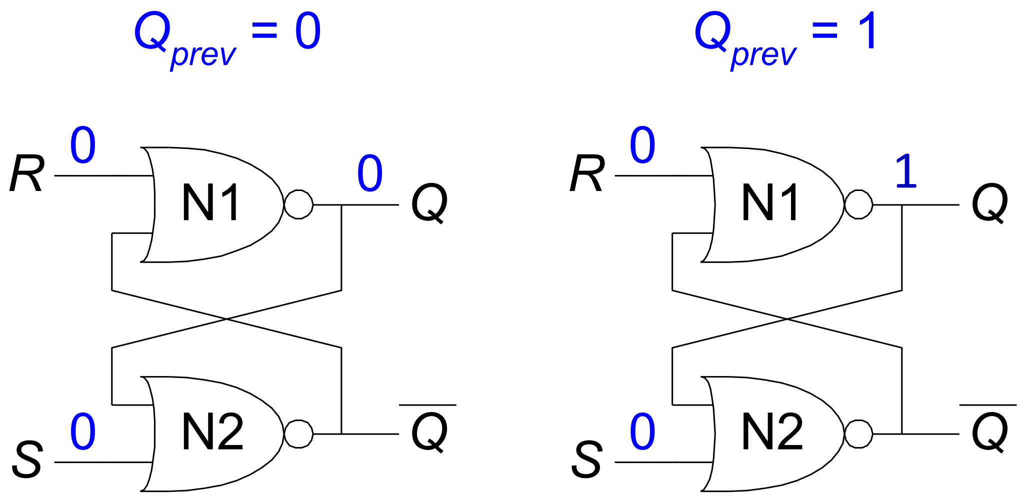
  then $Q =$ and $\overline{Q} =$



SR Latch Analysis

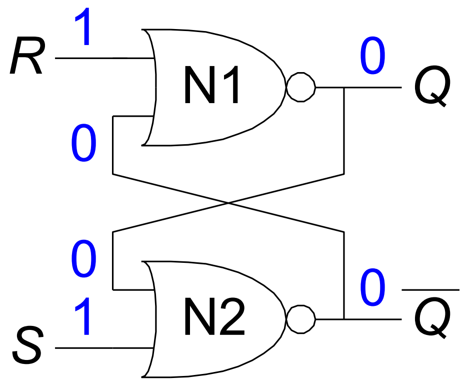
- ***S* = 0, *R* = 0:**

  then $Q = Q_{prev}$



- ***S* = 1, *R* = 1:**

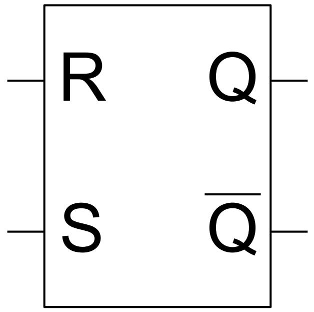
  then $Q = 0$, $\bar{Q} = 0$



SR Latch Symbol

- SR stands for Set/Reset Latch
  - Stores one bit of state (*Q*)
- Control what value is being stored with *S*, *R* inputs
  - **Set:** Make the output 1 (S = 1, R = 0, Q = **1**)
  - **Reset:** Make the output 0 (S = 0, R = 1, Q = **0**)
- **Avoid invalid state (when *S* = *R* = 1)**

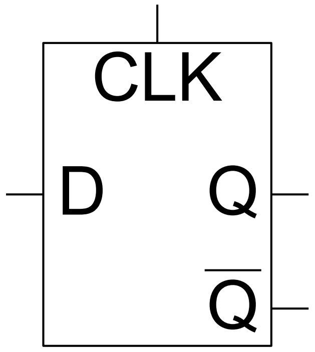
SR Latch Symbol



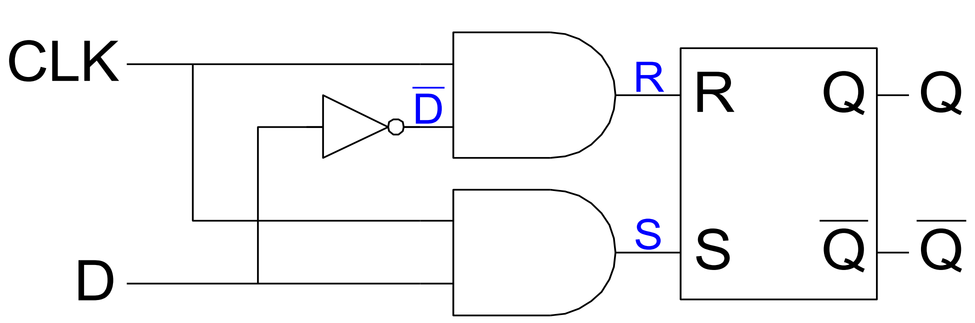
D Latch

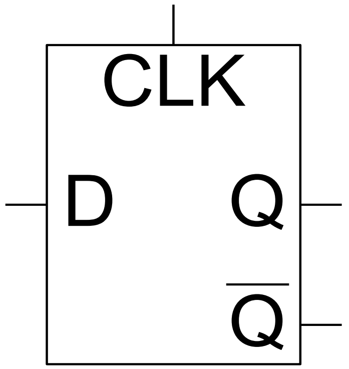
- Two inputs: *CLK*, *D*
  - ***CLK***: controls *when* the output changes
  - ***D*** (the data input): controls *what* the output changes to
- Function
  - When ***CLK* = 1**,
    D passes through to *Q* (*transparent*)
  - When ***CLK* = 0**,
    Q holds its previous value (*opaque*)
- Avoids invalid case when
  $Q \neq \text{NOT } \overline{Q}$

D Latch Symbol



D Latch Internal Circuit



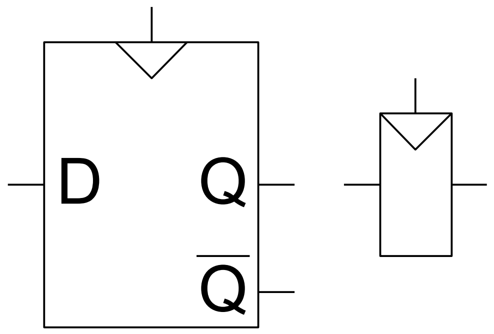


CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X					
1	0					
1	1					

D Flip-Flop

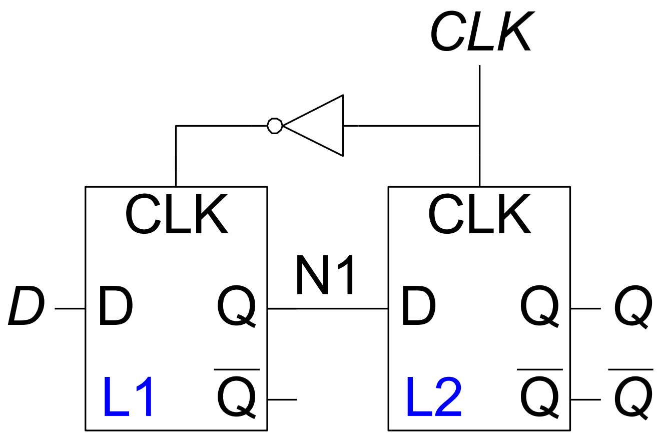
- **Inputs:** *CLK*, *D*
- **Function**
  - Samples *D* on rising edge of *CLK*
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - *Q* changes only on rising edge of *CLK*
- Called *edge-triggered*
- Activated on the clock edge

D Flip-Flop Symbols

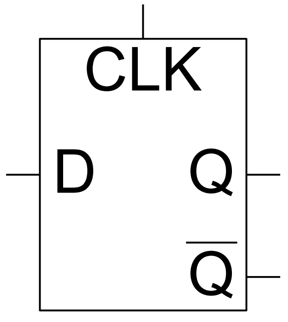


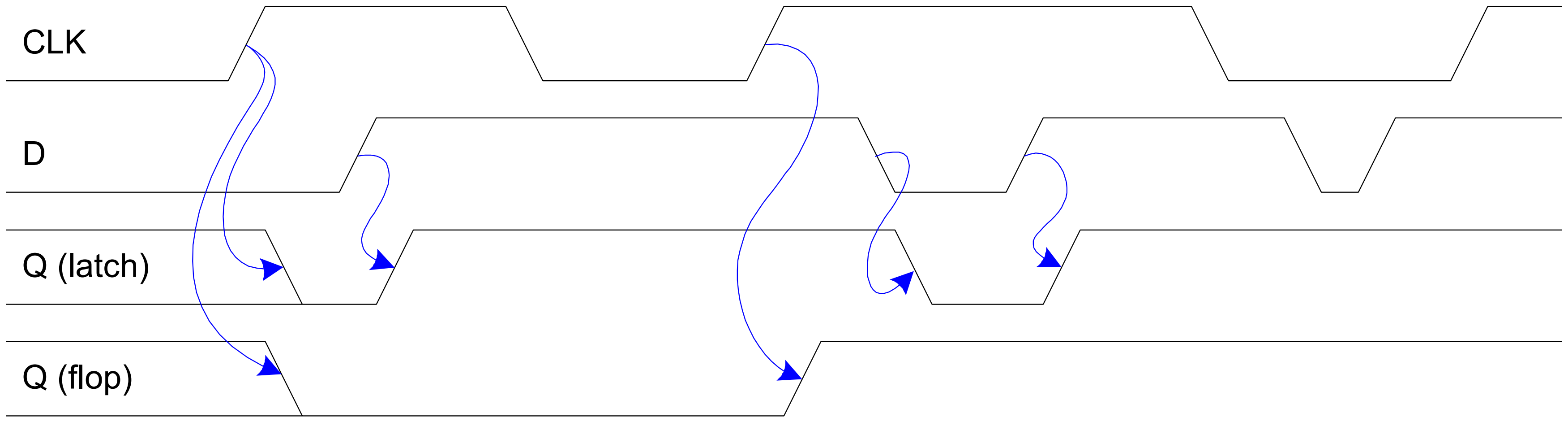
D Flip-Flop Internal Circuit

- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When ***CLK* = 0**
  - L1 is
  - L2 is
  - *D* passes through to N1
- When ***CLK* = 1**
  - L2 is
  - L1 is
  - N1 passes through to *Q*
- Thus, on the edge of the clock (when ***CLK* rises from 0→1**)
  - *D* passes through to *Q*

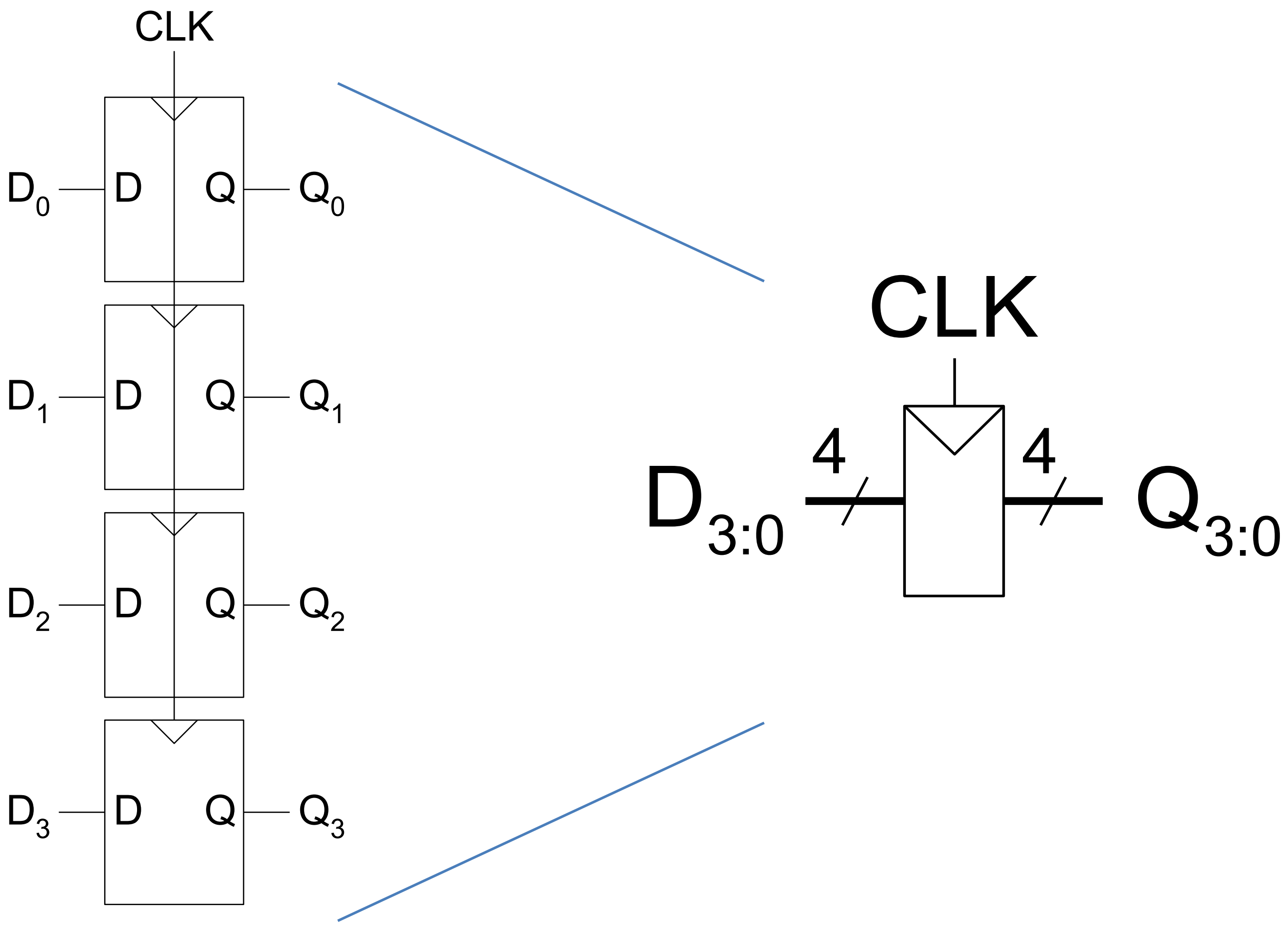


D Latch vs. D Flip-Flop





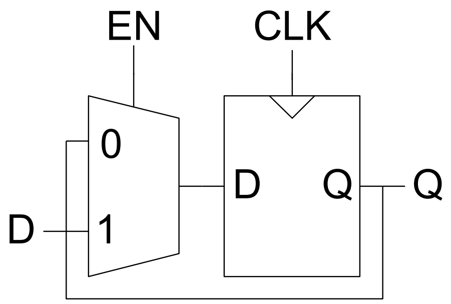
Registers: Multi-bit Flip-Flop



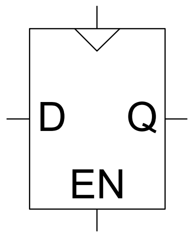
Enabled Flip-Flops

- **Inputs:** *CLK, D, EN*
  - The enable input (*EN*) controls when new data (*D*) is stored
- **Function**
  - ***EN* = 1:** *D* passes through to *Q* on the clock edge
  - ***EN* = 0:** the flip-flop retains its previous state

Internal Circuit



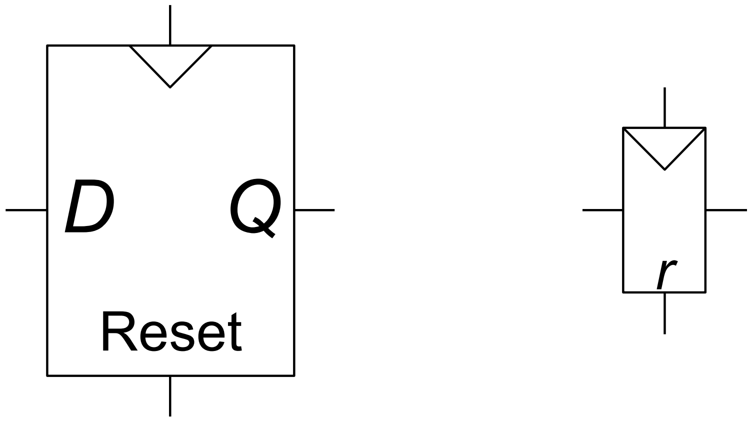
Symbol



Resettable Flip-Flops

- **Inputs:** *CLK*, *D*, *Reset*
- **Function:**
  - ***Reset* = 1:** *Q* is forced to 0
  - ***Reset* = 0:** flip-flop behaves as ordinary D flip-flop

Symbols

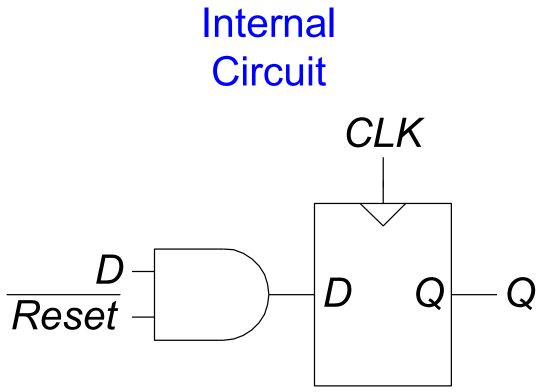


Resettable Flip-Flops

- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?

Resettable Flip-Flops

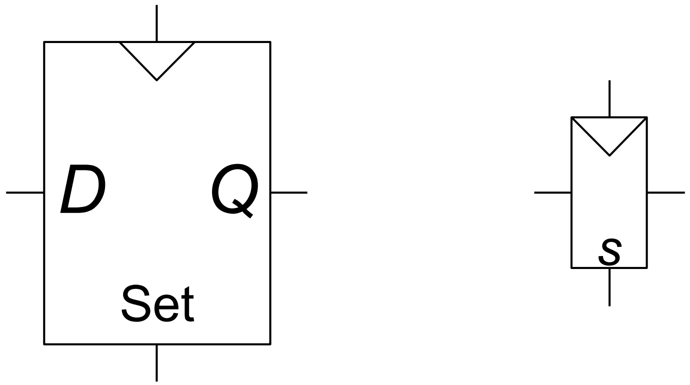
- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop



Settable Flip-Flops

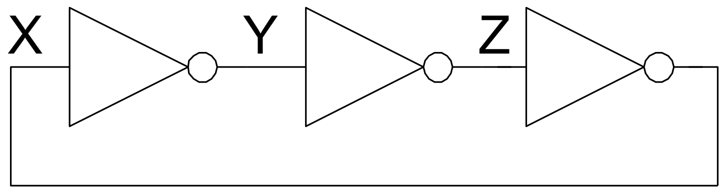
- **Inputs:** *CLK*, *D*, *Set*
- **Function:**
  - ***Set* = 1:** *Q* is set to 1
  - ***Set* = 0:** the flip-flop behaves as ordinary D flip-flop

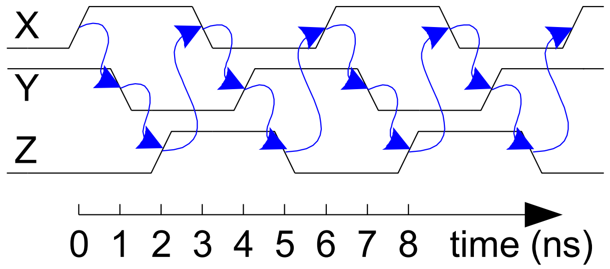
Symbols



Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:





- No inputs and 1-3 outputs
- Astable circuit, oscillates
- Period depends on inverter delay
- It has a ***cyclic path***: output fed back to input

SystemVerilog Description

```
module flop(input  logic clk, d,
            output logic q);

  always_ff @(posedge clk)
    q <= d;
endmodule
```

SystemVerilog Description

```
module flopenr(input  logic clk, en, reset, d,
               output logic q);

  always_ff @(posedge clk, posedge reset)
    if (reset)    q <= 0;
    else if (en) q <= d;
endmodule
```

• Multi-input XOR: Odd parity

ELSEVIER

SystemVerilog Description

```
module flopenr #(parameter WIDTH = 4)
                (input  logic clk, reset, en,
                 input  logic [WIDTH-1:0] d,
                 output logic [WIDTH-1:0] q);
  always_ff @(posedge clk, posedge reset)
    if (reset) q <= 0;
    else if (en) q <= d;
endmodule
```

• Multi-input XOR: Odd parity

Synchronous Sequential Logic Design

- Breaks cyclic paths by **inserting registers**
- Registers contain **state** of the system
- State changes at clock edge: system **synchronized** to the clock
- **Rules** of synchronous sequential circuit composition:
  - Every circuit element is either a register or a combinational circuit
  - At least one circuit element is a register
  - All registers receive the same clock signal
  - Every cyclic path contains at least one register
- Two common synchronous sequential circuits
  - Finite State Machines (FSMs)
  - Pipelines