ECE 320

Introduction to Logic Design
Combinational Logic Design Practices

- Documentation Standards
- Programmable Logic Devices
- Combinational Logic Design Structures
- Decoders and Encoders
- Three-State Buffers
- Multiplexers and Demultiplexers
- Exclusive OR Gates and Comparators
- Adders and Subtracters
- Arithmetic Logic Units (ALUs)
- Multipliers
- ROMs
Sequential Logic
Design Principles and Practices

- Bistable Logic Elements and Metastability
- Latches and Flip-Flops
- Analysis of State Machines
- Synthesis of State Machines
Combinational Logic Design Practices
Documentation Standards

- Block Diagrams
  - First step in hierarchical design

- Schematic Diagrams

- HDL Programs (ABEL, Verilog, VHDL)

- Timing Diagrams

- Circuit Descriptions
Schematic Diagrams

- Details of component inputs, outputs, and interconnections
- Pin numbers
- Title blocks
- Names for all signals
- Page-to-page connectors
Example Schematic
Schematic diagram/Logic Diagram

Logic Diagram

Schematic Diagram

Intro to Logic Design 4 — 9
Flat Schematic Structure
Other Documentation

**Timing diagrams**

- Output from simulator
- Specialized timing-diagram drawing tools

**Circuit descriptions**

- Text (word processing)
- Can be as big as a book (e.g., typical Cisco ASIC descriptions)
- Typically incorporate other elements (block diagrams, timing diagrams, etc.)
Gate Symbols

AND

OR

BUFFER

NAND

NOR

INVERTER
Which symbol to use?
Answer depends on signal names and active levels.
Signal Names and Active Levels

Signal names are chosen to be descriptive

**Active levels:** HIGH or LOW

<table>
<thead>
<tr>
<th>Active Low</th>
<th>Active High</th>
</tr>
</thead>
<tbody>
<tr>
<td>READY~</td>
<td>READY+</td>
</tr>
<tr>
<td>ERROR.L</td>
<td>ERROR.H</td>
</tr>
<tr>
<td>ADDR15(L)</td>
<td>ADDR15(H)</td>
</tr>
<tr>
<td>RESET*</td>
<td>RESET</td>
</tr>
<tr>
<td>ENABLE~</td>
<td>ENABLE</td>
</tr>
<tr>
<td>~GO</td>
<td>GO</td>
</tr>
<tr>
<td>/RECEIVE</td>
<td>RECEIVE</td>
</tr>
<tr>
<td>TRANSMIT_L</td>
<td>TRANSMIT</td>
</tr>
</tbody>
</table>
Example

- **HIGH** when error occurs
- **LOW** when error occurs

- **ERROR**
- **ERROR_L**
- **ERROR1_L**
Active Levels for Pins

In logic gates and logic structures the **inversion bubble** indicates the **active** level of the signal.

**Examples:** 2-to-4 Decoder

![Diagram of a 2-to-4 decoder showing active low and active high inputs and outputs.]
Bubble-to-Bubble Logic Design

Rules:
The active level of the output signal of a logic device should match the active level of the device’s output pin.

The active level of the input signal of a logic device should match the active level of the device’s input pin.
Bubble-to-Bubble Logic Design

**Purpose:** To make it easy to understand the function of the Logic circuit
Drawing Layouts

Inputs to the left, outputs to the right. Signals flow from left to right.

Crossing lines/Connected lines (T-type connection)

Buses should be named: DATA[0-7], CONTROL

A signal extracted from a bus should be named

Broken signal paths should be flagged to indicate the source or destination and direction.

Bus Flags:
- Unidirection
- Bidirection

Documentation
Delay depends on

• Internal circuit structure
• Logic Family type
• Source Voltage
• Temperature
Timing Diagram for Data Signals (Bus)

IN / WRITE Logic Circuit (Memory) OUT

/ WRITE

IN

OUT

old data

new data

new data

t1

t2

t3

t4

t hold

t setup

t OUTmin

t OUTmax

Intro to Logic Design 4– 22
Propagation Delay

The Propagation Delay is the delay time between input transitions and the output transitions due to the propagation delay of the logic gates.

$t_p$ of a signal depends on the signal path inside the logic circuit.

To find $t_p$ for a signal, add the propagation delays of all gates along the path of the signal.

For a logic gate $t_{PLH}$ may not equal $t_{PHL}$.

$t_p$ is specified in the manufacturer data sheets of the IC’s.

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>Typical (ns)</th>
<th>Maximum (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$t_{PLH}$</td>
<td>$t_{PHL}$</td>
</tr>
<tr>
<td>74LS00</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>74ACT00</td>
<td>5.5</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Programmable Logic Arrays (PLAs)

Any combinational logic function can be realized as a sum of products.

Idea: Build a large AND-OR array with lots of inputs and product terms, and programmable connections.

- \( n \) inputs
  - AND gates have 2\( n \) inputs - true and complement of each variable

- \( m \) outputs, driven by large OR gates
  - Each AND gate is programmably connected to each output’s OR gate

- \( p \) AND gates (\( p << 2^n \))
Example: 4x3 PLA, 6 Product Terms

Programmable Logic Arrays

$n$ inputs

$p$ AND gates

$m$ outputs
Compact Representation

Actually, closer to physical layout (‘wired logic’)
Some Product Terms

\[ O_1 = I_1 I_2 + I_1' I_2' I_3' I_4' \]

\[ O_2 = I_1 I_3' + I_1' I_3 I_4 + I_2 \]

\[ O_3 = I_1 I_2 + I_1 I_3' + I_1' I_2' I_4' \]
PLA Electrical Design
Programmable Array Logic (PALs)

How beneficial is product sharing?

Not enough to justify the extra AND array

PALs $\rightarrow$ fixed OR array

Example: PAL16L8
Programmable Array Logic

- 10 primary inputs
- 7 ANDs per output and 1 AND for 3-state enable
- 8 outputs
- 6 outputs available as inputs
Decoders

Converts input code words into output code words.

One-to-One mapping: Each input code produces only one output code.

Binary Code
Gray Code
BCD Code
Any Code...

Typically \( n \) inputs, \( 2^n \) outputs: 2-to-4, 3-to-8, 4-to-16, etc.
Binary 2-to-4 decoder

$n$-to-$2^n$ decoder: $n$ inputs and $2^n$ outputs.

Example: 2-to-4 decoder

<table>
<thead>
<tr>
<th>Binary Code</th>
<th>2-to-4 decoder</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>$Y_0$</td>
</tr>
<tr>
<td>$I_1$</td>
<td>$Y_1$</td>
</tr>
<tr>
<td>$Y_2$</td>
<td></td>
</tr>
<tr>
<td>$EN$</td>
<td>$Y_3$</td>
</tr>
</tbody>
</table>

1-out-of-$2^n$

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$EN$</td>
<td>$I_1$</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$X$: don’t care

One output is asserted for each input code.
2-to-4- Decoder Logic Diagram
MSI Decoder

Active Low Enable

Active Low outputs

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>/G</td>
<td>/Y3</td>
</tr>
<tr>
<td>B</td>
<td>/Y2</td>
</tr>
<tr>
<td>A</td>
<td>/Y1</td>
</tr>
<tr>
<td></td>
<td>/Y0</td>
</tr>
<tr>
<td>1 X X</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
</tbody>
</table>
Complete 74x139 Decoder

Input buffering → less load

NAND gates → faster
74x138 3-to-8 Decoder
### 74x138 3-to-8-Truth Table

<table>
<thead>
<tr>
<th>/G1</th>
<th>/G2A</th>
<th>/G2</th>
<th>C</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>1</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</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>
<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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<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>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>/Y7</th>
<th>/Y6</th>
<th>/Y5</th>
<th>/Y4</th>
<th>/Y3</th>
<th>/Y2</th>
<th>/Y1</th>
<th>/Y0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Decoder Cascading: 4-to-16

Two 3-to-8
More Cascading: Four 3-to-8 Decoder

Four 3-to-8
Decoder Applications

Microprocessor memory systems
  • selecting different banks of memory

Microprocessor input/output systems
  • selecting different devices

Microprocessor instruction decoding
  • enabling different functional units

Memory chips
  • enabling different rows of memory depending on address

Lots of other applications
Implementing the Canonical Sum

The binary decoder generates all minterms of n-variable logic function.

The canonical sum of a logic function is obtained by adding all minterms of that function:

- Match the order of input bits
- Activate Enable inputs

Example:

\[ F = \Sigma_{x,y,z} (2,4,7) \]
Logic design using Decoders

**Advantages:**
- Flexibility
- Multiple-output Logic functions

**Disadvantages:**
- Complexity: for large number of inputs

A practical alternative: PLD’s
Seven-Segment Displays

Displays decimal numbers and some characters

**LED** (Light Emitting Diode) or **LCD** (Liquid Crystal Display)

- **CommonAnode (CA)**
  - Requires Active Low inputs
- **CommonCathode (CC)**
  - Requires Active High inputs
# Seven-Segment Decoders/Drivers

<table>
<thead>
<tr>
<th>BCD Code</th>
<th>Seven-Segment Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input D</td>
<td>Output a</td>
</tr>
<tr>
<td>C</td>
<td>b</td>
</tr>
<tr>
<td>B</td>
<td>c</td>
</tr>
<tr>
<td>A</td>
<td>d</td>
</tr>
<tr>
<td>A</td>
<td>e</td>
</tr>
<tr>
<td>A</td>
<td>f</td>
</tr>
<tr>
<td>A</td>
<td>g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>C</th>
<th>B</th>
<th>A</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Decoder Applications**

BCD Code

Seven-Segment Code

Input Table

Output Table

Diagram of Seven-Segment Display
MSI Seven-Segment Decoders

CC Driver: 74x49

CA Driver: 74x47

Decoder Applications
74x49 Driver

Driving CC Seven-Segment Display

+5V

74x49

BCD Code

A
B
C
D

BI

a
b
c
d
e
f
g
Encoders vs. Decoders

Decoder

Encoder

Inverse function of a Decoder.

Outputs are less than inputs.

ENCODER

Converts input code words into output code words.
## Binary Encoders

$2^n$-to-$n$ encoder: $2^n$ inputs and $n$ outputs.

**Example:** $n=3$, 8-to-3 encoder

<table>
<thead>
<tr>
<th>Inputs</th>
<th>1-out-of-$2^n$</th>
<th>Outputs</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_0$</td>
<td>1 0 0 0 0 0 0 0</td>
<td>$Y_0$ 0 0 0</td>
<td>$Y_0 = I_4 + I_5 + I_6 + I_7$</td>
</tr>
<tr>
<td>$I_1$</td>
<td>0 1 0 0 0 0 0 0</td>
<td>$Y_1$ 0 0 1</td>
<td>$Y_1 = I_2 + I_3 + I_6 + I_7$</td>
</tr>
<tr>
<td>$I_2$</td>
<td>0 0 1 0 0 0 0 0</td>
<td>$Y_2$ 0 1 0</td>
<td>$Y_2 = I_1 + I_3 + I_5 + I_7$</td>
</tr>
<tr>
<td>$I_3$</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 1 1</td>
<td></td>
</tr>
<tr>
<td>$I_4$</td>
<td>0 0 0 0 1 0 0 0</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>$I_5$</td>
<td>0 0 0 0 0 1 0 0</td>
<td>1 0 1</td>
<td></td>
</tr>
<tr>
<td>$I_6$</td>
<td>0 0 0 0 0 0 1 0</td>
<td>1 1 0</td>
<td></td>
</tr>
<tr>
<td>$I_7$</td>
<td>0 0 0 0 0 0 0 1</td>
<td>1 1 1</td>
<td></td>
</tr>
</tbody>
</table>
Binary Encoders

Limitations:

- Only one input can be activated
- IO has no effect

Application:

- Handling multiple devices requests. But, no simultaneous requests.
- Establishing priorities solve the problem of multiple requests.

\[
Y_2 = I_1 + I_3 + I_5 + I_7 \\
Y_1 = I_2 + I_3 + I_6 + I_7 \\
Y_0 = I_4 + I_5 + I_6 + I_7
\]
Need Priority in Most Applications
8-Input Priority Encoder

1. Assign priorities to the inputs
2. When more than one input are asserted, the output generates the code of the input with the highest priority
Priority-Encoder Logic Equations

(Highest Priority)

\[
\begin{align*}
H_7 &= I_7 \\
H_6 &= I_6 . I_7' \\
H_5 &= I_5 . I_6' . I_7' \\
H_4 &= I_4 . I_5' . I_6' . I_7' \\
H_3 &= I_3 . I_4' . I_5' . I_6' . I_7' \\
H_2 &= I_2 . I_3' . I_4' . I_5' . I_6' . I_7' \\
H_1 &= I_1 . I_2' . I_3' . I_4' . I_5' . I_6' . I_7' \\
H_0 &= I_0 . I_1' . I_2' . I_3' . I_4' . I_5' . I_6' . I_7' \\
\text{IDLE} &= I_0' . I_1' . I_2' . I_3' . I_4' . I_5' . I_6' . I_7' \\
A_0 &= I_1 + I_3 + I_5 + I_7 \\
A_1 &= I_2 + I_3 + I_6 + I_7 \\
A_2 &= I_4 + I_5 + I_6 + I_7
\end{align*}
\]
74x148 8-Input Priority Encoder

- Active-low I/O
- EI: Enable Input
- GO: Got Something
- EO: Enable Output
Encoders
# 74x148 Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_L</td>
<td>A2_L</td>
</tr>
<tr>
<td>I0_L</td>
<td>A1_L</td>
</tr>
<tr>
<td>I1_L</td>
<td>A0_L</td>
</tr>
<tr>
<td>I2_L</td>
<td>GS_L</td>
</tr>
<tr>
<td>I3_L</td>
<td>EO_L</td>
</tr>
<tr>
<td>I4_L</td>
<td></td>
</tr>
<tr>
<td>I5_L</td>
<td></td>
</tr>
<tr>
<td>I6_L</td>
<td></td>
</tr>
<tr>
<td>I7_L</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E_L</th>
<th>I0_L</th>
<th>I1_L</th>
<th>I2_L</th>
<th>I3_L</th>
<th>I4_L</th>
<th>I5_L</th>
<th>I6_L</th>
<th>I7_L</th>
<th>A2_L</th>
<th>A1_L</th>
<th>A0_L</th>
<th>GS_L</th>
<th>EO_L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Encoders**
Encoder Application (Monitoring Unit)
BCD Encoder

Switch 0

Switch 9

Switch 0

Switch 9

+5V

BCD encoder

I0
I1
I2
I3
I4
I5
I6
I7
I8
I9

Idles

Y0
Y1
Y2

74x49

a
b

B
a
b

C
c
d

d

e

D
e
f

g

Intro to Logic Design 4

Encoders
Cascading Priority Encoders

32-input priority encoder
Three-State Buffers

Output = LOW, HIGH, or Hi-Z.

**Hi-Z:** The output is floating (High Impedance) when the enable input is deasserted →

The input is isolated from the output

---

### Application:

- Can tie multiple outputs together, if at most one at a time is driven.
- Controlling the access of a **single line/bus** by multiple devices.

---

<table>
<thead>
<tr>
<th>EN</th>
<th>A</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>Hi-Z</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>Hi-Z</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
</tbody>
</table>
Different Types

Buffers

Active High Enable  Active Low Enable

Inverters

Active High Enable  Active Low Enable

Three-State Buffers
8 Data Sources Sharing One Line

Three-State Buffers
Three-State Buffers
SSI/MSI Three-State Buffers

74x125: 4 independent buffers, Active Low enable

74x126: 4 independent buffers, Active High enable

74x540: 8 inverters with a common enable input

74x541: 8 buffers with a common enable input

74x240: 2 sets of 4 inverters with a common enable for each set

74x241: 2 sets of 4 buffers with a common enable for each set

74x245: Octal three-state transceiver
  8 pairs of buffers connected in opposite directions
Three-State Drivers

(a) 74x541
1  G1
19 G2
2   A1  Y1
3   A2  Y2
4   A3  Y3
5   A4  Y4
6   A5  Y5
7   A6  Y6
8   A7  Y7
9   A8  Y8

G1_L
(1)
(19)

G2_L

(b)  

A1 (2) Y1
A2 (3) Y2
A3 (4) Y3
A4 (5) Y4
A5 (6) Y5
A6 (7) Y6
A7 (8) Y7
A8 (9) Y8

Intro to Logic Design 4 – 64
Driver Application

Intro to Logic Design 4 – 65

Three-State Buffers
Three-State Transceiver

Intro to Logic Design
Transceiver Application
**Multiplexers**

**Multiplexing:** transmitting large number of signals over a small number of channels or lines

Digital multiplexer (**MUX**): selects one of many input lines and directs it to a single output.

Selection lines controls the selection of a particular input

n selection lines, $2^n$ inputs, single output
Multiplexers

A multiplexer is a circuit that selects one of several inputs to pass through to the output. The selection is controlled by a set of inputs called the select inputs. The enable input, when active, allows the multiplexer to function. The diagram shows a multiplexer with an enable input (EN), select inputs (SEL), and input data sources (D0, D1, ..., Dn1). The output (Y) is connected to the selected data input when the enable is active.
4-to-1 Line Multiplexer

<table>
<thead>
<tr>
<th>S1</th>
<th>SO</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>I0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>I1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>I2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>I3</td>
</tr>
</tbody>
</table>

Inputs

4x1 MUX

Select

Output

Intro to Logic Design 4 – 70
Logic Diagram of 4-to-1 Multiplexer

I0
I1
I2
I3
S1
S0
Y
Properties of Different Approaches

74x151: 8-to-1 MUX

74x251: 8-to-1 MUX with three-state output

74x153: 4-to-1 2 bit MUX

74x157: 2-to-1 4 bit MUX
74x151 8-Input Multiplexer
## 74x151 Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN_L</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Implementing Logic Functions

An $n$-variable logic function can be implemented using $2^n$-to-1 MUX.

The inputs variables are connected to the select input.

The function value for each input combination (0 or 1) is connected to the corresponding input of the MUX.

**Example:** $F = \sum_{x,y} (1,3)$

<table>
<thead>
<tr>
<th>Row</th>
<th>$X$</th>
<th>$Y$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Multiplexers
**Example:** \( F = \Sigma x,y,z \ (2,4,7) \) using 74x151

<table>
<thead>
<tr>
<th>Row</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram of 74x151 multiplexer](image-url)
**Functional Decomposition**

An effective way for using MUX to implement Logic Functions.

n-row truth table can be implemented using \( n/2 \)-to-1 MUX:

- Write the Logic function in terms of the least significant input variable.
- The truth table is reduced by one half.

For 3-variable Logic Function, the decomposed truth table is:

<table>
<thead>
<tr>
<th>Row</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>( F_{00}(Z) )</td>
</tr>
<tr>
<td>2,3</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>( F_{01}(Z) )</td>
</tr>
<tr>
<td>4,5</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>( F_{10}(Z) )</td>
</tr>
<tr>
<td>6,7</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>( F_{11}(Z) )</td>
</tr>
</tbody>
</table>

![Diagram of 4x1 MUX with inputs I0, I1, I2, I3 and select lines S1, S0, and output F]
### Functional Decomposition Example

#### Truth Table

<table>
<thead>
<tr>
<th>Row</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- \( F_{00}(Z) = 0 \)
- \( F_{01}(Z) = Z' \)
- \( F_{10}(Z) = Z' \)
- \( F_{11}(Z) = Z \)

#### Decomposed Truth Table

<table>
<thead>
<tr>
<th>Row</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2,3</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>Z'</td>
</tr>
<tr>
<td>4,5</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>Z'</td>
</tr>
<tr>
<td>6,7</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>Z</td>
</tr>
</tbody>
</table>

---

**Intro to Logic Design**

---

**Multiplexers**
Demultiplexers

Demultiplexer (DMUX) performs the opposite function of a MUX.

A digital Demultiplexer receives input data on a single input and transmits it on one of $2^n$ possible outputs according to the value of the $n$ select inputs.

MUX/DMUX are used in data transmission.
1-to-4 DMUX

<table>
<thead>
<tr>
<th>IN</th>
<th>S1</th>
<th>S2</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>IN</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>IN</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IN</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IN</td>
</tr>
</tbody>
</table>

Input | 1x4 DMUX | Outputs

Select

Intro to Logic Design 4 – 80

Multiplexers
Using Decoders as DMUX

A DMUX has the same structure of a Decoder with enable input. Decoder can be used as a DMUX by connecting the input data to the enable input.

**Example**: 2-to-4 Decoder can be used as 1-to-4 DMUX

<table>
<thead>
<tr>
<th>IN</th>
<th>S1</th>
<th>S2</th>
<th>D0</th>
<th>D1</th>
<th>D2</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>IN</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>IN</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IN</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>IN</td>
</tr>
</tbody>
</table>

**Diagram**

![Diagram of 2-to-4 Decoder as DMUX](image-url)
2-input XOR Gates

Like an OR gate, but **excludes** the case where both inputs are 1.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>XOR</th>
<th>XNOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

\[(X \oplus Y)' = X \cdot Y + X' \cdot Y' \]
XOR and XNOR Symbols
Gate-level XOR Circuits

No direct realization with just a few transistors.

\[ F = X \oplus Y \]

\[ F = X \oplus Y \]
SSI XOR

74x86: 4 XOR gates

74x266: 4 XNOR gates
Parity Circuit: Daisy Chain

Odd Parity Circuit: The output is 1 if odd number of inputs are 1

Even Parity Circuit: The output is 1 if even number of inputs are 1

- Used to generate and check parity bits in computer systems.

- Detects any single-bit error
Parity Tree

Faster with balanced tree structure
XOR Example:

**Daisy-Chain Structure**

Input: 1101

Even Parity output: 0

Odd Parity output: 1

**Tree structure**

Even Parity output: 0

Odd Parity output: 1
Next

1. Comparators
2. Adders
3. Multipliers
4. Read-only memories (ROMs)
A Generic Digital Processor

RAM, ROM, Registers, ...

MEMORY

CPU

CONTROL

DATAPATH

INPUT - OUTPUT

Interconnect:
Switches, Arbiters, Bus, ...

Building Blocks for Digital Architectures

Finite state machine:
PLA, Counters, Flip-flops, Latches, ...

Arithmetic Unit:
Adder, Multiplier, Shifter, Comparator, ...

Intro to Logic Design 4 — 90
Bit-Sliced Design

Tile identical processing elements
Comparators

Compares Two binary words and indicate if they are equal

A → Comparator → A=B
B → Comparator → A=B

Advanced Comparators:

A → Comparator → A=B
B → Comparator → A>B
B → Comparator → A<B

1-bit Comparator: XOR gate, the Output is 1 if A ≠ B

A → XOR gate → F
B → XOR gate → F
Equality Comparators

1-bit comparator

4-bit comparator

Intro to Logic Design 4 - 93
Iterative Comparator

1 bit comparator:

<table>
<thead>
<tr>
<th>EQI</th>
<th>X</th>
<th>Y</th>
<th>EQO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</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>
Multibit Iterative Comparator

Iterative Comparator: cascaded 1 bit comparators

Intro to Logic Design 4– 95
Comparators
3 Cascading inputs

Cascading inputs initial values:

(A=B IN) = 1
(A>B IN) = 0
(A<B IN) = 0
8 bit Comparator

Least Significant bits

Most Significant bits

Intro to Logic Design 4 - 97

Comparators
# Half Adder

## Truth Table

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>SUM</th>
<th>C\text{OUT}</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>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>
</tbody>
</table>

\[ \text{SUM} = X \oplus Y \]

\[ C\text{OUT} = X \cdot Y \]
Full Adder

1-bit-wide adder, produces sum and carry outputs

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Cin</th>
<th>S</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</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>
</tbody>
</table>

\[
S = X'Y'C_{IN} + X'YC_{IN}' + XY'C_{IN}' + XYC_{IN}
\]

\[
S = X \oplus Y \oplus C_{IN}
\]

\[
C_{OUT} = XY + XC_{IN} + YC_{IN}
\]
Full-Adder Circuit

\[ S = X \oplus Y \oplus C_{IN} \]

\[ C_{OUT} = XY + XC_{IN} + YC_{IN} \]