Last Lectures

- The CMOS Inverter
- CMOS Capacitance
- Driving a Load
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 CMOS Basic Concept
- 6.1.1 Classification
- 6.1.2 CMOS Structure
- 6.1.3 Constructing the PUN/PDN

6.2 Circuit Implementation
- 6.2.1 Universality
- 6.2.2 Series and Parallel
- 6.2.3 Complementary Networks
- 6.2.4 Synthesis of a Complex Gate

6.3 Transistor Sizing

6.4 Dealing with High Fan-In

6.5 Non-Standard Gates
6.1 CMOS Basic Concept

Better start with the

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

- *Sequential* circuits will be discussed in a later lecture.
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 more sensitive to noise.

- This lecture will discuss **Static or Complementary CMOS**, obviously belonging to the Static classification.
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.
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*.
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.
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**.
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:

\[
A \cdot B = \overline{A} + \overline{B}
\]

\[
A + B = \overline{A} \cdot \overline{B}
\]
Interestingly, we have a perfect solution to produce these complementary logic expressions.

Remember our good old friend DeMorgan.

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 \textit{top} and who on the \textit{bottom}…
  - Using our engineering intuition, we can look at the \textit{CMOS inverter} and realize that the \textit{pMOS} is probably in the \textit{Pull Up}…
  - But Why???

- The answer lies in a very important characteristic of the \textit{MOSFET} transistor.
  - A \textit{pMOS} is a device that generates \textit{“Strong Ones”}.
  - An \textit{nMOS} is a device that generates \textit{“Strong Zeros”}. 
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 \textit{nMOS}...

» Now, $V_{GS} = V_{DD} - V_{out}$, in other words, the state of the transistor \textit{depends on the output voltage}.

» Once the output reaches $V_{DD} - V_{Tn}$, the transistor \textit{switches off}, and the output capacitance \textit{stops charging}.

We’ve provided a “\textit{Weak One}”. 

Obviously, we’ll prefer \textit{pMOS} transistors in our \textit{Pull Up Networks}!
The same can be shown for a Pull Down Network.

- Here, an \textit{nMOS} provides a "Strong Zero", as \( V_{GS} = V_{DD} \) independent of the output voltage.

- A \textit{pMOS}, on the other hand, provides a "Weak Zero", only charging the output to \( V_{DD} - |V_{Tp}| \).

We will use \textit{nMOS} transistors in our \textit{Pull Down Networks}!
To summarize, we saw that:

- To get a "Strong One", we need \textit{pMOS} transistors in the \textit{PUN}.
- To get a "Strong Zero", we need \textit{nMOS} transistors in the \textit{PDN}.

This is the basic concept in constructing a \textit{Static CMOS} logic gate.

Again, we can look at the \textit{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.
Okay, we understand the concept. It’s time for

**CMOS DIGITAL LOGIC IMPLEMENTATION**
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**.
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.
Let’s see what happens with \textit{nMOS} transistors:

- **Series – AND – connection:**
  
  ![Series Combination Diagram]

- **Parallel – OR – connection:**
  
  ![Parallel Combination Diagram]
We need a low input to control pMOS transistors, so we get an inverting function:

» pMOS in Series:

```plaintext
A \rightarrow \overline{A} & \overline{B}
```

» Parallel pMOS connection:

```plaintext
\overline{A \cdot B} = \overline{A} + \overline{B}
```

With pMOS transistors, we’ve created NOR and NAND functions!
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?...
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:

\[
\begin{align*}
A \cdot B_{\text{PDN}} &= \overline{A + B}_{\text{PUN}} \\
\end{align*}
\]

» 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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>PUN</th>
<th>PDN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table above illustrates the behavior of a complementary network. The columns represent the input values for A and B, and the output columns show the corresponding PUN, PDN, and OUT states. Each row represents a different combination of A and B, with the output states for each combination.
Now, we’ll try the same method to construct a 2-input NOR:

\[ A + B = A \cdot B \]

» 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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>PUN</th>
<th>PDN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.
Now let’s try to implement a random Boolean Function using CMOS:

\[ F = D + A \cdot (B + C) \]

- Our function is invertling 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.
Next we’ll use \textit{DeMorgan} to derive the \textit{PUN}:

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

» Our \textit{PUN} function is \textit{non-inverting} with complemented inputs.

» \textit{B} and \textit{C} are in \textit{series}

» \textit{A} is \textit{parallel} with the \textit{BC} network

» \textit{D} is in \textit{serial} with the \textit{A+(BC)} network.

We could have also derived the \textit{PUN} by marking subsets of the \textit{PDN} and transforming them from \textit{parallel} to \textit{serial} and vice versa!
So we know about the topology, but to optimize the performance of the gate, let’s take a look at **TRANSISTOR SIZING**
We previously developed an optimum sizing for a CMOS inverter based on the $\beta$ ratio.

We first marked the ratio between the PUN and PDN size as $\beta$ and then expressed the load capacitance:

$$\beta \equiv \frac{(W/L)_p}{(W/L)_n}$$

$$C_{load} = (1 + \beta)(C_{dn1} + C_{gn2}) + C_{wire}$$

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

$$t_{pd} = \frac{0.69C_{load}}{2}(R_{eqn} + R_{eqp})$$

$$\frac{dt_{pd}}{d\beta} = 0$$

$$\beta_{opt} = \sqrt{\frac{R_{eqp}}{R_{eqn}} \left(1 + \frac{C_{wire}}{C_{dn1} + C_{gn2}}\right)} \approx \sqrt{\frac{R_{eqp}}{R_{eqn}}}$$

$$\beta_{opt} \approx 2$$
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.
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 \( H = \text{channel depth} \).

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:

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

- So to get equivalent resistance of a series connection:

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

\[
\frac{1}{(W/L)_{eq}} = \frac{1}{(W/L)_1} + \frac{1}{(W/L)_2} + \ldots
\]

- And for a parallel connection:

\[
(W/L)_{eq} = (W/L)_1 + (W/L)_2 + \ldots
\]

- For example, take 2 transistors with \(W/L=4\):

\[
(W/L)_{parallel} = 4 + 4 = 8 \quad R_{eq,\,par} = \frac{1}{8} R_{\text{min}} \quad (W/L)_{series} = \frac{1}{\frac{1}{4} + \frac{1}{4}} = 2 \quad R_{eq,\,ser} = \frac{2}{4} R_{\text{min}}
\]
- 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:
What about a NOR?

\[ V_{\text{out}} = \frac{I}{R_{\text{n}}} \]

\[ I = \frac{V_{\text{out}}}{R_{\text{p}}/2} + \frac{V_{\text{out}}}{R_{\text{p}}/2} \]

\[ V_{\text{out}} = \frac{V_{\text{out}}}{R_{\text{n}}} \]
A short conclusion:

» *Minimizing the area* of a transistor (*WxL*) is essential, as it affects:
  
  – Input Capacitance
  – Output Capacitance
  – Silicon area (cost)
  – Others

» If we assume \( W_{\text{min}} = L_{\text{min}} \) and calculate the areas of the *NAND* and *NOR* above, we will find:

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

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

» 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.
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**
How do we draw the VTC of a NAND gate?
And what about the tpd of a NAND gate?
**Delay Dependence on Input Patterns**

![Graph showing voltage over time for different input patterns.](image)

<table>
<thead>
<tr>
<th>Input Data Pattern</th>
<th>Delay (psec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=B=0 → 1</td>
<td>67</td>
</tr>
<tr>
<td>A=1, B=0 → 1</td>
<td>64</td>
</tr>
<tr>
<td>A= 0 → 1, B=1</td>
<td>61</td>
</tr>
<tr>
<td>A=B=1 → 0</td>
<td>45</td>
</tr>
<tr>
<td>A=1, B=1 → 0</td>
<td>80</td>
</tr>
<tr>
<td>A= 1 → 0, B=1</td>
<td>81</td>
</tr>
</tbody>
</table>

**Device Specifications**

- **NMOS**: 0.5μm/0.25 μm
- **PMOS**: 0.75μm/0.25 μm
- **C<sub>L</sub>**: 100 fF
### Delay Dependence on Input Patterns

<table>
<thead>
<tr>
<th>Input Data Pattern</th>
<th>Delay (psec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=B=0 → 1</td>
<td>67</td>
</tr>
<tr>
<td>A=1, B=0 → 1</td>
<td>64</td>
</tr>
<tr>
<td>A=0 → 1, B=1</td>
<td>61</td>
</tr>
<tr>
<td>A=B=1 → 0</td>
<td>45</td>
</tr>
<tr>
<td>A=1, B=1 → 0</td>
<td>80</td>
</tr>
<tr>
<td>A=1 → 0, B=1</td>
<td>81</td>
</tr>
</tbody>
</table>
Delay Estimation using the Elmore Delay

\[ \tau_{\text{elmore}} = R_1 C_1 + \left( R_1 + R_2 \right) C_2 + \left( R_1 + R_2 + R_3 \right) C_3 \]
**Fan-In Considerations**

Distributed RC model (Elmore delay)

\[
\text{Propagation delay deteriorates rapidly as a function of fan-in – quadratically in the worst case.}
\]

\[
t_{pHL} = 0.69 \ R_{\text{eqn}}(C_1 + 2C_2 + 3C_3 + 4C_L)
\]
- 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…)
Progressive sizing

Looking at the Elmore Delay, \( M_1 \) is on the path of all Capacitors, while \( MN \) is only on the path of \( C_L \).

Why not make \( M_1 \) have less resistance than \( MN \)...
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

![Diagram](image)

critical path
delay determined by time to discharge $C_L$, $C_1$ and $C_2$

critical path
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.
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.

<table>
<thead>
<tr>
<th>EN</th>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Standard CMOS Implementation

<table>
<thead>
<tr>
<th>EN</th>
<th>In</th>
<th>PUN</th>
<th>PDN</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Reduced Transistor Implementation

![Circuit Diagram]

<table>
<thead>
<tr>
<th>EN</th>
<th>In</th>
<th>PUN</th>
<th>PDN</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 Trigger Circuit Diagram]
Schmitt Triggers

- Increasing \( \frac{k_n}{k_p} \) 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 \( k_p \) and the switching threshold. If \( V_{in} = 0 \) the situation is reversed and \( k_n \) 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

![Schmitt Trigger Circuit Diagram]

Another Schmitt Trigger Implementation

![Schmitt Trigger Circuit Diagram]