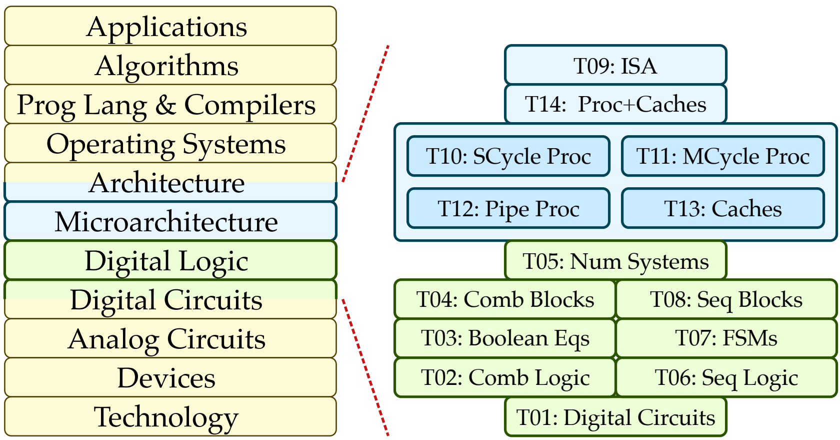
ECE 2300 Digital Logic and Computer Organization
Fall 2025

Topic 2: Combinational Logic

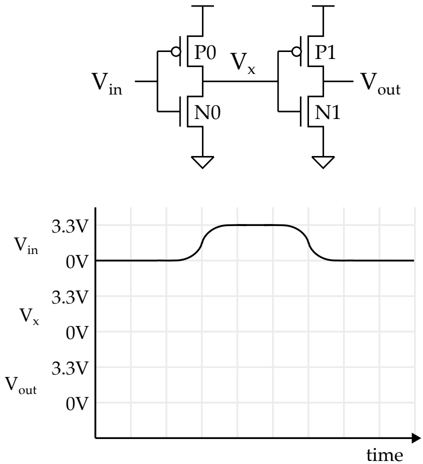
School of Electrical and Computer Engineering
Cornell University

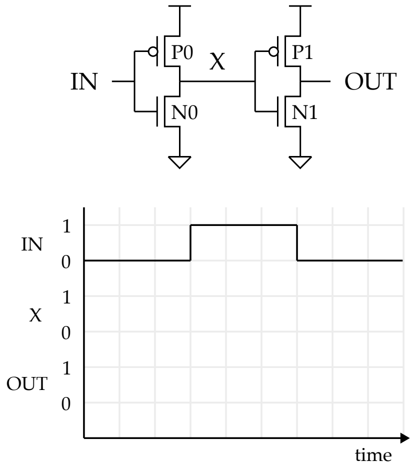
revision: 2025-09-04-10-50

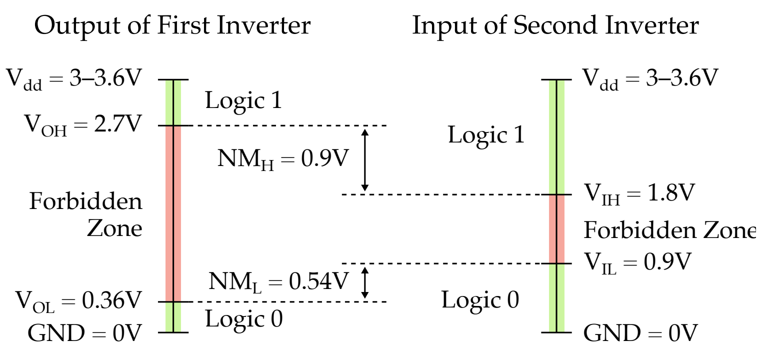

Applications
Algorithms
Prog Lang & Compilers
Operating Systems
Architecture
Microarchitecture
Digital Logic
Digital Circuits
Analog Circuits
Devices
Technology
T09: ISA
T14: Proc+Caches
T10: SCycle Proc
T11: MCycle Proc
T12: Pipe Proc
T13: Caches
T05: Num Systems
T04: Comb Blocks
T08: Seq Blocks
T03: Boolean Eqs
T07: FSMs
T02: Comb Logic
T06: Seq Logic
T01: Digital Circuits

1. From Voltages to Bits

- Abstract 3.3V to be logic 1, and abstract 0V to be logic 0
- Enables ignoring specific voltage levels at higher abstraction levels
- Can now interpret wires as representing *binary digits* (bits)
- Include noise margins to make our circuits more robust

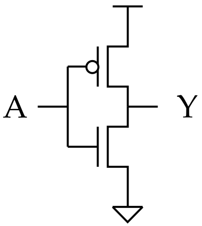






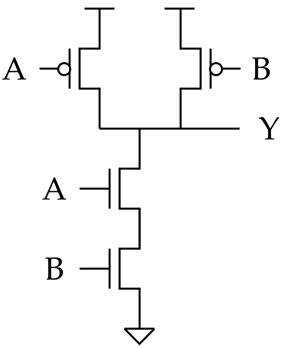
2. From Digital Circuits to Truth Tables

- 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 truth tables from digital circuits



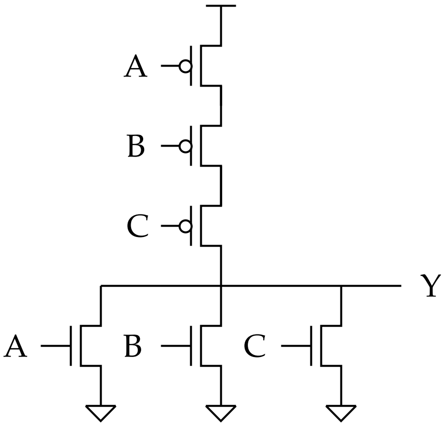
V_a	V_y
0V	
3.3V	

A	Y
0	
1	

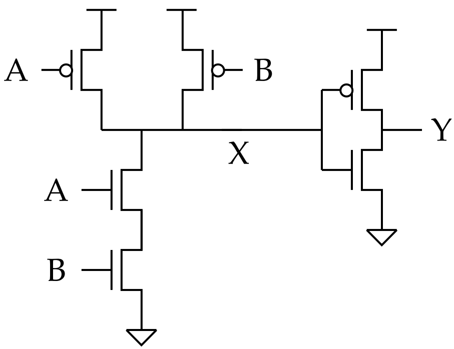


V_a	V_b	V_y
0V	0V	
0V	3.3V	
3.3V	0V	
3.3V	3.3V	

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 digital circuits from truth tables

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

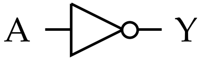
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

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

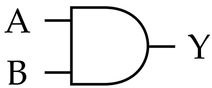
3. From Truth Tables to Combinational Logic Gates

NOT



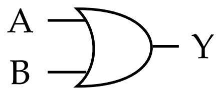
A	Y
0	
1	

AND



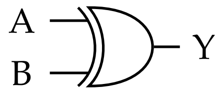
A	B	Y
0	0	
0	1	
1	0	
1	1	

OR



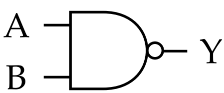
A	B	Y
0	0	
0	1	
1	0	
1	1	

XOR



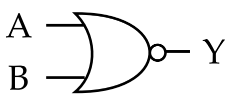
A	B	Y
0	0	
0	1	
1	0	
1	1	

NAND



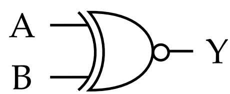
A	B	Y
0	0	
0	1	
1	0	
1	1	

NOR



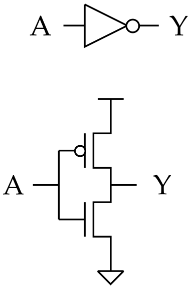
A	B	Y
0	0	
0	1	
1	0	
1	1	

XNOR

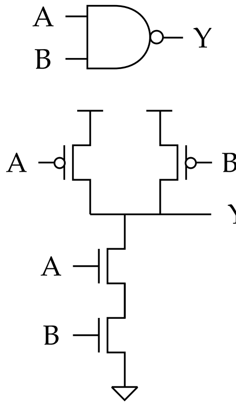


A	B	Y
0	0	
0	1	
1	0	
1	1	

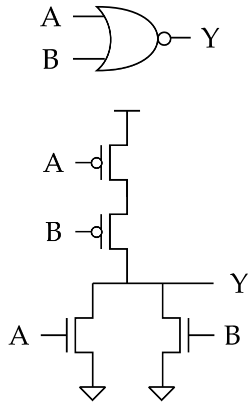
NOT



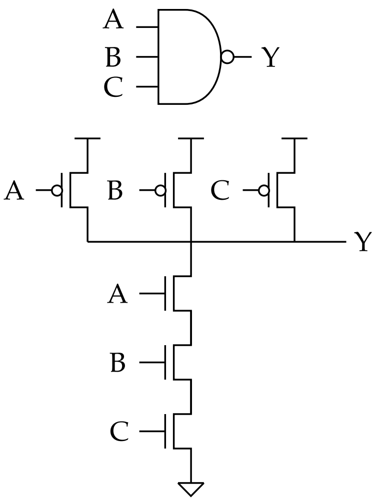
NAND2



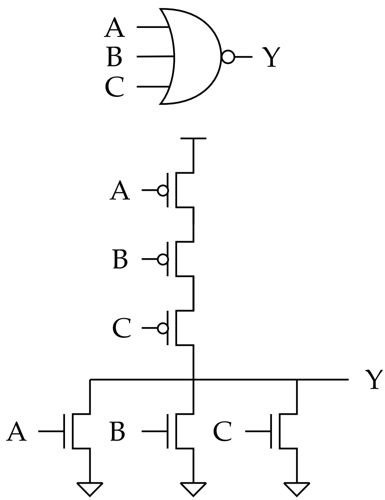
NOR2



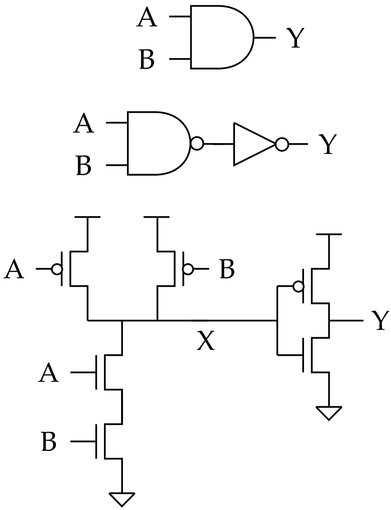
NAND3



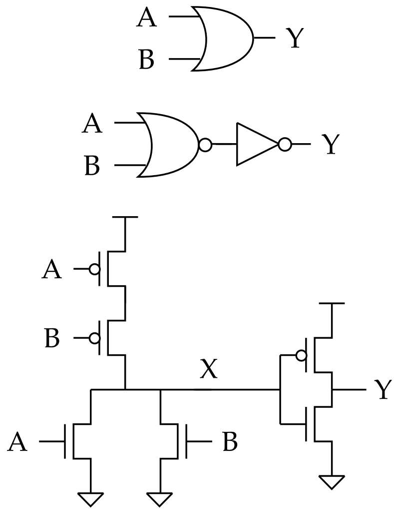
NOR3



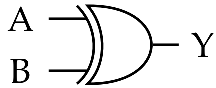
AND2



OR2



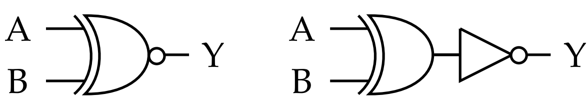
XOR



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

A	B	X	W	Z	Y
0	0				
0	1				
1	0				
1	1				

XNOR



3.1. Primitive Gate Sets

- Can implement a logic gate using other logic gates which can then be implemented again using more logic gates (e.g., XNOR)
- Can implement a logic gate using many different combinations of other logic gates
- ______________________ = lowest level "atomic" logic gates which are not implemented in terms of any other logic gates but are instead implemented using digital circuits

Incomplete Primitive Gate Sets

- Consider a primitive gate set with just an AND2 gate

- This is an *incomplete* primitive gate set since it is not possible to use this gate set to implement any other logic gate

Universal Primitive Gate Sets

- Consider a primitive gate set with just a NAND2 gate

- This is an *universal* primitive gate set since it is possible to use this gate set to implement any other logic gate
- A primitive gate set with just a NOR2 is also a universal
- It is inefficient (both in terms of area and timing) to implement all other logic gates just using NAND2 or NOR2 gates

Maximal Primitive Gate Sets

- Consider a primitive gate set with the following
  - NOT gate
  - n-input AND, n-input NAND gate
  - n-input OR, n-input NOR gate
  - n-input XOR, n-input XNOR gate
  - AOI21, AOI22 gate (and-or-inverting)
  - OAI12, OAI22 gate (or-and-inverting)
  - Multiplexors, half-adders, full-adders
  - Flip-flops, latches
- It is very efficient (both in terms of area and timing) to use a large primitive gate set
- It is difficult to understand how more complex gates work if we treat them as a block box implemented using digital circuits

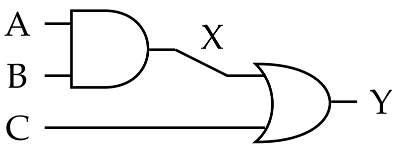
ECE 2300 Primitive Gate Set

- The course will use a primitive gate set with the following
  - NOT gate
  - n-input AND, n-input NAND gate
  - n-input OR, n-input NOR gate
  - n-input XOR, n-input XNOR gate

4. Combinational Gate-Level Networks

- A *gate-level network* is a collection of logic gates connected by wires
- Let's start by deriving truth tables from gate-level networks

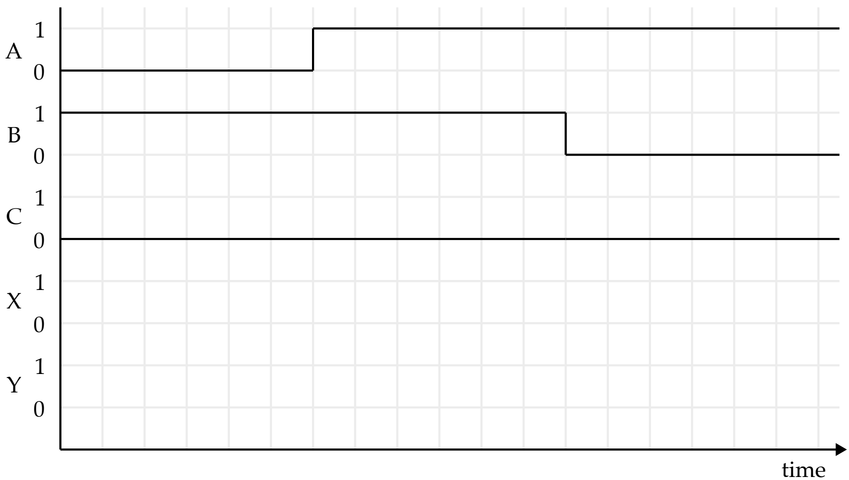
Gate-Level Network



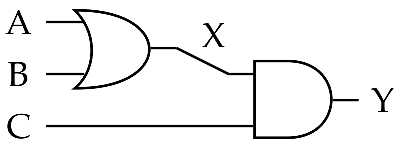
Truth Table

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		

Timing Diagram (assume zero delay model)



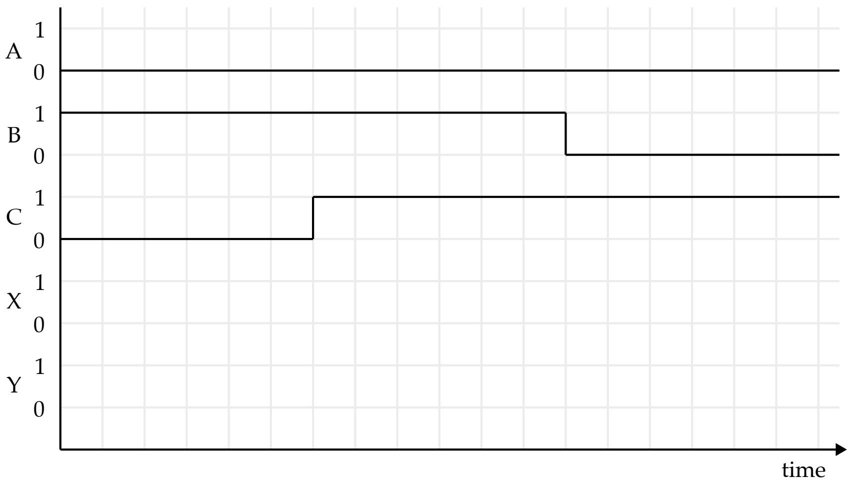
Gate-Level Network



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		

Timing Diagram (assume zero delay model)



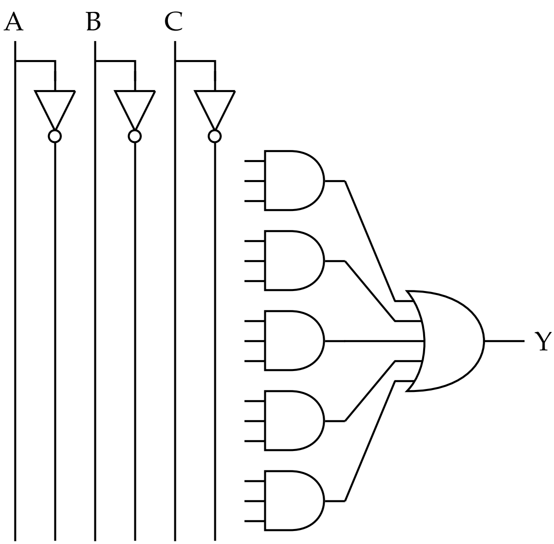
- Now let's derive gate-level networks from truth tables

Gate-Level Network

Truth Table

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

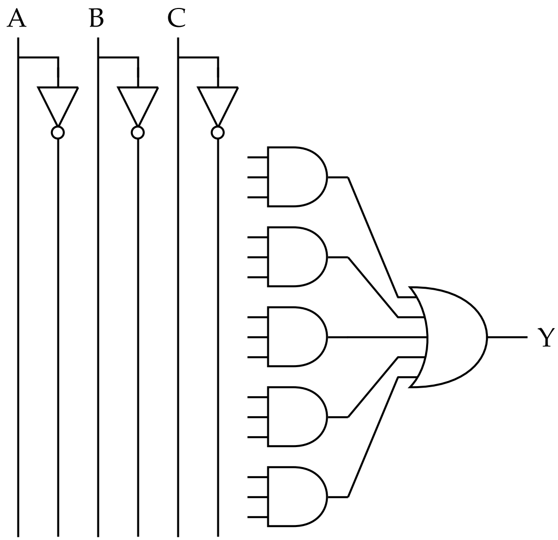
Gate-Level Network



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

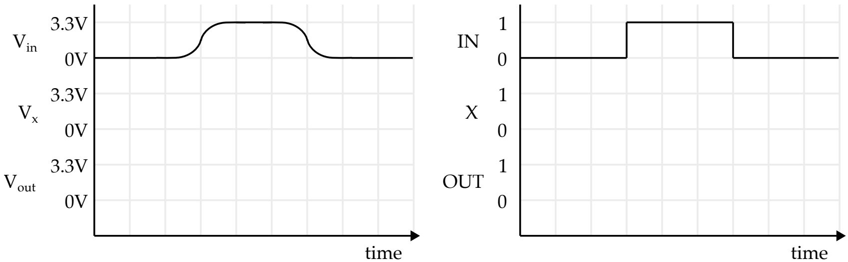


Truth Table

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

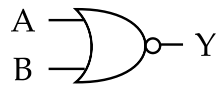
5. Combinational Logic Timing Analysis

- Use abstract time units (τ) instead of picoseconds (ps)

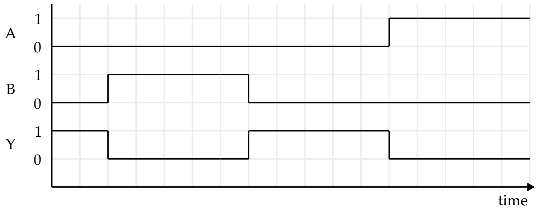


- Recall from previous topic that many delay models are possible
  - Zero delay models
  - Constant delay models
  - Input-dependent constant delay models
  - Transition- and input-dependent constant delay models
  - Load-, transition-, and input-dependent linear delay models
  - Slew-, load-, transition-, and input-dependent linear delay models
  - Non-linear delay models

5.1. Combinational Logic Gates Timing

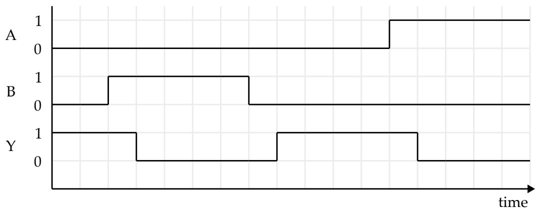


- Zero Delay Model



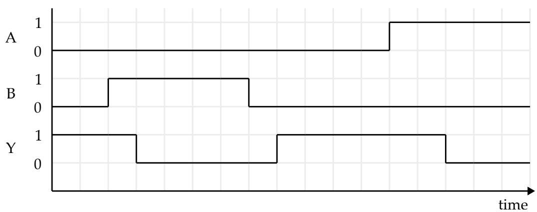
$t =$

- Constant Delay Model



$t =$

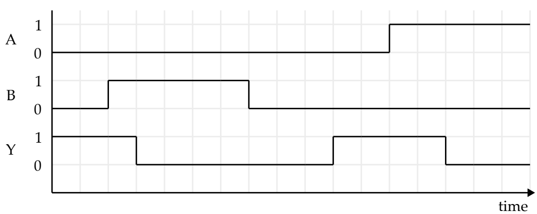
- Input-Dependent Constant Delay Model



$t_{A \to Y} =$

$t_{B \to Y} =$

- Transition- and Input-Dependent Constant Delay Model



$t_{A \to Y,hl} =$

$t_{A \to Y,lh} =$

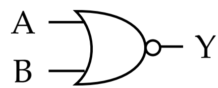
$t_{B \to Y,hl} =$

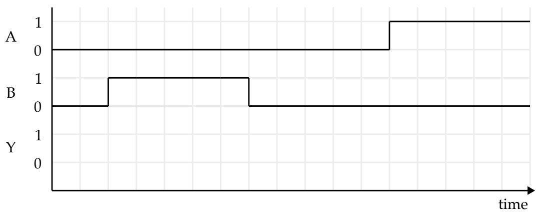
$t_{B \to Y,lh} =$

Propagation and Contamination Delay Modeling

- We want to improve the accuracy of the constant delay model but without the more complex of delay models
- ________________________ (t_{pd}) = maximum time from when input changes until the output reaches final value
- ________________________ (t_{cd}) = minimum time from when input changes until any output changes its value (might not be final value)

- Constant Delay Model
  (with propagation and contamination delays)

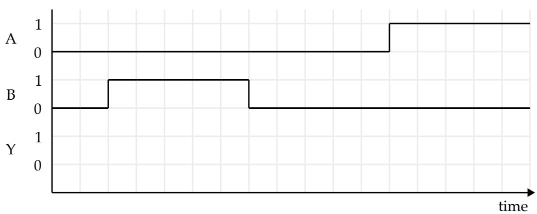




$t_{pd} =$

$t_{cd} =$

- Input-Dependent Constant Delay Model
  (with propagation and contamination delays)



$t_{pd,A \rightarrow Y} =$

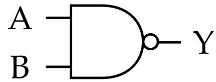
$t_{cd,A \rightarrow Y} =$

$t_{pd,B \rightarrow Y} =$

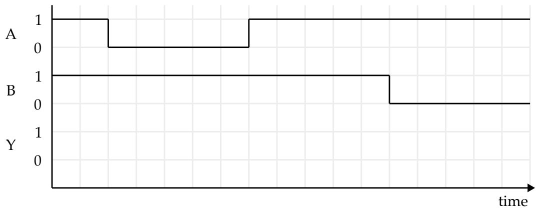
$t_{cd,B \rightarrow Y} =$

- Propagation and contamination delay modeling is conservative but simplifies our analysis

Complete the timing diagrams for a NAND gate.



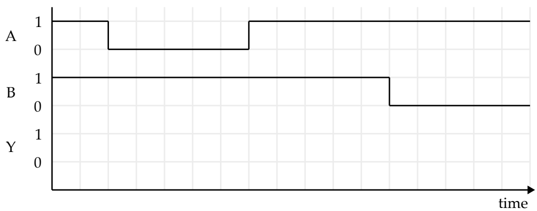
- Constant Delay Model
  (with propagation and contamination delays)



$t_{pd} = 2\tau$

$t_{cd} = 1\tau$

- Input-Dependent Constant Delay Model
  (with propagation and contamination delays)



$t_{pd,A\rightarrow Y} = 2\tau$

$t_{cd,A\rightarrow Y} = 1\tau$

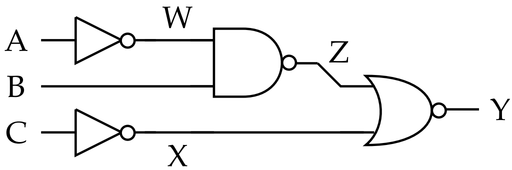
$t_{pd,B\rightarrow Y} = 1.5\tau$

$t_{cd,B\rightarrow Y} = 1\tau$

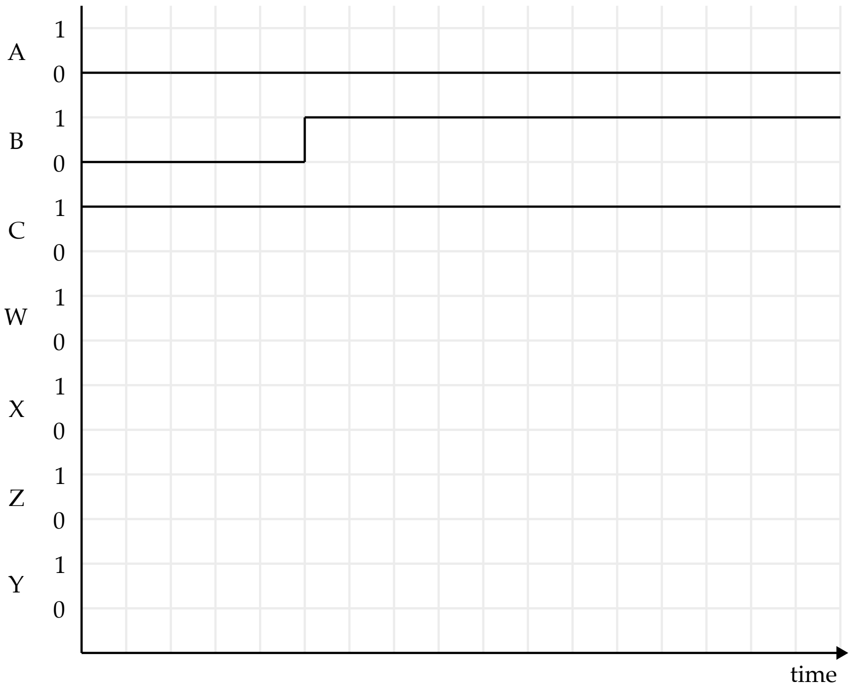
5.2. Combinational Gate-Level Network Timing

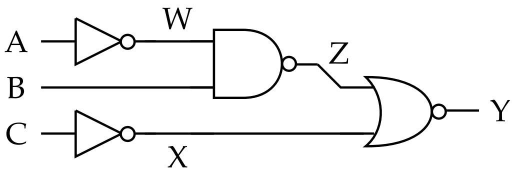
- Apply our delay model to all gates in a gate-level network
- Analyze all paths from every input to every output in network
- ________________________ = longest path through a gate-level network (only consider propagation delays!)
- ________________________ = shortest path through a gate-level network (only consider contamination delays!)

5.3. Combinational Gate-Level Network Timing



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



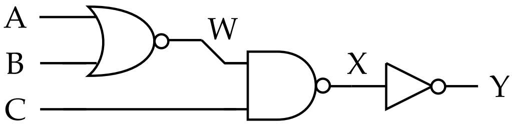


Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ

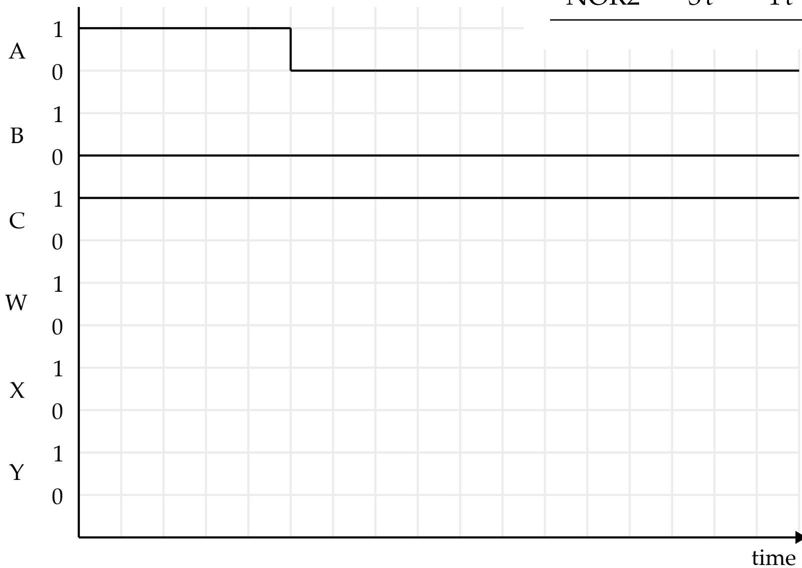
Path	**Propagation Delay**	**Contamination Delay**
A → NOT → NAND2 → NOR2 → Y		
B → NAND2 → NOR2 → Y		
C → NOT → NOR2 → Y		

- Which path is the critical path?
- Which path is the short path?

Complete the timing diagram and table for given gate-level network.



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



Path	**Propagation Delay**	**Contamination Delay**

6. Verilog Modeling of Combinational Logic

- Verilog is not a programming language (PL), it is a hardware description language (HDL)
- ______________________________ = portion of the language that models real hardware
- ______________________________ = portion of the language that does not model real hardware but is instead used to test hardware
- We will *simulate* designs implemented in Verilog to verify their functionality
- We will *synthesize* designs implemented in Verilog to map them to real hardware in the lab using a field programmable gate array

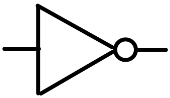
Verilog Datatypes

- All signals in Verilog can have one of four values:
  - Logic 1
  - Logic 2
  - X (undefined value or don't care about specific value)
  - Z (floating)

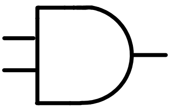
It is critical to always keep in mind the
real hardware you are trying to model using Verilog!

6.1. Modeling Logic Gates in Verilog

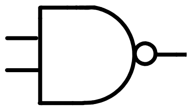
- Declare wires using `wire`
- Instantiate logic gate primitives (e.g., `not`, `and`)
- List wires to connect them to inputs and outputs, output is first
- Logic gates with more than two inputs are allowed



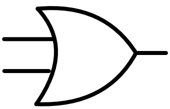
```
wire a;
wire y;
not( y, a );
```



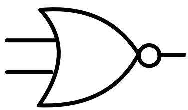
```
wire a;
wire b;
wire y;
and( y, a, b );
```



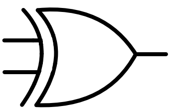
```
wire a;
wire b;
wire y;
nand( y, a, b );
```



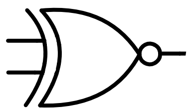
```
wire a;
wire b;
wire y;
or( y, a, b );
```



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



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

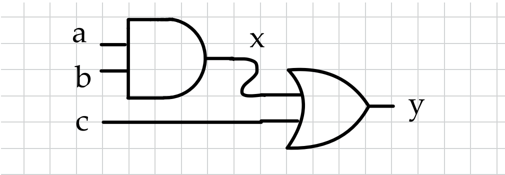


```
wire a;
wire b;
wire y;
xnor( y, a, b );
```

- What if one of the inputs is X?
- Consider an AND gate

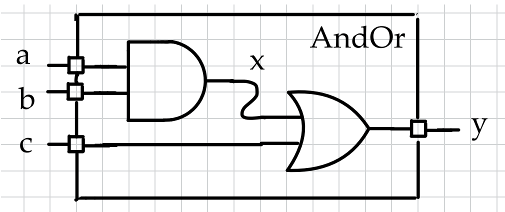
A	B	Y
0	X	
1	X	
X	0	
X	1	
X	X	

6.2. Modeling Gates-Level Netlists in Verilog




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

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

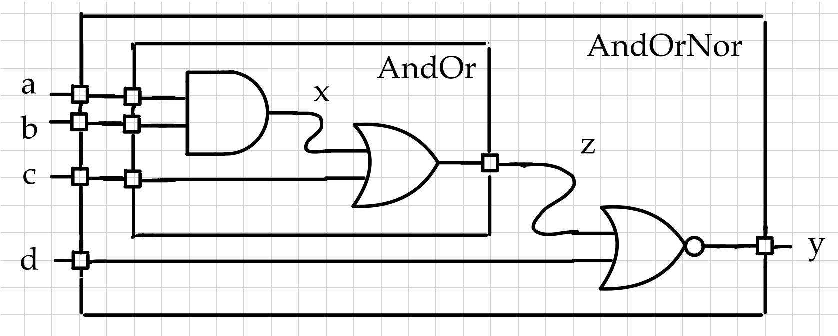
Defining Hardware Modules in Verilog




```
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
```

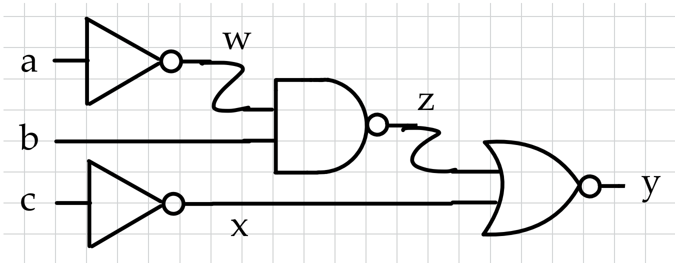
`https://www.edaplayground.com/x/KjwN`

Instantiating Hardware Modules in Verilog

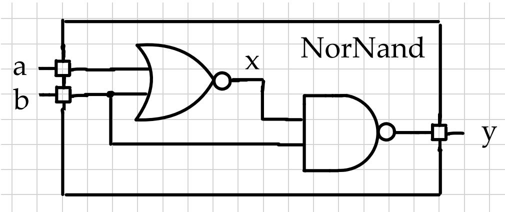



```
module AndOrNor
(
  input  wire a,
  input  wire b,
  input  wire c,
  input  wire d,
  output wire y
);

  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.



6.3. Modeling Delays in Verilog



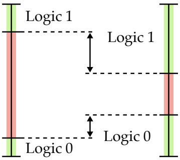
Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
NAND2	2τ	1τ
NOR2	3τ	1τ

```
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

7. Summary of Abstractions

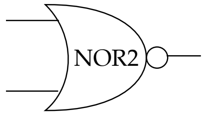
Combinational Logic



Bits

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

Truth Tables

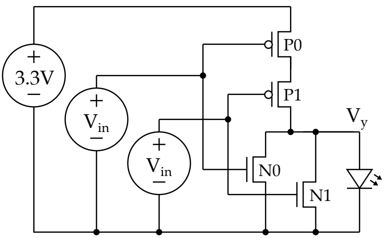


Logic Gates

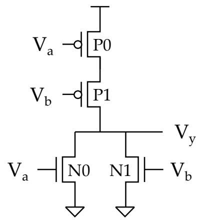
$$t_{pd} = 3\tau$$
$$t_{cd} = 1\tau$$

Delay Models

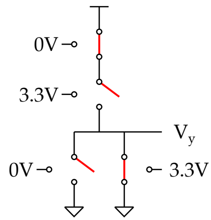
Digital Circuits



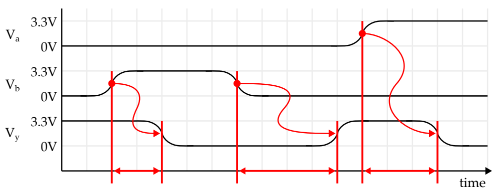
Circuit Diagrams



Transistor-Level Schematics



Switch-Level Model



Delay Models