Chapter 7

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 \(2^k - 1\)
      \((k = \text{number of address lines})\)

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Binary</th>
<th>decimal</th>
<th>Memory content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000000</td>
<td>0</td>
<td>0</td>
<td>101101010101101</td>
</tr>
<tr>
<td>0000000001</td>
<td>1</td>
<td>1</td>
<td>101010111001001</td>
</tr>
<tr>
<td>0000000010</td>
<td>2</td>
<td>2</td>
<td>00011010100110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1111111101</td>
<td>1021</td>
<td>...</td>
<td>100111010010100</td>
</tr>
<tr>
<td>1111111110</td>
<td>1022</td>
<td>...</td>
<td>000110100001110</td>
</tr>
<tr>
<td>1111111111</td>
<td>1023</td>
<td>...</td>
<td>110111100100101</td>
</tr>
</tbody>
</table>

Fig. 7-3 Content of a 1024 × 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 access time = 50 ns
  - The time required to complete a read or write operation
- 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

---

Fig. 6-5: Static RAM Cell
Fig. 6-12: Dynamic RAM Cell, Hydraulic Analogy of Cell Operation, and Cell Model
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
  
  \[
  404 = 0110010100 \\
  X = 01100 \text{ (first five)} \\
  Y = 10100 \text{ (last five)}
  \]

### 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
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
  - \( P_1 \ P_2 \ 1 \ P_4 \ 1 \ 0 \ 0 \ P_8 \ 0 \ 1 \ 0 \ 0 \)
  - \( P_1 = \text{XOR of bits (3,5,7,9,11)} = 0 \)
  - \( P_2 = \text{XOR of bits (3,6,7,10,11)} = 0 \)
  - \( P_4 = \text{XOR of bits (5,6,7,12)} = 1 \)
  - \( P_8 = \text{XOR of bits (9,10,11,12)} = 1 \)
- 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
  - \( C_1 = \text{XOR of bits (1,3,5,7,9,11)} \)
  - \( C_2 = \text{XOR of bits (2,3,6,7,10,11)} \)
  - \( C_4 = \text{XOR of bits (4,5,6,7,12)} \)
  - \( C_8 = \text{XOR of bits (8,9,10,11,12)} \)
  - \((001110010100) \rightarrow C = C_8 C_4 C_2 C_1 = 0000 : \text{no error}\)
  - \((101110010100) \rightarrow C = C_8 C_4 C_2 C_1 = 0001 : \text{bit 1 error}\)
  - \((001100010100) \rightarrow C = C_8 C_4 C_2 C_1 = 0101 : \text{bit 5 error}\)

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)
  - $2^k - 1 \geq n + k \rightarrow 2^k - 1 - k \geq n$

- 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=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≠0 and P=1, single error occurred (can be fixed)
  - If C≠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

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
Programming the 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

![5 x 32 decoder](image)

Design Comb. Circuit with ROM

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

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$</td>
<td>$A_1$</td>
<td>$A_0$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</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>

3 select lines = 8 minterms

![Block diagram](image)

(word length = 4)
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 ultraviolet 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) most popular
  - Field programmable gate array (FPGA)
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
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</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>AB'</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>AC</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>BC</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>A'BC'</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

F₁ = AB' + AC + A'BC'
F₂ = (AC + BC)'

Generate complemented outputs (if required)

Implementation with PLA

- Example 7-2: implement the two functions with PLA
  F₁(A, B, C) = ∑(0, 1, 2, 4)
  F₂(A, B, C) = ∑(0, 5, 6, 7)
- Goal: minimize the number of distinct product terms between two functions

F₁(A, B, C) = A'B' + A'C' + B'C'
F₂(A, B, C) = AB + AC + BC'
F₃ = (AB + AC + BC)'
F₄ = (A'C + A'B + AB'C)'

 PLA programming table

<table>
<thead>
<tr>
<th>Product term</th>
<th>Inputs A B C</th>
<th>Outputs F₁</th>
<th>F₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>1 1 1 - - - -</td>
<td>1 - - - -</td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td>2 1 1 1 1 1</td>
<td>1 1 1 1</td>
<td></td>
</tr>
<tr>
<td>BC</td>
<td>3 - 1 1 1 -</td>
<td>1 1 1 -</td>
<td></td>
</tr>
<tr>
<td>A'BC'</td>
<td>4 0 0 0 0 -</td>
<td>- 1 - -</td>
<td></td>
</tr>
</tbody>
</table>

F₃ = AB + AC + A'BC'
F₄ = (A'C + A'B + AB'C)'
**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**

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

<table>
<thead>
<tr>
<th>Product Term</th>
<th>AND Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<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>x = A</td>
</tr>
<tr>
<td>4</td>
<td>1 - - - -</td>
<td>+ BCD</td>
</tr>
<tr>
<td>5</td>
<td>- 1 1 1 -</td>
<td></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>- - - - 1</td>
<td>z = w</td>
</tr>
<tr>
<td>11</td>
<td>1 0 0 - 1</td>
<td>+ ACD'</td>
</tr>
<tr>
<td>12</td>
<td>0 0 0 1 -</td>
<td>+ A'B'CD</td>
</tr>
</tbody>
</table>

\[/\text{slide}^1\]
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
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
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
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

Internal Functions of a CLB

- Two 4-input tables implement two distinct functions (F’ and G’)
- F’ and G’ with another control (H1) feed into a third lookup table (H’)
- Two arbitrary functions of up to four variables and selected functions of up to nine variables can be implemented
- Properly setting the two MUXes can assign any pair of F’, G’, and H’ to the two combinational outputs (X and Y)
Internal Functions of a CLB

- Two D flip-flops directly drive outputs XQ and YQ
- Each of the D inputs can be selected from F’, G’, H’ and input DIN
- Two XORs select each flip-flop individually to be positive or negative edge triggered
- Two SR controls select the signal S/R to be an asynchronous Set or Reset for the flip-flops
- Two multiplexers allow the input EC to optionally act as a clock ENABLE signal for each flip-flop

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

Download to a FPGA Demo Board

Source: CIC training manual