Memory design

Top-level organization of a memory

Fig. 7-2  Block Diagram of a Memory Unit

n data input lines

k address lines

Memory unit

2^k words

n bit per word

Write

Read

n data output lines
Example of 1K x 16 memory addressing and contents

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Memory content</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000000</td>
<td>1011010101011101</td>
</tr>
<tr>
<td>00000000001</td>
<td>1010100110001001</td>
</tr>
<tr>
<td>00000000010</td>
<td>0000101010000110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11111111111</td>
<td>1101110000100101</td>
</tr>
</tbody>
</table>

Some Types

- Random Access Memory (RAM)
  - Data can Read and Written
- Read-Only Memory (ROM)
  - In normal operation, only reading is possible
  - Writing takes a special kind of operation

- RAM technologies:
  - Static ("SRAM")
    - Holds stored data as long as power is applied
    - Fast
    - Expensive, high power, low density
  - Dynamic ("DRAM")
    - Stored data fades with time – must be refreshed every few milliseconds
    - Slow (comparatively)
    - Cheap, low power, high density
Typical memory timing

Static Memory cell (using S-R FF)
A decoding problem

- A problem:
  - A $k \times 2^k$ decoder requires $2^k$, $k$-input AND gates
  - As memory size grows, the number and size of the AND gates explode

2D decoding

- Add 2nd enable line to cells
- Instead of a $k \times 2^k$ decoder, use two $k/2 \times 2^{k/2}$
  - $k \times 2^k$:
    - # ANDS = $2^k$
    - # inputs = $k$
  - $k/2 \times 2^{k/2}$:
    - # ANDS = $2(2^{k/2}) = 2^{k+1}$
    - # inputs = $k/2$
- Example: 1K x ?? RAM:
  - 1024 10-input ANDS
  - 64 5-input ANDS
Address multiplexing

Reduces # pins on chip and wires in buss

Error detection and correction
Error detection

- Can determine (within limits) that there is an error(s) in a binary data item, but cannot tell which bit(s) is wrong

- Example: Simple parity check will detect any odd number of bits in error (review)
  - Odd parity -- add a parity bit to the item that makes the total # 1’s odd
    - stored = 10110000
    - read = 10010000 → error
  - Even parity – same, except make # 1’s even
  - Recal that we can use XOR to implement this

Error correction

- Can determine (within limits) that there is an error(s) in a binary data item, and which bit(s) is wrong

- Example: 2D parity check

<table>
<thead>
<tr>
<th>Stored</th>
<th>Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>1011 0</td>
<td>1011 0</td>
</tr>
<tr>
<td>0001 0</td>
<td>0101 0 x</td>
</tr>
<tr>
<td>1111 1</td>
<td>1111 1</td>
</tr>
<tr>
<td>0000 1</td>
<td>0000 1</td>
</tr>
<tr>
<td>1010 1</td>
<td>1010 1 x</td>
</tr>
</tbody>
</table>
Hamming Code

- Basic Hamming Code can detect and correct any single error
- Can be extended to any length data
- Can be extended to more errors
- Efficient (low number of parity bits per data bit)

For 8-bit data:

\[
\begin{align*}
\text{p1} & = \text{XOR} (d1, d2, d4, d5, d7) \\
\text{p2} & = \text{XOR} (d1, d3, d4, d6, d7) \\
\text{p4} & = \text{XOR} (d2, d3, d4, d8) \\
\text{p8} & = \text{XOR} (d5, d6, d7, d8)
\end{align*}
\]

to check:

\[
\begin{align*}
\text{c1} & = \text{XOR} (\text{p1}, d1, d2, d4, d5, d7) \\
\text{c2} & = \text{XOR} (\text{p2}, d1, d3, d4, d6, d7) \\
\text{c4} & = \text{XOR} (\text{p4}, d2, d3, d4, d8) \\
\text{c8} & = \text{XOR} (\text{p8}, d5, d6, d7, d8)
\end{align*}
\]

"c8 c4 c2 c1" read as binary gives the error position

ROMs and Array Logic
Crosspoint is connected for "1", not for "0". (do this by blowing fuse where you want a "0")

**ROM types**

- **Mask Programmable ROM**
  - Programmed at factory

- **Programmable ROM (PROM)**
  - Shipped with all fuses intact.
  - Special eqpt blows fuses where "0" is wanted.
  - Cannot be changed

- **Erasable PROM (EPROM)**
  - Blown fuses can be restored by exposure to UV light

- **Electrically Erasable PROM (EEPROM or E²PROM)**
  - Blown fuses can be restored electrically (in-socket)
  - Subject to fatigue

- **Flash Memory**
  - Variant of EEPROM
  - Can selectively erase parts of memory
  - Subject to fatigue
A ROM is a minterm generator: we can implement combinational circuits

- **3-bit number**
- **Square of number**

![Block diagram](image)

<table>
<thead>
<tr>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</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>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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 7-12 ROM Implementation of Example 7-1

- **ROM truth table**

G. W. Cox – Spring 2010
Programmable Logic Devices (PLDs)

PLD Types

(a) Programmable read-only memory (PROM)

(b) Programmable array logic (PAL)

(c) Programmable logic array (PLA)
PLA

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

PAL

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

\[ w = ABC' + A'B'CD' \]
\[ x = A + BCD \]
\[ y = A'B + CD + B'D' \]
\[ z = w + AC'D' + A'B'C'D \]

Sequential PLDs

Fig. 7-17 False Map for PAL, as Specified in Table 7-6

G. W. Cox – Spring 2010
Macrocell

Fig. 7-19 Basic Macrocell Logic

Complex PLD (CPLD)

Fig. 7-20 General CPLD Configuration