Lecture 14: CAMs, ROMs, PLAs

Slides courtesy of Deming Chen

Slides based on the initial set from David Harris
Outline

- Content-Addressable Memories
- Read-Only Memories
- Programmable Logic Arrays

Readings: 12.4-12.7
CAMs

- Extension of ordinary memory (e.g. SRAM)
  - Read and write memory as usual
  - Also *match* to see which words contain a key
Add four match transistors to 6T SRAM
- 56 x 43 $\lambda$ unit cell
CAM Cell Operation

- Read and write like ordinary SRAM
- For matching:
  - Leave wordline low
  - Precharge matchlines
  - Place key on bitlines
  - Matchlines evaluate
- Miss line
  - Pseudo-nMOS NOR of match lines
  - Goes high if no words match
Read-Only Memories

- Read-Only Memories are nonvolatile
  - Retain their contents when power is removed
- Mask-programmed ROMs use one transistor per bit
  - Presence or absence determines 1 or 0
ROM Example

- 4-word x 6-bit ROM
  - Represented with dot diagram
  - Dots indicate 1’s in ROM

ROM Array

- Looks like 6 4-input pseudo-nMOS NORs

Word 0: 010101
Word 1: 011001
Word 2: 100101
Word 3: 101010
ROM Array Layout

- Unit cell is $12 \times 8 \lambda$ (about 1/10 size of SRAM)
Row Decoders

- ROM row decoders must pitch-match with ROM
  - Only a single track per word!
Complete ROM Layout
PROMs and EPROMs

- Programmable ROMs
  - Build array with transistors at every site
  - Burn out fuses to disable unwanted transistors

- Electrically Programmable ROMs
  - Use floating gate to turn off unwanted transistors
  - EPROM, EEPROM, Flash
Flash Programming

- Charge on floating gate determines $V_t$
- Logic 1: negative $V_t$
- Logic 0: positive $V_t$
- Cells erased to 1 by applying a high body voltage so that electrons tunnel off floating gate into substrate
- Programmed to 0 by applying high gate voltage

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 V</td>
<td>Erase</td>
</tr>
<tr>
<td>20 V</td>
<td>Inhibit Erase</td>
</tr>
<tr>
<td>20 V</td>
<td>Program 0</td>
</tr>
<tr>
<td>20 V</td>
<td>Do Not Program</td>
</tr>
<tr>
<td>10 V</td>
<td>Inhibit Program</td>
</tr>
<tr>
<td>8 V</td>
<td></td>
</tr>
<tr>
<td>8 V</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td></td>
</tr>
<tr>
<td>0 V</td>
<td></td>
</tr>
<tr>
<td>0 V</td>
<td></td>
</tr>
</tbody>
</table>
NAND Flash

- High density, low cost / bit
  - Programmed one page at a time
  - Erased one block at a time
- Example:
  - 4096-bit pages
  - 16 pages / 8 KB block
  - Many blocks / memory
64 Gb NAND Flash

- 64K cells / page
- 4 bits / cell (multiple $V_t$)
- 64 cells / string
  - 256 pages / block
- 2K blocks / plane
- 2 planes

[Trinh09]
Building Logic with ROMs

- Use ROM as lookup table containing truth table
  - n inputs, k outputs requires $2^n$ words x k bits
  - Changing function is easy – reprogram ROM

- Finite State Machine
  - n inputs, k outputs, s bits of state
  - Build with $2^{n+s} \times (k+s)$ bit ROM and (k+s) bit reg

![ROM Array Diagram]

![Finite State Machine Diagram]
Example: RoboAnt

Let’s build an Ant

**Sensors:** Antennae
(L,R) – 1 when in contact

**Actuators:** Legs
Forward step F
Ten degree turns TL, TR

**Goal:** make our ant smart enough to get out of a maze

**Strategy:** keep right antenna on wall

(RoboAnt adapted from MIT 6.004 2002 OpenCourseWare by Ward and Terman)
Lost in space

- Action: go forward until we hit something
  - Initial state

LOST

L

R

L+R
Bonk!!!

- Action: turn left (rotate counterclockwise)
  - Until we don’t touch anymore

- LOST $F$
  - $L+R$

- RCCW $TL$
  - $L+R$

- $\overline{L} \overline{R}$
A little to the right

- Action: step forward and turn right a little
  - Looking for wall
Then a little to the left

☐ Action: step and turn left a little, until not touching
Whoops – a corner!

☐ Action: step and turn right until hitting next wall
Simplification

- Merge equivalent states where possible
# State Transition Table

<table>
<thead>
<tr>
<th>$S_{1:0}$</th>
<th>L</th>
<th>R</th>
<th>$S_{1:0}'$</th>
<th>TR</th>
<th>TL</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>X</td>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>X</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>X</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
ROM Implementation

- 16-word x 5 bit ROM

![Diagram of a 4:16 decoder for a 16-word x 5 bit ROM](image)

- L, R to ROM
- TL, TR, F from ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>4:16 DEC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
</tr>
<tr>
<td>0101</td>
<td>0110</td>
</tr>
<tr>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>0111</td>
<td>1000</td>
</tr>
<tr>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>1001</td>
<td>1010</td>
</tr>
<tr>
<td>1010</td>
<td>1011</td>
</tr>
<tr>
<td>1011</td>
<td>1100</td>
</tr>
<tr>
<td>1100</td>
<td>1101</td>
</tr>
<tr>
<td>1101</td>
<td>1110</td>
</tr>
<tr>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>
16-word x 5 bit ROM

L, R

ROM

TL, TR, F

S

S'

1:0

S

S'

1:0

4:16 DEC

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

S', S', TR', TL', F'
PLAs

- A *Programmable Logic Array* performs any function in sum-of-products form.
- *Literals*: inputs & complements
- *Products / Minterms*: AND of literals
- *Outputs*: OR of Minterms

- Example: Full Adder

\[
s = a\overline{b}\overline{c} + \overline{a}b\overline{c} + \overline{a}\overline{b}c + abc
\]

\[
c_{\text{out}} = ab + bc + ac
\]
ANDs and ORs are not very efficient in CMOS
Dynamic or Pseudo-nMOS NORs are very efficient
Use DeMorgan’s Law to convert to all NORs
PLAs vs. ROMs

- The OR plane of the PLA is like the ROM array
- The AND plane of the PLA is like the ROM decoder
- PLAs are more flexible than ROMs
  - No need to have $2^n$ rows for $n$ inputs
  - Only generate the minterms that are needed
  - Take advantage of logic simplification
Example: RoboAnt PLA

Convert state transition table to logic equations

<table>
<thead>
<tr>
<th>$S_{1:0}$</th>
<th>L</th>
<th>R</th>
<th>$S'_{1:0}$</th>
<th>TR</th>
<th>TL</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>X</td>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>X</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>X</td>
<td>01</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

$s_{1'} = S_{1}S_0 + L\bar{S}_1 + LRS_0$

$s_{0'} = \bar{S}_{1}S_0 + R + L\bar{S}_1 + L\bar{S}_0$

$TR = S_1\bar{S}_0$

$TL = S_0$

$F = S_1 + \bar{S}_0$
RoboAnt Dot Diagram

\[ S_1' = S_1 S_0' + \overline{L} S_1 + \overline{L} \overline{R} S_0 \]
\[ S_0' = R + L \overline{S_1} + L S_0 \]
\[ TR = S_1 S_0' \]
\[ TL = S_0 \]
\[ F = S_1 + S_0' \]
Summary

- CAM, ROM, PLAs
- FLASH memories
- Building logic with ROMs and PLAs

- Next lecture
  - Circuit pitfalls
  - Readings: 7.1-7.3