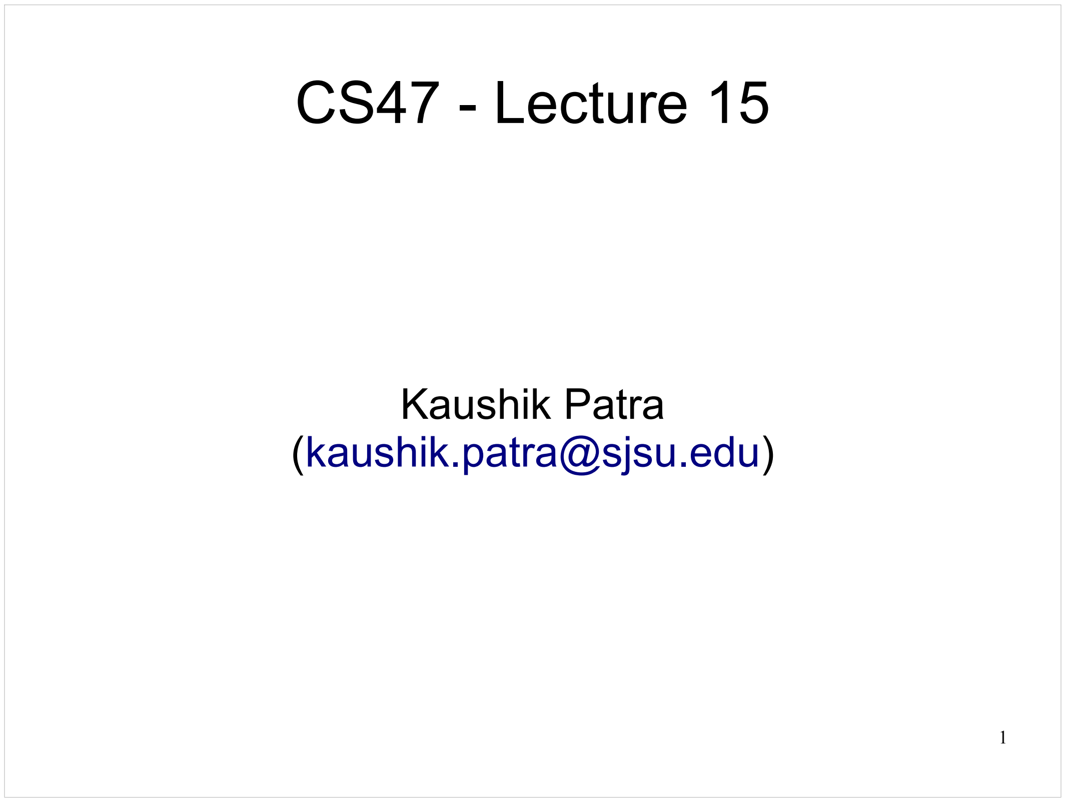


# CS47 - Lecture 15

Kaushik Patra
(kaushik.patra@sjsu.edu)

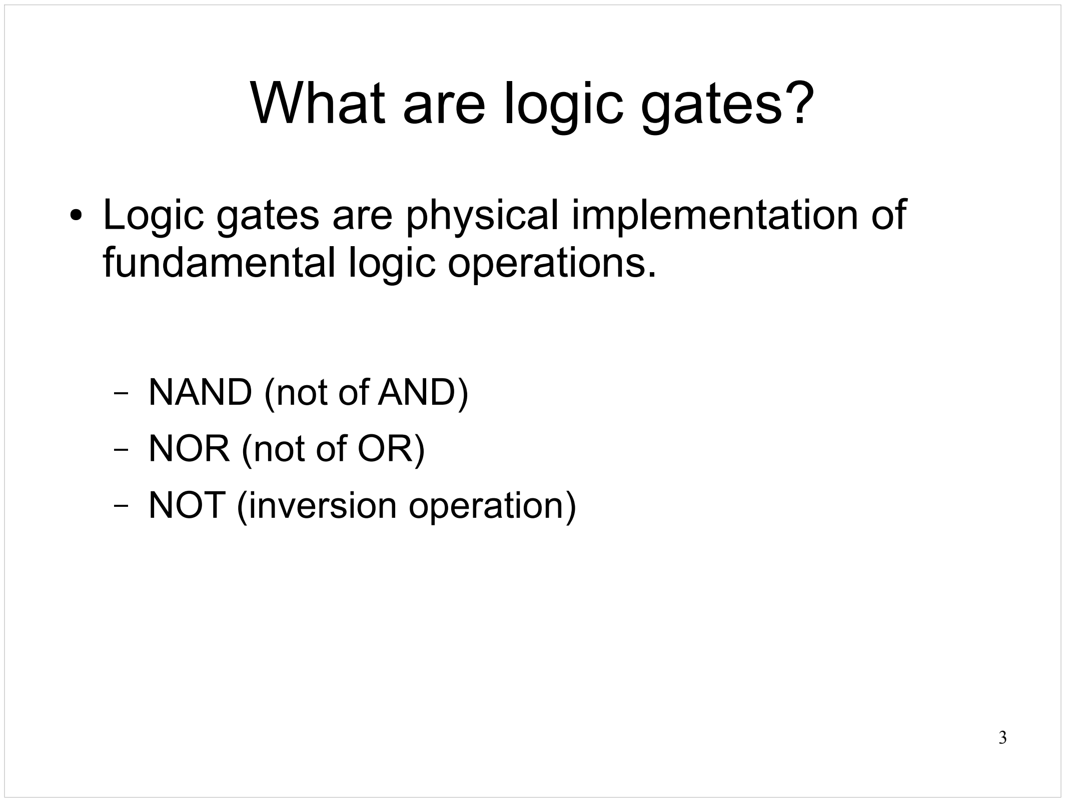
1

- Combinational Logic Gates
- Storage Elements

*[ Chapter 3 of Logic & Computer Design Fundamentals, 4th Edition, M. Morris Mano, Charles R. Kime ]*

## Logic Gates ...

2



## What are logic gates?

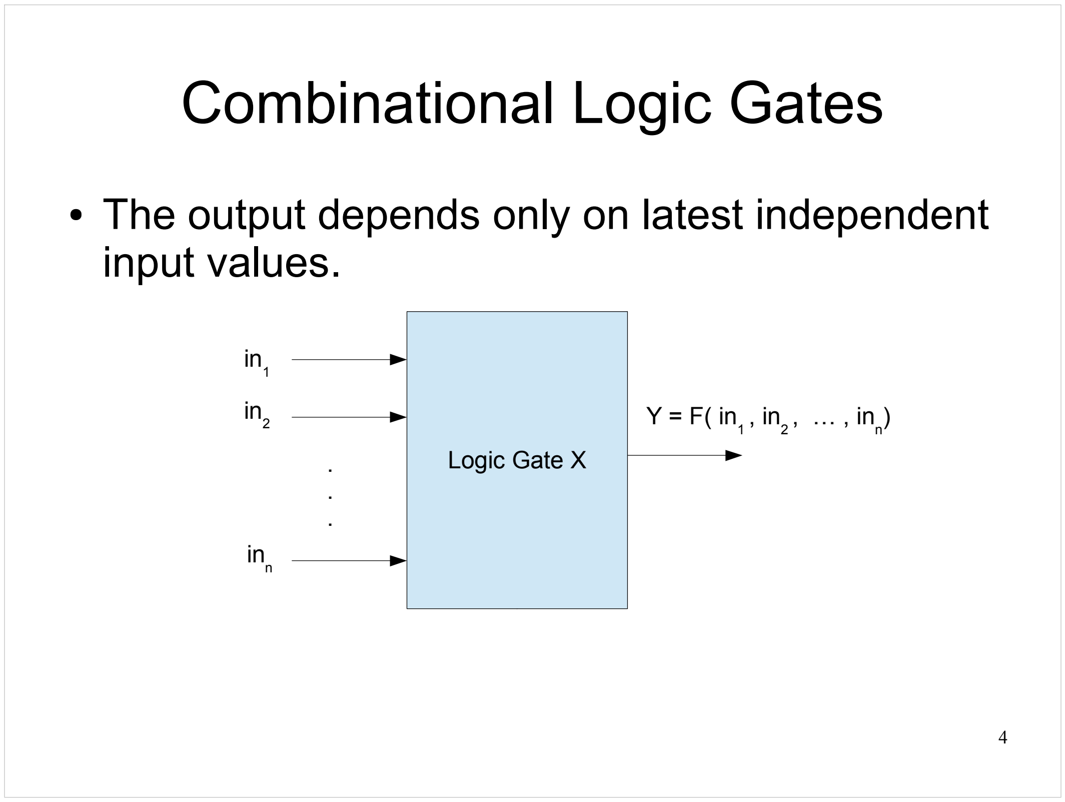
- Logic gates are physical implementation of fundamental logic operations.
  - NAND (not of AND)
  - NOR (not of OR)
  - NOT (inversion operation)

3

- Physical implementation of a fundamental logic function is known as logic gates. Logic gates are basic logic units imprinted on silicon material using IC fabrication technology. All these logical units are realized using combination of very fundamental electronic build blocks called transistor. Discussion of transistor level implementation of logic gates is a part of CS147 material and out of scope for CS47. We'll dwell at logic level of the hardware implementation.

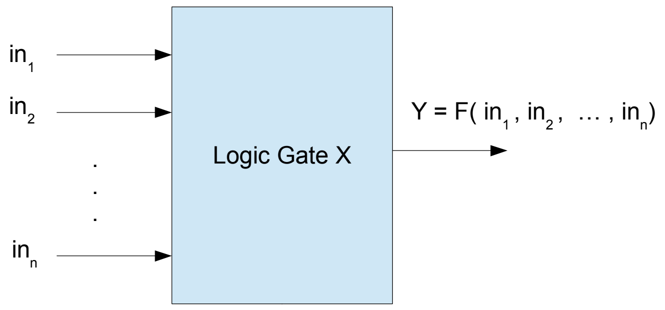
- All modern day transistors are implemented using a technology known as 'CMOS' (Complemented Metal Oxide Semiconductor). From the implementation point of view, CMOS technology gives only complemented Boolean functions. Non-complemented forms are implemented with using these fundamental gates. For example connecting INV and NAND in series would give Boolean AND function ((AB)')' = AB.

- Often DeMorgan theorem is used to transform a function into its complemented form.



## Combinational Logic Gates

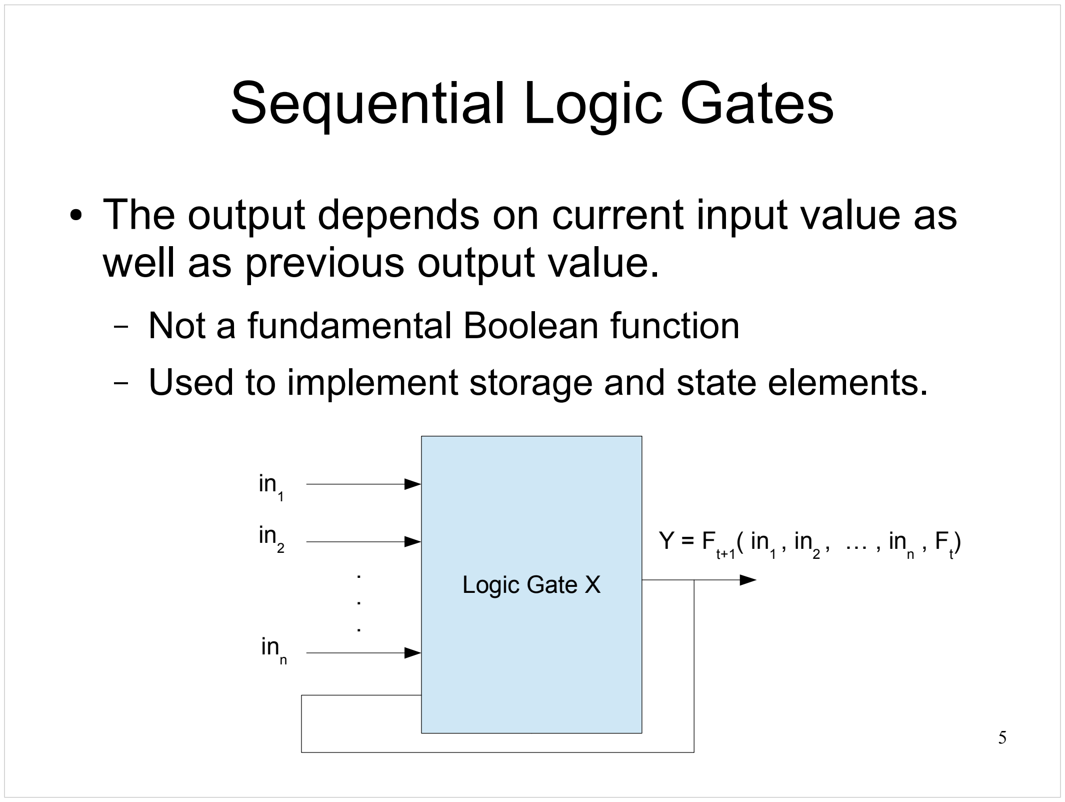
- The output depends only on latest independent input values.



4

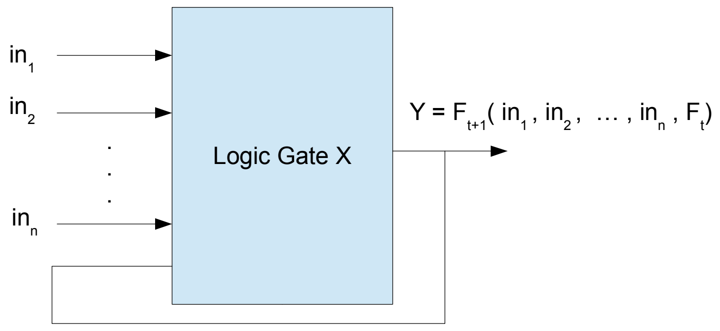
- There are two types of logic gates – combinational and sequential.

- The combinational logic gates realize a Boolean function that purely depends on latest values of its independent input parameters. For example, in this diagram, logic function value Y depends only on independent inputs $in_1$ .. $in_n$.



## Sequential Logic Gates

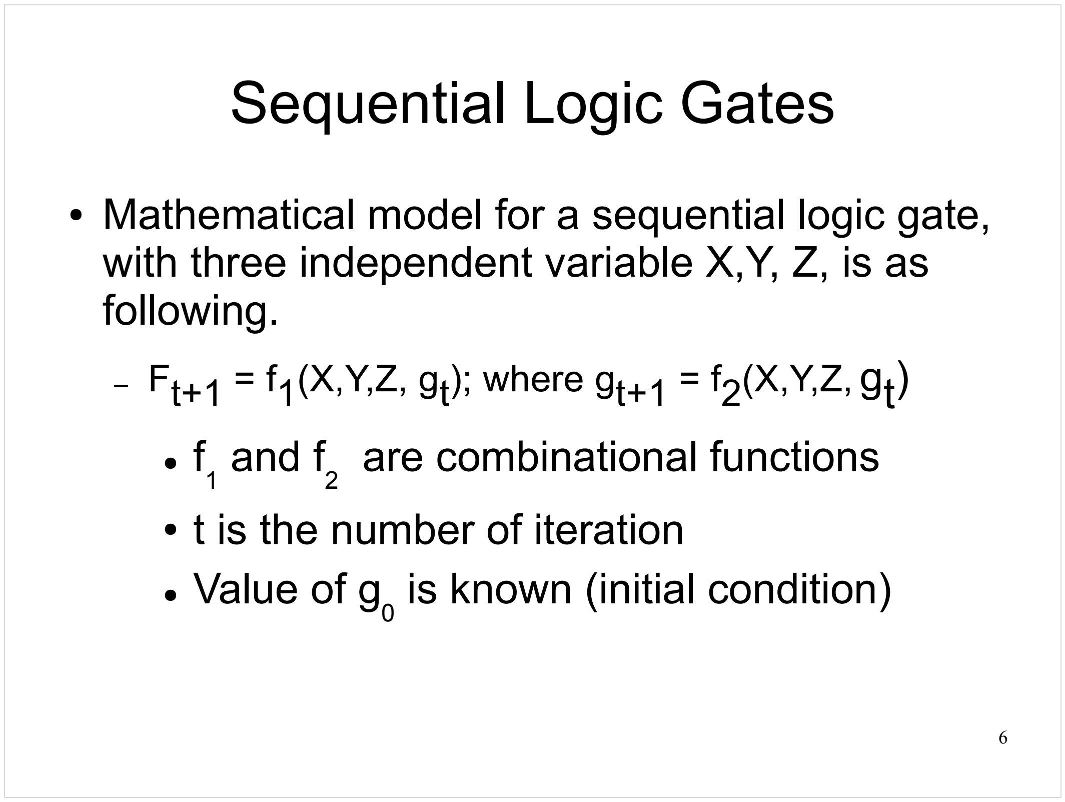
- The output depends on current input value as well as previous output value.
  - Not a fundamental Boolean function
  - Used to implement storage and state elements.



5

• On the other hand, sequential logic gates implements Boolean function that depends not only the latest value of its independent input parameters, but also previous output value.

• These are not to implement Boolean expression, but to implement storage and state-elements (discussed later).



## Sequential Logic Gates

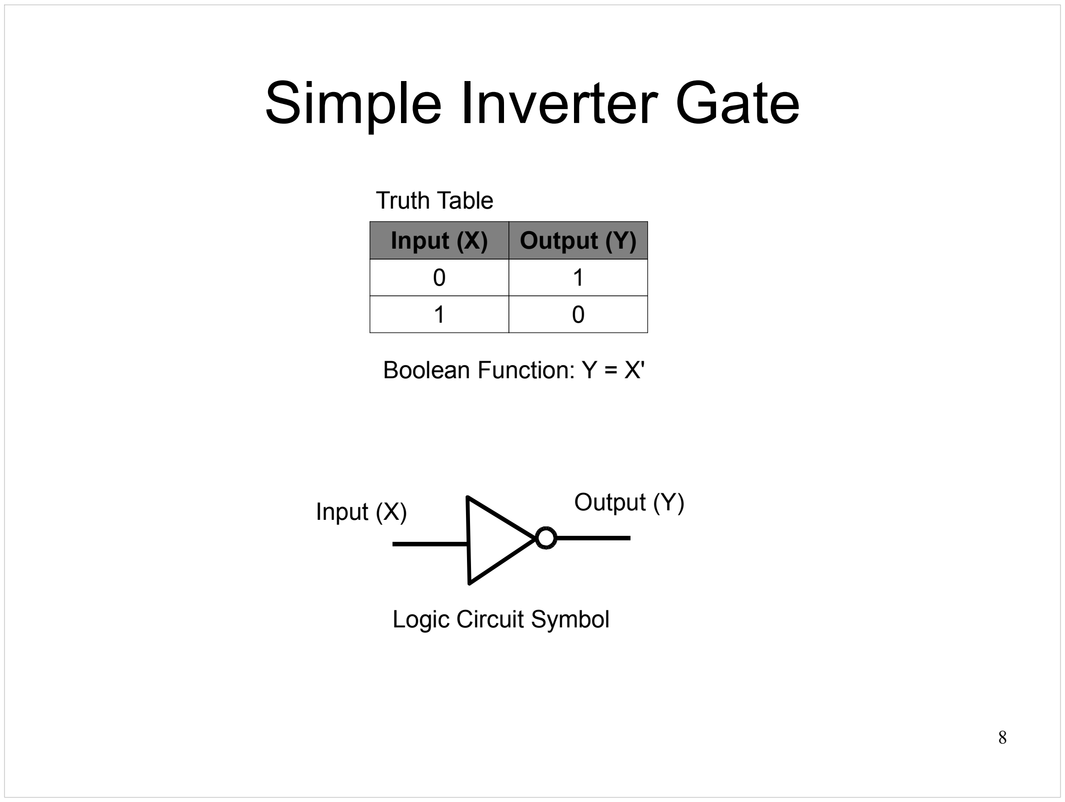
- Mathematical model for a sequential logic gate, with three independent variable X,Y, Z, is as following.
  - $F_{t+1} = f_1(X,Y,Z, g_t)$; where $g_{t+1} = f_2(X,Y,Z, g_t)$
    - $f_1$ and $f_2$ are combinational functions
    - t is the number of iteration
    - Value of $g_0$ is known (initial condition)

- Function evaluation starts at step 1 and then it is evaluated for step 2, 3, and so on.

$$
\begin{aligned}
&F_1 = f_1(X,Y,Z,g_0);\ \ g_1 = f_2(X,Y,Z,g_0)\\
&F_2 = f_1(X,Y,Z,g_1);\ \ g_2 = f_2(X,Y,Z,g_1)\\
&F_3 = f_1(X,Y,Z,g_2);\ \ g_3 = f_2(X,Y,Z,g_2)\\
&\quad\quad \ldots
\end{aligned}
$$

- The initial condition (value) of function g is important. Without this, function values of F are not deterministic. From electronic / digital sequential logic gate point of view this is either power up logic values or the system uses 'reset' signal to bring all the sequential elements in a digital system in a known logic state.

## Combinational Logic Gates ...

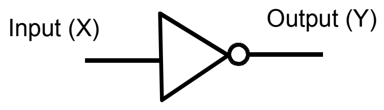


## Simple Inverter Gate

Truth Table

| Input (X) | Output (Y) |
|---|---|
| 0 | 1 |
| 1 | 0 |

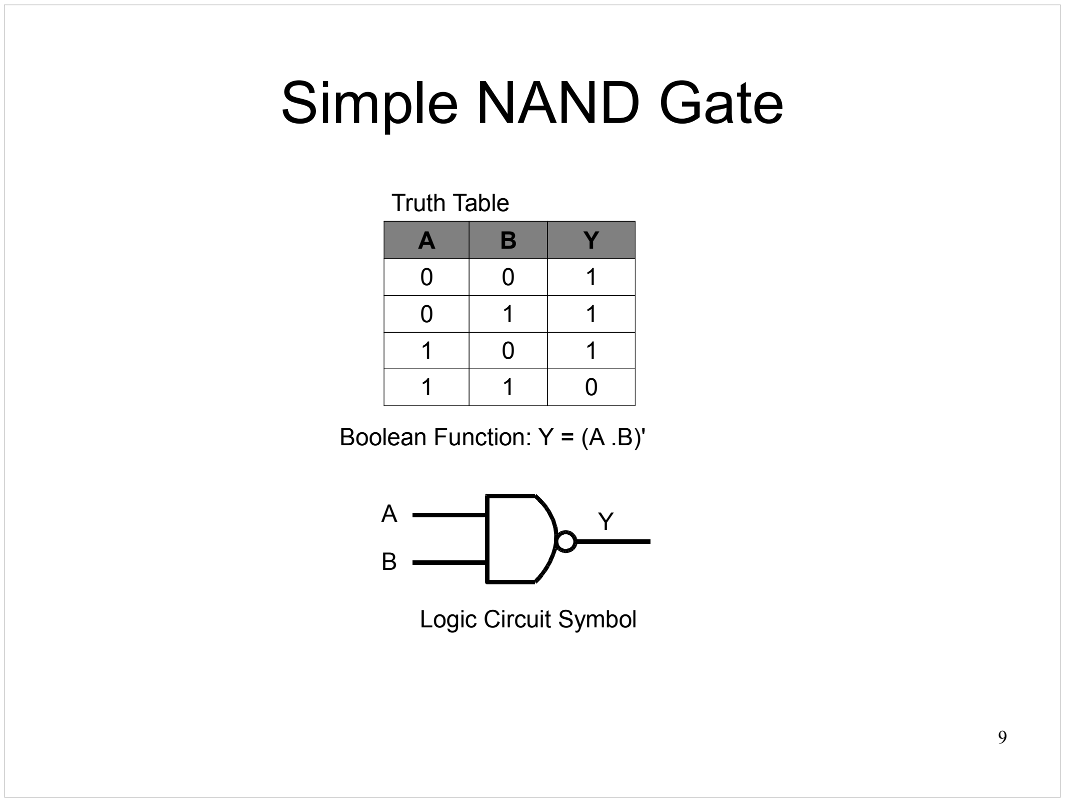
Boolean Function: Y = X'



Logic Circuit Symbol

8

- One of the fundamental logic gate is inverter which implements the Boolean 'not' operation. This gate has specific logic circuit symbol which is used to represent the logic function pictorially.
- This Logic gate is also called INV in short.

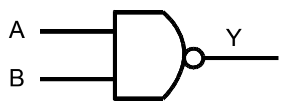


## Simple NAND Gate

Truth Table

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

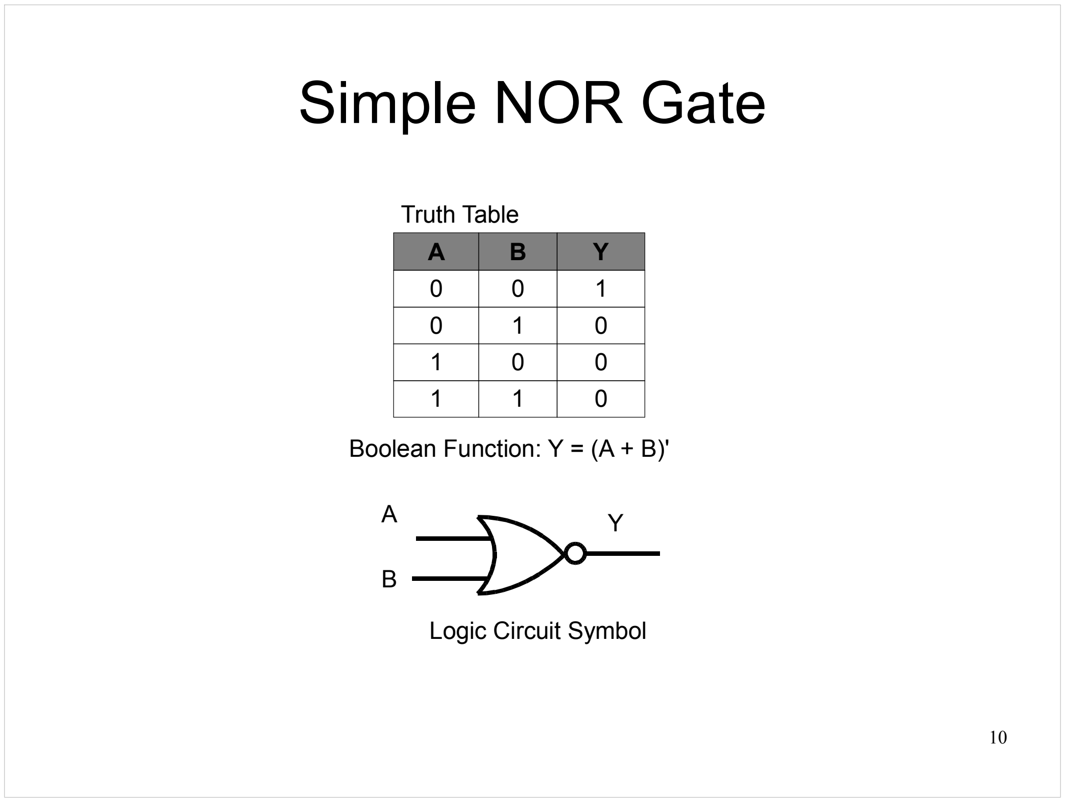
Boolean Function: Y = (A .B)'



Logic Circuit Symbol

9

- Another fundamental logic gate is nand, which implements the Boolean 'nand (not AND)' operation. This gate has specific logic circuit symbol which is used to represent the logic function pictorially.

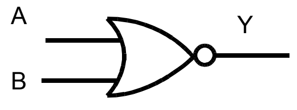


## Simple NOR Gate

Truth Table

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

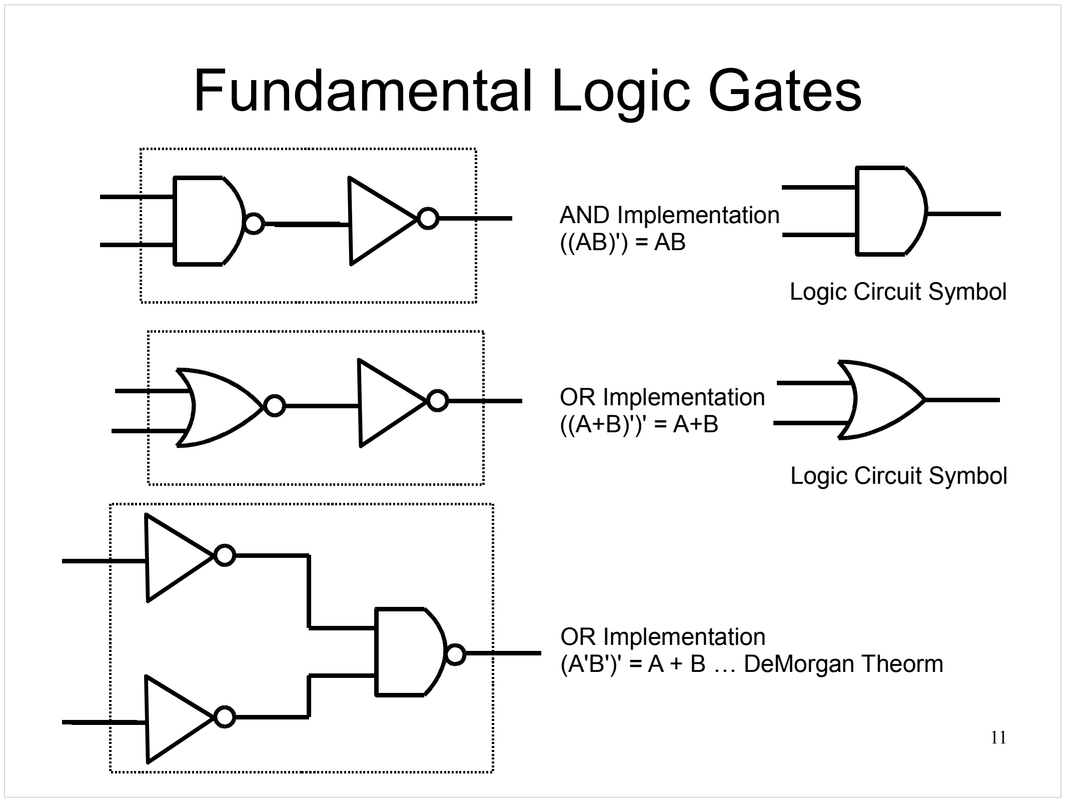
Boolean Function: Y = (A + B)'



Logic Circuit Symbol

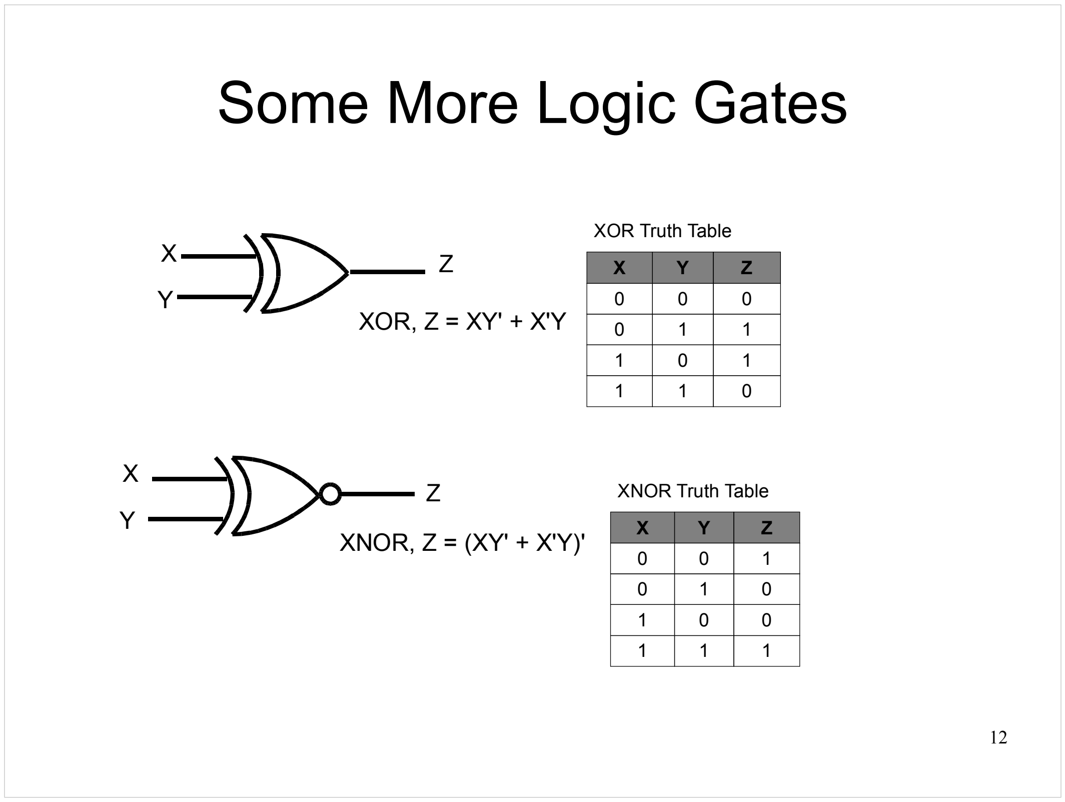
10

- The last type of  fundamental logic gate is nor, which implements the Boolean 'nor (not OR)' operation. This gate has specific logic circuit symbol which is used to represent the logic function pictorially.

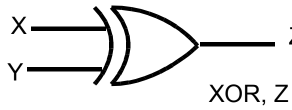


## Fundamental Logic Gates

- The fundamental Boolean logic functions (AND and OR operations) are derived by combining fundamental logic gates as shown above. This fundamental Boolean logic operations (AND and OR) are also assigned with pictorial symbol.



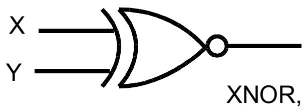
## Some More Logic Gates



XOR, Z = XY' + X'Y

XOR Truth Table

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



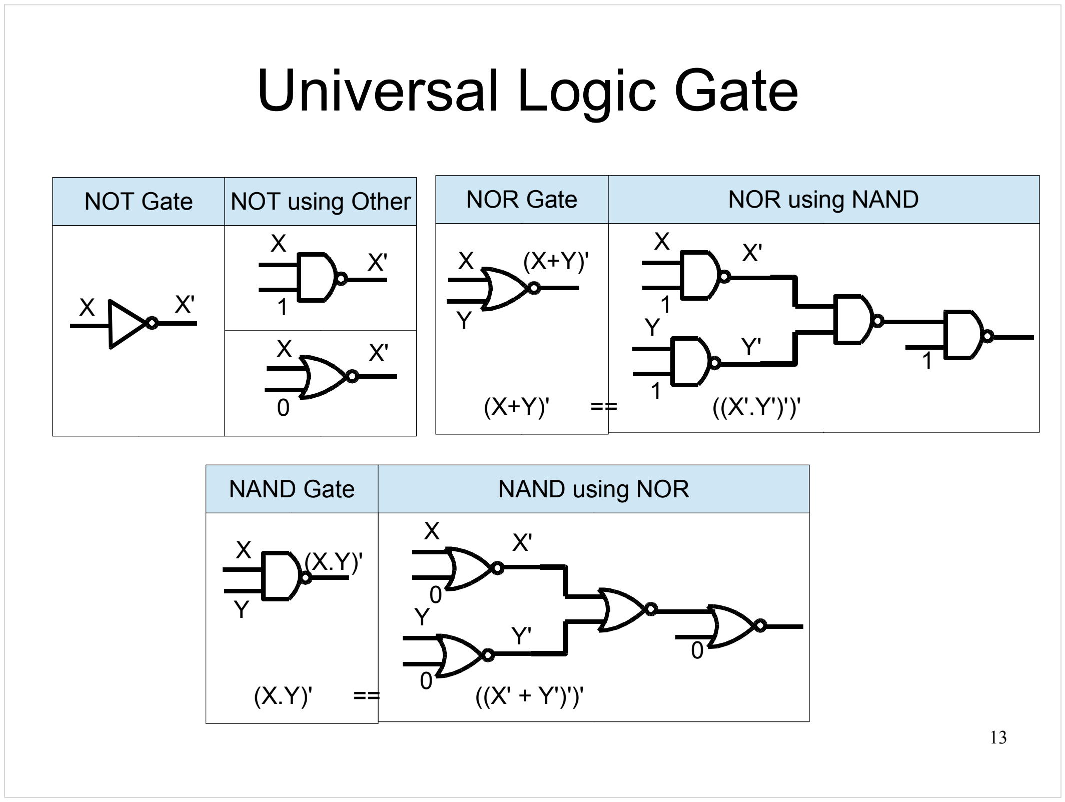
XNOR, Z = (XY' + X'Y)'

XNOR Truth Table

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

12

- Though XOR, XNOR (exclusive OR or NOR) is not fundamental Boolean operation, it is very important from digital logic / circuit design perspective. It has been given its own logic symbol though this function can be expressed in-terms of fundamental logic operation AND, OR and NOT as shown above.



## Universal Logic Gate

- NAND and NOR are called universal logic gate. It is sufficient to use only one type of this gate (either NAND or NOR) to realize any Boolean function.

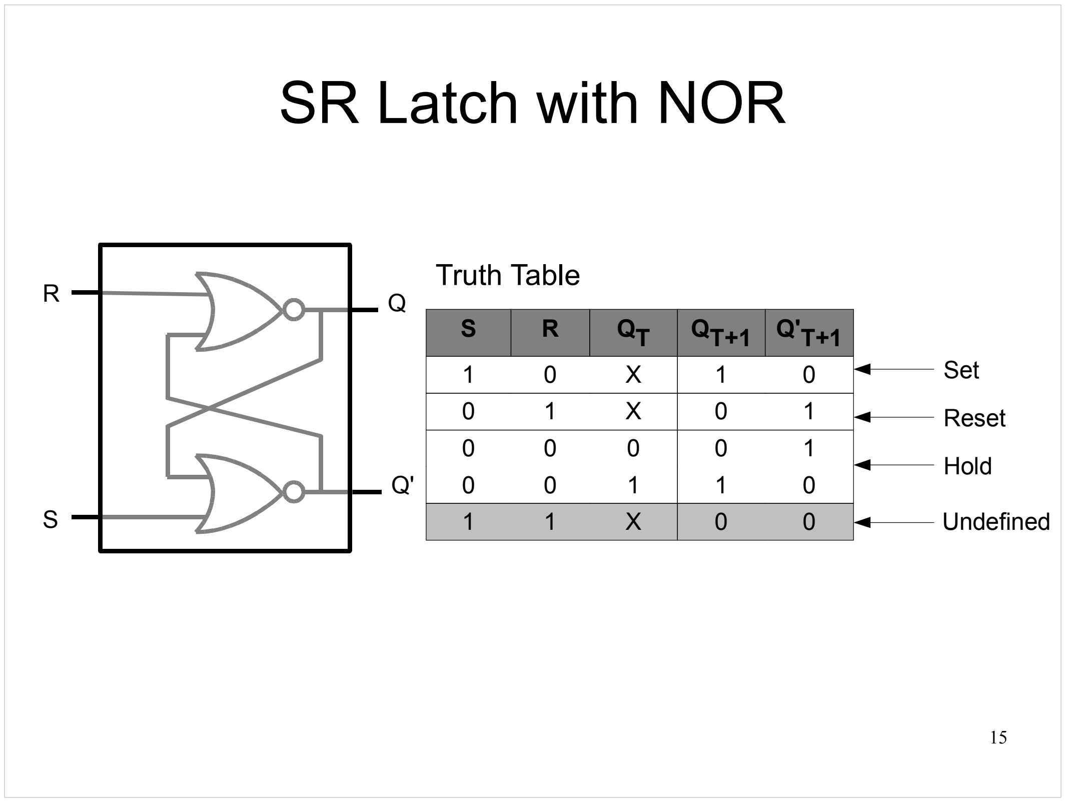
- To implement inverter, we need to set one of the input to a constant non-controlling logic value of the gate. For a NAND gate, non-controlling value is 1 (because if it is 0, the output is always 1 independent of value of other input) and for a NOR gate, non-controlling value is 0 (because if it is 1, the output is always 0 independent of value of other input).

  - (X.1)' = X' ; (X + 0)' = X'

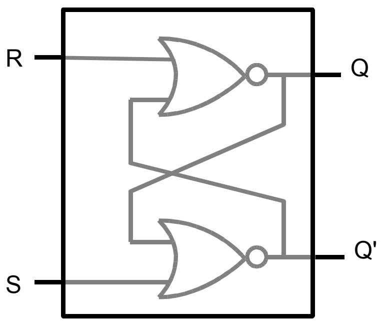
- Once we have inverter realized with NAND or NOR, we can use it to realize NAND using NOR only gates and vice-verse. The De-Morgan rule is applied to realize this transformation.

  - (X+Y)' = ((X'.Y')')' ; transform inner X+Y to (X'.Y')' using De-Morgan rule. This way we have inverter and nand only expression. Therefore it can be realized with NAND gate only.
  - (X.Y)' = ((X'+Y')')' ; transform inner X.Y to (X' + Y')' using De-Morgan rule. This way we have inverter and nor only expression. Therefore it can be realized with NOR gate only.

## Storage Elements ...



## SR Latch with NOR



Truth Table

| S | R | $Q_T$ | $Q_{T+1}$ | $Q'_{T+1}$ | |
|---|---|---|---|---|---|
| 1 | 0 | X | 1 | 0 | Set |
| 0 | 1 | X | 0 | 1 | Reset |
| 0 | 0 | 0 | 0 | 1 | Hold |
| 0 | 0 | 1 | 1 | 0 | |
| 1 | 1 | X | 0 | 0 | Undefined |

15

- This logic circuit with two NOR having mutual feedback is a storage element or sequential gate called SR-LATCH.

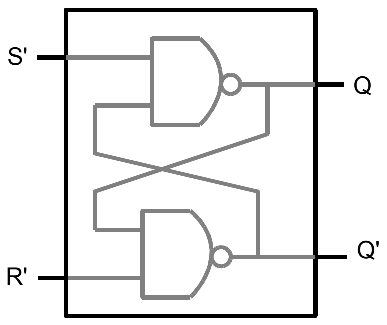
- S=1, R=0 is the set operation which brings the Q to Logic 1.

- S=0, R=1 is the reset operation which brings the Q to Logic 0.

- S=0, R=0 is the hold state which will hold the previous data of Q.

- S=1, R=1 will bring both output to 0 and a successive S=0, R=0 will put the gate in a race condition which will result in a indeterministic state. Hence this setting is not desirable during circuit operation.

## S'R' Latch with NAND

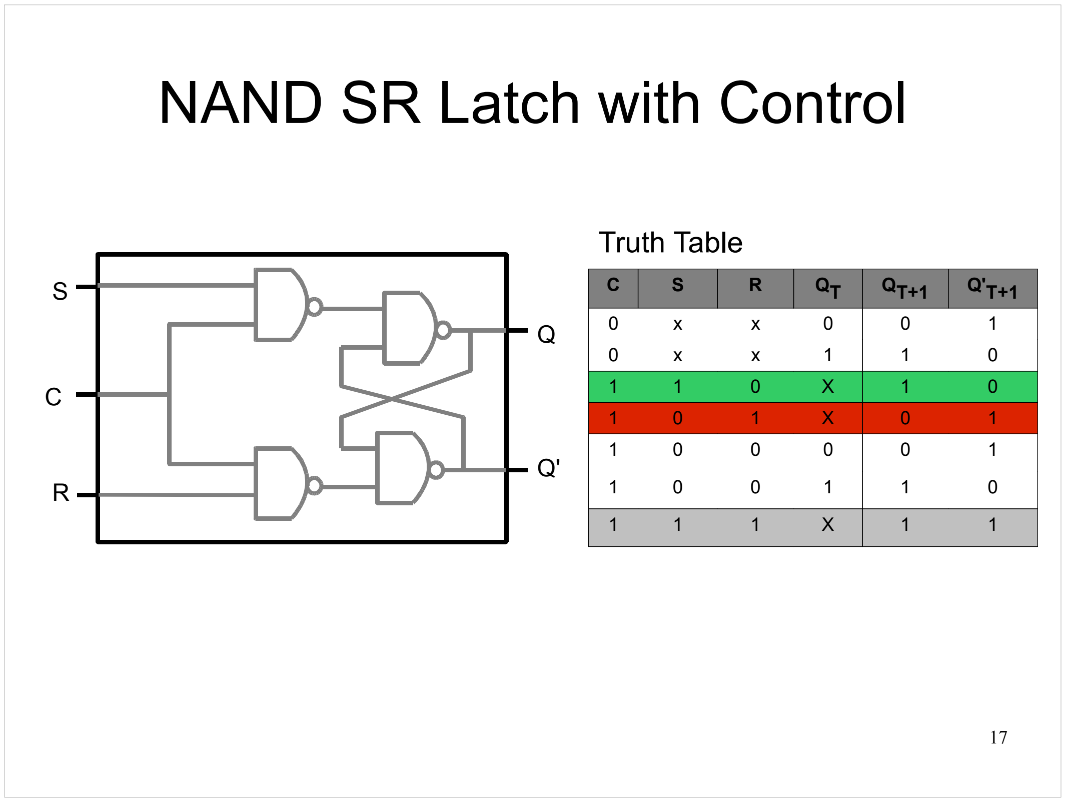


Truth Table

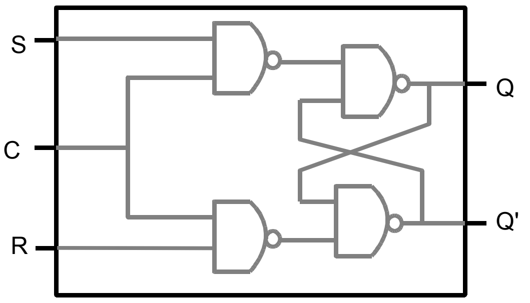
| S' | R' | $Q_T$ | $Q_{T+1}$ | $Q'_{T+1}$ | |
|---|---|---|---|---|---|
| 0 | 1 | X | 1 | 0 | Set |
| 1 | 0 | X | 0 | 1 | Reset |
| 1 | 1 | 0 | 0 | 1 | Hold |
| 1 | 1 | 1 | 1 | 0 | |
| 0 | 0 | X | 1 | 1 | Undefined |

16

- This logic circuit with two NAND having mutual feedback is a storage element or sequential gate called S'R'-LATCH.
- S=0, R=1 is the set operation which brings the Q to Logic 1.
- S=1, R=0 is the reset operation which brings the Q to Logic 0.
- S=1, R=1 is the hold state which will hold the previous data of Q.
- S=0, R=0 will bring both output to 1 and a successive S=1, R=1 will put the gate in a race condition which will result in a indeterministic state. Hence this setting is not desirable during circuit operation.



## NAND SR Latch with Control



Truth Table

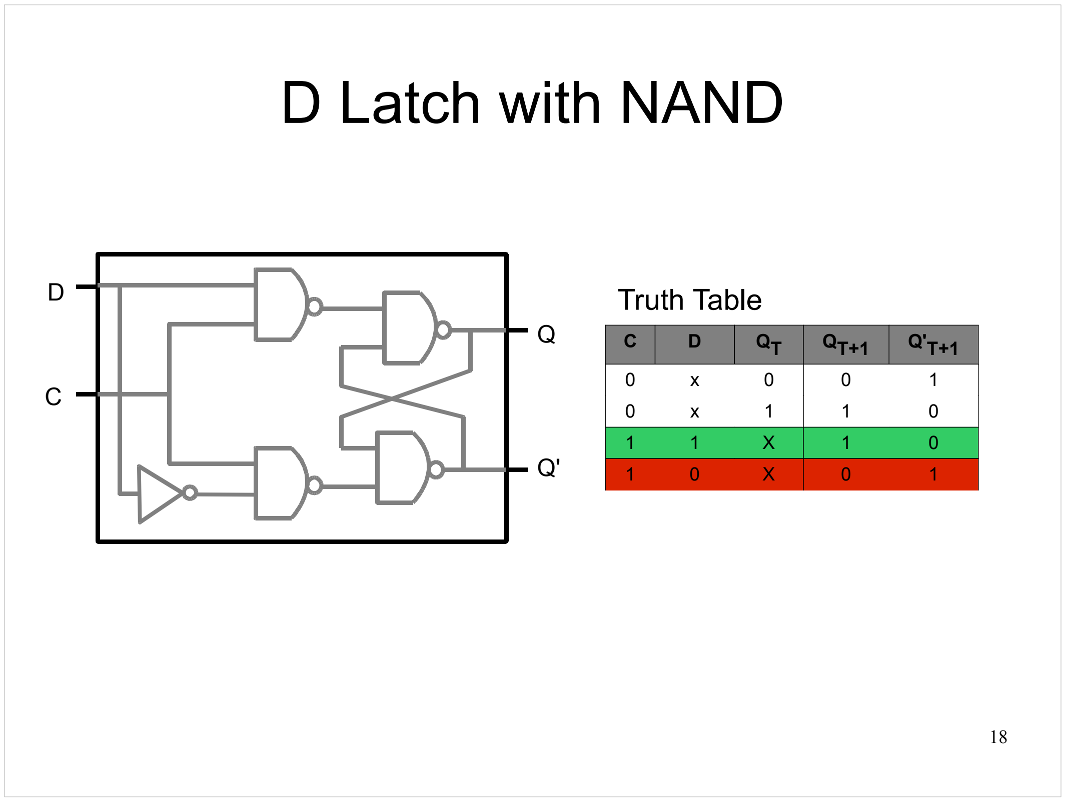
| C | S | R | $Q_T$ | $Q_{T+1}$ | $Q'_{T+1}$ |
|---|---|---|---|---|---|
| 0 | x | x | 0 | 0 | 1 |
| 0 | x | x | 1 | 1 | 0 |
| 1 | 1 | 0 | X | 1 | 0 |
| 1 | 0 | 1 | X | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | X | 1 | 1 |

17

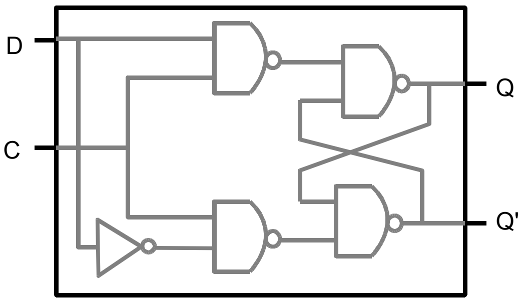
- Further a control input C is introduced. In this example, if C=0, latch holds its previous value. If C=1, latch will store the incoming value.

- Observation is that with this control C, this gate becomes SR latch (in place of S'R' latch in slide 16).

- Another observation is that there are two configurations for holding data (C=0, S=X, R=X; and other as C=1, S=0, R=0). We can exploit this fact to eliminate race configuration (C=1, S=1, R=1) by eliminating hold configuration of (C=1, S=0, R=0). In that case, S and R are in inverse relation always.



## D Latch with NAND



Truth Table

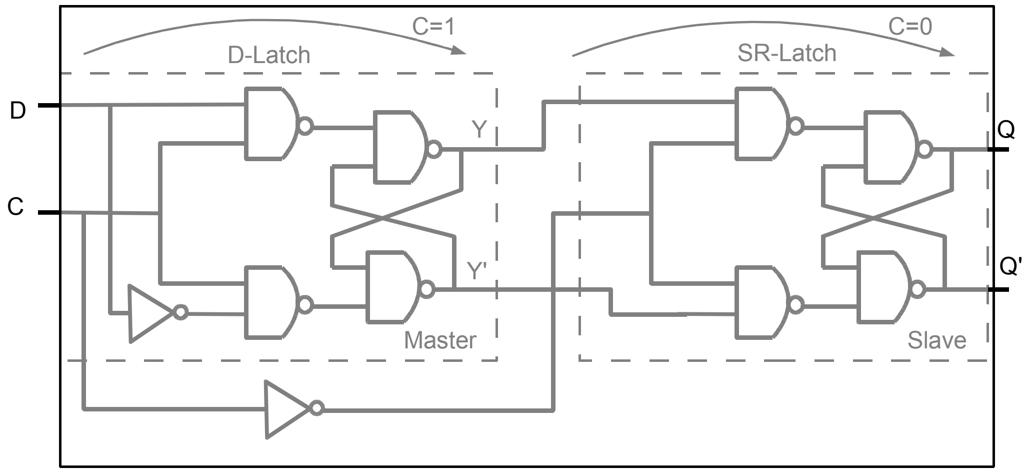
| C | D | $Q_T$ | $Q_{T+1}$ | $Q'_{T+1}$ |
|---|---|---|---|---|
| 0 | x | 0 | 0 | 1 |
| 0 | x | 1 | 1 | 0 |
| 1 | 1 | X | 1 | 0 |
| 1 | 0 | X | 0 | 1 |

18

- For a simple SR latch (slide 16), we need to keep both S and R input since, data holding configuration needs this two variables to be at 0 (S=0, R=0). S and R are complimentary to each other for set and reset operation. However, with SR latches having control variable C (slide 17), it gives two data hold configuration. One data hold configuration is C=0 and other is C=1, S=0, R=0. Now there is a possibility that we eliminate the C=1, S=0, R=0 data hold configuration. If we do that, then we can just make S and R complimentary to each other by an INV gate.

- In a D-Latch, it is exactly done, what has been described above. Two terminals (S & R) are connected with inverter and fed into rest of the circuit. This eliminates any possibility to put the sequential gate in an indeterministic state.

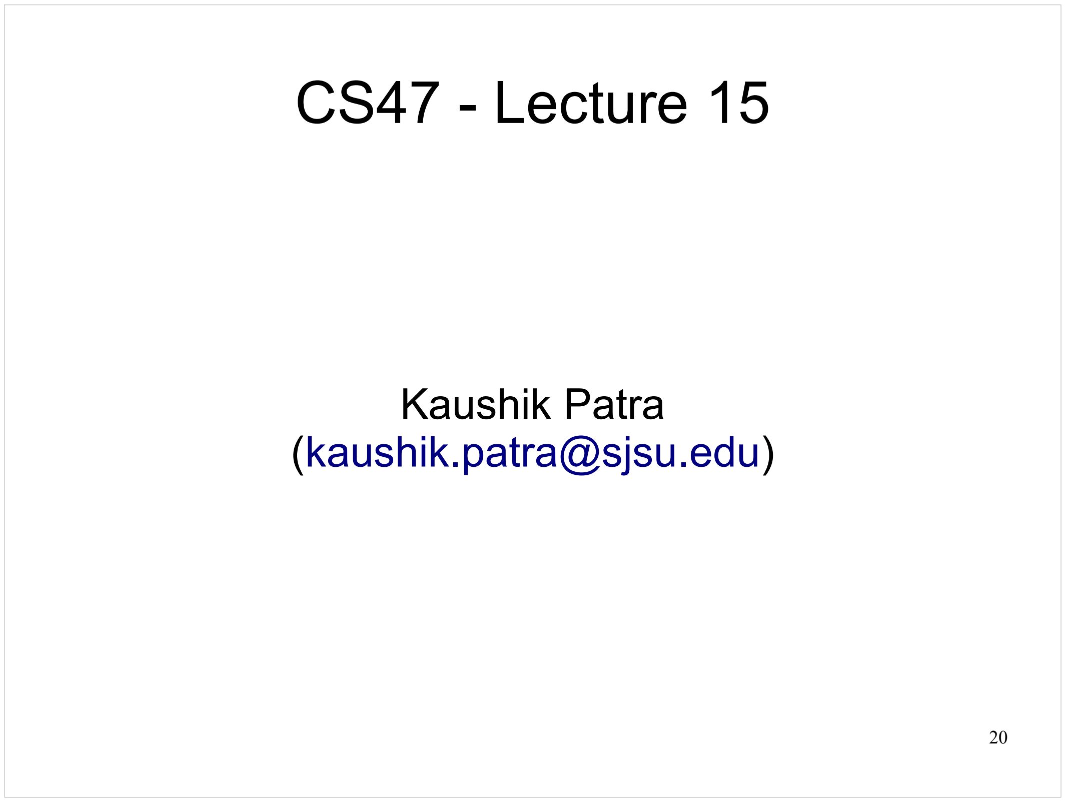
## D FlipFlop



19

- Latch provides a good mechanism to store digital logic value. However, it is prone to data corruption because when C=1, input data will appear to output terminal immediately. To make the gate more stable, two latches are connected back to back with control inverted between them. In that way, incoming logic value is registered at C=1 and then appear at output when C=0. This way, data corruption is avoided. More advanced discussion is done in this matter in CS147.

- The first stage latch is called master latch and the second stage is called slave. This overall flipflop configuration is known as 'master-slave' filpflop.



# CS47 - Lecture 15

Kaushik Patra
(kaushik.patra@sjsu.edu)

20

- Combinational Logic Gates
- Storage Elements

*[ Chapter 3 of Logic & Computer Design Fundamentals, 4th Edition, M. Morris Mano, Charles R. Kime ]*