Multiplexers (Data Selectors)

- A multiplexer (MUX for short) is a digital switch:
  - it passes (connects) one of its data inputs to the output
  - the data input selected is a function of a set of control inputs called selection inputs

\[ \text{OUT} = S' \cdot D_0 + S \cdot D_1 \]
Multiplexers Do Selecting

- Selecting of data or information is a critical function in digital systems and computers.
- Circuits that perform selecting have:
  - A set of $n$ information inputs $D_i$ from which the selection is made.
  - A set of $k$ control (select) lines for making the selection.
  - A single output.

$n \leq 2^k$ inputs

$k$ select lines

Multiplexer Analogy

Example Uses of Multiplexers

- In computers to select among signals
- To implement command:
  \[ \text{if } A=0 \text{ then } Z=X \cdot Y \]
  \[ \text{else } Z=X \oplus Y \]
- Trip controller in a car to display mileage, time, speed, etc.

Multiplexer Structure

- Switch circuit equivalent
- Multiplexer is unidirectional
Example: 4-to-1-line Multiplexer

- Expression for OUT:

\[ \text{OUT} = S_1 \cdot S_0 \cdot D_0 + S_1 \cdot S_0 \cdot D_1 + S_1 \cdot S_0 \cdot D_2 + S_1 \cdot S_0 \cdot D_3 \]

or: \( \text{OUT} = \sum_{j=0}^{2^n-1} M_j \cdot D_j \)

- Circuit implementation: Sum-of-Products
  - 4 AND gates (4 product terms)
  - 2-to-4 line decoder (to generate the minterms)

---

Example: 4-to-1-line Multiplexer

- 2-to-2^2-line decoder
- \(2^2 \times 2\) AND-OR
**General Multiplexer Equation**

- A general logic equation for a multiplexer output:
  \[ i_Y = \sum_{j=0}^{n-1} EN \cdot M_j \cdot iD_j \]
- Logical sum of product terms
- Variable \( i_Y \) is a particular output bit \((1 \leq i \leq b)\)
- \( M_j \) is a minterm \( j \) of the \( s \) select inputs

---

**Gate Level Implementation of MUXs**

- **2:1 MUX**
  - Positive logic:
  - Negative logic:

- **4:1 MUX**
Multiplexer Standard Packaging

IC has limited number of pins (16)
\[
n \cdot b + b + s + 1 \leq 16 - 2
\]

\[
\text{in} \quad \text{out} \quad \text{SEL} \quad \text{EN}
\]

\[
(n+1) \cdot b + \lceil \log_2 n \rceil \leq 13
\]

\[
\begin{array}{ccc|c|c}
\text{b} & \text{n} & \text{s} & \text{12} & \text{8 input 1 bit} & \text{74x151} \\
1 & 8 & 3 & (12) & & \\
2 & 4 & 2 & (12) & \text{dual 4 input 2 bit} & \text{74x153} \\
4 & 2 & 1 & (13) & \text{quad 2 input 4 bit} & \text{74x157}
\end{array}
\]

Truth table of 74x151

- Truth table for 74x151 8-input, 1-bit multiplexer
- Only “control” inputs are listed under “Inputs”
- Outputs specified as 0” or “1”, or a simple logic function of “data” inputs (e.g., D0 or D0’)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN_L</td>
<td>S2 S1 S0</td>
</tr>
<tr>
<td>1 x x x</td>
<td>0 1</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>D0 D0’</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>D1 D1’</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>D2 D2’</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>D3 D3’</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>D4 D4’</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>D5 D5’</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>D6 D6’</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>D7 D7’</td>
</tr>
</tbody>
</table>
74x151 8-input 1-bit Multiplexer

- **74x151 logic diagram and logic symbol**

74x151

- EN
- S0
- S1
- S2
- D0
- D1
- D2
- D3
- D4
- D5
- D6
- D7

74x157 2-input 4-bit Multiplexer

- **74x157 selects between two 4-bit inputs**

74x157

- EN
- S
- D0
- D1
- 2D0
- 2D1
- 3D0
- 3D1
- 4D0
- 4D1
- 1Y
- 2Y
- 3Y
- 4Y
74x157 2-input 4-bit Multiplexer

- 74x157
  truth table:

<table>
<thead>
<tr>
<th>EN_L</th>
<th>S</th>
<th>1Y</th>
<th>2Y</th>
<th>3Y</th>
<th>4Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1D0</td>
<td>2D0</td>
<td>3D0</td>
<td>4D0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1D1</td>
<td>2D1</td>
<td>3D1</td>
<td>4D1</td>
</tr>
</tbody>
</table>

Control signals S1 and S0 simultaneously choose one of D0, D1, D2, D3 and one of D4, D5, D6, D7

Control signal S2 chooses which of the upper or lower mux's output to gate to OUT

Cascading/Expanding Multiplexers

- Large multiplexers can be made by cascading smaller ones

---

alternatives implementation

---

OUT
### Cascading/Expanding Multiplexers

- Combining four 74x151s to make a 32-to-1 multiplexer
- 74x138 3-to-8 decoder used as 2-to-4 decoder for two high-order bits to enable one of 74x151s

### Multiplexers as General-purpose Logic

- A $2^n:1$ multiplexer can implement any function of $n$ variables
  - with the variables used as control inputs and
  - the data inputs tied to 0 or 1

- Example:

  \[
  F(A,B,C) = \sum_{m=0}^{7} M_m = A'B'C' + A'B'C + AB'C' + AB'C + A'B'C + A'B'C + AB'C + AB'C
  \]

  \[
  = A'B'C'+(1) + A'B'C-(0) + A'B'C'+(1) + A'B'C-(0) +
  A'B'C-(0) + A'B'C-(0) + A'B'C'+(1) + A'B'C-(1)
  \]

  \[
  \text{OUT} = A'B'C'I0 + A'B'C'I1 + A'B'C'I2 + A'B'C'I3 +
  A'B'C'I4 + A'B'C'I5 + A'B'C'I6 + A'B'C'I7
  \]
Multiplexers as General-purpose Logic

- **Generalization:**
  data inputs can also be tied to variables not just 0’s and 1’s

<table>
<thead>
<tr>
<th>( I_0 )</th>
<th>( I_1 )</th>
<th>...</th>
<th>( I_{n-1} )</th>
<th>( I_n )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

\( n-1 \) mux control variables

single mux data variable

four possible configurations of truth table rows can be expressed as a function of \( I_n \)

---

Multiplexers as Function Generators

- We can generate the four possible \( Z \) values (0, 1, \( Z \), \( Z' \)) and realize the function with half the values
- Realizing \( F = \sum_{X,Y,Z} \) (0,3,5,6) with a 4-input multiplexer:
4-variable Function using 8-input Multiplexer

- Realizing \( F = \sum_{N_0,N_1,N_2,N_3} (1,2,3,5,7,11,13) \) with an 8-input multiplexer:

<table>
<thead>
<tr>
<th>Row</th>
<th>( N_0 )</th>
<th>( N_1 )</th>
<th>( N_2 )</th>
<th>( N_3 )</th>
<th>( F )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Demultiplexers

- Route a single input to one of many outputs, as a function of a set of control inputs
Demultiplexers

- Demultiplexer function is just the inverse of multiplexer’s
- A binary decoder with enable input can be used as a demultiplexer

Demultiplexer Analogy

- A MUX driving a bus and a demultiplexer receiving the bus
Exclusive-OR Gates

- **Exclusive-OR (XOR)** gate is a 2-input gate whose output is “1” if exactly one of its input is “1” – or: XOR gate produces a “1” output if its inputs are different.

- **Exclusive-NOR (XNOR) or Equivalence** just opposite—produces output “1” if its inputs are the same.

  - **XOR:** \( X \oplus Y = X' \cdot Y + X \cdot Y' \)
  
  - **XNOR:** \( (X \oplus Y)' = X \cdot Y + X' \cdot Y' \)

Exclusive-OR Gates

- Truth table and gate-level implementation

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>X \oplus Y</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

AND-OR implementation:

three-level NAND implementation:

\[(X \cdot Y)' \cdot (X+Y) = (X'+Y') \cdot (X+Y) = X' \cdot Y + Y' \cdot X\]
XOR versus XNOR

\[(X \oplus Y)' = (X' \cdot Y + X \cdot Y')' = (X + Y') \cdot (X' + Y)\]
\[= X' \cdot X' + X' \cdot Y' + X \cdot Y + Y' \cdot Y\]
\[= X \cdot Y + X' \cdot Y'\]

\(X \rightarrow X'\) out \(\rightarrow\) out’ … the circuit XOR = XOR

\[(X' \cdot Y' + X \cdot Y)' = X' \cdot Y + X \cdot Y\]

\(X \rightarrow X'\) out \(\rightarrow\) out’ … the circuit XNOR = XNOR

\[X' \cdot Y' + X \cdot Y\]

\[X \oplus 1 = X \cdot 0 + X' \cdot 1 = X'\]

Equivalent Symbols for XOR/XNOR

- **XOR gates**

- **XNOR gates**

- **Simple rule:**
  - Any two signals (inputs or output) of an XOR or XNOR gate may be complemented w/o changing the resulting logic function

\[X \oplus 0 = X\quad X \oplus 1 = X'\]
Cascading XOR gates

- Daisy-chain connection
  - Sum modulo 2
  - Parity computation

- Balanced tree structure
  - \( \log_2(N) \)

The tree structure is faster than daisy-chain connection because the gates depth of its treelike structure is \( \log_2(N) \), which is much less than \( N-1 \) for a daisy-chain structure.

Both are called odd-parity circuits because its output is “1” if an odd number of inputs are “1”
  - Used to generate and check parity bits in computer systems
    - Detects any single-bit error

Even-parity circuit has odd-parity circuit’s output inverted—its output is “1” if an even number of its inputs are “1”
Example Parity Computation

F = I₁ ⊕ I₂ ⊕ I₃

F = 1  ODD number of “1” in the input

<table>
<thead>
<tr>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
<th>F</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>0</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>1</td>
</tr>
</tbody>
</table>