Basic Circuit Elements

- Readings: 4-4.1.1, 4.2, 4.3-4.3.2
- Standard TTL Small-Scale Integration:
  - 1 chip = 2-8 gates
  - Requires numerous chips to build interesting circuits
- Alternative: Complex chips for standard functions
  - Single chip that performs very complex computations
- Multiplexer/Decoder/Encoder: Standard routing elements for interconnections
- FPGAs: Programmable for arbitrary functions

Design Example: Basic Telephone System

- Put together a simple telephone system

Diagram of a basic telephone system with: Ear, Mouth, Bell, Start.
Basic Telephone System (cont.)

- Multiple subscribers, one operator.
  - Operator controls all connections

Standard Circuit Elements

- Develop implementations of important “Building Blocks”
  - Used in Networks, Computers, Stereos, etc.
- Multiplexer: Combine N sources onto 1 wire
- Encoder: Determine which input is active
- Decoder: Convert binary to one-of-N wires
Decoders

- Used to select one of $2^N$ outputs based on N input bits
- Input: N bits; output: $2^N$ outputs -- only one is true
- A decoder that has $n$ inputs and $m$ outputs is referred to as an $n \times m$, $N:M$, or $n$-to-$m$ decoder
- Example: 3-to-8 decoder

Decoder Implementation

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Enabled Decoder Implementation

- Active High enable

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

Enabled Decoders in Verilog

```verilog
module enDecoder2_4 (out, in, enable);
output logic [3:0] out;
input logic [1:0] in;
inport logic enable;

always_comb begin

endmodule
```
Decoder Expansion

- Construct a 4:16 decoder using 2:4 decoders

Decoders in General Logic Implementation

- Implement $F = WXZ + Y\overline{Z}$ 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

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>
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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Priority Encoder Implementation
Priority Encoder Implementation (cont.)

Basic Encoders in Verilog

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

endmodule
Priority Encoders in Verilog

module priorityEncoder4_2 (out, in, valid);
output logic [1:0] out;
input logic [3:0] in;
output logic valid;

always_comb begin

endmodule

Multiplexer

- An element that selects data from one of many input lines and directs it to a single output line
- Input: $2^N$ input lines and $N$ selection lines
- Output: the data from one selected input line
- Multiplexer often abbreviated as MUX
Multiplexer Implementation

- 4:1 MUX

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>D3</td>
</tr>
</tbody>
</table>

Multiplexer Expansion

- Construct a 16:1 MUX using 4:1 MUX's
Multiplexers in General Logic

- Implement $F = XYZ + YZ$ with a 8:1 MUX

![8:1 MUX Diagram](image1)

Multiplexers in General Logic (cont.)

- Implement $F = XYZ + YZ$ with a 4:1 MUX

![4:1 MUX Diagram](image2)