Digital Circuit Design and Language

Memory and Programmable Logic

Chang, Ik Joon
Kyunghhee University
Memory

- Classification based on functionality
  - ROM : Read-Only Memory
  - RWM : Read-Write Memory

<table>
<thead>
<tr>
<th>RWM</th>
<th>NVRWM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Access Memory (RAM)</td>
<td>Non-Random Access</td>
<td>EPROM EEPROM FLASH</td>
</tr>
<tr>
<td>SRAM DRAM</td>
<td>FIFO LIFO Register CAM</td>
<td></td>
</tr>
</tbody>
</table>
Memory Hierarchy

- Register in CPU
- L1 / L2 Cache: SRAM
- Mass Storage: HDD, Non-volatile memory
How to Make Programmable Logic?

- Fuse / Anti-Fuse
- SRAM-based Wiring
- Flash-based Wiring
# Memory

## Classification based on functionality

- **ROM**: Read-Only Memory
- **RWM**: Read-Write Memory

## Table

<table>
<thead>
<tr>
<th>RWM</th>
<th>NVRWM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Access Memory (RAM)</td>
<td>Non-Random Access</td>
<td>ROM</td>
</tr>
<tr>
<td>SRAM, DRAM</td>
<td>FIFO, LIFO, Register, CAM</td>
<td>EPROM, EEPROM, FLASH</td>
</tr>
</tbody>
</table>
Storage Cell (SRAM vs. DRAM)

- **SRAM**: Large Size, but fast speed (compared to DRAM), no refresh operation
- **DRAM**: Small Size, but low speed (compared to SRAM), refresh operation is indispensable
- **WL**: Word-line, **BL**: Bit-line
Random Access Memory: Architecture

- We need a row decoder to reduce # of address pin
- But, Height >> Width
Random Access Memory: Architecture (Cont.)

- Row Decoder + Column Decoder
Random Access Memory: Hierarchical Architecture

- Hierarchical architecture reduces wiring
- Only one block is activated → low power dissipation
Random Access Memory: read and write operation

- **Write operation**
  1. Transfer the binary address of the desired word to the address lines.
  2. Transfer the data bits that must be stored in memory to the data input lines.
  3. Activate the write input

- **Read operation**
  1. Transfer the binary address of the desired word to the address lines.
  2. Activate the read input.

### Control Inputs to Memory Chip

<table>
<thead>
<tr>
<th>Memory Enable</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>

<table>
<thead>
<tr>
<th>Timing Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Timing Diagram" /></td>
</tr>
</tbody>
</table>

(a) Write cycle
(b) Read cycle
Random Access Memory: Address Multiplexing

Address bus | Row Address | Column Address
---|---|---
RAS
CAS

RAS-CAS timing

DRAM: Timing Multiplexed Addressing

Address Bus | Address
---|---
Address transition initiates memory operation

SRAM: Timing Self-timed

To reduce # of address pin, DRAM uses timing multiplexed addressing

Address Multiplexing in 64K DRAM
Memory Yield and Reliability Degradation

- Semiconductor memories trade off noise-margin for density and performance.

- Highly Sensitive to Noise.

- High Density and Large Die size cause Yield Problems.

\[
Y = 100 \times \frac{\text{Number of Good Chips on Wafer}}{\text{Number of Chips on Wafer}}
\]

- In scaled technologies, it is challenging to deliver good yield and reliability in memory.
What Degrades Yield and Reliability? (PVT Variation)

Process variation
- Random dopant fluctuation
- Line edge roughness

Voltage variation
- Power Line IR Drop
- Ldi/dt noise

Temperature variation
- Global T variation (Environmental variation)
- Local T variation

M. Hane, SISPAD 03
H. Chen, DAC 97
R. Krishnamurthy, ISLPED 05
What Degrades Yield and Reliability? (Many Noise Source)

Coupling Noise (Cross-talk)

1 Particle ~ 1 Million Carriers

Soft-Error Noise
Row / Column replacement improves memory yield
Solution: Hamming Code

Parity Generation Rule

<table>
<thead>
<tr>
<th>Bit position</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>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoded data bits</td>
<td>p1</td>
<td>p2</td>
<td>d1</td>
<td>p4</td>
<td>d2</td>
<td>d3</td>
<td>d4</td>
<td>p8</td>
<td>d5</td>
<td>d6</td>
<td>d7</td>
<td>d8</td>
<td>d9</td>
<td>d10</td>
<td>d11</td>
<td>p16</td>
<td>d12</td>
<td>d13</td>
<td>d14</td>
<td>d15</td>
</tr>
<tr>
<td>Parity bit coverage</td>
<td>p1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>p2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>p4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>p8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>p16</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Ex) 8-bit data = 11000100

Bit position

<table>
<thead>
<tr>
<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>1</td>
<td>P_1</td>
<td>P_2</td>
<td>1</td>
<td>P_4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>P_8</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

\[ C_1 = \text{XOR of bits } 1,3,5,7,9,11 \]
\[ C_2 = \text{XOR of bits } 2,3,6,7,10,11 \]
\[ C_4 = \text{XOR of bits } 4,5,6,7,12 \]
\[ C_8 = \text{XOR of bits } 8,9,10,11,12 \]
### Solution: Hamming Code (Cont.)

<table>
<thead>
<tr>
<th>Bit position</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<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>
</tr>
<tr>
<td>1</td>
<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>
</tr>
<tr>
<td>0</td>
<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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</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>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- No Error
- Error in bit 1
- Error in bit 5
- Error in bit 6
- Error in bit 1, 5

<table>
<thead>
<tr>
<th>C_8</th>
<th>C_4</th>
<th>C_2</th>
<th>C_1</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Error Detect (O), Correction (O)
- Error Detect (O), Correction (O)
- Error Detect (O), Correction (O)
- Error Detect (O), Correction (O)
- Error Detect (X), Correction (X)

- Hamming Code enables single-bit error detection and correction
Solution: SECDED (Single Error Cor., Double Error Det.)

<table>
<thead>
<tr>
<th>Bit position</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>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<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>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
P_{13} = \text{XOR of bits (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)}
\]

\[
C_1 = \text{XOR of bits (1, 3, 5, 7, 9, 11)} \quad C_2 = \text{XOR of bits (2, 3, 6, 7, 10, 11)}
\]

\[
C_4 = \text{XOR of bits (4, 5, 6, 7, 12)} \quad C_8 = \text{XOR of bits (8, 9, 10, 11, 12)}
\]

\[
C = C_8 + C_4 + C_2 + C_1,
\]

\[
P = \text{XOR of bits (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)}
\]

If \( C = 0 \) and \( P = 0 \), No error occurred
If \( C = 1 \) and \( P = 1 \), A single error occurred, which can be corrected
If \( C = 1 \) and \( P = 0 \), A double error occurred, which is detected but cannot be corrected
If \( C = 0 \) and \( P = 1 \), An error occurred in the P13 bit
## Memory

### Classification based on functionality
- **ROM**: Read-Only Memory
- **RWM**: Read-Write Memory

<table>
<thead>
<tr>
<th>RWM</th>
<th>NVRWM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Access Memory (RAM)</td>
<td>Non-Random Access</td>
<td>EPROM EEPROM FLASH</td>
</tr>
<tr>
<td>SRAM DRAM</td>
<td>FIFO LIFO Register CAM</td>
<td>PROM</td>
</tr>
</tbody>
</table>
ROM (Read-Only Memory)

$k = 5,$
$n = 8$
Programming Rom According to Table

**ROM Truth Table (Partial)**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_4 )</td>
<td>( A_7 )</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>1 0 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 1</td>
<td>0 0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 1 0</td>
<td>1 1 1 0 0</td>
</tr>
<tr>
<td>0 0 0 1 1</td>
<td>1 0 1 1 0</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>0 0 1 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>0 1 0 0 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>0 0 1 1 0</td>
</tr>
</tbody>
</table>

![Diagram](image-url)
Read-Only Memory Cells

Diode ROM

MOS ROM 1

MOS ROM 2
Read-Only Memory: MOS-NOR ROM


$V_{DD}$
Pull-up devices

GND
Read-Only Memory: Mask Programming

Programming using the Active Layer Only

Cell (9.5\(\lambda\) x 7\(\lambda\))

- Vender should prepare customized mask (expensive)
Read-Only Memory:
Contact Programming (PROM)

Cell (11\(\lambda\) x 7\(\lambda\))

- ‘Inact fuse’ will be removed by high field
Memory

### Classification based on functionality

- **ROM**: Read-Only Memory
- **RWM**: Read-Write Memory

<table>
<thead>
<tr>
<th>RWM</th>
<th>NVRWM</th>
<th>ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Access Memory (RAM)</td>
<td>Non-Random Access</td>
<td>EPROM, EEPROM, FLASH</td>
</tr>
<tr>
<td>SRAM, DRAM</td>
<td>FIFO, LIFO, Register, CAM</td>
<td>PROM</td>
</tr>
</tbody>
</table>
Flash Memory

Device cross-section

Schematic symbol

Tunneling injection

Removing programming voltage leaves charge trapped

Programming results in higher $V_T$. 
Programmable Logic Device (PLD)

(a) Programmable read-only memory (PROM)
(b) Programmable array logic (PAL)
(c) Programmable logic array (PLA)
Programmable Logic Array

Programmable AND array

Programmable OR array

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

PAL—With a fixed OR array and a programmable AND array. Not as flexible as the PLA (Only the AND gate are programmable.)
Programmable Array Logic: Example

\[ w(A, B, C, D) = \sum (2,12,13) \]
\[ x(A, B, C, D) = \sum (7,8,9,10,11,12,12,14,15) \]
\[ y(A, B, C, D) = \sum (0,2,3,4,5,6,7,8,10,11,15) \]
\[ z(A, B, C, D) = \sum (1,2,8,12,13, ) \]

\[ w = ABC' + A'B'CD' \]
\[ x = A + BCD \]
\[ y = A'B + CD + B'D' \]
\[ z = ABC' + A'B'CD' + AC'D' + A'B'C'D \]
\[ = w + AC'D' + A'B'C'D \]
Programmable Array Logic: Example (Cont.)

[Diagram of a programmable array logic example with AND gates and product terms labeled with variables A, A', B, B', C, C', D, D', w, w', x, y, z, and fuses marked as intact or blown.]
Sequential Programmable Devices

- Sequential (or simple) Programmable Logic Device (SPLD)
- Complex Programmable Logic Device (CPLD)
- Field Programmable Gate Array (FPGA)

Unlike combinational PLD’s, includes both gates and flip-flops
Sequential Programmable Logic Device

Sequential Programmable Logic Device (SPLD)

Basic Macro-cell Logic of SPLD
Complex Programmable Logic Device

- Multiple PLD’s are interconnected through a programmable switch matrix
Look up table

- A truth table stored in SRAM, which provides the combinational circuit functions

Multiplexers / Gates / Flip-flops

Example: Xilinx, Altera
Xilinx Spartan: Basic Architecture

The loop up table of CLB can be utilized as block memory.
Xilinx Spartan: Programmable Interconnect Point

- **PIP** = transmission gate whose is controlled by SRAM cell
Xilinx Spartan: I/O Blocks (IOB)

- IOB’s are bi-directional
- The output buffer should be implemented as tri-gates
Xilinx Spartan: Distributed RAM

Single-Port RAM

- CLB is able to form single-port / dual-port RAM
Xilinx Spartan II Architecture
## Xilinx FPGA

<table>
<thead>
<tr>
<th>Part</th>
<th>Spartan</th>
<th>Spartan XL</th>
<th>Spartan II</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>XC4000 Based</td>
<td>XC4000 Based</td>
<td>Virtex Based</td>
</tr>
<tr>
<td>Max # System Gates</td>
<td>5K–40K</td>
<td>5K–40K</td>
<td>15K–200K</td>
</tr>
<tr>
<td>Memory</td>
<td>Distributed RAM</td>
<td>Distributed RAM</td>
<td>Block + Distributed</td>
</tr>
<tr>
<td>I/O Performance</td>
<td>80 MHz</td>
<td>100 MHz</td>
<td>200 MHz</td>
</tr>
<tr>
<td>I/O Standards</td>
<td>4</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>Core Voltage</td>
<td>5 V</td>
<td>3.3 V</td>
<td>2.5 V</td>
</tr>
<tr>
<td>DLLs</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>