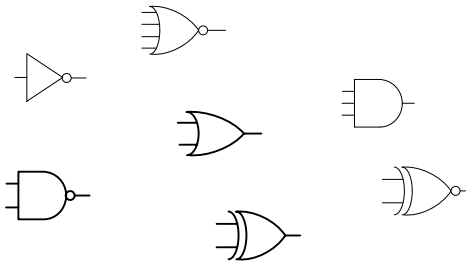
EE 330
Lecture 36

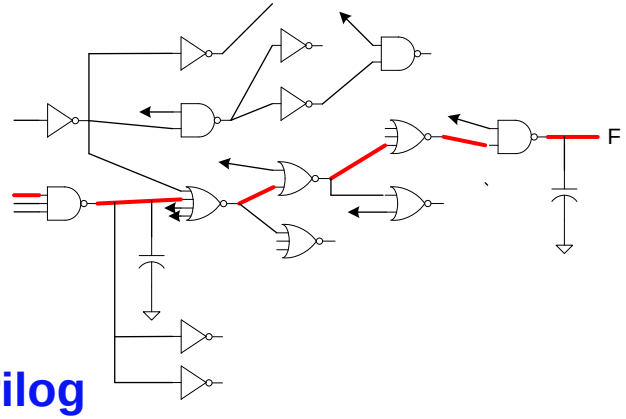
Digital Circuit Design

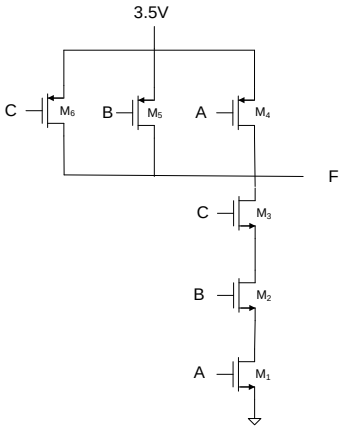
- Basic Logic Gates
- Properties of Logic Families
- Characterization of CMOS Inverter
- One device sizing strategy
- Multiple-input gates

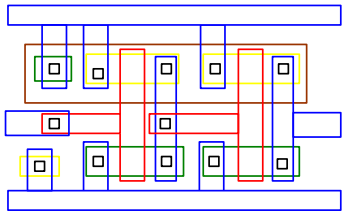
Digital Circuit Design

Most of the remainder of the course will be devoted to digital circuit design







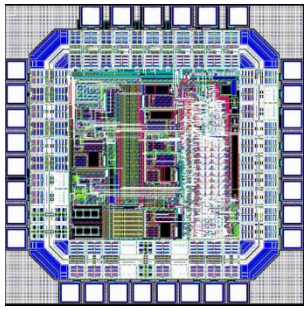


Verilog

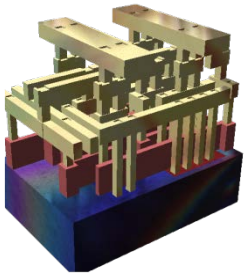
```
module gates (input logic [3:0] a,b,
              output logic [3:0] y1,y2,y3,y4,y5);
  assign y1 = a&b;  //AND
  assign y2 = a | b;  //OR
  assign y3 = a ^ b;  //XOR
  assign y4 = ~(a & b);  //NAND
  assign y5 = ~( a | b);  //NOR
endmodule
```

VHDL

```
library IEEE; use IEEE.STD_LOGIC_1164.all;

entity gates is
   port(a,b: in STD_LOGIC_VECTOR(3 dowto 0);
        y1,y2,y3,y4,y5:out STD_LOGIC_VECTOR(3 downto 0));
end;

architecture synth of gates is
begin

y1 <= a and b;
y2 <= a or b;
y3 <= a xor b;
y4 <= a nand b;
y5 <= a nor b;
end;
```



A rendering of a small standard cell with three metal layers (dielectric has been removed). The sand-colored structures are metal interconnect, with the vertical pillars being contacts, typically plugs of tungsten. The reddish structures are polysilicon gates, and the solid at the bottom is the crystalline silicon bulk

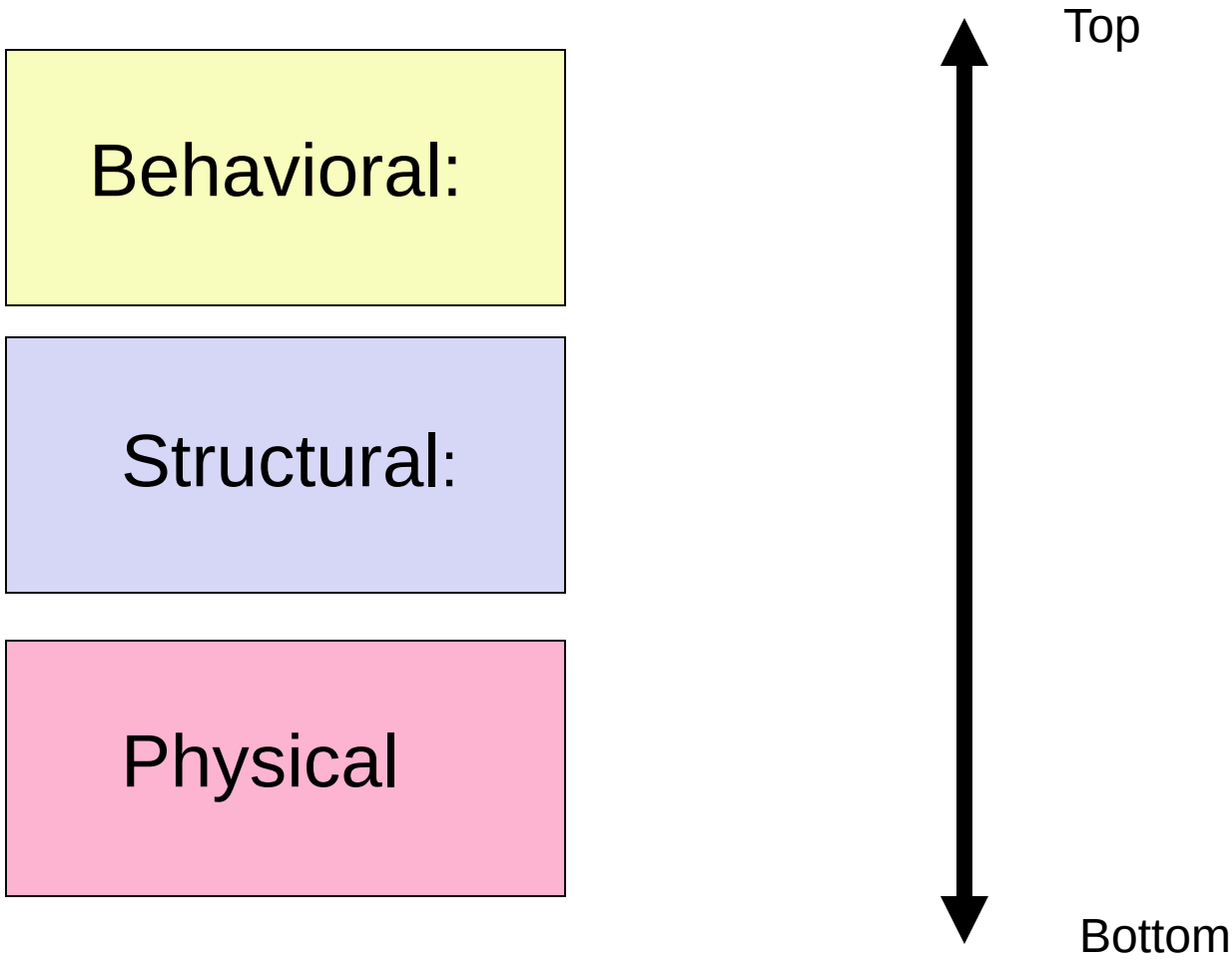


Standard Cell Library

Digital Circuit Design

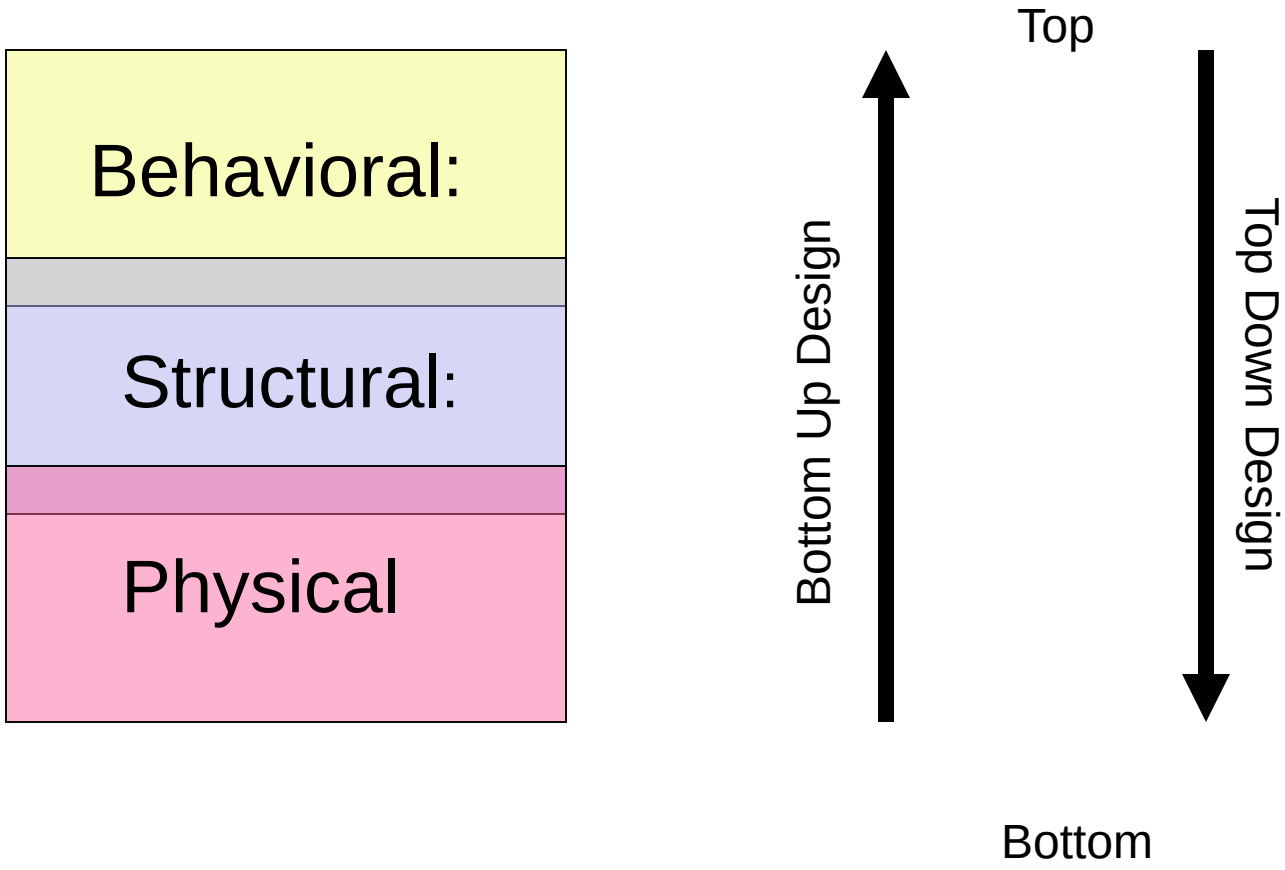
- Hierarchical Design
- Basic Logic Gates
- Properties of Logic Families
- Characterization of CMOS Inverter
- Static CMOS Logic Gates
  - Ratio Logic
- Propagation Delay
  - Simple analytical models
  - Elmore Delay
- Sizing of Gates
- Propagation Delay with Multiple Levels of Logic
- Optimal driving of Large Capacitive Loads
- Power Dissipation in Logic Circuits
- Other Logic Styles
- Array Logic
- Ring Oscillators

Hierarchical Digital Design Domains:

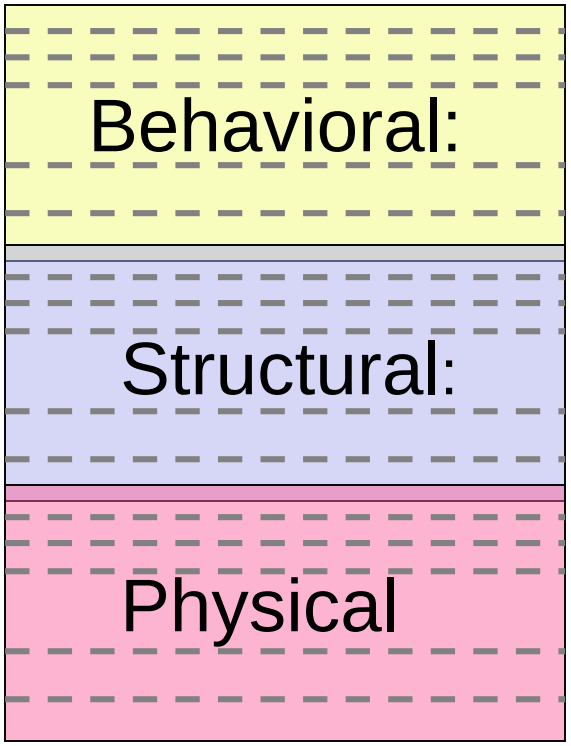


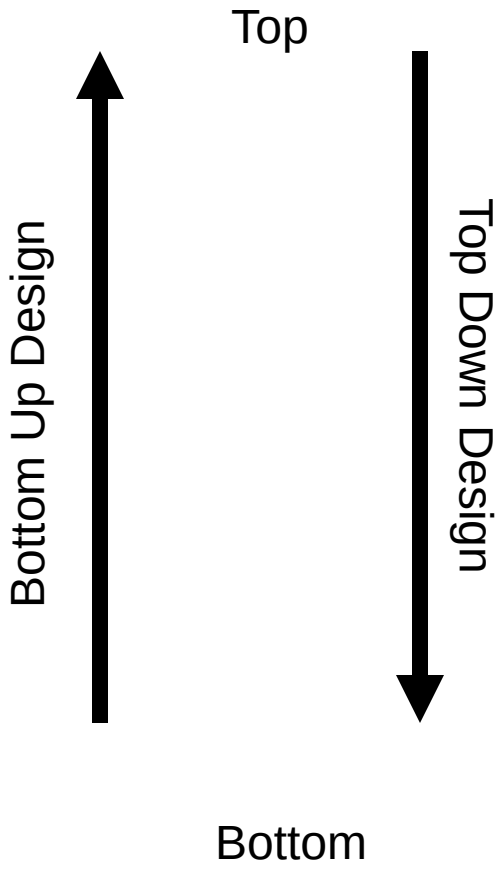
Multiple Levels of Abstraction

Hierarchical Digital Design Domains:



Hierarchical Digital Design Domains:





Multiple Sublevels in Each Major Level

All Design Steps may not Fit Naturally in this Description

Hierarchical Digital Design Domains:

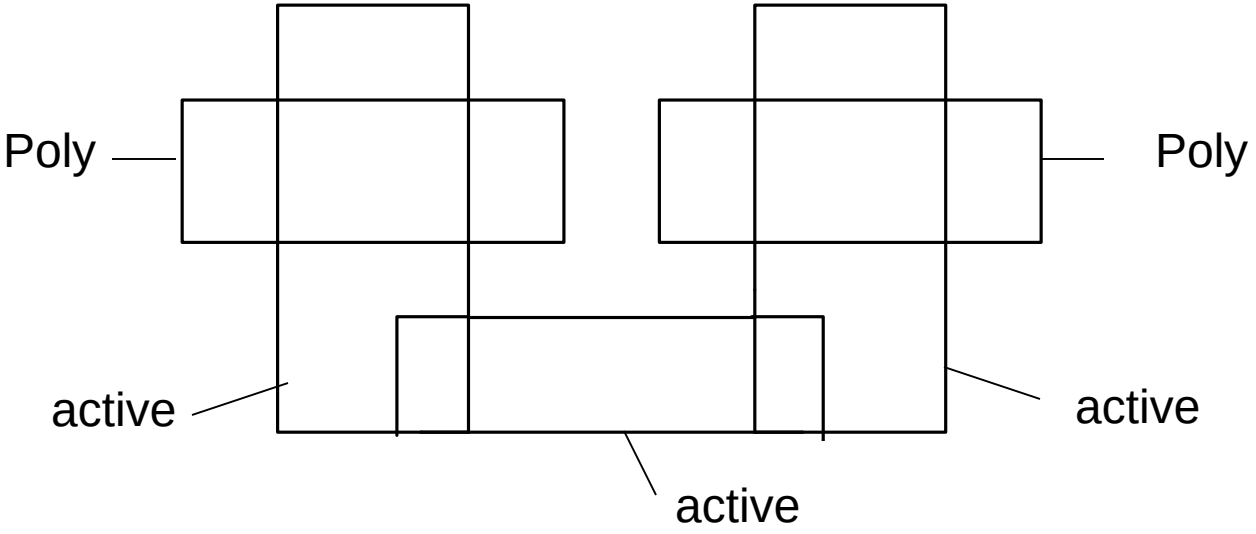
Behavioral : Describes what a system does or what it should do

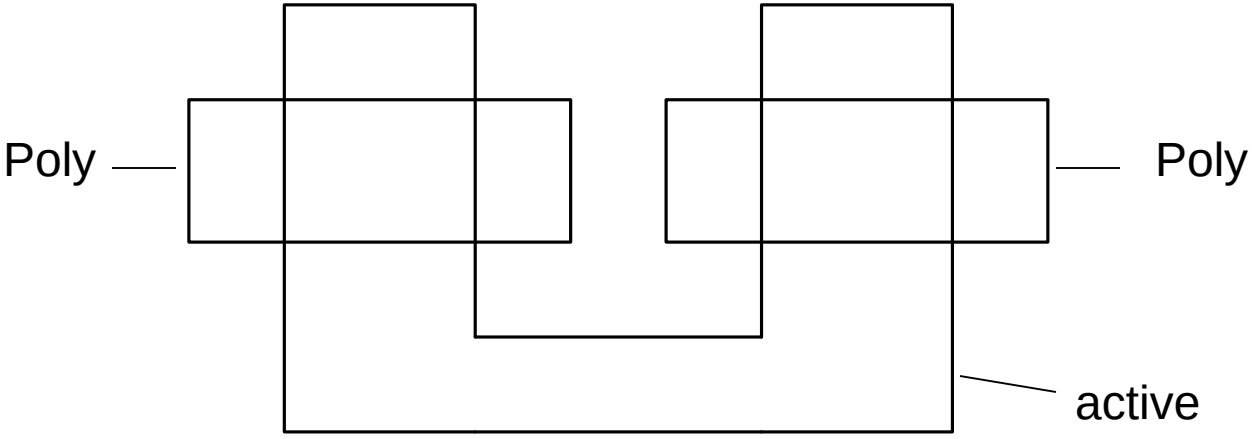
Structural : Identifies constituent blocks and describes how these blocks are interconnected and how they interact

Physical : Describes the constituent blocks to both the transistor and polygon level and their physical placement and interconnection

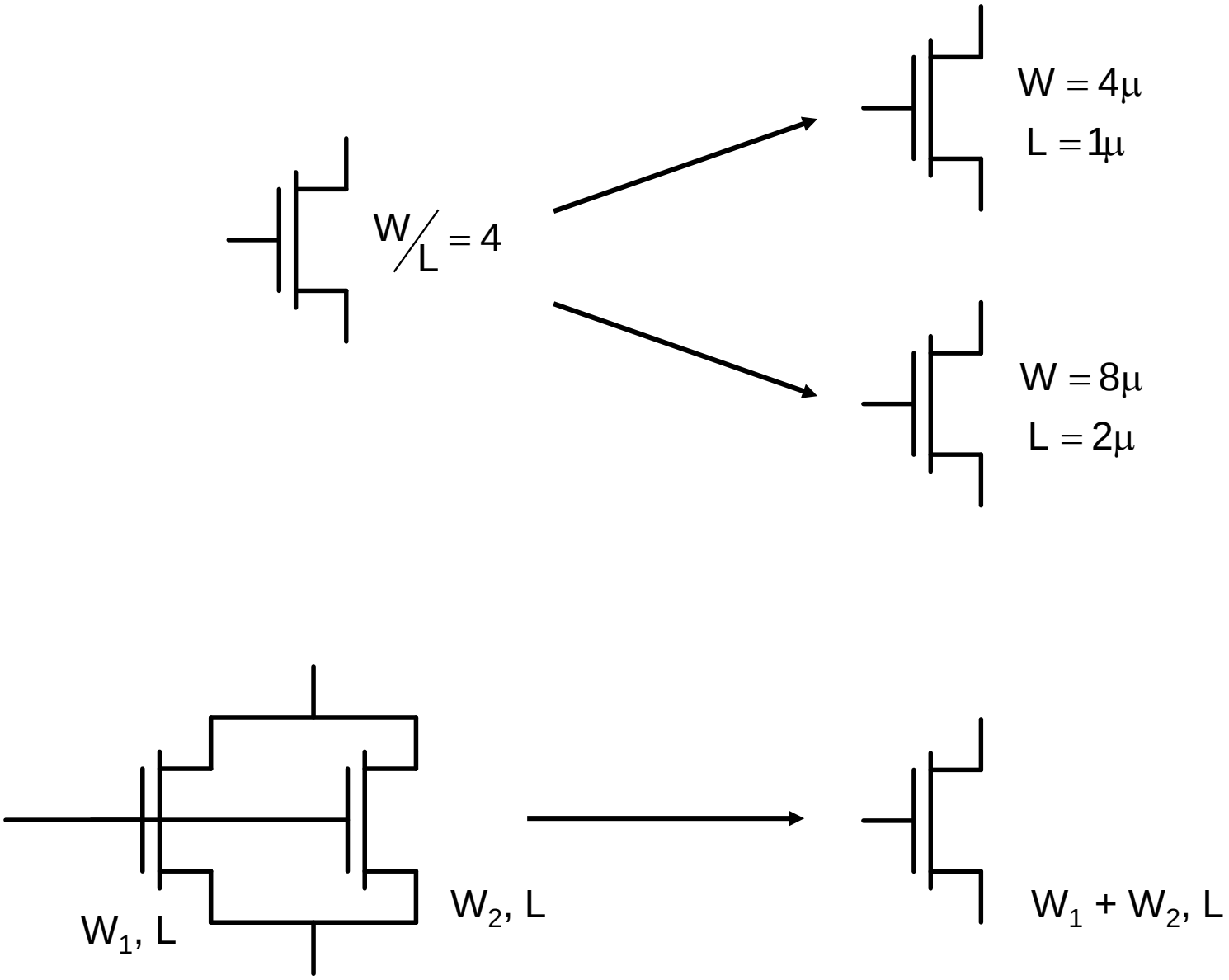
Multiple representations often exist at any level or sublevel

Example: Two distinct representations at the physical level (polygon sublevel)

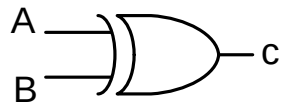




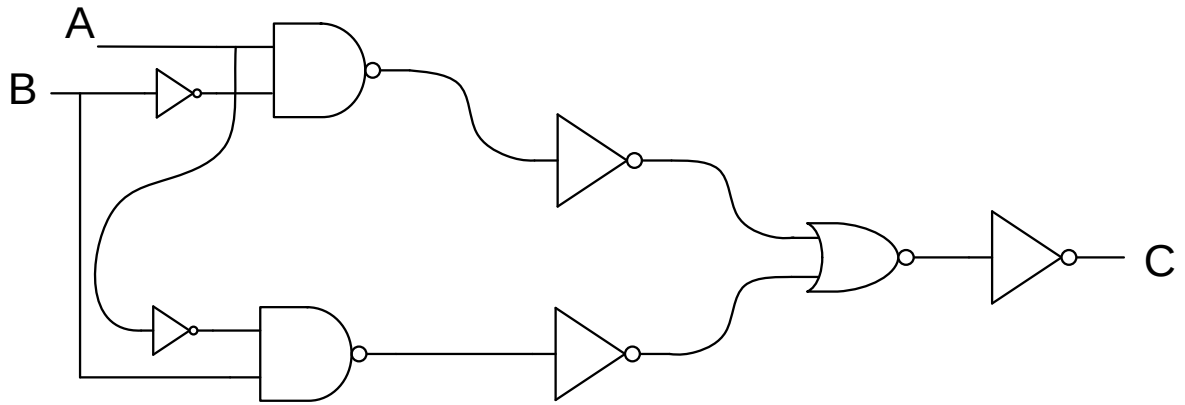
Example: Two distinct representations at physical level (schematic sublevel)

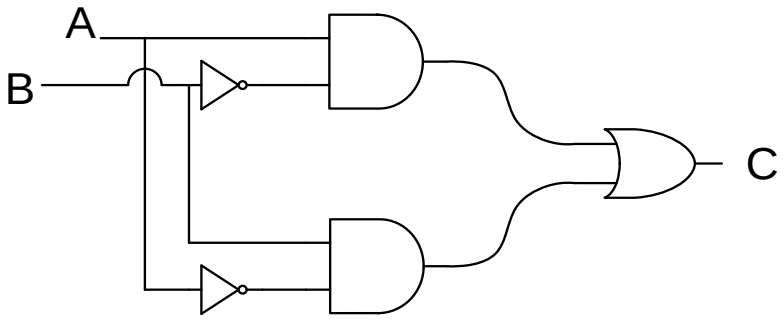


Example: Two distinct representations at the structural/behavioral level (gate sublevel)



$$C = A \oplus B$$





In each domain, multiple levels of abstraction are generally used.

Consider Physical Domain

– Consider lowest level to highest

0 - placement of diffusions, thin oxide regions, field oxide, ect. on a substrate.

1 - polygons identify all mask information (not unique)

2 - transistors (not unique)

3 - gate level (not unique)

4 - cell level Adders, Flip Flop, MUTs,…

Information Type

PG data
G.D.F
Netlist
HDL Description

Structural Level:

- DSP
- Blocks (Adders, Memory, Registers, etc.
- Gates
- Transistor

Information Type

HDL

Netlists

Behavior Level (top down):

- Application
- Programs
- Subroutines
- Boolean Expressions

Information Type

High-Level Language
HDL

Representation of Digital Systems

Standard Approach to Digital Circuit Design

8 – level representation

1. Behavioral Description
   - Technology independent
2. RTL Description (Register Transfer Level)

   (must verify (1) ⇔ (2))
3. RTL Compiler

   Registers and Combinational Logic Functions
4. Logic Optimizer
5. Logic Synthesis

   Generally use a standard call library for synthesis

(sublevels 6-8 not shown on this slide)

Frontend design

Representation of Digital Systems

Standard Approach to Digital Circuit Design

1. Behavioral Description
   - Technology independent
2. RTL Description

   (must verify (1) ⇔ (2))
3. RTL Compiler

   Registers and Combinational Logic Functions
4. Logic Optimizer

5. Logic Synthesis

   Generally use a standard call library for synthesis

Backend design

6. Place and Route

(physically locates all gates and registers and interconnects them)

7. Layout Extraction
   - DRC
   - Back Annotation
8. Post Layout simulation

   May necessitate a return to a higher level in the design flow

Logic synthesis, though extensively used, often is not as efficient nor as optimal for implementing some important blocks or some important functions

These applications generally involve transistor level logic circuit design that may combine one or more different logic design styles

Logic Optimization

What is optimized (or minimized) ?

- Number of Gates
- Number or Levels of Logic
- Speed
- Delay
- Power Dissipation
- Area
- Cost
- Peak Current
- • •

Depends Upon What User Is Interested In

Standard Cell Library

- Set of primitive building blocks that have been pre-characterized for dc and high frequency performance
- Generally includes basic multiple-input gates and flip flops
- P-cells often included
- Can include higher-level blocks
  - Adders, multipliers, shift registers,counters,…
- Cell library often augmented by specific needs of a group or customer

Digital Circuit Design

- Hierarchical Design
- Basic Logic Gates
- Properties of Logic Families
- Characterization of CMOS Inverter
- Static CMOS Logic Gates
  - Ratio Logic
- Propagation Delay
  - Simple analytical models
  - Elmore Delay
- Sizing of Gates
- Propagation Delay with Multiple Levels of Logic
- Optimal driving of Large Capacitive Loads
- Power Dissipation in Logic Circuits
- Other Logic Styles
- Array Logic
- Ring Oscillators

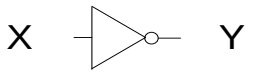
Logic Circuit Block Design

Many different logic design styles

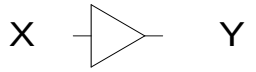
- Static Logic Gates
- Complex Logic Gates
- Pseudo NMOS
- Pass Transistor Logic
- Dynamic Logic Gates
  - Domino Logic
  - Zipper Logic
  - Output Prediction Logic

Various logic design styles often combined in the implementation of one logic block

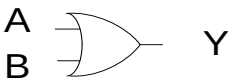
The basic logic gates



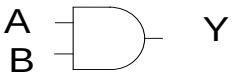
$$Y = \overline{X}$$



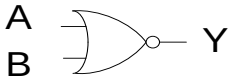
$$Y = X$$



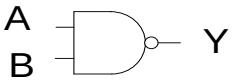
$$Y = A + B$$



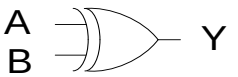
$$Y = A \bullet B$$



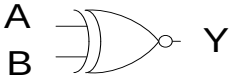
$$Y = \overline{A + B}$$



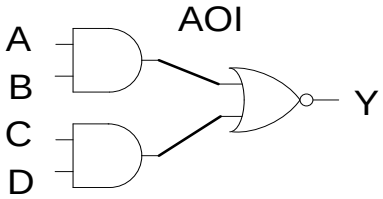
$$Y = \overline{A \bullet B}$$



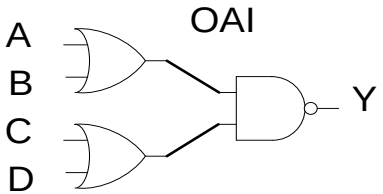
$$Y = A \oplus B$$



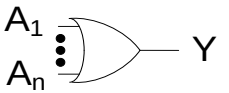
$$Y = \overline{A \oplus B}$$



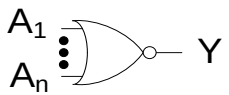
$$Y = \overline{A \bullet B + C \bullet D}$$



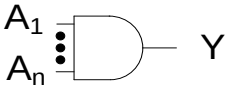
$$Y = \overline{(A + B) \bullet (C + D)}$$



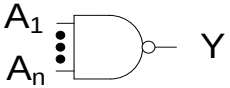
$$Y = A_1 + A_2 + \ldots A_n$$



$$Y = \overline{A_1 + A_2 + \ldots A_n}$$

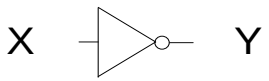


$$Y = A_1 \bullet A_2 \bullet \ldots A_n$$

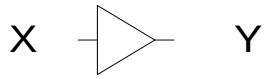


$$Y = \overline{A_1 \bullet A_2 \bullet \ldots A_n}$$

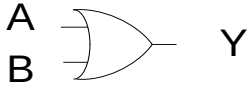
The basic logic gates



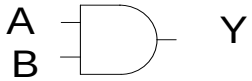
$$Y = \overline{X}$$



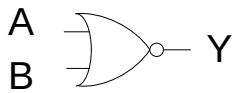
$$Y = X$$



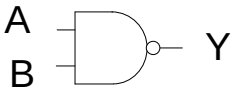
$$Y = A + B$$



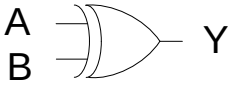
$$Y = A \bullet B$$



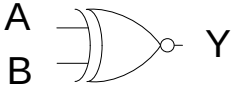
$$Y = \overline{A + B}$$



$$Y = \overline{A \bullet B}$$



$$Y = A \oplus B$$



$$Y = \overline{A \oplus B}$$

Question: How many basic one and two input gates exist and how many of these are useful?

The basic logic gates

The set of NOR gates is complete

Any combinational logic function can be realized with only multiple-input NOR gates

The set of NAND gates is complete

Any combinational logic function can be realized with only multiple-input NOR gates

Performance of the BASIC gates is critical!

The basic logic gates

A gate logic family can be formed based upon a specific design style for implementing logic functions

Many different gate logic family types exist

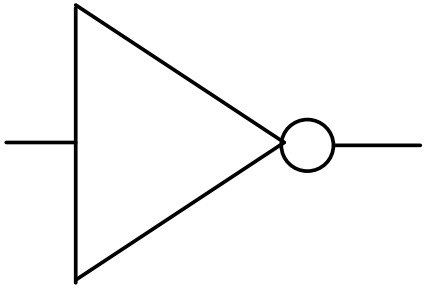
NMOS, PMOS, CMOS, TTL, ECL, RTL, DCTL,…

Substantial differences in performance from one family type to another

Power, Area, Noise Margins, ….

The basic logic gates

It suffices to characterize the inverter of a logic family and then express the performance of other gates in that family in terms of the performance of the inverter.



What characteristics are required and desirable for an inverter to form the basis for a useful logic family?

What restrictions are there on the designer for building Boolean circuits?

- None !!!!
- It must “work” as expected
- Designer is Master of the silicon !

Desirable and/or Required Logic Family Characteristics

1. High and low logic levels must be uniquely distinguishable (even in a long cascade)
2. Capable of driving many loads (good fanout)
3. Fast transition times (but in some cases, not too fast)
4. Good noise margins (low error probabilities)
5. Small die area
6. Low power consumption
7. Economical process requirements

Desirable and/or Required Logic Family Characteristics

8. Minimal noise injection to substrate
9. Low leakage currents
10. No oscillations during transitions
11. Compatible with synthesis tools
12. Characteristics do not degrade too much with temperature
13. Characteristics do not vary too much with process variations

Are some of these more important than others?

Desirable and/or Required Logic Family Characteristics

8. Minimal noise injection to substrate
9. Low leakage currents
10. No oscillations during transitions
11. Compatible with synthesis tools
12. Characteristics do not degrade too much with temperature
13. Characteristics do not vary too much with process variations

Are some of these more important than others?

Yes ! – must have well-defined logic levels for circuits to even function as logic

Desirable and/or Required Logic Family Characteristics

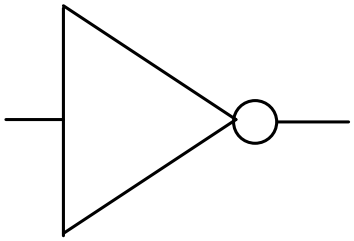
Are some of these more important than others?

Yes ! – must have well-defined logic levels for circuits to even function as logic

What properties of an inverter are necessary for it to be useful for building a logic family

What are the logic levels for a given inverter of for a given logic family?

What are the logic levels for a given inverter of for a given logic family?



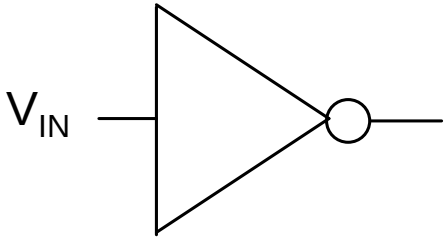
V_H=?

V_L=?

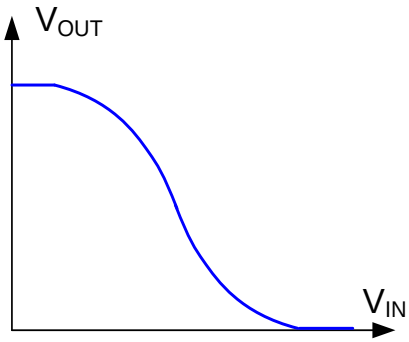
Can we legislate them ?

- Some authors choose to simply define a value for them
- Simple and straightforward approach
- **But what if the circuit does not interpret them the same way they are defined !!**

Ask the inverter how it will interpret logic levels



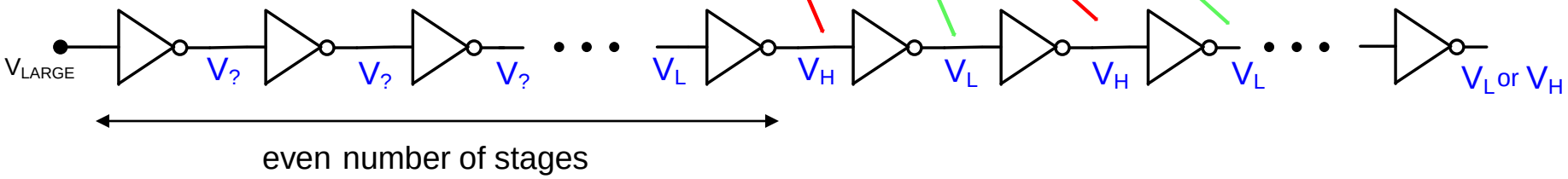
V_{OUT} V_H=? V_L=?



Consider a very long cascade of inverters

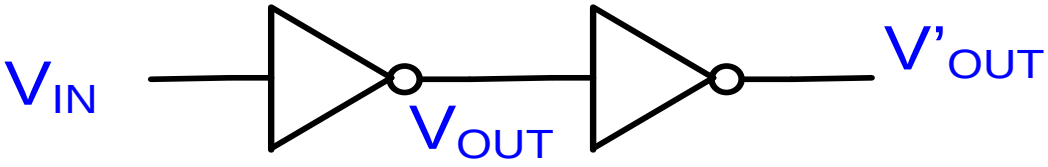
Apply a large voltage at the input (alternatively a small input could be used)

w.l.o.g. assume an even number of inverters in chain indicated

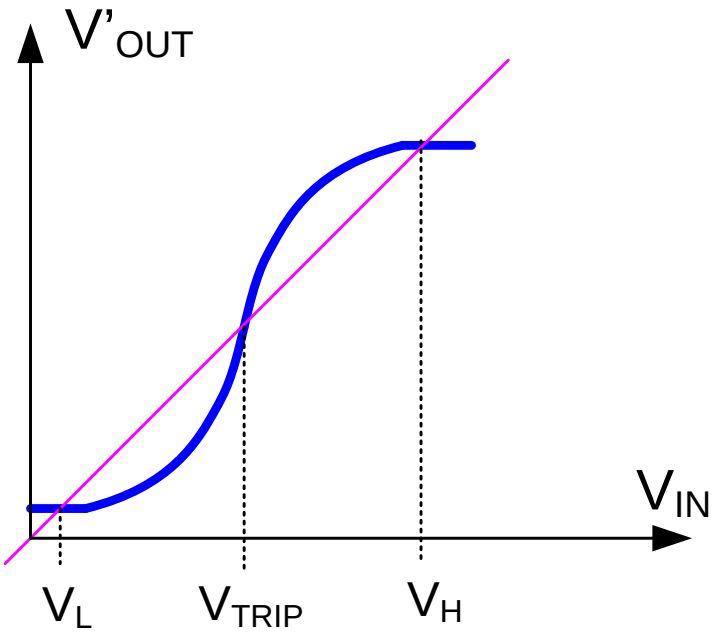


If logic levels are to be maintained, the voltage at the end of this even number of stages must be V_H, that of the next must be V_L, the next V_H, etc. until the start of the cascade is approached

Ask the inverter how it will interpret logic levels



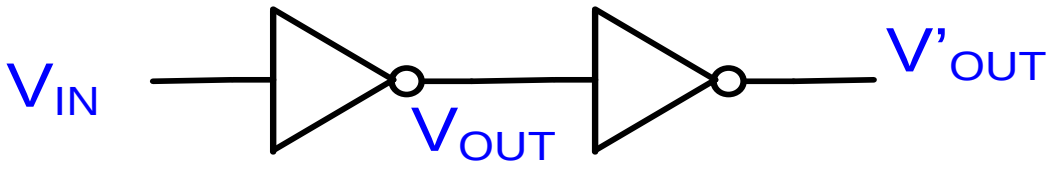
Inverter pair



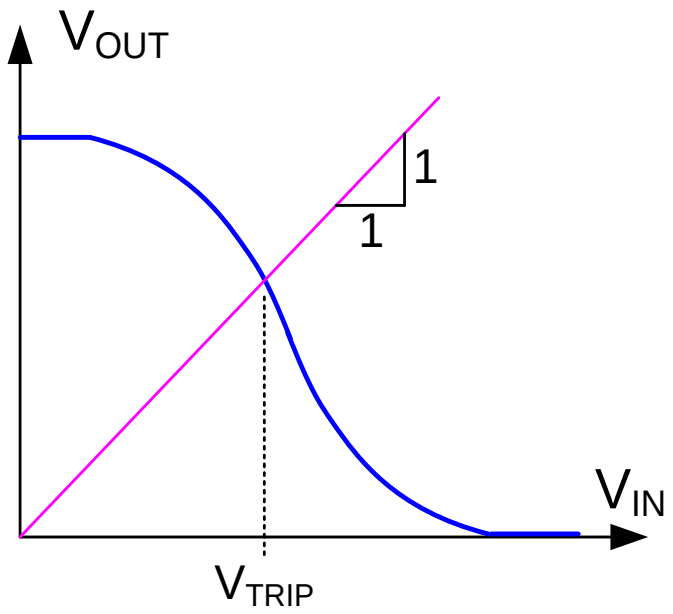
When $V'_{OUT}=V_{IN}$, V_H and V_L are stable operating points, V_{TRIP} is a quasi-stable operating point

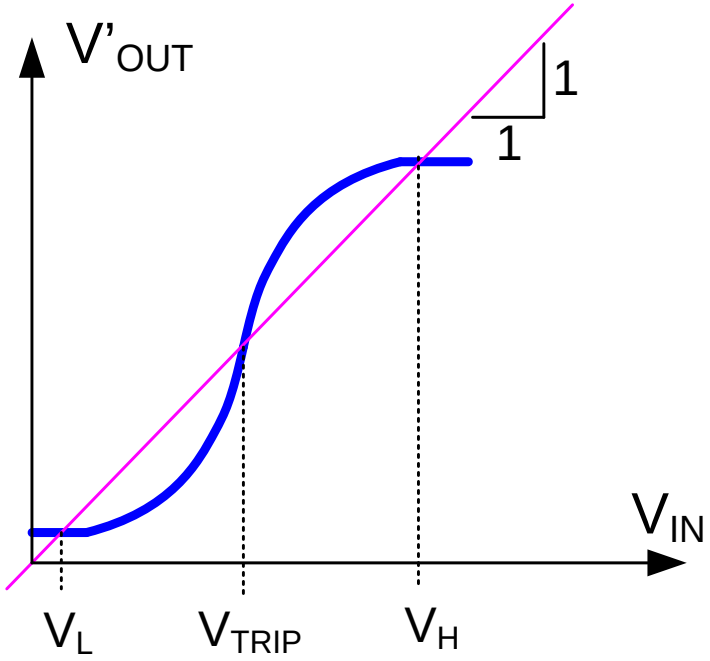
Observe: slope of IPTC is greater than 1 at V_{TRIP} and less than 1 at V_H and V_L

Observation



When $V_{OUT}=V_{IN}$ for the inverter, V'_{OUT} is also equal to V_{IN}. Thus the intersection point for $V_{OUT}=V_{IN}$ in the inverter transfer characteristics (ITC) is also an intersection point for $V'_{OUT}=V_{IN}$ in the inverter-pair transfer characteristics (IPTC)





Implication: Inverter characteristics can be used directly to obtain V_{TRIP}

End of Lecture 36