Fig. 7-1 Conventional and Array Logic Diagrams for OR Gate
Fig. 7-2 Block Diagram of a Memory Unit
<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</td>
<td>0</td>
<td>1011010101011101</td>
</tr>
<tr>
<td></td>
<td>0000000001</td>
<td>1</td>
<td>1010101110001001</td>
</tr>
<tr>
<td></td>
<td>0000000010</td>
<td>2</td>
<td>0000110101000110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1111111101</td>
<td>1021</td>
<td>1001110100010100</td>
</tr>
<tr>
<td></td>
<td>1111111110</td>
<td>1022</td>
<td>0000110100011110</td>
</tr>
<tr>
<td></td>
<td>1111111111</td>
<td>1023</td>
<td>1101111000100101</td>
</tr>
</tbody>
</table>

Fig. 7-3  Content of a 1024 × 16 Memory
Steps to Write into RAM

• Apply address to address lines
• Apply data to data input lines
• Activate *write* input & enable chip

For reads: do 1 and 3 using *read* input
Fig. 7-4 Memory Cycle Timing Waveforms
Fig. 7-5 Memory Cell
Fig. 7-6 Diagram of a $4 \times 4$ RAM
Row/Column decoding

• 1K-word memory requires 10 address bits and a $10 \times 1024$ decoder

• The decoding can also be done with two $5 \times 32$ decoder, one for the row and one for the column. The cell connected to the row-column intersection is selected.
Fig. 7-7 Two-Dimensional Decoding Structure for a 1K-Word Memory
Fig. 7-8 Address Multiplexing for a 64K DRAM
Fig. 7-9  ROM Block Diagram
Used as programmable logic, a PROM stores the truth table for $N$ functions of $M$ inputs. $N =$ number of bits in each cell. $M =$ number of address bits; there are $2^M$ memory locations in the PROM.
“x” indicates a connection, and a “1” in the truth table

Address 00000: cell contents is 10110110

Fig. 7-11 Programming the ROM According to Table 7-3
Design a combinatorial circuit using a ROM. The circuit accepts a 3-bit input number and outputs a binary number equal to the square of the input.

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$B_5$</td>
<td>$B_4$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$A_1$</td>
<td>$A_0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Fig. 7-12  ROM Implementation of Example 7-1
Fig. 7-13 Basic Configuration of Three PLDs
"x" indicates a connection

Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs
Example: Implement the following boolean functions in a PLA:

\[ F_1(A,B,C) = \Sigma(0,1,2,4) \]
\[ F_2(A,B,C) = \Sigma(0,5,6,7) \]
\( \times \) = connection
\( \oplus \) = no connection
F_1 = \overline{A' B'} + \overline{A' C'} + \overline{B' C'}
F_1 = (A B + A C + B C)'

F_2 = A B + A C + A' B' C'
F_2 = (A' C + A' B + A B' C')'

PLA programming table

<table>
<thead>
<tr>
<th>Product term</th>
<th>Inputs ( A \ B \ C )</th>
<th>Outputs ( (C) \ (T) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A B )</td>
<td>1 1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>( A C )</td>
<td>2 - 1</td>
<td>1 1</td>
</tr>
<tr>
<td>( B C )</td>
<td>3 - 1 1</td>
<td>1 -</td>
</tr>
<tr>
<td>( A' B' C' )</td>
<td>4 0 0 0</td>
<td>- 1</td>
</tr>
</tbody>
</table>

Fig. 7-15 Solution to Example 7-2
7-21 Derive the PLA programming table for the combinational circuit that squares a 3-bit number. Minimize the number of product terms. (See Fig. 7-12 for the equivalent ROM implementation.)

![Block diagram](image)

![ROM truth table](image)

Fig. 7-12 ROM Implementation of Example 7-1
$X = \text{connection}$
$+ = \text{no connection}$
3. (25 pts) Determine the PLA programming table needed to implement the following two boolean functions. Minimize the number of product terms. Show all your work, including the Karnaugh maps used in the minimization.

\[ F_1 (A,B,C,D) = \sum(1, 3, 4, 5, 7, 13, 15) \]
\[ F_2 (A,B,C,D) = \sum(0, 2, 3, 6, 7, 8, 10, 11, 12, 14) \]

Write your result in the following table.

<table>
<thead>
<tr>
<th>Product terms</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note: not all rows need to be used</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PALS: Only inputs to AND gates can be programmed but one term ($F_1$) can be re-used in other functions.

Fig. 7-16 PAL with Four Inputs, Four Outputs, and Three-Wide AND-OR Structure

PAL: Only inputs to AND gates can be programmed but one term ($F_1$) can be re-used in other functions.
Manipulate expressions so that a **common term** is identified. Assign common term to $F_1$.

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

$w = ABC' + A'B'CD'$

$x = A + BCD$

$y = A'B + CD + B'D'$

$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$

$$z = w + AC'D' + A'B'C'D$$
### Table 7-6
PAL Programming Table

<table>
<thead>
<tr>
<th>Product Term</th>
<th>(A)</th>
<th>(B)</th>
<th>(C)</th>
<th>(D)</th>
<th>(W)</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>(w = ABC')</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>+ (A'B'CD')</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(x = A)</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>+ (BCD)</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>(y = A'B)</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>+ (CD)</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>+ (B'D')</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>(z = w)</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>+ (AC'D')</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>+ (A'B'C'D)</td>
</tr>
</tbody>
</table>
Inputs are \( I \) AND \( I' \) and thus produces a 0 always thus there is no connection at these points.

Fig. 7-17  Fuse Map for PAL as Specified in Table 7-6
• PAL practice: problem 7-24
Fig. 7-18  Sequential Programmable Logic Device
Fig. 7-20  General CPLD Configuration
Xilinx FPGAs are based on Configurable Logic Blocks (CLBs)
More generally called logic cells
Programmable

I/O blocks not shown
Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)
Hamming code: Error Detection and Correction

Bit position: 1 2 3 4 5 6 7 8 9 10 11 12
\[ P_1 \quad P_2 \quad 1 \quad P_4 \quad 1 \quad 0 \quad 0 \quad P_8 \quad 0 \quad 1 \quad 0 \quad 0 \]

\[ P_1 = \text{XOR of bits (3, 5, 7, 9, 11)} = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0 \]
\[ P_2 = \text{XOR of bits (3, 6, 7, 10, 11)} = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0 \]
\[ P_4 = \text{XOR of bits (5, 6, 7, 12)} = 1 \oplus 0 \oplus 0 \oplus 0 = 1 \]
\[ P_8 = \text{XOR of bits (9, 10, 11, 12)} = 0 \oplus 1 \oplus 0 \oplus 0 = 1 \]

\[ 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \]

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

- **Compute correction bits**

\[ C_1 = \text{XOR of bits (1, 3, 5, 7, 9, 11)} \]
\[ C_2 = \text{XOR of bits (2, 3, 6, 7, 10, 11)} \]
\[ C_4 = \text{XOR of bits (4, 5, 6, 7, 12)} \]
\[ C_8 = \text{XOR of bits (8, 9, 10, 11, 12)} \]
For no error: 
For no error:

\[ C_8 \quad C_4 \quad C_2 \quad C_1 \]

\begin{align*}
\text{With error in bit 1:} \quad & 0 \quad 0 \quad 0 \quad 1 \\
\text{With error in bit 5:} \quad & 0 \quad 1 \quad 0 \quad 1 \\
\end{align*}

\textbf{Table 7-2} \\
\textit{Range of Data Bits for \( k \) Check Bits} \\

\begin{tabular}{|c|c|}
\hline
\textbf{Number of Check Bits, } \( k \) & \textbf{Range of Data Bits, } \( n \) \\
\hline
3 & 2-4 \\
4 & 5-11 \\
5 & 12-26 \\
6 & 27-57 \\
7 & 58-120 \\
\hline
\end{tabular}

\[ 2^k - 1 - k \geq n \]
Single-error correction, double-error

• To detect a double-error, add an additional parity bit $P = \text{XOR (all other bits)}$

• 12-bit example: $P_{13} = \text{XOR(1...12)}$

• If

  • $C=0 \& P=0$: no error
  • $C\neq0 \& P=1$: single error at bit indicated by $C$
  • $C\neq0 \& P=0$: double error detected
  • $C=0 \& P=1$: error in $P_{13}$
7-10 Given the 8-bit data word 01011011, generate the 13-bit composite word for the Hamming code that corrects single errors and detects double errors.

7-11 Obtain the 15-bit Hamming code word for the 11-bit data word 11001001010.

7-12 A 12-bit Hamming code word containing 8 bits of data and 4 parity bits is read from memory. What was the original 8-bit data word that was written into memory if the 12-bit word read out is as follows:

(a) 000011101010  
(b) 101110000110  
(c) 101111101000
Fig. P7-17