Memory and Programmable Logic
Outline

- Introduction
- Random-Access Memory
- Memory Decoding
- Error Detection and Correction
- Read-Only Memory
- Programmable Devices
- Sequential Programmable Devices
Mass Memory Elements

- Memory is a collection of binary cells together with associated circuits needed to transfer information to or from any desired location.

- Two primary categories of memory:
  - Random access memory (RAM)
  - Read only memory (ROM)
Programmable Logic Device

- The binary information within the device can be specified in some fashion and then embedded within the hardware
  - Most of them are programmed by breaking the fuses of unnecessary connections

- Four kinds of PLD are introduced
  - Read-only memory (ROM)
  - Programmable logic array (PLA)
  - Programmable array logic (PAL)
  - Field-programmable gate array (FPGA)
Random Access Memory

- **A word** is the basic unit that moves in and out of memory.
  - The length of a word is often multiples of a byte (=8 bits).
- Memory units are specified by its **number of words** and the **number of bits** in each word.
  - Ex: 1024(words) x 16(bits)
  - Each word is assigned a particular **address**, starting from 0 up to $2^k - 1$ ($k =$ number of address lines).

<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>10110101010111101</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>000011010100011100</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 $1024 \times 16$ Memory
Write and Read Operations

- **Write to RAM**
  - Apply the binary address of the desired word to the **address lines**
  - Apply the data bits that must be stored in memory to the **data input lines**
  - Activate the **write control**

- **Read from RAM**
  - Apply the binary address of the desired word to the **address lines**
  - Activate the **read control**
Timing Waveforms

- CPU clock = 50 MHz
  - cycle time = 20 ns
- Memory cycle time = 50 ns
  - The time required to complete a write operation
- Memory access time
  - The time required to read it
- The control signals must stay active for at least 50 ns
  - 3 CPU cycles are required
Types of Memories

Access mode:
- Random access: any locations can be accessed in any order
- Sequential access: accessed only when the requested word has been reached (ex: hard disk)

Operating mode:
- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Volatile mode:
- Volatile memory: lose stored information when power is turned off (ex: RAM)
- Non-volatile memory: retain its storage after removal of power (ex: flash, ROM, hard-disk, …)
SRAM vs. DRAM

Static RAM:
- Use internal latch to store the binary information
- Stored information remains valid as long as power is on
- Shorter read and write cycles
- Larger cell area and power consumption

Dynamic RAM:
- Use a capacitor to store the binary information
- Need periodically refreshing to hold the stored info.
- Longer read and write cycles
- Smaller cell area and power consumption
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];          // 64 x 4 memory
always @ (Enable or ReadWrite)
  if (Enable)
    if (ReadWrite)
      DataOut = Mem[Address];   // Read
    else
      Mem[Address] = DataIn;    // Write
  else DataOut = 4'b0;          // High impedance state
endmodule
Outline

- Introduction
- Random-Access Memory
- Memory Decoding
- Error Detection and Correction
- Read-Only Memory
- Combinational Programmable Devices
- Sequential Programmable Devices
Memory Construction

A SRAM Cell

Large memory will require large decoder
Coincident Decoding

- Address decoders are often divided into two parts
  - A two-dimensional scheme
- The total number of gates in decoders can be reduced
- Can arrange the memory cells to a square shape
- EX: 10-bit address 404 = 0110010100
  - X = 01100 (first five)
  - Y = 10100 (last five)
Address Multiplexing

- Memory address lines often occupy too much I/O pads
  - 64K = 16 lines
  - 256M = 28 lines
- Share the address lines of X and Y domains
  - Reduce the number of lines to a half
  - An extra register is required for both domain to store the address
- Two steps to send address
  - RAS=0: send row address
  - CAS=0: send column address

![Diagram of Address Multiplexing](image-url)

Fig. 7-8 Address Multiplexing for a 64K DRAM
Outline

- Introduction
- Random-Access Memory
- Memory Decoding
- Error Detection and Correction
- Read-Only Memory
- Combinational Programmable Devices
- Sequential Programmable Devices
Error Detection & Correction

- Memory arrays are often very huge
  - May cause occasional errors in data access

- Reliability of memory can be improved by employing error-detecting and correcting codes

- Error-detecting code: only check for the existence of errors
  - Most common scheme is the parity bit

- Error-correcting code: check the existence and locations of errors
  - Use multiple parity check bits to generate a syndrome that can indicate the erroneous bits
  - Complement the erroneous bits can correct the errors
Hamming Code (1/2)

- k parity bits are added to an n-bit data word
- The positions numbered as a power of 2 are reserved for the parity bits
  - Ex: original data is 11000100 (8-bit)
  - Bit position: 1 2 3 4 5 6 7 8 9 10 11 12
  - P1 = XOR of bits (3,5,7,9,11) = 0
  - P2 = XOR of bits (3,6,7,10,11) = 0
  - P4 = XOR of bits (5,6,7,12) = 1
  - P8 = XOR of bits (9,10,11,12) = 1
- The composite word is 001110010100 (12-bit)
Hamming Code (2/2)

- When the 12 bits are read from memory, the parity is checked over the **same combination** of bits **including** the parity bit
  - $C_1 = \text{XOR of bits } (1,3,5,7,9,11)$
  - $C_2 = \text{XOR of bits } (2,3,6,7,10,11)$
  - $C_4 = \text{XOR of bits } (4,5,6,7,12)$
  - $C_8 = \text{XOR of bits } (8,9,10,11,12)$

- $(001110010100) \rightarrow C = C_8C_4C_2C_1 = 0000 : \text{no error}$
- $(101110010100) \rightarrow C = C_8C_4C_2C_1 = 0001 : \text{bit 1 error}$
- $(001100010100) \rightarrow C = C_8C_4C_2C_1 = 0101 : \text{bit 5 error}$

**viewed as a binary number**
General Rules of Hamming Code

- The number of parity bits:
  - The syndrome C with k bits can represent $2^k - 1$ error locations (0 indicates no error)
  - $2^k - 1 = n + k \rightarrow 2^k - 1 - k = n$

- The members of each parity bit:
  - C1(P1): have a “1” in **bit 1** of their location numbers 1(0001), 3(0011), 5(0101), 7(0111), 9(1001), ...
  - C2(P2): have a “1” in **bit 2** of their location numbers 2(0010), 3(0011), 6(0110), 7(0111), 10(1010), ...
  - C: with parity bit;  P: without parity bit itself

<table>
<thead>
<tr>
<th>Number of Check Bits, k</th>
<th>Range of Data Bits, n</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2-4</td>
</tr>
<tr>
<td>4</td>
<td>5-11</td>
</tr>
<tr>
<td>5</td>
<td>12-26</td>
</tr>
<tr>
<td>6</td>
<td>27-57</td>
</tr>
<tr>
<td>7</td>
<td>58-120</td>
</tr>
</tbody>
</table>
Extension of Hamming Code

- Original Hamming code can detect and correct only a **single error**
  - Multiple errors are not detected

- Add an extra bit as the parity of total coded word
  - Ex: 001110010100P₁₃ (P₁₃=\text{XOR of bits 1 to 12})
  - Still single-error correction but double-error detection

- Four cases can occur:
  - If C=0 and P=0, no error occurred
  - If C \neq 0 and P=1, single error occurred (can be fixed)
  - If C \neq 0 and P=0, double error occurred (cannot be fixed)
  - If C=0 and P=1, an error occurred in the P₁₃ bit
Outline

- Introduction
- Random-Access Memory
- Memory Decoding
- Error Detection and Correction
- **Read-Only Memory**
- Combinational Programmable Devices
- Sequential Programmable Devices
Basic ROM Structure

Figure 3-1 Basic ROM Structure
Read Only Memory

- A memory device that can permanently keep binary data
  - Even when power is turned off and on again
- For a $2^k \times n$ ROM,
  it consists of
  - $k$ inputs (address line) and $n$ outputs (data)
  - $2^k$ words of $n$-bit each
  - A $k \times 2^k$ decoder (generate all minterms)
  - $n$ OR gates with $2^k$ inputs
  - Initially, all inputs of OR gates and all outputs of the decoder are fully connected

Fig. 7-10 Internal Logic of a $32 \times 8$ ROM
Each intersection (crosspoint) in the ROM is often implemented with a fuse.

 Blow out unnecessary connections according to the truth table:

- “1” means connected (marked as X)
- “0” means unconnected

Cannot recovered after programmed.

Fig. 7-11 Programming the ROM According to Table 7-3
Design Comb. Circuit with ROM

- Derive the truth table of the circuit
- Determine minimum size of ROM
- Program the ROM

### Truth Table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₂ A₁ A₀</td>
<td>B₅ B₄ B₃ B₁ B₀</td>
<td></td>
</tr>
<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>

3 select lines = 8 minterms

Word length = 4

(a) Block diagram  
(b) ROM truth table
Mealy Sequential Network

Figure 3-2 Realization of a Mealy Sequential Network

ROM
16 Words
X 4 Bits

X → Z

D1
Q1
CK

D2
Q2
CK

D3
Q3
CK

Clock
## ROM Truth Table

### Table 3-1 ROM Truth Table

<table>
<thead>
<tr>
<th>$Q_1$</th>
<th>$Q_2$</th>
<th>$Q_3$</th>
<th>$X$</th>
<th>$Q_1^+$</th>
<th>$Q_2^+$</th>
<th>$Q_3^+$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<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>1</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>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>
<td>0</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>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</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>1</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>1</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>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Types of ROMs

- Mask programming
  - Program the ROM in the semiconductor factory
  - Economic for large quantity of the same ROM

- Programmable ROM (PROM)
  - Contain all fuses at the factory
  - Program the ROM by burning out the undesired fuses (irreversible process)

- Erasable PROM (EPROM)
  - Can be restructured to the initial state under a special ultra-violet light for a given period of time

- Electrically erasable PROM (EEPROM or E²PROM)
  - Like the EPROM except being erased with electrical signals
Programmable Logic Devices

- ROM provides full decoding of variables
  - Waste hardware if the functions are given
- For known combinational functions, Programmable Logic Devices (PLD) are often used
  - Programmable read-only memory (PROM)
  - Programmable array logic (PAL)
  - Programmable logic array (PLA)
- For sequential functions, we can use
  - Sequential (simple) programmable logic device (SPLD)
  - Complex programmable logic device (CPLD)
  - Field programmable gate array (FPGA)

*most popular*
Outline

- Introduction
- Random-Access Memory
- Memory Decoding
- Error Detection and Correction
- Read-Only Memory
- **Combinational Programmable Devices**
- Sequential Programmable Devices
Configurations of Three PLDs

(a) Programmable read-only memory (PROM)

(b) Programmable array logic (PAL)

(c) Programmable logic array (PLA)

Fig. 7-13 Basic Configuration of Three PLDs
PLA Structure

Figure 3-4 Programmable Logic Array Structure
PLAs

- The decoder in ROM is replaced by an AND array
  - More than 1 product terms can be selected
- The OR array Ors together the product terms to form the outputs
PLA

\[ F_0 = A'B' + AC' \]
\[ F_1 = B + AC' \]
\[ F_2 = A'B' + BC' \]
\[ F_3 = AC + B \]
Circuit Structure

Figure 3-5 PLA with 3 Inputs, 5 Product Terms, and 4 Outputs
AND-OR Equivalent Structure

Figure 3-8 AND-OR Array Equivalent to Figure 3-5

\[
\begin{align*}
F_0 &= A'B' + AC' \\
F_1 &= B + AC' \\
F_2 &= A'B' + BC' \\
F_3 &= AC + B
\end{align*}
\]
### Table 3-2 PLA Table for Figure 3-5

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A$</td>
<td>$B$</td>
</tr>
<tr>
<td>$A'B'$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$AC'$</td>
<td>1</td>
<td>–</td>
</tr>
<tr>
<td>$B$</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>$BC'$</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>$AC$</td>
<td>1</td>
<td>–</td>
</tr>
</tbody>
</table>
Multiple-Output Optimization

Figure 3-9 Multiple-Output Karnaugh Maps
Multiple-Output Function

- Minimize each function separately
  - 8 product terms
  - \( F_1 = bd + b'c + ab' \)
  - \( F_2 = c + a'bd \)
  - \( F_3 = bc + ab'c' + abd \)
### Reduced PLA Table

**Table 3-3 Reduced PLA Table**

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$F_1$</th>
<th>$F_2$</th>
<th>$F_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<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>
<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>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</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>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
F_1 = a'bd + abd + ab'c' + b'c \\
F_2 = a'bd + b'c + bc \\
F_3 = abd + ab'c' + bc
\]
Figure 3-10 PLA Realization of Equations (3-4)
PLA vs ROM

**PLA**
- Each row represents a term
- More than one rows may be selected by each input combination
- Selected rows are Ored

**ROM**
- Each row represents a minterm
- Exactly one row is selected
- Output is the bit pattern stored in each row
Programmable Logic Array

- PLA does not provide full decoding of the variables
  - Only generate the terms you need
- The decoder is replaced by an array of AND gates that can be programmed

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs</th>
<th>Outputs (T)</th>
<th>(C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB'</td>
<td>1 1 0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>AC</td>
<td>2 1 - 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BC</td>
<td>3 - 1 1</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>A'BC'</td>
<td>4 0 1 0</td>
<td>1</td>
<td>-</td>
</tr>
</tbody>
</table>

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

Generate complemented outputs (if required)

Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs
Implementation with PLA

- Example 7-2: implement the two functions with PLA
  \[ F_1(A, B, C) = \sum (0, 1, 2, 4) \]
  \[ F_2(A, B, C) = \sum (0, 5, 6, 7) \]
- Goal: minimize the number of distinct product terms between two functions

\[
\begin{array}{cccc}
  & BC & 00 & 01 \\
A & 0 & 1 & 1 & 0 & 1 \\
 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
  & BC & 00 & 01 \\
A & 0 & 1 & 0 & 0 & 0 \\
 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
  & BC & 00 & 01 \\
A & 0 & 1 & 1 & 0 & 0 \\
 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

PLA programming table

<table>
<thead>
<tr>
<th>Product term</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F_1 )</td>
<td>( A \ B \ C )</td>
<td>( (C) )</td>
</tr>
<tr>
<td>( A B )</td>
<td>1 1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>( A C )</td>
<td>2 1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>( B C )</td>
<td>3 1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>( A' B' C' )</td>
<td>4 0 0 0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ F_1 = A' B' + A' C' + B' C' \]
\[ F_1 = (A B + A C + B C)' \]
\[ F_2 = A B + A C + A' B' C' \]
\[ F_2 = (A' C + A' B + A B' C')' \]
Programmable Array Logic (PAL)

- AND array is programmable
  - Not shared
- OR array is fixed
- Less expensive
- Easier to program
Figure 3-12  Combinational PAL Segment
Programming PAL

(b) Programmed

\[ I_1 I_2' + I_1' I_2 \]
Sequential PAL Segment

Figure 3-13 Segment of a Sequential PAL

\[ Q^+ = D = A'BQ' + AB'Q \]
Example

AND gate is logic 1 when there are no connections to it

\[ Z' = XQ_3' + X'Q_3 \]

\[ D_3 = Q_1Q_2Q_3' + X'Q_1Q_3' + XQ_1'Q_2' \]
16R4 PAL

- 8 dedicated inputs
- 4 I/O ports
- 4 D FFs with inverting tristate buffers
  - Can be fed back to AND array
- AND array with 16 input variables
- Each OR gate is fed from 8 AND gates
Programmable Array Logic

- PAL has a fixed OR array and a programmable AND array
  - Easier to program but not as flexible as PLA
- Each input has a buffer-inverter gate
- One of the outputs is fed back as two inputs of the AND gates
- Unlike PLA, a product term cannot be shared among gates
  - Each function can be simplified by itself without common terms
### Implementation with PAL

\[ w = \Sigma(2,12,13) \quad x = \Sigma(7,8,9,10,11,12,13,14,15) \quad y = \Sigma(0,2,3,4,5,6,7,8,10,11,15) \quad z = \Sigma(1,2,8,12,13) \]

<table>
<thead>
<tr>
<th>Product Term</th>
<th>AND Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A:1 B:1 C:0 D:0 W:0</td>
<td>( w = ABC' ) + A'B'CD'</td>
</tr>
<tr>
<td>2</td>
<td>A:0 B:0 C:1 D:0 W:0</td>
<td>x = A + BCD</td>
</tr>
<tr>
<td>3</td>
<td>A:0 B:0 C:0 D:0 W:0</td>
<td>y = A'B + CD + B'D'</td>
</tr>
<tr>
<td>4</td>
<td>A:1 B:1 C:0 D:0 W:0</td>
<td>z = w + AC'D' + A'B'C'D</td>
</tr>
<tr>
<td>5</td>
<td>A:0 B:0 C:1 D:1 W:0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A:0 B:0 C:0 D:0 W:0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A:0 B:1 C:0 D:0 W:0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A:0 B:0 C:1 D:1 W:0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A:0 B:0 C:0 D:0 W:0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A:0 B:0 C:0 D:1 W:0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>A:1 B:0 C:0 D:0 W:0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>A:0 B:0 C:0 D:1 W:0</td>
<td></td>
</tr>
</tbody>
</table>

**AND gates inputs**

- **w**
- **x**
- **y**
- **z**

**Product terms**: A A' B B' C C' D D' w w'.

**AND gates**: fuse intact or blown.
Outline

- Introduction
- Random-Access Memory
- Memory Decoding
- Error Detection and Correction
- Read-Only Memory
- Combinational Programmable Devices
- **Sequential Programmable Devices**
Sequential PLD

- The most simple sequential PLD = PLA (PAL) + Flip-Flops

The mostly used configuration for SPLD is constructed with 8 to 10 macrocells as shown right.
Figure 3-15 Block Diagram for 22V10
22V10

- 12 dedicated inputs, 10 Input/Output
- 10 OR gates
  - 8 to 16 AND gates
  - Each OR drives an output logic macrocell
- 10 D FFs
  - Common clock
  - asynchronous reset (AR)
  - Synchronous preset (SP)
Output Macrocell

Figure 3-16 Output Macrocell

(a) Paths with $S_1 = S_0 = 0$
Output Cell Configuration

(b) Paths with $S_1 = S_0 = 1$
Complex PLD

- Complex digital systems often require the connection of several devices to produce the complex specification
  - More economical to use a complex PLD (CPLD)
- CPLD is a collection of individual PLDs on a single IC with programmable interconnection structure
Field Programmable Gate Array

- Gate array: a VLSI circuit with some pre-fabricated gates repeated thousands of times
  - Designers have to provide the desired interconnection patterns to the manufacturer (factory)
- A field programmable gate array (FPGA) is a VLSI circuit that can be programmed in the user’s location
  - Easier to use and modify
  - Getting popular for fast and reusable prototyping
- There are various implementations for FPGA
FPGA Structure (Altera)

Fig. 6-28  Altera® MAX 7000™ Structure (Reprinted with Permission of Altera Corporation, © Altera Corp., 1991)
FPGA Structure (Xilinx)

Fig. 6-29: Xilinx® XC4000™ FPGA Structure (Adapted with Permission of Xilinx, Inc.)

- Configurable logic block (CLB)
- Input/Output Block (IOB)
- Switch matrix

Single length
Long lines
Xilinx XC3020

- 64 Configurable logic blocks (CLBs)
- 64 input-output interface blocks
- Interconnection programmed by storing data in internal configurable memory cells
- Each CLB with combinational logic and 2 D FFs
- Programmed logic functions and interconnections are retained until power is off
Each memory is selected in turn

Each connection point has an associated memory cell
Xilinx 3000 Series

Figure 6-3  Xilinx 3000 Series Logic Cell

- DATA IN  DI
- LOGIC VARIABLES
  - A
  - B
  - C
  - D
  - E
- COMBINATORIAL FUNCTION
  - QX
  - QY
- MUX
  - 0 MUX 1
- D Q
  - RD
- CLB OUTPUTS
  - X
  - Y
- ENABLE CLOCK  EC
- 1 (ENABLE)
- CLOCK  K
- DIRECT
  - RD
- 0 (INHIBIT)
  - (GLOBAL RESET)
CLB

- 5 logic inputs
- Data input (DI)
- Clock (K)
- Clock enable (EC)
- Direct reset (RD)
- 2 outputs (X,Y)
Store the Programming Info.

- SRAM technology is used
  - \( M = 1 \)-bit SRAM
  - Loaded from the PROM after power on

- Store control values
  - Control pass transistor
  - Control multiplexer

- Store logic functions
  - Store the value of each minterm in the truth table
Xilinx FPGA Routing

- Fast direct interconnect
  - Adjacent CLBs

- General purpose interconnect
  - CLB – CLB or CLB – IOB
  - Through switch matrix

- Long lines
  - Across whole chip
  - High fan-out, low skew
  - Suitable for global signals (CLK) and buses
  - 2 tri-states per CLB for busses
General Interconnect

Figure 6-9
Direct Interconnect

Figure 6-10
Long Lines

Figure 6-11 Vertical and Horizontal Long Lines
Xilinx Switch Matrix

- Six pass transistors to control each switch node
- The two lines at point 1 are joined together
- At point 2, two distinct signal paths pass through one switch node

Fig. 6-31 Example of Xilinx® Switch Matrix (Adapted with Permission of Xilinx®, Inc.)
Configurable Logic Block (CLB)

- Combinational logic via lookup table
  - Any function(s) of available inputs
- Output registered and/or combinational
Simplified CLB Structure
I/O Block (IOB)

- Periphery of identical I/O blocks
  - Input, output, or bidirectional
  - Registered, latched, or combinational
  - Three-state output
  - Programmable output slew rate
Input/Output Mode of an IOB

- **Input**
  - 3-state control places the output buffer into high impedance
  - Direct in and/or registered in

- **Output**
  - 3-state driver should be enabled by TS signal
  - Direct output or registered output
Design with FPGA

- Using HDL, schematic editor, SM chart or FSM diagram to capture the design
- Simulate and debug the design
- Work out detail logic and feed the logic into CLBs and IOBs
  - Completed by a CAD tool
- Generate bit pattern for programming the FPGA and download into the internal configurable memory cells
- Test the operations
FPGA Design Flow

Advantages: Fast and reusable prototyping
- Can be reprogrammed and reused
- Implementation time is very short

Disadvantages: Expensive and high volume

logic + layout synthesis
Download to a FPGA Demo Board

Source: CIC training manual
HDL Modeling for Memory

- Modeling ROM and combinational PLDs
  - Similar to modeling a combinational code converter

- Modeling RAM
  - Use memory array declaration in Verilog
    ex: `reg [3:0] MY_MEM [0:63]; // 64 4-bit registers`
    `MY_MEM[0] <- 4-bit variable`
  - Can load memory by using a system task
    ex: `$readmemb("mem_content", MY_MEM, 0, 63);`
  - If synthesized, only SRAM (array of registers) will be generated
    - Use memory compiler or pre-designed layout instead