Lecture -1a
Logic Gates

Prepared by:

Dr. Shahriyar Masud Rizvi

Arajit Saha

Logic Gates

- Logic gates are the basic building blocks of any digital system.
- A logic gate can have one input or multiple inputs. However, they will have only one output.
- The relationship between the input(s) and the output is based on a certain Boolean logic function.
- Digital logic (Boolean logic) operates on the basis of whether a logical function is TRUE or FALSE. Digital logic is implemented by logic gates.
- When a logic function is TRUE, the output of the corresponding logic gate is in ON state or HIGH state.
- Similarly, when a logic function is FALSE, the output of the corresponding logic gate is in OFF state or LOW state.
- In HIGH state, the output of the logic gate receives full supply voltage (Vcc or Vdd). In LOW state, the output maintains the ground voltage, which is zero.
- Since digital logic can have only two states (HIGH or LOW), they can be described by binary number system where the available numbers are 0 and 1.
- HIGH state would correspond to a Logic-1 or just 1. LOW state would correspond to a Logic-0 or just 0.

Logic Gates

- The gates are named based on the logic function performed. For example, the AND gate performs the AND function.
- Logic gates can perform either unary operation (operation on single input) such as NOT or binary operation (operation on 2 inputs) such as AND, OR and XOR.

Basic Logic Gates

- NOT gate
- AND gate
- OR gate

Negative Logic Gates

- NAND gate
- NOR gate

Exclusive Logic Gates

- XOR gate
- XNOR gate

Universal Logic Gates

- NAND gate
- NOR gate

Not Gate (Inverter)

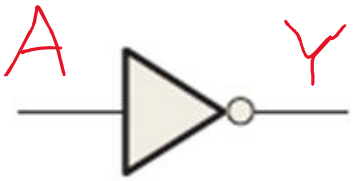
- A NOT gate performs logical inversion/complement operation.
- So, a HIGH input produces LOW output, while a LOW input produces HIGH output.

A	Y
TRUE	FALSE
FALSE	TRUE

A	Y
Vcc	Gnd
Gnd	Vcc

A	Y
HIGH	LOW
LOW	HIGH

A	Y
1	0
0	1

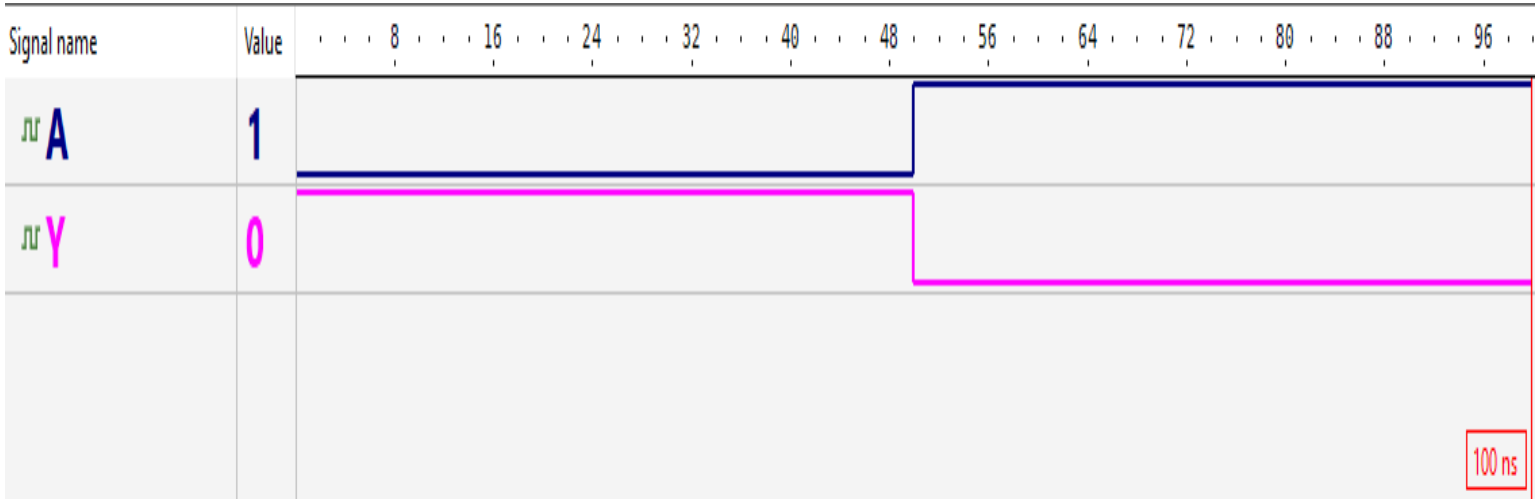


Boolean Logic → Logic Circuit → Logic States (Switches) → Binary representation

Verilog HDL code for the NOT gate

```
module my_not (input wire A,
               output wire Y);

not (Y, A);

endmodule
```

Functional Simulation of the Verilog HDL code for the NOT gate



Hardware Description Languages (HDL)

- Hardware Description Languages (HDL) allows designers to describe both behavior (what does the circuit do functionally) and structure (how it is constructed from smaller sub-systems) of a digital logic circuits, which can be verified by an HDL simulator.
- Note that HDL languages describe hardware, not any instructions (programs) for hardware.
- HDL languages execute code-segments (processes) in parallel.

- On the other hand, programming languages, which are used to instruct/program microprocessors/microcontrollers, execute statements serially (one after another).

- Hardware blocks have no relative precedence, no one block is more or less important in terms of when to activate. A block does not wait for another block to finish computation to "turn on". They all have to operate at the same time regardless of the fact that some will receive inputs earlier and some will receive them later.

- The 3 standard HDL languages are
  - Verilog HDL (IEEE standard 1364)
  - VHDL (IEEE Standard 1076)
  - SystemVerilog (IEEE Standard 1800)

Electronic Design Automation (EDA)

- Electronic Design Automation (EDA) tools automates various stages of the design process for digital circuits.
- EDA tools allow HDL descriptions written at Register Transfer Level (RTL) to be realized in hardware through certain automated steps such as Logic Synthesis and Place and Route (PnR).

RTL Modeling

- All digital design are now described at Register Transfer Level (RTL) with an HDL language first. This description is largely behavioral. In other words, here, sub-systems are generally modeled behaviorally using control structures such as *if* and *case* structures and arithmetic operators such as +, - for combinational logic and clock-edge-sensitive code for sequential logic.

Logic Synthesis

- The verified HDL code is then taken through logic synthesis that automatically translates this description into a more detailed gate-level description utilizing available logic cells of selected technology library.

Place and Route (PnR)

- After the synthesized gate-level description is ready, it is automatically placed in the layout of the chosen target chip/IC and then the interconnections between sub-systems are routed.

Device Configuration or Fabrication

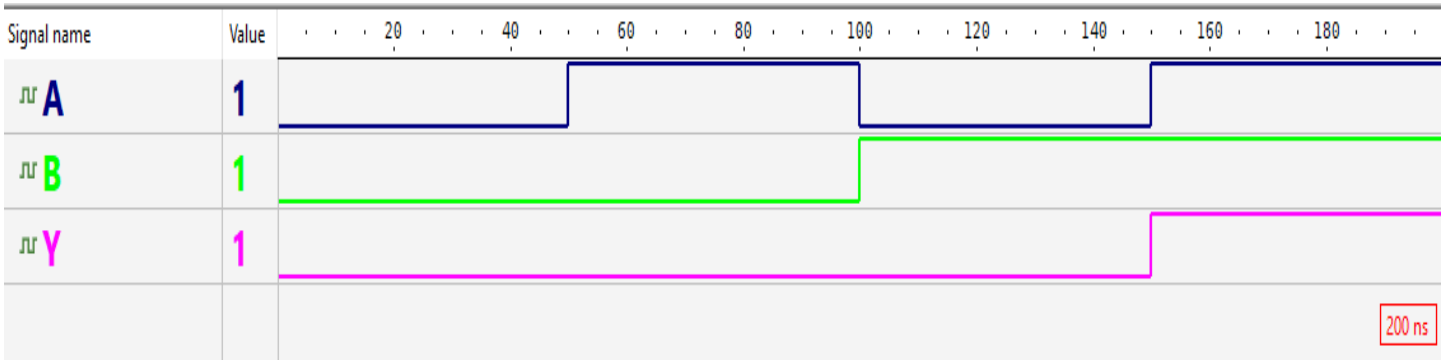
- Note that after PnR, the layout is either configured on a fully-fabricated (but reconfigurable) Field Programmable Logic Device (FPLD)* or sent to fabrication to be fabricated as an Application Specific Integrated Circuit (ASIC).
- *such as a Field Programmable Gate Array (FPGA)

AND gate

- A 2-input AND function produces a TRUE output when both the inputs are TRUE. Otherwise, output is FALSE.
- An n-input AND function produces a TRUE output when all the inputs are TRUE. Otherwise, output is FALSE.
- So, AND gate produces a 1 in the output when **all the inputs are at 1.** Otherwise, the output is 0.

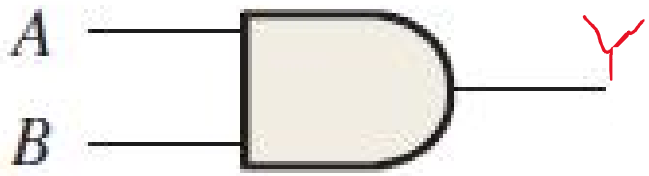
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

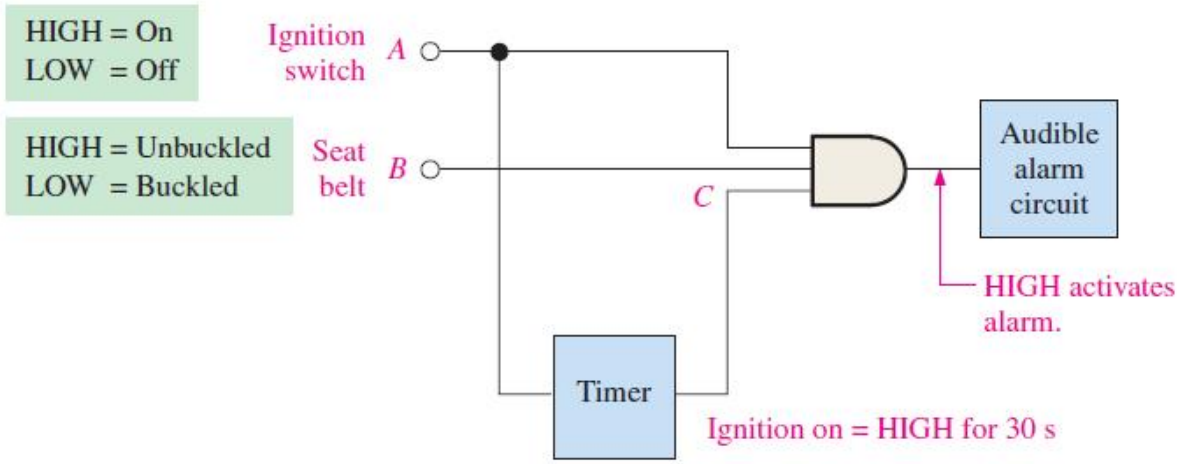
```
module and_2_to_1 (input wire A, B,
                   output wire Y);

and (Y, A, B);

endmodule
```



AND Gate

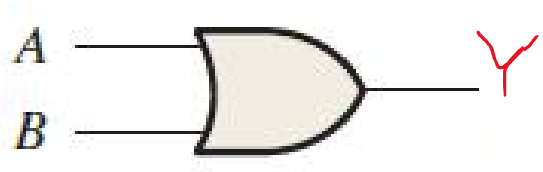
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1





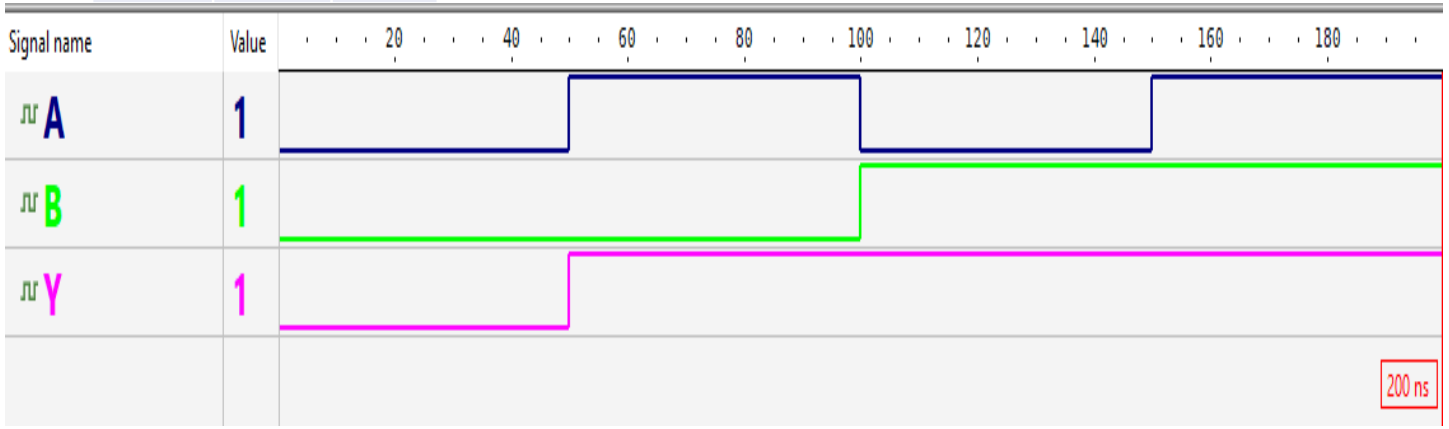
OR gate

- A 2-input OR function produces a TRUE output when any of the two inputs is TRUE. Otherwise, output is FALSE.
- An n-input OR function produces a TRUE output when any of the inputs is TRUE. Otherwise, output is FALSE.
- So, OR gate produces a 1 in the output when **any of the the inputs are at 1.** Otherwise, the output is 0.

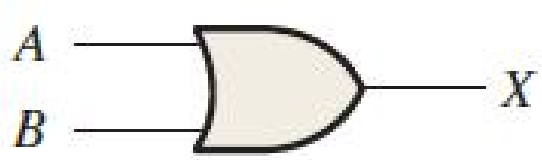


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

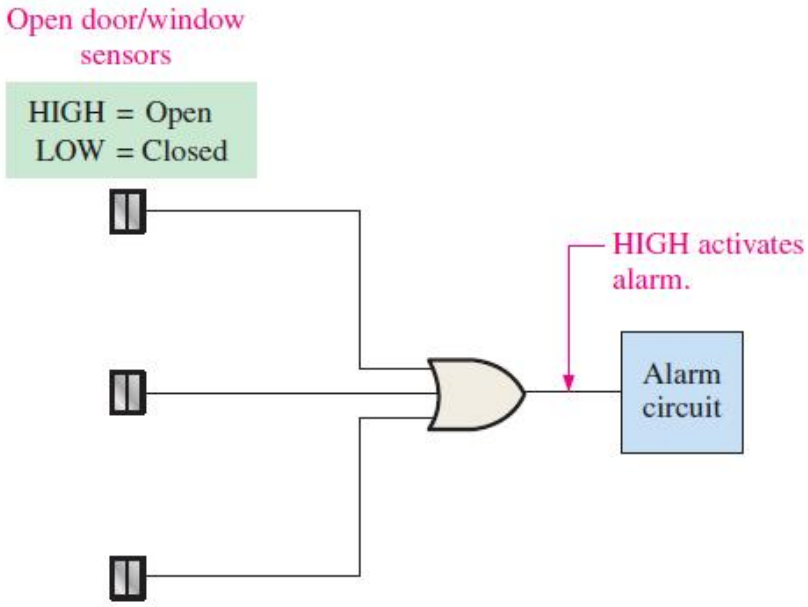
```
module or_2_to_1 (input wire A, B,
                  output wire Y);

or (Y, A, B);

endmodule
```



OR gate

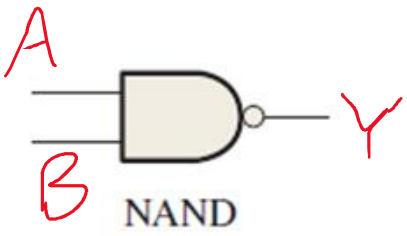


A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

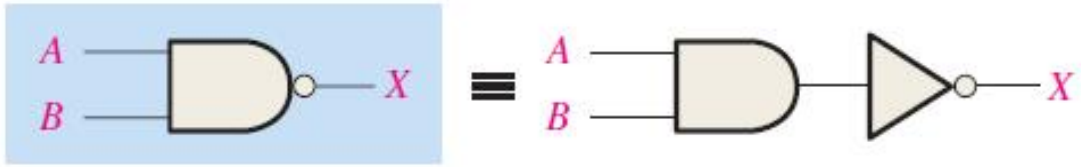


NAND gate

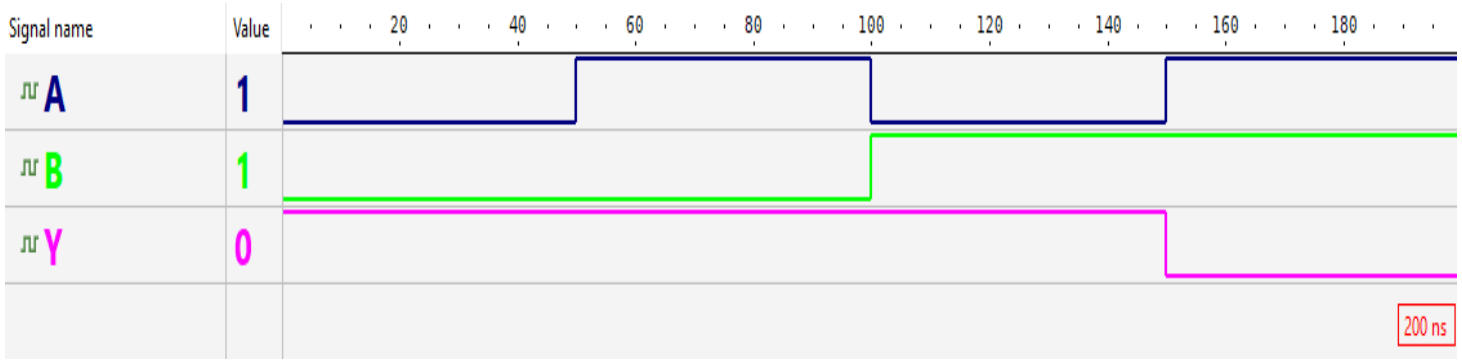
- A 2-input NAND function produces a TRUE output when one the two inputs is FALSE. Otherwise, output is FALSE.
- An n-input NAND function produces a TRUE output when any of the inputs is FALSE. Otherwise, output is FALSE.
- So, NAND gate produces a 1 in the output when **any of the inputs is at 0**. Otherwise, the output is 0.
- A NAND operation is the opposite (compliment) of AND operation.



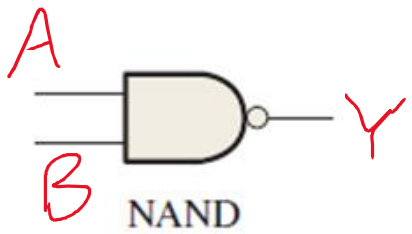
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



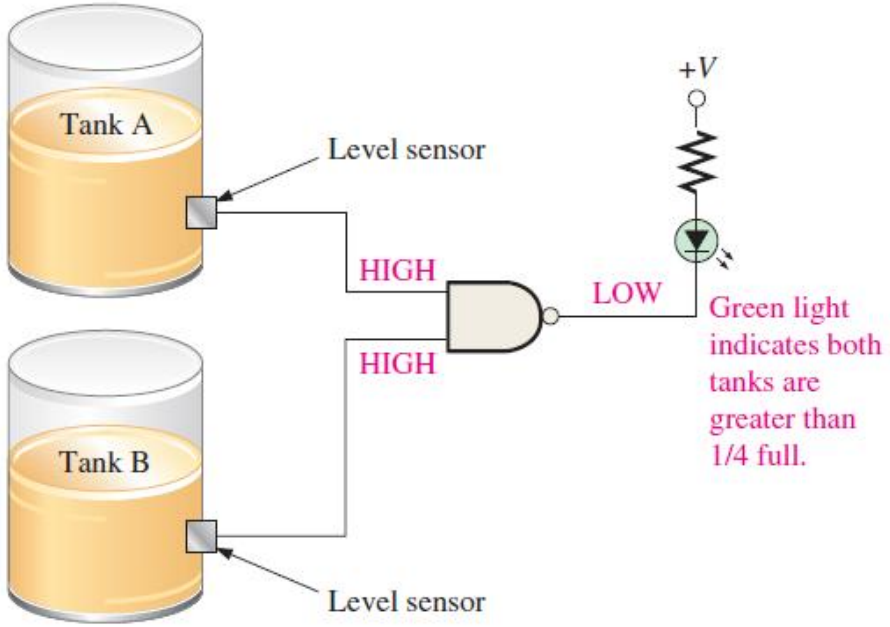

```
module nand_2_to_1 (input wire A, B,
                    output wire Y);

nand (Y, A, B);

endmodule
```



NAND gate

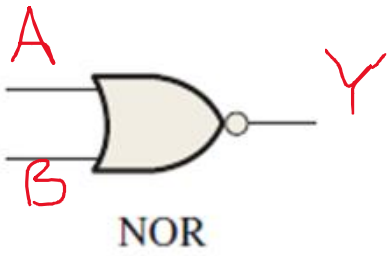


A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

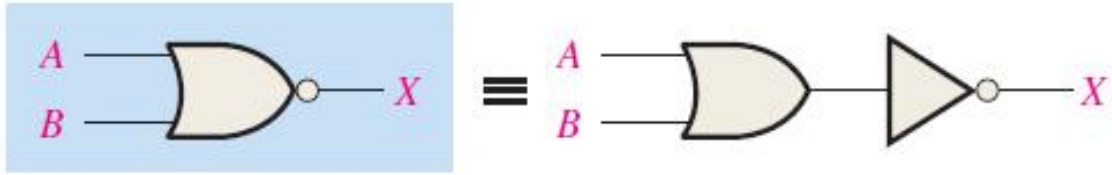


NOR gate

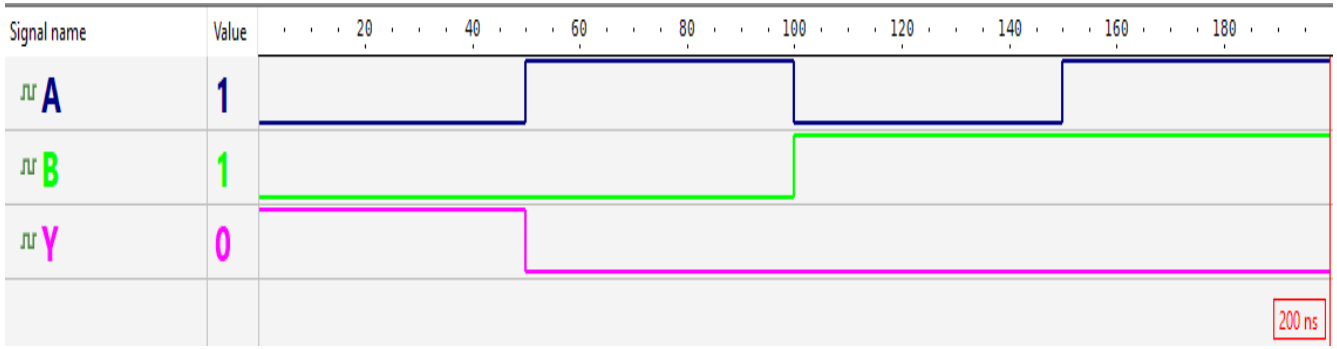
- A 2-input NOR function produces a TRUE output when both inputs are FALSE. Otherwise, output is FALSE.
- An n-input NOR function produces a TRUE output when all the inputs are FALSE. Otherwise, output is FALSE.
- So, NOR gate produces a 1 in the output when **all the the inputs are at 1**. Otherwise, the output is 0.
- A NOR operation is the opposite (compliment) of OR operation.

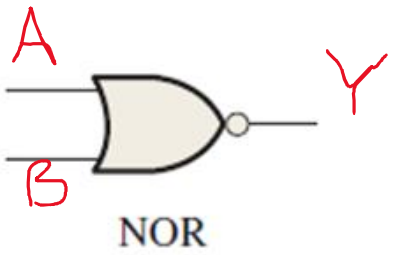


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

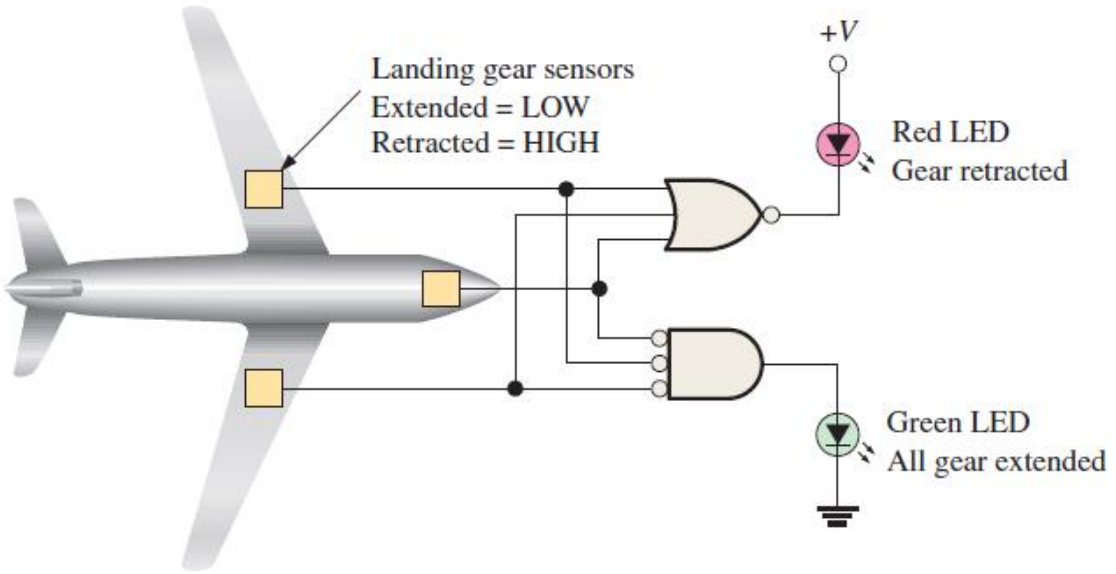



```
module nor_2_to_1 (input wire A, B,
                   output wire Y);

nor (Y, A, B);

endmodule
```





A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

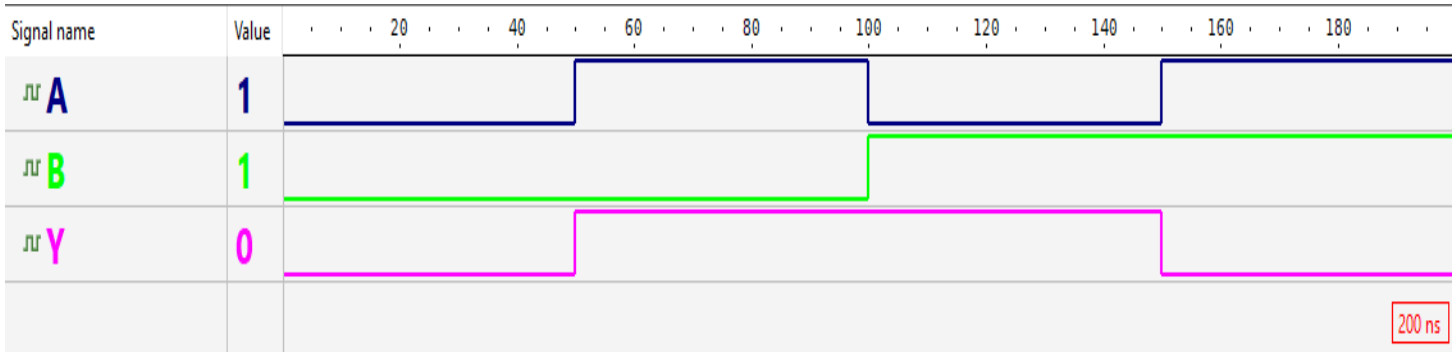


XOR gate

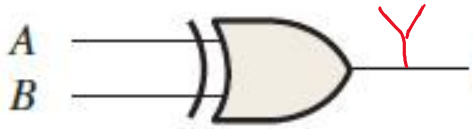
- A 2-input XOR function produces a TRUE output when **one and only one of the two** inputs is TRUE. Otherwise, output is FALSE.
- XOR operation for more than 2 inputs is not well-defined.
- In one interpretation, an n-input XOR function produces a TRUE output when **odd number of inputs** is TRUE. Otherwise, output is FALSE.
- So, XOR gate produces a 1 in the output when **odd number of inputs is at 1**. Otherwise, the output is 0.

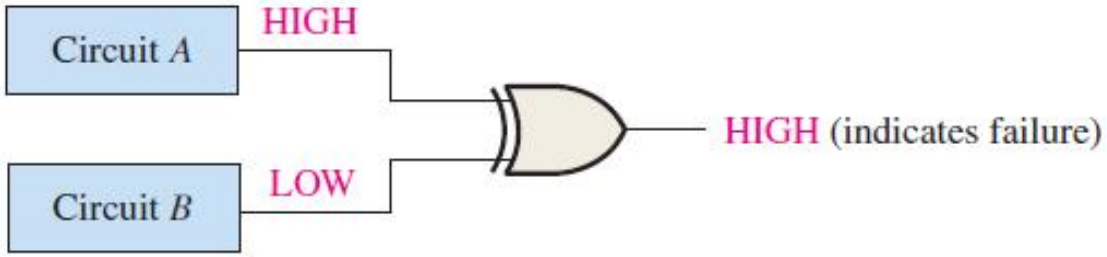
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

```
module xor_2_to_1 (input wire A, B,
                   output wire Y);

xor (Y, A, B);

endmodule
```



XOR gate





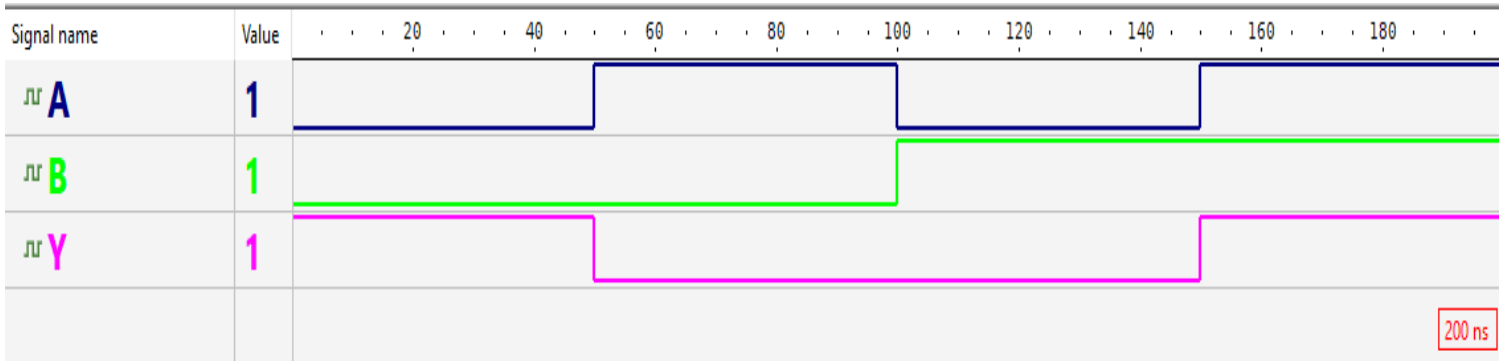
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR gate

- A 2-input XNOR function produces a TRUE output when **both** inputs are the same (both TRUE or both FALSE). Otherwise, output is FALSE.
- XNOR operation for more than 2 inputs is not well-defined.
- For even numbered inputs, XNOR gate produces a 1 in the output when **even number of inputs is at 1 or all the inputs are at 0**. Otherwise, the output is 0.
- Note that XNOR is not necessarily compliment of XOR. Try computing XOR and XNOR outputs for 3 inputs.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

```
module xnor_2_to_1 (input wire A, B,
                    output wire Y);

xnor (Y, A, B);

endmodule
```

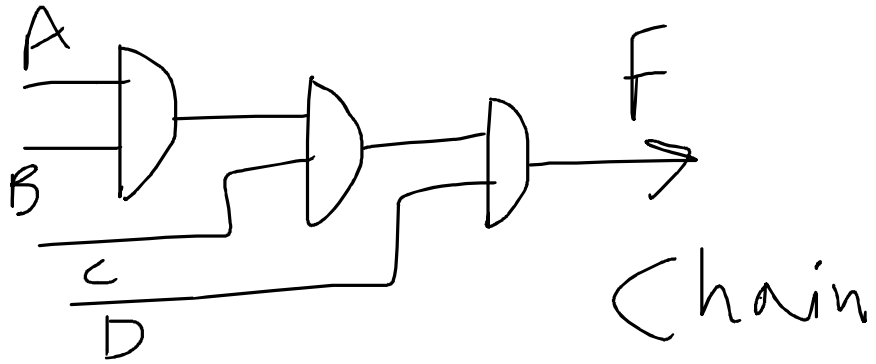


What conditions can be represented with each logic gates?

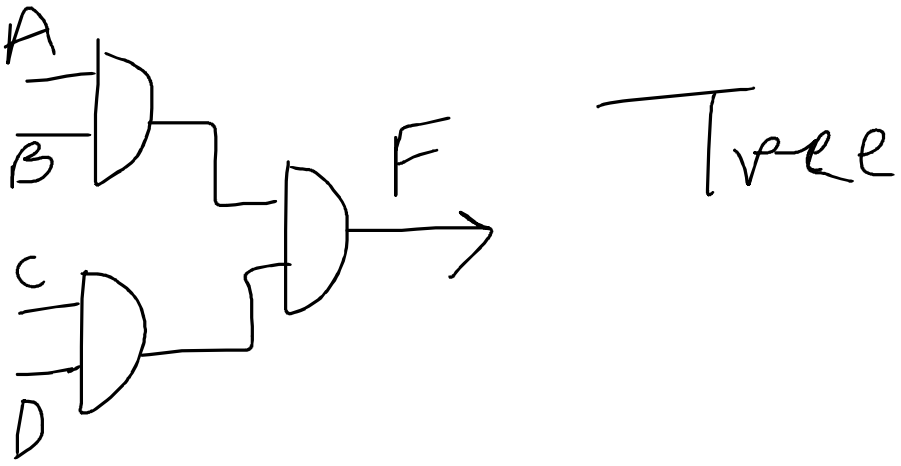
- All inputs are HIGH
- At least one input is HIGH
- Inputs are equal
- Inputs are unequal
- Even parity
- Odd parity

Chain vs Tree structure

- Consider a logical operation with 4 variables. F = A . B. C .D.
  Each gate has a delay of 1 unit. Compute output delay of each case.




```
module and_chain(input wire A, B, C, D,
                 output wire F);

assign F = ( ( (A & B) & C) & D);

endmodule
```




```
module and_tree(input wire A, B, C, D,
                output wire F);

assign F = (A & B) & (C & D);

endmodule
```

Verilog Logical Operators and Gate Primitive (pre-defined gates)

Logical Function	Gate Primitives	Logical (Boolean) Operators
not/inversion/complementation	not	~
and	and	&
or	or	|
xor/exor	xor	^
nand	nand	~&
nor	nor	~|
xnor/exnor	xnor	~^

References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

Thank You