Combinational Circuits

Chapter 3
S. Dandamudi
Outline

• Introduction
• Multiplexers and demultiplexers
  * Implementing logical functions
  * Efficient implementation
• Decoders and encoders
  * Decoder-OR implementations
• Comparators

<table>
<thead>
<tr>
<th>• Adders</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Half-adders</td>
</tr>
<tr>
<td>* Full-adders</td>
</tr>
</tbody>
</table>
• Programmable logic devices
  * Programmable logic arrays (PLAs)
  * Programmable array logic (PALs)
• Arithmetic and logic units (ALUs)
Introduction

• Combinational circuits
  » Output depends only on the current inputs

• Combinational circuits provide a higher level of abstraction
  * Helps in reducing design complexity
  * Reduces chip count
  * Example: 8-input NAND gate
    » Requires 1 chip if we use 7430
    » Several 7400 chips (How many?)

• We look at some useful combinational circuits
Multiplexers

• Multiplexer
  * $2^n$ data inputs
  * n selection inputs
  * a single output

• Selection input determines the input that should be connected to the output

4-data input MUX

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$O$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$I_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$I_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$I_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$I_3$</td>
</tr>
</tbody>
</table>
Multiplexers (cont’d)

4-data input MUX implementation
Multiplexers (cont’d)

MUX implementations

Majority function

Even-parity function
Multiplexers (cont’d)

Example chip: 8-to-1 MUX

(a) Connection diagram

(b) Logic symbol
Multiplexers (cont’d)

Efficient implementation: Majority function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F₁</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>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</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>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram of a multiplexer](image)
Multiplexers (cont’d)

Efficient implementation: Even-parity function

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F₁</th>
<th>A</th>
<th>B</th>
<th>F₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original truth table

New truth table

Diagram of multiplexer implementation for even-parity function.
Multiplexers (cont’d)

74153 can be used to implement two output functions

(a) Connection diagram
(b) Logic symbol
Demultiplexers

Demultiplexer (DeMUX)
Demultiplexers (cont’d)

74138 can used as DeMUX and decoder

(a) Connection diagram  (b) Logic symbol  (c) Enable input logic details

2003 © S. Dandamudi  Chapter 3: Page 12
Decoders

- Decoder selects one-out-of-N inputs

<table>
<thead>
<tr>
<th>I₁</th>
<th>I₀</th>
<th>O₃</th>
<th>O₂</th>
<th>O₁</th>
<th>O₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Decoders (cont’d)

### Logic function implementation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C\textsubscript{in}</th>
<th>Sum</th>
<th>C\textsubscript{out}</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>

![Diagram of a decoder with inputs A, B, C\textsubscript{in}, and outputs O\textsubscript{0}, O\textsubscript{1}, O\textsubscript{2}, O\textsubscript{3}, O\textsubscript{4}, O\textsubscript{5}, O\textsubscript{6}, O\textsubscript{7}, Sum, and C\textsubscript{out}.

Decoders (cont’d)

74139: Dual decoder chip

(a) Connection diagram

(b) Logic symbol

Encoders

- Encoders
  - Take $2^B$ input lines and generate a $B$-bit binary number on $B$ output lines
  - Cannot handle more than one input with 1

<table>
<thead>
<tr>
<th>Enable input</th>
<th>$I_3$ $I_2$ $I_1$ $I_0$</th>
<th>$O_1$ $O_0$</th>
<th>Input active control signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 $X$ $X$ $X$ $X$</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 0</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 0 0</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 0 0 0</td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
Encoders (cont’d)

- Priority encoders
  * Handles inputs with more than one 1

<table>
<thead>
<tr>
<th>Enable input</th>
<th>I₃ I₂ I₁ I₀</th>
<th>O₁ O₀</th>
<th>Input active control signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X X X X</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>0 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 0 1 X</td>
<td>0 1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0 1 X X</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1 X X X</td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

![Diagram of priority encoder circuit]
Comparator

- Used to implement comparison operators (\(=\), \(>\), \(<\), \(\geq\), \(\leq\))
Comparator (cont’d)

4-bit magnitude comparator chip

(a) Connection diagram

(b) Logic symbol

Comparator (cont’d)

Serial construction of an 8-bit comparator
Adders

• Half-adder
  * Adds two bits
    » Produces a sum and carry
  * Problem: Cannot use it to build larger inputs

• Full-adder
  * Adds three 1-bit values
    » Like half-adder, produces a sum and carry
  * Allows building N-bit adders
    » Simple technique
      – Connect C_{out} of one adder to C_{in} of the next
    » These are called ripple-carry adders
Adders (cont’d)

(a) Half-adder truth table and implementation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>C_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>

(b) Full-adder truth table and implementation

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C_in</th>
<th>Sum</th>
<th>C_out</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>1</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>
Adders (cont’d)

A 16-bit ripple-carry adder
Adders (cont’d)

• Ripple-carry adders can be slow
  * Delay proportional to number of bits

• Carry lookahead adders
  * Eliminate the delay of ripple-carry adders
  * Carry-ins are generated independently

    » \( C_0 = A_0 \cdot B_0 \)
    » \( C_1 = A_0 \cdot B_0 \cdot A_1 + A_0 \cdot B_0 \cdot B_1 + A_1 \cdot B_1 \)
    » . . .

  * Requires complex circuits
  * Usually, a combination carry lookahead and ripple-carry techniques are used
Adders (cont’d)

4-bit carry lookahead adder

(a) Connection diagram

(b) Logic symbol
Programmable Logic Arrays

• PLAs
  * Implement sum-of-product expressions
    » No need to simplify the logical expressions
  * Take $N$ inputs and produce $M$ outputs
    » Each input represents a logical variable
    » Each output represents a logical function output
  * Internally uses
    » An AND array
      – Each AND gate receives $2N$ inputs
        » $N$ inputs and their complements
    » An OR array
Programmable Logic Arrays (cont’d)

A blank PLA with 2 inputs and 2 outputs

I₀
I₁

AND array

P₀  P₁  P₂  P₃

OR array

F₀  F₁
Programmable Logic Arrays (cont’d)

Implementation examples

\[ F_0 = \overline{A} \overline{B} + \overline{A} B + \overline{A} B \]
\[ F_1 = A \overline{B} + A B + A B \]
Programmable Logic Arrays (cont’d)

Simplified notation

\[ F_0 = A \overline{B} + \overline{A} B + \overline{A} \overline{B} \]
\[ F_1 = A \overline{B} + \overline{A} B + A B \]
Programmable Array Logic Devices

• Problem with PLAs
  * Flexible but expensive
  * Example
    » 12 × 12 PLA with
      – 50-gate AND array
      – 12-gate OR array
    » Requires 1800 fuses
      – 24 × 50 = 1200 fuses for the AND array
      – 50 × 12 = 600 fuses for the OR array

• PALs reduce this complexity by using fixed OR connections
  * Reduces flexibility compared PLAs
Programmable Array Logic Devices (cont’d)

Notice the fixed OR array connections

\[
F_0 = A\overline{B} + \overline{A}\overline{B}
\]

\[
F_1 = AB + \overline{A}\overline{B}
\]
Programmable Array Logic Devices (cont’d)

- An example PAL (Texas Instruments TIBPAL22V10-10C)
  - 22 X 10 PAL (24-pin DIP package)
    - 120-gate AND array
    - 10-gate OR array
  - 44 x 120 = 5280 fuses
    - Just for the AND array
      - OR array does not use any fuses
  - Uses variable number of connections for the OR gates
    - Two each of 8-, 10-, 12-, 14-, and 16-input OR gates
  - Uses internal feedback through a programmable output cell
Programmable Array Logic Devices (cont’d)

- MUX selects the input
  * $S_0$ and $S_1$ are programmed through fuses $F_0$ and $F_1$
Arithmetic and Logic Unit

Preliminary ALU design

<table>
<thead>
<tr>
<th>F₁</th>
<th>F₀</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>A and B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A or B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A + B</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A - B</td>
</tr>
</tbody>
</table>
Arithmetic and Logic Unit (cont’d)

Final design
Arithmetic and Logic Unit (cont’d)

16-bit ALU

![Diagram of 16-bit ALU with inputs and outputs labeled]
Arithmetic and Logic Unit (cont’d)

4-bit ALU

(a) Connection diagram

(b) Active low operands

(c) Active high operands
Summary

• Combinational circuits provide a higher level of abstraction
  * Output depends only on the current inputs

• Sample combinational circuits
  * Multiplexers and demultiplexers
  * Decoders and encoders
  * Comparators
  * Adders
    » Half-adder
    » Full-adder
Summary (cont’d)

• Programmable logic devices
  * PLAs
  * PALs

• Some more complete sets
  * Multiplexers
  * Decoder-OR
  * PLAs
  * PALs

• Looked at a very simple ALU design