

rakisha1302 / study-of-basic-gates

🔍

📁

👤

<> Code

🔗 Pull requests

🎬 Actions

📅 Projects

📖 Wiki

🛡 Security

📈 Insights

⚙ Settings

👁

🔗

★

★ 0 stars

🔗 1.9k forks

👁 0 watching

🔗 Branches

📈 Activity

🏷 Tags

🌐 Public repository · Forked from [naavaneetha/study-of-basic-gates](#)

🔗

🔗 1 Branch

🏷 0 Tags

🔗

🏷

🔍 Go to file

t

Go to file

Add file +

Code

⋮

This branch is 1 commit ahead of [naavaneetha/study-of-basic-gates:main](#) .

🔗 Contribute ▾

🔄 Sync fork ▾

👤

rakisha1302

Update README.md

66f0c27 · 4 minutes ago 🕒

📁 db	project created	last year
📁 hc_output	project created	last year
📁 incremental_db	project created	last year
📁 output_files	project created	last year
📁 simulation	project created	last year
📄 GATES_TT.qpf	project created	last year
📄 GATES_TT.qsf	project created	last year
📄 GATES_TT.v	project created	last year
📄 GATES_TT.v.bak	project created	last year
📄 README.md	Update README.md	4 minutes ago
📄 Waveform1a.vwf	project created	last year
📄 c5_pin_model_dump.txt	project created	last year
📄 cio_dump_disallowed_lists.e...	project created	last year

study-of-basic-gates

AIM:

To study and verify the truth table of logic gates in Quartus II using Verilog programming.

Equipments Required:

Software – Quartus prime

Theory

Introduction Logic gates are the basic building blocks of any digital system. Logic gates are electronic circuits having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as

AND gate OR gate NOT gate NAND gate NOR gate Ex-OR gate Ex-NOR gate

AND gate

The AND gate is an electronic circuit that gives a high output (1) only if all its inputs are high. A dot (.) is used to show the AND operation i.e. $A.B$ or can be written as AB $Y = A.B$

OR gate

The OR gate is an electronic circuit that gives a high output (1) if one or more of its inputs are high. A plus (+) is used to show the OR operation. $Y = A+B$

NOT gate

The NOT gate is an electronic circuit that produces an inverted version of the input at its output. It is also known as an inverter. If the input variable is A, the inverted output is known as NOT A. This is also shown as A' or A with a bar over the top, as shown at the outputs. $Y = A'$

README

gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion. $Y = (AB)'$

NOR gate

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion. $Y = (A+B)'$

Ex-OR gate

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both of its two inputs are high. An encircled plus sign (\oplus) is used to show the Ex-OR operation. $Y = A \oplus B$


Ex-NOR gate

The 'Exclusive-NOR' gate circuit does the opposite to the EX-OR gate. It will give a low output if either, but not both of its two inputs are high. The symbol is an EX-OR gate with a small circle on the output. The small circle represents inversion. $Y = A \oplus B$

Procedure

1. Type the program in Quartus software.
2. Compile and run the program.
3. Generate the RTL schematic and save the logic diagram.
4. Create nodes for inputs and outputs to generate the timing diagram.
5. For different input combinations generate the timing diagram.

PROGRAM


Program for logic gates and verify its truth table in quartus using Verilog programming 

Developed by: Rakisha.R
RegisterNumber: 25008669



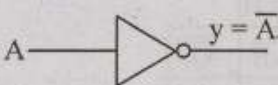
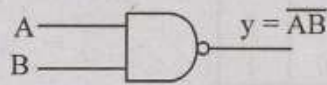
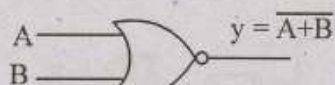

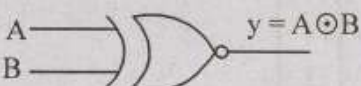
```
module Gate_1 (
    input wire a,      // Input A
    input wire b,      // Input B
    output wire and_out,
    output wire or_out,
    output wire not_out, // only on A
    output wire nand_out,
    output wire nor_out,
    output wire xor_out,
    output wire xnor_out
);

    assign and_out  = a & b;      // AND gate
    assign or_out   = a | b;      // OR gate
    assign not_out  = ~a;         // NOT gate (on input A)
    assign nand_out = ~(a & b);   // NAND gate
    assign nor_out  = ~(a | b);   // NOR gate
    assign xor_out  = a ^ b;      // XOR gate
    assign xnor_out = ~(a ^ b);   // XNOR gate

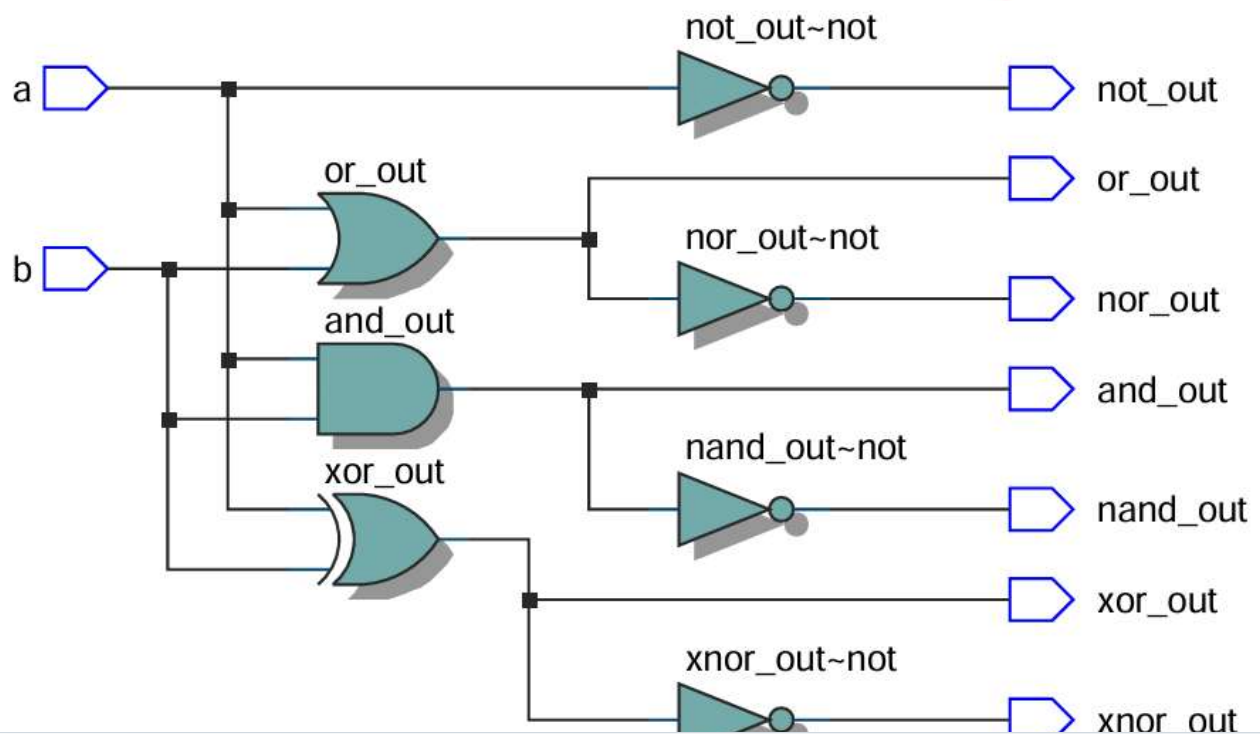
endmodule
```



Logic symbol & Truthtable

Logical gate	Truth table															
<p>AND gate</p> 	<table><tr><th>A</th><th>B</th><th>$y=AB$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$y=AB$	0	0	0	0	1	0	1	0	0	1	1	1
A	B	$y=AB$														
0	0	0														
0	1	0														
1	0	0														
1	1	1														
<p>OR gate</p> 	<table><tr><th>A</th><th>B</th><th>$y=A+B$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$y=A+B$	0	0	0	0	1	1	1	0	1	1	1	1
A	B	$y=A+B$														
0	0	0														
0	1	1														
1	0	1														
1	1	1														
<p>NOT gate</p> 	<table><tr><th>A</th><th>$y=\bar{A}$</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	$y=\bar{A}$	0	1	1	0									
A	$y=\bar{A}$															
0	1															
1	0															
<p>NAND gate</p> 	<table><tr><th>A</th><th>B</th><th>$y=\overline{AB}$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$y=\overline{AB}$	0	0	1	0	1	1	1	0	1	1	1	0
A	B	$y=\overline{AB}$														
0	0	1														
0	1	1														
1	0	1														
1	1	0														
<p>NOR gate</p> 	<table><tr><th>A</th><th>B</th><th>$y=\overline{A+B}$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$y=\overline{A+B}$	0	0	1	0	1	0	1	0	0	1	1	0
A	B	$y=\overline{A+B}$														
0	0	1														
0	1	0														
1	0	0														
1	1	0														
<p>EX-OR gate</p> 	<table><tr><th>A</th><th>B</th><th>$y=A \oplus B$</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$y=A \oplus B$	0	0	0	0	1	1	1	0	1	1	1	0
A	B	$y=A \oplus B$														
0	0	0														
0	1	1														
1	0	1														
1	1	0														
<p>EX-NOR gate</p> 	<table><tr><th>A</th><th>B</th><th>$y=A \odot B$</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$y=A \odot B$	0	0	1	0	1	0	1	0	0	1	1	1
A	B	$y=A \odot B$														
0	0	1														
0	1	0														
1	0	0														
1	1	1														

RTL realization Output:



Releases

No releases published

[Create a new release](#)

Packages

No packages published

[Publish your first package](#)

Languages

● VHDL 54.4% ● HTML 14.9% ● Stata 14.7% ● Verilog 13.3% ● Standard ML 2.7%

Suggested workflows

Based on your tech stack



SLSA Generic generator

Generate SLSA3 provenance for your existing release workflows

Configure



Jekyll using Docker image

Configure

Package a Jekyll site using the jekyll/builder Docker image.

[More workflows](#)

[Dismiss suggestions](#)