Chapter # 4: Programmable and Steering Logic

Section 4.2
Non-Gate Logic

Introduction

AND-OR-Invert  Generalized Building Blocks
PAL/PLA  Beyond Simple Gates

Kinds of "Non-gate logic":

• switching circuits built from CMOS transmission gates
• multiplexer/selecter functions
• decoders
• tri-state and open collector gates
• read-only memories
Multiplexers/Selectors

Use of Multiplexers/Selectors

Multi-point connections

Mux: Chooses one of many inputs to steer to its single output under direction of control inputs

DeMux: Takes single data input and steers it to one of its many outputs under the direction of its control inputs
Steering Logic

Use of Multiplexer/Demultiplexer in Digital Systems

So far, we've only seen point-to-point connections among gates

Mux/Demux used to implement multiple source/multiple destination interconnect
Multiplexers/Selectors

Named by the number of data inputs and number of data outputs

\[ Z = A' I_0 + A I_1 \]

\[ Z = A' B' I_0 + A' B I_1 + A B' I_2 + A B I_3 \]

\[ Z = A' B' C' I_0 + A' B' C I_1 + A' B C' I_2 + A' B C I_3 + A B' C' I_4 + A B' C I_5 + A B C' I_6 + A B C I_7 \]

In general, \[ Z = \sum_{k=0}^{2^n-1} m_k \]
in minterm shorthand form for a \( 2^n:1 \) Mux
**Multiplexers/Selectors**

**General Concept**

\[ 2^n \text{ data inputs, } n \text{ control inputs, 1 output} \]

used to connect \( 2^n \) points to a single point

control signal pattern form binary index of input connected to output

\[ Z = A' I_0 + A I_1 \]

<table>
<thead>
<tr>
<th>( A )</th>
<th>( Z )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( I_1 )</th>
<th>( I_0 )</th>
<th>( A )</th>
<th>( Z )</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>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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Two alternative forms for a 2:1 Mux Truth Table*
Multiplexers/Selectors

Alternative Implementations

Gate Level Implementation of 4:1 Mux

thirty six transistors
Multiplexer/Selector

Large multiplexers can be implemented by cascaded smaller ones

Control signals B and C simultaneously choose one of I0-I3 and I4-I7

Control signal A chooses which of the upper or lower MUX’s output to gate to Z
Alternative 8:1 Mux Implementation

Multiplexer/Selector
Multiplexer/Selector

Multiplexers/selectors as a general purpose logic block

A \( 2^{n-1}:1 \) multiplexer can implement any function of \( n \) variables

\( n-1 \) control variables; remaining variable is a data input to the mux

**Example:** Implement as (1) \( 8:1 \) MUX and (2) \( 4:1 \) MUX

\[
F(A,B,C) = m_0 + m_2 + m_6 + m_7
\]

\[
= A' B' C' + A' B C' + A B C' + A B C
\]

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

**8:1 MUX**

<table>
<thead>
<tr>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>8:1 MUX</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0 0 1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 1 0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 0 1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1 1 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

"Lookup Table"

**4:1 MUX**

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>4:1 MUX</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>

© R.H. Katz  Transparency No. 8-10
Multiplexer/Selector
Generalization

n-1 Mux control variables
single Mux data variable

\[ \begin{array}{cccc}
I_1 & I_2 & \cdots & I_n \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
0 & I_n & I_n & 1 \\
\end{array} \]

Four possible configurations of the truth table rows
Can be expressed as a function of \( I_n, 0, 1 \)
Example:

G(A,B,C,D) can be implemented by an 8:1 MUX:

\[ G = A'B'C'(1) + A'B'C(D) + A'BC'(0) + A'BC(1) + AB'C'(D') + AB'C(D) + ABC'(D') + ABC(D') \]
Decoders/Demultiplexers

**Decoder:** single data input, \( n \) control inputs, \( 2^n \) outputs

Control inputs (called select \( S \)) represent Binary index of output to which the input is connected

Data input usually called "enable" (\( G \))

Named by the number of control signals and number of output signals

1:2 Decoder: (2 outputs)

\[
\begin{align*}
O_0 &= G \cdot S; \\
O_1 &= G \cdot S
\end{align*}
\]

2:4 Decoder: (4 outputs)

\[
\begin{align*}
O_0 &= G \cdot S_0 \cdot S_1 \\
O_1 &= G \cdot S_0 \cdot S_1 \\
O_2 &= G \cdot S_0 \cdot S_1 \\
O_3 &= G \cdot S_0 \cdot S_1
\end{align*}
\]

3:8 Decoder: (8 outputs)

\[
\begin{align*}
O_0 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_1 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_2 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_3 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_4 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_5 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_6 &= G \cdot S_0 \cdot S_1 \cdot S_2 \\
O_7 &= G \cdot S_0 \cdot S_1 \cdot S_2
\end{align*}
\]
Decoders/Demultiplexers

Alternative Implementations

1:2 Decoder, Active High Enable

1:2 Decoder, Active Low Enable

2:4 Decoder, Active High Enable

2:4 Decoder, Active Low Enable
Decoder/Demultiplexer

Decoder as a Logic Building Block

Decoder Generates Appropriate Minterm based on Control Signals

Decoder as a Logic Building Block

Decoder Generates Appropriate Minterm based on Control Signals

Decoder as a Logic Building Block

Decoder Generates Appropriate Minterm based on Control Signals
**Decoder/Demultiplexer**

*Decoder as a Logic Building Block*

Example Function: \( F(A, B, C, D) \)

\[
\begin{align*}
F_1 &= A' \ B \ C' \ D + A' \ B' \ C \ D + A \ B \ C \ D \\
F_2 &= A \ B \ C' \ D' + A \ B \ C \\
F_3 &= (A' + B' + C' + D')
\end{align*}
\]

It is more convenient to reexpress the functions in their canonical sum of products form.

\[
\begin{align*}
F_1 &= A' \ B \ C' \ D + A' \ B' \ C \ D + A \ B \ C \ D \\
F_2 &= A \ B \ C' \ D' + A \ B \ CD' + ABCD \\
F_3 &= (ABCD)' \end{align*}
\]
Decoder/Demultiplexer

Decoder as a Logic Building Block

Example Function:  \( F(A,B,C,D) \)

<table>
<thead>
<tr>
<th>A B C D</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Decoder/Demultiplexer

Decoder as a Logic Building Block

If active low enable, then use NAND gates!

\[
\begin{align*}
F_1 &= A'B'C'D + A'B'CD + ABCD \\
F_2 &= ABC'D + ABCD' + ABCD \\
F_3 &= (ABCD)' \\
\end{align*}
\]
Decoder/Demultiplexer

Active Low Output Decoder

<table>
<thead>
<tr>
<th></th>
<th>G'</th>
<th>B</th>
<th>A</th>
<th>YO'</th>
<th>Y1'</th>
<th>Y2'</th>
<th>Y3'</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>H</td>
<td>X</td>
<td>X</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>L</td>
</tr>
</tbody>
</table>

© R.H. Katz   Transparency No. 8-19
Multiplexers/Decoders

5:32 Decoder

Contemporary Logic Design
Prog. & Steering Logic

© R.H. Katz  Transparency No. 8-20
Read-Only Memories

ROM: Two dimensional array of 1's and 0's

Internal storage elements of the ROM are set to their values once. After that are read only or may be erased and rewritten at a later time.

Row is called a "word";

Index selected by control inputs is called an "address"

Width of row is called bit-width or wordsize

Address is input, selected word is output

Internal Organization
Read-Only Memories

Not unlike a PLA structure with a fully decoded AND array!

ROM vs. PLA:

ROM approach advantageous when
(1) design time is short (no need to minimize output functions)
(2) most input combinations are needed (e.g., code converters)
(3) little sharing of product terms among output functions

ROM problem: size doubles for each additional input, can't use don't cares

PLA approach advantageous when
(1) design tool like espresso is available
(2) there are relatively few unique minterm combinations
(3) many minterms are shared among the output functions

PAL problem: constrained fan-ins on OR planes
Read-Only Memories

Example: Combination Logic Implementation

\[
\begin{align*}
F_0 &= A' \cdot B' \cdot C + A \cdot B' \cdot C' + A \cdot B \cdot C \\
F_1 &= A' \cdot B' \cdot C + A' \cdot B \cdot C' + A \cdot B \cdot C \\
F_2 &= A' \cdot B' \cdot C' + A' \cdot B' \cdot C + A \cdot B' \cdot C' \\
F_3 &= A' \cdot B \cdot C + A \cdot B' \cdot C' + A \cdot B \cdot C'
\end{align*}
\]

Address: 0 0 0
Word Contents:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F_0</th>
<th>F_1</th>
<th>F_2</th>
<th>F_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Read-Only Memories

2764 EPROM
8K x 8

<table>
<thead>
<tr>
<th>2764</th>
</tr>
</thead>
<tbody>
<tr>
<td>VPP</td>
</tr>
<tr>
<td>PGM</td>
</tr>
<tr>
<td>A12</td>
</tr>
<tr>
<td>A11</td>
</tr>
<tr>
<td>A10</td>
</tr>
<tr>
<td>A9</td>
</tr>
<tr>
<td>A8</td>
</tr>
<tr>
<td>A7</td>
</tr>
<tr>
<td>A6</td>
</tr>
<tr>
<td>A5</td>
</tr>
<tr>
<td>A4</td>
</tr>
<tr>
<td>A3</td>
</tr>
<tr>
<td>A2</td>
</tr>
<tr>
<td>A1</td>
</tr>
<tr>
<td>A0</td>
</tr>
<tr>
<td>CS</td>
</tr>
<tr>
<td>OE</td>
</tr>
</tbody>
</table>

16K x 16 Subsystem

buses -- several logically related wires that share a common function.
Steering Logic: Switches

Voltage Controlled Switches

"n-Channel MOS"

Metal Gate, Oxide, Silicon Sandwich

Diffusion regions: negatively charged ions driven into Si surface

Si Bulk: positively charged ions

By "pulling" electrons to the surface, a conducting channel is formed
Steering Logic

Voltage Controlled Switches

Logic 1 on gate, Source and Drain connected

Logic 0 on gate, Source and Drain connected
Steering Logic

- CMOS transmission gate is constructed from a normally open switch (nmos transistor) wired in parallel with a normally closed switch (pmos transistor), with complementary control signals.
Steering Logic

Transmission gates provide an efficient way to build steering logic. Steering logic circuits route data inputs to outputs based on the setting of control signals.

Logic Gates from Switches

- Inverter
- NAND Gate
- NOR Gate

*Pull-up* network constructed from pMOS transistors

*Pull-down* network constructed from nMOS transistors
### Inverter Operation

**Input is 1**
- Pull-up does not conduct
- Pull-down conducts
- Output connected to GND

**Input is 0**
- Pull-up conducts
- Pull-down does not conduct
- Output connected to VDD
Steering Logic

**NAND Gate Operation**

A = 1, B = 1
Pull-up network does not conduct
Pull-down network conducts
Output node connected to GND

A = 0, B = 1
Pull-up network has path to VDD
Pull-down network path broken
Output node connected to VDD
**Steering Logic**

**NOR Gate Operation**

\[ \begin{align*}
\text{A} = 0, \text{B} = 0 & \quad \text{Pull-up network conducts} \\
\text{Pull-down network broken} & \quad \text{Output node at VDD}
\end{align*} \]

\[ \begin{align*}
\text{A} = 1, \text{B} = 0 & \quad \text{Pull-up network broken} \\
\text{Pull-down network conducts} & \quad \text{Output node at GND}
\end{align*} \]
Read Appendix B.4 --- MOS Transistors, p. 675-681
Steering Logic

CMOS Transmission Gate

nMOS transistors good at passing 0's but bad at passing 1's

pMOS transistors good at passing 1's but bad at passing 0's

perfect "transmission" gate places these in parallel:

Switches  Transistors  Transmission or "Butterfly" Gate
Steering Logic

Selection Function/Demultiplexer Function with Transmission Gates

Selector:
Choose I0 if S = 0
Choose I1 if S = 1

Demultiplexer:
I to Z0 if S = 0
I to Z1 if S = 1
**Steering Logic**

**Well-formed Switching Networks**

Problem with the Demux implementation: multiple outputs, but only one connected to the input!

The fix: additional logic to drive every output to a known value

*Never allow outputs to "float"*
Decoders/Demultiplexers

Switch Logic Implementations

Naive, Incorrect Implementation
All outputs not driven at all times

Correct 1:2 Decoder Implementation
Steering Logic

Complex Steering Logic Example

N Input Tally Circuit: count # of 1's in the inputs
N+1 Outputs

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>Zero</th>
<th>One</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Conventional Logic for 1 Input Tally Function

Zero

One
Steering Logic

Switch Logic Implementation of Tally Function
**Steering Logic**

**Complex Steering Logic Example**

**Operation of the 1 Input Tally Circuit**

Input is 0, straight through switches enabled
Steering Logic

Complex Steering Logic Example
Operation of 1 input Tally Circuit

Input = 1, diagonal switches enabled
Steering Logic

Complex Steering Logic Example

Extension to the 2-input case

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>$I_2$</th>
<th>Zero</th>
<th>One</th>
<th>Two</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Conventional logic implementation

Switching Network -- 24 transistors

Gate Method -- 26 transistors
Steering Logic

Complex Steering Logic Example

Switch Logic Implementation: 2-input Tally Circuit

Cascade the 1-input implementation!
Complex Steering Logic Example

Operation of 2-input implementation

(a) I1=0, I2=0

(b) I1=1, I2=0

(c) I1=0, I2=1

(d) I1=1, I2=1
**Decoders/Demultiplexers**

**Switch Implementation of 2:4 Decoder**

**Operation of 2:4 Decoder**

- $S_0 = 0$, $S_1 = 0$
  - one straight thru path
  - three diagonal paths
Transmission Gate Implementation of 4:1 Mux

twenty transistors
Tri-State and Open-Collector

The Third State

- Logic States: "0", "1"
- Don't Care/Don't Know State: "X" (must be some value in real circuit!)
- Third State: "Z" — high impedance — infinite resistance, no connection

Tri-state gates: output values are "0", "1", and "Z"
- additional input: output enable (OE)

<table>
<thead>
<tr>
<th>A</th>
<th>OE</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When OE is high, this gate is a non-inverting "buffer"

When OE is low, it is as though the gate was disconnected from the output!

This allows more than one gate to be connected to the same output wire, as long as only one has its output enabled at the same time

Non-inverting buffer's timing waveform
Using tri-state gates to implement an economical multiplexer:

When SelectInput is asserted high
Input1 is connected to F

When SelectInput is driven low
Input0 is connected to F

This is essentially a 2:1 Mux
Tri-state and Open Collector

Alternative Tri-state Fragment

Active low tri-state enables plus inverting tri-state buffers
Switch Level Implementation of tri-state gate, inverting with active low enable

Tri-state and Open Collector
Open Collector

another way to connect multiple gates to the same output wire

gate only has the ability to pull its output low; it cannot actively drive the wire high

this is done by pulling the wire up to a logic 1 voltage through a resistor

Output is 0 only when A and B are both asserted.
Otherwise node F is floating.
The resistor will pull it up to a logic 1 voltage.
Tri-State and Open Collector

4:1 Multiplexer, Revisited

Decoder + 4 tri-state Gates
Tri-State and Open Collector

4:1 Multiplexer

Decoder + 4 Open Collector Gates