Create a truth table for a circuit which tells if a 3-bit number is evenly divisible by 3 (num/3 leaves no remainder). Have a separate output for the unsigned, 2's comp, and sign-magnitude versions.

<table>
<thead>
<tr>
<th>B2</th>
<th>B1</th>
<th>B0</th>
<th>Uns</th>
<th>S - M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Review Problem 38
Converting (9)\text{10} to 6-bit Two's Complement

\[ 111011 = 1 + 110100 = (100100)_{\text{2}} = (9)_{\text{10}} \]

Convert (9)\text{10} to 6-bit Two's Complement

\[ 01010 \]

Converting Decimal to Two's Complement

Fixed width, then negate if necessary

Convert absolute value to unsigned binary, then
Convert (01101)\textsuperscript{2} to Decimal

\[ -610 = - (01100) = - (10100) = - (111111) = - - 01010 = - - (0000) = 01010 \]

Convert (11010)\textsuperscript{2} to Decimal

If Negative, negate then convert.

If Positive, convert as normal.

Converting Two's Complement to Decimal
\[ 5 - 2 = \begin{aligned} \binom{10100000}{1+00100000} - 2 & = \\
\binom{11011111}{11011111} - 2 & = \end{aligned} \]

Convert (11112) to 8-bit Two's Complement

\[ -5 + 2 = \begin{aligned} \binom{00000000}{00000000} - 2 & = \\
\binom{11111111}{11111111} & = \end{aligned} \]

Convert (00102) to 8-bit Two's Complement

\[ \text{Complement (N>M), simply duplicate sign bit:} \]

\[ \text{To convert from N-bit to M-bit Two's} \]

\[ \text{Sign Extension} \]
Design individual subsystems.

Connection

Break problem into subsystems, identifying

Understand the problem

Design process:

Controller

Light

Traffic

Timer

Replace with communicating sub-circuits

Many problems too complex to build as one

Solving Complex Problems
Design a digital clock, which can display the seconds, minutes, and hours. Have three inputs: reset seconds, increment minute, and increment hour. The clock can be implemented using a set of flip-flops. Example Problem: Input verification.
Complex Problem Example (cont.)
Complex Problem Example (cont.)
What is the smallest number of bits required to represent that number in 2's complement?

For the 6-bit 2's complement number (111010), how would it be represented in 10-bit 2's complement?
FPGAs: Programmable for arbitrary functions

Elements for interconnections

Multiplexer/Decoder/Encoder: Standard routing

Single chip that performs very complex computations

Alternatively: Complex chips for standard functions

Requires numerous chips to build interesting circuits

1 chip = 2-8 gates

Standard TTL Small-Scale Integration:

Readings: 4-4.1, 4.2, 4.3-4.3.2

Basic Circuit Elements
Put together a simple telephone system

Design Example: Basic Telephone System
Operator controls all connections
- Multiple subscribers, one operator.

Basic Telephone System (cont.)
Decoder: Convert binary to one-of-N wires

Encoder: Determine which input is active

Multiplexer: Combine N sources onto 1 wire

Used in Networks, Computers, Stereos, etc.

Blocks

Develop implementations of important "Building Standard Circuit Elements"
Example: 3-to-8 decoder

A decoder that has $n$ inputs and $m$ outputs is referred to as an $n \times m$, $N \times M$, or $n$-$to$-$m$ decoder.

A decoder that has $N$ bits, output $2^N$ outputs -- only one is true.

Input: $N$ bits; output: $2^N$ outputs based on $N$ input bits

Used to select one of $2^N$ outputs based on $N$ input bits

Decoders
Problem: Draw a circuit.

Decoded (5:16, 16:1)

\[ D_0 = \overline{5150} \]
\[ D'_0 = 5150 \]
\[ D_1 = \overline{5150} \]
\[ D'_1 = 5150 \]
\[ D_2 = \overline{5150} \]
\[ D'_2 = 5150 \]

Decoder Implementation

<table>
<thead>
<tr>
<th>SI</th>
<th>S0</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D0</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>En</td>
<td>S1</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>I</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>I</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Active High enable

Enable Decoder Implementation
module decoder2 # (in, out, enable);

always_comb begin
    case (in)
        0, 1, 2, 3, 4, 5, 6, 7:
            out = 4'b0000;
        default:
            out = 4'b0000;
    endcase
end

endmodule
Decoders in General Logic Implementation

Implement F = \overline{YZ} + WXZ with a 4x16 Decoder.
Function: the output is the binary representation

Output: \( N \) output lines

Given time

Input: \( 2^n \) or less lines -- only 1 is asserted at any

Perform the inverse operation of decoders

---

Encoders