CT Verilog Series

# Verilog Overview Part 1
# Structural Modeling

黃稚存
Chih-Tsun Huang
cthuang@cs.nthu.edu.tw

國立清華大學
資訊工程學系

03

# 聲明

- 本課程之內容 (包括但不限於教材、影片、圖片、檔案資料等)，僅供修課學生個人合理使用，非經授課教師同意，不得以任何形式轉載、重製、散布、公開播送、出版或發行本影片內容(例如將課程內容放置公開平台上，如 Facebook, Instagram, YouTube, Twitter, Google Drive, Dropbox 等等)。如有侵權行為，需自負法律責任。

# Outline

- Background
- Digital Switches and Logic Gates
- Tri-state Gates
- Structural Modeling
- Delay Model

# Background

# Major Hardware Description Languages (HDLs)

Intepreted Language Script

Object Oriented

- Verilog
  - Started by Gateway in 1983
  - Became open to public by Cadence in 1990
  - IEEE standard 1364 in 1995/2001/2005
  - Slightly better at gate/transistor level
  - Language style close to C/C++
  - Pre-defined data type, easy to use
- VHDL
  - Started by VHSIC project in 1980s
  - IEEE standard 1076 in 1987, IEEE 1164 in 1993
  - Slightly better at system level
  - Language style close to Ada/Pascal
  - User-defined data type, more flexible

Equally effective, personal preference

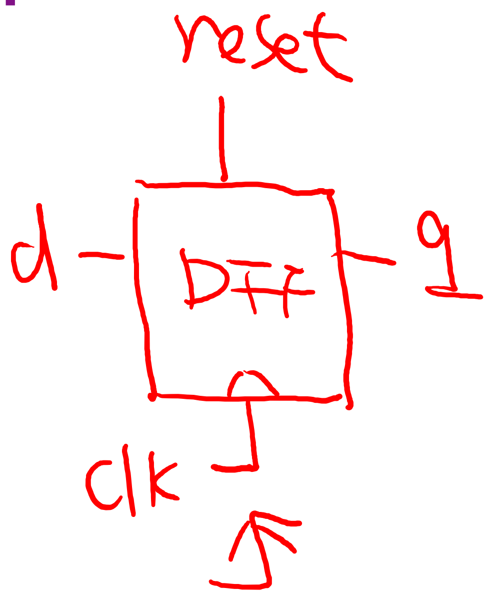
# Design Entry

- Schematic entry (gate-level entry)
  - Or called structural modeling
  - Starting with structural details of a design
- HDL (RTL) entry
  - Or *behavioral entry*
  - Describing a design by structure level, register transfer level (RTL), behavior level or a mixture
    - Dataflow modeling
    - Behavioral modeling
- We choose Verilog Hardware Description Language (HDL)

# Language Conventions for Verilog

- Case sensitive
- Instance of a `module` must be named
- Keywords are lower-case
- Identifiers:
  - May use upper- and lower-case alphabetical characters (`a-zA-Z`), decimal digits (`0-9`), underscore (`_`), and dollar sign (`$`)
  - May contain up to 1024 characters, and the first character must not be a digit or $
- Single-line comments start with "`//`", and blocked (multi-line) comments begin with "`/*`" and end with "`*/`"
- Compiler directives begin with grave accent `` ` `` (e.g., `` `define state0 3'b001 ``)
- All names beginning with $ denote built-in system tasks or functions (e.g., `$display`)

# Four Logic Values in Verilog Simulation




- **0**: Logic zero, false, ground (GND)
- **1**: Logic one, true, supply voltage (Vdd, Vss)
- **X**: Unknown logic value ≠ don't care
  - When we are not sure if it is 0 or 1
- **Z**: High impedance (high-z), floating state

- X and Z are not practical logic values
  - Assignment of Z to synthesize tri-state buffers by CAD tool
  - Assignment of X to propagate errors
    - Pessimistic in Verilog simulation

# Digital Switches and Logic Gates

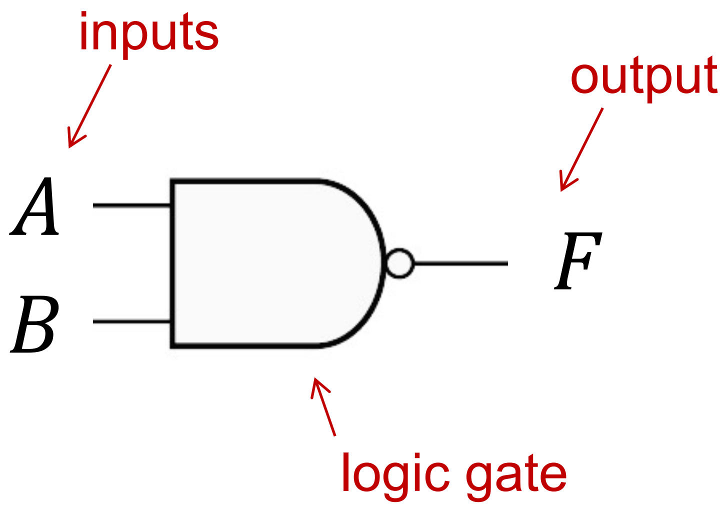
# Logic Gates

- NAND

complement

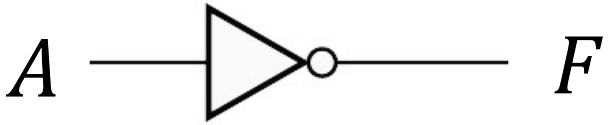
$$F = \overline{A \cdot B}$$

and



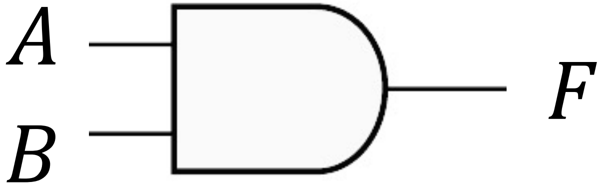

- Inverter (NOT, INV)

$$F = \bar{A}$$



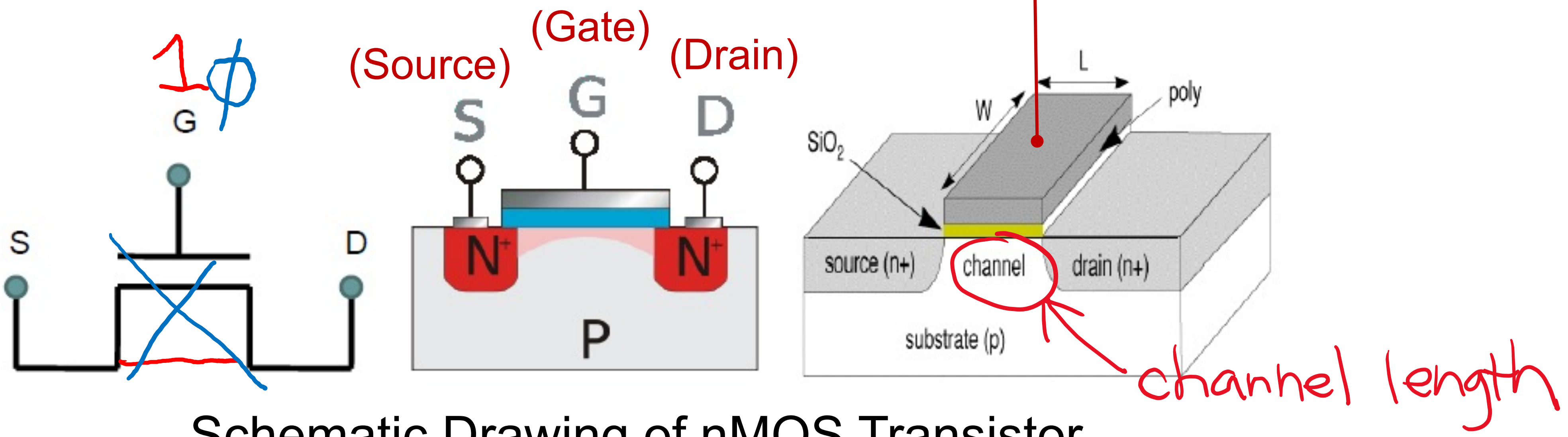

- AND

$$F = A \cdot B$$



# Metal-Oxide Semiconductor (MOS) Transistor

- Transistor: Transfer + Resistor
- A three-terminal device: gate, source, drain
  - (Analog) amplifier
  - (Binary) switch

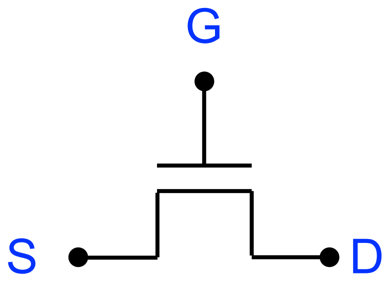


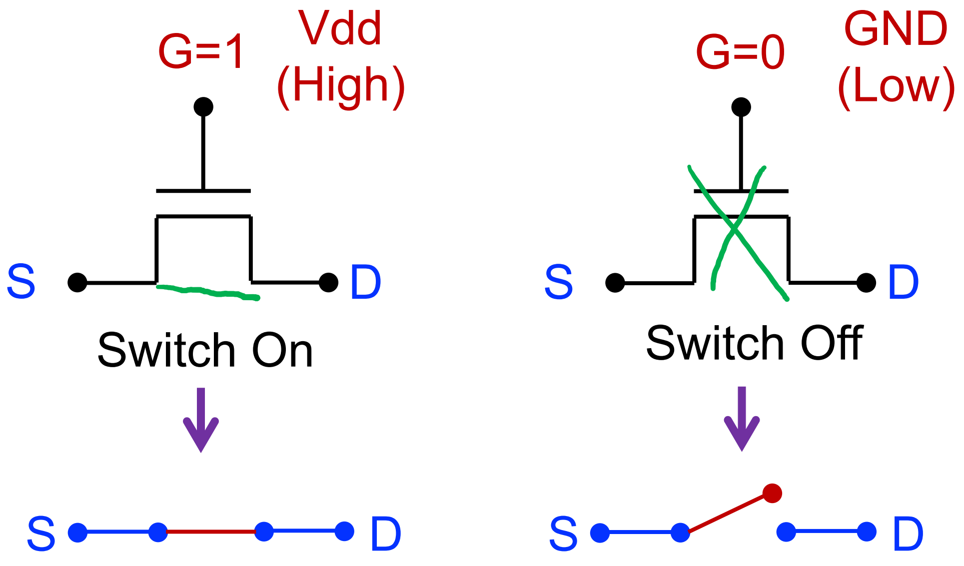

Schematic Drawing of nMOS Transistor

[Source: Prof. Jing-Jia Liou]

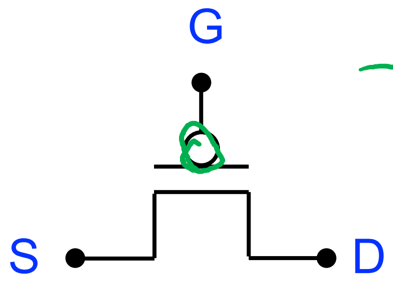
# MOS Transistors as Switches

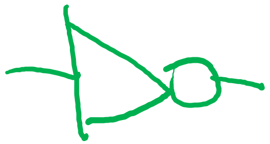
- nMOS transistor

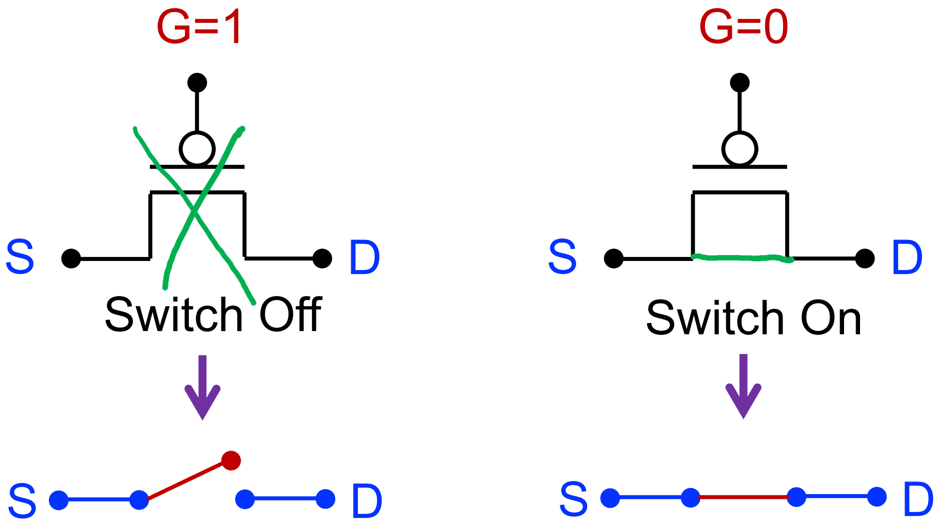



- pMOS transistor



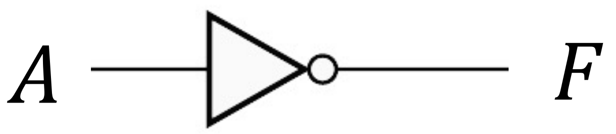





# Logic Gate: Inverter (INV, NOT)

- CMOS inverter
  - Complementary MOS: nMOS + pMOS switches

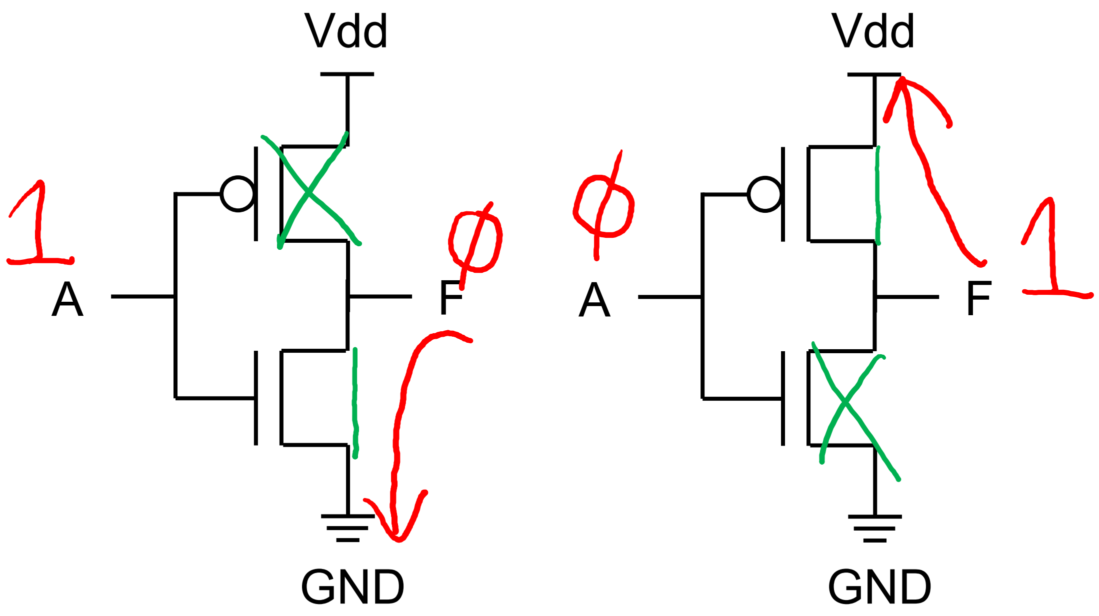
Gate-Level Schematic




Truth Table

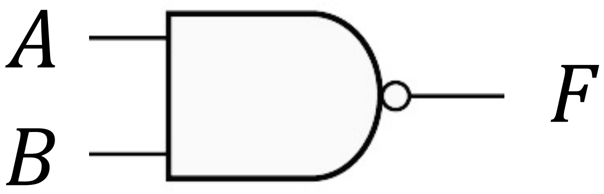
| $A$ | $F=\overline{A}$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

Transistor-Level Schematic



# Logic Gate: NAND

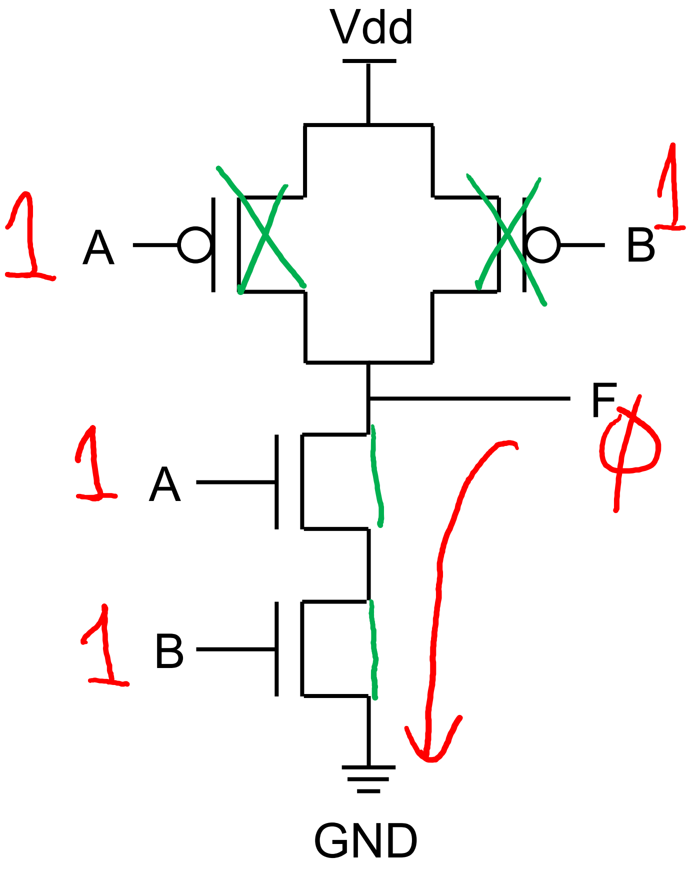
## Gate-Level Schematic



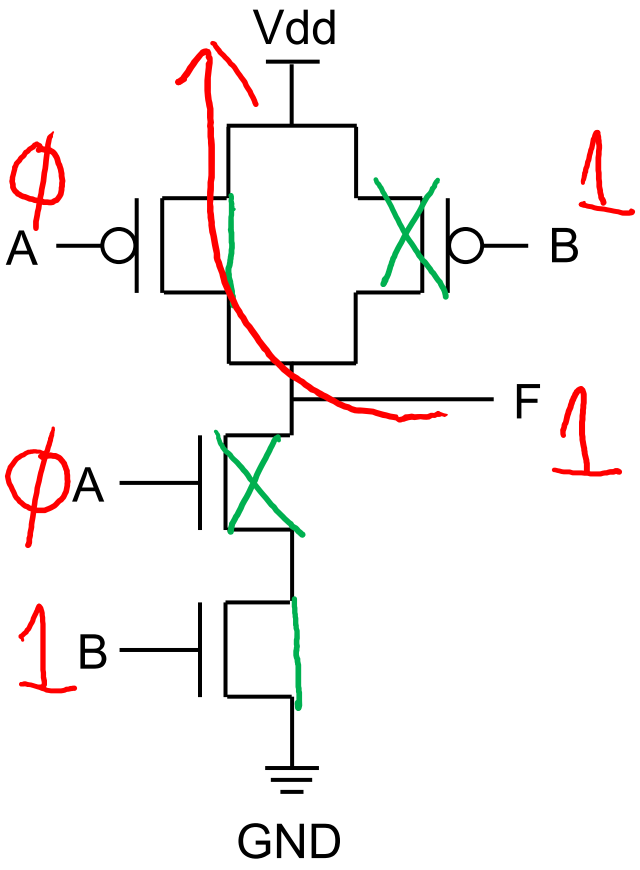

## Truth Table

| $A$ | $B$ | $F = \overline{A \cdot B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## Transistor-Level Schematic



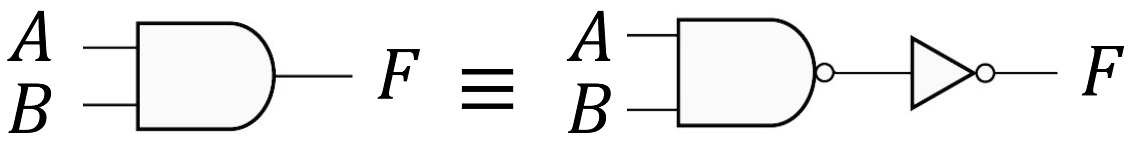

A=1, B=1 ➔ F=0




A=0, B=1 ➔ F=1

# Logic Gate: AND

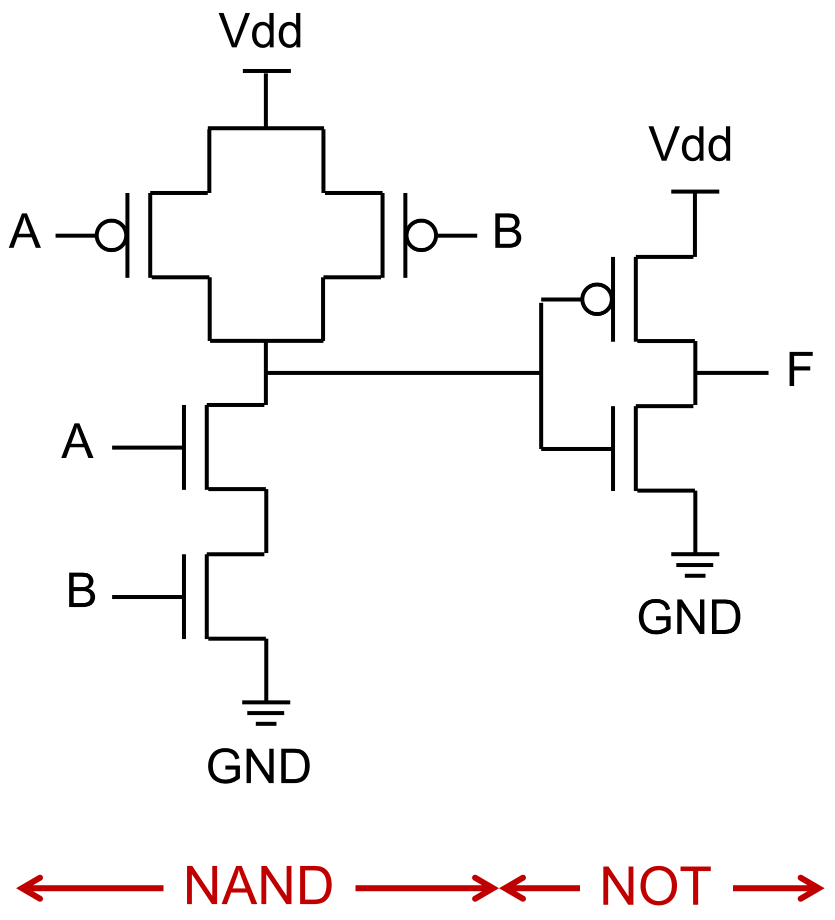
## Gate-Level Schematic




## Truth Table

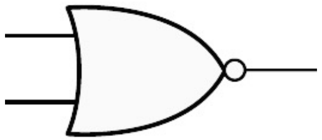
| $A$ $B$ | $F = A \cdot B$ |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

## Transistor-Level Schematic



# How about NOR Gate?

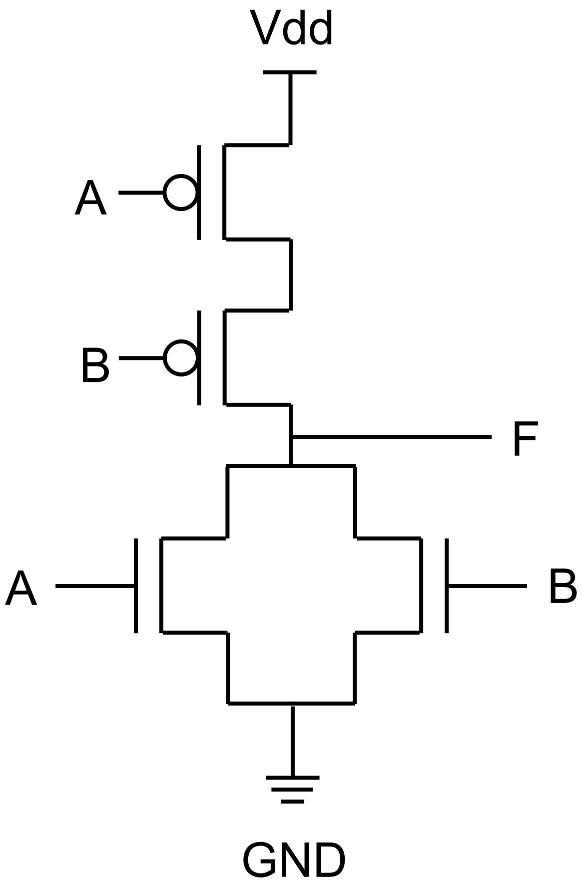
## Gate-Level Schematic

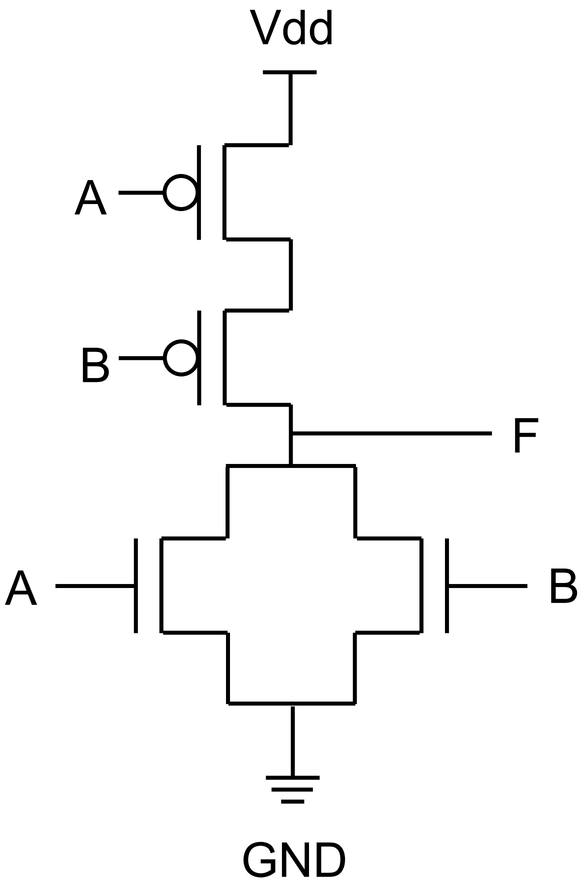


## Truth Table

| $A$ | $B$ | $F = \overline{A + B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

## Transistor-Level Schematic

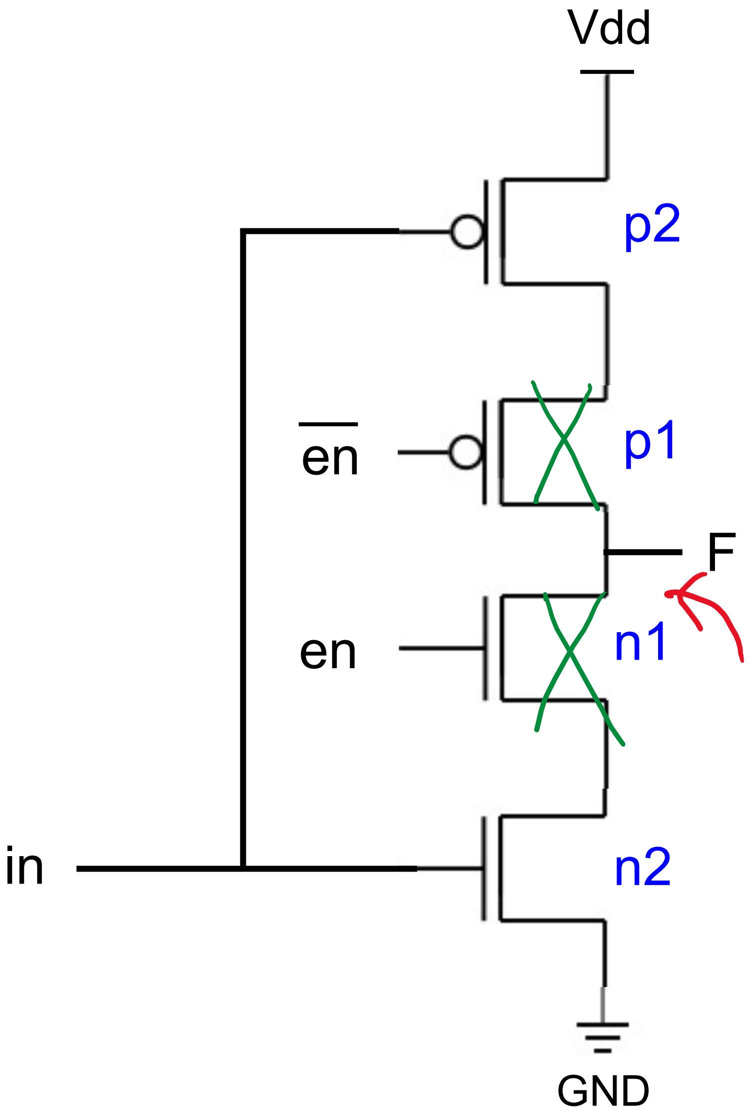


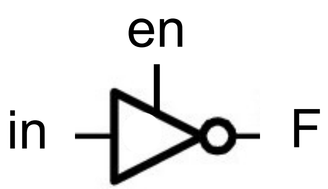



# Tri-state Gates

- Or three-state gate

# Tri-State Inverter




| *en* | *F* |
|---|---|
| 0 | *Z* |
| 1 | *in* |

① en = 1 ➔ n1 on, p1 on

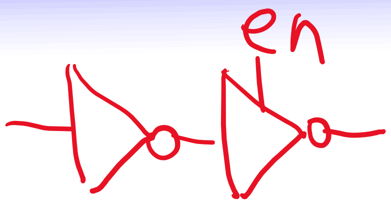
F = $\overline{\text{in}}$

② en = 0 ➔ n1 off, p1 off

F ➔ high Z (hi-Z)

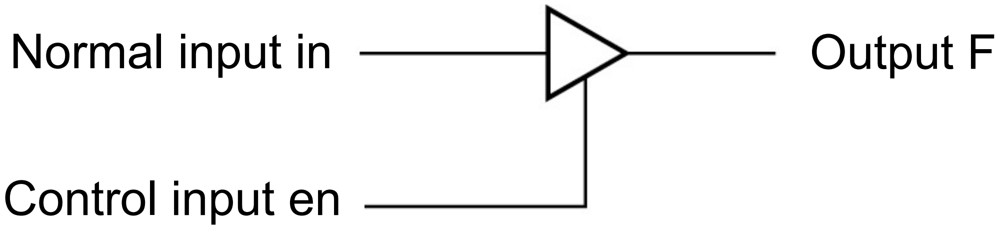
or high impedance

# Tri-state Buffer

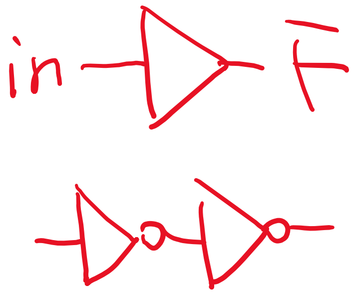



- Output states: 0, 1, and Z (high-impedance, high-Z, or open circuits)

**Tri-state Buffer**

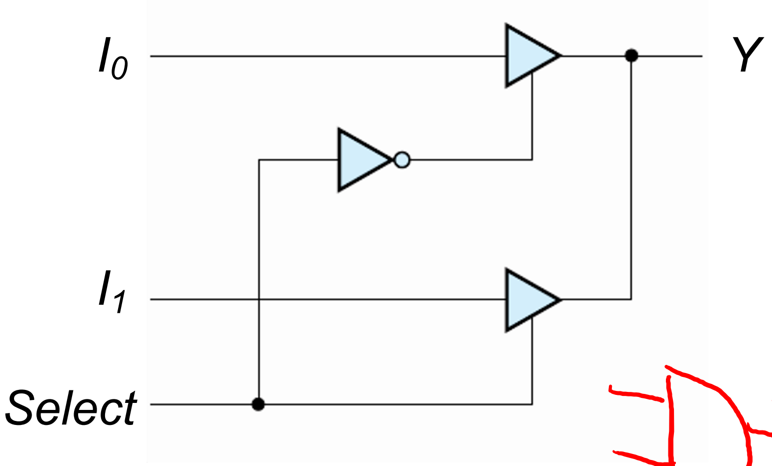


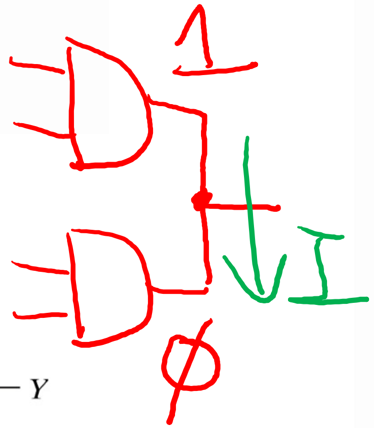

| $en$ | $F$ |
|---|---|
| 0 | $Z$ |
| 1 | $\overline{in}$ |

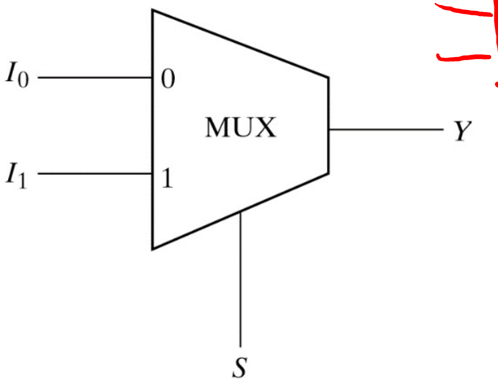


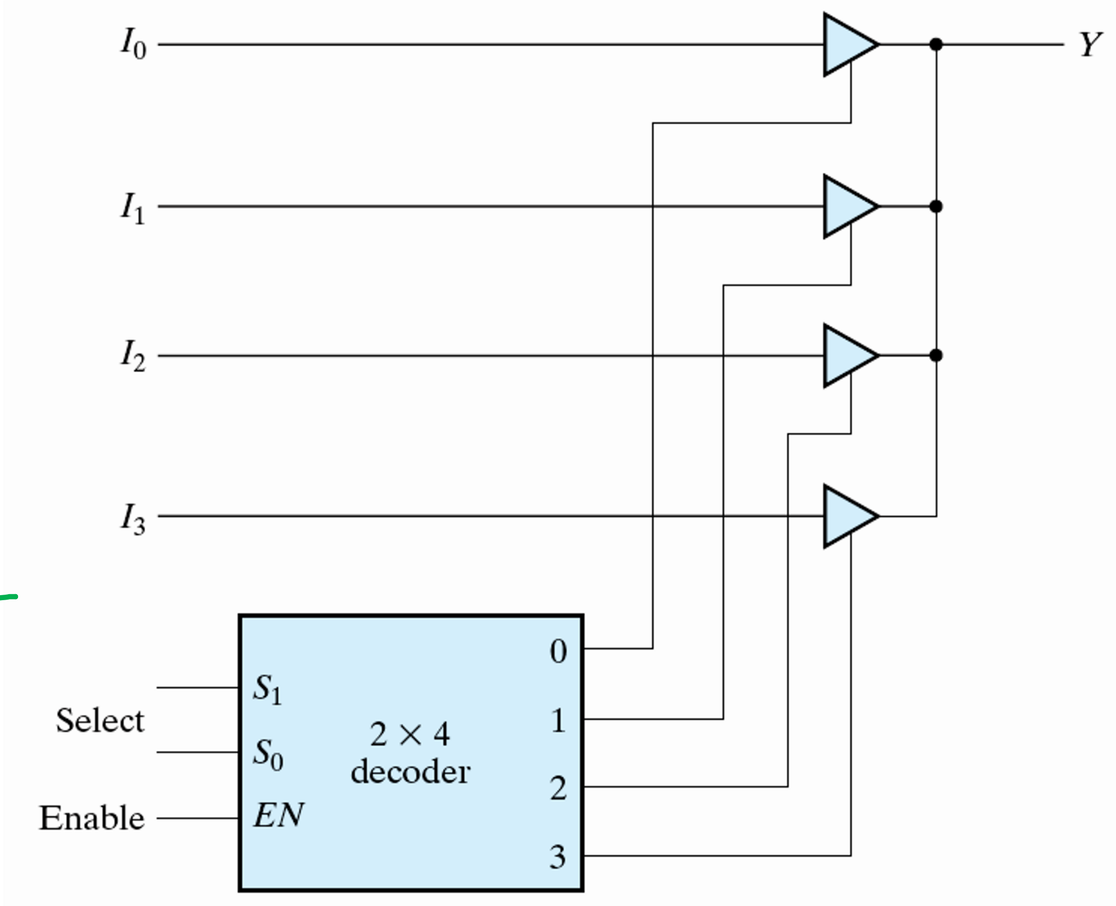

- Tri-state gates can be used to constructed multiplexer
- Tri-state gates can be used to build up a bus
  - A communication channel among different modules in a digital system
- Tri-state gates are not preferred inside local designs

# Multiplexer with Tri-state Gates



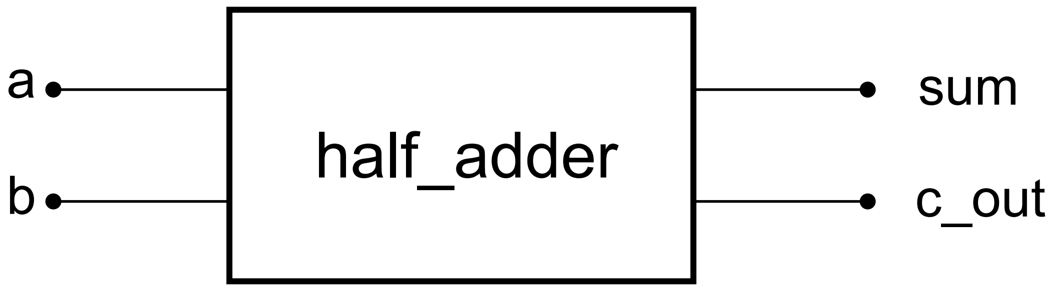



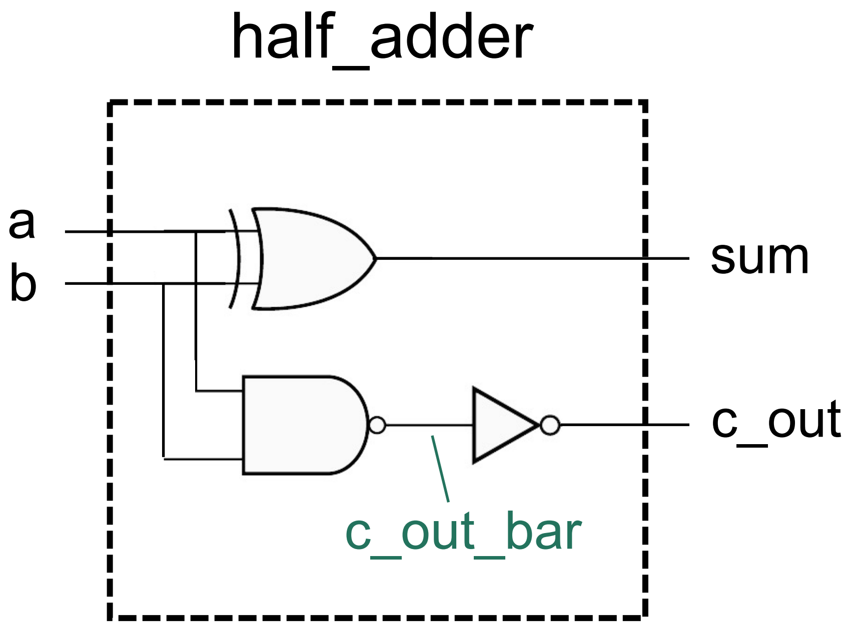
# Structural Modeling

- Or gate-level modeling

# Block Diagram and Schematic




Block Diagram




Logic Diagram

(Gate-level Design)

# Verilog Structural Description

Module Name

Module Ports

```
module half_adder (sum, c_out, a, b);
  input  a,b;
  output sum, c_out;
  wire   c_out_bar;

  xor (sum, a, b);
  nand (c_out_bar, a, b);
  not (c_out, c_out_bar);
endmodule
```

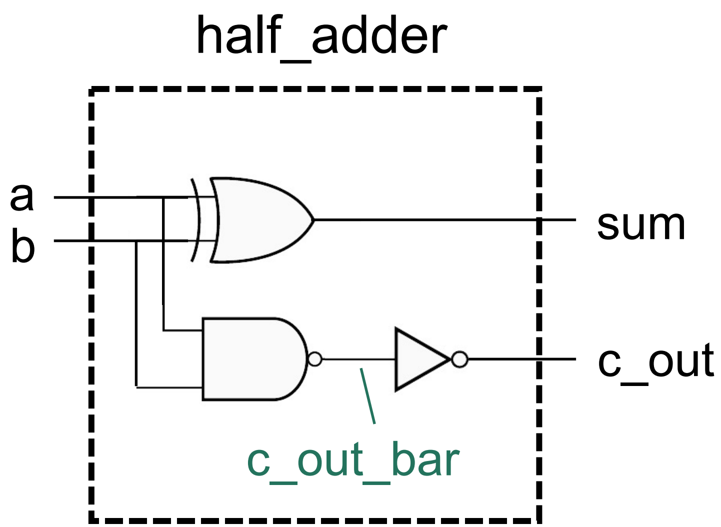
Declaration of Port Directions

Declaration of Internal Signal

Output

Inputs

Instantiation of Primitive Gates




*Note: All bold-faced items are Verilog keywords.*

# Verilog Predefined Primitives (1/2)

- Single-input gates
  - `buf, not`
- Multi-input gates
  - `and, nand, or, nor, xor, xnor`
- Tri-state gates
  - `bufif0, bufif1, notif0, notif1`

# Verilog Predefined Primitives (2/2)

- Never used stand-alone in a design, must be instantiated within a module
- Identifier (instance name) is optional
- Default delay = 0
  - Zero-delay model
- Port elements of an instantiated primitive must be output followed by inputs
  - The number of inputs can vary
- Verilog also supports User-Defined Primitive (UDP)

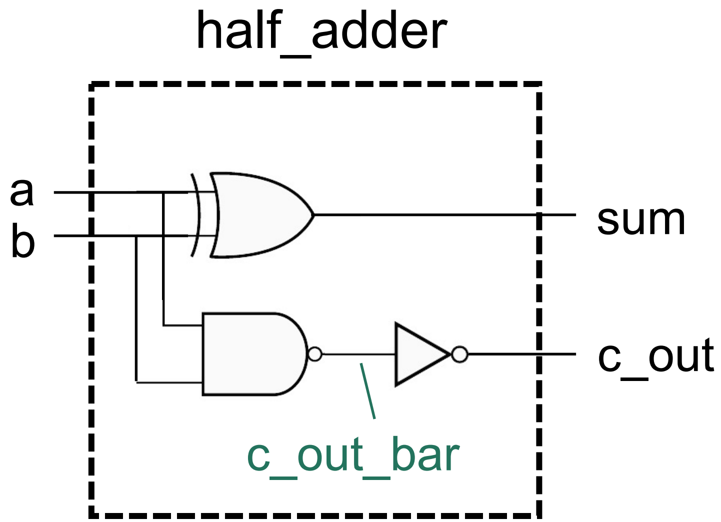
# Optional Primitive Identifiers

```
module half_adder (sum, c_out, a, b);
  input    a, b;
  output   sum, c_out;
  wire     c_out_bar;
```

Reference | Instance Name

```
  xor g1 (sum, a, b);
  nand (c_out_bar, a, b);
  not g2 (c_out, c_out_bar);
endmodule
```

c_out_bar




*g1, g2 are primitive identifiers.*
*An undeclared variable is assumed to be a single-bit **wire**.*

# Truth Tables of Primitive Gates

- For Verilog **simulation**

| and | 0 | 1 | x | z |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | x | x |
| 0/1 x | 0 | x 0/1 | x | x |
| z | 0 | x | x | x |

| nor | 0 | 1 | x | z |
|---|---|---|---|---|
| 0 | 1 | 0 | x | x |
| 1 | 0 | 0 | 0 | 0 |
| x | x | 0 | x | x |
| z | x | 0 | x | x |

| xor | 0 | 1 | x | z |
|---|---|---|---|---|
| 0 | 0 | 1 | x | x |
| 1 | 1 | 0 | x | x |
| x | x | x | x | x |
| z | x | x | x | x |

| in | buf |
|---|---|
| 0 | 0 |
| 1 | 1 |
| x | x |
| z | x |

| nand | 0 | 1 | x | z |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | x | x |
| x | 1 | x | x | x |
| z | 1 | x | x | x |

| or | 0 | 1 | x | z |
|---|---|---|---|---|
| 0 | 0 | 1 | x | x |
| 1 | 1 | 1 | 1 | 1 |
| x | x | 1 | x | x |
| z | x | 1 | x | x |

| xnor | 0 | 1 | x | z |
|---|---|---|---|---|
| 0 | 1 | 0 | x | x |
| 1 | 0 | 1 | x | x |
| x | x | x | x | x |
| z | x | x | x | x |

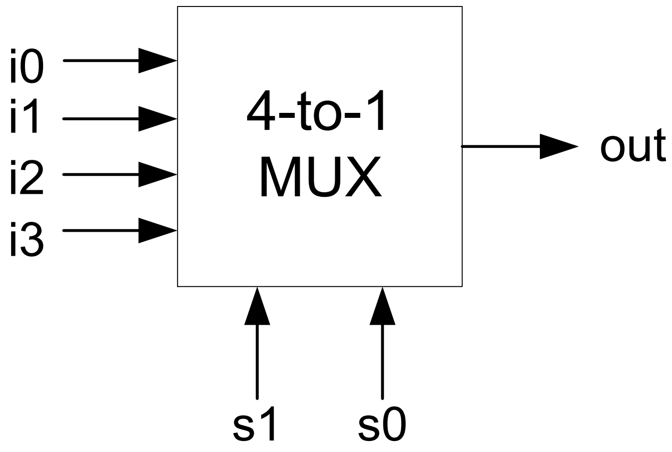
| in | not |
|---|---|
| 0 | 1 |
| 1 | 0 |
| x | x |
| z | x |

# Four-Value Simulation for Binary System

- `0`: Logic zero, false
- `1`: Logic one, true
- `z`: High impedance, floating state
- `x`: Unknown logic value for pessimistic simulation
  - `x & 0 = ____________`    `z | 0 = ____________`
  - `x & 1 = ____________`    `z | 1 = ____________`
  - `x ^ 1 = ____________`    `x & z = ____________`
  - `z ^ 1 = ____________`    `z & z = ____________`
  - `~x = ____________`    `~z = ____________`

# Gate-Level Multiplexer (1/3)

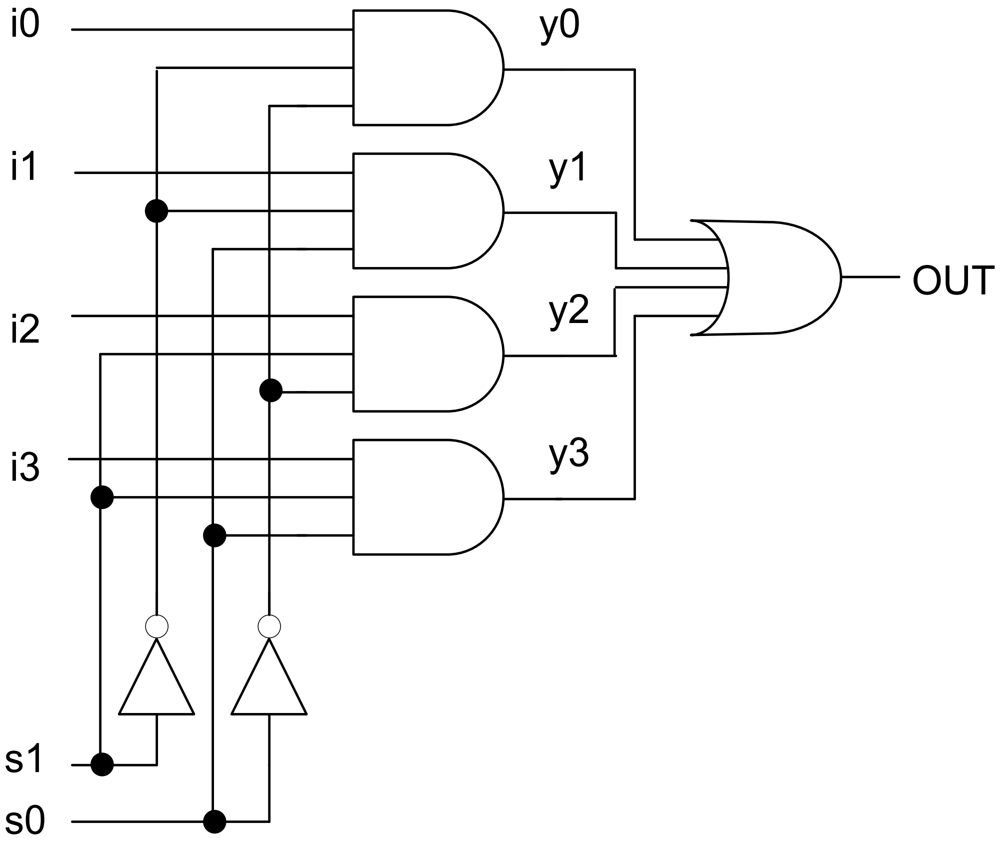
## Block Diagram




## Truth Table

| s1 | s0 | out |
|---|---|---|
| 0 | 0 | i0 |
| 0 | 1 | i1 |
| 1 | 0 | i2 |
| 1 | 1 | i3 |

## Logic Diagram



# Gate-Level Multiplexer (2/3)

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
  output out;
  input i0, i1, i2, i3;
  input s1, s0;
  wire s1n, s0n;
  wire y0, y1, y2, y3;
  not (s1n, s1);
  not (s0n, s0);
  and (y0, i0, s1n, s0n);
  and (y1, i1, s1n, s0);
  and (y2, i2, s1, s0n);
  and (y3, i3, s1, s0);
  or (out, y0, y1, y2, y3);
endmodule
```

- Instance names are optional for Verilog primitives
  - Mandatory for user-defined modules

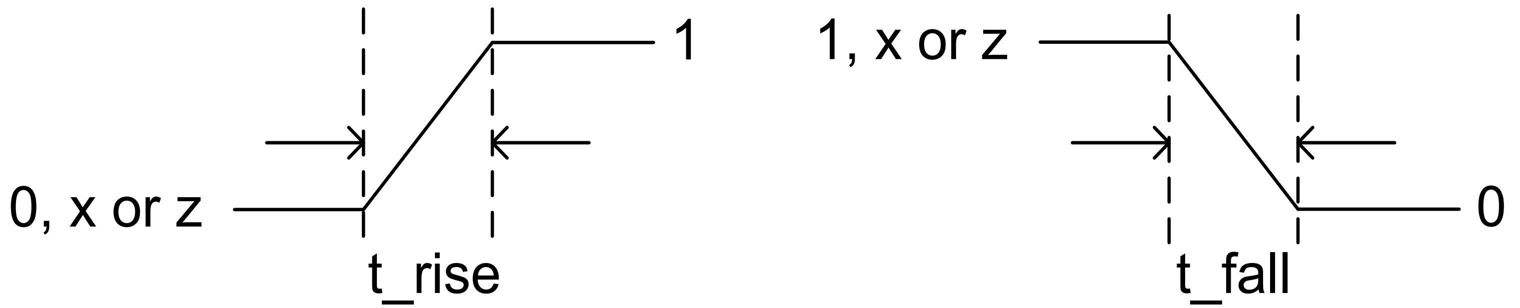
# Gate-Level Multiplexer (3/3)

```
`timescale 1ns/100ps
module stimulus;
  reg IN0, IN1, IN2, IN3;
  reg S1, S0;
  wire OUT;
  mux4_to_1 mymux(OUT, IN0, IN1, IN2, IN3, S1, S0);
  initial begin
    IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
    #1 S1 = 0; S0 = 0;
    #1 S1 = 0; S0 = 1;
    #1 S1 = 1; S0 = 0;
    #1 S1 = 1; S0 = 1;
  end
  initial
    $monitor($time,
      " {IN3, IN2, IN1, IN0} = %b, {S1, S0} = %b, OUT = %b",
      {IN3, IN2, IN1, IN0}, {S1, S0}, OUT);
endmodule
```

# Delay Model

# Basic Delay Model

- Rise delay: rising edge (or positive edge): 0, x, or z → 1
- Fall delay: falling edge (or negative edge): 1, x, or z → 0




- Turn-off delay
  - The delay associated with a gate output transition to the high z from another value
    0 → z
    1 → z
- When the value changes to x
  - min{rise delay, fall delay, turn-off delay}

# Gate Delay (1/2)

```
bufif0 #(rise_del, fall_del, turnoff_del) b1(out, in, ctrl);
```

```
and #(rise_del, fall_del) a2(out, i1, i2);
```

- If 2 delays specified, e.g., **`#(delay1, delay2)`**
  - **`#(delay1, delay2, min(delay1, delay2))`**

```
and #(rise_del) a2(out, i1, i2);
```

- If only 1 delay specified, e.g., **`#(delay1)`**
  - **`#(delay1, delay1, delay1)`**

- If no delay specified
  - The default value is zero (zero-delay model)

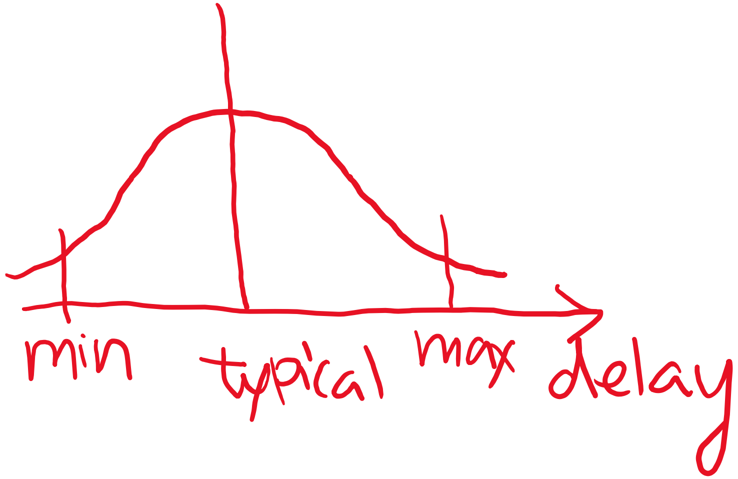
# Gate Delay (2/2)

- Each delay specification can be further modeled as
  - `min:typical:max` (minimum:typical:maximum)

```
and #(4:5:6) a1(out, i1, i2);
and #(3:4:5, 5:6:7) a2(out, i1, i2);
and #(2:3:4, 3:4:5, 4:5:6) a3(out, i1, i2);
```

- For NC Verilog simulator

```
$ ncverilog test.v +maxdelays
$ ncverilog test.v +mindelays
$ ncverilog test.v +typdelays
```



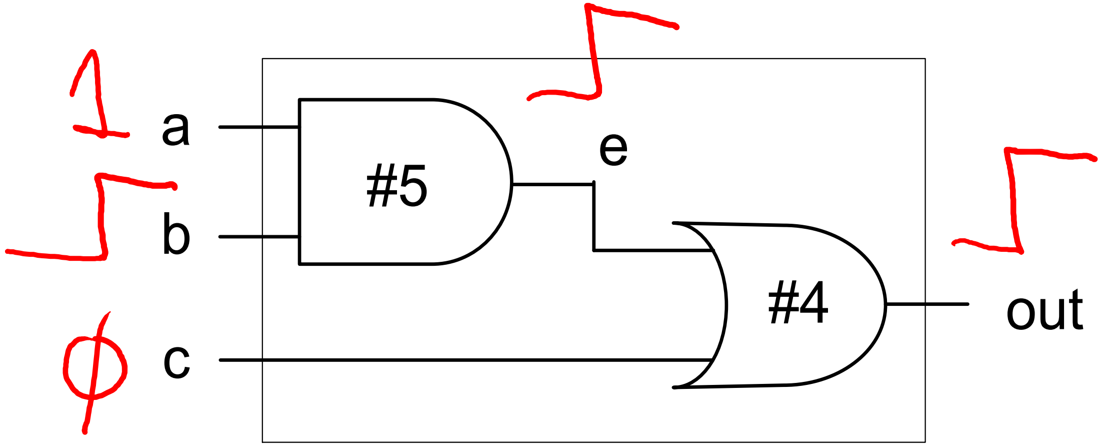
# Critical Delay and Critical Path

- Critical delay (timing) determines the operating clock speed (frequency)
  - Improve speed -> shorten critical delay

```
module D (out, a, b, c);
  output out;
  input a, b, c;

  // Internal nets
  wire e;

  and #(5) a1(e, a, b);
  or  #(4) o1(out, e, c);
endmodule
```



# Test Stimulus Example

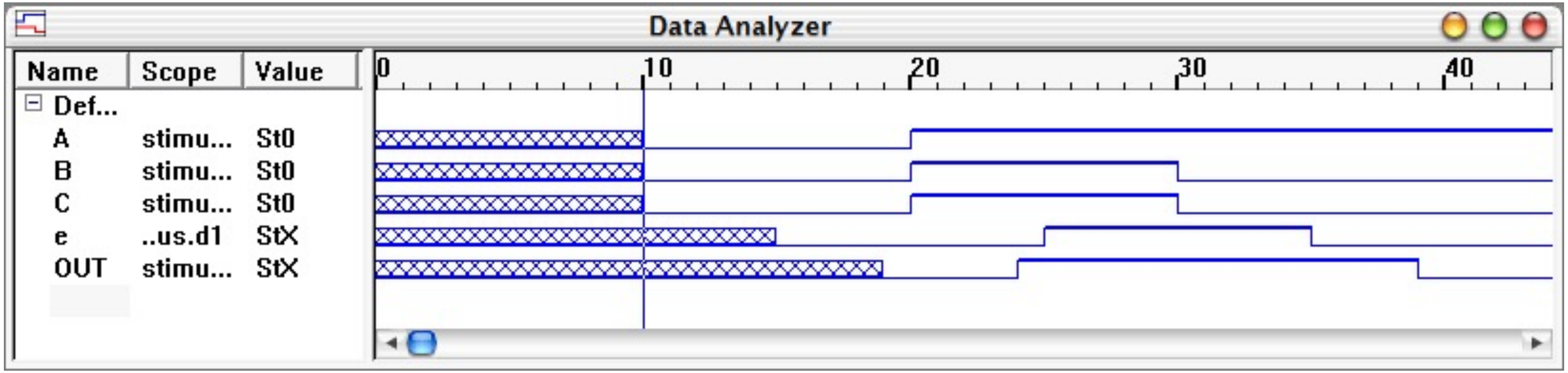
```
`timescale 1ns/100ps
module stimulus;
  reg A, B, C;
  wire OUT;
  D d1( OUT, A, B, C);

  initial
    $monitor($time,
      " A= %b, B=%b, C= %b, OUT= %b\n", A, B, C, OUT);

initial begin
  #10 A= 1'b0; B= 1'b0; C= 1'b0;
  #10 A= 1'b1; B= 1'b1; C= 1'b1;
  #10 A= 1'b1; B= 1'b0; C= 1'b0;
  #20 $finish;
end
endmodule
```

# Simulation with Delay Model

- `reg`-type variable is unknown (X) initially for NC Verilog
  - Different simulators may behave differently
    - Some other simulators assume that the initial value is zero
  - Pessimistic




- How about `wire`-type variable? Z

```
wire out;
```