

ডিজিটাল ইলেক্ট্রনিক্স-১ (২৬৮৩১) অধ্যায়ঃ-১

মোঃ শাখাওয়াত হোসেন
ইন্সট্রাকটর (ইলেকট্রিক্যাল)
ফেনী পলিটেকনিক ইন্সটিটিউট, ফেনী

আলোচ্য বিষয়সমূহঃ-

- ❖ ডিজিটাল ইলেকট্রনিক্স
- ❖ লিজিক গেইট

ডিজিটাল ইলেকট্রনিক্স

- ডিজিটাল (Digital) শব্দের অর্থ যে সংকেত বা প্রতীকের মাত্র দুটি নির্ধারিত স্তর থাকে(1,0)।
- ইলেকট্রনিক্স এর যে শাখায় digital সংখ্যা, সংকেত, বিভিন্ন digital সার্কিট এর ঠন,কার্যপ্রণালী, ডিজাইন সম্পর্কে বিস্তারিত বিশ্লেষণ ও আলোচনা করা হয় তাকে ডিজিটাল ইলেকট্রনিক্স বলে।

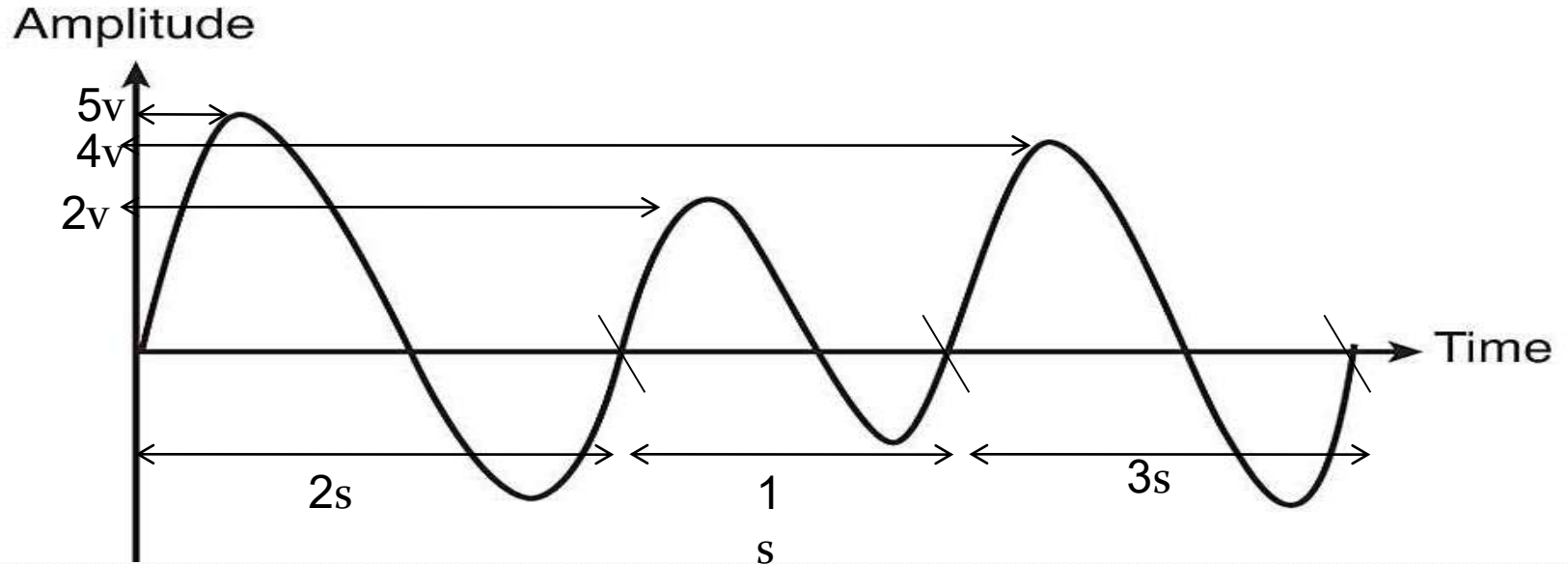
ডিজিটাল ইলেকট্রনিক্সের

প্রয়োগক্ষেত্রঃ-

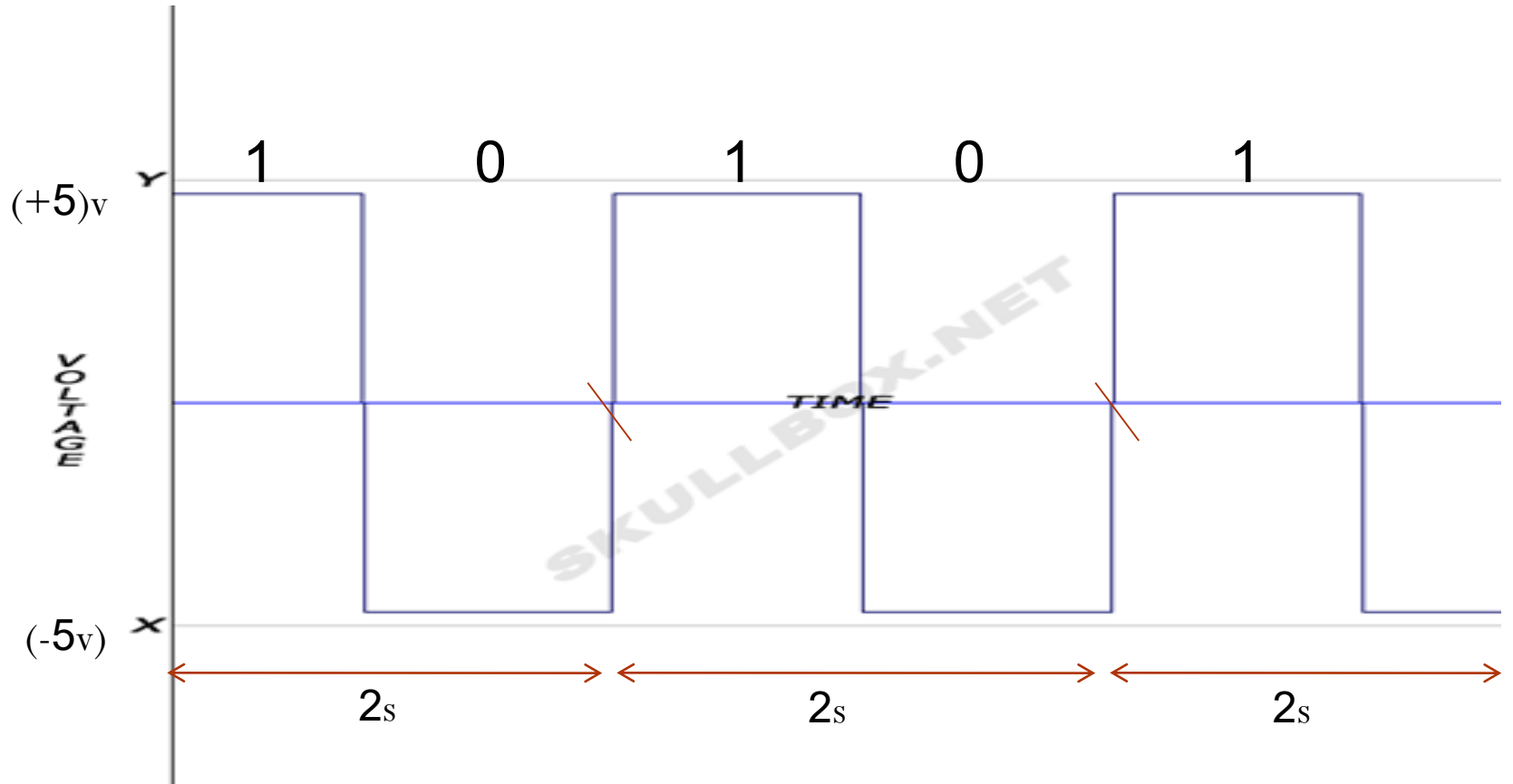
- ডিজিটাল কমপিউটার
- মোবাইল
- ডিজিটাল ঘড়ি
- ক্যালকুলেটর
- ডিজিটাল টেলিভিশন
- এনকোডার
- ডিকোডার
- মাল্টিপ্লেক্সার
- ডি-মাল্টিপ্লেক্সার
- ফ্লিপ-ফ্লপ, রেজিস্টার, কাউন্টার ইত্যাদি।

এনালগ ও ডিজিটাল সিগন্যাল

- এনালগ সিগন্যালঃ- যেসব সিগন্যাল এক এক সময়ে এক এক রকম মানের ফুল সাইকেল সম্পূর্ণ করে সঞ্চালিত হয় তাকে এনালগ সিগন্যাল বলে।



❖ ডিজিটাল সিগন্যালঃ- যেসব সিগন্যাল একাট
নির্দিষ্ট সময় পর পর একই রকম মানের ফুল
সাইকেল সম্পূর্ণ করে সঞ্চালিত হয় তাকে ডিজিটাল
সিগন্যাল বলে।



ধন্যবাদ

ডিজিটাল ইলেক্ট্রনিক্স-১

(২৬৮৩১)

অধ্যায়ঃ- ৩

মোঃ শাখাওয়াত হোসেন
ইন্সট্রাকটর (ইলেকট্রিক্যাল)
ফেনী পলিটেকনিক ইন্সটিটিউট, ফেনী

আলোচ্য বিষয়সমূহঃ-

- ❖ লিডিক গেইটঃ-
- ▶ মৌলিক গেইট

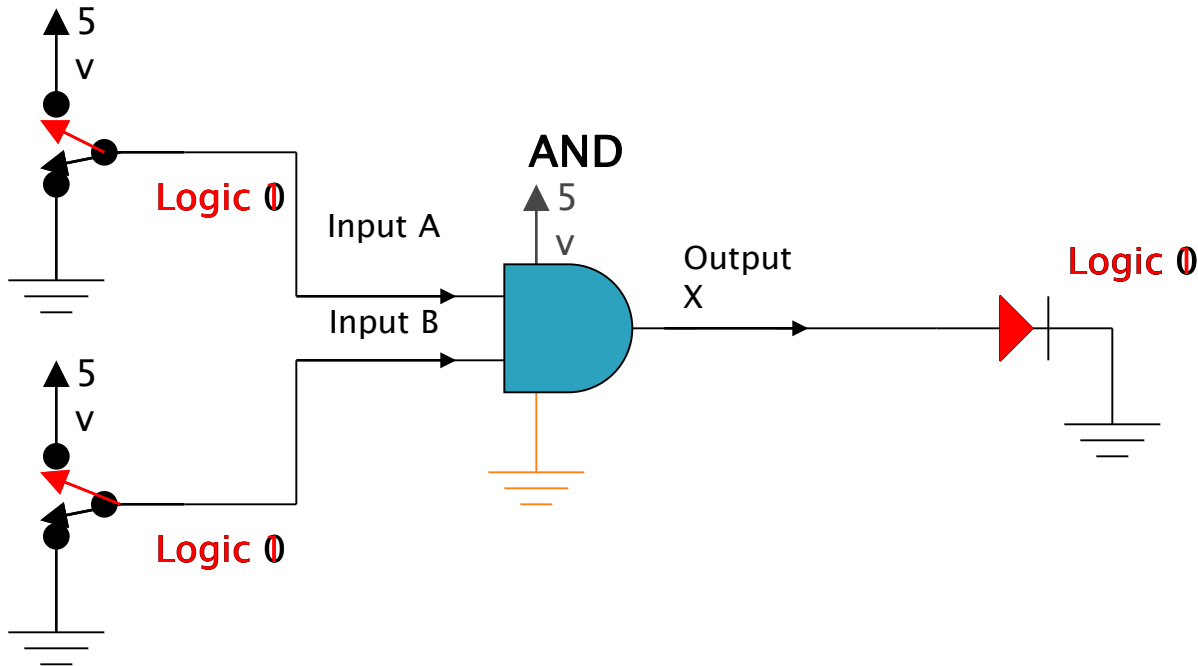
The AND Gate:



The AND is the last of the remaining fundamental logic gates. You will learn its behaviour using a Truth Table analysis and an animation.

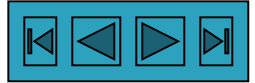
Truth Table. The table shows that the AND gate responds with a high at the output if the signal applied to the input A and B are both high.

Animation. In order to see how it works, the gate has been connected to 2 switches and LED. Continue to see the system in action...



| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

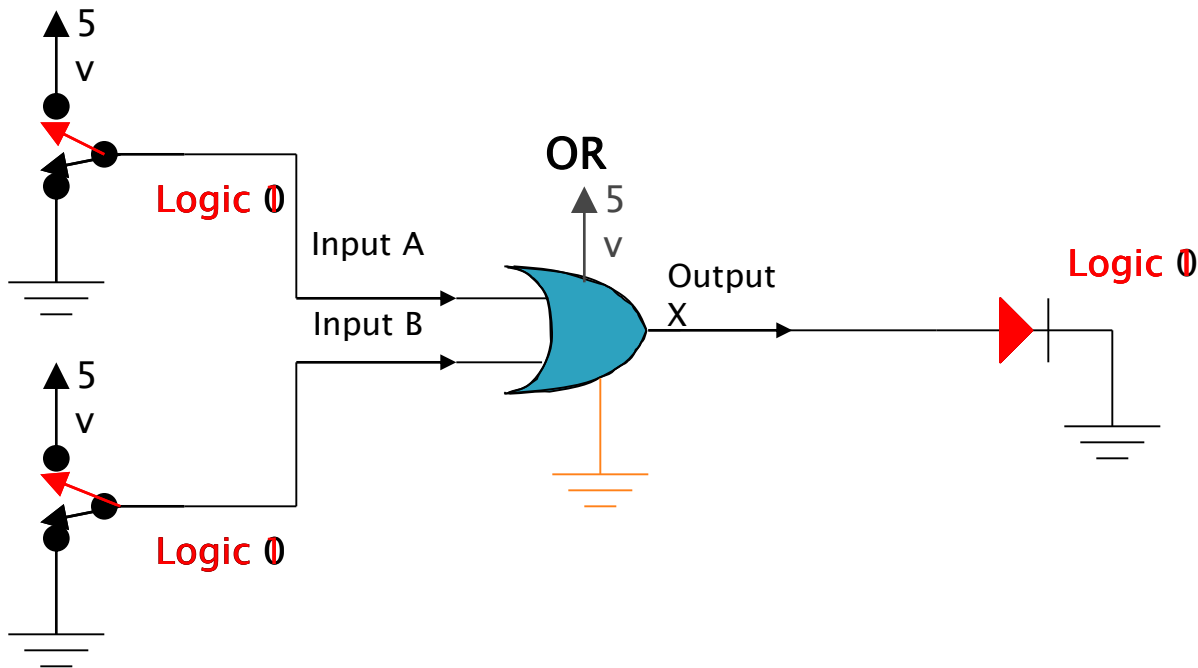
The OR Gate:



The OR gate is the second of three fundamental logic gates. You will learn its behaviour using a Truth Table analysis and an animation.

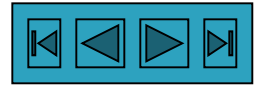
Truth Table The table shows that the OR gate responds with a high at the output if the signal applied to the input A or B is high.

Animation In order to see how it works, the gate has been connected to 2 switches and LED. Continue to see the system in action...



| A | B | X |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

The NOT Gate (inverter):

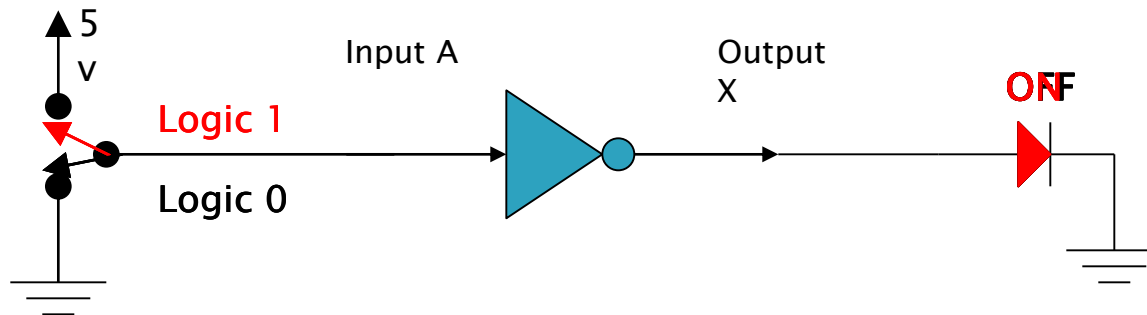


The NOT gate is the first of the three fundamental logic gates. You will learn its operation using Truth Table analysis and an animation.

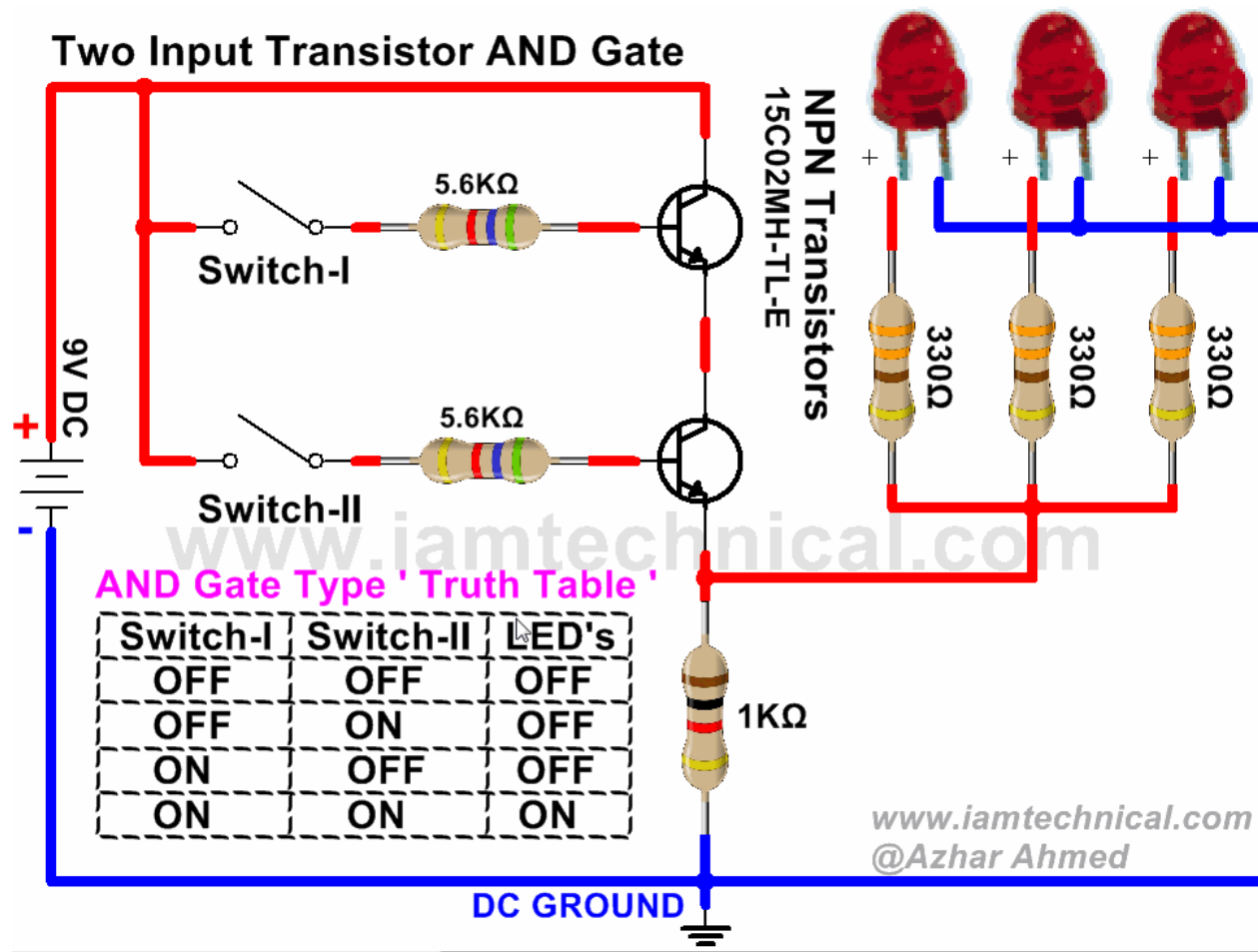
Truth Table Is a chart that lists the input condition on the left and the gate's output response on the right. The table shows that the NOT gate responds at the output with the inverse of the signal applied to the input.

| A | X |
|---|---|
| 0 | 1 |
| 1 | 0 |

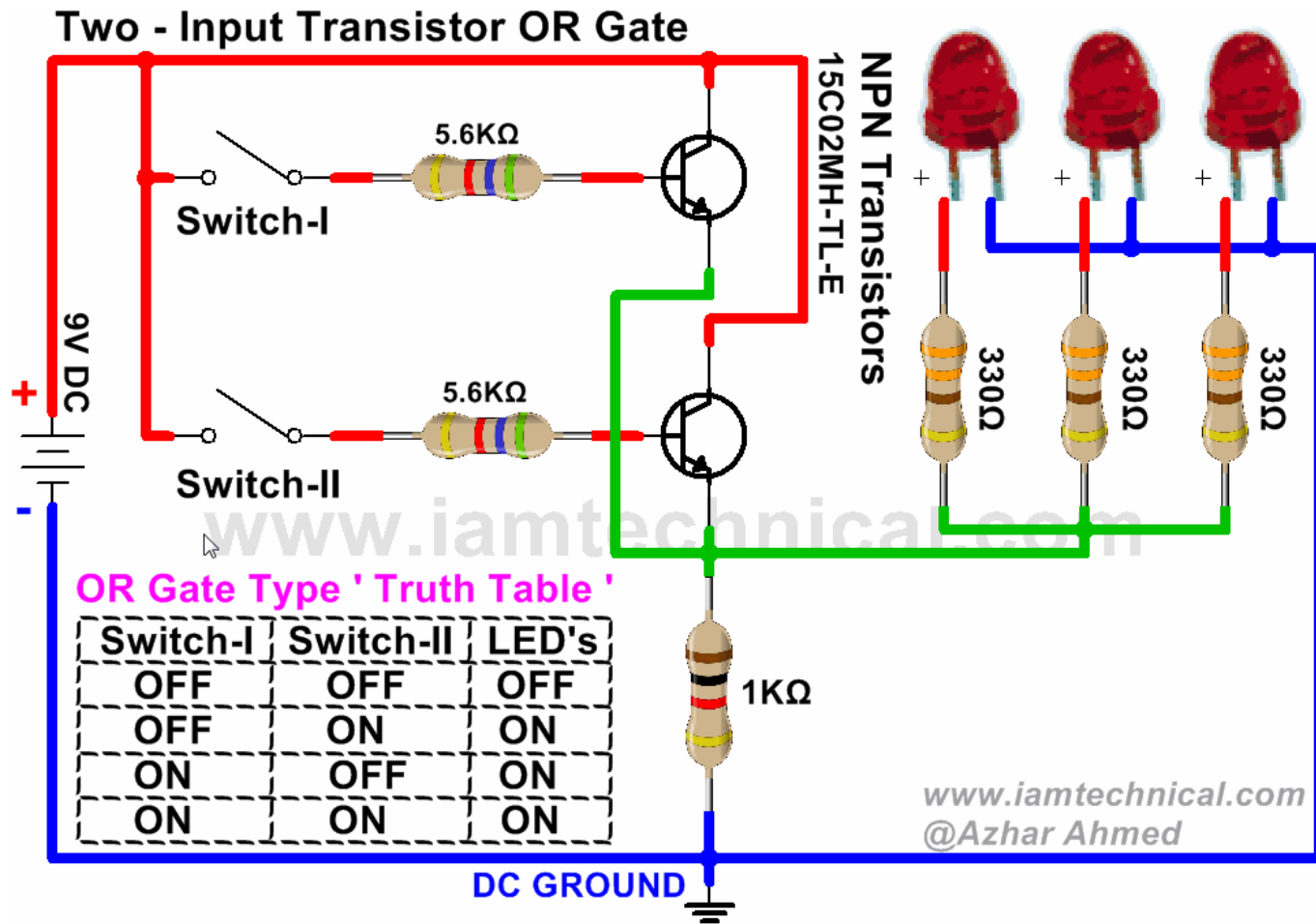
Animation. In order to see how it works, the gate has been connected to a switch and LED. Continue to see the system in action...



AND Gate

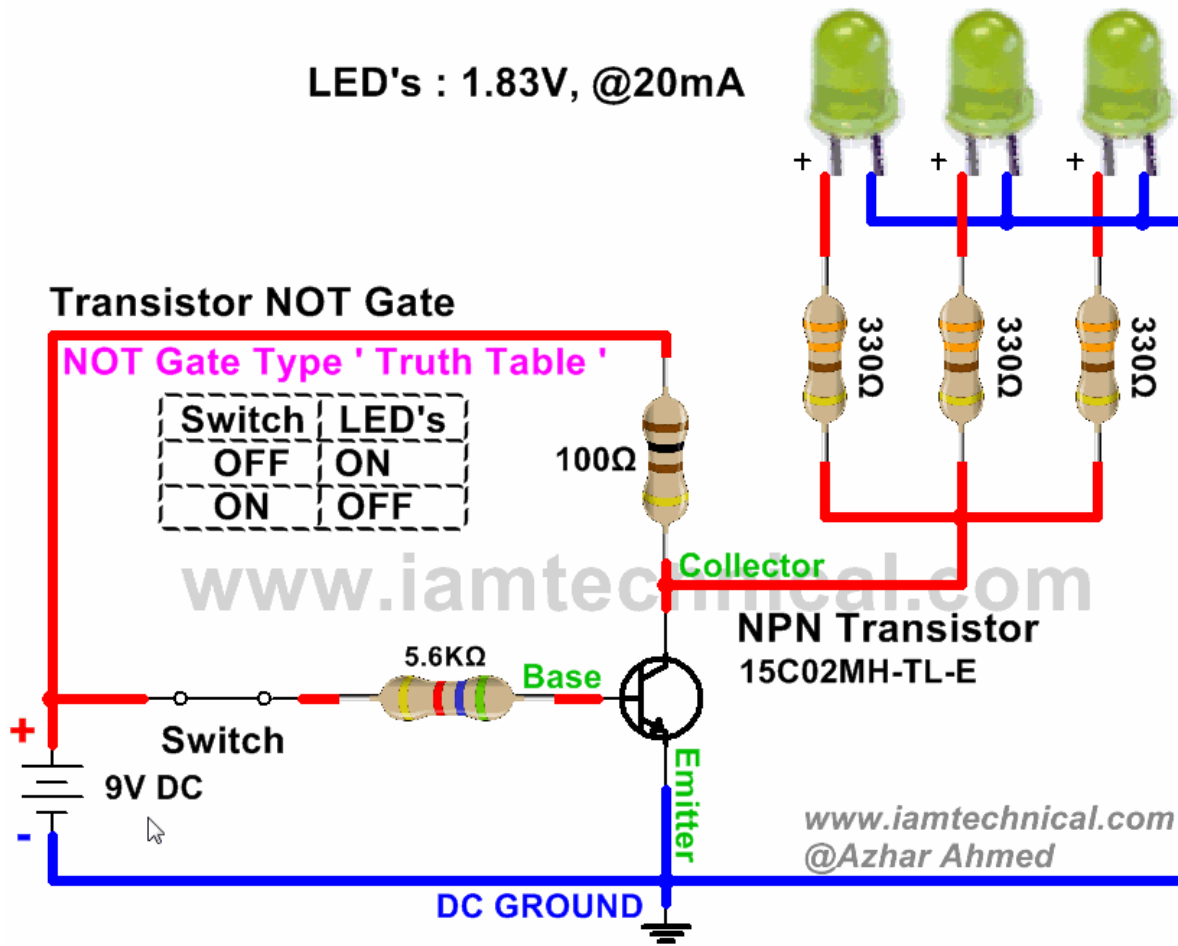


OR Gate



NOT Gate

LED's : 1.83V, @20mA



ধন্যবাদ



Md. Shakawat Hossain
Instructor (Electrical)

Digital Electronics :-1(26831)

Chapter-3



Universal Gate – NAND&NOR

Digital Electronics

Universal Gate – NAND

This presentation will demonstrate

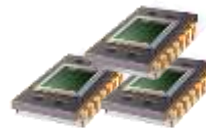
- The basic function of the **NAND** gate.
- How a **NAND** gate can be used to replace an **AND** gate, an **OR** gate, or an **INVERTER** gate.
- How a logic circuit implemented with **AOI** logic gates can be re-implemented using only **NAND** gates.
- That using a single gate type, in this case **NAND**, will reduce the number of integrated circuits (IC) required to implement a logic circuit.

AOI Logic



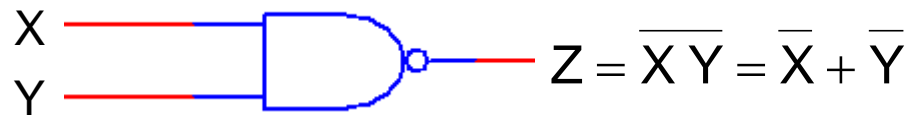
More ICs = More \$\$

NAND Logic



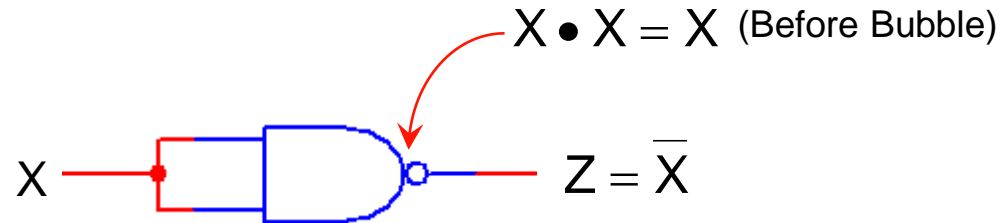
Less ICs = Less \$\$

NAND Gate



| X | Y | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

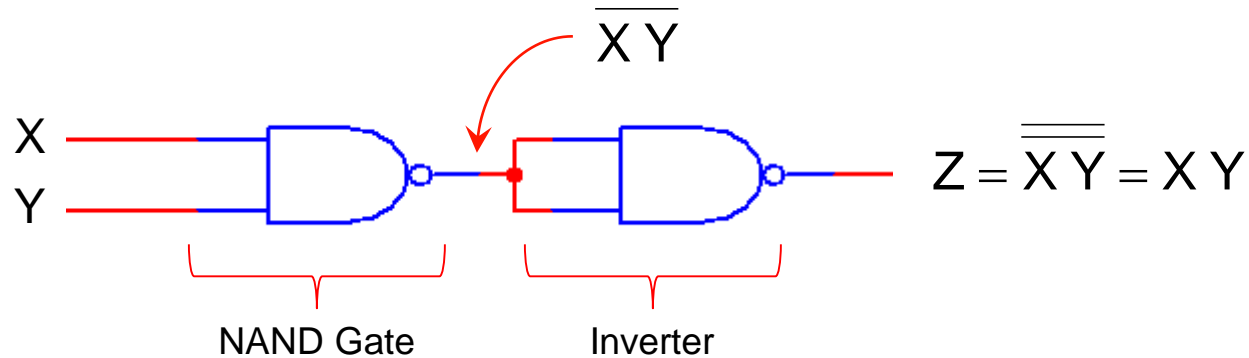
NAND Gate as an Inverter Gate



| X | Z |
|---|---|
| 0 | 1 |
| 1 | 0 |

Equivalent to Inverter

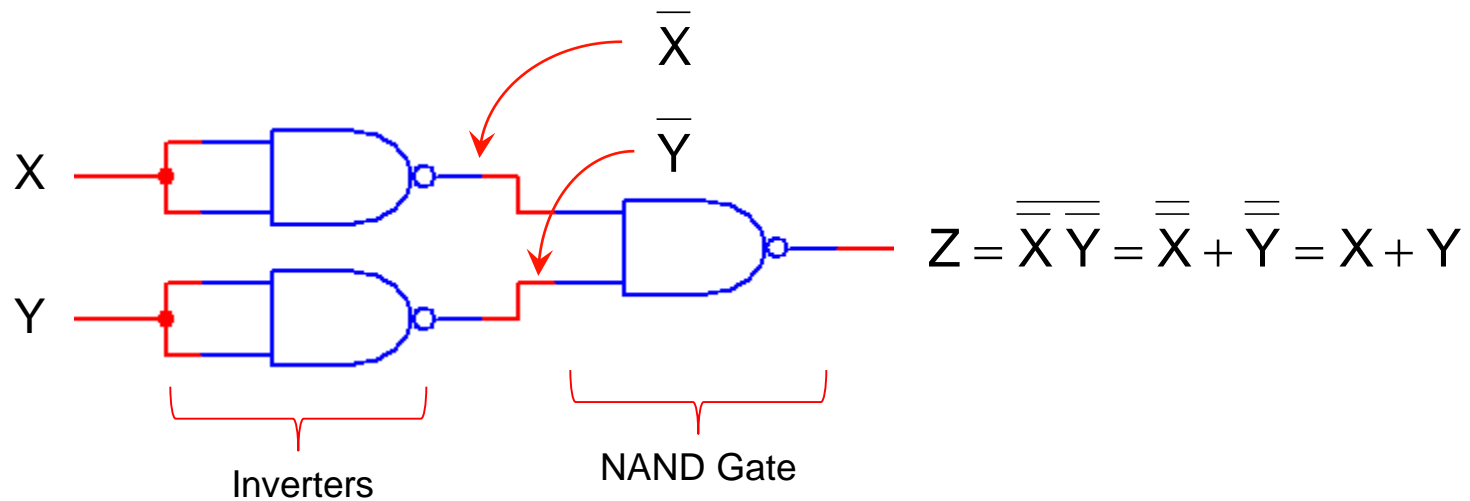
NAND Gate as an AND Gate



| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Equivalent to AND Gate

NAND Gate as an OR Gate

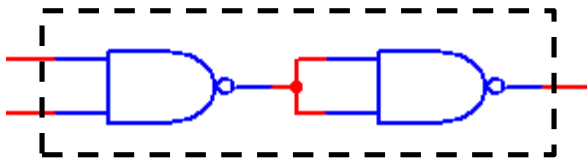
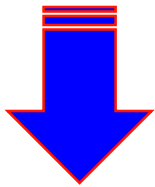


| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

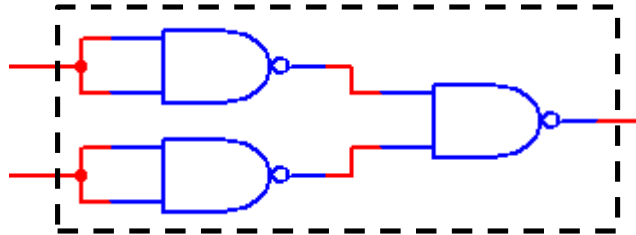
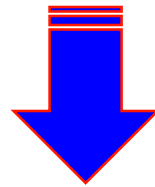
Equivalent to OR Gate

NAND Gate Equivalent to AOI Gates

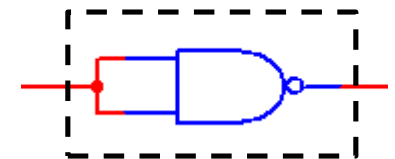
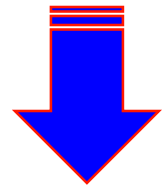
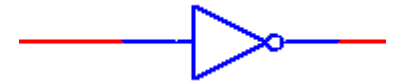
AND



OR



INVERTER



Process for NAND Implementation

1. If starting from a logic expression, implement the design with AOI logic.
2. In the AOI implementation, identify and replace every AND, OR, and INVERTER gate with its NAND equivalent.
3. Redraw the circuit.
4. Identify and eliminate any double inversions (i.e., back-to-back inverters).
5. Redraw the final circuit.

Universal Gate – NOR

This presentation will demonstrate...

- The basic function of the **NOR** gate.
- How an **NOR** gate can be used to replace an **AND** gate, an **OR** gate or an **INVERTER** gate.
- How a logic circuit implemented with **AOI** logic gates could be re-implemented using only **NOR** gates
- That using a single gate type, in this case **NOR**, will reduce the number of integrated circuits (IC) required to implement a logic circuit.

AOI Logic



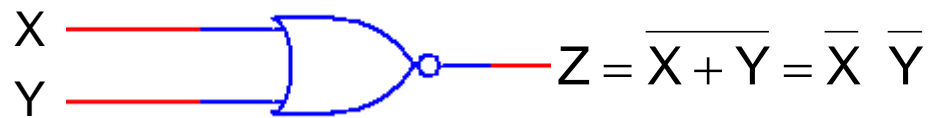
More ICs = More \$\$

NOR Logic



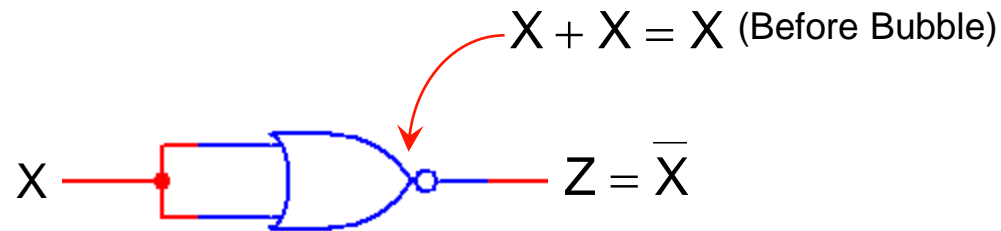
Less ICs = Less \$\$

NOR Gate



| X | Y | Z |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

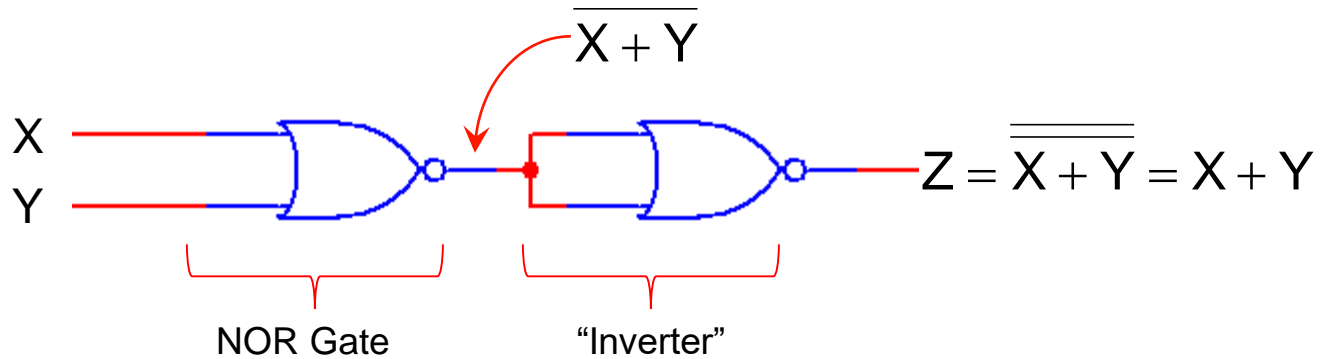
NOR Gate as an Inverter Gate



| X | Z |
|----------|----------|
| 0 | 1 |
| 1 | 0 |

Equivalent to Inverter

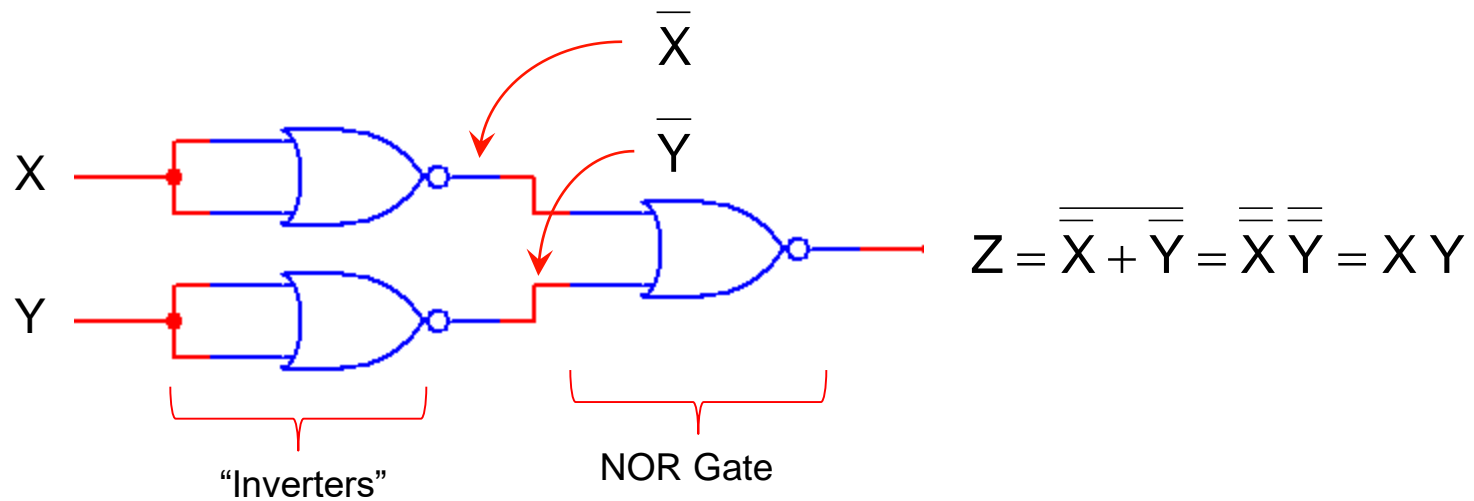
NOR Gate as an OR Gate



| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Equivalent to OR Gate

NOR Gate as an AND Gate

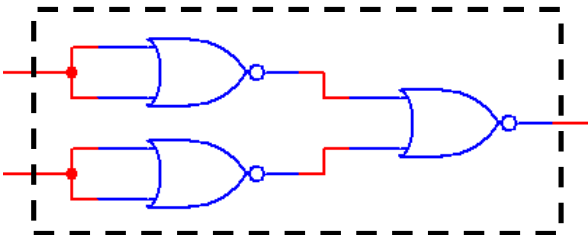
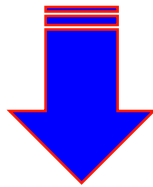


| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

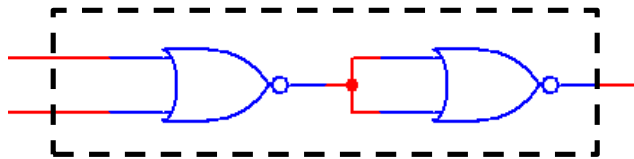
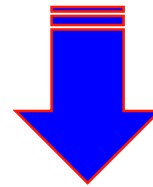
Equivalent to AND Gate

NOR Gate Equivalent of AOI Gates

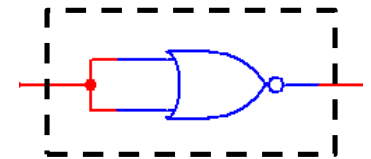
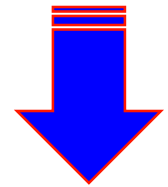
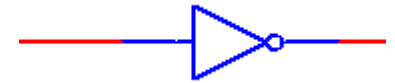
AND



OR



INVERTER



Process for NOR Implementation

1. If starting from a logic expression, implement the design with AOI logic.
2. In the AOI implementation, identify and replace every AND, OR, and INVERTER gate with its NOR equivalent.
3. Redraw the circuit.
4. Identify and eliminate any double inversions. (i.e. back-to-back inverters)
5. Redraw the final circuit.

THANK YOU

wellcome

Md. Shakawat Hossain

Instructor (Electrical)

Digital Electronics-1 (26831)

Chapter-6

COMBINATIONAL LOGIC

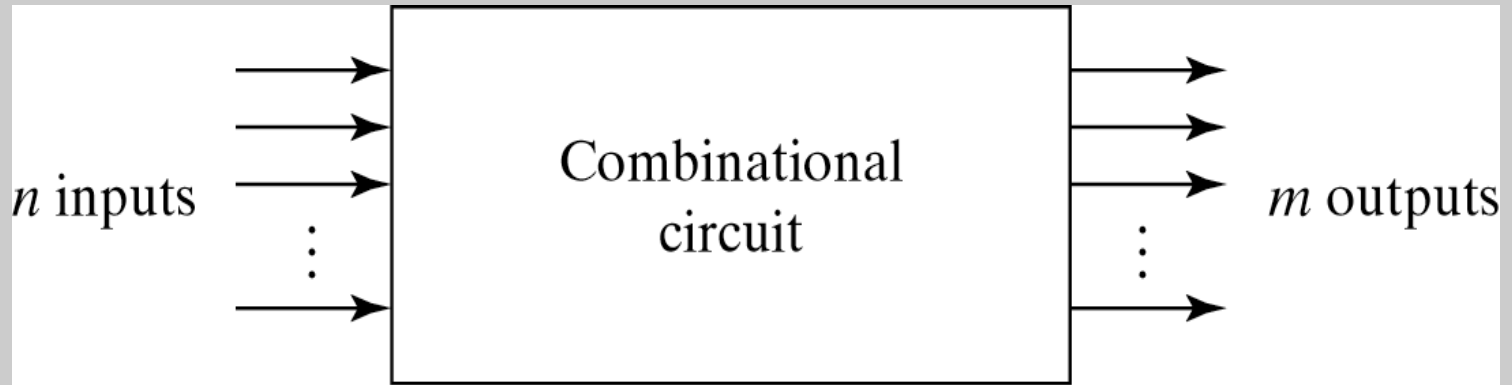


Fig. 4-1 Block Diagram of Combinational Circuit

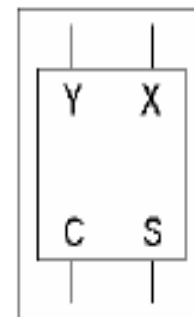
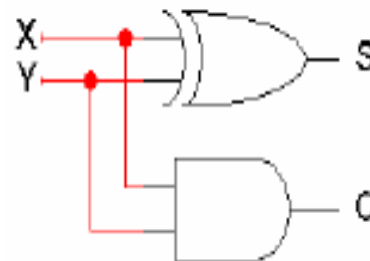
HALF ADDER

Adding two bits

- We'll make a hardware adder based on our human addition algorithm.
- We start with a **half adder**, which adds two bits X and Y and produces a two-bit result: a **sum** S (the right bit) and a **carry out** C (the left bit).
- Here are truth tables, equations, circuit and block symbol.

| X | Y | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

$$C = XY$$
$$S = X'Y + XY'$$
$$= X \oplus Y$$



FULL ADDER

Adding three bits

- But what we really need to do is add *three* bits: the augend and addend bits, *and* the carry in from the right.
- A **full adder** circuit takes three inputs X , Y and C_{in} , and produces a two-bit output consisting of a sum S and a carry out C_{out} .

$$\begin{array}{r} 1 \quad 1 \quad 1 \quad 0 \\ \quad 1 \quad 0 \quad 1 \quad 1 \\ + \quad 1 \quad 1 \quad 1 \quad 0 \\ \hline 1 \quad 1 \quad 0 \quad 0 \quad 1 \end{array}$$

| X | Y | C_{in} | C_{out} | S |
|---|---|----------|-----------|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Full adder equations

- Using Boolean algebra, we can simplify S and C_{out} as shown here.

| X | Y | C_{in} | C_{out} | S |
|---|---|----------|-----------|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

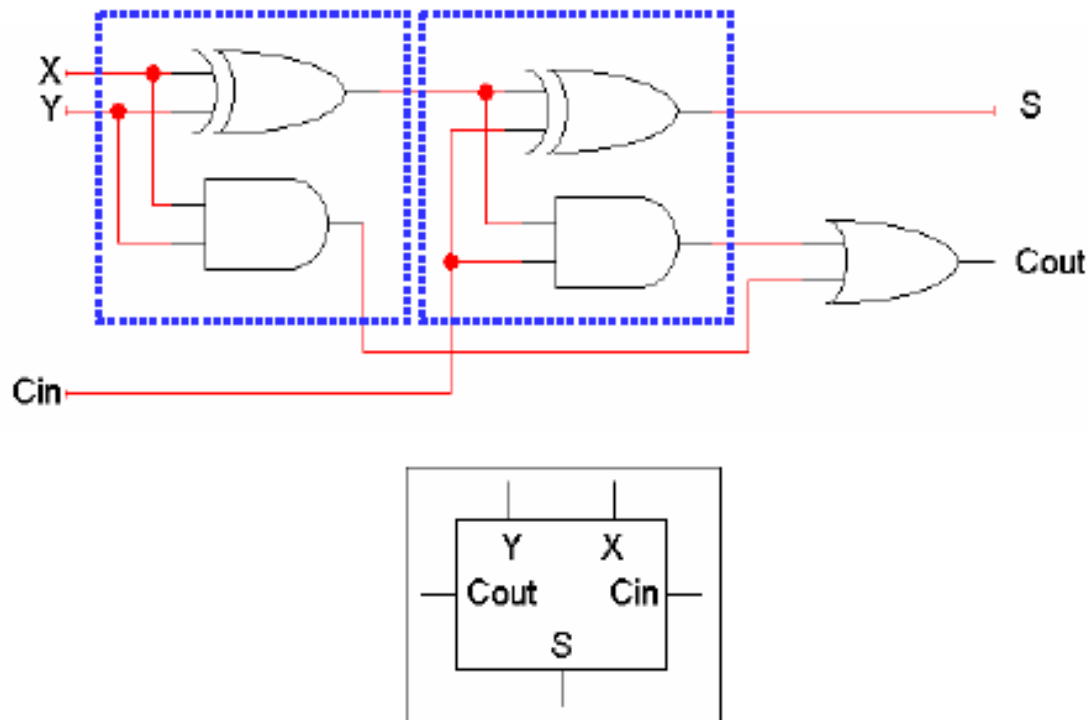
$$\begin{aligned}S &= \Sigma m(1,2,4,7) \\&= X'Y'C_{in} + X'YC_{in}' + XY'C_{in}' + XYC_{in} \\&= X'(Y'C_{in} + YC_{in}') + X(Y'C_{in}' + YC_{in}) \\&= X'(Y \oplus C_{in}) + X(Y \oplus C_{in})' \\&= X \oplus Y \oplus C_{in}\end{aligned}$$

$$\begin{aligned}C_{out} &= \Sigma m(3,5,6,7) \\&= X'YC_{in} + XY'C_{in} + XYC_{in}' + XYC_{in} \\&= (X'Y + XY')C_{in} + XY(C_{in}' + C_{in}) \\&= (X \oplus Y)C_{in} + XY\end{aligned}$$

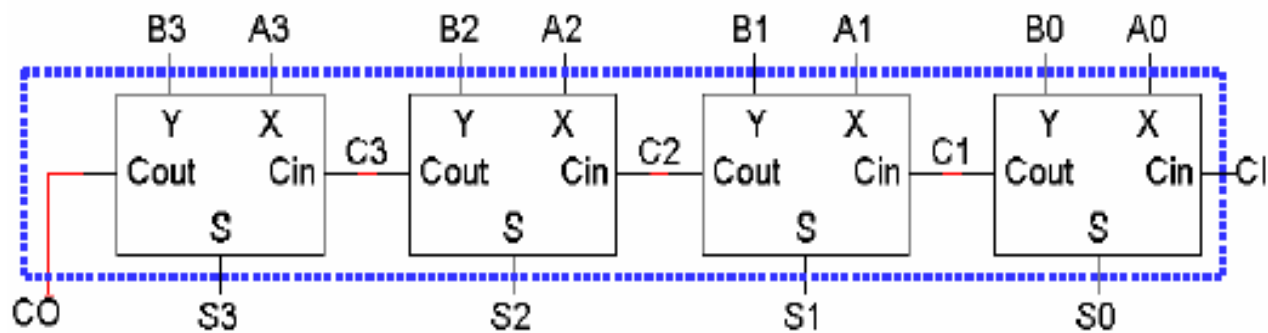
Full adder circuit

- We write the equations this way to highlight the hierarchical nature of adder circuits—you can build a full adder by combining two half adders!

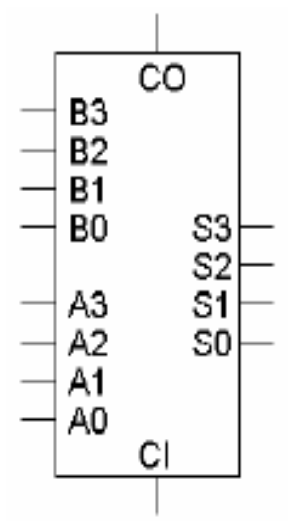
$$S = X \oplus Y \oplus C_{in}$$
$$C_{out} = (X \oplus Y) C_{in} + XY$$



A four-bit adder

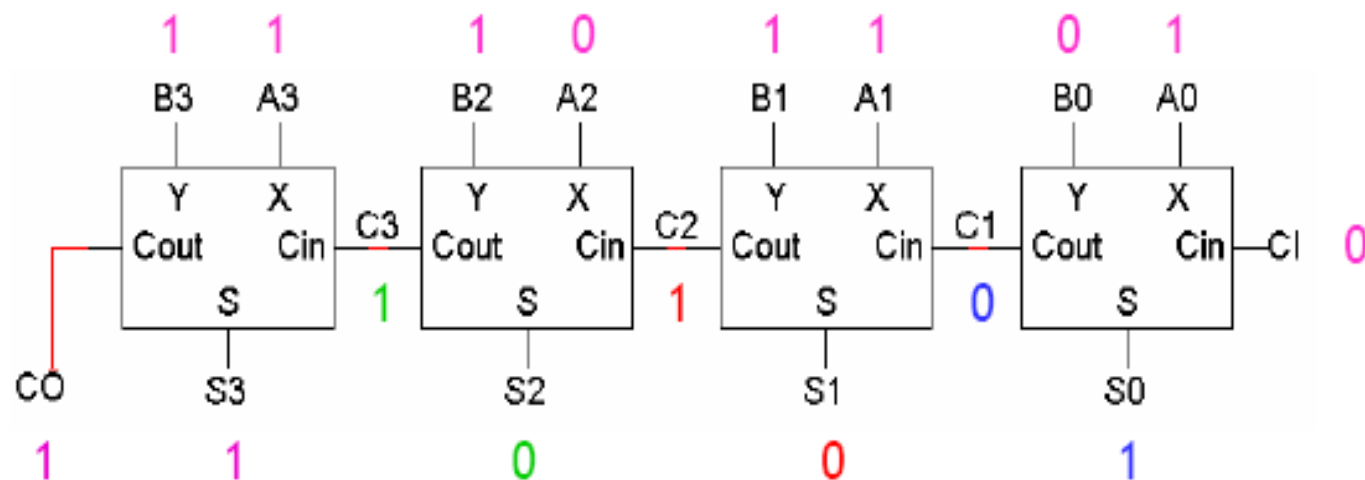


- Similarly, we can cascade four full adders to build a four-bit adder.
 - The inputs are two four-bit numbers ($A_3A_2A_1A_0$ and $B_3B_2B_1B_0$) and a carry in C_i .
 - The two outputs are a four-bit sum $S_3S_2S_1S_0$ and the carry out C_o .
- If you designed this adder without taking advantage of the hierarchical structure, you'd end up with a 512-row truth table with five outputs!



An example of 4-bit addition

- Let's put our initial example into this circuit, with $A=1011$ and $B=1110$.



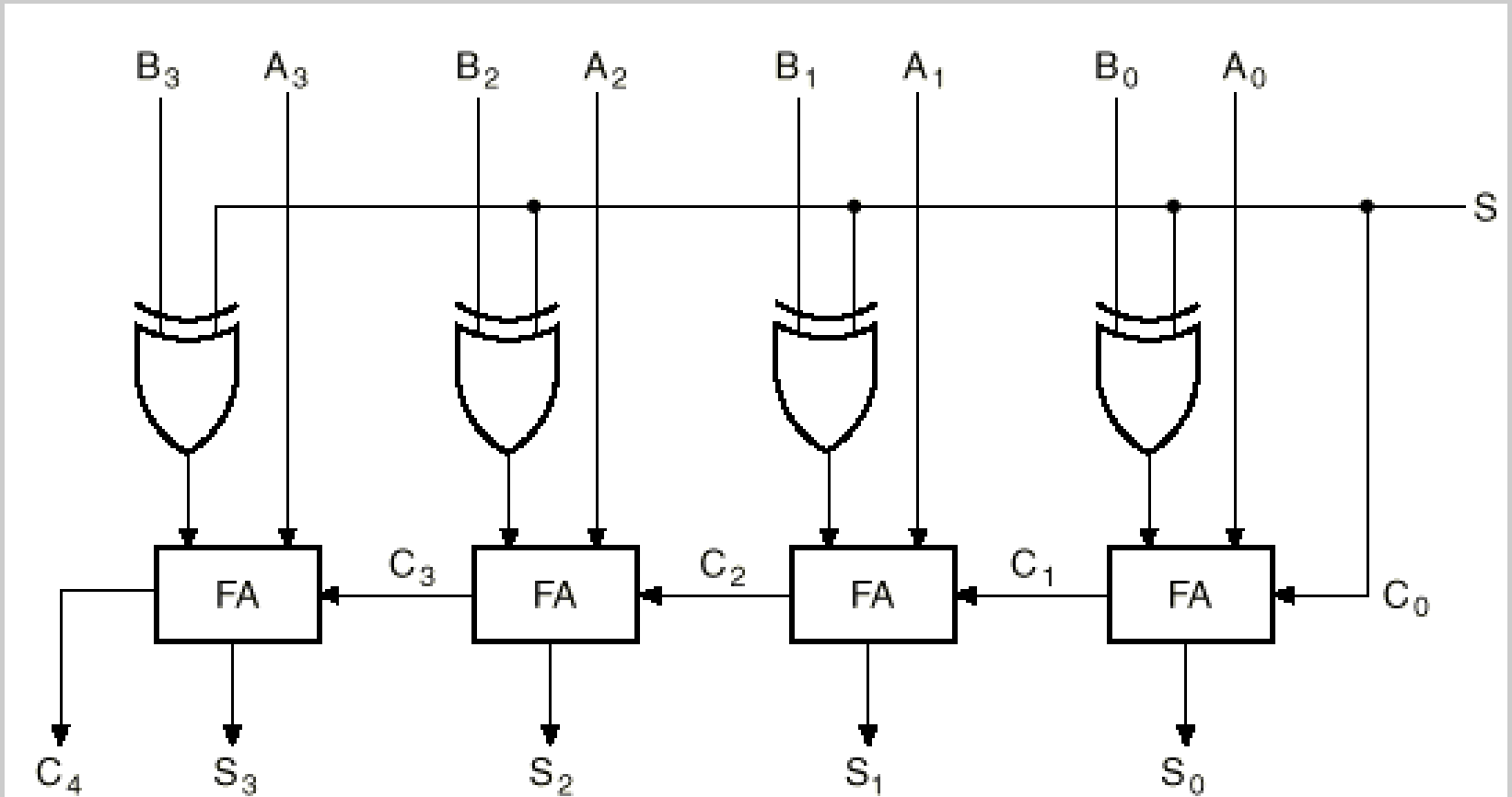
- Fill in all the inputs, including $C_i=0$
- The circuit produces C_1 and S_0 ($1 + 0 + 0 = 01$)
- Use C_1 to find C_2 and S_1 ($1 + 1 + 0 = 10$)
- Use C_2 to compute C_3 and S_2 ($0 + 1 + 1 = 10$)
- Use C_3 to compute C_0 and S_3 ($1 + 1 + 1 = 11$)

Binary Adder/Subtractors

- The subtraction $A-B$ can be performed by taking the 2's complement of B and adding to A .
- The 2's complement of B can be obtained by complementing B and adding one to the result.

$$\begin{aligned}A-B &= A + 2C(B) \\ &= A + 1C(B) + 1 \\ &= A + B' + 1\end{aligned}$$

4-bit Binary Adder/Subtractor

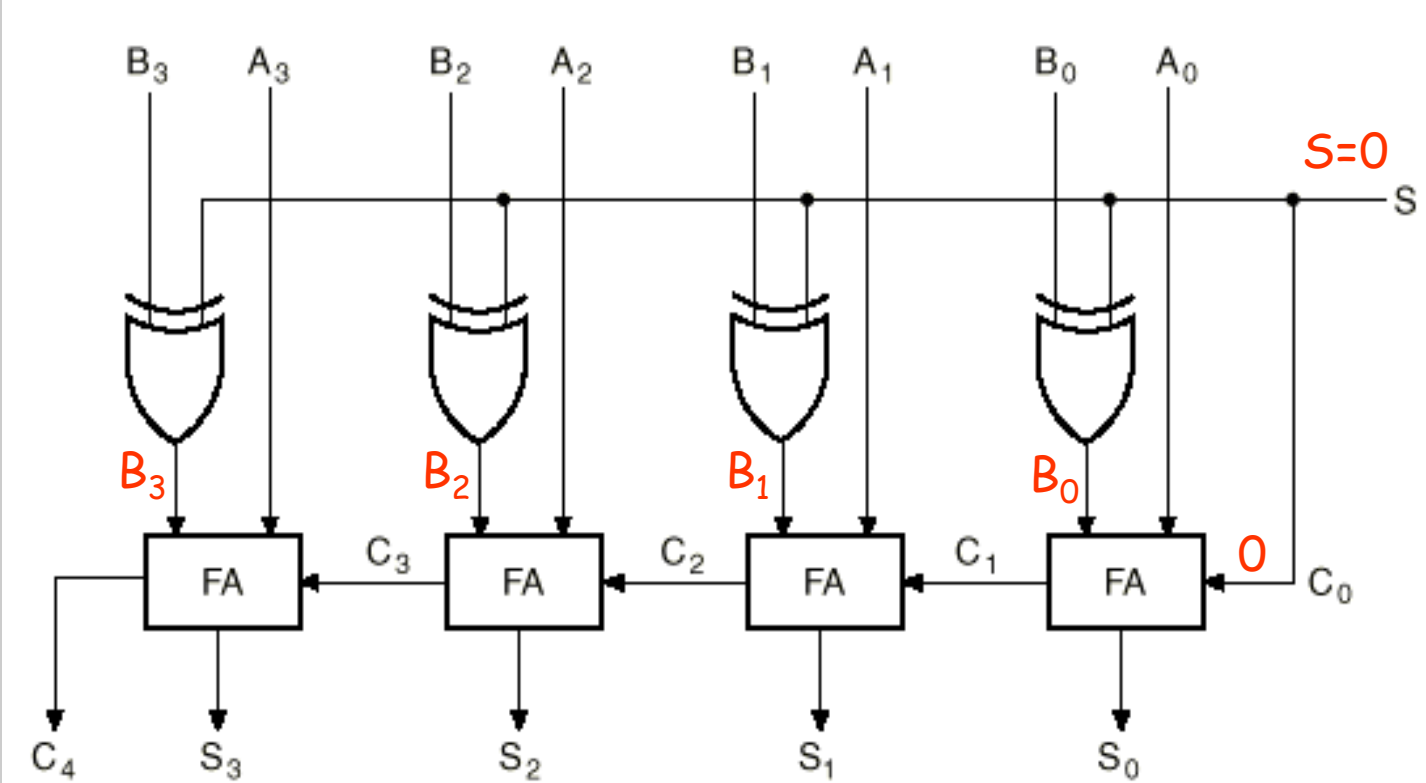


-XOR gates act as programmable inverters

4-bit Binary Adder/Subtractor (cont.)

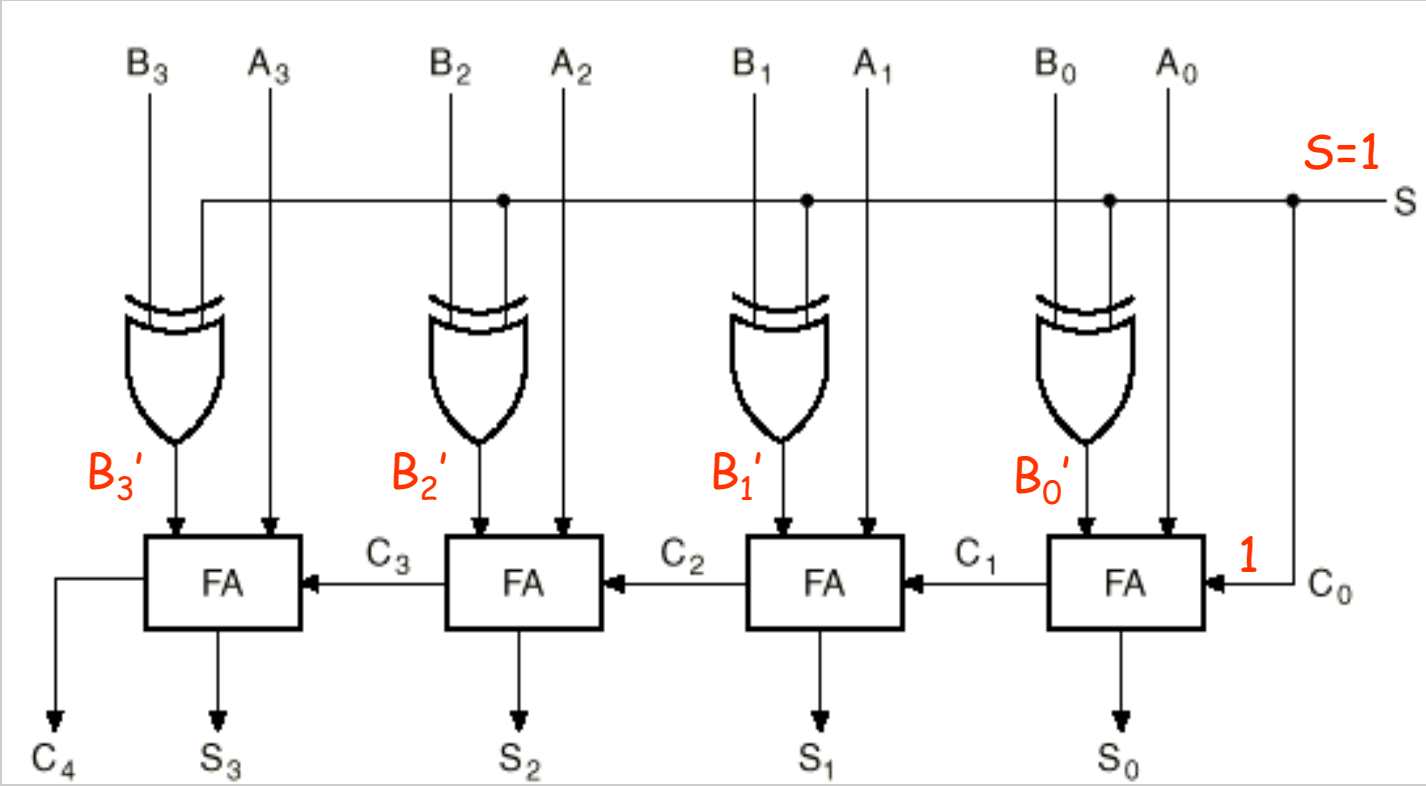
- When $S=0$, the circuit performs $A + B$. The carry in is 0, and the XOR gates simply pass B untouched.
- When $S=1$, the carry into the least significant bit (LSB) is 1, and B is complemented (1's complement) prior to the addition; hence, the circuit adds to A the 1's complement of B plus 1 (from the carry into the LSB).

4-bit Binary Adder/Subtractor (cont.)



$S=0$ selects addition

4-bit Binary Adder/Subtractor (cont.)



$S=1$ selects subtraction

THANK YOU

Flip-Flops

lecture notes by
Md.Shakawat Hossain
Instructor (Electrical)

Digital Electronics-1 (26831)
Chapter-10

Objectives of Lecture

The objectives of this lecture are:

- to discuss the difference between **combinational** and **sequential** logic as well as the difference between **asynchronous** and **synchronous** circuits and to show why the operation of synchronous circuits is more predictable, given propagation delays.
- to explain the operation of the common **latches** and **flip-flops**
 - SR or set–reset latch, which may also be called a SR flip-flop
 - D or data flip-flop
 - T or toggle flip-flop
 - JK flip-flop
- to describe clocking and the differences between **positive edge** and **negative edge** triggering and discuss the type of control inputs — active high and active low; asynchronous, jam or direct.

Combinational Logic

- The outputs depend only on the state of the inputs all of the time. Any change in the state of one of the inputs will ripple through the circuit immediately.
 - Examples of combinational logic are NAND and NOR gates, Inverters, and Buffers. These four logic gates form the basis of almost all combinational logic circuits as well as **flip flops**.
- Circuits that change the state of the output in this manner are also known as **asynchronous** circuits.
 - However, not all asynchronous circuits are combinational logic circuits.

Sequential Logic

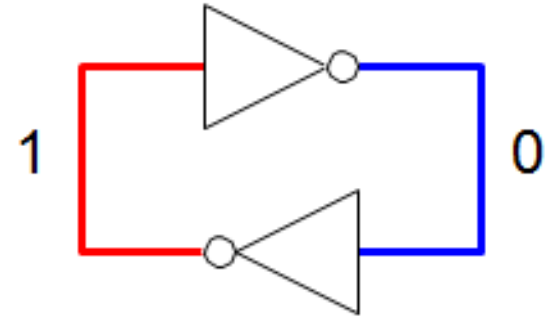
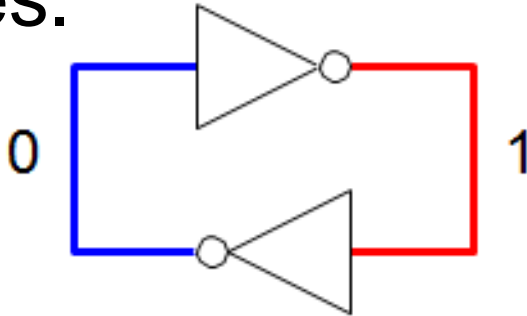
- Has **memory**; the circuit stores the result of the previous set of inputs. The current output depends on inputs **in the past** as well as present inputs.
 - The basic element in sequential logic is the **bistable latch** or **flip-flop**, which acts as a memory element for one bit of data.

Clocked Circuits

- Most flip-flops are **clocked** so that the output change state based upon the state of the inputs at precisely determined times.
 - Usage varies — in this course, ‘flip-flops’ will be used for clocked circuits and ‘latches’ for circuits that are asynchronous.
- A common clock used in many flip-flops in one circuit ensures that all parts of a digital system change state at the same time. This is called a **synchronous** system

Bistable Circuit

- At the heart of a bistable circuit is a pair of inverters connected in a loop — with **feedback**, in other words. It has two stable states.



- Without some control, there isn't a way to force the bistable circuit into one or the other state.

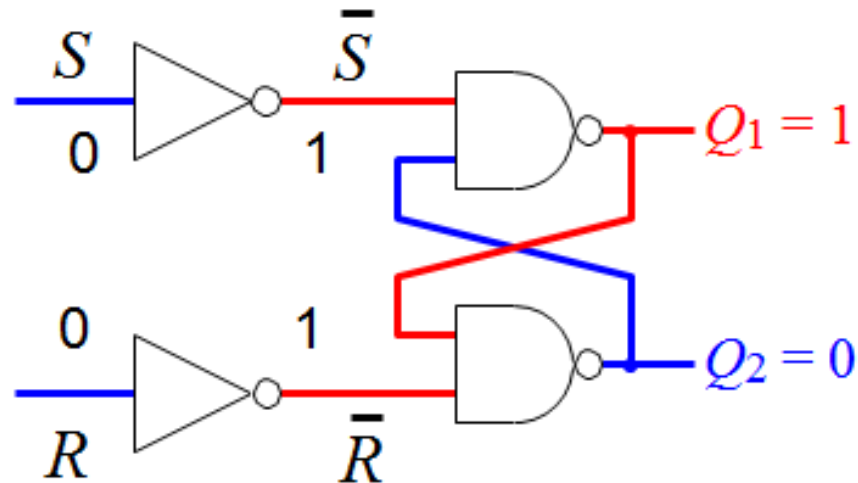
Bistable Circuits

- The bistable circuit is used as a ‘bus keeper’ to hold a node at a definite 1 or 0. It is also the heart of a ‘static random access memory’ (SRAM) cell.
 - Similar operation occurs for any ring composed of an even number of invertors.
 - What would happen if 3 inverters (or larger odd number) are connected in series?
 - This type of circuit is called a ring oscillator.
 - Check this out in the laboratory or in PSpice!

Core of a Flip-Flop:

The set–reset or *SR* Latch

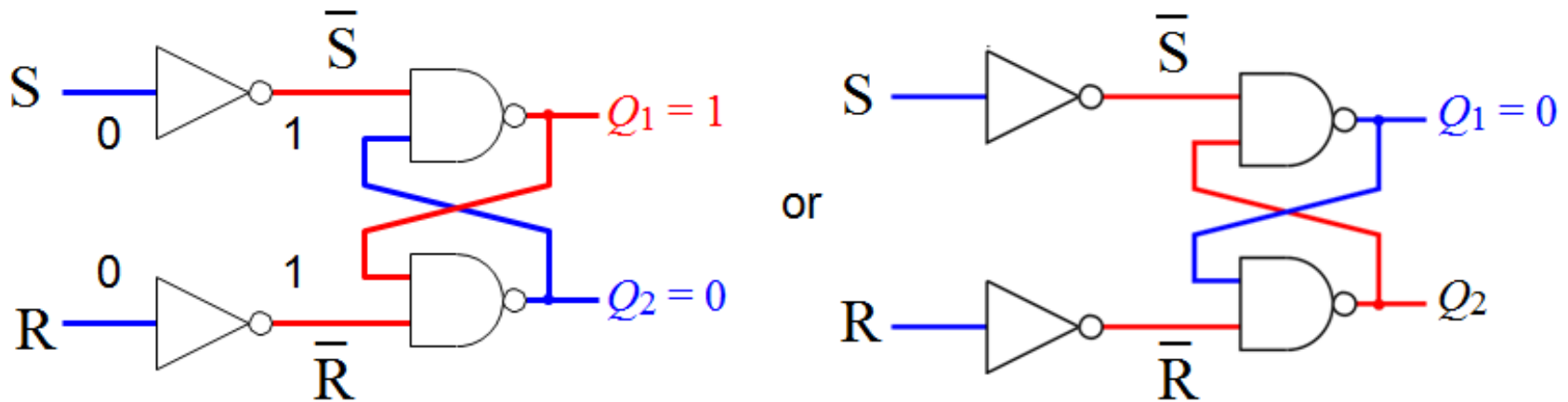
- Acts as a simple memory with **two stable states at the two output** when $S = R = 0$



- Q_1 and Q_2 are the outputs of the S-R latch.
- When Q_1 is known as Q and Q_2 is also called Q' or \bar{Q} (spoken as Q bar), meaning that its value is not Q or the opposite of Q .

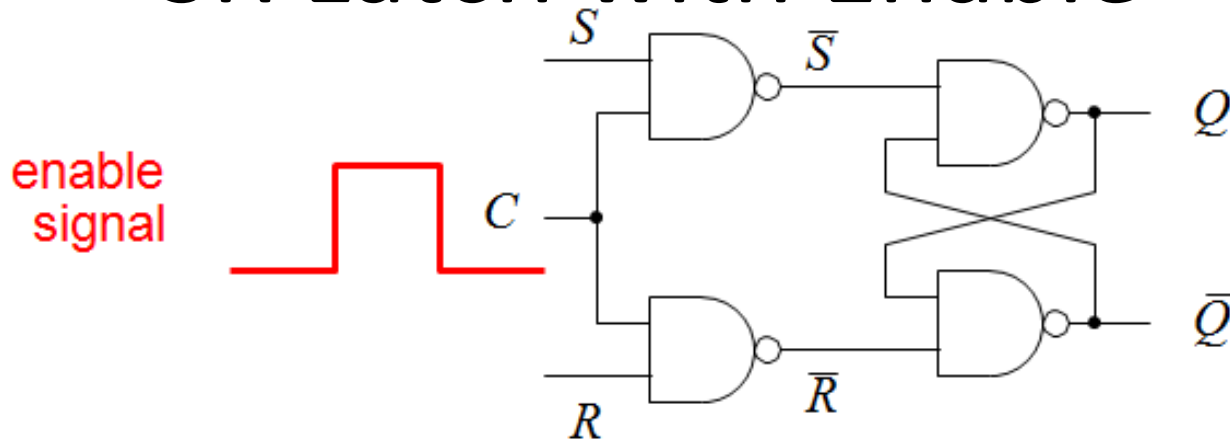
S-R Latch

Acts as a simple memory with **two stable states** when $S = R = 0$:



- The latch
 - **holds** (stores) when $S = R = 0$
 - is **set (to 1)** by bringing $S = 1$ with $R = 0$
 - is **reset (to 0)** or **cleared** by bringing $R = 1$ with $S = 0$
- The condition $S = R = 1$ must be avoided because it leads to an **indeterminate** condition, where the output can not be predicted at any one point in time. This can cause a **race** condition to occur when the inputs change to $S = R = 0$.

SR Latch with Enable



- The S and R inputs only effect the output states when the **enable** input C is high.
 - This controls when the latch responds to its inputs.
- The latch holds (stores) its value while the enable input is low — latches it!
 - Any changes in the inputs **during** the time when enable is high will affect the output immediately: the circuit is said to be **transparent**.
 - This circuit still has a major problem: the stored value is indeterminate if $S = R = 1$ when the clock goes low

Logic Table

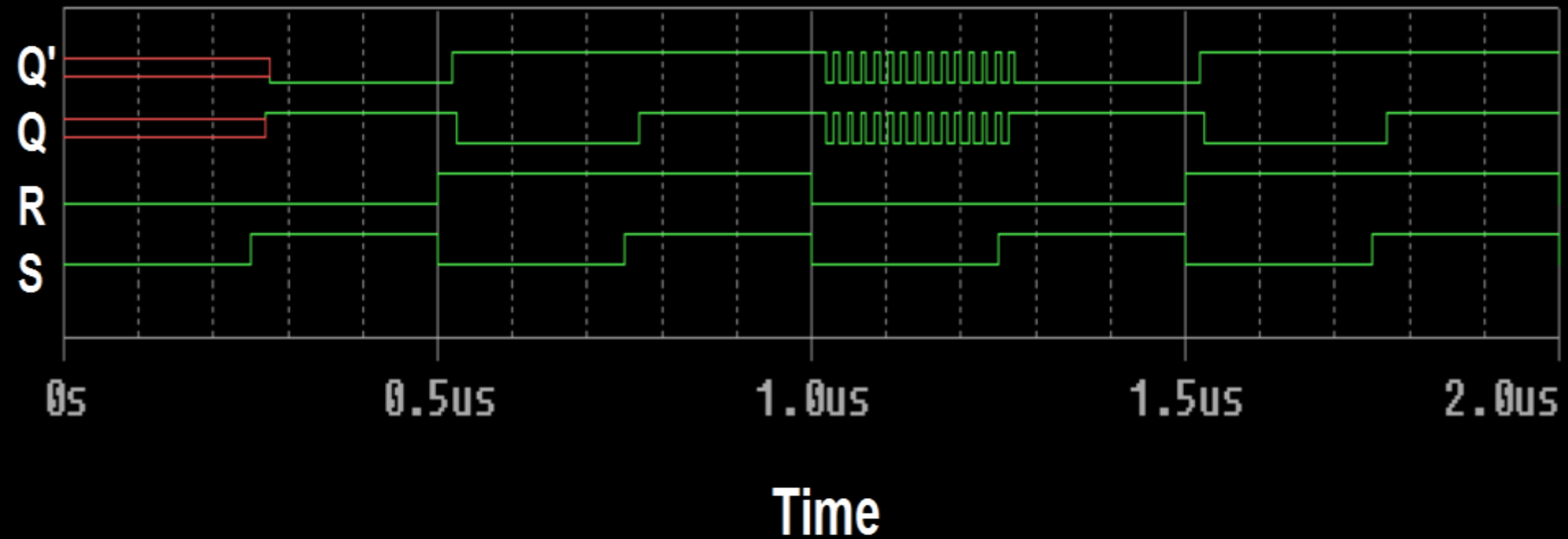
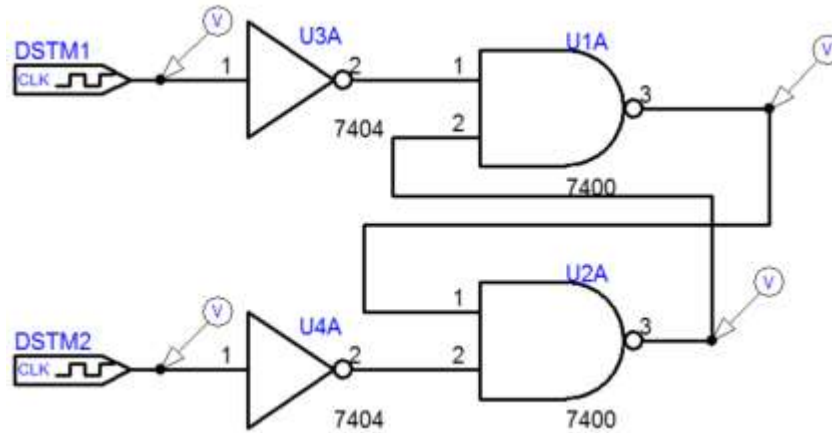
SR Latch

| S | R | Q |
|----------|----------|--------|
| 0 | 0 | Last Q |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | |

SR Latch with Enable

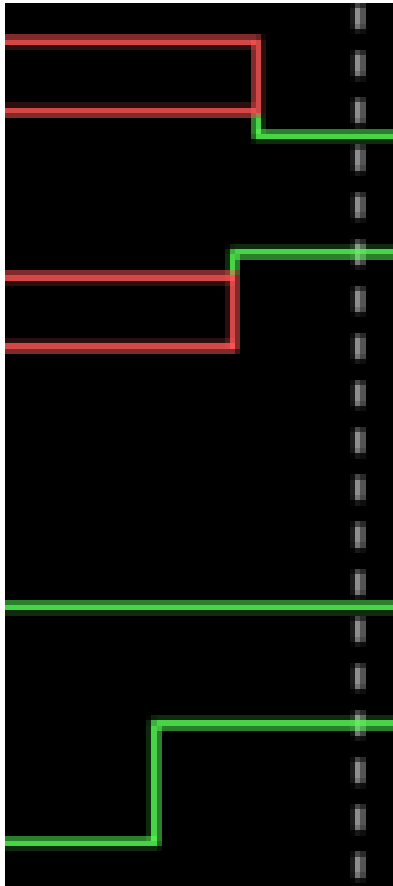
| S | R | E | Q |
|----------|----------|----------|--------|
| 0 | 0 | 1 | Last Q |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | |
| X | X | 0 | Last Q |

Timing Diagram

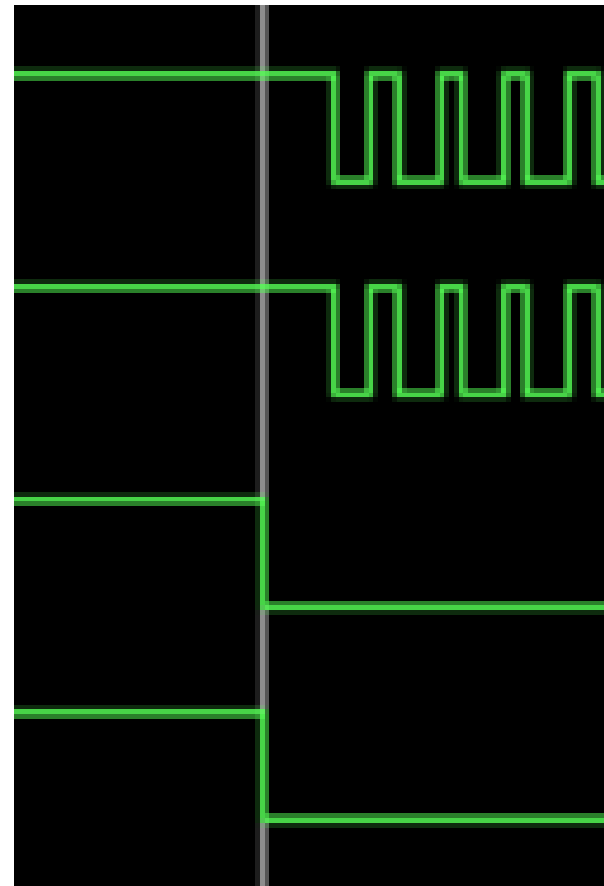


Timing Diagram

**No Initial Condition
Propagation Delay Time**



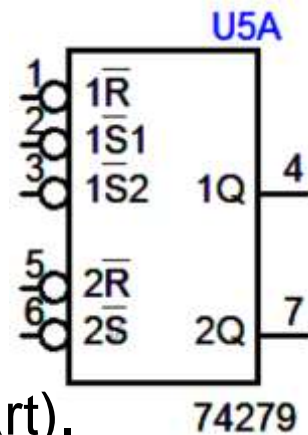
**Indeterminate Condition, $Q = Q'$
Followed by a Race Condition**



Task

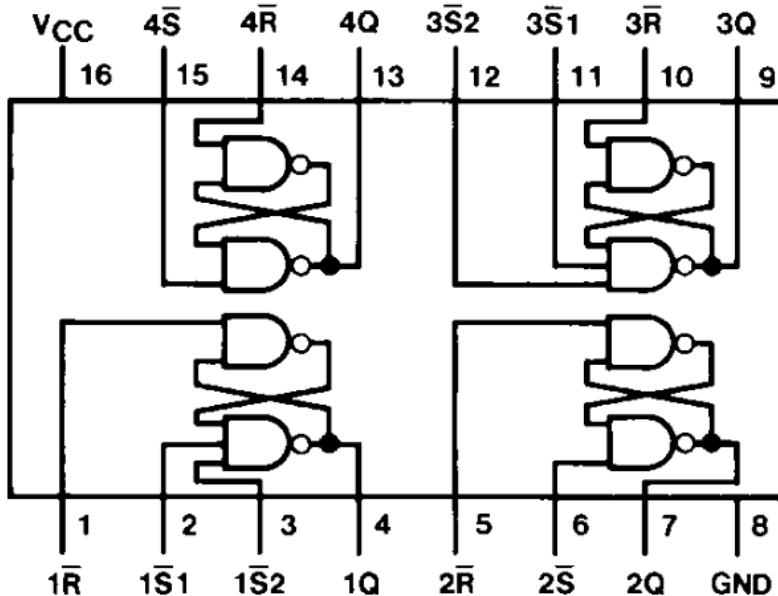
- The two NAND gates form a $\overline{S}\overline{R}$ latch.
 - SR latches can be made using OR, NOR, or AND gates. Can you design one and work out the logic table for its operation?
 - Depending on the design and which output is called Q, the race condition occurs when $S = R = 0$ and the **last Q** state occurs when $S = R = 1$.
 - Can you predict the logic table before you have constructed the circuit and simulated/measured output for the four sets of input states?

74279



- Note that there is dual SR bar latch in PSpice (2 in 1 part).
 - It may appear that the undefined operation has been designed out of its operation when you use this part in a simulation. However, the datasheet indicates that the race condition may show up.

Connection Diagram



Function Table

| Inputs | | Output |
|--------------------|-----------|------------|
| \bar{S} (Note 1) | \bar{R} | Q |
| L | L | H (Note 2) |
| L | H | H |
| H | L | L |
| H | H | Q_0 |

H = HIGH Level

L = LOW Level

Q_0 = The Level of Q before the indicated input conditions were established.

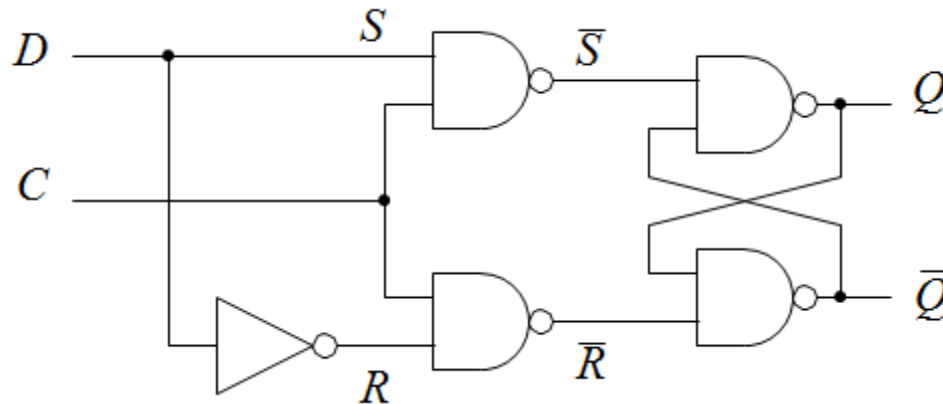
Note 1: For latches with double \bar{S} inputs:

H = both \bar{S} inputs HIGH

L = one or both \bar{S} inputs LOW

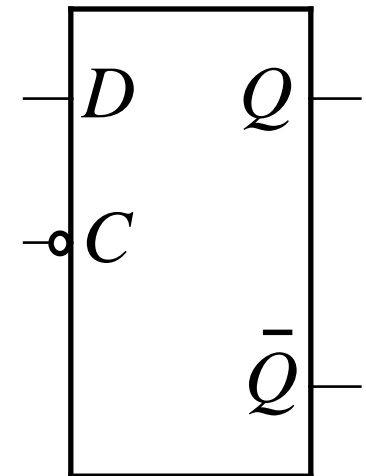
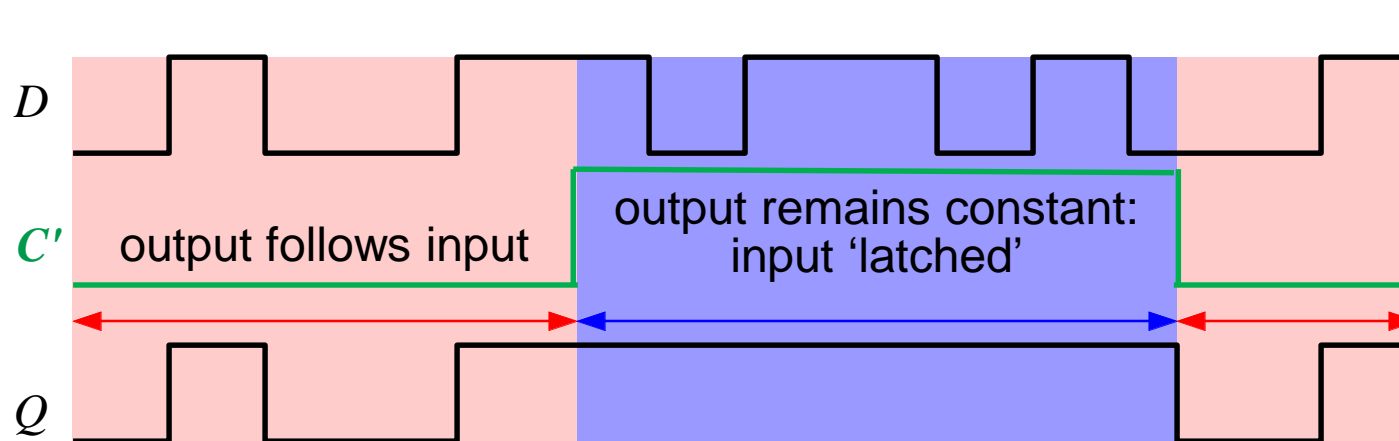
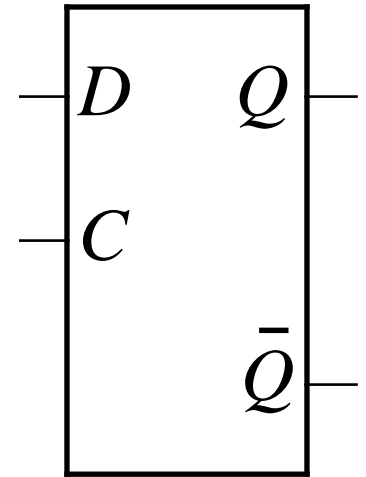
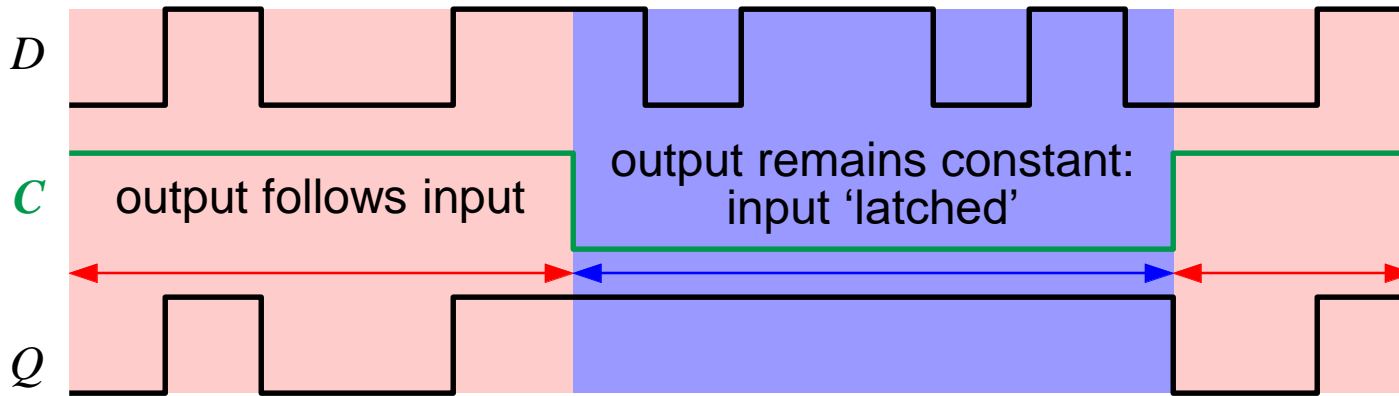
Note 2: This output level is pseudo stable; that is, it may not persist when the \bar{S} and \bar{R} inputs return to their inactive (HIGH) level.

D Flip-Flop



- The problem with $S = R = 1$ can be avoided using a common input D as shown above so that $S = \bar{R}$.
- The output of the latch now:
 - **follows the D input** while $C = 1$ (transparent)
 - **holds its value** while $C = 0$ ($Q = \text{last } Q \text{ when } C \text{ went low}$) no matter what happens at the input
- This circuit is often called a **transparent latch**. It can be bought as an integrated circuit, usually with several latches in a package.
 - The input C may be called **control**, **clock**, **gate**, or **enable**.

Timing Diagrams



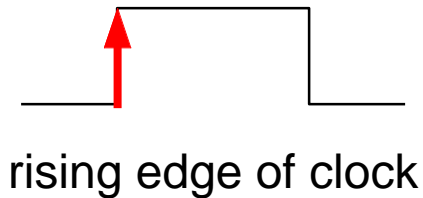
Transparent Latches

- Transparent latches have important applications in digital electronics.
 - However, the stability of the output can be an issue in noisy environments.
 - It is more convenient if the behaviour depends on the inputs only at a particular time, which led to the invention of edge-triggered flip-flops.
 - Transparent latches are sometimes called ‘level-sensitive’ flip-flops to distinguish them from edge-sensitive devices. Yet another name is ‘half flip-flop’.

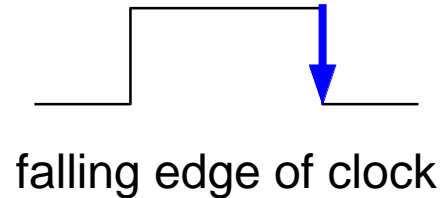
Edge-Triggered Flip-Flops

These circuits respond to their inputs on either the **rising** or **falling** edge of the clock — a precise point in time rather than an interval.

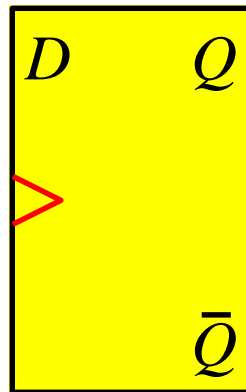
Positive edge triggered



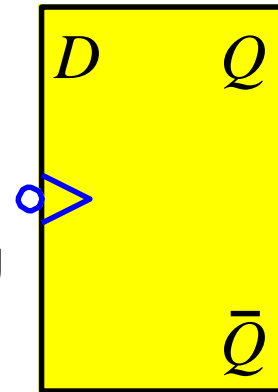
Negative edge triggered



wedge shows **positive** edge triggering



additional of a circle means that there is **negative** edge triggering



Older flip-flops may be 'pulse-triggered', which require a clock pulse that goes from $0 \rightarrow 1 \rightarrow 0$ or a 'master-slave' types but these are now obsolete.

Excitation Tables

Logic tables show the state of the output(s) of a logic circuit as a function of its inputs **at the same time**.


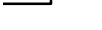

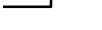
Since, clocked digital systems have **memory**, their behaviour depends on inputs **in the past** as well as the present values of the inputs.

Thus, flip-flops cannot be described by simple truth tables. Instead, we use **excitation** or **transition tables**. These show:

- **output before** the clock transition — often labelled Q_n
- **inputs at** the clock transition — such as S and R
- occasionally the type of clock transition – positive/negative edge-triggered
- the resulting **output after** the clock transition — often labelled Q_{n+1}

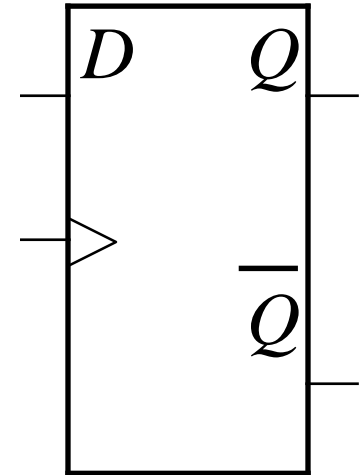
It is important to remember that Q_n and Q_{n+1} describe the **same** signal but at **different times**. The notation can vary, e.g. Q_0 and Q instead.

D Flip-Flop

| D | C | Q_n | Q_{n+1} | description |
|-----|---|-------|-----------|------------------|
| 0 |  | 0 | 0 | Clear (reset) |
| 0 |  | 1 | 0 | |
| 1 |  | 0 | 1 | Set |
| 1 |  | 1 | 1 | |

input at clock output before clock output after clock

$$Q_{n+1} = D$$



A D flip-flop simply stores the value on its D input at the clock transition.

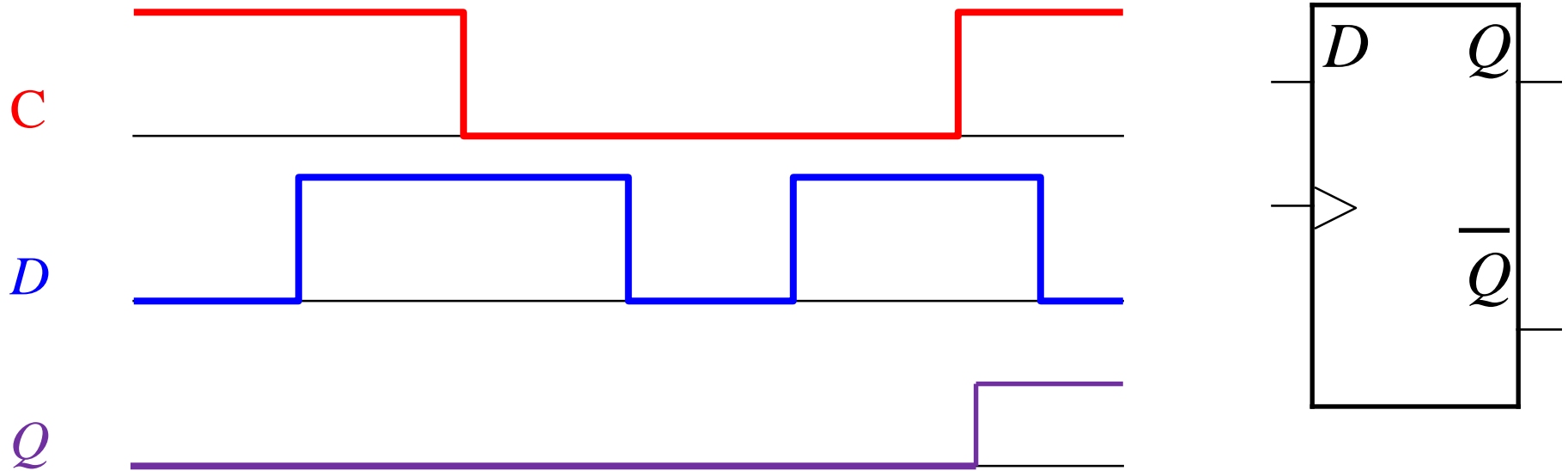
The previous value stored, Q_n , has no effect, unlike other flip-flops.

It therefore acts as a simple memory or 'latch'.

The most widely used flip-flops: simple to build and design with.

A **register** comprises several D flip-flops, one for each bit to be stored.

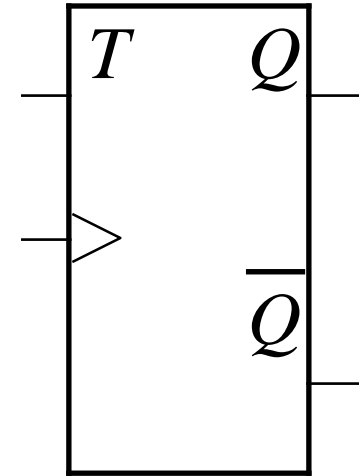
Timing Diagram: Edge-Triggered FF



The input D can change at any time because it comes from other parts of the system — it is not necessarily synchronized to the clock (it may be from a switch on the front panel, for instance). However, the flip-flop only changes its output when the clock pulse rises.

Toggle (T) Flip-Flop

| T | Q_n | Q_{n+1} | description |
|-----|-------|-----------|-------------|
| 0 | 0 | 0 | hold |
| 0 | 1 | 1 | |
| 1 | 0 | 1 | toggle |
| 1 | 1 | 0 | |



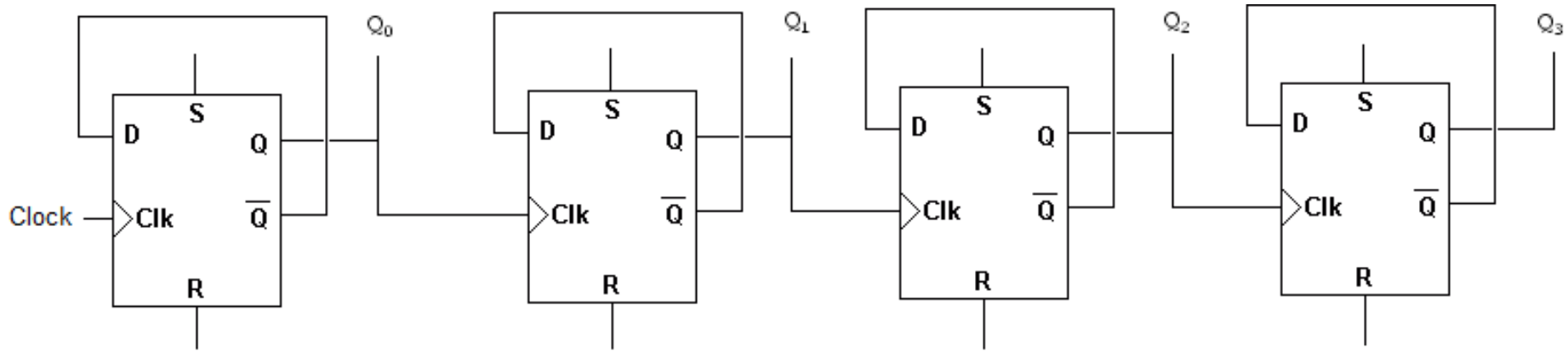
$$Q_{n+1} = T \oplus Q_n = T \diamond \bar{Q}_n + \bar{T} \diamond Q_n$$

Note that the output depends on the previous value stored, Q_n .

This type of flip-flop is rarely bought as a 'dedicated' device — you can easily make a T from a D flip-flop.

Aside: it is not possible to put a specific value into a T flip-flop — it can only toggle or hold.

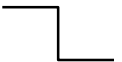



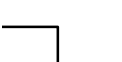
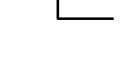


What does this circuit do?



Assume that the initial state of Q_0 , Q_1 , Q_2 , and Q_3 are all logical zeros.

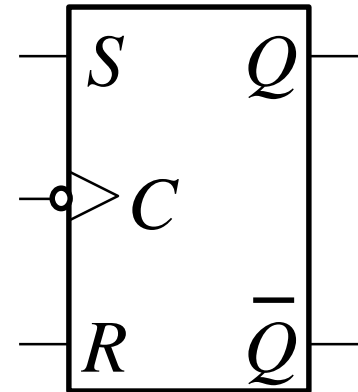
1. Determine how Q_0 changes as the clock pulse goes from $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$.
2. After how many clock pulses will Q_1 change state from '0' to '1'?
3. After how many clock pulses will Q_2 change state from '0' to '1'?
4. After how many clock pulses will Q_3 change state from '0' to '1'?

S-R Latch

| S | R | C | Q_n | Q_{n+1} | description |
|-----|-----|---|-------|-----------|---------------|
| 0 | 0 |  | 0 | 0 | hold |
| 0 | 0 |  | 1 | 1 | |
| 0 | 1 |  | 0 | 0 | clear (reset) |
| 0 | 1 |  | 1 | 0 | |
| 1 | 0 |  | 0 | 1 | set |
| 1 | 0 |  | 1 | 1 | |
| 1 | 1 |  | 0 | ? | indeterminate |
| 1 | 1 |  | 1 | ? | — avoid |

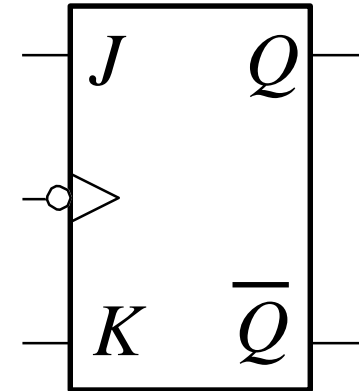
inputs
at
clock transition

output before
clock output after
clock



JK Flip-Flop

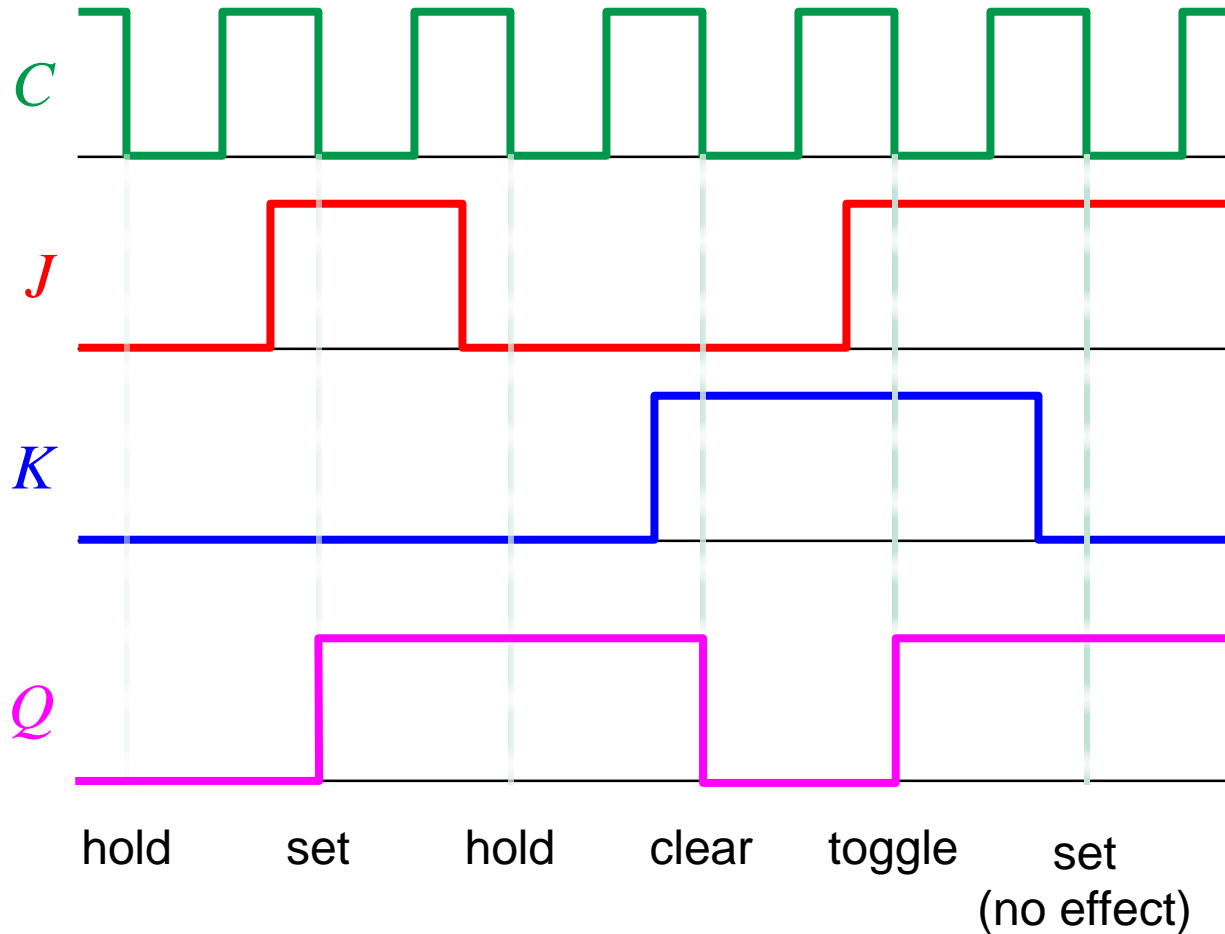
| J | K | Q_n | Q_{n+1} | description |
|-----|-----|-------|-----------|---------------|
| 0 | 0 | 0 | 0 | hold |
| 0 | 0 | 1 | 1 | |
| 0 | 1 | 0 | 0 | clear (reset) |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 1 | set |
| 1 | 0 | 1 | 1 | |
| 1 | 1 | 0 | 1 | toggle |
| 1 | 1 | 1 | 0 | |



$$Q_{n+1} = J \diamond \bar{Q}_n + \bar{K} \diamond Q_n$$

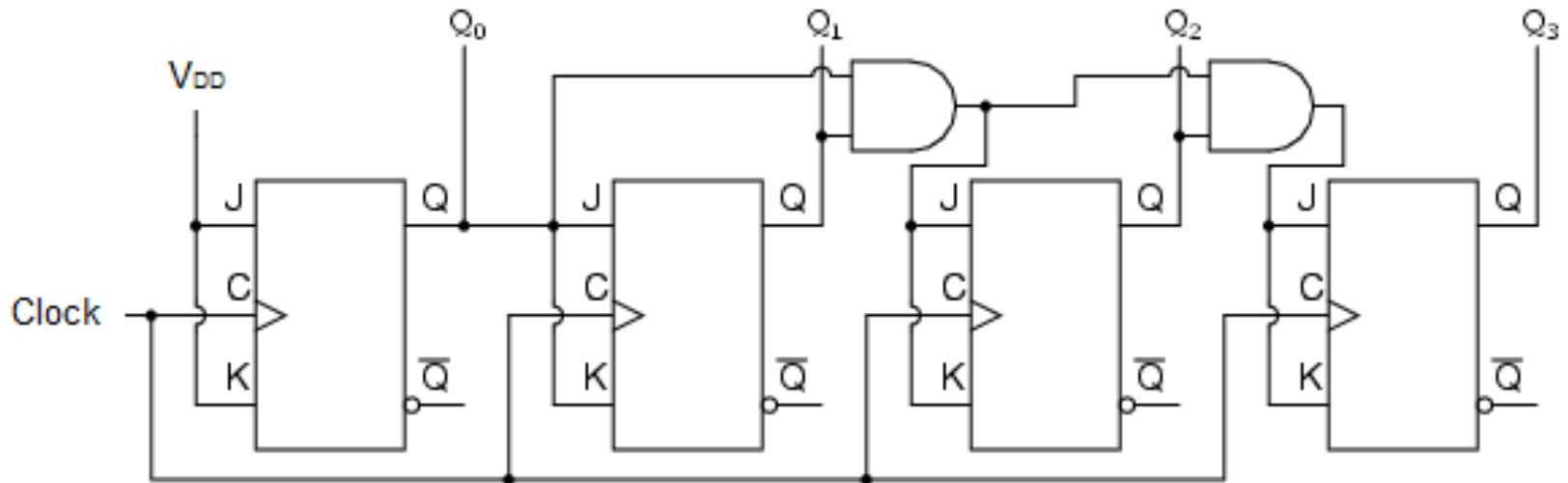
The excitation table for a JK flip-flop is similar to SR flip-flop but doesn't have the problem of $S = R = 1$. It can perform all the operations of the simpler types of flip-flop. However, the design of the circuit internal to the flip-flop makes it more expensive to manufacture than a number of other flip-flops so JK flip-flops are now rarely used.

Timing Diagram: JK Flip-Flop



Suggestion: Determine the Q output for a negative edge triggered JK flip-flop with the inputs shown above, assuming that Q starts at logic 0.

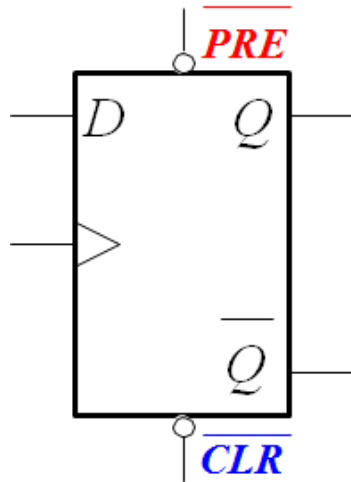
Same Operation as Circuit on Slide 24



Task: Prove this to yourself.

Control Pins

Flip-flops and more complicated circuits often have inputs, such as **clear**, **preset**, **enable**, and **load**. The state of the control pins has priority of the D and Clock inputs when determining the state of the output.

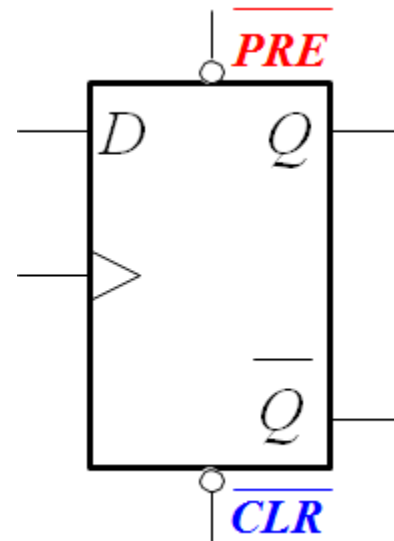


- **Clear** (*CLR*) resets the flip-flop output to 0 — the most common control input
- **Preset** (*PRE*) sets the flip-flop output to 1

More complicated circuits such as counters may have additional control inputs for up/down, count/hold, load, *etc.* Microprocessors usually have a reset pin.

Active Low Controls

Control inputs are often **active low**, shown by a bar over the label or a circle for negation (or both as in the component below).



Active low inputs should be:

- **kept high for normal operation**
- changed to low to **preset** or **clear** the device.

The reason for making these active low is historical.

Check the data sheet to be sure!

Asynchronous Control

Another feature of control inputs is that they are often **asynchronous**. This means that they take effect **immediately** and do not wait for a clock transition.

Compare

- **D** takes effect **only at a clock transition** (positive edge)
- **clear** and **preset** act **immediately** to 'overrule' **D**

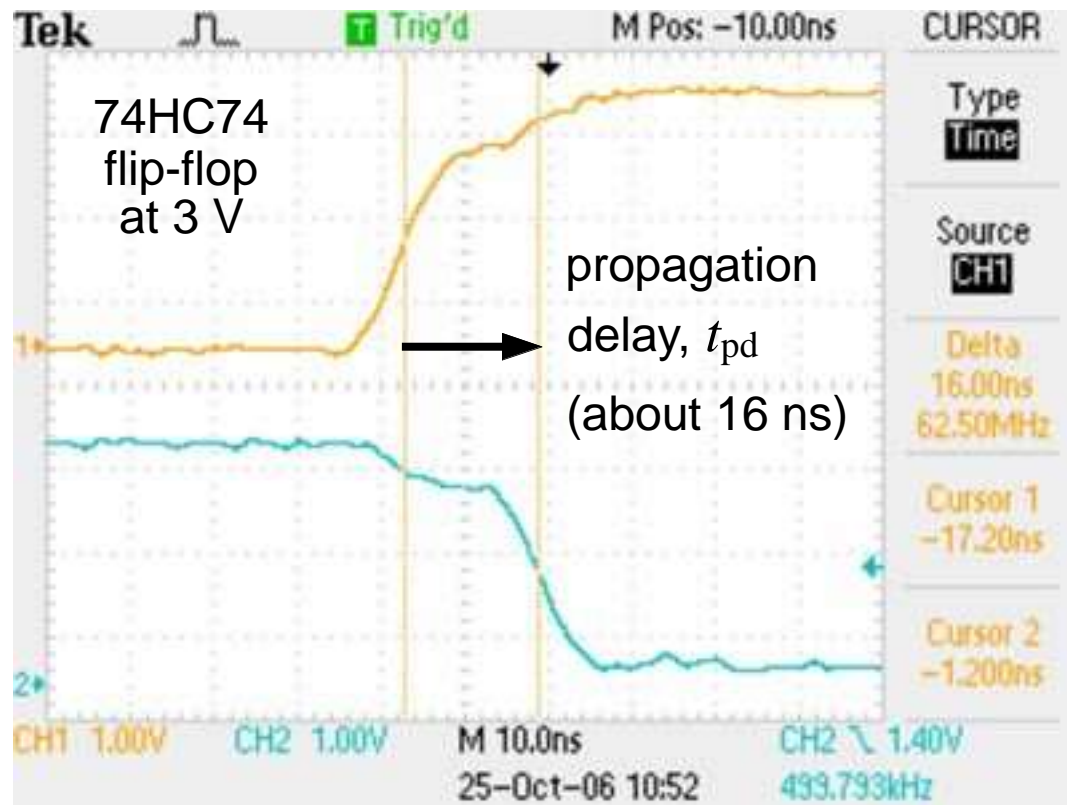
Asynchronous inputs are sometimes called **direct** or **jam** inputs.

Always check the data sheet because some control inputs are synchronous!

Propagation Delay

We have seen that most modern logic devices are triggered on either the rising or falling edge of the clock. The output does not respond instantly, but only after a time called the **propagation delay**, t_{pd} .

Here is an example for a *D* flip-flop that was measured in a UoG lab class.



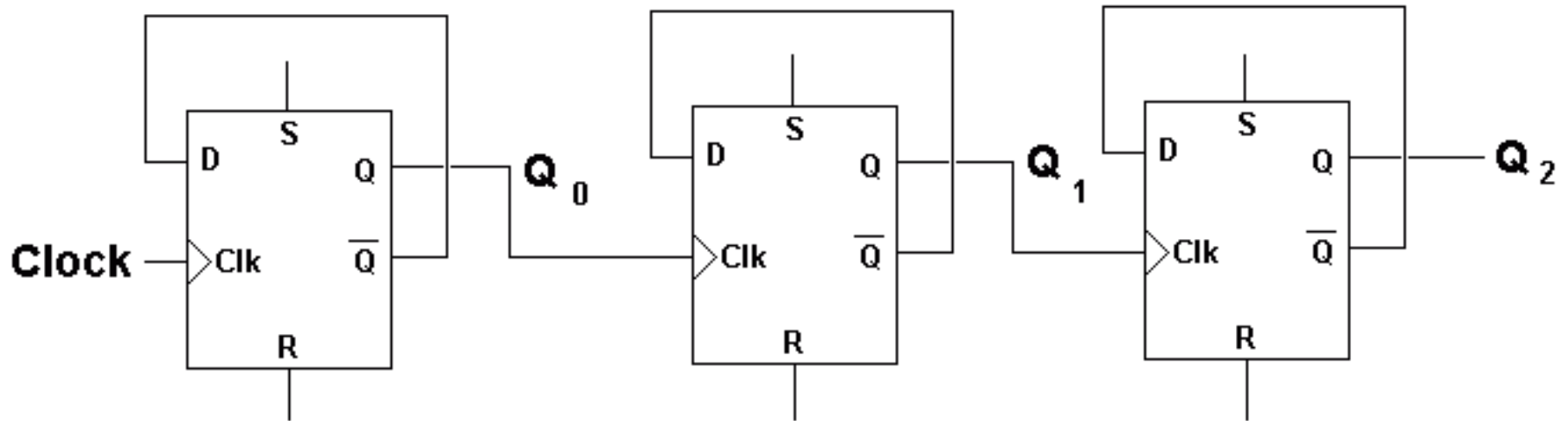
Limitations on Circuit: t_{pd}

The propagation delay is important for several reasons:

- It limits the speed at which circuits can be clocked (20 or 30 MHz for the 'HC' family of components, used in the laboratory)
- Signals that pass through different numbers of components receive different delays, as in a ripple counter. Time must be allowed for all outputs to settle down before the system attempts to change state again.
- The delay helps to keep digital circuits with feedback stable (this applies to virtually all practical circuits).
 - Each logic gate responds to its at the clock transition.
 - Because of t_{pd} , the outputs change after the propagation delay.
 - This affects the inputs that are connected to outputs.
 - However, these gates are no longer acting on their inputs until the next clock transition arrives.

Would this circuit work?

What would happen if t_{pd} is longer than the period of the clock?
What would happen if t_{pd} is about equal to the clock's period?



Lecture Review Questions

1. What is the fundamental difference between **combinational** and **sequential** logic?
2. What inputs should you put on a *SR* flip-flop to set it (to 1)?
3. Why must $S = R = 1$ be avoided?
4. Why do sequential logic circuits need a **clock**?
5. What is meant by the term **edge-triggered** flip-flop?
6. Describe the operation of a **type D** flip-flop. For what are they used?
Describe the operation of a **JK flip-flop** with $J = K = 1$.
7. Why do some flip-flops have **control inputs**? In what ways do they differ from the normal inputs, such as J and K ?
8. What is the **propagation delay**? Why is it important?

THANK YOU