Memory & Programmable Logic

Logic and Digital System Design - CS 303
Erkay Savaş
Sabanci University
Memory Unit

• A device
  – to which binary information is transferred for storage and
  – from which information is available when needed for processing

• When data processing takes place
  – information from the memory is transferred to selected register in the processing unit
  – Intermediate and final results obtained in the processing unit are transferred back to the memory for storage
Memory Unit

- Used to communicate with an input/output device
  - binary information received from an input device is stored in memory
  - information transferred to an output device is taken from memory
- A collection of cells capable of storing a large quantity of binary information
- Two types
  - RAM (random-access memory)
  - ROM (read-only memory)
Classification

• RAM
  - Accepts new information for storage (write operation)
  - We can read stored information (read operation)
  - Perform both read and write operation

• ROM
  - only performs read operation
  - existing information cannot be modified
  - a.k.a. programmable logic device
  - Programming the device
    • specifying the binary information and storing it within the programmable device
Programmable Logic Devices

- **PLD**
  - ROM is one example
  - Programmable Logic Array (PLA)
  - Programmable Array Logic (PAL)
  - Field Programmable Gate Array (FPGA)

- **PLD is**
  - an integrated circuit (IC) with internal logic gates
  - Interconnection between the gates can be programmed through fuses
  - At the beginning they are all intact
  - By programming we remove some of them, while keeping the others
Random Access Memory (RAM)

- RAM
- The reason for the name
  - The time it takes to transfer information to or from any desired random location is always the same.
- Word
  - groups of bits in which a memory unit stores information
  - At one time, memory move in and out of the storage a word of information
  - 8 bit - byte
  - 16 bit
  - 32 bit
  - Capacity is usually given in number of bytes
Memory Unit

- Block diagram

![Diagram of Memory Unit]

- $2^k$ words
- $n$ bit per word
- $k$ address lines
- $n$ data input lines
- Read
- Write
- $n$ data output lines
Specification

• A memory unit is specified by
  1. the number of words it contains
  2. number of bits in each word

• Each word in memory is assigned an identification number
  - address
  - 0 to $2^k - 1$

• Selection
  - selection process to read and write a word is done by applying $k$-bit address to the address lines
  - A decoder accepts the address and selects the specified word in the memory
Memory Map and Address Selection

- 1 K x 16 Memory

\[ 1 \text{ K} = 2^{10} \quad 1 \text{ M} = 2^{20} \quad 1 \text{ G} = 2^{30} \quad 1 \text{ T} = 2^{40} \]
Write and Read Operations

• Write
  - transfer in
• Read
  - transfer out
• Steps for write operation
  1. Apply the binary address of the desired word to the address lines
  2. Apply the data bits that must be stored in memory to the (data) input lines
  3. Activate the “write” input
Read Operation

• Steps
  1. Apply the binary address of the desired word to the address lines
  2. Activate the “read” input
     - The desired word will appear on the (data) output lines
     - reading does no affect the content of the word
Control Inputs to Memory Chip

- Commercial memory components usually provide a “memory enable” (or “chip select”) control input.
- Memory enable is used to activate a particular memory chip in a multi-chip implementation of a large memory.

<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</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>read</td>
</tr>
</tbody>
</table>
Timing

• **Memory does not use an internal clock**
  - operation of a memory unit is controlled by an external device (e.g. CPU) that has its own clock
  - It only reacts to the “read” and “write” control inputs

• **Access time**
  - the time required to select a word and read it

• **Cycle time**
  - the time required to complete a write operation
Write Cycle

- CPU clock: 50 Mhz
- Cycle time: 50 ns

CPU must devote three clock cycles for each write operation
Read Cycle

- CPU clock: 50 Mhz
- Access time: 50 ns

The desired word appears at output after 50 ns memory enable is activated
Types of Memory - 1

- **RAM**
  - access time is always the same no matter where the desired data is actually located

- **Sequential-access memory**
  - Access time is variable
  - Magnetic disks, tapes

- **RAM**
  - SRAM (static RAM)
    - latches, stores information as long as power is on
  - DRAM (dynamic RAM)
    - information is stored as charge on a capacitor
    - refreshing is necessary due to discharge
Types of Memory - 2

• Volatile memory
  - When the power is turned off, the stored information is lost
  - RAM

• Nonvolatile memory
  - retains the stored information after removal of power
  - magnetic disks
    • data is represented as the direction of magnetization
  - ROM
    • programs needed to start a computer is kept in ROM
Memory Cell

- Equivalent logic of a memory cell for storing one bit of information
4 x 4 RAM

Address inputs

2x4 decoder

memory enable

EN

word 0

word 1

word 2

word 3

read/write

input data

output data

19
Commercial RAMs

- Physical construction
  - Capacity of thousands of words
  - each word may range from 1 to 64 bits

- Example:
  - We have 1024×4 memory chips
  - Logical construction: 1024×8
Combining Memories

2048×4 RAM

address lines

1024×4

ME

R/W

0

1

9

0

1

9

Memory Enable
Coincident Decoding

- A memory with $2^k$ words requires a $k \times 2^k$ decoder
- $k \times 2^k$ decoder requires $2^k$ AND gates with $k$ inputs per gate
- There are ways to reduce the total number of gates and number of inputs per gate
- Two dimensional selection scheme
  - Arrange the memory words in an array that is close as possible to square
  - Use two $k/2$-input decoders instead of one $k$-input decoder.
    - One decoder performs the row selection
    - The other does the column selection
Example: Coincident Decoding

Binary address:
01100 10100

5x32 decoder
0 1 2 20 31

5x32 decoder
0 1 2
0 12 31
DRAMs

• SRAM memory is expensive
  - One cell typically contains six transistors
  - Usually used for internal registers in CPU and for caches

• DRAM is much less expensive
  - One MOS transistor and a capacitor
  - Four times the density of SRAM in a given chip area
  - Cost per bit storage is three to four times less than SRAM
  - Low power requirement
  - Perfect technology for large memories such as main memory
  - Most DRAMs have a 1-bit word size
**DRAMs and Address Multiplexing**

- In order to reduce number of pins on a memory chip, the same pins are used for both row and column addresses
- Example: 64K DRAM

![Diagram showing the multiplexing process and components](image-url)
Read-Only Memory

• **ROM**
  - memory device in which permanent binary information is stored
  - Binary information must be specified by the designer
  - It then is embedded in the unit to form the required interconnection pattern
  - nonvolatile

• **Block diagram**

```
  k inputs  2^k x n  n outputs
    
  ROM
```

• no data inputs
• enable inputs
• three-state outputs
Example: ROM

- 32 x 8 ROM

Each OR gates are considered as having 32 inputs
Example: ROM

- Number of connections
  - $32 \times 8$ ROM has $32 \times 8 = 256$ internal connections

- In general
  - $2^k \times n$ ROM will have $k \times 2^k$ decoder and $n$ OR gates
  - Each OR gate has $2^k$ inputs
  - OR gate inputs are connected to each of the output of the decoder

- These intersections are programmable
  - these intersections are initially closed (connected to the input of OR gate)
  - A fuse is used to connect two lines
  - During programming, some of these fuses are blown by applying high voltage.
Programming ROM

- Internal storage in specified by a truth table
- Example:

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A&lt;sub&gt;7&lt;/sub&gt;</td>
</tr>
<tr>
<td>I&lt;sub&gt;4&lt;/sub&gt;</td>
<td>I&lt;sub&gt;3&lt;/sub&gt;</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Programming ROM

5x32 decoder

I₀, I₁, I₂, I₃, I₄

0, 1, 2, 3, 28, 29, 30, 31

A₀, A₁, A₂, A₃, A₄, A₅, A₆, A₇
Combinational Circuit Design with ROM

• Formerly,
  - we have shown that a $k \times 2^k$ decoder generates $2^k$ minterms of $k$ input variables

• Furthermore,
  - by inserting OR gates to sum these minterms, we were able to realize any desired combinational circuit.

• A ROM is essentially a device that includes both the decoder and the OR gates within a single device.
  - **first interpretation**: a memory unit that contains a fixed pattern of stored words
Combinational Circuit Design with ROM

• ROM (cont.)
  - Second interpretation: a programmable device that can realize any combinational circuit

• Example: Truth table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_2$ $A_1$ $A_0$</td>
<td>$B_5$ $B_4$ $B_3$ $B_2$ $B_1$ $B_0$</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0 1 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 1 0 0 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1 0 0 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1 1 0 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0 0 1 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 0 0 0 1</td>
</tr>
</tbody>
</table>
Example: Design with ROM

- Observations:
  - \( B_0 = A_0 \) and \( B_1 = 0 \)
  - \( 8 \times 4 \) ROM would suffice

\[
\begin{array}{cccc|cc}
A_2 & A_1 & A_0 & B_5 & B_4 & B_3 & B_2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\end{array}
\]
Types of ROM - 1

• Programming can be done in different ways
  - Mask programming:
    • customer provides the truth table
    • manufacturer generates the mask for the truth table
    • Can be costly, since generating a custom mask is charged to the customer.
    • economical only if a large quantity of the same ROM configuration is to be ordered.
  - Field Programmable
    • Programmable ROM (PROM):
    • Customer can program the ROM by blowing fuses by applying high voltage through a special pin
    • Special instrument called PROM programmer is needed.
Types of ROM - 2

• Programming ROM and PROMs is irreversible.

• Erasable PROM (EPROM)
  - can be programmed repeatedly.
  - EPROM is placed under a special ultra-violet light for a given period of time
  - At the end, the former program is erased
  - After erasure, EPROM becomes ready for another programming

• Electronically erasable PROM (EEPROM or E²PROM)
Programmable Logic Devices

- EPROM is an example of combinational programmable logic device (PLD)
- Configuration of PROM
Other PLDs

- Two other types

Programmable Logic Array (PLA)

Programmable Array Logic (PAL)
Programmable Logic Array (PLA)

• Similar to PROM
• However, PLA does not generate all the minterms
• Decoder is replaced by an array of AND gates
  – can be programmed to generate any product term of input variables
• The product terms are then connected to OR gates
  – that provide the sum of product
Programmable Logic Array (PLA)

- **Example**: 3 inputs, 4 product terms and 2 outputs

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

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

<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>

\[ F_2 = (AC + BC)' \]
Size of PLA

• Specified by
  - number of inputs, number of product, number of outputs

• A typical IC PLA
  - 16 inputs, 48 product terms, and 8 outputs

• n input, k product terms, m output PLA has
  - k AND gates, m OR gates, m XOR gates
  - $2n \times k$ connections between input and the AND array
  - $k \times m$ connections between the AND and OR arrays
  - $2m$ connections associated with XOR gates
Programming PLA

• Optimization
  - number of literals in a product term is not important
  - both the true and complement of each function should be simplified to see which one requires fewer number of product terms
  - When implementing more than one function, functions must be optimized together in order to share more product terms
Example: Programming PLA

- Two functions
  - \( F_1 (A, B, C) = \Sigma(0, 1, 2, 4) \)
  - \( F_2 (A, B, C) = \Sigma(0, 5, 6, 7) \)

\[
\begin{array}{c|c|c|c|c}
A & B & C & \text{Truth Table} \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
\end{array}
\]

\[
F_1 = A'B' + A'C' + B'C' \\
F_1 = (AB + AC + BC)' \\
F_2 = AB + AC + A'B'C' \\
F_2 = (A'C + A'B + AB'C')'
\]
Example: Programming PLA

- PLA programming table

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

<table>
<thead>
<tr>
<th>Product Term</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(\bar{B}\bar{C})</td>
<td>(C)</td>
<td>(T)</td>
</tr>
<tr>
<td>AB</td>
<td>1 (1) (1)</td>
<td>-</td>
</tr>
<tr>
<td>AC</td>
<td>2 (1)</td>
<td>-</td>
</tr>
<tr>
<td>BC</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>A'B'C'</td>
<td>4 (0) (0) (0)</td>
<td>-</td>
</tr>
</tbody>
</table>
Programmable Array Logic (PAL)

- Easier to program than PLA
- But, not as flexible
- A typical PAL
  - 8 inputs, 8 outputs, 8-wide AND-OR array
Example: PAL

AND gates inputs

I₁ → 1 2 3 4 5 6 7 8 9 10
I₂ → 1 2 3 4 5 6 7 8 9 10
I₃ → 1 2 3 4 5 6 7 8 9 10
I₄ → 1 2 3 4 5 6 7 8 9 10

product term

F₁ → 1 2 3 4 5 6 7 8 9 10
F₂ → 1 2 3 4 5 6 7 8 9 10
F₃ → 1 2 3 4 5 6 7 8 9 10
F₄ → 1 2 3 4 5 6 7 8 9 10
Design with PAL

• Each Boolean function must be simplified to fit into each section.
• Product terms cannot be shared among OR gates
  – Each function can be simplified by itself without regard to common product terms
• The number of product terms in each section is fixed
  – If the number of product terms is too many, we may use two sections to implement it.
Example: Design with PAL

- **Four functions**
  - $A(x, y, z, t) = \Sigma (2, 12, 13)$
  - $B(x, y, z, t) = \Sigma (7, 8, 9, 10, 11, 12, 13, 14, 15)$
  - $C(x, y, z, t) = \Sigma (0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$
  - $D(x, y, z, t) = \Sigma (1, 2, 8, 12, 13)$
- **First step is to simplify four functions separately**
  - $A = xyz' + x'y'zt'$
  - $B = x + yzt$
  - $C = x'y + zt + y't'$
  - $D = xyz' + x'y'zt' + xy't' + x'y'z't'$
Example: Design with PAL

\[- D = A + xy't' + x'y'z't \]

<table>
<thead>
<tr>
<th>Product Term</th>
<th>AND Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x  y  z  t  A</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1  1  1  0  -  -</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0  0  1  1  0  -</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-  -  -  -  -  -</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1  -  -  -  -  -</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>-  1  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></td>
</tr>
<tr>
<td>8</td>
<td>-  -  1  1  1  -</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-  0  -  0  0  -</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-  -  -  -  -  1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1  0  -  0  0  -</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0  0  0  1  -  -</td>
<td></td>
</tr>
</tbody>
</table>

\[A = xyz' \]
\[B = x + yz't \]
\[C = x'y + zt + y't' \]
\[D = A + xy't' + x'y'z't \]
Example: Design with PAL

AND gates inputs

x
y
z
\( x' \)
\( y' \)
\( z' \)
\( t' \)
\( A \)
\( A' \)
Sequential Programmable Devices

- So far, we have seen PLD that can realize only combinational circuits
- However, digital systems are designed using both combinational circuits (gates) and flip-flops.
  - With PLDs, we need to use external flip-flops to realize sequential circuit functions.
- Different types
  1. Sequential (or simple) programmable logic device (SPLD)
  2. Complex programmable logic device (CPLD)
  3. Field programmable gate array (FPGA)
Sequential Programmable Logic Device (SPLD)

- Additional programmable connections are available to include flip-flop outputs in the product terms formed with AND array.
- Flip-flops may be of the D or the JK type.
SPLD Macrocell

- SPLD is usually PAL + D flip-flops
- Each section in SPLD is called **macrocell**.
- A macrocell
  - sum-of-products combinational logic + optional flip-flop
  - 8-10 macrocells in one IC package
Additional SPLD Functionalities

• Additional SPLD Functionalities
  - Bypass circuitry for the flip-flop
  - selection of clock edge polarity
  - XOR gate for selection of true or complement of output

• Example: serial adder
  - Flip-flop input equation:
    • \( D = Q(t+1) = xy + xQ + yQ \)
  - Output equation:
    • \( S = x \oplus y \oplus Q \)
    • \( S = x'y'Q + x'yQ' + xyQ + xy'Q' \)
Example: Serial Adder with SPLD

\[ Q(t+1) = xy + xQ + yQ \]
Example: Serial Adder with SPLD

\[ S = x'y'Q + x'yQ' + xyQ + xy'Q' \]
Complex Programmable Logic Device

IO block

programmable switch matrix

IO block

PLD

PLD

PLD

PLD

PLD

PLD

PLD

PLD
FPGA

- Field programmable gate array
  - FPGA is a VLSI circuit
  - **Field programmable** means user can program it in his own location
  - **Gate array** consists of a pattern of gates fabricated in an area of silicon
    - pattern of gates are repeated many (thousand) times
  - one thousand to hundreds of thousands of gates are fabricated within a single IC chip
Basics of FPGA
Basics of FPGA

• A typical FPGA consists of an array of hundreds or thousands of configurable logic blocks (CLB)
  - CLBs are connected to each other via programmable interconnections
• CLBs are surrounded by I/O blocks for basic communication with outside world.
• CLBs consists of look-up tables, multiplexers, gates, and flip flops
• look-up table
  - is a truth table stored in a SRAM
  - provides the combinational circuit functions for the logic block.
  - It is like a ROM implemented as SRAM
Xilinx XC 4000 CLB

Logic Function of $G_1 - G_4$

Logic Function of $F', G', H_1$

Logic Function of $F_1 - F_4$

$K$ (clock)

$x_1 \ldots x_4$

$H_1$

$D_{IN}/H_2$

$S/R/EC$

$D$ $Q$

$SD$

$RD$

$yQ$

$xQ$
Xilinx XC 4000 CLB

Main components in XC 4000 CLB
- Two $16 \times 1$ RAM blocks
- One $8 \times 1$ RAM block
  * RAM can be used as LUT to implement combinational logic through truth tables
- Two edge-triggered D flip-flops
  * Can also be configured as latches
- Several multiplexers to configure the interconnection within the CLB.
LUTs in FPGA

- Using three function generators F, G, and H
  - A single CLB can implement any Boolean function of five variables and some functions of up to nine variables
  - It has nine logic inputs: $F_1, F_2, F_3, F_4, G_1, G_2, G_3, G_4$, and $H_1$.
  - General form of Boolean function of five variables:
    $$H = F(F_1, F_2, F_3, F_4) \cdot H_1' + G(F_1, F_2, F_3, F_4) \cdot H_1$$

- In some 4000 CLBs, LUTs can be configured as memory block
  - One $32 \times 1$ module
  - Dual ported $16 \times 1$ memory module
Design with Programmable Devices

- Requires CAD tools
- Entry tools
  - For entering a design
  - Schematic entry package
  - Hardware description languages (HDL)
    - VHDL, Verilog, ABEL,
- Synthesis tools
  - Allocate
  - Configure
  - Connect logic blocks