CSE 220:  
System Fundamentals I  
Unit 9:  
Digital Logic Design:  
Combinational Building Blocks and Timing
Timing of Circuits

- Electricity does not flow through a circuit instantaneously.
- There are delays as signals travel down wires and through gates and combinational units.
- There is a delay between a change in the output to a gate and the subsequent change on the output.
- On the right we see a timing diagram.
- A transition from LOW to HIGH is called the rising edge.
Timing of Circuits

• Likewise, a transition from HIGH to LOW is a **falling edge**
• Delays are measured from the **50% point** of the input signal (A) to the 50% point of the output signal (Y)
• Such delays are ordinarily on the order of picoseconds (1 ps = 10^{-12} seconds) to nanoseconds (1 ns = 10^{-9} seconds)
Timing of Circuits

• Gates and combinational logic circuits have two kinds of delay: propagation delay and contamination delay
• Quite simply, the propagation ($t_{pd}$) delay is the maximum time from when an input changes until the output(s) reach their final value
• The contamination delay ($t_{cd}$) is the minimum time from when an input changes until any output starts to change its value

![Diagram of a simple circuit with time delays marked as $t_{pd}$ and $t_{cd}$]
Critical Path Timing

• To determine the propagation delay of a circuit we must find its **critical path**, which is the path between the input values and the output value which has the longest delay

• To do so, assume that each input value was available at time 0

• For each gate which the input values propagate through, calculate the time when the output will be available

• For each internal gate, take the input value with the largest delay and add the gate delay to produce the output time

• Continue for all gates, until the output is reached
Critical Path Timing

• The critical path delay time (i.e., a circuit’s propagation delay) is the length of time for the output value to be generated
• The critical path is the set of gates which dictate this longest time
• If there is a tie between inputs for the longest time, both inputs are required in the critical path
• Assuming all gates in this figure have the same propagation delays, then the blue path is the critical path
Critical Path Timing

Critical Path

Short Path
Critical Path Timing

- Suppose $t_{pd_{AND}}$ and $t_{pd_{OR}}$ are the respective propagation delays of the AND and OR gates in this example.
- Then the circuit’s critical path is $t_{pd} = 2t_{pd_{AND}} + t_{pd_{OR}}$.
Example #1: Critical Paths

- Calculate the critical path delay where the gates have the propagation delays indicated in the table

<table>
<thead>
<tr>
<th>Gate</th>
<th>Propagation Delay (ps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT gate</td>
<td>30</td>
</tr>
<tr>
<td>3-input AND gate</td>
<td>80</td>
</tr>
<tr>
<td>4-input OR gate</td>
<td>90</td>
</tr>
</tbody>
</table>

critical path delay = 30 ps + 80 ps + 90 ps = 200 ps
Example #2: Critical Paths

- Highlight the critical path and calculate the critical path timing where the gates have the propagation delays indicated in the table.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND gate</td>
<td>3</td>
</tr>
<tr>
<td>OR gate</td>
<td>4</td>
</tr>
<tr>
<td>NAND gate</td>
<td>2</td>
</tr>
</tbody>
</table>
Example #2: Critical Paths

- Highlight the critical path and calculate the critical path timing where the gates have the propagation delays indicated in the table.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND gate</td>
<td>3</td>
</tr>
<tr>
<td>OR gate</td>
<td>4</td>
</tr>
<tr>
<td>NAND gate</td>
<td>2</td>
</tr>
</tbody>
</table>
Example #2: Critical Paths

- Highlight the critical path and calculate the critical path timing where the gates have the propagation delays indicated in the table.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND gate</td>
<td>3</td>
</tr>
<tr>
<td>OR gate</td>
<td>4</td>
</tr>
<tr>
<td>NAND gate</td>
<td>2</td>
</tr>
</tbody>
</table>
Example #3: Critical Paths

- Highlight the critical path and calculate the critical path timing where the gates have the propagation delays indicated in the table.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-input AND gate</td>
<td>4</td>
</tr>
<tr>
<td>3-input AND gate</td>
<td>2</td>
</tr>
<tr>
<td>OR gate</td>
<td>3</td>
</tr>
<tr>
<td>NAND gate</td>
<td>5</td>
</tr>
</tbody>
</table>
Example #3: Critical Paths

- Highlight the critical path and calculate the critical path timing where the gates have the propagation delays indicated in the table.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-input AND gate</td>
<td>4</td>
</tr>
<tr>
<td>3-input AND gate</td>
<td>2</td>
</tr>
<tr>
<td>OR gate</td>
<td>3</td>
</tr>
<tr>
<td>NAND gate</td>
<td>5</td>
</tr>
</tbody>
</table>

![Diagram of the circuit with timing delays]
Example #3: Critical Paths

- Highlight the critical path and calculate the critical path timing where the gates have the propagation delays indicated in the table.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Delay (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-input AND gate</td>
<td>4</td>
</tr>
<tr>
<td>3-input AND gate</td>
<td>2</td>
</tr>
<tr>
<td>OR gate</td>
<td>3</td>
</tr>
<tr>
<td>NAND gate</td>
<td>5</td>
</tr>
</tbody>
</table>
Multiplexers

- A **multiplexer** (mux for short) is a combinational unit with $N$ data inputs and 1 output bit.
- It takes another $\lceil \log_2 N \rceil$ input bits which together form a **control signal** (specifically, a **select signal**) that selects which of the $N$ inputs to connect to the output.
- For example, a 2:1 (“2-to-1”) multiplexer has two data inputs: a single select bit and a single output bit.
- When $S = 0$, the mux selects $D_0$. When $S = 1$, it selects $D_1$. 

```
<table>
<thead>
<tr>
<th>S</th>
<th>D_1</th>
<th>D_0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Alternative Mux Symbol

- You will also see muxes depicted in this manner, using a long oval shape. This is how they are drawn in the famous Hennessy & Patterson book on computer architecture.
- We will sometimes use this style in CSE 220 when drawing muxes, so be aware of that.
- The letters “mux” are optional
Multiplexer Implementation

\[ Y = D_0 \overline{S} + D_1 S \]

- Thanks to abstraction, once we understand how the mux works internally, we can then treat it as a black box and use it in circuits
Wider Multiplexers

• To create multiplexers with more inputs we have a few options
• One is to simply implement a larger circuit from the sum-of-products equation
• Another is to combine multiple, smaller multiplexers together
• For example, a 4:1 multiplexer has 4 inputs and 2 select signals →
• An implementation based on the SOP form would suggest four 3-input AND gates (1 data bit, 2 select bits) and a single 4-way OR gate
Wider Multiplexers

- As you can imagine, as we increase the number of inputs to 8, 16, etc., the number of gates gets very large
Wider Multiplexers

- The other option is to connect smaller multiplexers together
- This is more scalable and simpler to work with
- However, if the target platform doesn’t have these smaller multiplexers available, we may have to use the other approach
- As with software designers, hardware designers have to contend with trade-offs and the availability of implementation technology (or lack thereof)
Muxes for Bus Selection

- Suppose we had two devices connected to a shared 32-bit bus
- We want to select one device to transmit data over the bus
- We can use 32 1-bit multiplexers
- The same select bit is used as the select bit for all 32 2:1 muxes
- The circuit on the right takes two 32-bit input data signals and sends only one of the 32-bit data inputs on the output (F)
- This will be really useful later when we see how to implement an ALU
Example #1: Mux as Lookup Table

- A mux can be used to implement a **lookup table**
- That is, we can use a mux to simply map every combination of the input signals to a desired output (0 or 1)
- The mux on the right shows how we could implement $A \text{ AND } B$ using a 4:1 mux
- The input values are in fact used as the control signals
- Connect *ground* (triangle) to those mux inputs that should be mapped to 0, and connect high voltage (flat bar) to those mux inputs that should be mapped to 1
Example #2: Mux as Lookup Table

• With some analysis and careful thought, we can use a multiplexer with $2^{N-1}$ inputs to implement an $N$-input logic function

• The idea is to use one of the literals (as well as 0s and 1s) to the mux data inputs

• The basic procedure is this: start with the truth table you want to implement (both the inputs and the output)

• Combine pairs of rows to eliminate the rightmost input variable by expressing the output in terms of either (a) this variable, or (b) 0, or (c) 1

• Let’s go back to the AND lookup table to see how this works
Example #2: Mux as Lookup Table

- Take a close look at the first two rows
- What is the relationship between the values of $Y$ and $B$ when $A$ is 0?
- In a sense there isn’t one. $Y$ is 0 regardless of the value of $B$ (when $A$ is 0)
- What about the bottom two rows? How are $Y$ and $B$ related?
- Actually, $Y$ is just the value of $B$
- Use $A$ as the select signal, connect 0 to input #0 and $B$ to input #1
Example #3: Mux as Lookup Table

• Let’s try this process again with another function: XOR

\[ Y = A \oplus B \]

<table>
<thead>
<tr>
<th></th>
<th>( A )</th>
<th>( B )</th>
<th>( Y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ A \]

\[ \overline{B} \]

\[ Y \]

\[ 0 \]

\[ 1 \]
Example #4: Mux as Lookup Table

- Suppose we want to implement this function:
  \[ Y = A\bar{B} + \bar{B}\bar{C} + \bar{A}BC \]
- How would we implement this using an 8:1 mux?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Example #5: Mux as Lookup Table

- \( Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC \)
- How would we implement this using a 4:1 mux?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Example #5: Mux as Lookup Table

\[ Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC \]

• How would we implement this using a 4:1 mux?
Example #6: Mux as Lookup Table

- $Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC$
- How would we implement this using only 2:1 muxes?
- First we need to choose two variables that will be used as selector bits and connect muxes in series.
- Let's take $A$ again as the first selector bit and then $B$. You can use whichever ones you want, really, but then you would have to rewrite the truth table to reorder the var’s.
- For the $A$ mux, the expression for the 0 input is $\overline{B}\overline{C} + BC$
- For the 1 input it is $\overline{B}\overline{C} + \overline{B}C = \overline{B}$
Example #6: Mux as Lookup Table

- We need to replace the expression on the mux’s 1 input with a 2:1 multiplexer

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ B'C' + BC \]
Example #6: Mux as Lookup Table

<table>
<thead>
<tr>
<th>B</th>
<th>C</th>
<th>$\overline{B} \overline{C} + BC$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>

When $B = 0$, output is $C'$
When $B = 1$, output is $C$
Aside: Minimal Complete Sets

• A set of gates is called a **complete** or **universal set** if you can implement *any* logical function using only the types of gates in the set
  • You can use as many gates as you like
• A **minimal complete set** is a complete set with no redundant elements
• For example, \{AND, OR, NOT\} is a (non-minimal) complete set because any other gate can be implemented using them
• \{AND, NOT\} is a minimal complete set. By De Morgan’s law we can implement OR using just AND and NOT gates
• \{MUX\} is a minimal complete set
• \{NAND\} and \{NOR\} are also minimal complete sets
Aside: Minimal Complete Sets

• For example, here is how AND, NOT and OR gates can be implemented using only NAND gates:

AND gate

NOT gate

OR gate
Implementing Circuits with Muxes

• The truth table-based way is not the only way to build a circuit using multiplexers
• An alternate approach relies on Boolean algebra and is a more general way of proceeding
• First, we need an expression written in SOP form or something close to it. If the expression is not in SOP form, we might rewrite it using rules of Boolean algebra.
• Then, we choose which input variable(s) will be used as input(s) to the multiplexer(s). Pick variables that appear frequently and in both complemented and true forms.
• Next, factor out these variables from all terms
• The remaining terms are then inputted into the mux at the corresponding inputs
Example #1: Mux w/ Algebra

• Implement the function $F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC$ using only 2:1 muxes. Note: not in SOP form!
Example #1: Mux w/ Algebra
Example #1: Mux w/ Algebra

• Implement the function \( F = ABC + \overline{A}BC + \overline{A}\overline{B}C + BC \) using only 2:1 muxes. Note: not in SOP form!
• We factor out only one variable at a time because a 2-input mux has only 1 selector bit
  \[
  F = ABC + \overline{A}BC + \overline{A}\overline{B}C + (A + \overline{A})BC
  \]
  \[
  F = ABC + \overline{A}BC + \overline{A}\overline{B}C + ABC + \overline{A}BC
  \]
• Let’s choose \( A \) as the first selector bit
  \[
  F = A(BC + BC) + \overline{A}(BC + \overline{B}C + BC)
  \]
• Simplify the inner expressions
• Note: \( BC + BC = B \) and \( BC + \overline{B}C + BC = \overline{C} + BC \)
  \[
  F = A(B) + \overline{A}(\overline{C} + BC)
  \]
  \[
  F = A(B) + \overline{A}(\overline{C} + B)
  \]
Example #1: Mux w/ Algebra

- $F = A(B) + \bar{A}(\bar{C} + B)$
- So $B$ is attached to the 1 input for the $A$ mux and $\bar{C} + B$ is attached to the 0 input for the $A$ mux
- But, we have only 2-input muxes, so we need to apply the procedure again to implement $\bar{C} + B$
- We will attach $B$ to the 1 input for the $C$ mux and 1 to the 0 input for the $C$ mux
Example #2: Mux w/ Algebra

• Implement the function \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC \) using only 2:1 muxes. Use \( C \) and then \( B \) as the selector bits.

• Often we can avoid writing out the complete POS form, which is unwieldy for functions of four or more variables!

• Pick variables that occur most frequently in the expression
Example #2: Mux w/ Algebra

• Implement the function $F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC$ using only 2:1 muxes. Use $C$ and then $B$ as the selector bits.

• Often we can avoid writing out the complete POS form, which is unwieldy for functions of four or more variables!

• Pick variables that occur most frequently in the expression

• $F = C(B) + \bar{C}(AB + \bar{A}B + \bar{A}\bar{B})$

• $F = C(B) + \bar{C}(\bar{A} + AB)$

• $F = C(B) + \bar{C}(\bar{A} + B)$
Example #3: Mux w/ Algebra

• Implement the function $G = \overline{A}C + BC + AB\overline{C}$ using only 2:1 muxes. Use $C$ and then $B$ as the selector bits.
Example #3: Mux w/ Algebra

• Implement the function $G = \bar{A}C + BC + AB\bar{C}$ using only 2:1 muxes. Use $C$ and then $B$ as the selector bits.

  $G = C(\bar{A} + B) + \bar{C}(AB)$

  $G = C(B1 + \bar{B}\bar{A}) + \bar{C}(BA + \bar{B}0)$

• Using the “no name” identity twice here
Example #4: Mux w/ Algebra

• Implement the function $F = ABC + \overline{A}BC + \overline{A}\overline{B}C + BC$ using only 4:1 muxes

• $F = ABC + \overline{A}BC + \overline{A}\overline{B}C + ABC + \overline{A}BC$
Example #4: Mux w/ Algebra
Example #4: Mux w/ Algebra

• Implement the function \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC \) using only 4:1 muxes
• \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + \bar{A}BC \)
• We need to factor out two variables because a 4-input mux has two selector bits
• We need four 2-variable minterms. We will have this if we choose \( A \) and \( B \).
• But we are missing \( A\bar{B} \) from \( F \). So we add it: \( A\bar{B}0 \)
• \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + \bar{A}BC + A\bar{B}0 \)
• \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + \bar{A}BC + A\bar{B}0 \)
• \( F = AB(C + \bar{C}) + \bar{A}B(C + \bar{C}) + \bar{A}\bar{B}\bar{C} + A\bar{B}0 \)
• \( F = AB1 + \bar{A}B1 + \bar{A}\bar{B}\bar{C} + A\bar{B}0 \)
Example #4: Mux w/ Algebra

- \( F = AB1 + \bar{A}B1 + \bar{A}\bar{B}\bar{C} + A\bar{B}0 \)
Example #5: Mux w/ Algebra

• Implement the function $F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC$
  using only 4:1 muxes

• $F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + \bar{A}BC$

• Once again we need to factor out two variables because a 4-input mux has two selector bits

• We need four 2-variable minterms. This time let’s pick $B$ and $C$ as the selector bits.
Example #5: Mux w/ Algebra
Example #5: Mux w/ Algebra

- Implement the function $F = ABC + \overline{A}BC + \overline{A}\overline{B}C + BC$ using only 4:1 muxes

$F = ABC + \overline{A}BC + \overline{A}\overline{B}C + ABC + \overline{A}BC$

- Once again we need to factor out two variables because a 4-input mux has two selector bits

- We need four 2-variable minterms. This time let’s pick $B$ and $C$ as the selector bits.

- But we are missing $\overline{B}C$ from $F$. So we add it: $\overline{B}C0$

$F = ABC + \overline{A}BC + \overline{A}\overline{B}C + ABC + \overline{A}BC + \overline{B}C0$

$F = ABC + \overline{A}BC + \overline{A}\overline{B}C + ABC + \overline{A}BC + \overline{B}C0$

$F = B\overline{C}(A + \overline{A}) + BC(A + \overline{A}) + \overline{A}\overline{B}C + \overline{B}C0$

$F = B\overline{C}1 + BC1 + \overline{A}\overline{B}C + \overline{B}C0$
Example #5: Mux w/ Algebra

\[ F = B\bar{C}1 + BC1 + \bar{A}\bar{B}\bar{C} + \bar{B}C0 \]
Tips on using Muxes

• Selecting different variables as the selector bits will result in different networks and possibly different numbers of required gates

• By selecting a variable which appears in most terms first and also variables which appear in a roughly equal number of times in complemented and uncomplemented forms you will obtain a smaller implementation and possibly a shorter critical path

• If only the uncomplemented value of an input variable is available (e.g., $A$ but not $\bar{A}$), then the input signal must be split and an inverter be used to generate the complemented variable
Decoders

- A decoder is a combinational unit with $N$ inputs and $2^N$ outputs. Only one of the outputs is asserted (i.e., has a value 1) at a given point in time. All other outputs are 0.
- In general, a circuit can provide an $N$-bit “code” to an $N:2^N$ decoder to select one of the $2^N$ outputs.
- So for example, a decoder could be used to select from among a set of $2^N$ devices.

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$Y_3$</th>
<th>$Y_2$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Decoder Implementation

• How can we implement the logic of a decoder?
• Consider this: the particular combination of input signals determines which output is 1
• For instance, if \( A_1 = 0 \) and \( A_0 = 1 \), that indicates the circuit is selecting output line #1. So if \( \overline{A}_1A_0 = 1 \), we select #1.
• So we need to combine the inputs with an AND gate to determine which output line to select
• In other words, each output is a minterm
2:4 Decoder Implementation

Compare with multiplexer:

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$Y_3$</th>
<th>$Y_2$</th>
<th>$Y_1$</th>
<th>$Y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
3:8 Decoder

\[ A_2A_1A_0 \rightarrow \text{3:8 Decoder} \]

| \( O_0 \) | \( A_2'A_1'A_0' \) |
| \( O_1 \) | \( A_2'A_1'A_0 \) |
| \( O_2 \) | \( A_2'A_1A_0' \) |
| \( O_3 \) | \( A_2'A_1A_0 \) |
| \( O_4 \) | \( A_2A_1'A_0' \) |
| \( O_5 \) | \( A_2A_1'A_0 \) |
| \( O_6 \) | \( A_2A_1A_0' \) |
| \( O_7 \) | \( A_2A_1A_0 \) |
3:8 Decoder

<table>
<thead>
<tr>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$O_0$</th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
<th>$O_4$</th>
<th>$O_5$</th>
<th>$O_6$</th>
<th>$O_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Decoder Implementation

- Decoders provide a very convenient means of implementing Boolean functions written in SOP form.
- Consider XNOR: \( Y = \overline{A \oplus B} = \overline{A} \overline{B} + AB \)
- The idea is to combine the outputs of the decoder that correspond to the minterms of this equation with an OR gate.

- An \( N \)-input function with \( M \) 1s in the truth table can be built with an \( N: 2^N \) decoder and an \( M \)-input OR gate.
Multiplexer from Decoder

• We can even implement a multiplexer using a decoder
Use 2:4 Decoder for 3-var Function

- To implement: $F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC$
- Suppose we use a 2:4 decoder with $A$ and $B$ as inputs

2:4 Decoder

$O_0 = A'B'$
$O_1 = A'B$
$O_2 = AB'$
$O_3 = AB$
Use 2:4 Decoder for 3-var Function

• To implement: \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + BC \)

• Suppose we use a 2:4 decoder with \( A \) and \( B \) as inputs

• \( F = AB\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}\bar{C} + ABC + \bar{A}BC \)

• \( F = O_3\bar{C} + O_1\bar{C} + O_0\bar{C} + O_3C + O_1C \)

• \( F = O_3 + O_1 + O_0\bar{C} \)
Supplementary Material
Encoders

- An encoder performs the opposite function of a decoder
- It has $2^N$ inputs and $N$ outputs
- For a **simple encoder**, whichever (single) line is selected on the input (has a signal of 1), the corresponding binary number is presented on the output
- For unspecified input combinations (e.g., 0011, 0110), the outputs are don’t cares
- As its name suggests, an encoder is used to encode data – change it from one format to another

<table>
<thead>
<tr>
<th>$I_3$</th>
<th>$I_2$</th>
<th>$I_1$</th>
<th>$I_0$</th>
<th>$O_1$</th>
<th>$O_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Priority Encoders

• Suppose we relax the constraint that only one input line is selected

• Which output should be selected in that case?

• For a priority encoder, the input line with the highest priority will be selected

• A priority encoder has a variety of applications

• Consider several computer components that wish to use the system bus at the same time. A priority encoder will pick the device with highest priority.

\[
\begin{array}{cccccc}
I_3 & I_2 & I_1 & I_0 & O_1 & O_0 \\
0 & 0 & 0 & 0 & X & X \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & X & 0 & 1 \\
0 & 1 & X & X & 1 & 0 \\
1 & X & X & X & 1 & 1 \\
\end{array}
\]
Example: 3-bit SM to 1’s Comp.

- Design a circuit that converts from 3-bit sign/magnitude to 3-bit 1’s complement

<table>
<thead>
<tr>
<th>Base 10 Value</th>
<th>Sign/magnitude Representation (X)</th>
<th>1’s Complement Representation (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>111 (7)</td>
<td>100 (4)</td>
</tr>
<tr>
<td>-2</td>
<td>110 (6)</td>
<td>101 (5)</td>
</tr>
<tr>
<td>-1</td>
<td>101 (5)</td>
<td>110 (6)</td>
</tr>
<tr>
<td>-0</td>
<td>100 (4)</td>
<td>111 (7)</td>
</tr>
<tr>
<td>0</td>
<td>000 (0)</td>
<td>000 (0)</td>
</tr>
<tr>
<td>1</td>
<td>001 (1)</td>
<td>001 (1)</td>
</tr>
<tr>
<td>2</td>
<td>010 (2)</td>
<td>010 (2)</td>
</tr>
<tr>
<td>3</td>
<td>011 (3)</td>
<td>011 (3)</td>
</tr>
</tbody>
</table>
Example: 3-bit SM to 1’s Comp.
Example: 3-bit SM to 1’s Comp.

- **Decoder**: $X_2X_1X_0 \rightarrow 3:8$ Decoder
  - Outputs: $0, 1, 2, 3, 4, 5, 6, 7$

- **Encoder**: $0, 1, 2, 3, 4, 5, 6, 7 \rightarrow Y_2Y_1Y_0$

- **Connection**: $X_2X_1X_0$ to $3:8$ Decoder, $3:8$ Decoder to $8:3$ Encoder, $8:3$ Encoder to $Y_2Y_1Y_0$
Demultiplexers

- A demultiplexer (demux) performs the opposite function of a multiplexer.
- It connects or routes its single input to one of $2^N$ outputs depending on the value of the selector/control bits.
- If the binary value of the control bits is $k$, then the input is connected to the $k^{\text{th}}$ output.
- If the input value is always 1, then a demux behaves exactly like a decoder.
- Basically, a demux is a decoder plus some extra AND gates. Control bits select exactly which AND gate to let the input value through to.
- Used, for example, to send an incoming communications signal to the correct recipient on a network.
Demultiplexers

\[
\begin{array}{c|c|c|c|c|c|c|c}
S_1 & S_0 & I_0 & F_3 & F_2 & F_1 & F_0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]