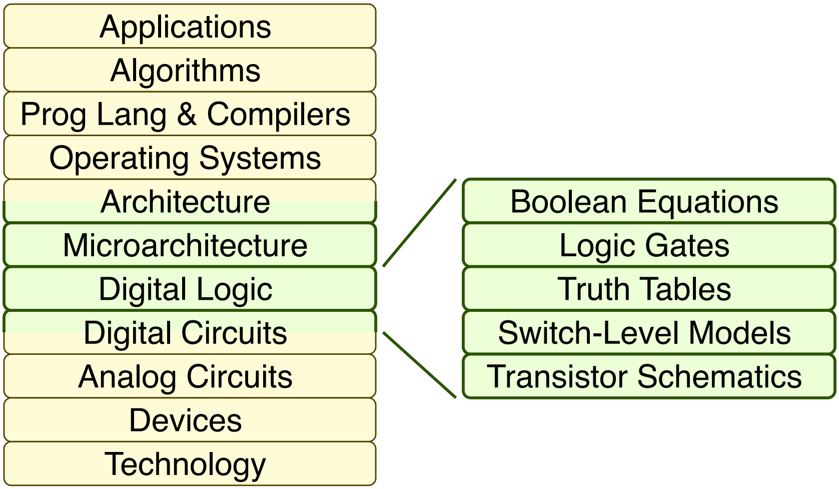
ECE 2300 Digital Logic and Computer Organization Fall 2024

Topic 2: Logic Gates

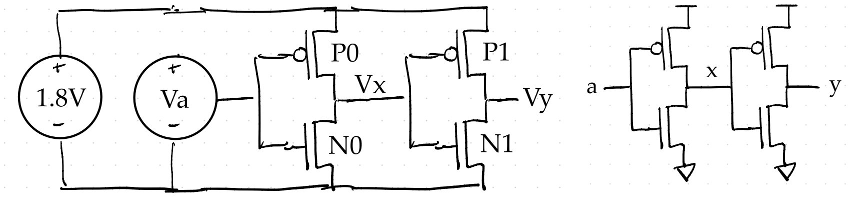
School of Electrical and Computer Engineering
Cornell University

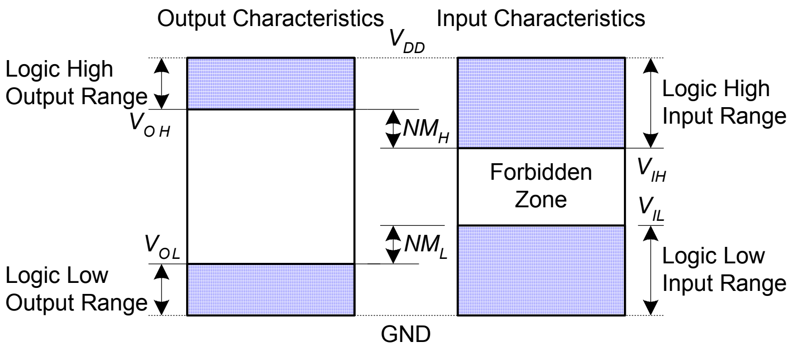
revision: 2024-09-05-09-17


Applications
Algorithms
Prog Lang & Compilers
Operating Systems
Architecture
Microarchitecture
Digital Logic
Digital Circuits
Analog Circuits
Devices
Technology
Boolean Equations
Logic Gates
Truth Tables
Switch-Level Models
Transistor Schematics

1. From Voltages to Bits

- Abstract specific voltage levels to be logic 0 and logic 1
- Include noise margins to make our circuits more robust
- Enables ignoring specific voltage levels at higher abstraction levels
- Can now interpret wires as representing *binary digits* (bits)

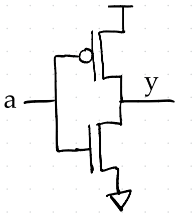




Logic Family	V_{DD}	V_{IL}	V_{IH}	V_{OL}	V_{OH}
TTL	5 (4.75–5.25)	0.8	2.0	0.4	2.4
CMOS	5 (4.5–6)	1.35	3.15	0.33	3.84
LVTTL	3.3 (3–3.6)	0.8	2.0	0.4	2.4
LVCMOS	3.3 (3–3.6)	0.9	1.8	0.36	2.7

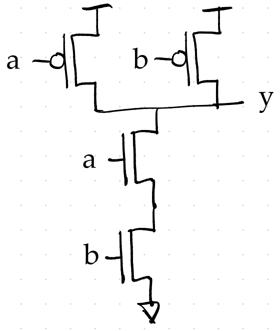
2. From Digital Circuits to Truth Tables

- A *truth table* specifies the output values for specific input values, with one row for every possibly combination of input values
- Let's start by deriving a truth table from a digital circuit



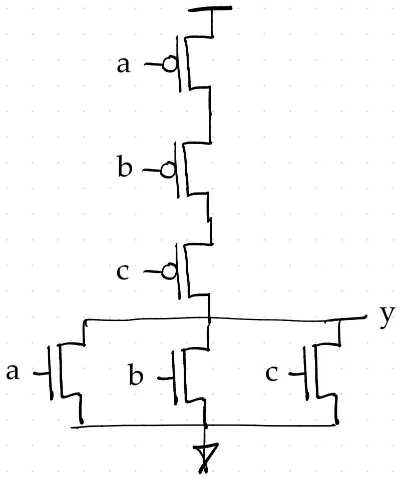
Va	**Vy**
0V	
1.8V	

a	**y**
0	
1	

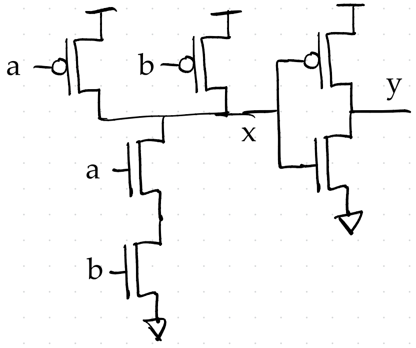


Va	**Vb**	**Vy**
0V	0V	
0V	1.8V	
1.8V	0V	
1.8V	1.8V	

a	**b**	**y**
0	0	
0	1	
1	0	
1	1	



a	b	c	y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



a	b	x	y
0	0		
0	1		
1	0		
1	1		

- Now let's derive a digital circuit from a truth table

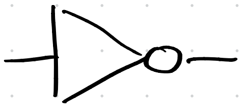
a	**b**	**c**	**y**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

a	**b**	**c**	**y**
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

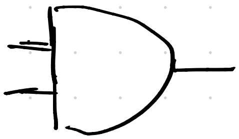
a	**b**	**c**	**y**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

a	**b**	**y**
0	0	0
0	1	1
1	0	1
1	1	1

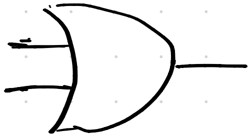
3. From Truth Tables to Logic Gates



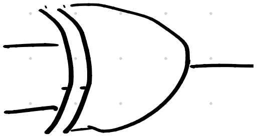
a	y
0	
1	



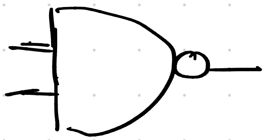
a	b	y
0	0	
0	1	
1	0	
1	1	



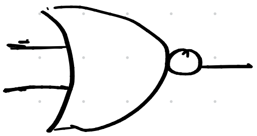
a	b	y
0	0	
0	1	
1	0	
1	1	



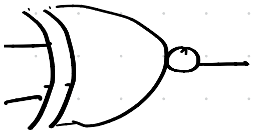
a	b	y
0	0	
0	1	
1	0	
1	1	



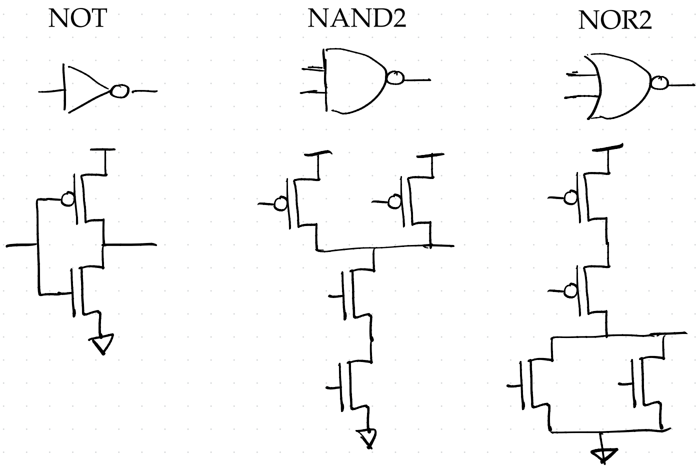
a	b	y
0	0	
0	1	
1	0	
1	1	

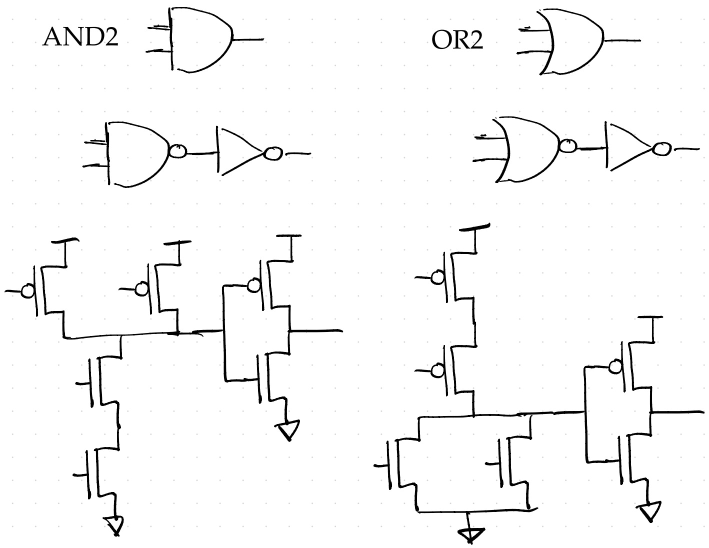


a	b	y
0	0	
0	1	
1	0	
1	1	

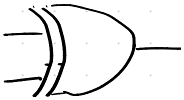


a	b	y
0	0	
0	1	
1	0	
1	1	


NOT
NAND2
NOR2


AND2
OR2

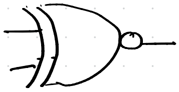
XOR

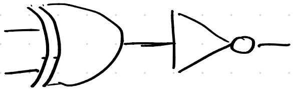


Derive the truth table for the above gate-level network. Verify it matches the truth table for an XOR gate.

a	b	x	z	w	y
0	0				
0	1				
1	0				
1	1				

XNOR





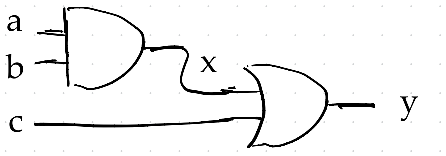
3.1. Primitive Gate Set

- AND2 must be implemented using NAND2 and NOT gate
- OR2 must be implemented using OR2 and NOT gate
- XOR2, XNOR2 gates must be implemented using other logic gates
- NAND2, NOT can be implemented using just NOR2
- NOR2, NOT can be implemented using just NAND2

- More complicated gates are also possible
  - AOI21, AOI22 gates (and-or-inverting)
  - OAI12, OAI22 gates (or-and-inverting)
  - Multiplexors, half-adders, full-adders
  - Flip-flops, latches
- We will use the following "primitive gate set" in the course
  - NOT gate
  - n-input AND, n-input NAND gates
  - n-input OR, n-input NOR gates
  - n-input XOR, n-input XNOR gates

3.2. Gate-Level Networks

Gate-Level Network



Simulation Table

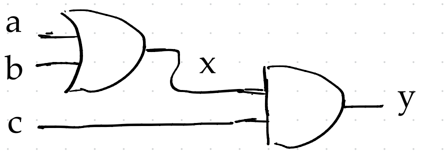
a	b	c	x	y
0	0	0		
1	1	0		
1	0	0		

Truth Table

a	b	c	x	y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Waveform (assume zero delay model)

Gate-Level Network



Truth Table

a	b	c	x	y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Simulation Table

a	b	c	x	y
0	0	0		
0	1	1		
0	1	0		

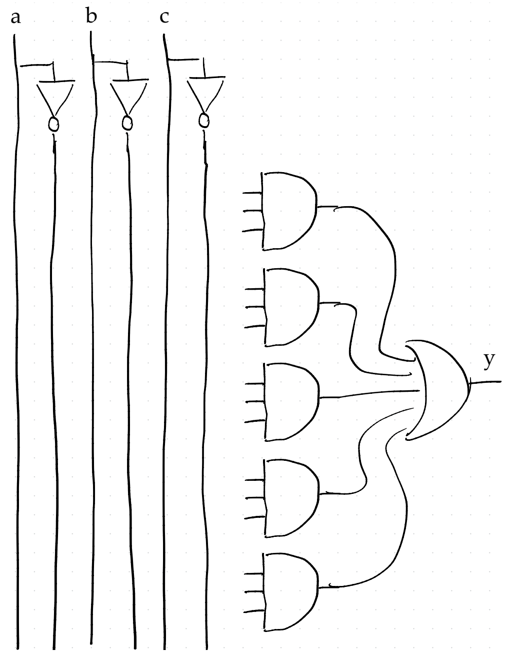
Waveform (assume zero delay model)

Gate-Level Network

Truth Table

a	**b**	**y**
0	0	1
0	1	0
1	0	1
1	1	1

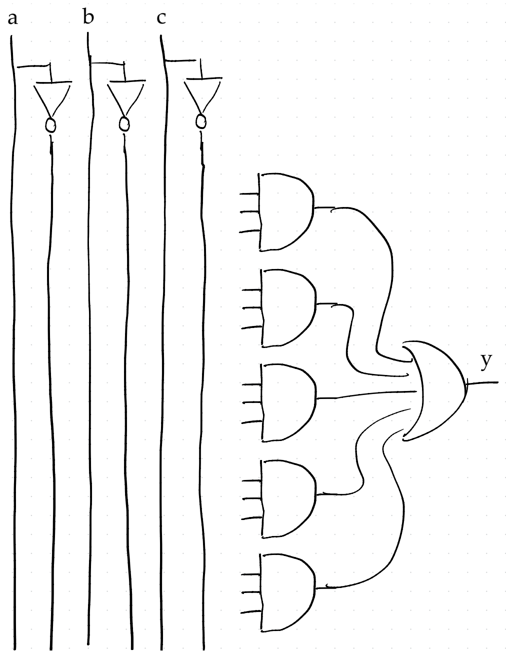
Gate-Level Network



Truth Table

a	**b**	**c**	**y**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Gate-Level Network

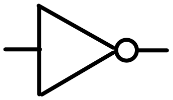


Truth Table

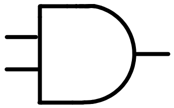
a	**b**	**c**	**y**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

3.3. Modeling Logic Gates with Verilog

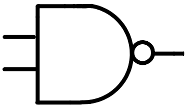
- Declare wires using `wire`
- Instantiate logic gate primitives (e.g., `not`, `and`)
- List wires to connect them to inputs and outputs of logic gate
- Output is always first
- Logic gates with more than two inputs are allowed
- This is not “calling a function”; this is instantiating hardware!



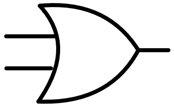
```
wire a;
wire y;
not( y, a );
```



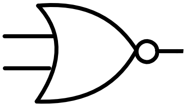
```
wire a;
wire b;
wire y;
and( y, a, b );
```



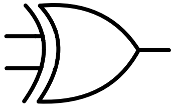
```
wire a;
wire b;
wire y;
nand( y, a, b );
```



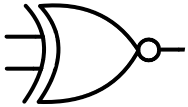
```
wire a;
wire b;
wire y;
or( y, a, b );
```



```
wire a;
wire b;
wire y;
nor( y, a, b );
```

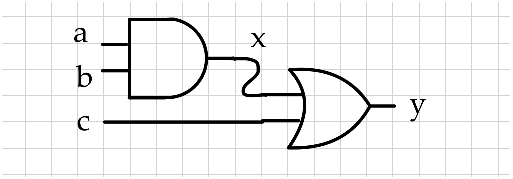


```
wire a;
wire b;
wire y;
xor( y, a, b );
```



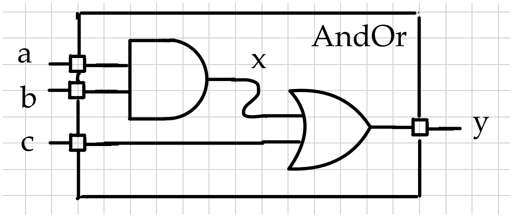
```
wire a;
wire b;
wire y;
xnor( y, a, b );
```

Creating Gate-Level Netlists in Verilog



```
wire a;
wire b;
wire c;
wire x;
wire y;

and( x, a, b );
or ( y, x, c );
```

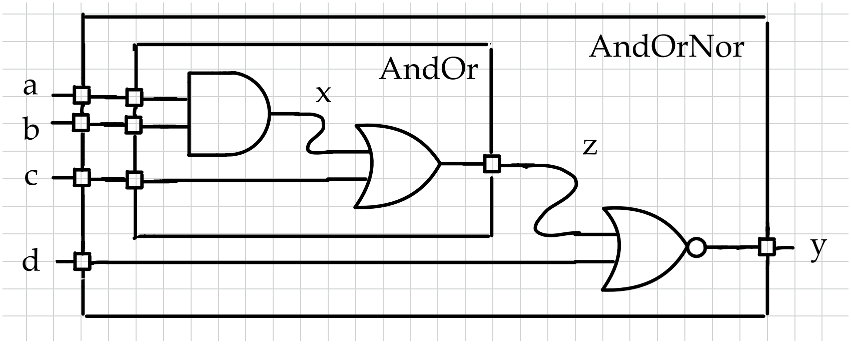
Defining Hardware Modules in Verilog



```
module AndOr
(
  input  wire a,
  input  wire b,
  input  wire c,
  output wire y
);

  wire x;

  and( x, a, b );
  or ( y, x, c );

endmodule
```

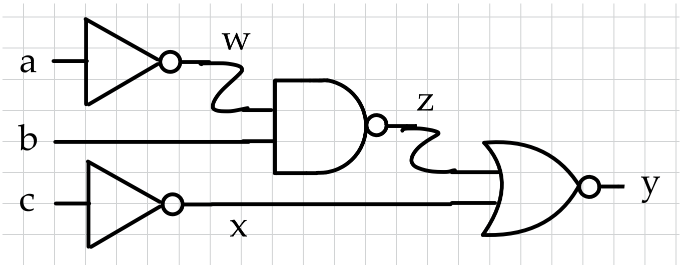
https://www.edaplayground.com/x/KjwN

Instantiating Hardware Modules in Verilog



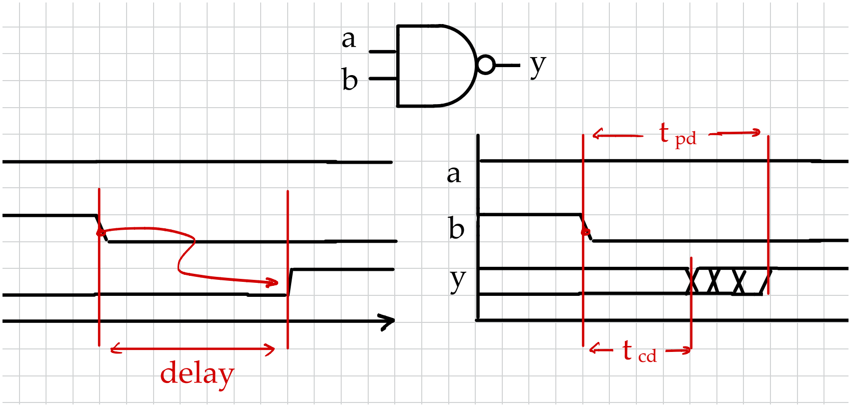
```
module AndOrNor
(
  input  wire a,
  input  wire b,
  input  wire c,
  input  wire d,
  output wire y
);

  wire z;

  AndOr and_or
  (
    .a (a),
    .b (b),
    .c (c),
    .y (z)
  );

  or( y, z, d );

endmodule
```

Implement the following gate-level network in Verilog. Match the coding style from the previous examples.



4. Logic Gate Timing

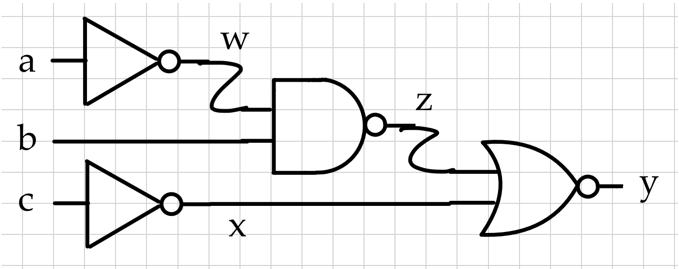
- We will do most of our analysis using abstract units of time (τ)
- We will consider three simple delay models:
  - Zero-delay model (every gate updates it's outputs instantly)
  - Uniform constant delay model (every gate has same delay)
  - Non-uniform constant delay model (gates can have different delays)
- More complex delay models are also possible:
  - Different inputs have different delays
  - Delay of 0 to 1 transition is different from 1 to 0 transition
  - Delay depends on what the gate is driving

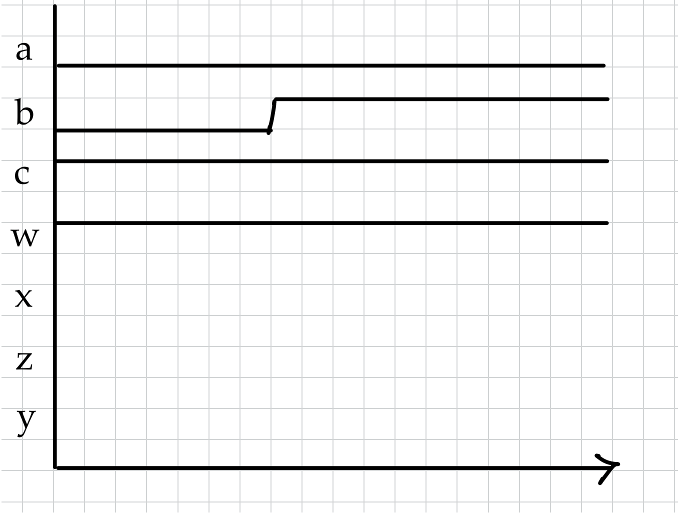


- **Propagation Delay:** (t_{pd}) is maximum time from when any input changes until the output or outputs reach their final value
- **Contamination Delay:** (t_{cd}) is minimum time from when any input changes until any output changes its value

- **Critical Path:** longest path through a gate-level network (only consider propagation delays!)
- **Short Path:** shortest path through a gate-level network (only consider contamination delays!)

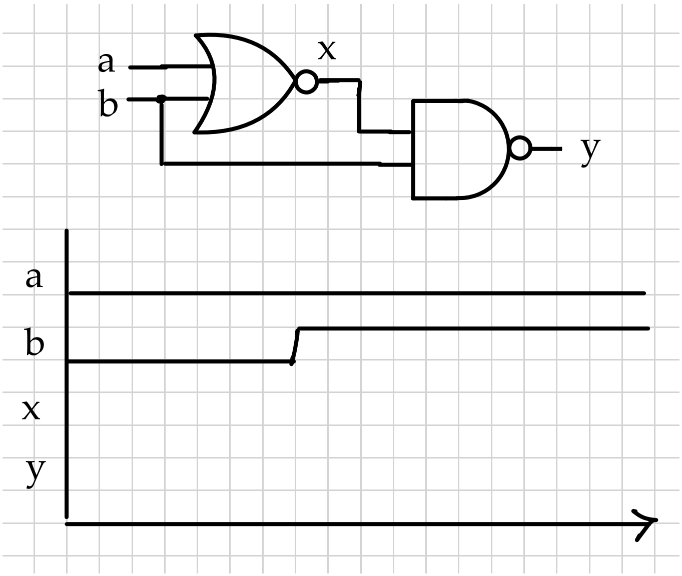
Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ



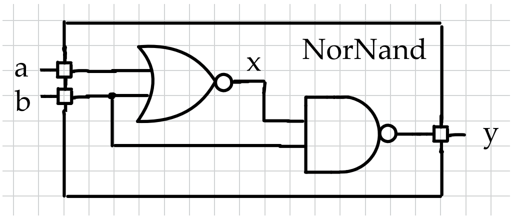


Draw and label the critical path and the short path on the gate-level network. What is the propagation delay of this gate-level network? What is the contamination delay of this gate-level network? Draw waveform.

Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ



Creating Gate-Level Netlists with Delays in Verilog

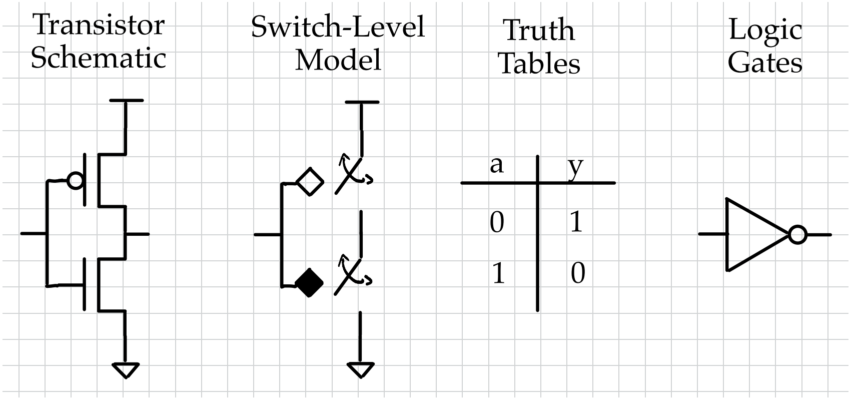


Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ

```
module NorNand
(
  input  wire a,
  input  wire b,
  output wire y
);

  wire x;

  nor  #(3) ( x, a, b );
  nand #(2) ( y, x, b );

endmodule
```

https://www.edaplayground.com/x/Hpjf

5. Summary of Abstractions



a	y
0	1
1	0