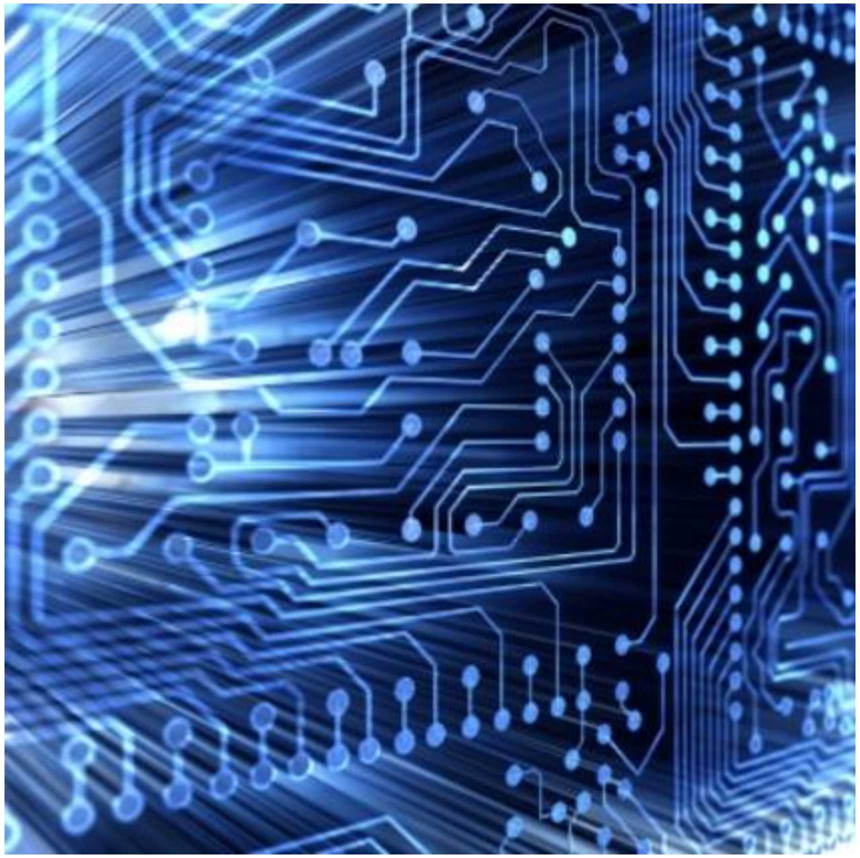
Digital Microelectronic Circuits (361-1-3021)

Presented by: Adam Teman



Lecture 6: CMOS Digital Logic

Last Lectures

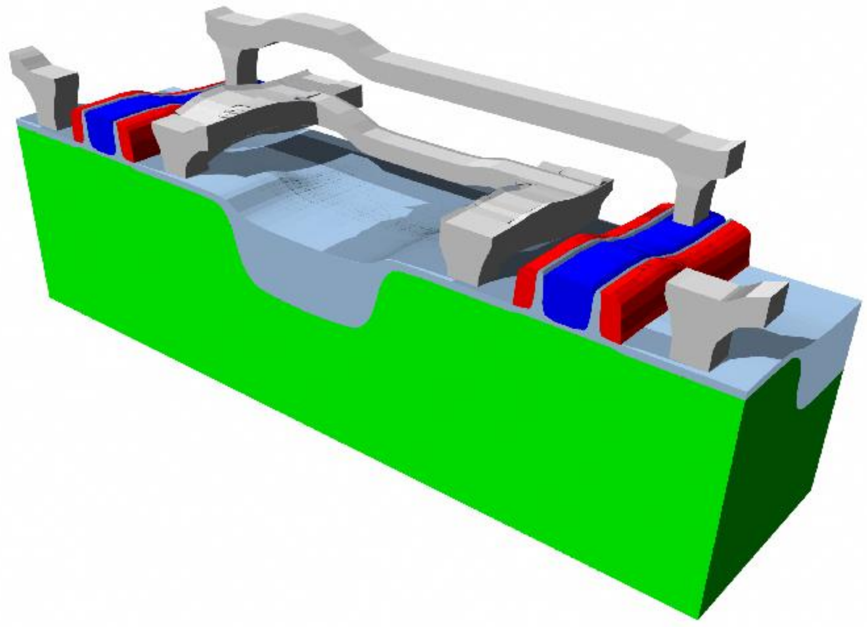
- The CMOS Inverter
- CMOS Capacitance
- Driving a Load

This Lecture

- Now that we know all about an inverter, you are probably asking "*So what can we do with it*"?
- Well, with an inverter by itself, you can't do much, but the principles in constructing and analyzing the ***CMOS inverter*** *are the same* as those used to make up the whole ***CMOS digital logic*** family.
- During this lecture, we will learn how to compile *any combinational Boolean function* from ***CMOS logic*** and analyze the pros and cons.

What will we learn today?

6.1

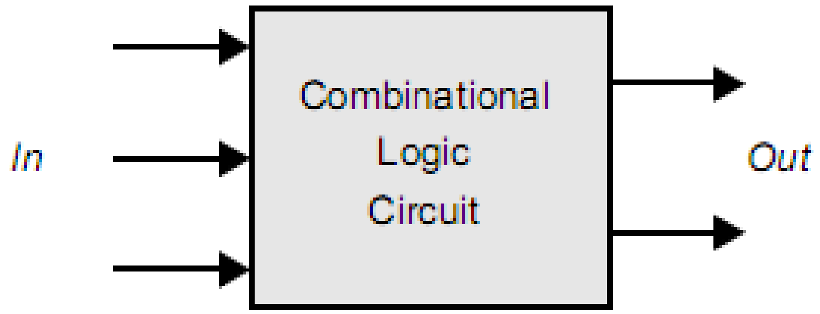


Better start with the

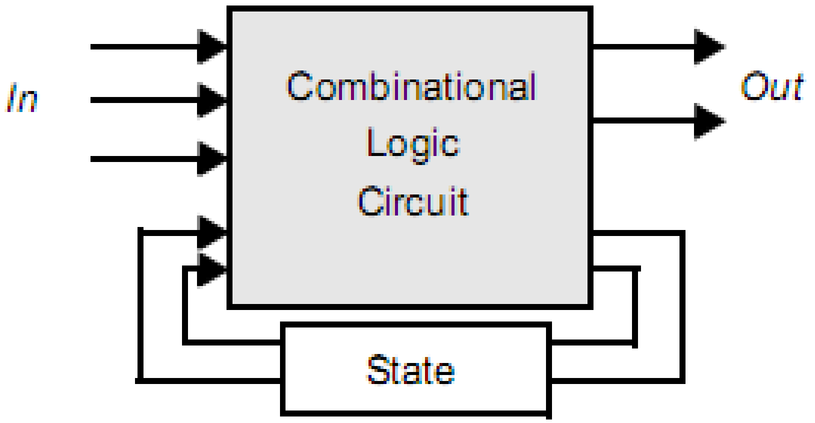
CMOS BASIC CONCEPT

Classification

- A first classification of logic circuits or logic gates is ***Combinational*** vs. ***Sequential*** operation.
  - ***Combinational*** logic (a.k.a. *non-regenerative*) circuits are characterized by an output that at any point in time is a function of the current inputs by some Boolean expression.
  - ***Sequential*** (a.k.a. *regenerative*) circuits, are a function of the current inputs as well as previous states through feedback.



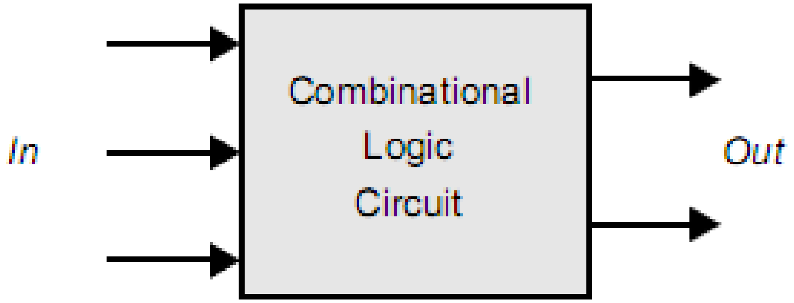
(a) Combinational



(b) Sequential

Classification

- The focus of this lecture is ***Static CMOS*** or ***Complementary CMOS*** implementation of ***Combinational Logic***.



- ***Sequential*** circuits will be discussed in a later lecture.

Classification

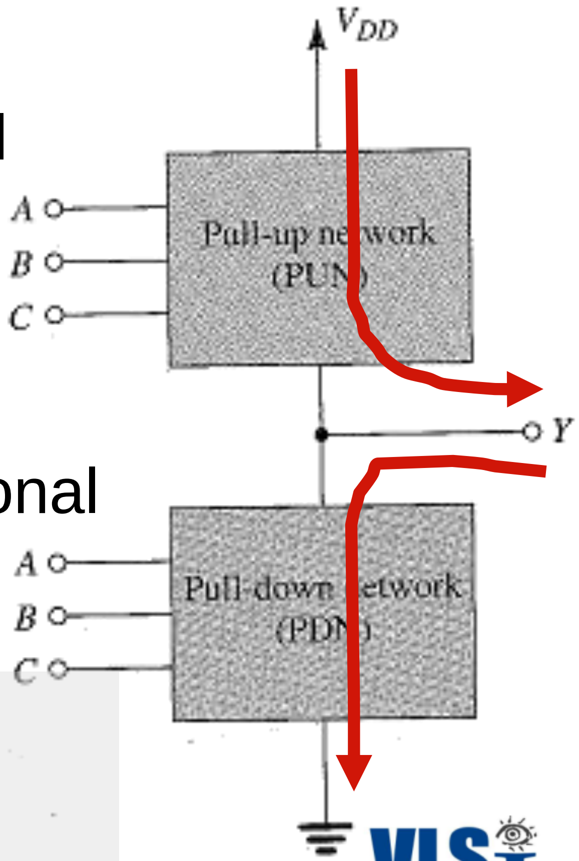
- Another classification of logic circuits is ***Static*** vs. ***Dynamic*** Logic Circuits.
  - ***Static Circuits*** assume the value of the Boolean function implemented by the circuit *at all times* (after the initial transient).
  - ***Dynamic Circuits*** *temporarily* store a predefined value, and only show the function output *after a fixed interval*.

Classification

- ***Static Circuits*** are *easier to implement* and provide *lower power consumption* when the activity factor is low.
- ***Dynamic Circuits*** provide *faster* and *smaller* gates, but design and operation are more *complex* and they are mores *sensitive to noise*.
- This lecture will discuss ***Static*** or ***Complementary CMOS***, obviously belonging to the ***Static*** classification.

CMOS Structure

- Let's remember the concepts of a ***Pull Up Network*** (***PUN***) and ***Pull Down Network*** (***PDN***) that we discussed in *Lecture 2*:
  - » A ***Pull Up Network*** is a combinational block that presents a high output level when activated.
  - » A ***Pull Down Network*** is a combinational block that presents a low output level when activated.

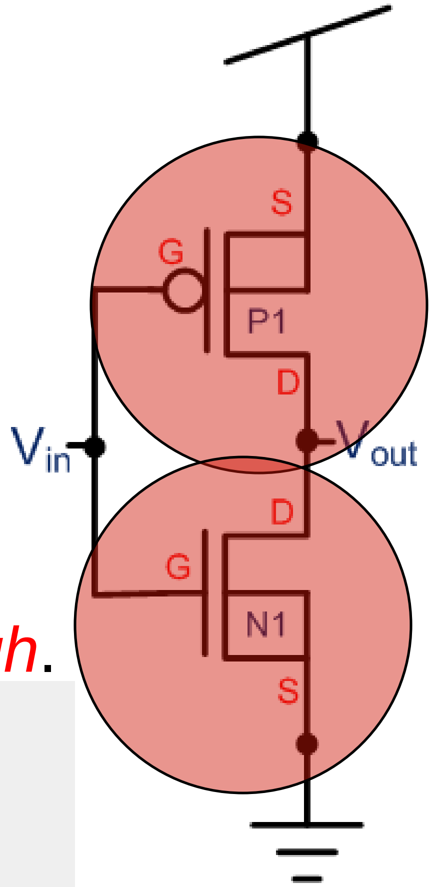


CMOS Structure

- In the case of the ***CMOS inverter***:

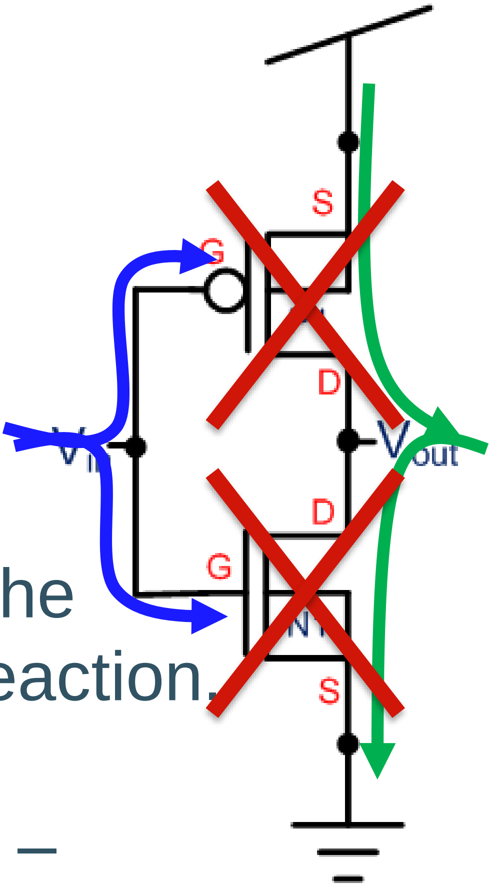
  - » The ***PUN*** was the ***pMOS*** that connected V_{DD} to the output when the input was *low*.

  - » The ***PDN*** was the ***nMOS*** that connected ***GND*** to the output when the input was *high*.



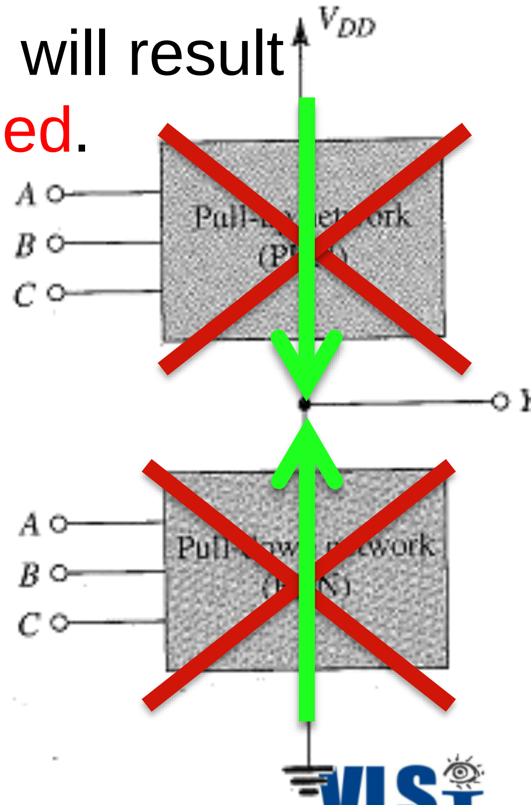
CMOS Structure

- Looking more closely at the ***CMOS inverter***, we can see that:
  - Applying a *positive value* to the input opened the ***PDN*** and closed the ***PUN***.
  - Applying a *negative value* to the input closed the ***PDN*** and opened the ***PUN***.
- Essentially, applying the same input to the ***PUN*** and ***PDN*** resulted in an opposite reaction.
- This is the basic characteristic of ***CMOS*** – *complementary pull up and pull down networks*.



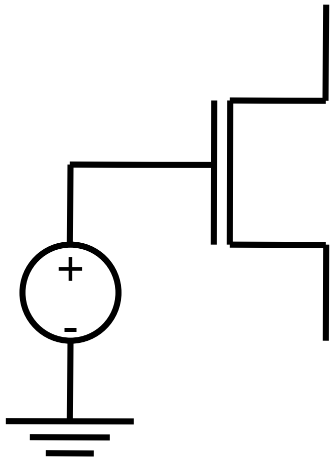
CMOS Structure

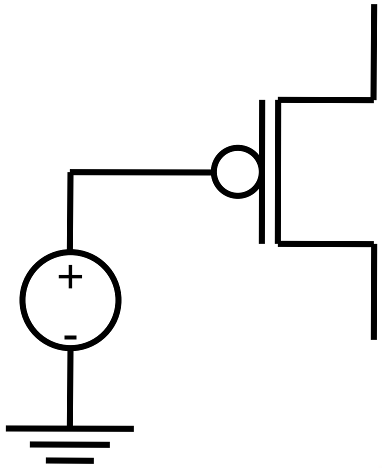
- Extending the concept beyond the inverter:
  - » We can apply any ***N-inputs*** to both the ***PUN*** and ***PDN***.
  - » The logic functions of the ***PUN*** and ***PDN*** will be *complementary* so all input combinations will result in one of the two open and the other closed.
  - » If the output is meant to be ***'1'***, the ***PUN*** will be open and the ***PDN*** will be closed, resulting in the propagation of V_{DD} to the output.
  - » If the output is meant to be ***'0'*** the ***PDN*** will be open and the ***PUN*** will be closed, resulting in the propagation of ***GND*** to the output.



Constructing the PUN/PDN

- How do we construct the ***PUN*** and ***PDN***?
- Let's look at a ***MOSFET*** transistor as a *voltage-controlled switch*:
  - » An ***nMOS*** is *on* when the input is *high* and *off* when it is *low*.
  - » A ***pMOS*** is an inverted ***nMOS*** – on when the input is *low* and *off* when the input is *high*.



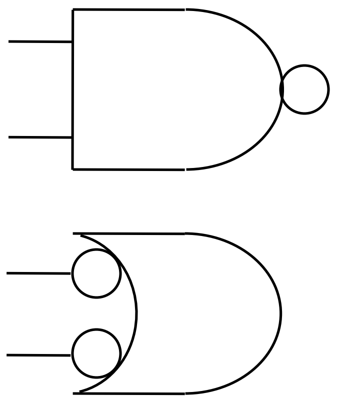


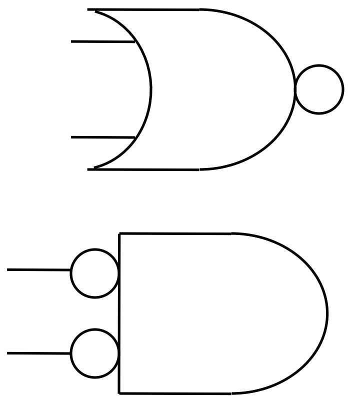
Constructing the PUN/PDN

- Taking this into account, we need to create complementary reactions to the same inputs.
  - » For example, let's look at the ***NAND*** and ***NOR*** functions.
  - » Using inverted inputs, we can arrive at the same logic function:

$$\boxed{\overline{A \cdot B} = \overline{A} + \overline{B}}$$

$$\boxed{\overline{A + B} = \overline{A} \cdot \overline{B}}$$





Constructing the PUN/PDN

- Interestingly, we have a perfect solution to produce these *complementary* logic expressions.
- Remember our good old friend ***DeMorgan***.

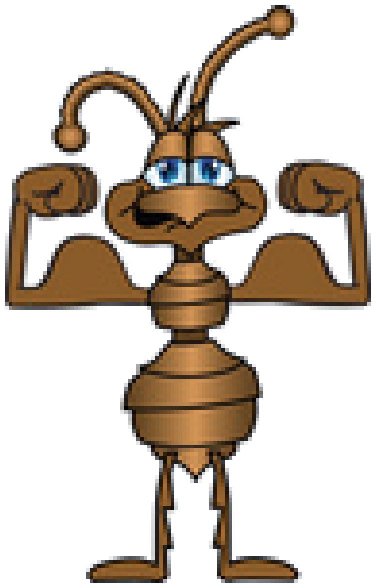
$$\overline{A+B} = \overline{A}\cdot\overline{B}$$

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

- Don't ***DeMorgan's Equations*** look just like the expressions on the last slide?
- Using ***DeMorgan***, we are able to make a ***Pull Up*** and ***Pull Down Network*** of the same ***Boolean Function***!

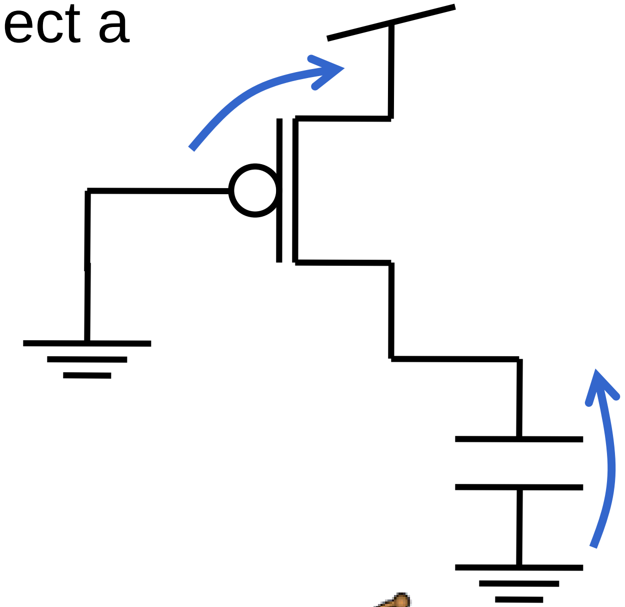
Constructing the PUN/PDN

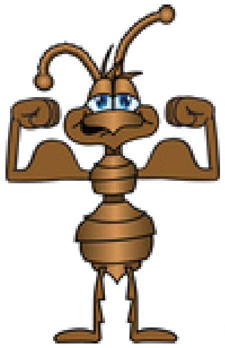
- Now comes the question of who to put on *top* and who on the *bottom*…
  - Using our engineering intuition, we can look at the ***CMOS inverter*** and realize that the ***pMOS*** is probably in the ***Pull Up***…
  - But Why???
- The answer lies in a very important characteristic of the ***MOSFET*** transistor.
  - A ***pMOS*** is a device that generates "***Strong Ones***".
  - An ***nMOS*** is a device that generates "***Strong Zeros***".



Constructing the PUN/PDN

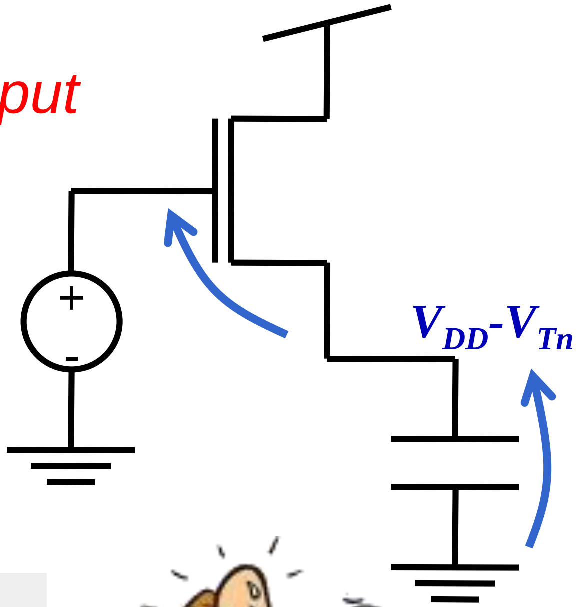
- So what are "***Strong Ones***" and "***Strong Zeros***"?
  - Let's see what happens when we connect a ***pMOS*** in a ***Pull Up Network***:
  - As we can see, $V_{SG}=V_{DD}>|V_{Tp}|$ *independent of the output voltage*.
  - Therefore, the output capacitance will load until $V_{out}=V_{DD}$ causing $V_{SD}=0$.
- We've provided a "***Strong One***".

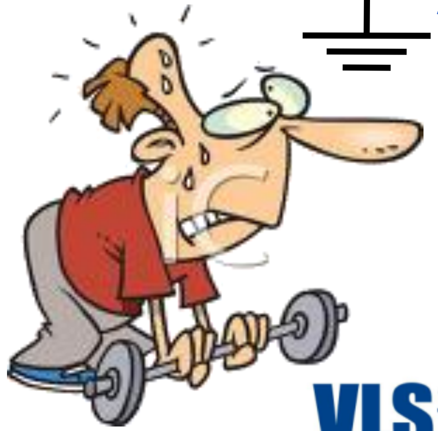




Constructing the PUN/PDN

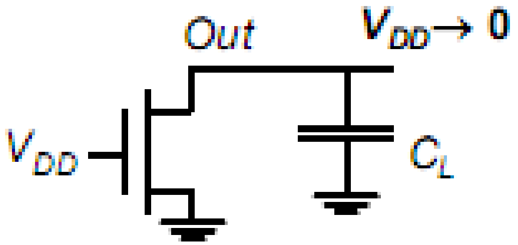
- On the other hand, if we connect an ***nMOS***…
  - Now, $V_{GS}=V_{DD}-V_{out}$, in other words, the state of the transistor *depends on the output voltage*.
  - Once the output reaches $V_{DD}-V_{Tn}$, the transistor *switches off*, and the output capacitance *stops charging*.
- We've provided a "***Weak One***".
- Obviously, we'll prefer ***pMOS*** transistors in our ***Pull Up Networks***!

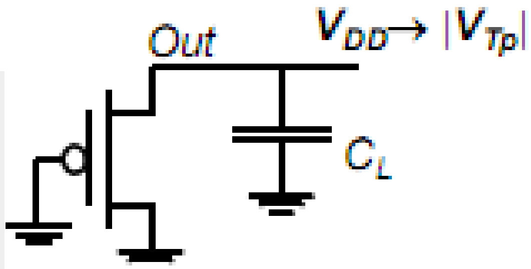




Constructing the PUN/PDN

- The same can be shown for a Pull Down Network.
  - » Here, an ***nMOS*** provides a "***Strong Zero***", as $V_{GS}=V_{DD}$ independent of the output voltage.
  - » A ***pMOS***, on the other hand, provides a "***Weak Zero***", only charging the output to $V_{DD}-|V_{Tp}|$.

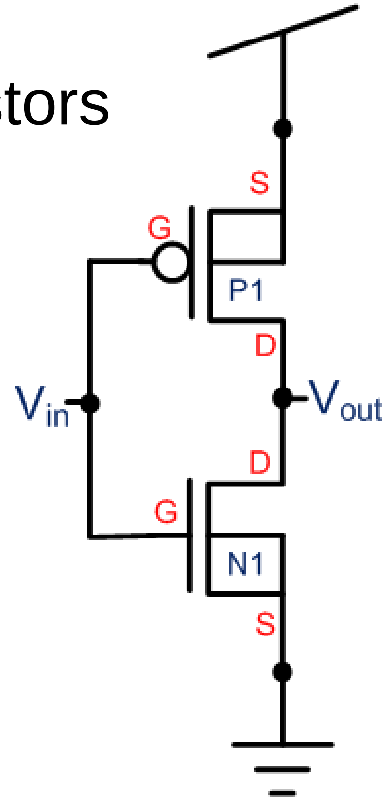




- We will use ***nMOS*** transistors in our ***Pull Down Networks***!

Constructing the PUN/PDN

- To summarize, we saw that:
  - » To get a "***Strong One***", we need ***pMOS*** transistors in the ***PUN***.
  - » To get a "***Strong Zero***", we need ***nMOS*** transistors in the ***PDN***.
- This is the basic concept in constructing a ***Static CMOS*** logic gate.
- Again, we can look at the ***CMOS Inverter*** and see how our basic concept is used.



Summary

- Now we know:
  - » How to write our logic functions (***DeMorgan***)
  - » What our building blocks are (***PUN=pMOS***, ***PDN=nMOS***).

- The only thing we're missing is how to *implement* the functions in the ***PUN*** and ***PDN***.

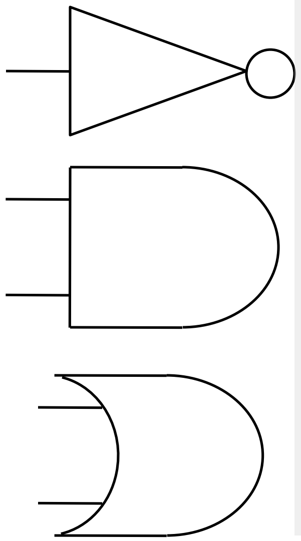
6.2

Okay, we understand the concept. It's time for

CMOS DIGITAL LOGIC IMPLEMENTATION

Universality

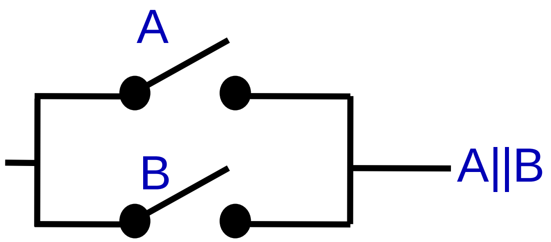
- According to ***Boolean Algebra***, to make *any function*, we need a ***universal set***, comprising:
  - » An ***Inverter*** (***NOT***)
  - » An ***AND*** function
  - » An ***OR*** function

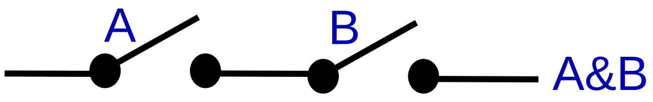


- Luckily, they are very easy to implement using ***MOSFETs***.

Series/Parallel Connection

- Back in *Lecture 3*, we discussed the "***Switching Concept***":
  - Connecting Switches in ***Parallel*** provides us with an ***OR*** function.
  - Connecting Switches in ***Serial*** provides us with an ***AND*** function.

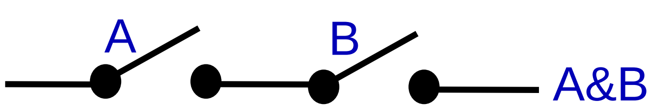


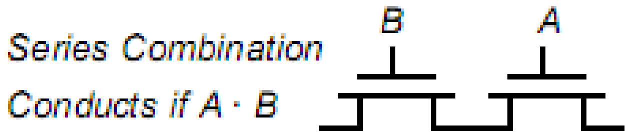


- Now all we have to do is replace our switches with ***MOSFET*** transistors.

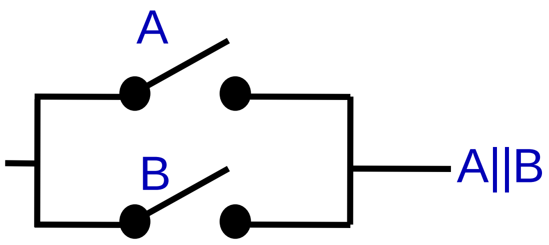
Series/Parallel Connection

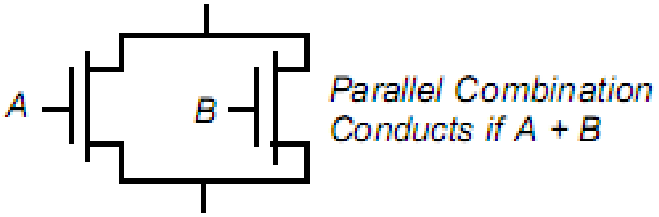
- Let's see what happens with ***nMOS*** transistors:
  - Series – AND – connection:





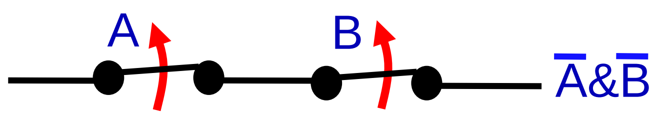
  - Parallel – OR – connection:

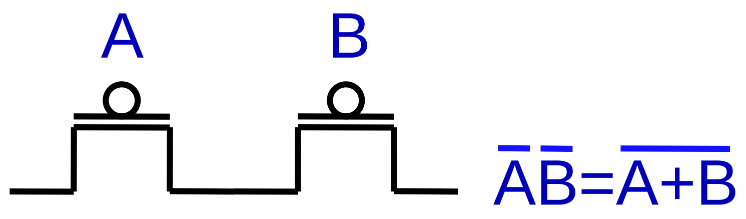




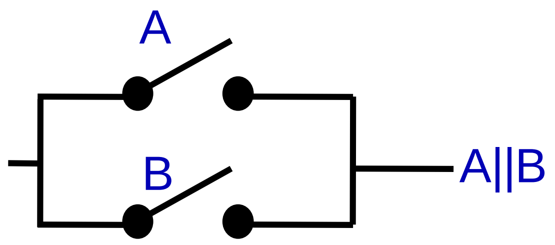
Series/Parallel Connection

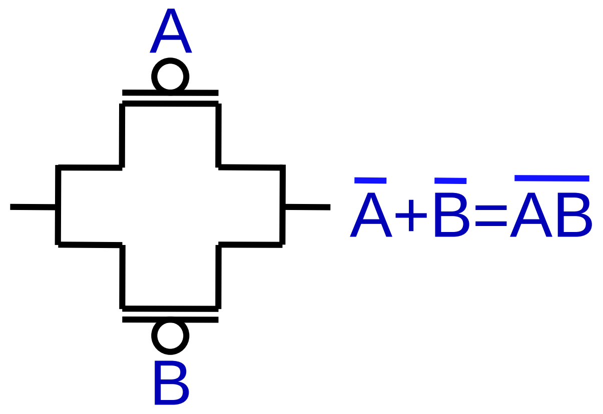
- We need a *low input* to control ***pMOS*** transistors, so we get an *inverting* function:
  - ***pMOS*** in Series:





  - Parallel ***pMOS*** connection:

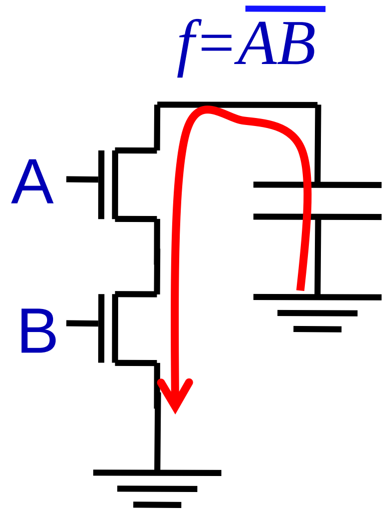




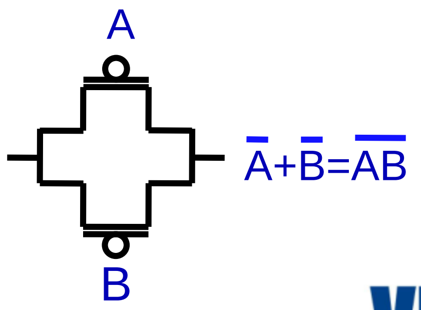
- With ***pMOS*** transistors, we've created ***NOR*** and ***NAND*** functions!

Complementary Networks

- So we saw that we can easily create *inverting functions* (***NAND, NOR***) using ***pMOS*** transistors and *non-inverting functions* (***AND, OR***) using ***nMOS***.
- Amazingly, this works out perfectly:
  - » When we implement a function in the ***PDN***, it *discharges* the capacitance, or *inverts* the output.
  - » So if we use ***nMOS*** transistors in the ***PDN***, we are getting *inverting functions*, just like the ***pMOS*** we'll use in the ***PUN***!

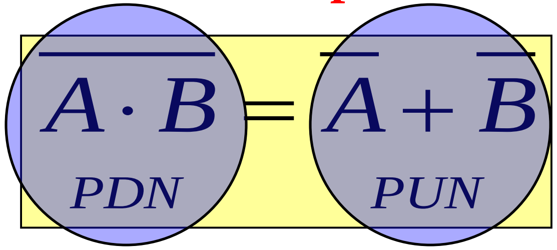


- So can we get a ***NOR*** by using a pair of parallel ***nMOS*** in the ***PDN*** and a pair of parallel ***pMOS*** in the ***PUN***?
  - » No, the parallel pair of ***pMOS*** give us a ***NAND***, remember?...

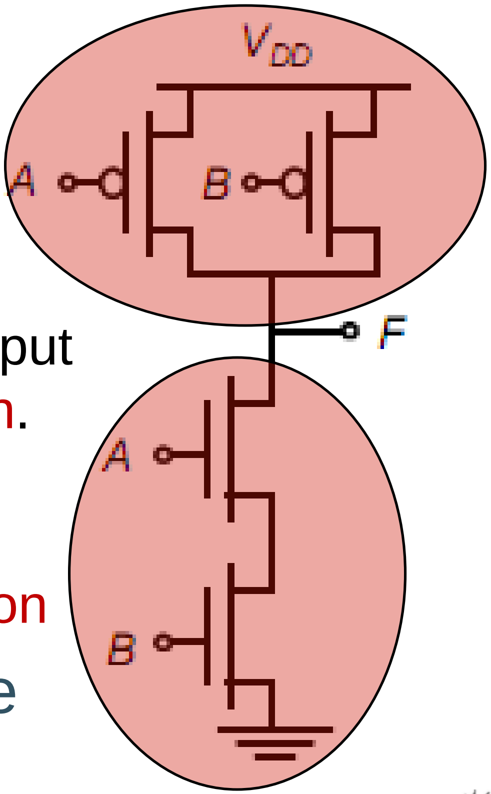


Complementary Networks

- How then, do we realize our function properly?
  - You've probably figured out by now that a parallel pair of transistors in the ***PDN*** cooperates with a series pair in the ***PUN***.
  - Let's look at the ***2-input NAND***:



  - Discharging the load capacitance *inverts* the output of the ***PDN***, giving us the left side of the equation.
  - The "*little circle*" on the ***pMOS*** inverts the inputs to the ***PUN***, giving us the right side of the equation



- We've used *complementary functions* in the ***PUN*** and ***PDN*** to realize our function.

Complementary Networks

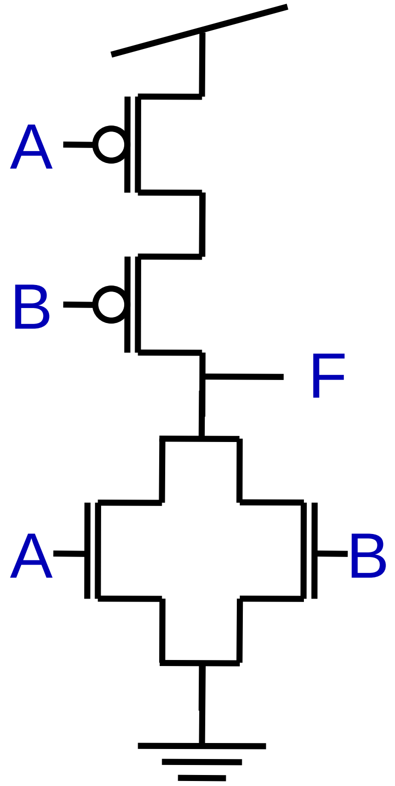
A	B	PUN	PDN	OUT
0	0			
0	1			
1	0			
1	1			

Complementary Networks

- Now, we'll try the same method to construct a ***2-input NOR***:

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

  - » The left side of the equation is *inverting*, describing a parallel ***PDN***.
  - » The right side is *non-inverting* with inverted inputs. This, of course, is the series ***PUN***.
  - » We've constructed a ***NOR*** gate with *complementary* ***Pull Up*** and ***Pull Down Networks***.



Complementary Networks

A	B	PUN	PDN	OUT
0	0			
0	1			
1	0			
1	1			

A few conclusions

- Looking at the ***NAND*** and ***NOR*** gates, we can reach a few conclusions about ***CMOS Digital Logic***:

  - We have constructed a *Universal Set*, meaning we can create any ***Boolean Function*** with ***CMOS***.

  - The *complementary gate* is naturally inverting, implementing only functions such as ***NAND***, ***NOR*** and ***XNOR***. To implement non-inverting functions (***AND***, ***OR***, ***XOR***, etc.) we must add an ***Inverter*** after the initial stage.

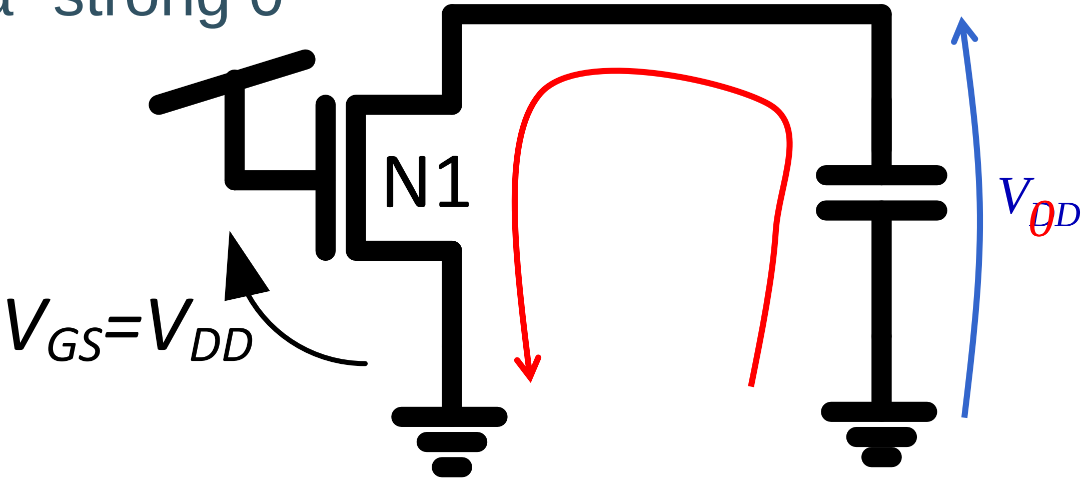
  - The number of transistors required to implement an ***N-input*** logic gate is ***2N***.

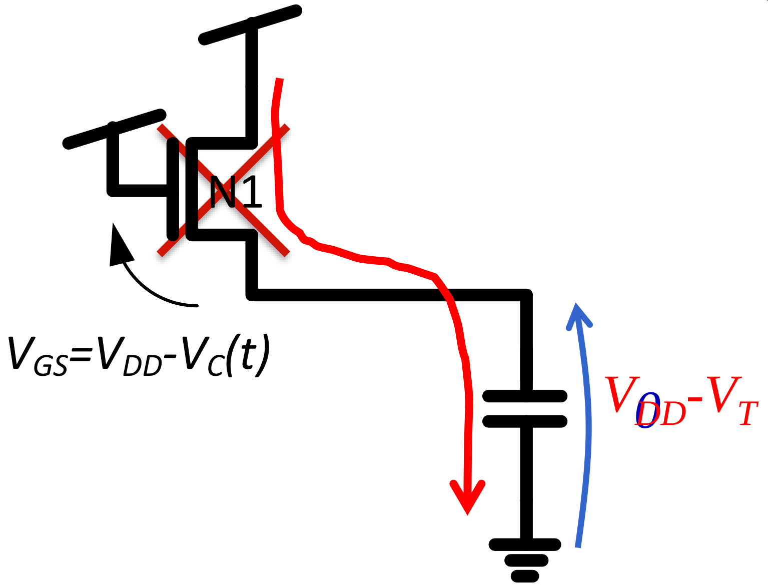
Last Week

- Driving a load
- CMOS Concept – NAND, NOR

How to choose nMOS/pMOS?

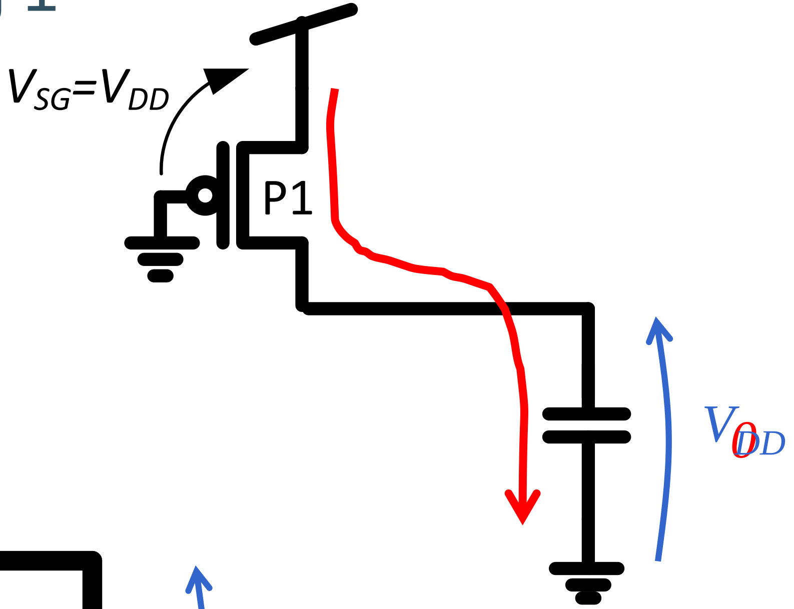
- nMOS devices pass a "strong 0"
- but a "weak 1"



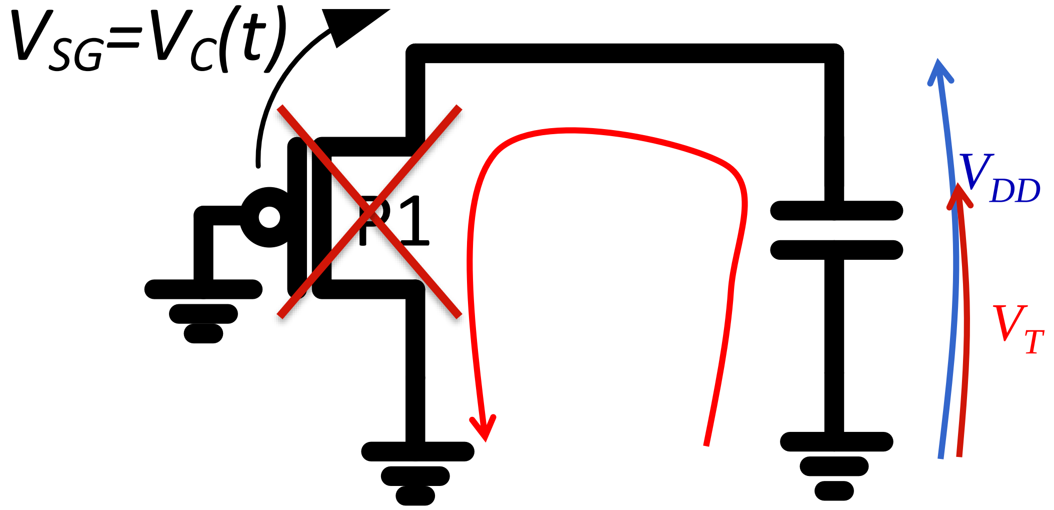


How to choose nMOS/pMOS?

- pMOS devices pass a "strong 1"



- but a "weak 0"



How to choose nMOS/pMOS?

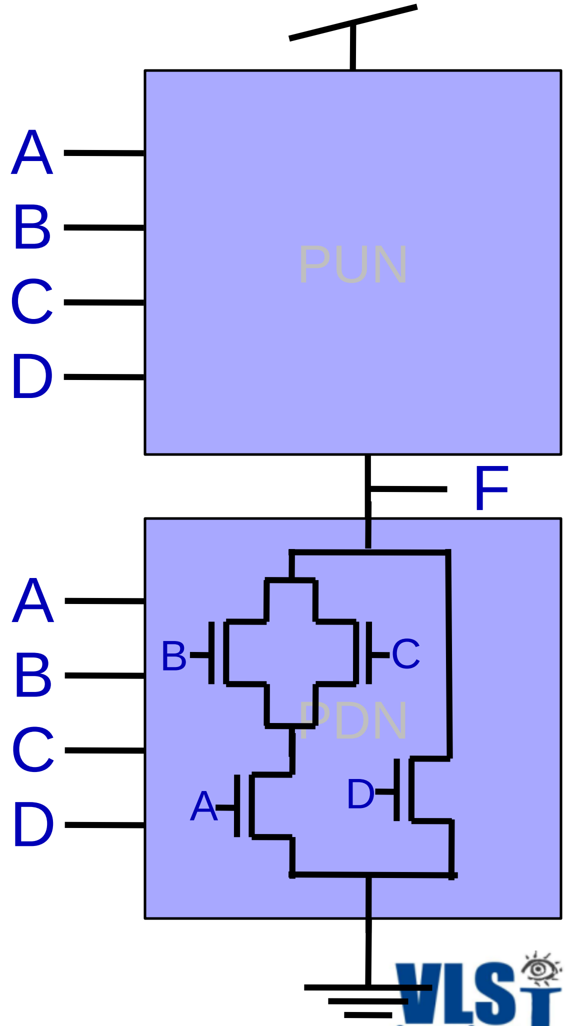
- Therefore, we will always:
- Implement the PDN with nMOS devices.
  - These will create *inverting* functions.
- Implement the PUN with pMOS devices.
  - These will create *non-inverting* functions.

Synthesis of a complex gate

- Now let's try to implement a random ***Boolean Function*** using ***CMOS***:

$$F = \overline{D + A\cdot(B+C)}$$

  - » Our function is *inverting* with *uncomplemented inputs*, so we can immediately derive the ***PDN*** from the given function.
  - » ***B*** and ***C*** are *parallel*
  - » ***A*** is in *series* with the ***B+C*** network
  - » ***D*** is *parallel* to the ***A(B+C)*** network.



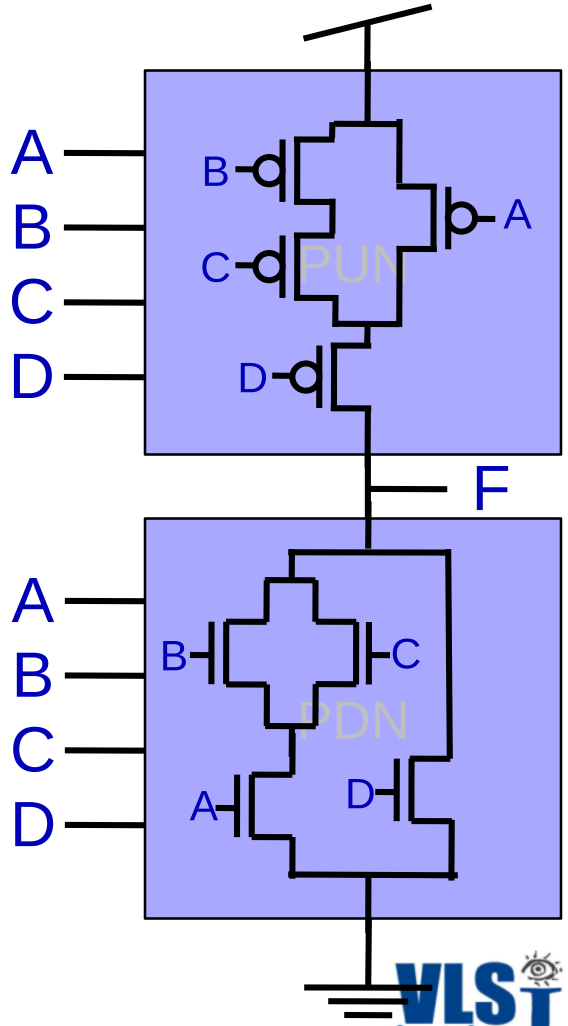
Synthesis of a complex gate

- Next we'll use ***DeMorgan*** to derive the ***PUN***:

$$F = \overline{D + A\cdot(B+C)} = \overline{D}\cdot\overline{A\cdot(B+C)} =$$
$$= \overline{D}\cdot\left(\overline{A}+\overline{B+C}\right) = \overline{D}\cdot\left(\overline{A}+\overline{B}\cdot\overline{C}\right)$$

  - » Our ***PUN*** function is *non-inverting* with complemented inputs.
  - » ***B*** and ***C*** are in *series*
  - » ***A*** is *parallel* with the ***BC*** network
  - » ***D*** is in *serial* with the ***A+(BC)*** network.

- We could have also derived the ***PUN*** by marking subsets of the ***PDN*** and transforming them from *parallel* to *serial* and vice versa!



Another Example

6.3

So we know about the topology, but to optimize the performance of the gate, let's take a look at

TRANSISTOR SIZING

Device Sizing – β - Reminder

- We previously developed an optimum sizing for a CMOS inverter based on the ***β*** ratio.
- We first marked the ratio between the PUN and PDN size as ***β*** and then expressed the load capacitance:

$$\boxed{\beta \triangleq \frac{(W/L)_p}{(W/L)_n}} \qquad \boxed{C_{load} = (1+\beta)\left(C_{dn1} + C_{gn2}\right) + C_{wire}}$$

- We then expressed the delay and optimized it as a function of the resistance ratio between the PUN and PDN:

$$t_{pd} = \frac{0.69 C_{load}}{2}\left(R_{eqn} + R_{eqp}\right) \qquad \frac{dt_{pd}}{d\beta} = 0 \qquad \beta_{opt} = \sqrt{\frac{R_{eqp}}{R_{eqn}}\left(1 + \frac{C_{wire}}{C_{dn1} + C_{gn2}}\right)} \approx \sqrt{\frac{R_{eqp}}{R_{eqn}}}$$

$$\boxed{\beta_{opt} \approx 2}$$

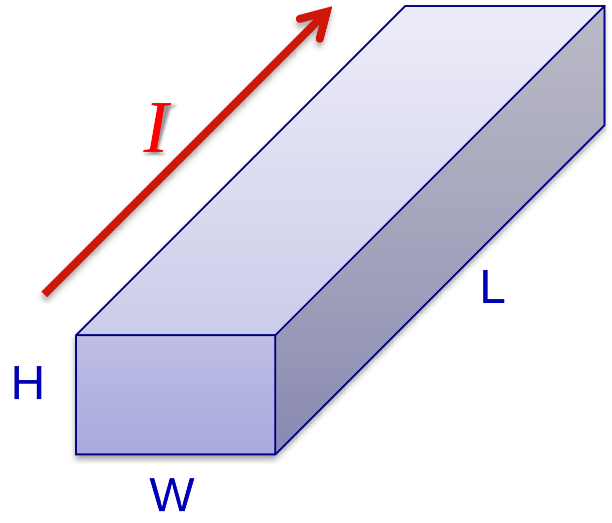
Transistor Sizing Methodology

- We will not prove it now, but it can be shown that the fastest (unloaded) CMOS gate is the optimal inverter.
- Therefore, our methodology for sizing an arbitrary CMOS gate is to try and make it have *the same output resistance* as the optimal inverter.
- To do this, we should look at how transistor *sizing* effects *resistance*.

Transistor Resistance

- We can intuitively look at a transistor’s channel as a resistor.
- In physics we learned that resistance is given by:

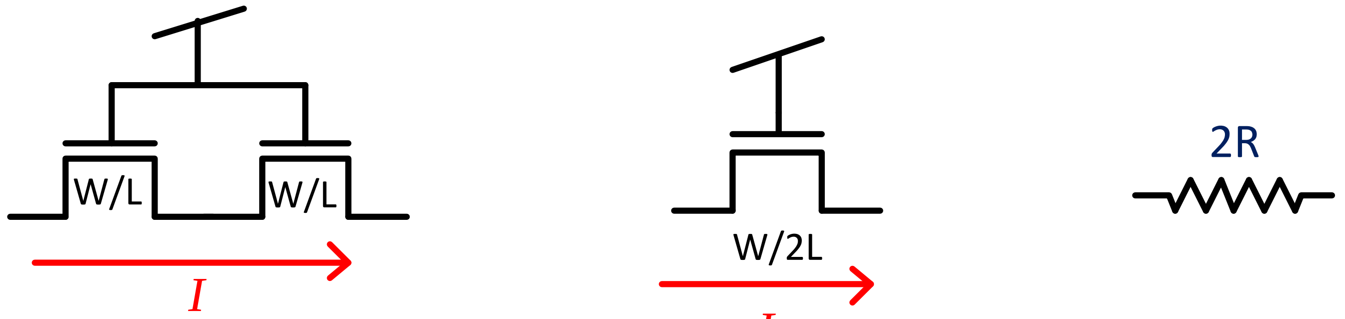
$$R = \rho \frac{L}{A} = \rho \frac{L}{W \cdot H}$$



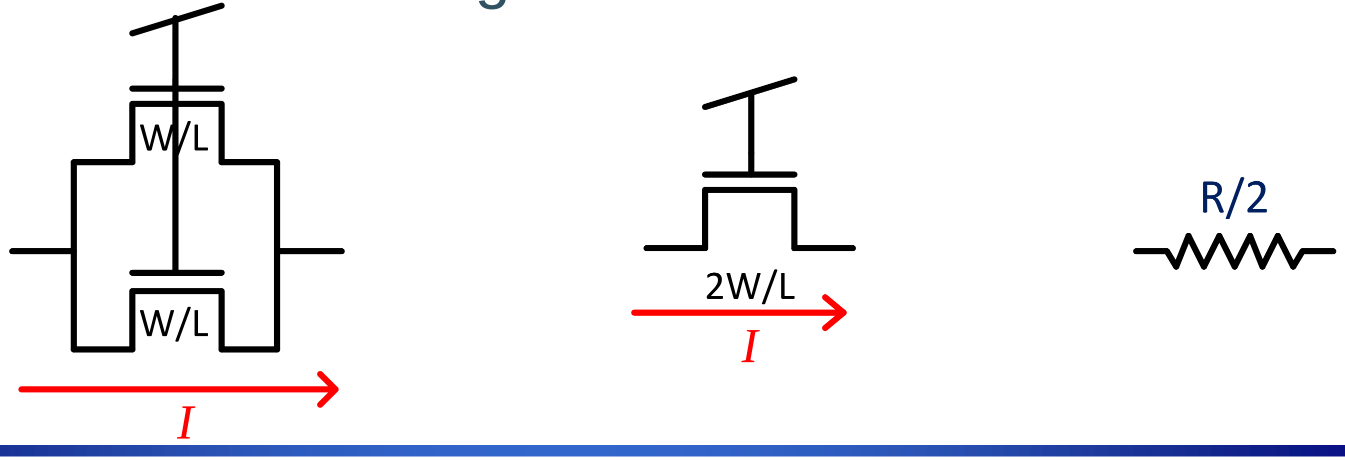
- A transistor’s channel is similar with a constant *H=channel depth*
- So *widening* a transistor will reduce the resistance, whereas a longer channel will increase the resistance.

Transistor Resistance

- Another look at transistor connections, shows that a series connection of constant width transistors is equivalent to increasing the length.



- A parallel connection of constant length transistors is equivalent to increasing the width.



Transistor Resistance

- This also can be shown according to our current model:

$$R_{eq} \approx \frac{3}{4}\frac{V_{DD}}{I_{DSAT}\left(1+\lambda V\right)}\left(1-\frac{7}{9}\lambda V_{DD}\right) \propto \frac{1}{\left(W/L\right)_n}$$

- So to get equivalent resistance of a series connection:

$$R_{series} = R_{eq1} + R_{eq2} + \ldots = const\left(\frac{1}{\left(W/L\right)_1} + \frac{1}{\left(W/L\right)_2} + \ldots\right) = \frac{const}{\left(W/L\right)_{eq}}$$

$$\frac{1}{\left(W/L\right)_{eq}} = \frac{1}{\left(W/L\right)_1} + \frac{1}{\left(W/L\right)_2} + \ldots$$

- And for a parallel connection:

$$\left(W/L\right)_{eq} = \left(W/L\right)_1 + \left(W/L\right)_2 + \ldots$$

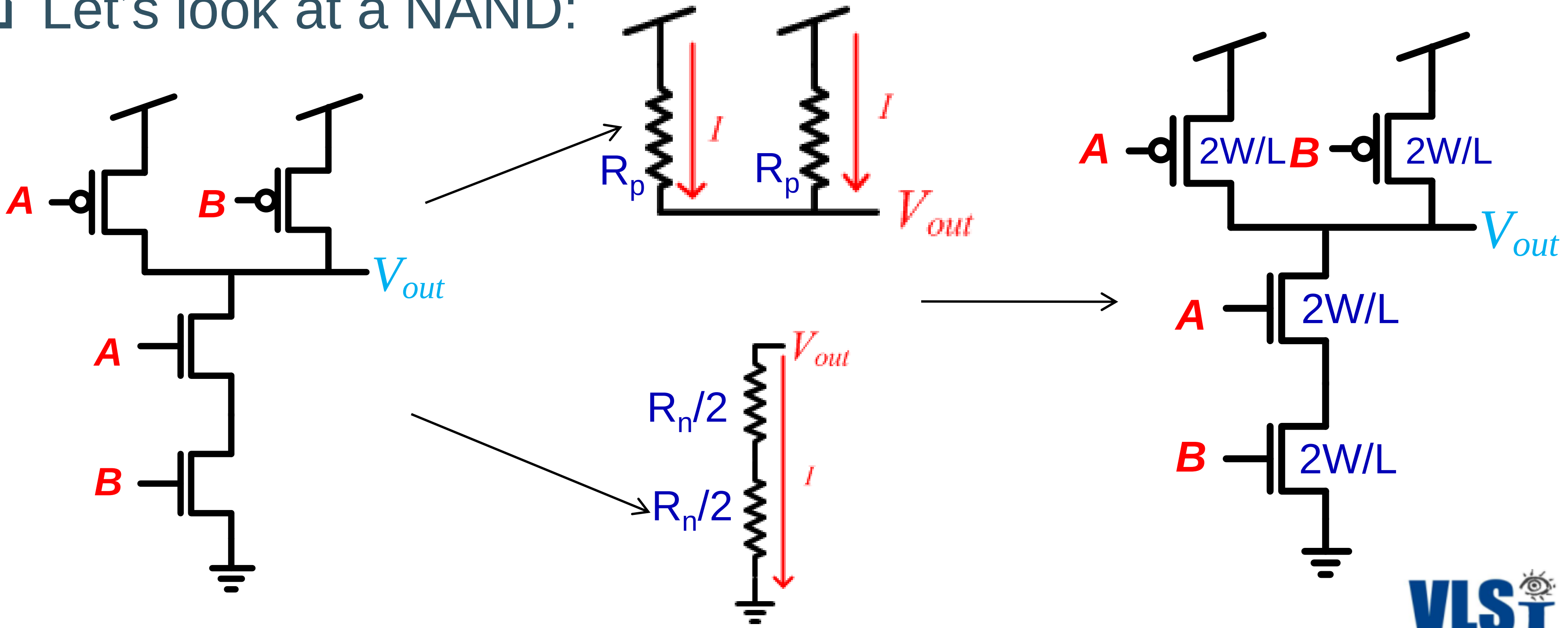
- For example, take 2 transistors with ***W/L=4***:

$$\left(W/L\right)_{parallel} = 4+4=8 \qquad R_{eq,par} = {}^{1}\!/_{8}\, R_{\min}$$

$$\left(W/L\right)_{series} = \frac{1}{1/4+1/4} = 2 \qquad R_{eq,ser} = {}^{2}\!/_{4}\, R_{\min}$$

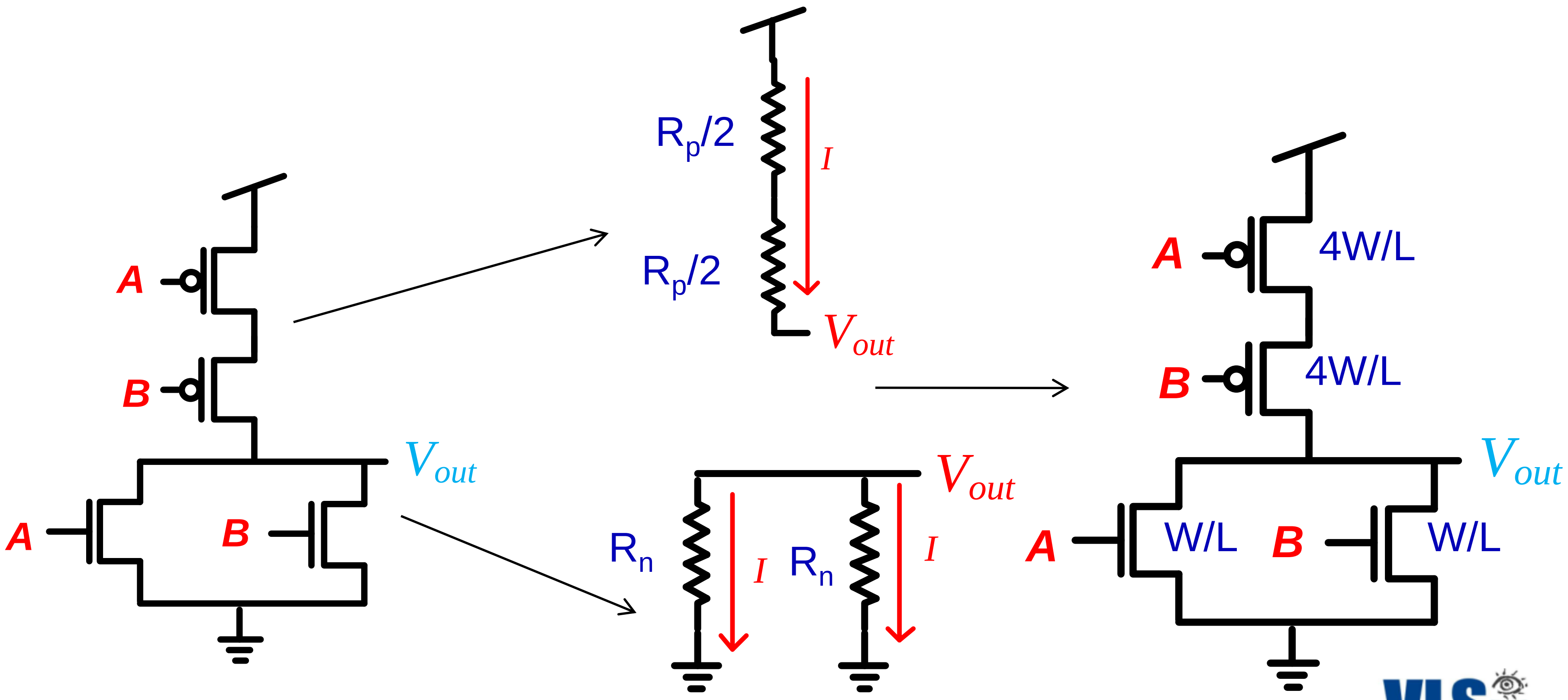
Transistor Sizing

- We'd like to now apply this to get the equivalent resistance of a gate to be similar to an optimum inverter.
- Since a complex gate has several paths, we will always take the *worst case* path for the PUN and PDN.
- Let's look at a NAND:



Transistor Sizing

- What about a NOR?



Conclusions

- A short conclusion:
  - *Minimizing the area* of a transistor (***W***x***L***) is essential, as it affects:
    - Input Capacitance
    - Output Capacitance
    - Silicon area (cost)
    - Others
  - If we assume ***W***$W_{min}=L_{min}$ and calculate the areas of the ***NAND*** and ***NOR*** above, we will find:

$$A_{NAND}=2\left(W_{eqp}+W_{eqn}\right)L_{\min}=2\left(2W_{\min}+2W_{\min}\right)L_{\min}=8L_{\min}{}^{2}$$

$$A_{NOR}=2\left(W_{eqp}+W_{eqn}\right)L_{\min}=2\left(4W_{\min}+W_{\min}\right)=10L_{\min}{}^{2}$$

  - So a ***NAND*** is much more efficient as a ***CMOS Logic Gate*** than a ***NOR***!

Transistor Sizing

- Another conclusion:
  - » As we saw, each additional input requires *2 additional transistors*, an ***nMOS*** and a ***pMOS***.
  - » This drastically increases both the *chip area* and the *capacitances* of a gate, increasing the ***Propagation Delay***.
  - » Using the *transistor sizing* technique, we can try to preserve the gate's *performance* at the expense of *size*, but this only works up to a limit.
  - » A practical limit has been found to be a *maximal **Fan In** of **4***.
  - » If a more complex function needs to be implemented, it should be done by *cascading multiple stages of logic gates*.

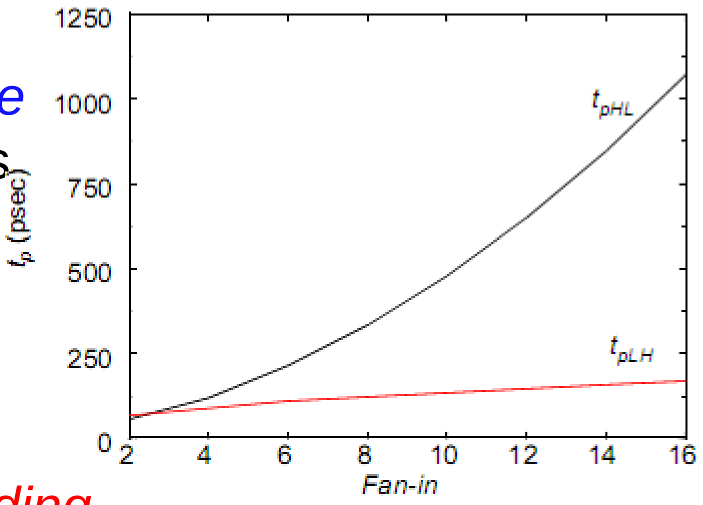


Figure 6.13 Propagation delay of CMOS NAND gate as a function of fan-in. A fan-out of one inverter is assumed, and all pull-down transistors are minimal size.

6.4

Okay, so we see that CMOS has a problem with high fan-in. So here are a few concepts in

DEALING WITH HIGH FAN-IN

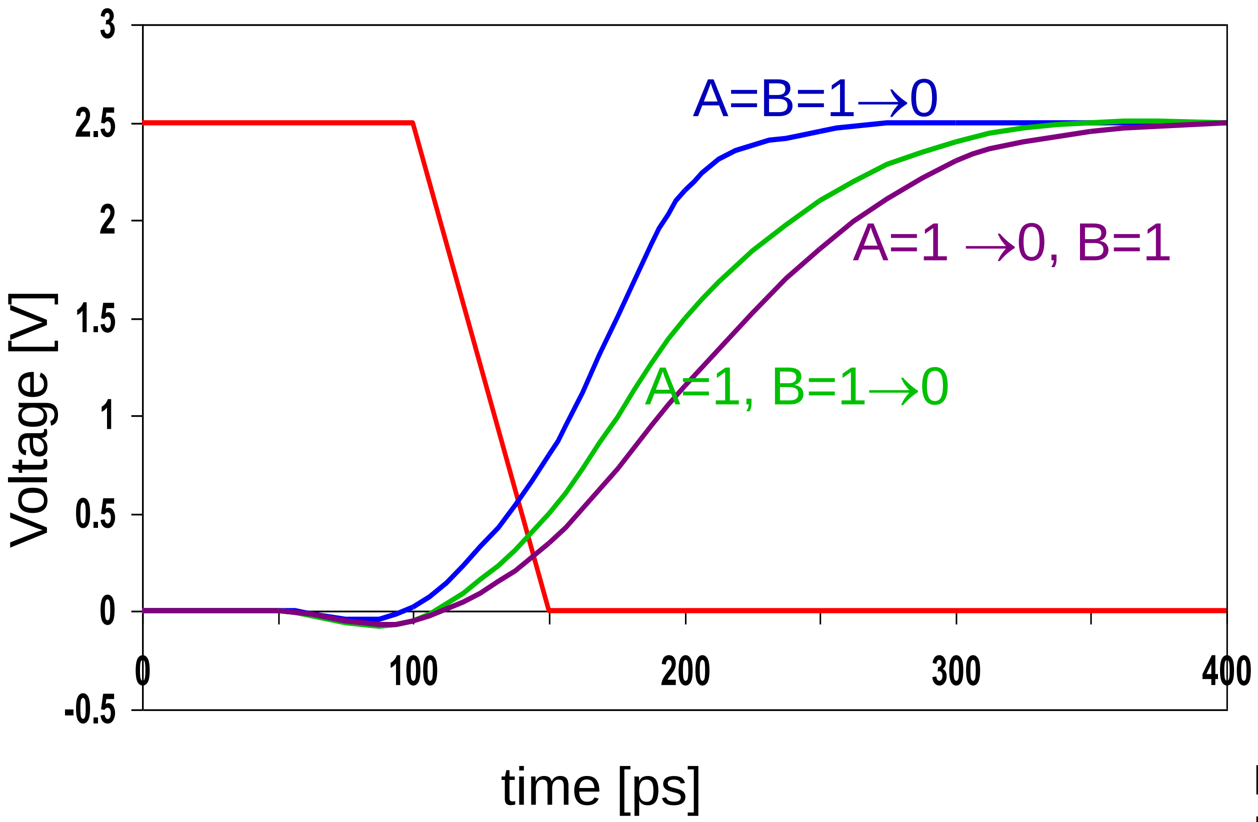
Input Pattern Effects on Static Properties

- How do we draw the VTC of a NAND gate?

Input Pattern Effects on Delay

- And what about the tpd of a NAND gate?

Delay Dependence on Input Patterns



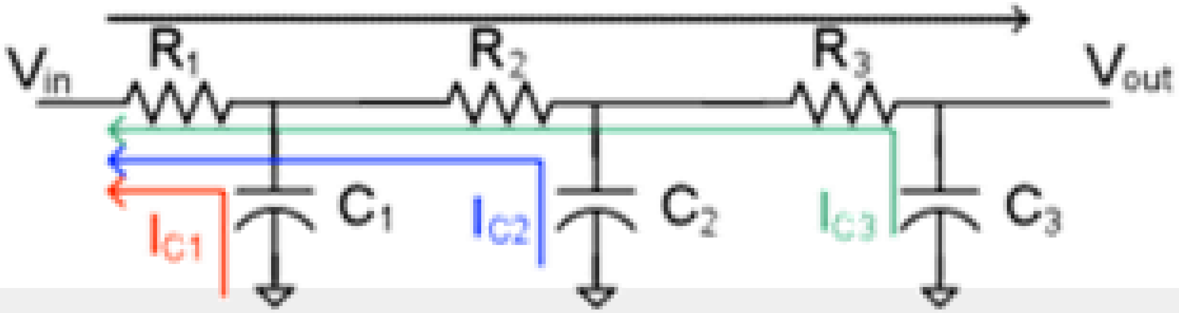
Input Data Pattern	Delay (psec)
A=B=0→1	67
A=1, B=0→1	64
A= 0→1, B=1	61
A=B=1→0	45
A=1, B=1→0	80
A= 1→0, B=1	81

NMOS = 0.5μm/0.25 μm
PMOS = 0.75μm/0.25 μm
C_L = 100 fF

Delay Dependence on Input Patterns

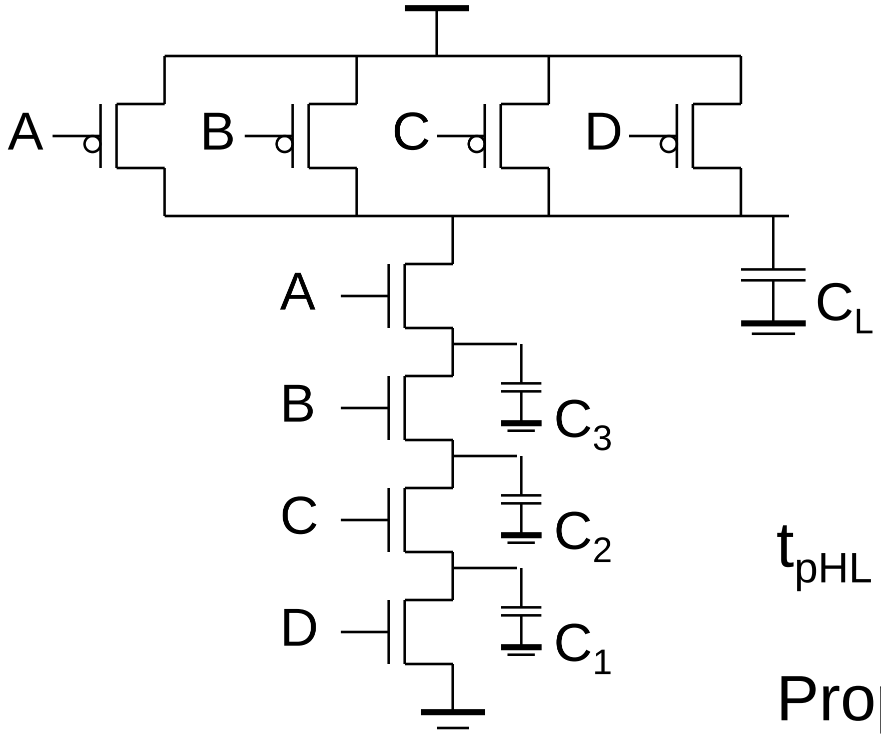
Input Data Pattern	Delay (psec)
A=B=0→1	67
A=1, B=0→1	64
A= 0→1, B=1	61
A=B=1→0	45
A=1, B=1→0	80
A= 1→0, B=1	81

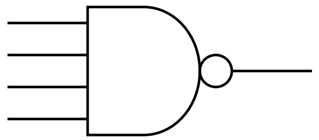
Delay Estimation using the Elmore Delay



$$\tau_{elmore} = R_1C_1 + (R_1 + R_2)C_2 + (R_1 + R_2 + R_3)C_3$$

Fan-In Considerations





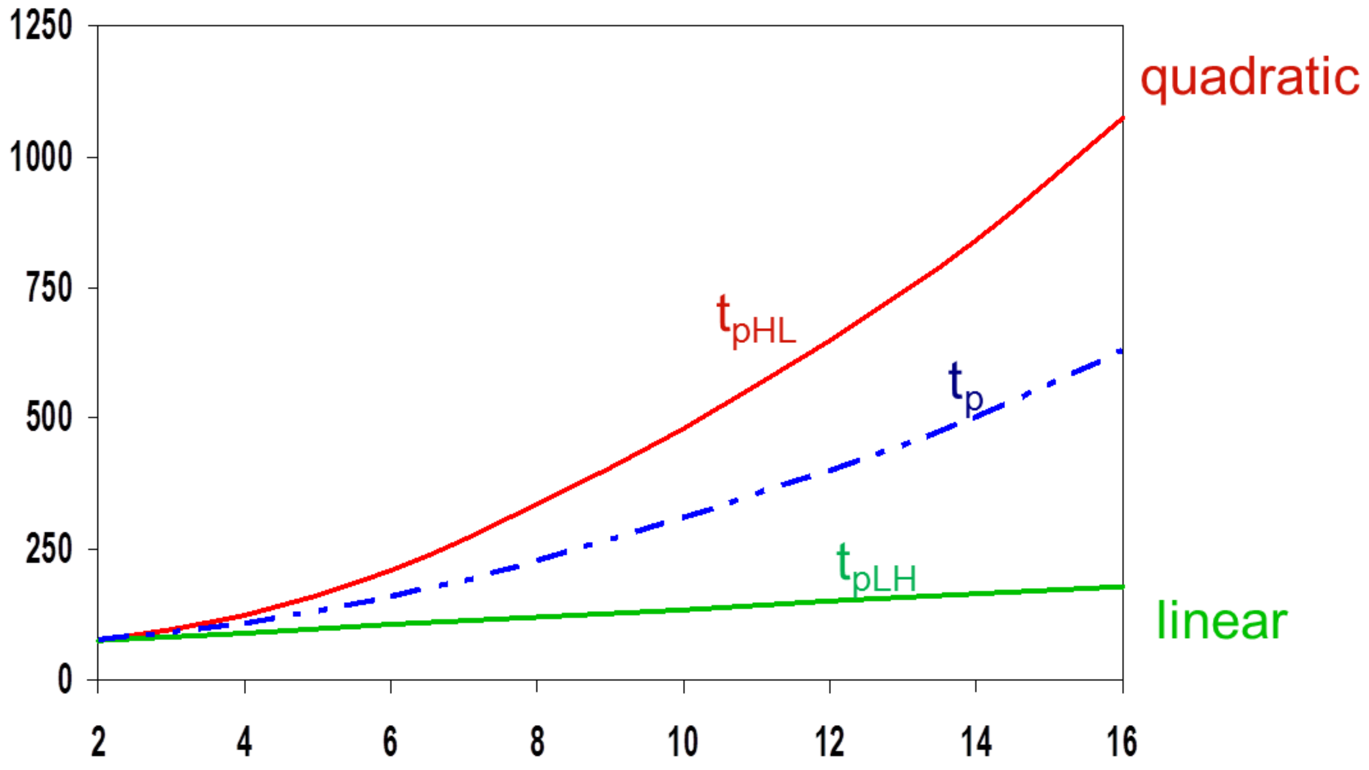
Distributed RC model
(Elmore delay)

$$t_{pHL} = 0.69\ R_{eqn}(C_1+2C_2+3C_3+4C_L)$$

Propagation delay deteriorates rapidly as a function of fan-in – quadratically in the worst case.

Fan-In Considerations

- CMOS Gates with a Fan In greater than 4 should be avoided!

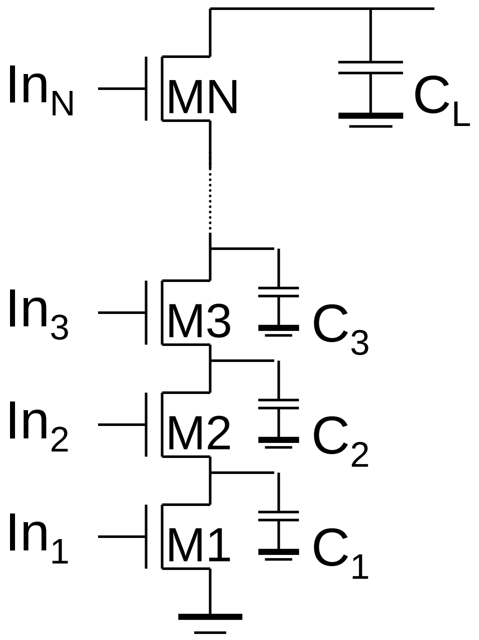


Dealing with Fan In: Transistor Sizing

- Make the transistors bigger, their resistance goes down, and the time constant decreases.
- BUT, the capacitance gets bigger presenting a bigger load to previous gates.
- (We will come back to this later…)

Dealing with Fan In: Progressive Sizing

- Progressive sizing

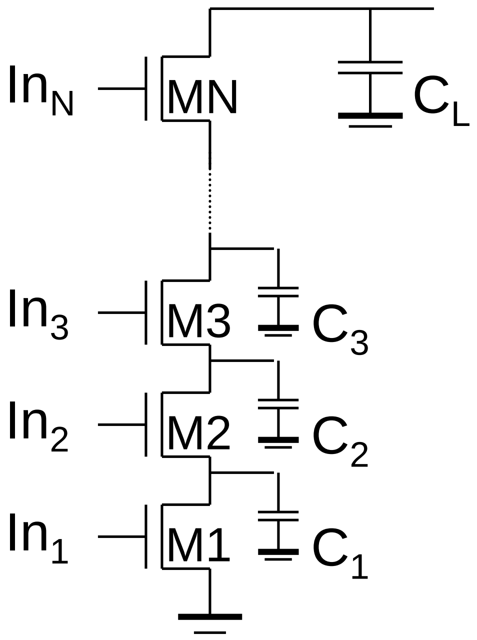


Looking at the Elmore Delay, M1 is on the path of all Capacitors, while MN is only on the path of CL.

Why not make M1 have less resistance than MN…

Dealing with Fan In: Progressive Sizing

- Progressive sizing



M1 > M2 > M3 > … > MN
(the FET closest to the output is the smallest)

Can reduce delay by more than 20%; decreasing gains as technology shrinks

But it can have a large area overhead in layout…

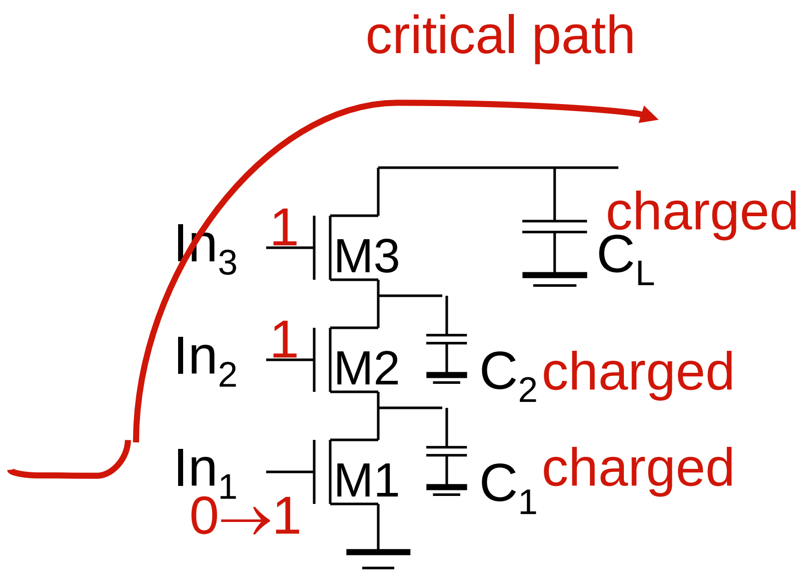
Fan In Considerations: Input Ordering

- Not all logic paths are equal.
- The frequency is measured according to the slowest path, or better known as the *critical path*.

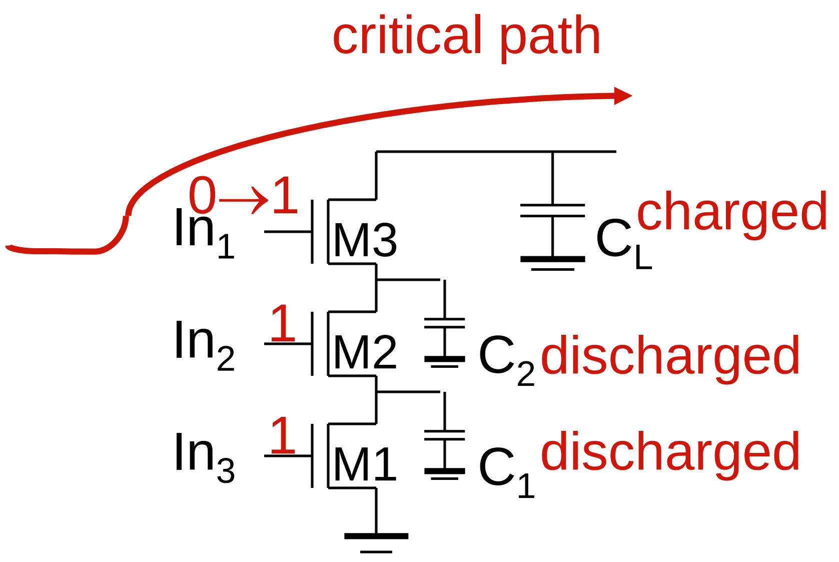
- So we should connect the critical path to the faster inputs.

Dealing with Fan In: Input Reordering

- Transistor ordering



delay determined by time to discharge C_L, C_1 and C_2



delay determined by time to discharge C_L

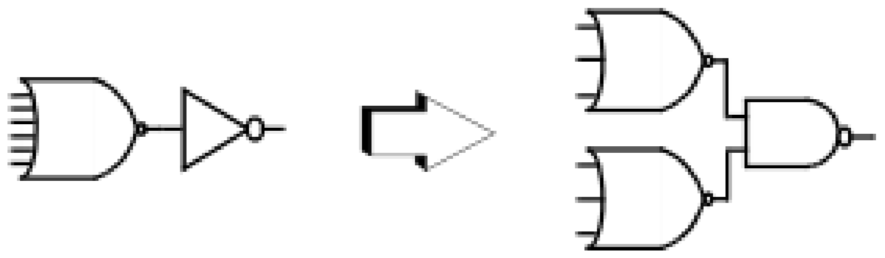
Dealing with Fan In: Input Reordering

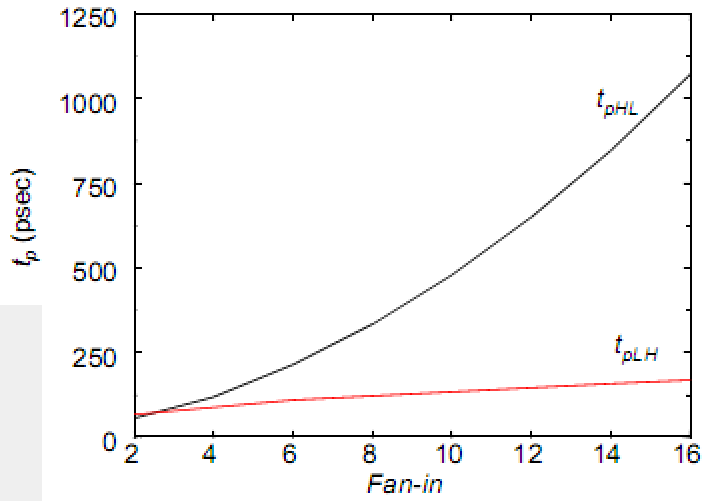
Fan In Considerations: Logic Restructure

- Sometimes we just have to have large Fan-Ins.
- For Example: A Decoder

Dealing with Fan In: Logic Restructuring

- We can usually restructure our logic (Boolean manipulations) to decrease the Fan-In of each gate by trading off number of stages with Fan In of each stage.





- Next lecture, we will learn how to optimize this consideration.

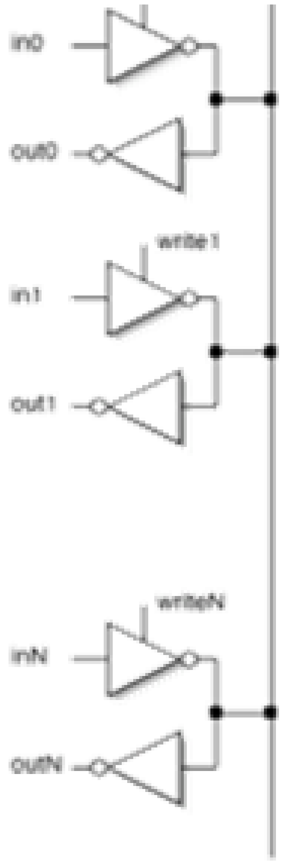
6.5

Okay, we now know how to synthesize any gate with the CMOS concept. Or do we? Here are some

NON-STANDARD GATES

Tri-State Buffers

- The outputs of two or more CMOS gates cannot be connected to each other.
- But often, we need to drive a bus.

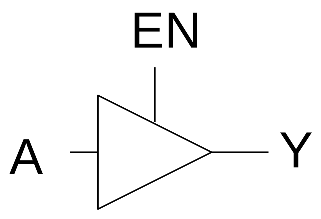


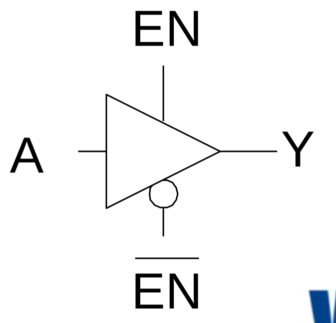
- Therefore, we need to implement a *tri-state* buffer.

Tri State Buffers

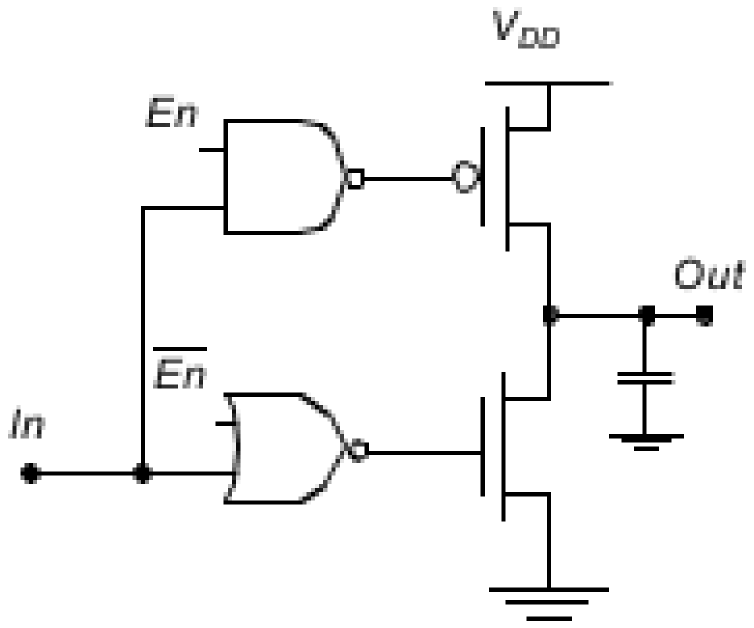
- Tri-state buffers are used when multiple circuits all connect to a common wire. Only one circuit at a time is allowed to drive the bus. All others can "disconnect" their outputs, but can "listen".
- Tri-state buffers enable "bidirectional" connections.

EN	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1



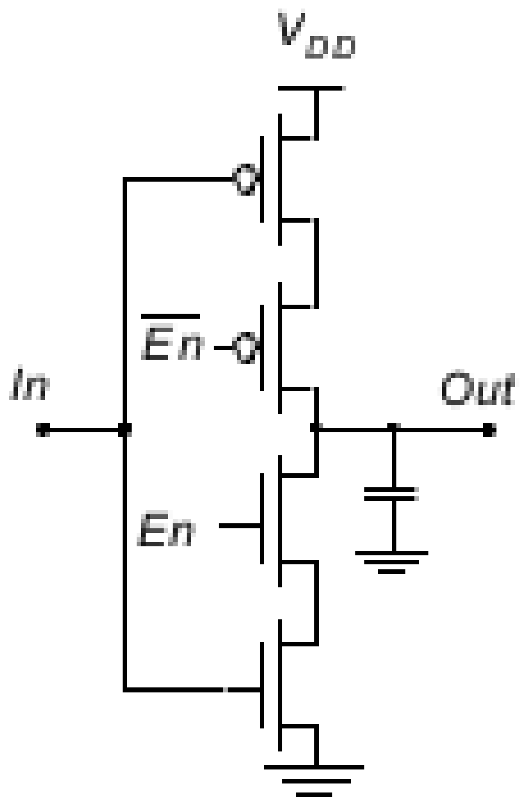


Standard CMOS Implementation



EN	In	PUN	PDN	Y
0	0			
0	1			
1	0			
1	1			

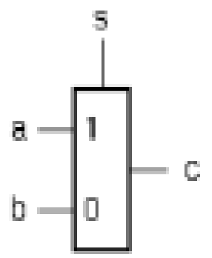
Reduced Transistor Implementation



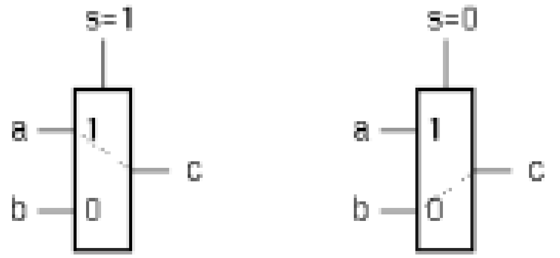
EN	In	PUN	PDN	Y
0	0			
0	1			
1	0			
1	1			

Tri-State Multiplexor

Multiplexor

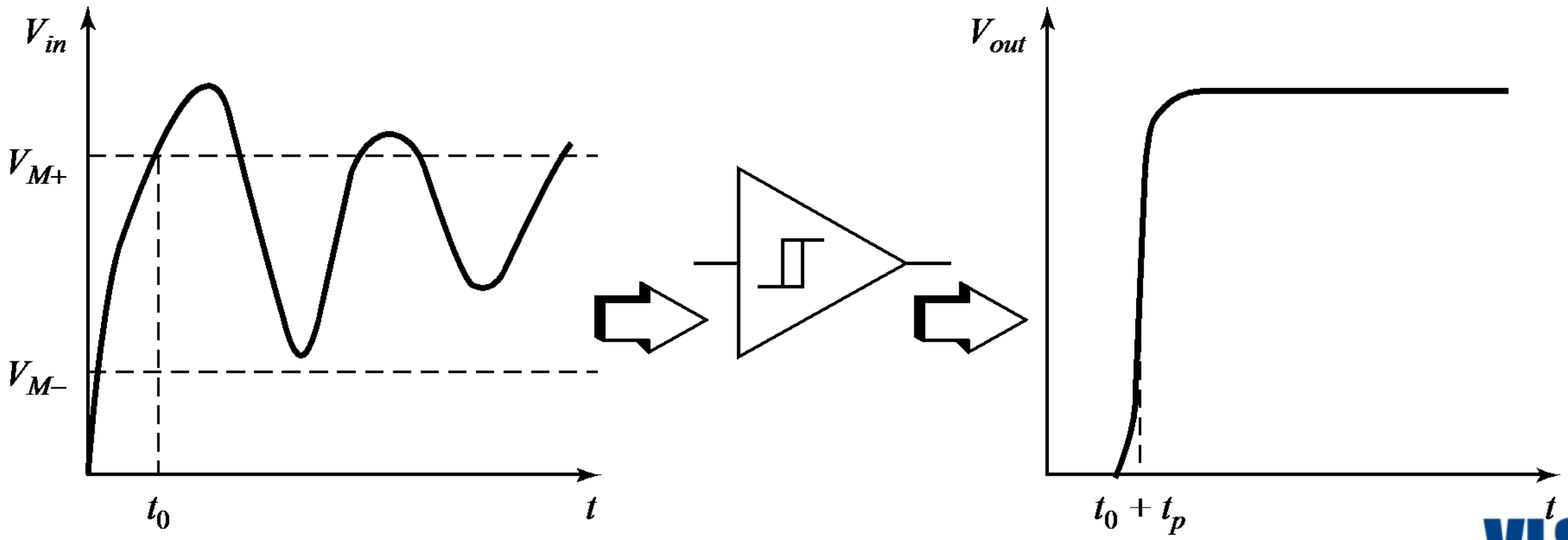


If s=1 then c=a else



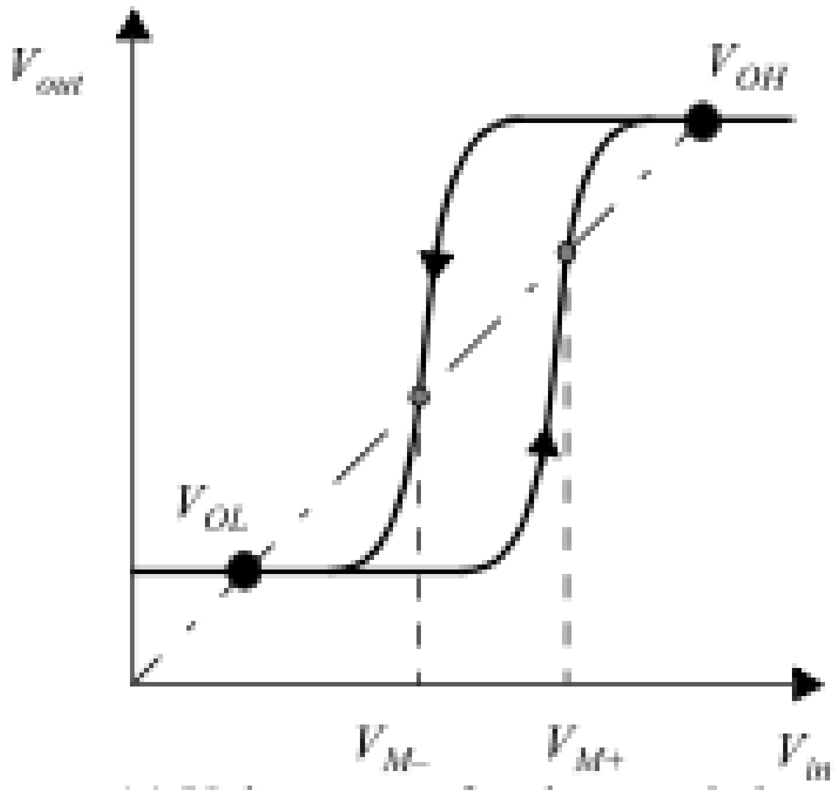
Shmitt Trigger

- Sometimes, we have a very noisy or slow varying signal, and would like to "clean it up".
- This is often the case in inter-chip interfaces, where there are many noise sources.

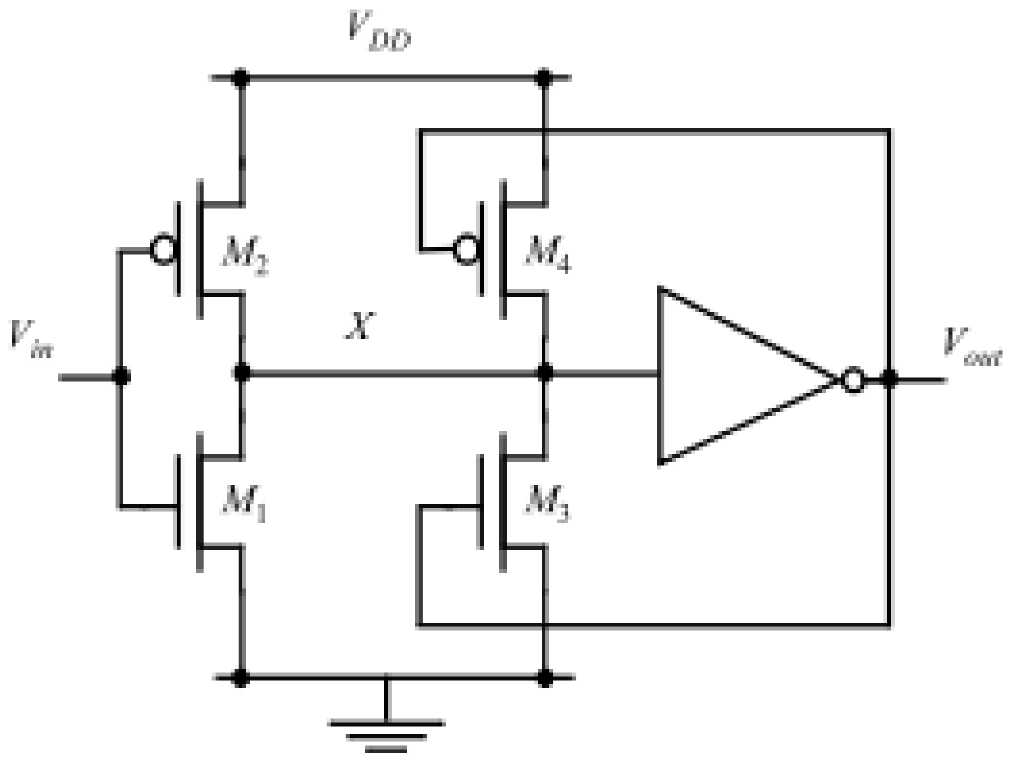


Schmitt Triggers

- A Schmitt Trigger achieves our goal by using a *hysteresis* in its VTC:

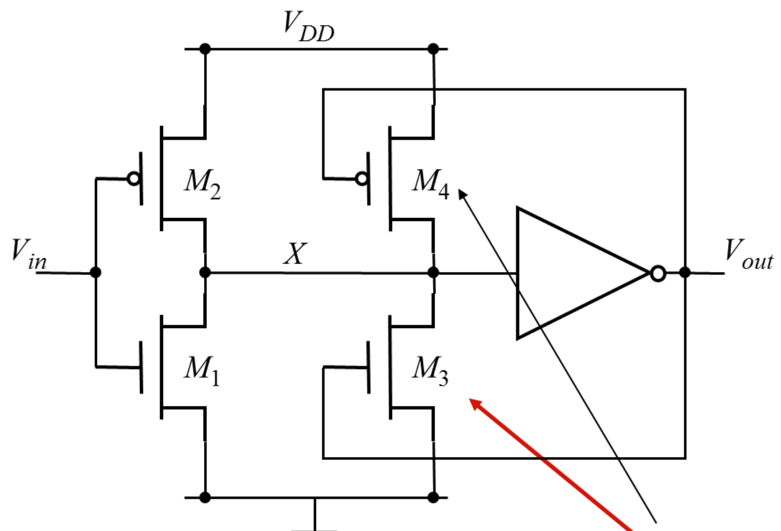


Schmitt Trigger – CMOS Implementation

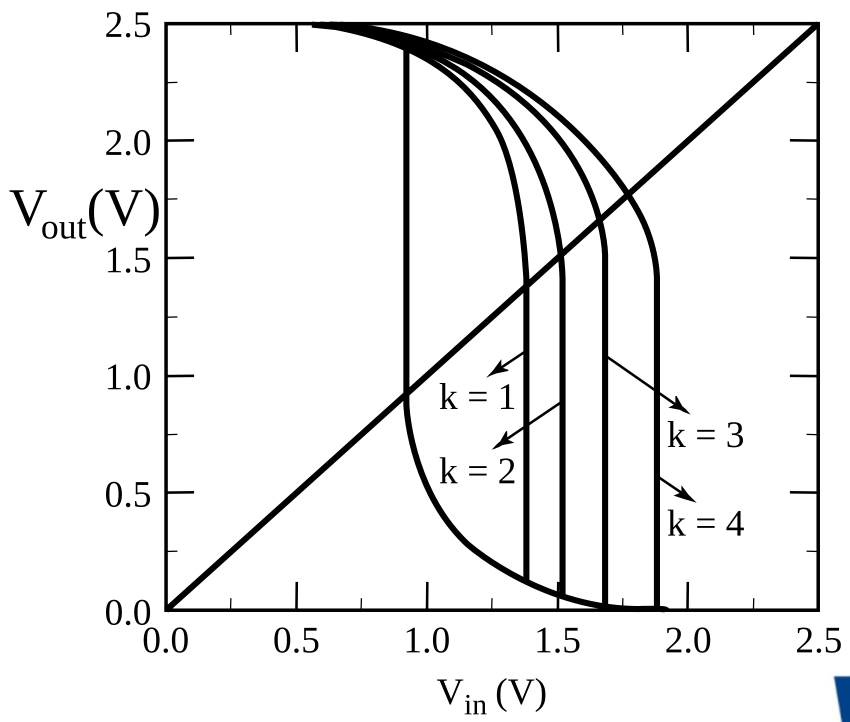


Schmitt Triggers

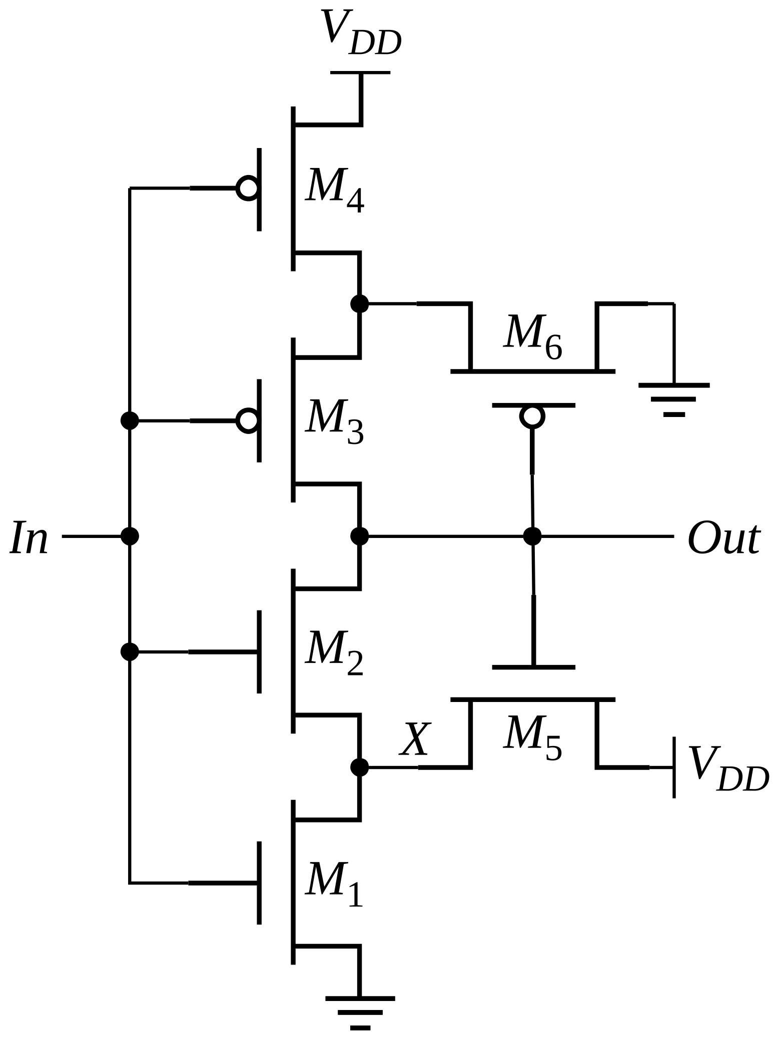
- Increasing kn/kp ratio decreases the logical switching threshold
- If V_{in}=0 then V_{out} (connected to M_4) is also zero
- So effectively the input is connected to M_2 and M_4 in parallel
- This increases kp and the switching threshold. If V_{in}=0 the situation is reversed and kn increases reducing the switching threshold



These transistors resist the change in the X signal Move switching threshold of the first inverter



Another Schmitt Trigger Implementation


V_{DD}
M_4
M_6
M_3
In
Out
M_2
X
M_5
V_{DD}
M_1