Review Problem 43

- If we only have inverters and standard 2-input gates, how many gates are needed to build a 3:8 decoder?

\[ D_7: (A \cdot B) \cdot \overline{C} \]
\[ D_6: (A \cdot B) \cdot \overline{C} \]

2 gates

3 inverters: \( \overline{A}, \overline{B}, \overline{C} \)

4 gates: \( A \cdot B, AB, \overline{A} \cdot \overline{B}, \overline{A} \overline{B} \)

8 gates: \( C \oplus (AB), C \oplus (AB) \) ...
Decoders in General Logic Implementation

- Implement $F = WXZ + \bar{YZ}$ w/4x16 Decoder
Encoders

- Performs the inverse operation of decoders
- Input: $2^N$ or less lines -- only 1 is asserted at any given time
- Output: N output lines
- Function: the output is the binary representation of the ID of the input line that is asserted

![Diagram of an encoder with inputs D3, D2, D1, D0 and outputs A1, A0]
Encoder Implementation

- **4:2 Encoder**

<table>
<thead>
<tr>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<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>

\[ A1 = D2 \lor D3 \]
\[ AO = D1 \lor D3 \]

\[ \text{Problems} \]

1. Can't tell the difference between \( D0 = 0 \) calls and \( D0 = 1 \) extra "valid" output.
2. If \( D2 = 1 \lor D1 = 1 \) looks like \( D3 \) is calling.

Priority: highest \# wins.
Priority Encoder

- Use priorities to resolve the problem of 2 or more input lines active at a time.
- One scheme: Highest ID active wins
- Also add an output to identify when at least 1 input active

<table>
<thead>
<tr>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>A1</th>
<th>A0</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Priority Encoder Implementation

\[ A1 = D3 + D2 \]

\[ A0 = D3 + \overline{D2} \overline{D1} \]

\[ \text{Valid} = D3 + D2 + D1 + D0 \]
Priority Encoder Implementation (cont.)
module basicEncoder4_2 (out, in);
    output logic [1:0] out;
    input logic [3:0] in;
    always_comb begin
        assert (in == 4'b0001 || in == 4'b0010 || in == 4'b0100 || in == 4'b1000);
    end

    always_comb begin
        case (in)
            4'b0001: out = 2'b00;
            4'b0010: out = 2'b01;
            4'b0100: out = 2'b10;
            4'b1000: out = 2'b11;
            default: out = 2'bxx;
        endcase
    end
endmodule