SEQUENTIAL LOGIC DESIGN

Chapter 3

Professor Brendan Morris, SEB 3216, brendan.morris@unlv.edu
http://www.ee.unlv.edu/~b1morris/cpe100/

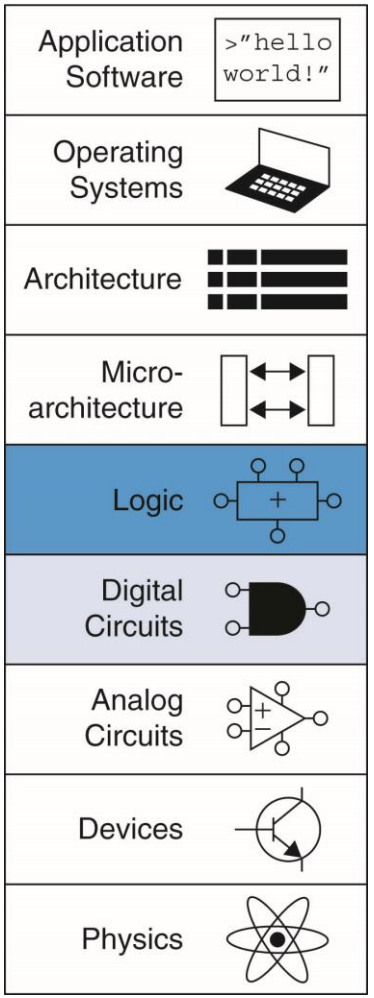
CPE100: Digital Logic Design I

Section 1004: Dr. Morris

Sequential Logic Design

Chapter 3 :: Topics

- **Introduction**
- **Latches and Flip-Flops**
- **Synchronous Logic Design**
- **Finite State Machines**
- **Timing of Sequential Logic**
- **Parallelism**



Introduction

- Previously, Combinational Logic design had outputs only depend on current value of inputs
- Outputs of sequential logic depend on current *and* prior input values – it has ***memory***.
- Some definitions:
  - **State:** all the information about a circuit 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

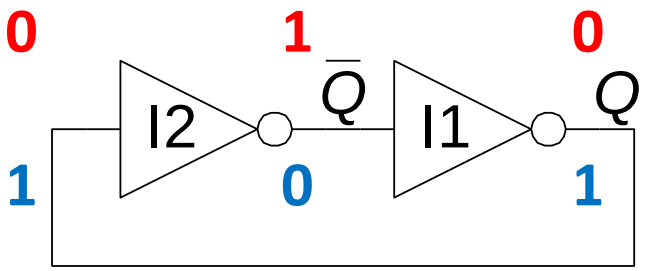
- Give sequence to events (i.e. a notion of time)
- Have memory (short-term)
- Use feedback from output to input to store information
  - Need to "remember" past output

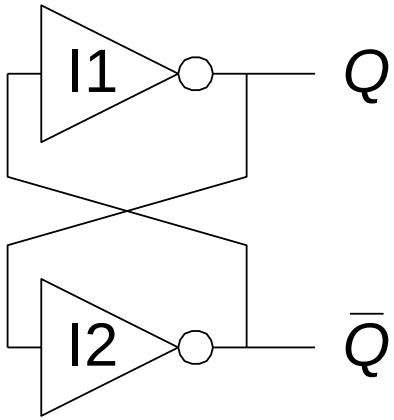
State Elements

- 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, $\overline{Q}$ (state)
- No inputs



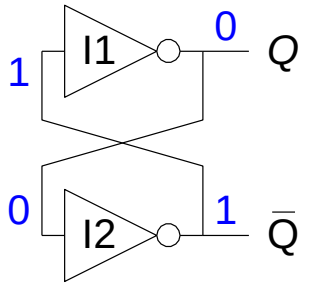


Redrawn circuit to emphasize symmetry

Bistable Circuit Analysis

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

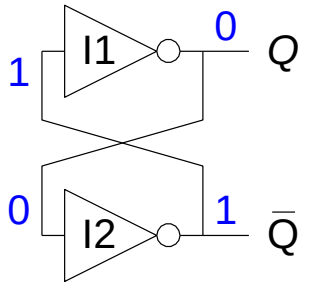
    then $\bar{Q} = 1$, $Q = 0$ (consistent)



Bistable Circuit Analysis

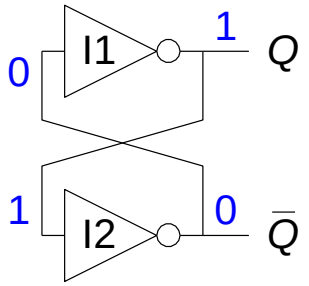
- Consider the two possible cases:
  - $Q = 0$:

    then $\bar{Q} = 1$, $Q = 0$ (consistent)



  - $Q = 1$:

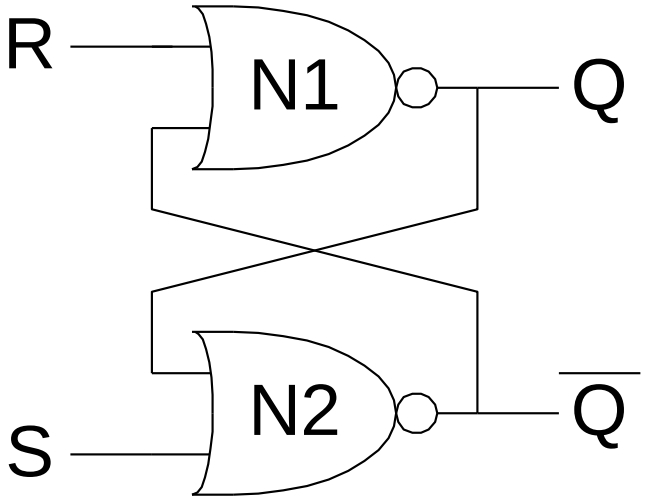
    then $\bar{Q} = 0$, $Q = 1$ (consistent)



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

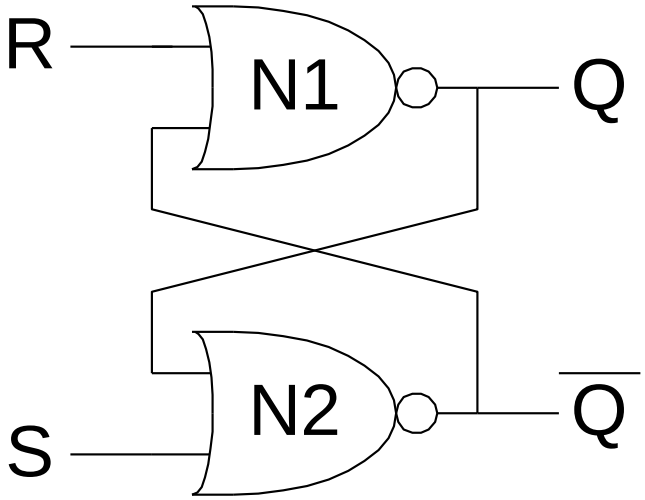
SR (Set/Reset) Latch

- SR Latch
  - S – set Q=1
  - R – reset Q=0



SR (Set/Reset) Latch

- SR Latch

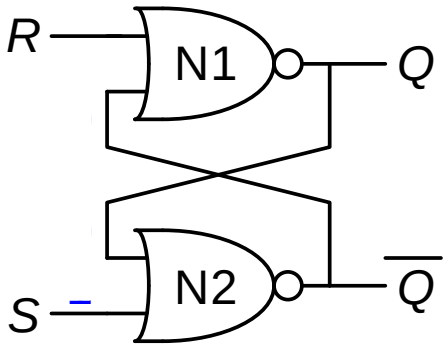


- Consider the four possible cases:
  - $S = 1, R = 0$
  - $S = 0, R = 1$
  - $S = 0, R = 0$
  - $S = 1, R = 1$

SR Latch Analysis

- $S = 1, R = 0$:

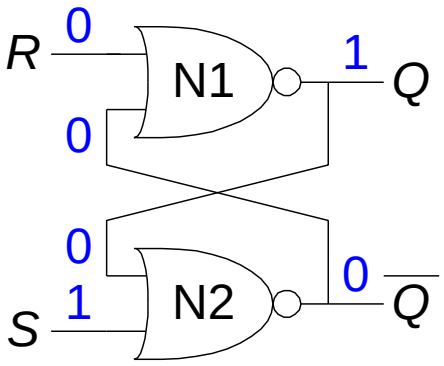
  then $Q = 1$ and $\overline{Q} = 0$



SR Latch Analysis

- $S = 1, R = 0$:

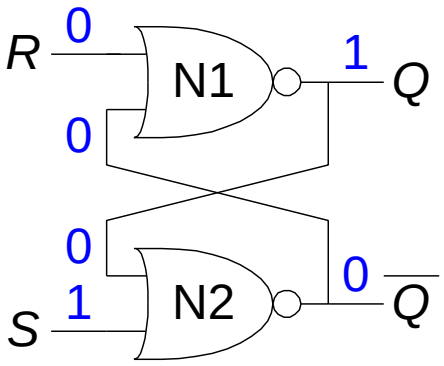
  then $Q = 1$ and $\bar{Q} = 0$



SR Latch Analysis

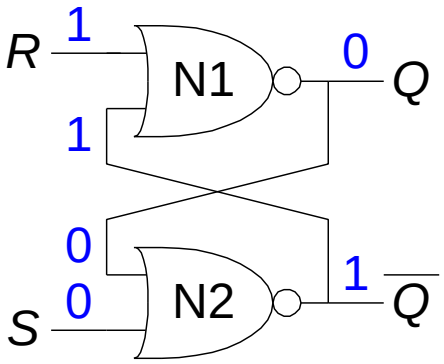
- $S = 1, R = 0$:

  then $Q = 1$ and $\overline{Q} = 0$



- $S = 0, R = 1$:

  then $Q = 0$ and $\overline{Q} = 1$

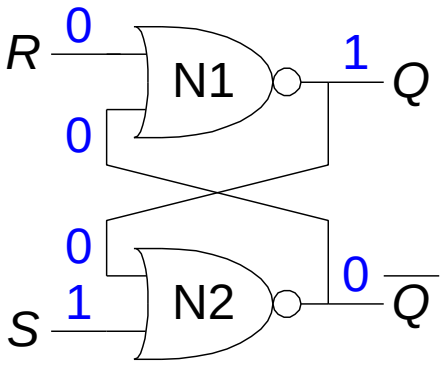


SR Latch Analysis

- $S = 1, R = 0$:

  then $Q = 1$ and $\overline{Q} = 0$

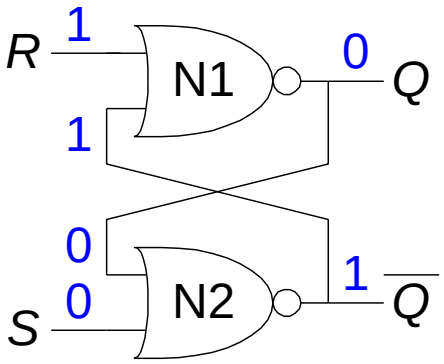
  ***Set* the output**



- $S = 0, R = 1$:

  then $Q = 0$ and $\overline{Q} = 1$

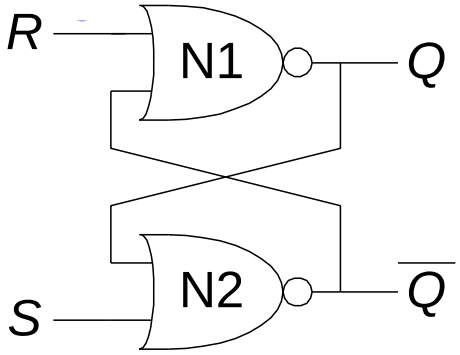
  ***Reset* the output**

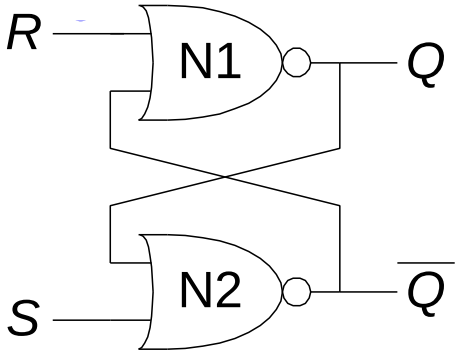


SR Latch Analysis

- $S = 0, R = 0$:

  then $Q = Q_{prev}$



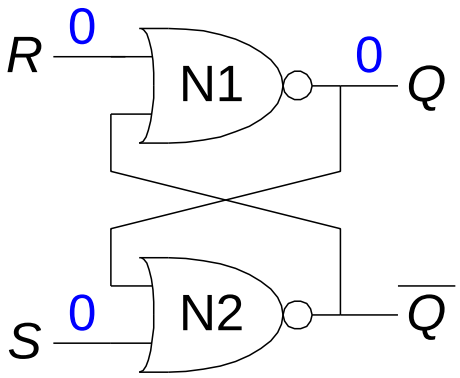


SR Latch Analysis

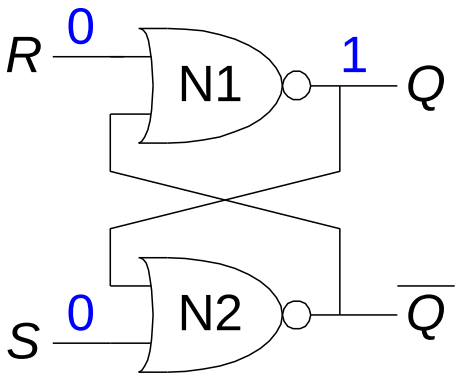
- $S = 0, R = 0$:

  then $Q = Q_{prev}$

$Q_{prev} = 0$



$Q_{prev} = 1$

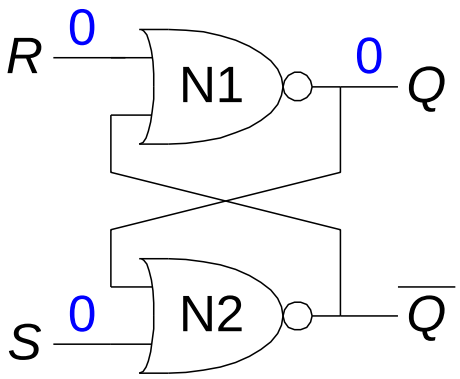


SR Latch Analysis

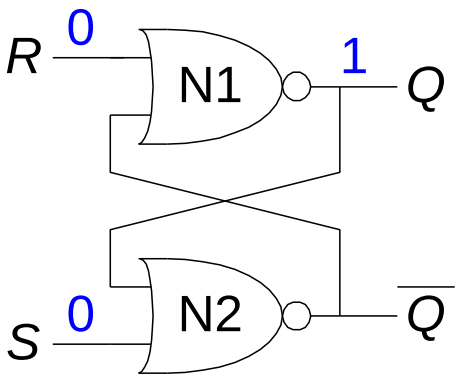
- $S = 0, R = 0$:

  then $Q = Q_{prev}$

$Q_{prev} = 0$

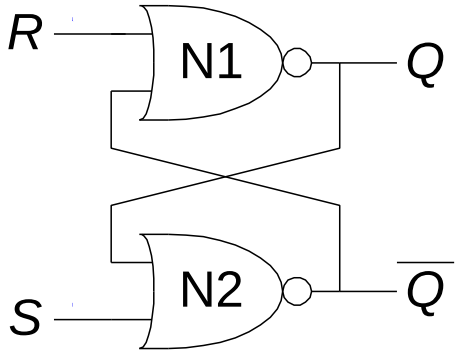


$Q_{prev} = 1$



- $S = 1, R = 1$:

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

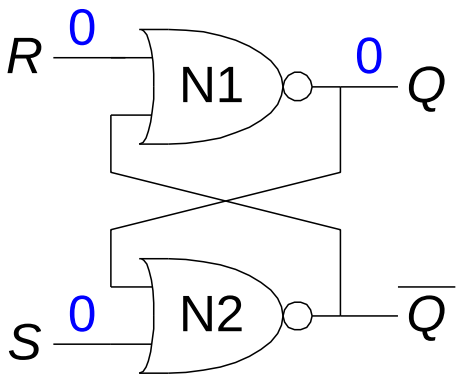


SR Latch Analysis

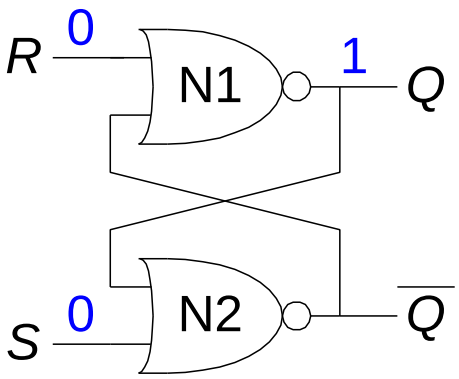
- $S = 0, R = 0$:

  then $Q = Q_{prev}$

$Q_{prev} = 0$

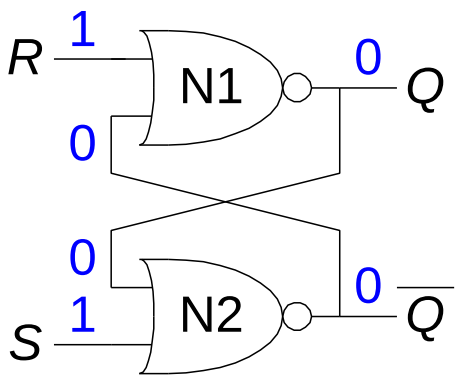


$Q_{prev} = 1$



- $S = 1, R = 1$:

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



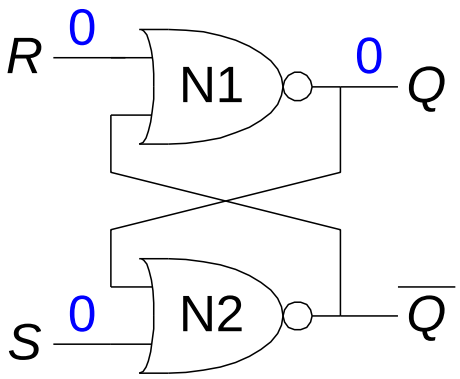
SR Latch Analysis

- $S = 0, R = 0$:

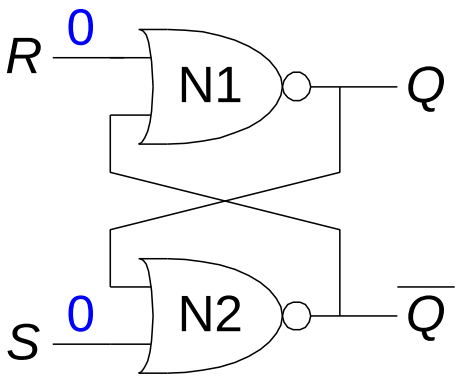
  then $Q = Q_{prev}$

  Memory!

$Q_{prev} = 0$



$Q_{prev} = 1$

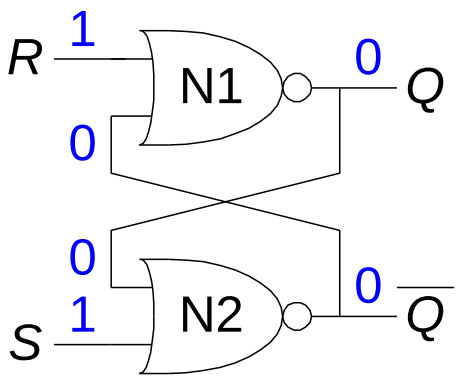


- $S = 1, R = 1$:

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

  Invalid State

  $\bar{Q} \neq$ NOT Q



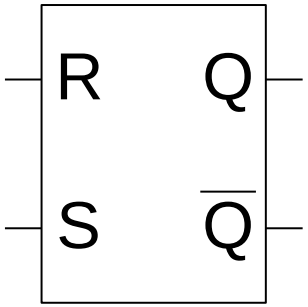
SR Latch Symbol

- 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

    ($S = 1$, $R = 0$, $Q = \mathbf{1}$)
  - **Reset:** Make the output 0

    ($S = 0$, $R = 1$, $Q = \mathbf{0}$)

SR Latch Symbol



D Latch

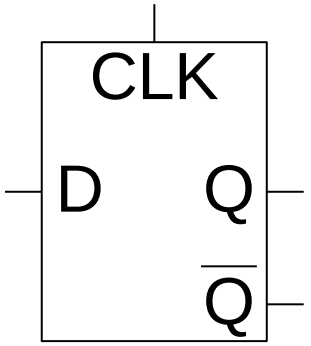
- Two inputs: *CLK*, *D*
  - ***CLK*:** controls *when* the output changes
  - ***D*** (the data input): controls *what* the output changes to
- Function
  - When ***CLK* = 1**,

    D passes through to *Q* (*transparent*)
  - When ***CLK* = 0**,

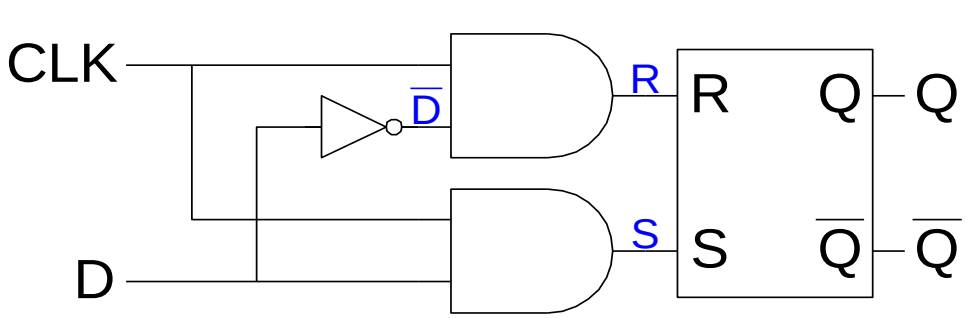
    Q holds its previous value (*opaque*)
- Avoids invalid case when

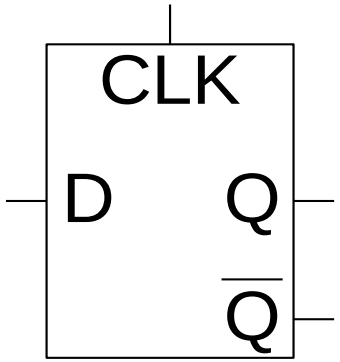
  $Q \neq \text{NOT } \overline{Q}$

D Latch Symbol



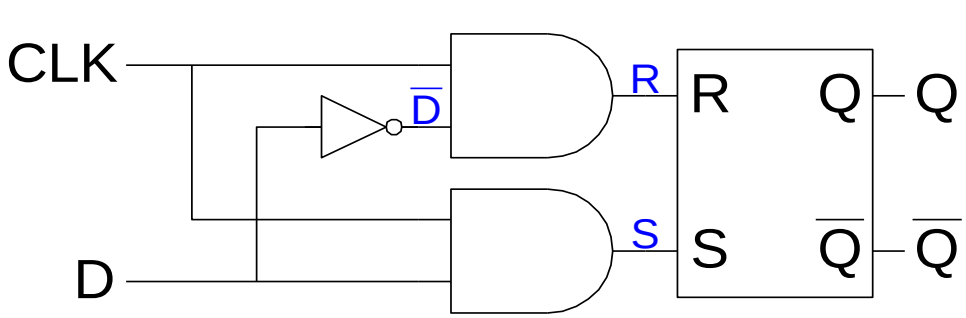
D Latch Internal Circuit

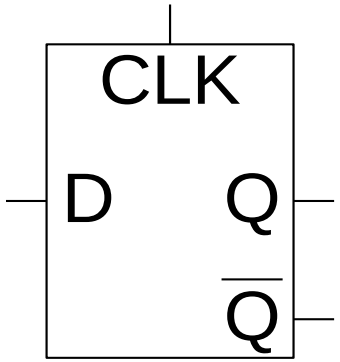




CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X					
1	0					
1	1					

D Latch Internal Circuit



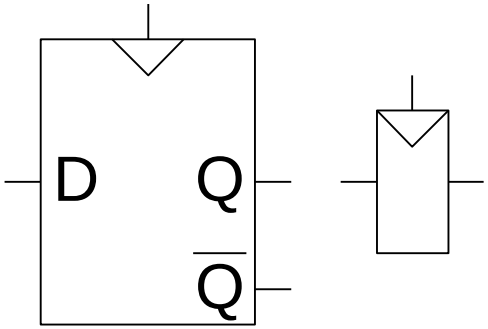


CLK	*D*	$\overline{D}$	*S*	*R*	*Q*	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

D Flip-Flop

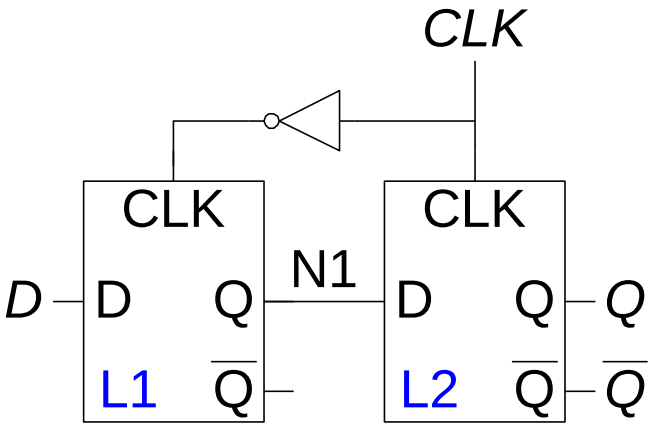
- **Inputs:** *CLK*, *D*
- **Function**
  - Samples *D* on rising edge of *CLK*
    - When *CLK* rises from 0 to 1, *D* passes through to *Q*
    - Otherwise, *Q* holds its previous value
  - *Q* changes only on rising edge of *CLK*
- Called *edge-triggered*
- Activated on the clock edge

D Flip-Flop Symbols



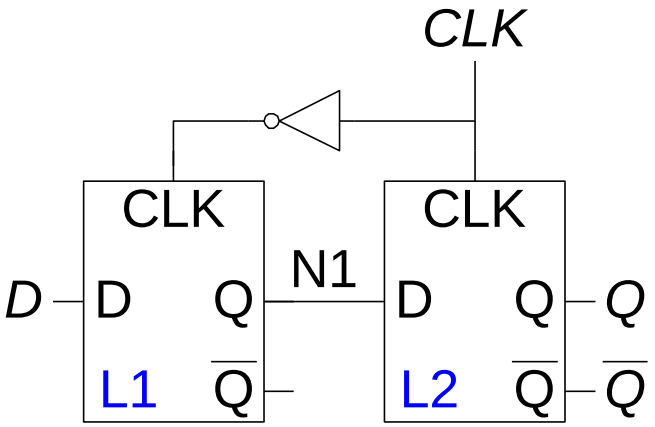
D Flip-Flop Internal Circuit

- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When ***CLK = 0***
  - L1 is transparent
  - L2 is opaque
  - *D* passes through to N1



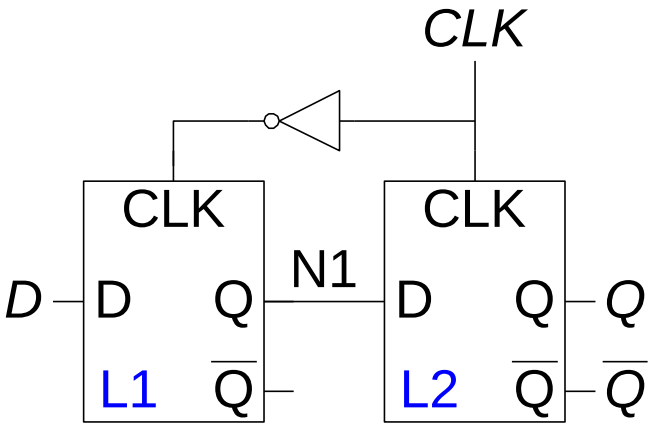
D Flip-Flop Internal Circuit

- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When ***CLK* = 0**
  - L1 is transparent
  - L2 is opaque
  - *D* passes through to N1
- When ***CLK* = 1**
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to *Q*

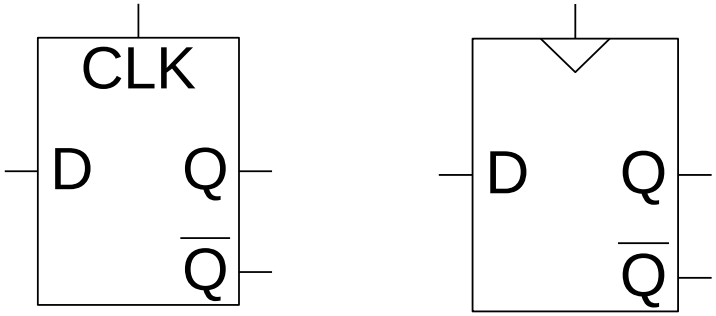


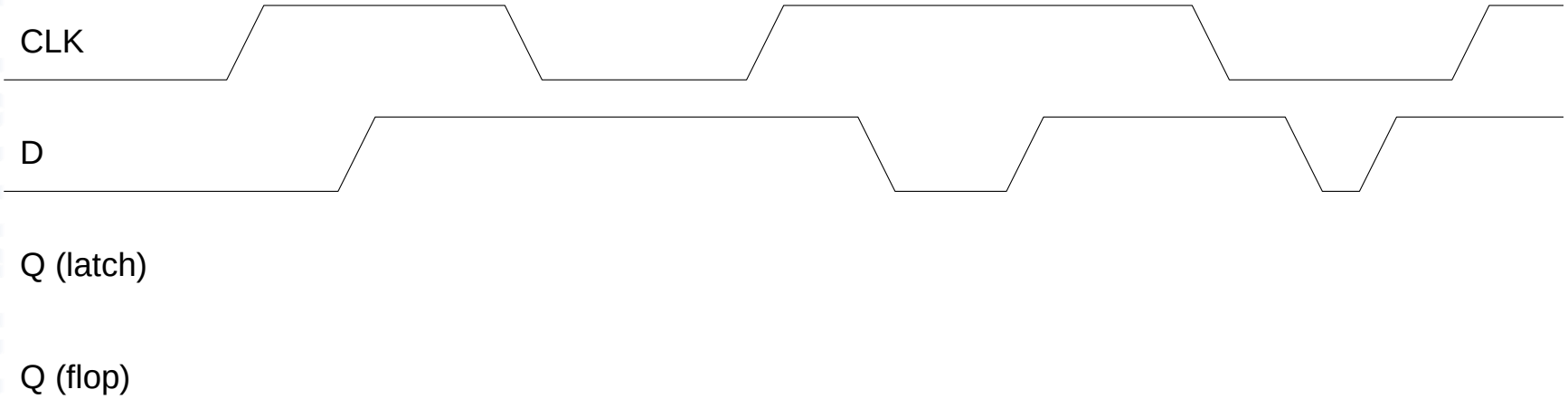
D Flip-Flop Internal Circuit

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

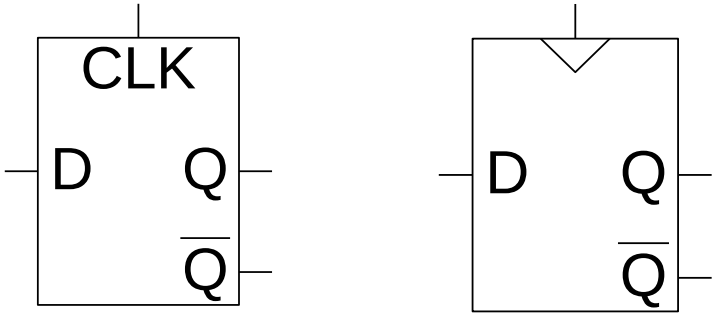


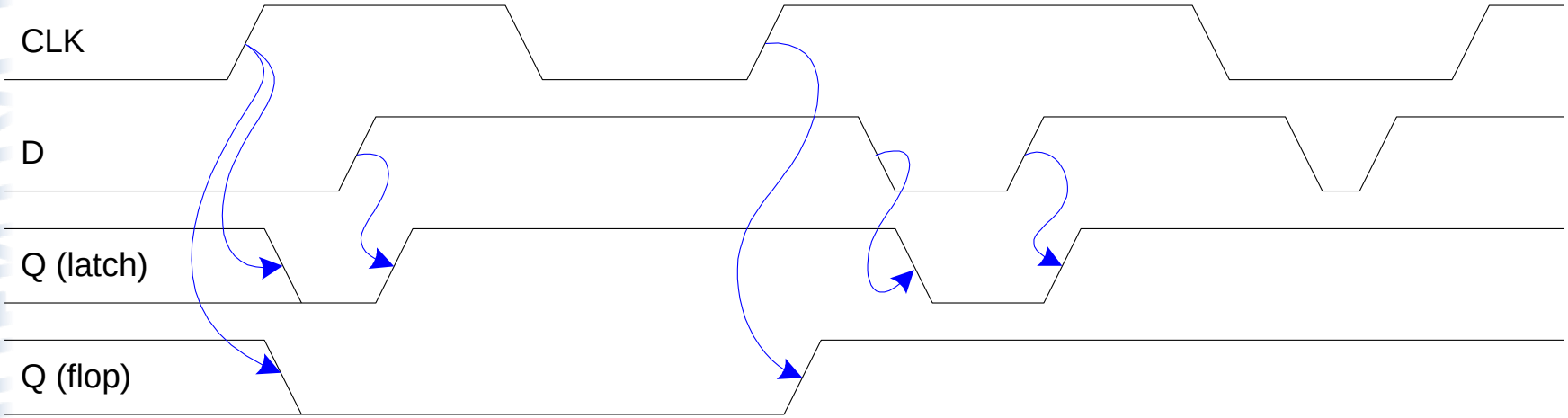
D Latch vs. D Flip-Flop





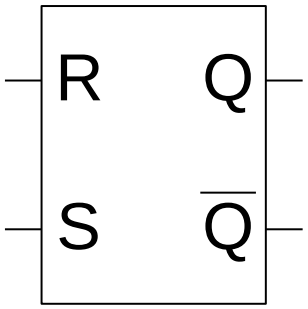
D Latch vs. D Flip-Flop





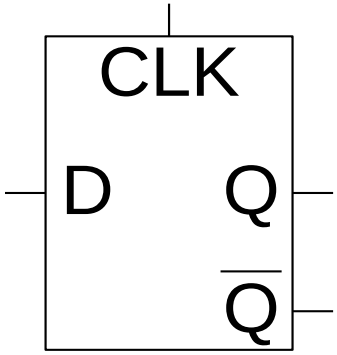
Review

SR Latch



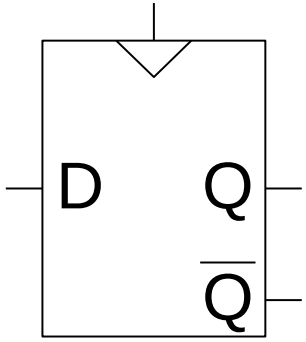
S = 1, R = 0: Q = 1
S = 0, **R= 1**: Q = 0

D Latch



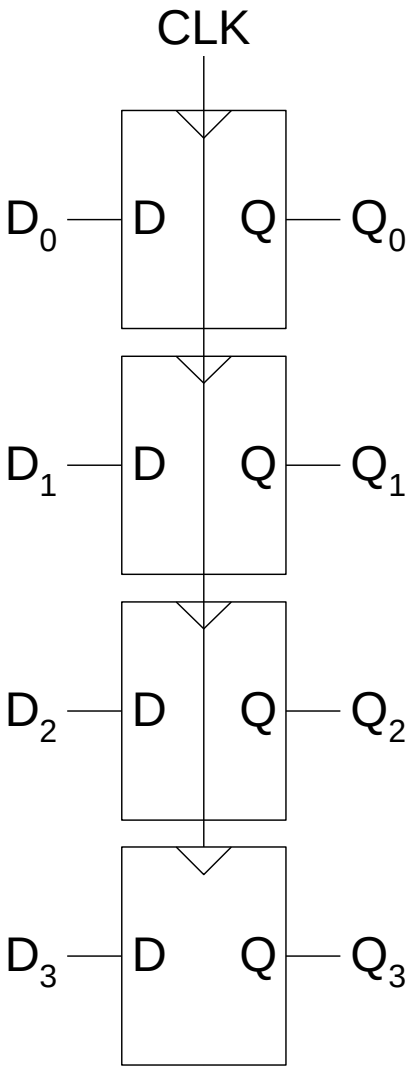
CLK = 1: Q = D
CLK = 0: $Q = Q_{prev}$

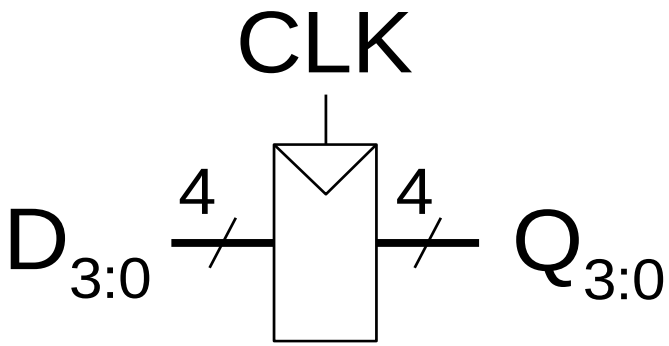
D Flip-flop



CLK = 0→1: Q = D
Otherwise: $Q = Q_{prev}$

Registers

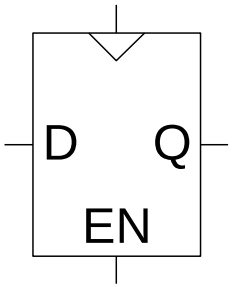




Enabled Flip-Flops

- **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

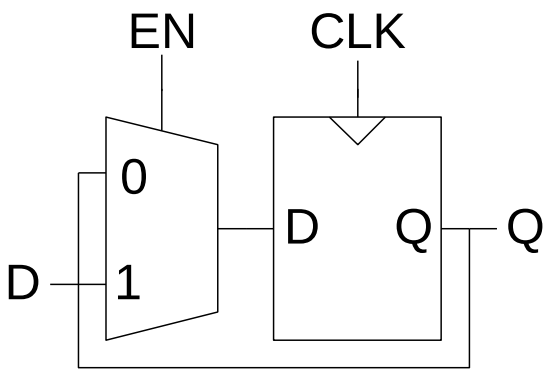
Symbol



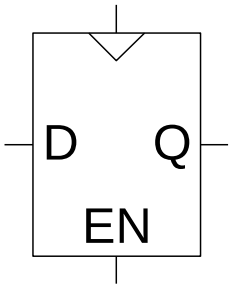
Enabled Flip-Flops

- **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



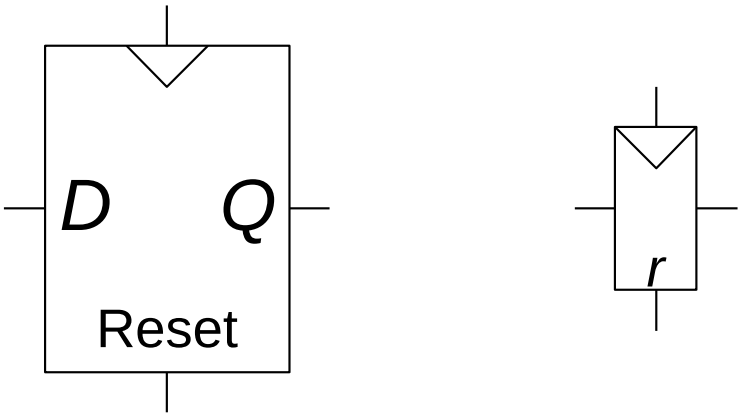
Symbol



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

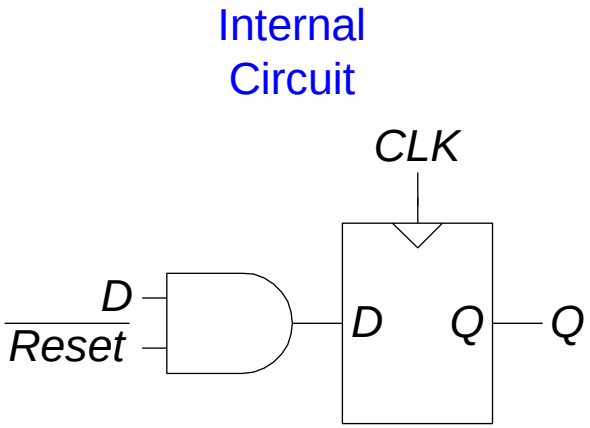


Resettable Flip-Flops

- 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

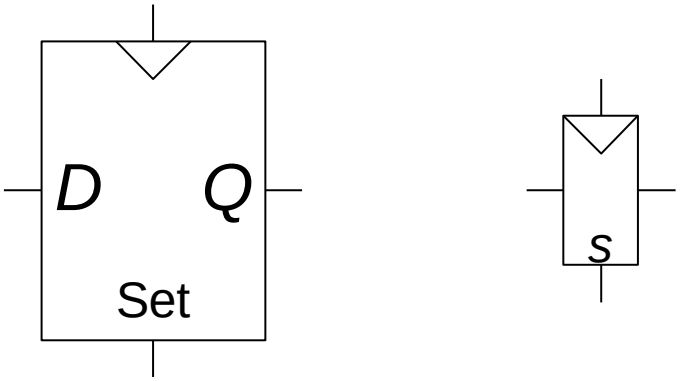
- Two types:
  - **Synchronous:** resets at the clock edge only
  - **Asynchronous:** resets immediately when *Reset* = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?



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



Synchronous Sequential Logic Design

- Registers **inserted between combinational logic**
- Registers contain **state** of the system
- State changes at clock edge: system **synchronized** to the clock

Synchronous Sequential Logic Design

- **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

Synchronous Sequential Logic Design

- **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