L7: Memory Basics and Timing

Acknowledgement: Nathan Ickes, Rex Min

## Memory Classification & Metrics

### Key Design Metrics:
1. Memory Density (number of bits/μm²) and Size
2. Access Time (time to read or write) and Throughput
3. Power Dissipation

<table>
<thead>
<tr>
<th>Read-Write Memory</th>
<th>Non-Volatile Read-Write Memory</th>
<th>Read-Only Memory (ROM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Access</td>
<td>Non-Random Access</td>
<td>EROM</td>
</tr>
<tr>
<td>SRAM</td>
<td>FIFO</td>
<td>LIFO</td>
</tr>
<tr>
<td>DRAM</td>
<td></td>
<td>Mask-Programmed</td>
</tr>
</tbody>
</table>
Memory Array Architecture

- **2^{L-K} row by Mx2^K column cell array**
- Row Decode: Selects appropriate word (i.e., multiplexor)
- Column Decode: Amplify swing to rail-to-rail amplitude
- Sense Amps/Driver: Input-Output (M bits)
- Storage Cell: Word Line, Bit Line
- Amplify swing to rail-to-rail amplitude
- Selects appropriate word (i.e., multiplexor)
Latch and Register Based Memory

Positive Latch  Negative Latch

- Works fine for small memory blocks (e.g., small register files)
- Inefficient in area for large memories – density is the key metric in large memory circuits

Register Memory

How do we minimize cell size?
Static RAM (SRAM) Cell (The 6-T Cell)

- State held by cross-coupled inverters (M1-M4)
- Retains state as long as power supply turned on
- Feedback must be overdriven to write into the memory

**Write:** set BL and \( \overline{BL} \) to 0 and \( V_{DD} \) or \( V_{DD} \) and 0 and then enable WL (i.e., set to \( V_{DD} \))

**Read:** Charge BL and \( BL \) to \( V_{DD} \) and then enable WL (i.e., set to \( V_{DD} \)). Sense a small change in BL or \( \overline{BL} \)
Interacting with a Memory Device

- **Address pins** drive row and column decoders
- **Data pins** are bidirectional and shared by reads and writes

**Output Enable** gates the chip’s tristate driver

**Write Enable** sets the memory’s read/write mode

**Chip Enable/Chip Select** acts as a “master switch”
MCM6264C 8k x 8 Static RAM

On the outside:
- Same (bidirectional) data bus used for reading and writing
- Chip Enables (\(E_1\) and \(E_2\))
  - \(E_1\) must be low and \(E_2\) must be high to enable the chip
- Write Enable (\(W\))
  - When low (and chip is enabled), the values on the data bus are written to the location selected by the address bus
- Output Enable (\(G\))
  - When low (and chip is enabled), the data bus is driven with the value of the selected memory location

On the inside:
- Memory matrix 256 rows 32 Column
- Row Decoder
- Column Decoder
- Sense Amps/Drivers

Pinout
- \(E_1\) E2
- \(W\)
- \(G\)

[Diagram of MCM6264C 8k x 8 Static RAM with pinouts and connections]
Reading an Asynchronous SRAM

- Read cycle begins when all enable signals (\(E1, E2, G\)) are active.
- Data is valid after read access time.
  - Access time is indicated by full part number: \(MCM6264CP-12 \rightarrow 12\)ns
- Data bus is tristated shortly after \(G\) or \(E1\) goes high.

\[\begin{array}{cccc}
\text{Address} & \text{Address Valid} & \text{Access time (from address valid)} & \text{Access time (from enable low)} \\
\hline
\overline{E1} & & \text{Access time (from enable low)} & \text{Bus enable time} \\
\overline{G} & & \text{Bus enable time} & \text{Bus tristate time} \\
\text{Data} & (\text{Tristate}) & \text{Data Valid} \\
\end{array}\]

\(E2\) assumed high (enabled), \(W = 1\) (read mode)
Address Controlled Reads

- Can perform multiple reads without disabling chip
- Data bus follows address bus, after some delay

Access time (from address valid)
Contamination time

Bus enable time
Bus tristate time

E1 assumed high (enabled), \( \overline{W} = 1 \) (read mode)
Data latched when $\overline{W}$ or $E1$ goes high (or $E2$ goes low)
- Data must be stable at this time
- Address must be stable before $\overline{W}$ goes low

Write waveforms are more important than read waveforms
- Glitches to address can cause writes to random addresses!
Drive data bus only when clock is low
- Ensures address and are stable for writes
- Prevents bus contention
- Minimum clock period is twice memory access time
Multi-Cycle Read/Write
(less aggressive, recommended timing)
Simulation from Previous Slide

write completes

read, address is stable

address/data stable

Data latched into FPGA

write states 1-3

read states 1-3
module memtest (clk, reset, G_b, W_b, address, ext_address, write_data, read_data, ext_data, read, write, state, data_oen, address_load, data_sample);

input clk, reset, read, write;
output G_b, W_b;
output [12:0] ext_address;
reg [12:0] ext_address;
input [12:0] address;
input [7:0] write_data;
output [7:0] read_data;
reg [7:0] read_data;
inout [7:0] ext_data;
reg [7:0] int_data;
output [2:0] state;
reg [2:0] state, next;
output data_oen, address_load, data_sample;
reg G_b, W_b, G_b_int, W_b_int, address_load, data_oen, data_oen_int, data_sample;

wire [7:0] ext_data;
parameter IDLE = 0;
parameter write1 = 1;
parameter write2 = 2;
parameter write3 = 3;
parameter read1 = 4;
parameter read2 = 5;
parameter read3 = 6;

// Sequential always block for state assignment
assign ext_data = data_oen ? int_data : 8'hz;
always @ (posedge clk)
begin
if (!reset) state <= IDLE;
else state <= next;

G_b <= G_b_int;
W_b <= W_b_int;
data_oen <= data_oen_int;
if (address_load) ext_address <= address;
if (data_sample) read_data <= ext_data;
if (address_load) int_data <= write_data;
end

// note that address_load and data_sample are not registered signals
// Combinational always block for next-state
// computation
always @ (state or read or write) begin
    W_b_int = 1;
    G_b_int = 1;
    address_load = 0;
    data_oen_int = 0;
    data_sample = 0;
    case (state)
        IDLE: if (write) begin
                next = write1;
                address_load = 1;
                data_oen_int = 1;
            end
            else if (read) begin
                next = read1;
                address_load = 1;
                G_b_int = 0;
            end
            else next = IDLE;
        write1: begin
                next = write2;
                W_b_int = 0;
                data_oen_int = 1;
            end
        write2: begin
                next = write3;
                data_oen_int = 1;
            end
        write3: begin
                next = IDLE;
                data_oen_int = 0;
            end
        read1: begin
                next = read2;
                G_b_int = 0;
                data_sample = 1;
            end
        read2: begin
                next = read3;
            end
        read3: begin
                next = IDLE;
            end
        default: next = IDLE;
    endcase
end
endmodule
Testing Memories

- **Common device problems**
  - Bad locations: *rare for individual locations to be bad*
  - Slow (out-of-spec) timing(s): *access incorrect data or violates setup/hold*
  - Catastrophic device failure: *e.g., ESD*
  - Missing wire-bonds/devices (!): *possible with automated assembly*
  - Transient Failures: *Alpha particles, power supply glitch*

- **Common board problems**
  - Stuck-at-Faults: *a pin shorted to V_{DD} or GND*
  - Open Circuit Fault: *connections unintentionally left out*
  - Open or shorted address wires: *causes data to be written to incorrect locations*
  - Open or shorted control wires: *generally renders memory completely inoperable*

- **Approach**
  - Device problems generally affect the entire chip, almost any test will detect them
  - Writing (and reading back) many different data patterns can detect data bus problems
  - Writing unique data to every location and then reading it back can detect address bus problems
An Approach

- An idea that almost works
  1. Write 0 to location 0
  2. Read location 0, compare value read with 0
  3. Write 1 to location 1
  4. Read location 1, compare value read with 1
  5. …

- What is the problem?
  - Suppose the memory was missing (or output enable was disconnected)

Data bus is undriven but wire capacitance briefly maintains the bus state: memory appears to be ok!
A Simple Memory Tester

- Write to all locations, then read back all locations
  - Separates read/write to the same location with reads/writes of different data to different locations
  - (both data and address busses are changed between read and write to same location)

- SRAM

  - Data
  - Address
  - Control

- Counter
  - Reset counter
  - Read address <counter>
  - Compare data read with 8-LSB's of <counter>
  - Increment counter

- Comparator
  - Does not match?
  - Matched?

- Enable memory test
  - Write 0 to address 0
  - Write 1 to address 1
  - ...
  - Write \((n \mod 256)\) to address \(n\)
  - Read address 0, compare with 0
  - Read address 1, compare with 1
  - ...
  - Read address \(n\), compare with \((n \mod 256)\)
Clocking provides input synchronization and encourages more reliable operation at high speeds.

The difference between read and write timings creates wasted cycles ("wait states").

Long "flow-through" combinational path creates high CLK-Q delay.

Clocking

Address Pins

Row Decoder

Memory matrix

Sense Amps/Drivers

Column Decoder

Write Logic

Write Enable Chip Enable

Read Logic

Output Enable

Data Pins

Address

Data

W_3

R_4

W_5

R_2

R_1

A_1

A_2

A_3

A_4

A_5

Q_1

Q_2

D_3

Q_4

D_5

CE

WE

CLK
The wait state occurs because:
- On a read, data is available after the clock edge
- On a write, data is set up before the clock edge

ZBT (“zero bus turnaround”) memories change the rules for writes
- On a write, data is set up after the clock edge (so that it is read on the following edge)
- Result: no wait states, higher memory throughput
Pipelining Allows Faster CLK

- Pipeline the memory by registering its output
  - Good: Greatly reduces CLK-Q delay, allows higher clock (more throughput)
  - Bad: Introduces an extra cycle before data is available (more latency)

As an example, see the CY7C147X ZBT Synchronous SRAM
EPROM Cell – The Floating Gate Transistor

Avalanche injection

Removing programming voltage leaves charge trapped

Programming results in higher $V_T$.

This is a non-volatile memory (retains state when supply turned off)
Reading from flash or (E)EPROM is the same as reading from SRAM

Vpp: input for programming voltage (12V)
- EPROM: Vpp is supplied by programming machine
- Modern flash/EEPROM devices generate 12V using an on-chip charge pump

EPROM lacks a write enable
- Not in-system programmable (must use a special programming machine)

For flash and EEPROM, write sequence is controlled by an internal FSM
- Writes to device are used to send signals to the FSM
- Although the same signals are used, one can’t write to flash/EEPROM in the same manner as SRAM
Dynamic RAM (DRAM) Cell

To Write: set Bit Line (BL) to 0 or \( V_{DD} \) & enable Word Line (WL) (i.e., set to \( V_{DD} \))

To Read: set Bit Line (BL) to \( V_{DD} / 2 \) & enable Word Line (i.e., set it to \( V_{DD} \))

- DRAM relies on charge stored in a capacitor to hold state
- Found in all high density memories (one bit/transistor)
- Must be “refreshed” or state will be lost – high overhead
Clever manipulation of RAS and CAS after reads/writes provide more efficient modes: early-write, read-write, hidden-refresh, etc. (See datasheets for details)
‘138 is a 3-to-8 decoder
- Maps 16-bit address space to 8, 13-bit segments
- Upper 3-bits of address determine which chip is enabled

SRAM-like interface is often used for peripherals
- Referred to as “memory mapped” peripherals

Addressing with Memory Maps

Memory Map

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>EPROM</td>
</tr>
<tr>
<td>0xE000</td>
<td>SRAM 2</td>
</tr>
<tr>
<td>0xDFFF</td>
<td>SRAM 1</td>
</tr>
<tr>
<td>0xC000</td>
<td>ADC</td>
</tr>
<tr>
<td>0xBF00</td>
<td>ADC</td>
</tr>
<tr>
<td>0xA000</td>
<td>ADC</td>
</tr>
<tr>
<td>0x9FFF</td>
<td>ADC</td>
</tr>
<tr>
<td>0x2000</td>
<td>ADC</td>
</tr>
<tr>
<td>0x1FFF</td>
<td>ADC</td>
</tr>
<tr>
<td>0x0000</td>
<td>ADC</td>
</tr>
</tbody>
</table>

SRAM 1
- G
- W
- E1
- Address[12:0]
- Data[7:0]

SRAM 2
- G
- E1
- Address[12:0]
- Data[7:0]

EPROM
- Address[12:0]
- Data[7:0]

ADC
- Address[12:0]
- Data[7:0]

138 is a 3-to-8 decoder that maps a 16-bit address space to 8, 13-bit segments. The upper 3-bits of the address determine which chip is enabled. SRAM-like interfaces are often used for peripherals and are referred to as “memory mapped” peripherals.
Key Messages on Memory Devices

SRAM vs. DRAM
- SRAM holds state as long as power supply is turned on. DRAM must be “refreshed” – results in more complicated control.
- DRAM has much higher density, but requires special capacitor technology.
- FPGA usually implemented in a standard digital process technology and uses SRAM technology.

Non-Volatile Memory
- Fast Read, but very slow write (EPROM must be removed from the system for programming!)
- Holds state even if the power supply is turned off.

Memory Internals
- Has quite a bit of analog circuits internally -- pay particular attention to noise and PCB board integration.

Device details
- Don’t worry about them, wait until 6.012 or 6.374.