M. Morris Mano and Michael D. Ciletti
Prentice-Hall, Inc.

E-mail: kevinsu@yuntech.edu.tw
Chapter 7

Memory and Programmable Logic
Outline of Chapter 7

7.1  Introduction
7.2  Random-Access Memory
7.3  Memory Decoding
7.4  Error Detection and Correction
7.5  Read-Only Memory
7.6  Programmable Logic Array
7.7  Programmable Array Logic
7.8  Sequential Programmable Devices
7.1 Introduction
7.2 Random-Access Memory
7.3 Memory Decoding
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
7.1 Introduction

Terms

1. Memory Unit: A memory unit is a collection of cells capable of storing a large quantity of binary information.

2. Types of Memory: Random-Access Memory (RAM), Read-Only Memory (ROM)

3. Write: The process of storing new information into memory is referred to as a memory write operation.

4. Read: The process of transferring the stored information out of memory is referred to as a memory read operation.

5. RAM can perform both the write and read operations.

6. ROM can perform only the read operation.

7. ROM is a programmable logic device.
7.1 Introduction

- Two Symbols for OR Gates

Conventional Symbol = Array Logic Symbol
7.2 Random-Access Memory

7.1 Introduction
7.2 Random-Access Memory
7.3 Memory Decoding
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
Random-Access Memory

1. RAM: The *transfer time* of memory units to or from any desired *random location is always the same*, hence, the name random-access memory abbreviated RAM.

2. Word: A memory unit stores binary information in groups of bits called words.

3. Byte: A group of eight bits is called a byte.

4. The *capacity of a memory* is usually stated as the total *number of bytes* that is store.
7.2 Random-Access Memory

- Block Diagram of a Memory Unit

Memory Unit:
- $2^k$ Words
- $n$ Bits Per Word
- $k$ address lines
- Read
- Write
- $n$ Data Input Lines
- $n$ Data Output Lines
7.2 Random-Access Memory

- **Memory Size**
  1. K (Kilo): $2^{10}$
  2. M (Mega): $2^{20}$
  3. G (Giga): $2^{30}$
# 7.2 Random-Access Memory

## Memory with a Capacity of 1K Words of 16 Bits

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Binary</th>
<th>Decimal</th>
<th>Memory Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000000000 0</td>
<td>1001100101010001</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0000000001 1</td>
<td>1001110100001000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0000000010 2</td>
<td>1110000101010001</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11111111101</td>
<td>1021</td>
<td>0100110011001000</td>
<td></td>
</tr>
<tr>
<td>11111111110</td>
<td>1022</td>
<td>0001111101010001</td>
<td></td>
</tr>
<tr>
<td>11111111111</td>
<td>1023</td>
<td>0001100101010011</td>
<td></td>
</tr>
</tbody>
</table>
7.2 Random-Access Memory

- Memory Write Process

1. Apply binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the input data lines.
3. Active the write input.
7.2 Random-Access Memory

- **Memory Read Process**

1. Apply binary address of the desired word to the address lines
2. Active the read input.
3. The memory will take the bits from the word that has been selected by the address and apply them to the output data bus.
## Memory Control

<table>
<thead>
<tr>
<th>Memory Enable (Chip Select)</th>
<th>Read/Write</th>
<th>Memory Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Write to selected word</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Read from selected word</td>
</tr>
</tbody>
</table>
7.2 Random-Access Memory

Memory Description in HDL: Array of Register

// Read and write operations of memory.
// Memory size is 64 words of 4 bit each

module memory (Enable, ReadWrite, Address, DataIn, DataOut);
    Input Enable, ReadWrite;
    Input [3:0] DataIn;
    Input [5:0] Address;
    Output [3:0] DataOut;
    reg [3:0] DataOut;
    reg [3:0] Mem [0:63]; // 64x4 memory

always @ (Enable or ReadWrite)
    if (Enable)
        if (ReadWrite)
            DataOut = Mem[Address]; // Read
        else
            Mem[Address] = DataIn; // Write
    else
        DataOut = 4’bz; // High Impedance State
endmodule
7.2 Random-Access Memory

- **Read Operation Symbolic Statement**
  
  \[ \text{DataOut} \leftarrow \text{Mem} \ [\text{Address}] \]

- **Write Operation Symbolic Statement**
  
  \[ \text{Mem} \ [\text{Address}] \leftarrow \text{DataIn} \]
7.2 Random-Access Memory

Relation to CPU and Memory

- **CPU**
- **MEM**

- **Address Bus**
- **Data Bus**
- **Chip Select**
- **Read/Write**
- **Control Signal**
- **Clock**
7.2 Random-Access Memory

Memory Write Cycle Timing Waveforms

50Mhz CPU Clock

- $T_1$
- $T_2$
- $T_3$
- $T_1$

Address Valid

Memory Address

Memory Enable

Read/Write

Data Input

Data Valid

50 nsec (Cycle Time)
Memory Read Cycle Timing Waveforms

50Mhz CPU Clock

- $T_1$
- $T_2$
- $T_3$
- $T_1$

Memory Address

Address Valid

Memory Enable

Read/Write

Data Output

50 nsec (Access Time)
7.2 Random-Access Memory

- **Types of Memories**

**Access Sequence**

- Random Access: RAM
- Sequential Access: Magnetic Disk/ Magnetic Tape

**RAM**

- Static RAM (SRAM/Cache): Latch Structure
- Dynamic RAM (DRAM): Electric Charge on Capacitor

**DRAM**

- The electric charge is implemented by MOS transistor
- Refresh: The stored charge on the capacitors trends to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory

**DRAM:** Low Power Consumption & Large Storage Capacitor

**SRAM:** Easy to Use & Shorter Read/Write Cycle
7.2 Random-Access Memory

Memory Terms

1. Volatile: Memory units that lose stored information when power is turned off.
2. Volatile Memory: DRAM, SRAM
3. Nonvolatile Memory: Magnetic Disk, ROM
4. SoC stores the program and data in ROM.
5. When power is turned on, SoC can use the programs from ROM to RAM for data execution.
7.1 Introduction
7.2 Random-Access Memory
**7.3 Memory Decoding**
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
7.3 Memory Decoding

The storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address.
7.3 Memory Decoding

**Memory Cell**

- **Logic Diagram**
  - Input: 1/0
  - 0/1
  - Select
  - Output
  - Read/Write
  - 4~6 Transistors

- **Block Diagram**
  - Input
  - BC
  - Read/Write
  - Output
7.3 Memory Decoding

- **Memory: 4 Words of 4 Bit Each**

```
+-----------------+-----------------+-----------------+-----------------+
|      |      |      |      |
|      |      |      |      |
|      |      |      |      |
+-----------------+-----------------+-----------------+-----------------+
```

- **Input Data**
- **Address Input**
- **Memory Enable**
- **Read/Wrire**
- **Output Data**
7.3 Memory Decoding

- **Coincident Decoding**
  
  $k$ to $2^k$ decoder requires $2^k$ AND gate with $k$ inputs per gate

![Diagram of 5X32 Decoder](image)

- 5X32 Decoder
- Binary Address: 01100 10100
- Select Word: 404
- 1024 Words
- 20 12

Select Word 404

Binary Address

01100 10100

X Y
### 7.3 Memory Decoding

#### DRAM vs. SRAM

1. SRAM Cell: 6 Transistors
2. DRAM Cell: 1 Transistor
3. Cell Density: DRAM vs. SRAM = 4:1
4. Cell Cost: SRAM vs. DRAM = 3 or 4 : 1
5. DRAM: Low Power Consumption
6. DRAM Word Size: 1 Byte
7.3 Memory Decoding

- **Address Multiplexing for a 64K DRAM**

1. Reduce the I/O pin count of RAM IC package
2. Row Address First/ Column Address Second
3. Multiplexing access the RAS and CAS signals
4. RAS = Row Address Strobe
5. CAS = Column Address Strobe
7.3 Memory Decoding

Address Multiplexing for a 64K DRAM (Continued)
7.4 Error Detection and Correction
7.4 Error Detection and Correction

- **Error Detection**

1. Error may cause in storing and retrieving data.
2. The most common error-detection scheme is the parity bit (Ref 3.8).
3. The parity of the word is checked after reading it from memory.
4. The data word is accepted if the parity of the bits read out is correct.
5. Parity check can detect errors, but cannot correct errors.
7.4 Error Detection and Correction

Error Correction

1. An error-correction code generates multiple parity check bits that are stored with the data word in memory.
2. If the check bits are correct, it signifies that no error has occurred.
3. If the check bits do not compare with the stored parity, they generate a unique pattern, called a syndrome, that can be used to identify the bit in error.
4. A single error occurs when a bit changes from 1 to 0 or 0 to 1.
Hamming Code: An Error Correction Code

- **Hamming Code**
- An Error Correction Code

- **n-bit Source Data**
- **Generate k Parity Bits**

- **Hamming Coding**
- **Generate k Parity Bits**

- **Insert**
7.4 Error Detection and Correction

Hamming Code Example: 8-Bit Source Data

8-Bit Source Data (n=8) : 11000100

4 Parity Bits (k=4)

New Data : 12 Bits (n+k=12)

Bit Position : 1 2 3 4 5 6 7 8 9 10 11 12

P_1 P_2 1 P_4 1 0 0 P_8 0 1 0 0
7.4 Error Detection and Correction

- **Hamming Code Example** (Continued): Parity Bit Generation

\[
\begin{align*}
P_1 &= \text{XOR of Bits in Position (3, 5, 7, 9, 11)} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0 \\
P_2 &= \text{XOR of Bits in Position (3, 6, 7, 10, 11)} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0 \\
P_4 &= \text{XOR of Bits in Position (5, 6, 7, 12)} = 1 \oplus 0 \oplus 0 \oplus 0 = 1 \\
P_8 &= \text{XOR of Bits in Position (9, 10, 11, 12)} = 0 \oplus 1 \oplus 0 \oplus 0 = 1
\end{align*}
\]

Substitute P Bits into the 8-Bit Source Data:

Position: 1 2 3 4 5 6 7 8 9 10 11 12
0 0 1 1 1 0 0 1 0 1 0 0 (Stored to Memory)
Hamming Code Example (Continued): Check Parity Bits

- $C_1 = \text{XOR of Bits in Position } (1, 3, 5, 7, 9, 11)$
- $C_2 = \text{XOR of Bits in Position } (2, 3, 6, 7, 10, 11)$
- $C_4 = \text{XOR of Bits in Position } (4, 5, 6, 7, 12)$
- $C_8 = \text{XOR of Bits in Position } (8, 9, 10, 11, 12)$

Error Free: $C = C_1 \, C_2 \, C_4 \, C_8 = 0 \, 0 \, 0 \, 0 = \text{Even Parity}$
### Hamming Code Example: Error Cases

**Position:**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</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>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- **No Error**
- **Error in Bit 1**
- **Error in Bit 5**

**Syndrome**:

<table>
<thead>
<tr>
<th>Syndrome</th>
<th>$C_8$</th>
<th>$C_4$</th>
<th>$C_2$</th>
<th>$C_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Error</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Error in Bit 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Error in Bit 5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- $0 0 0 0 1 = 1$ (Complement Bit 1)
- $0 1 0 1 1 = 5$ (Complement Bit 5)
7.4 Error Detection and Correction

Range of $n$ Data Bits for $k$ Check Bits in Hamming Code

- Hamming Code = $n$-Bit Data + $k$-Bit Parity
- Syndrome value $C$ consists of $k$ bits and checks data range from bit 1 to bit $2^n$ (Number of total checked bit $2^{k-1}$)
- The $2^{k-1}$ must be equal to or larger than $n+k$
- Therefore, $2^{k-1} \geq n+k$
- Solving for $n$ in terms of $k$, we obtain $2^{k-1}-k \geq n$
- The table of the relation to $n$ and $k$ are:

<table>
<thead>
<tr>
<th>Range of Data Bits, $n$</th>
<th>Number of Check Bits, $k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-4</td>
<td>3</td>
</tr>
<tr>
<td>5-11</td>
<td>4</td>
</tr>
<tr>
<td>12-26</td>
<td>5</td>
</tr>
<tr>
<td>27-57</td>
<td>6</td>
</tr>
<tr>
<td>58-120</td>
<td>7</td>
</tr>
</tbody>
</table>
## Syndrome vs. Check Position

<table>
<thead>
<tr>
<th>Position</th>
<th>Binary</th>
<th>$C_4$</th>
<th>$C_2$</th>
<th>$C_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P_1$</td>
<td>0 0 1</td>
<td></td>
<td></td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0 1 0</td>
<td></td>
<td></td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$P_3$</td>
<td>0 1 1</td>
<td></td>
<td></td>
<td>$\checkmark$ $\checkmark$</td>
</tr>
<tr>
<td>$P_4$</td>
<td>1 0 0</td>
<td></td>
<td></td>
<td>$\checkmark$</td>
</tr>
<tr>
<td>$P_5$</td>
<td>1 0 1</td>
<td></td>
<td></td>
<td>$\checkmark$ $\checkmark$</td>
</tr>
<tr>
<td>$P_6$</td>
<td>1 1 0</td>
<td></td>
<td></td>
<td>$\checkmark$ $\checkmark$</td>
</tr>
<tr>
<td>$P_7$</td>
<td>1 1 1</td>
<td></td>
<td></td>
<td>$\checkmark$ $\checkmark$ $\checkmark$</td>
</tr>
</tbody>
</table>
### 7.4 Error Detection and Correction

#### Single-Error Correction, Double-Error Detection

Position: 1 2 3 4 5 6 7 8 9 10 11 12 13

0 0 1 1 1 0 0 1 0 1 0 0 1 (Even Parity)

\[ P_{13} = \text{XOR} ( P_1, \ldots, P_{12} ) \]

\[ P = \text{XOR} ( P_1, \ldots, P_{13} ) = 0 \text{, Parity is correct (Even Parity)} \]

\[ P = \text{XOR} ( P_1, \ldots, P_{13} ) = 1 \text{, Parity is incorrect (Odd Parity)} \]

**4 Cases in Hamming Code**

<table>
<thead>
<tr>
<th>C</th>
<th>P</th>
<th>Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>=0</td>
<td>=0</td>
<td>No Error</td>
</tr>
<tr>
<td>≠0</td>
<td>=1</td>
<td>Single Error Correction</td>
</tr>
<tr>
<td>≠0</td>
<td>=0</td>
<td>Double Error Detection</td>
</tr>
<tr>
<td>=0</td>
<td>=1</td>
<td>Single Error in ( P_{13} ) Correction</td>
</tr>
</tbody>
</table>
7.1 Introduction
7.2 Random-Access Memory
7.3 Memory Decoding
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
ROM Block Diagram

\[ 2^k \times n \text{ ROM} \]
(\(2^k\) words, \(n\) bits per word)

- \(k\) Inputs (Address Bus)
- \(n\) Outputs (Data Bus)
Internal Logic of a 32X8 ROM

5 X 32 Decoder

32 Input OR Gate
## 7.5 Read-Only Memory

### Example for Programming ROM

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_4$ $I_3$ $I_2$ $I_1$ $I_0$</td>
<td>$A_7$ $A_6$ $A_5$ $A_4$ $A_3$ $A_2$ $A_1$ $A_0$</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>1 0 1 1 0 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>0 0 0 1 1 1 0 1</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>1 1 0 0 0 1 0 1</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>1 0 1 1 0 1 0 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>0 0 0 0 1 0 0 1</td>
</tr>
<tr>
<td>1 1 1 0 1</td>
<td>1 1 1 0 0 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 0</td>
<td>0 1 0 0 1 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>0 0 1 1 0 0 1 1</td>
</tr>
</tbody>
</table>

- **$X$: Fuse Connection**

### 5 x 32 Decoder

- **Inputs:** $I_5, I_4, I_3, I_2, I_1, I_0$
- **Outputs:** $A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0$
7.5 Read-Only Memory

Combinational Circuit Implementation

<table>
<thead>
<tr>
<th>Inputs $I_4 I_3 I_2 I_1 I_0$</th>
<th>Outputs $A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>1 0 1 1 0 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>0 0 0 1 1 1 0 1</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>1 1 0 0 0 1 0 1</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>1 0 1 1 0 0 1 0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>0 0 0 0 1 0 0 1</td>
</tr>
<tr>
<td>1 1 1 0 1</td>
<td>1 1 1 0 0 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 0</td>
<td>0 1 0 0 1 0 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>0 0 1 1 0 0 1 1</td>
</tr>
</tbody>
</table>

$A_7 (I_4, I_3, I_2, I_1, I_0) = \Sigma (0, 2, 3, \ldots, 29)$
Example for Combinational Circuit Implementation Using a ROM

Question: Design a combinational circuit using a ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of input number.

<table>
<thead>
<tr>
<th>Inputs $A_2 A_1 A_0$</th>
<th>Outputs $B_5 B_4 B_3 B_2 B_1 B_0$</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0 0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0 1 0 0</td>
<td>4</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 1 0 0 1</td>
<td>9</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1 0 0 0 0</td>
<td>16</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1 1 0 0 1</td>
<td>25</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0 0 1 0 0</td>
<td>36</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 0 0 0 1</td>
<td>49</td>
</tr>
</tbody>
</table>
Example for Combinational Circuit Implementation Using a ROM (Continued)

### ROM Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2 A_1 A_0$</td>
<td>$B_5 B_4 B_3 B_2 B_1 B_0$</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0 1 0 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 1 0 0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0 0 1 0 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 0 0 0 1 1</td>
</tr>
</tbody>
</table>

Equal
7.5 Read-Only Memory

- Types of ROMs

1. The required path in a ROM may be programmed in 4 different ways.
2. The first is called mask programming (mask ROM) and is done by the semiconductor company during the last fabrication process of the unit.
3. For small quantities, the second type of ROM is called programmable read-only memory or PROM.
4. The programming ROM and PROM is irreversible.
5. The third type of ROM is the erasable PROM or EPROM.
6. The fourth type of ROM is the electrically-erasable PROM (EEPROM or E²RPOM)
7.5 Read-Only Memory

Combinational PLDs (Programmable Logic Device)

- Programmable Read-Only Memory (PROM)
  - Fixed AND Array (Decoder)
  - Programmable OR Array

- Programmable Array Logic (PAL)
  - Programmable AND Array
  - Fixed OR Array

- Programmable Logic Array (PLA)
  - Programmable AND Array
  - Programmable OR Array
7.1 Introduction
7.2 Random-Access Memory
7.3 Memory Decoding
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
7.6 Programmable Logic Array

- **PLA**

**Programmable Read-Only Memory (PROM)**
- Fixed AND Array (Decoder)
- Programmable OR Array

**Programmable Array Logic (PAL)**
- Programmable AND Array
- Fixed OR Array

**Programmable Logic Array (PLA)**
- Programmable AND Array
- Programmable OR Array
7.6 Programmable Logic Array

- PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

\[ F_1 = AB' + AC + A'BC' \]
\[ F_2 = (AC + BC)' \]

Simplification: 
- \( AB' \)
- \( AC \)
- \( BC \)
- \( A'BC' \)
### PLA Programming Table

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A  B  C</td>
<td>(T) (C)</td>
</tr>
<tr>
<td>AB'</td>
<td>1 0 -</td>
<td>1 -</td>
</tr>
<tr>
<td>AC</td>
<td>1 - 1</td>
<td>1 1</td>
</tr>
<tr>
<td>BC</td>
<td>- 1 1</td>
<td>- 1</td>
</tr>
<tr>
<td>A'BC'</td>
<td>0 1 0</td>
<td>1 -</td>
</tr>
</tbody>
</table>

1: True Output  
0: Complement Output  
- : Variable Absent
7.6 Programmable Logic Array

**Size of PLA**

- **n Inputs**: \( I_1, \ldots, I_n \)
- **k AND**: (Dashed box)
- **k Product Terms**: \( F_1, \ldots, F_m \)
- **2n x k Connections**: (Red lines and nodes)
- **m OR**: (Red lines and nodes)
- **m XOR**: (Red lines and nodes)
- **m Outputs**: \( F_1, \ldots, F_m \)

Diagram illustrates the structure and connections within a Programmable Logic Array (PLA).
Implementation a Boolean Function with a PLA

\[ F_1(A,B,C) = \Sigma(0,1,2,4) \]
\[ F_2(A,B,C) = \Sigma(0,5,6,7) \]

\[ F_1 = A'B' + A'C' + B'C' \]
\[ F_1 = (AB + AC + BC)' \]

\[ F_2 = AB + AC + A'B'C' \]
\[ F_2 = (A'C' + A'B + AB'C')' \]
## Implementation a Boolean Function with a PLA (Continued)

### PLA Programming Table

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A  B  C</td>
<td>F₁</td>
</tr>
<tr>
<td>AB</td>
<td>1 1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>AC</td>
<td>1 - 1</td>
<td>1 1</td>
</tr>
<tr>
<td>BC</td>
<td>- 1 1</td>
<td>1 -</td>
</tr>
<tr>
<td>A'B'C'</td>
<td>0 0 0</td>
<td>- 1</td>
</tr>
</tbody>
</table>

(Continued)
7.1 Introduction
7.2 Random-Access Memory
7.3 Memory Decoding
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
7.7 Programmable Array Logic

- **PAL**

  Fixed AND Array (Decoder) \(\rightarrow\) Programmable OR Array \(\rightarrow\) Outputs

  Programmable Read-Only Memory (PROM)

  Programmable AND Array \(\rightarrow\) Fixed OR Array \(\rightarrow\) Outputs

  Programmable Array Logic (PAL)

  Programmable AND Array \(\rightarrow\) Programmable OR Array \(\rightarrow\) Outputs

  Programmable Logic Array (PLA)
7.7 Programmable Array Logic

- PAL with 4 Inputs, 4 Outputs, and 3-Wire AND-OR Structure
### 7.7 Programmable Array Logic

**PAL Example**

\[
\begin{align*}
w(A,B,C,D) &= \Sigma(2,12,13) \\
x(A,B,C,D) &= \Sigma(7,8,9,10,11,12,13,14,15) \\
y(A,B,C,D) &= \Sigma(0,2,3,4,5,6,7,8,10,11,15) \\
z(A,B,C,D) &= \Sigma(1,2,8,12,14)
\end{align*}
\]

\[
\begin{align*}
w(A,B,C,D) &= ABC' + A'B'CD' \\
x &= A + BCD \\
y &= A'B + CD + B'D' \\
z &= ABC' + A'B'CD' + AC'D' + A'B'C'D' \\
&= w' + AC'D' + A'B'C'D
\end{align*}
\]

#### PAL Programming Table

<table>
<thead>
<tr>
<th>Product Terms</th>
<th>AND Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1 1 0 -</td>
<td>w = ABC'</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0 -</td>
<td>+A'B'CD'</td>
</tr>
<tr>
<td>3</td>
<td>- - - -</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1 - - -</td>
<td>x = A</td>
</tr>
<tr>
<td>5</td>
<td>- 1 1 1 -</td>
<td>+BCD</td>
</tr>
<tr>
<td>6</td>
<td>- - - -</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 1 - -</td>
<td>y = A'B</td>
</tr>
<tr>
<td>8</td>
<td>- - 1 1 -</td>
<td>+CD</td>
</tr>
<tr>
<td>9</td>
<td>- 0 0 -</td>
<td>+B'D'</td>
</tr>
<tr>
<td>10</td>
<td>- - - - -</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1 - 0 0 -</td>
<td>+ AC'D'</td>
</tr>
<tr>
<td>12</td>
<td>0 0 0 1 -</td>
<td>+ A'B'C'D</td>
</tr>
</tbody>
</table>
7.7 Programmable Array Logic

PAL Example (Continued)
7.8 Sequential Programmable Devices

7.1 Introduction
7.2 Random-Access Memory
7.3 Memory Decoding
7.4 Error Detection and Correction
7.5 Read-Only Memory
7.6 Programmable Logic Array
7.7 Programmable Array Logic
7.8 Sequential Programmable Devices
Three Types of Sequential Programmable Devices

1. Sequential (or Simple) Programmable Logic Device (SPLD)
2. Complex Programmable Logic Device (CPLD)
3. Field Programmable Gate Array (FPGA)
7.8 Sequential Programmable Devices

- Sequential (or Simple) Programmable Logic Devices

![Diagram showing the architecture of a sequential programmable device including inputs, AND-OR Array (PAL or PLA), Flip-Flops, and outputs.]

- Inputs → AND-OR Array (PAL or PLA) → Flip-Flops → Outputs
Basic Macrocell Logic: One Section of a SPLD

- Input
- Output
- 3-State Buffer
- 3-State Inverter

CLK OE

Output Enable

3-State Buffer or 3-State Inverter

Output
7.8 Sequential Programmable Devices

- Complex PLD (CPLD)

**MC: Macrocell**

- Programmable Switch Matrix

- I/O Block

- (S)PLD 6~8 MC

- I/O Block

- (S)PLD 6~8 MC

- (S)PLD 6~8 MC

- (S)PLD 6~8 MC