Chapter 5

Combinatorial Components

- Transformational Components
- Conversion Components
- Interconnection Components
- Universal Components
Digital Components

- Digital components are divided into
  - **Combinatorial components**
    - Easy to design, partition, and test
  - **Sequential components**

Combinatorial Logic Circuit
(Logic Gates)

outputs = \( f \) (inputs)

\( n \) inputs \( \rightarrow \) \( m \) outputs

Sequential Logic Circuit
(Logic Gates)

outputs = \( f \) (inputs, time)

\( n \) inputs \( \rightarrow \) \( m \) outputs

Memory
Combinatorial Components

- **Data Transformation Components**
  - Arithmetic Operation (Add, Subtract, Multiply, Divide)
  - Logic Operation (AND, OR, NOT, ...)
  - Data Comparison (Greater-than, Equal, Less-than, ...)
  - Bit Manipulation (Shift, Rotate, Extract, ...)

- **Data Conversion Components**
  - Data Encoding
  - Data Decoding

- **Interconnection Components**
  - Source and Destination Selection
  - Bus Connections and Interface

- **Universal Components (Found in Control Units for Random Boolean Functions)**
  - Read-only Memories (ROMs)
  - Programmable Logic Arrays (PLA)
Design Principles

- **Encapsulation**
  - Define simple building blocks

- **Iteration**
  - Replicate building blocks as much as possible

- **Hierarchy**
  - Compose larger building blocks from smaller ones
Ripple-Carry (Serial) Adder

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$y_i$</th>
<th>$c_i$</th>
<th>$c_{i+1}$</th>
<th>$s_i$</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth Table

Map Representation

Graphic Symbol

Logic Schematic

8-bit Adder Unit Schematic
Carry Computation

Carry function:
\[ c_{i+1} = x_i y_i + c_i (x_i \oplus y_i) \]

Let \( g_i = x_i y_i \), \( p_i = x_i \oplus y_i \), then
\[
\begin{align*}
    c_{i+1} &= g_i + p_i c_i & (1) \\
    c_{i+2} &= g_{i+1} + p_{i+1} c_{i+1} & (2) \\
    c_{i+3} &= g_{i+2} + p_{i+2} c_{i+2} & (3) \\
    c_{i+4} &= g_{i+3} + p_{i+3} c_{i+3} & (4)
\end{align*}
\]

After forward substitution
\[
\begin{align*}
    c_{i+1} &= g_i + p_i c_i & (5) \\
    c_{i+2} &= g_{i+1} + p_{i+1} g_i + p_{i+1} p_i c_i & (6) \\
    c_{i+3} &= g_{i+2} + p_{i+2} g_{i+1} + p_{i+2} p_{i+1} g_i + p_{i+2} p_{i+1} p_i c_i & (7) \\
    c_{i+4} &= g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i + p_{i+3} p_{i+2} p_{i+1} p_i c_i & (8)
\end{align*}
\]

Carry-look-ahead function:
\[
    c_{i+4} = g_{(i,i+3)} + p_{(i,i+3)} c_i & (9)
\]

where
\[
\begin{align*}
    g_{(i,i+3)} &= g_{i+3} + p_{i+3} g_{i+2} + p_{i+3} p_{i+2} g_{i+1} + p_{i+3} p_{i+2} p_{i+1} g_i & (10) \\
    p_{(i,i+3)} &= p_{i+3} p_{i+2} p_{i+1} p_i & (11)
\end{align*}
\]
CLA Generator

4-bit Slice of Ripple–Carry Adder

4-bit Slice of Adder with CLA Generator

Copyright © 2004-2005 by Daniel D. Gajski
CLA Generator

Logic Schematic of CLA
Carry Chain Delays

4-bit Ripple–Carry Adder Slice

Carry Chains

<table>
<thead>
<tr>
<th>Carry Chains</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>from ( c_i(x_i, y_i) ) to ( c_{i+1} )</td>
<td>4.8 (9.0)</td>
</tr>
<tr>
<td>from ( c_i(x_i, y_i) ) to ( c_{i+2} )</td>
<td>9.6 (13.8)</td>
</tr>
<tr>
<td>from ( c_i(x_i, y_i) ) to ( c_{i+3} )</td>
<td>14.4 (18.6)</td>
</tr>
<tr>
<td>from ( c_i(x_i, y_i) ) to ( c_{i+4} )</td>
<td>19.2 (23.4)</td>
</tr>
<tr>
<td>from ( c_i(x_i, y_i) ) to ( g_{(i+1)} )</td>
<td>N/A</td>
</tr>
<tr>
<td>from ( c_i(x_i, y_i) ) to ( p_{(i+1)} )</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Ripple and CLA Delays for 4-bit Adder Slice
16-bit CLA Adder

16-bit Adder with Single-Level CLA Generator

16-bit Adder with Two-Level CLA Generator

<table>
<thead>
<tr>
<th>Carry Chains</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ripple</td>
</tr>
<tr>
<td>from (c_i(x_i, y_i)) to (c_4)</td>
<td>19.2 (23.4)</td>
</tr>
<tr>
<td>from (c_i(x_i, y_i)) to (c_8)</td>
<td>38.4 (42.6)</td>
</tr>
<tr>
<td>from (c_i(x_i, y_i)) to (c_{12})</td>
<td>57.6 (61.8)</td>
</tr>
<tr>
<td>from (c_i(x_i, y_i)) to (c_{16})</td>
<td>76.8 (81.0)</td>
</tr>
</tbody>
</table>

Ripple and CLA Delays for 16-bit Adder Slice
Two’s Complement Adder/Subtractor

- Two’s complement subtraction
  \[ A - B = A + B' + 1 \]

- Subtraction Procedure
  - Complement \( B \)
  - Set input carry to 1
  - Add to \( A \)

Truth Table

<table>
<thead>
<tr>
<th>( S )</th>
<th>Function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( A + B )</td>
<td>Addition</td>
</tr>
<tr>
<td>1</td>
<td>( A + B' + 1 )</td>
<td>Subtraction</td>
</tr>
</tbody>
</table>

8-bit Adder/Subtractor Unit Schematic

Copyright © 2004-2005 by Daniel D. Gajski

Slides by Philip Pham, University of California, Irvine
16-Function Logic Unit

- Boolean functions of two variables have two inputs, one output, and four minterms.
- There are 16 Boolean functions of four variables (4 control lines)

\[ s_i = S_0 m_0 + S_1 m_1 + S_2 m_2 + S_3 m_3 \]
\[ = S_0 x'_1 y'_i + S_1 x'_1 y_i + S_2 x_i y'_i + S_3 x_i y_i \]
16-Function Logic Unit

- Boolean functions of two variables have two inputs, one output, and four minterms.
- There are 16 Boolean functions of four variables (4 control lines)

\[ s_i = S_0 m_0 + S_1 m_1 + S_2 m_2 + S_3 m_3 \]
\[ = S_0 x'_i y'_i + S_1 x'_i y_i + S_2 x_i y'_i + S_3 x_i y_i \]
Arithmetic-Logic Unit

- **AE = Arithmetic Extender** (Add, Subtract, Increment, Decrement)
- **LE = Logic Extender** (AND, OR, PASS, NOT)

![4-bit Arithmetic-Logic Unit Schematic](image-url)
Arithmetic Extender

<table>
<thead>
<tr>
<th>$M$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>Function Name</th>
<th>$F$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$c_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Decrement</td>
<td>$A - 1$</td>
<td>$A$</td>
<td>all 1’s</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Add</td>
<td>$A + B$</td>
<td>$A$</td>
<td>$B'$</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Subtract</td>
<td>$A - B$</td>
<td>$A$</td>
<td>$B'$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Increment</td>
<td>$A + 1$</td>
<td>$A$</td>
<td>all 0’s</td>
<td>1</td>
</tr>
</tbody>
</table>

**Truth Table**

<table>
<thead>
<tr>
<th>$M$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$b_i$</th>
<th>$y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<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>
<td>0</td>
</tr>
</tbody>
</table>

**Map Representation**

$y_i = M S'_1 b_i + M S'_0 b'_i$

**Logic Schematic**
# Logic Extender

<table>
<thead>
<tr>
<th>$M$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>Function Name</th>
<th>$F$</th>
<th>$X$</th>
<th>$Y$</th>
<th>$c_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Complement</td>
<td>$A'$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>AND</td>
<td>$A$ AND $B$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Identity</td>
<td>$A$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>OR</td>
<td>$A$ OR $B$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

## Truth Table

<table>
<thead>
<tr>
<th>$M$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$a'_i$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$a_ib_i$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$a_i$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$a_i + b_i$</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>$a_i$</td>
</tr>
</tbody>
</table>

## Functional Table

### $M = 0$

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$x_i, y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1, 1, 3, 2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>5, 1, 6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>12, 13, 14</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>8, 9, 11, 10</td>
</tr>
</tbody>
</table>

### $M = 1$

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$x_i, y_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>16, 17, 19, 18</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>26, 21, 25</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>24, 25, 27, 28</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1, 1, 1, 1</td>
</tr>
</tbody>
</table>

## Map Representation

$$x_i = M'S_1S_0a' + M'S_1S_0b + S_0a + S_1a_i + M a_i$$

## Logic Schematic

$S_0$ $S_1$ $M$ $x_i$ $a_i$ $b_i$
Final ALU Schematic

4-bit Arithmetic Logic Unit Schematic
1-to-2 Decoder

- Decoders (Demultiplexers) are used for enabling components

**Truth Table**

<table>
<thead>
<tr>
<th>$E$</th>
<th>$A_0$</th>
<th>$C_1$</th>
<th>$C_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Boolean Expression**

\[
C_0 = EA'_0 \\
C_1 = EA_0
\]

**Graphic Symbol**

**Logic Schematic**
2-to-4 Decoder

Diagram and Truth Table:

**Boolean Expression:**
- $C_0 = E_0 A'_1 A'_0$
- $C_1 = E_0 A'_1 A_0$
- $C_2 = E_0 A_1 A'_0$
- $C_2 = E_0 A_1 A_0$

**Truth Table:**

<table>
<thead>
<tr>
<th>$E$</th>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$C_3$</th>
<th>$C_2$</th>
<th>$C_1$</th>
<th>$C_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</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>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>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Logic Schematic**
**3-to-8 Decoder**

- Larger decoders can be built as a tree of smaller decoders

<table>
<thead>
<tr>
<th>$E$</th>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$C_7$</th>
<th>$C_6$</th>
<th>$C_5$</th>
<th>$C_4$</th>
<th>$C_3$</th>
<th>$C_2$</th>
<th>$C_1$</th>
<th>$C_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<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>0</td>
<td>0</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>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<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>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Implementation with 1-to-2 Decoders

Implementation with 2-to-4 Decoders

Truth Table

Decoder

Graphic Symbol
2-to-1 Selector

- Selectors (Multiplexers) are used for selecting one of many sources of data

$$Y = S'D_0 + S'D_1$$

Graphic Symbol

Truth Table

Logic Schematic
4-to-1 Selector

Truth Table

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$D_0$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$D_1$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$D_2$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$D_3$</td>
</tr>
</tbody>
</table>

Boolean Expression

$$Y = S'_1 S'_0 D_0 + S'_1 S_0 D_1 + S_1 S'_0 D_2 + S_1 S_0 D_3$$

Logic Schematic
8-to-1 Selector

Implementation with 2-to-1 Selectors

Truth Table

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$D_0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$D_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$D_2$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$D_3$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$D_4$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$D_5$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$D_6$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$D_7$</td>
</tr>
</tbody>
</table>

Implementation with 3-to-8 Decoder

Copyright © 2004-2005 by Daniel D. Gajski
- Bus drivers have three possible output values:
  0, 1, and Z (High Impedance ≈ Disconnection)
2-to-1 Priority Encoder

- Encoder is opposite of decoder, but with priority for MSB

```
A_0 = D_1
Any = D_0 + D_1
```

Truth Table

<table>
<thead>
<tr>
<th>D_1</th>
<th>D_0</th>
<th>A_0</th>
<th>Any</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>1</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Logic Schematic
4-to-2 Priority Encoder

**Graphic Symbol**

\[ A_0 = D'_2 D_1 + D_3 \]
\[ A_1 = D_2 + D_3 \]
\[ \text{Any} = D_0 + D_1 + D_2 + D_3 \]

**Boolean Expression**

**Truth Table**

<table>
<thead>
<tr>
<th>( D_3 )</th>
<th>( D_2 )</th>
<th>( D_1 )</th>
<th>( D_0 )</th>
<th>( A_1 )</th>
<th>( A_0 )</th>
<th>( \text{Any} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</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>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Logic Schematic**
8-to-3 Priority Encoder

- Larger encoders can be built from smaller encoders and selectors

<table>
<thead>
<tr>
<th>D_7</th>
<th>D_6</th>
<th>D_5</th>
<th>D_4</th>
<th>D_3</th>
<th>D_2</th>
<th>D_1</th>
<th>D_0</th>
<th>A_2</th>
<th>A_1</th>
<th>A_0</th>
<th>Any</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</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>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
</tr>
<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>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Truth Table

Implementation with 2-to-1 Encoders and Selectors

Implementation with 4-to-2 Encoder
2-bit Magnitude Comparator

- \( G = 1 \) when \( X > Y \),
- \( L = 1 \) when \( X < Y \),
- \( G = L = 0 \) when \( X = Y \).

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_0 )</th>
<th>( y_1 )</th>
<th>( y_0 )</th>
<th>( G )</th>
<th>( L )</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>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>0</td>
<td>1</td>
<td>1</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>0</td>
<td>1</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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<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>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\( G = x_1y'_1 + x_0y_0' + x_1x_0y_0' \)

\( L = x'_1y'_1 + x'_0x'd_0 + x_0y'_1y_0 \)

*Truth Table*

*Logic Schematic*
8-bit Magnitude Comparator

- Larger magnitude comparators can be constructed from basic 2-bit comparators using the following equations:

\[ G_i = (x_i > y_i) \text{ OR } ((x_i = y_i) \text{ AND } (G_{i-1} > L_{i-1})) \]

\[ L_i = (x_i < y_i) \text{ OR } ((x_i = y_i) \text{ AND } (G_{i-1} < L_{i-1})) \]
8-bit Shifter

- 1-bit left or right shift or rotation

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$Y$</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>X</td>
<td>$D$</td>
<td>No Shift</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>Not Used</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$\text{shl}(D)$</td>
<td>Shift Left</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$\text{rtl}(D)$</td>
<td>Rotate Left</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\text{shr}(D)$</td>
<td>Shift Right</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$\text{rtr}(D)$</td>
<td>Rotate Right</td>
</tr>
</tbody>
</table>

Shifter Implemented with 4-to-1 Selectors
8-bit Barrel Right Rotator

- Right rotation from 0 to 7 bit position

<table>
<thead>
<tr>
<th>$S_2$</th>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$y_7$</th>
<th>$y_6$</th>
<th>$y_5$</th>
<th>$y_4$</th>
<th>$y_3$</th>
<th>$y_2$</th>
<th>$y_1$</th>
<th>$y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$d_7$</td>
<td>$d_6$</td>
<td>$d_5$</td>
<td>$d_4$</td>
<td>$d_3$</td>
<td>$d_2$</td>
<td>$d_1$</td>
<td>$d_0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>$d_0$</td>
<td>$d_7$</td>
<td>$d_6$</td>
<td>$d_5$</td>
<td>$d_4$</td>
<td>$d_3$</td>
<td>$d_2$</td>
<td>$d_1$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$d_1$</td>
<td>$d_0$</td>
<td>$d_7$</td>
<td>$d_6$</td>
<td>$d_5$</td>
<td>$d_4$</td>
<td>$d_3$</td>
<td>$d_2$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>$d_2$</td>
<td>$d_1$</td>
<td>$d_0$</td>
<td>$d_7$</td>
<td>$d_6$</td>
<td>$d_5$</td>
<td>$d_4$</td>
<td>$d_3$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$d_3$</td>
<td>$d_2$</td>
<td>$d_1$</td>
<td>$d_0$</td>
<td>$d_7$</td>
<td>$d_6$</td>
<td>$d_5$</td>
<td>$d_4$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$d_4$</td>
<td>$d_3$</td>
<td>$d_2$</td>
<td>$d_1$</td>
<td>$d_0$</td>
<td>$d_7$</td>
<td>$d_6$</td>
<td>$d_5$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$d_5$</td>
<td>$d_4$</td>
<td>$d_3$</td>
<td>$d_2$</td>
<td>$d_1$</td>
<td>$d_0$</td>
<td>$d_7$</td>
<td>$d_6$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>$d_6$</td>
<td>$d_5$</td>
<td>$d_4$</td>
<td>$d_3$</td>
<td>$d_2$</td>
<td>$d_1$</td>
<td>$d_0$</td>
<td>$d_7$</td>
</tr>
</tbody>
</table>

Truth Table

Rotator Implemented with 2-to-1 Selectors
Read-Only Memory (ROM)

- ROMs can be used for implementation of SOP forms

<table>
<thead>
<tr>
<th>Computational Symbols</th>
<th>Programmable Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A B C D</td>
</tr>
</tbody>
</table>

Truth Table

OR Array

16 × 4 ROM

F₀ F₁ F₂ F₃
Adder/Subtractor Bit-slice Using a 16×4 ROM

Truth Table

<table>
<thead>
<tr>
<th>$A_3$ ($S$)</th>
<th>$A_2$ ($a_i$)</th>
<th>$A_1$ ($b_i$)</th>
<th>$A_0$ ($c_i$)</th>
<th>$F_3$</th>
<th>$F_2$</th>
<th>$F_1$</th>
<th>$F_0$ ($c_{i+1}$)</th>
<th>$s_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

OR Array

16 × 4 ROM
Programmable Logic Arrays (PLA)

- Contrary to ROMs, PLA’s have a programmable decoder
- In comparison to ROMs, PLA’s can implement Boolean functions with more variables, but less terms
**Full-Adder with a PLA**

![Diagram](image)

### Truth Table

<table>
<thead>
<tr>
<th>$A_3$</th>
<th>$A_2$</th>
<th>$A_1$</th>
<th>$A_0$</th>
<th>$F_3$</th>
<th>$F_2$</th>
<th>$F_1$</th>
<th>$F_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Map Representation

$s_i = x_i \oplus y_i \oplus c_i$

$e_{i+1} = x_i y_i + c_i (x_i \oplus y_i)$

### PLA Implementation

- **OR Array**
- **AND Array**
- **Output Array**

---

Copyright © 2004-2005 by Daniel D. Gajski

Slides by Philip Pham, University of California, Irvine
Chapter Summary

- Described basic combinatorial components:
  - Adders
  - Subtractors
  - ADLUs
  - Logic Units
  - Decoders
  - Selectors
  - Buses
  - Encoders
  - Comparators
  - Shifters
  - ROMs
  - PLAs

- Presented design for combinatorial components

- Discussed procedures for building larger components from smaller ones