L7: Memory Basics and Timing

Acknowledgement: Nathan Ickes, Rex Min

### Memory Classification & Metrics

**Key Design Metrics:**

1. **Memory Density** (number of bits/\(\mu\text{m}^2\)) and Size
2. **Access Time** (time to read or write) and Throughput
3. **Power Dissipation**

![Memory Classification Table]

<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>EPROM (E^2\text{PROM})</td>
</tr>
<tr>
<td>SRAM</td>
<td>FIFO</td>
<td>FLASH</td>
</tr>
<tr>
<td>DRAM</td>
<td>LIFO</td>
<td></td>
</tr>
</tbody>
</table>
Memory Array Architecture

- **Row Decode**: $A_0$ to $A_{L-1}$
- **Word Line**: $A_K$ to $A_{K+1}$
- **Bit Line**: $2^{L-K}$
- **Storage Cell**: $2^{L-K}$ row by $M\times2^K$ column cell array
- **Sense Amps/Driver**: Amplify swing to rail-to-rail amplitude
- **Column Decode**: Selects appropriate word (i.e., multiplexor)
- **Input-Output (M bits)**
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

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 BL̅ to 0 and $V_{DD}$ or $V_{DD}$ and 0 and then enable WL (i.e., set to $V_{DD}$)

Read: Charge BL and BL̅ to $V_{DD}$ and then enable WL (i.e., set to $V_{DD}$). Sense a small change in BL or BL̅.
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 with \(W=0\)), the data bus is driven with the value of the selected memory location

On the inside:

- Memory matrix
  - 256 rows
  - 32 columns
- Row Decoder
- Column Decoder
- Sense Amps/Drivers

Pinout Table:

<table>
<thead>
<tr>
<th>Pin</th>
<th>Number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>10</td>
<td>DQ0</td>
</tr>
<tr>
<td>A1</td>
<td>11</td>
<td>DQ1</td>
</tr>
<tr>
<td>A2</td>
<td>12</td>
<td>DQ2</td>
</tr>
<tr>
<td>A3</td>
<td>13</td>
<td>DQ3</td>
</tr>
<tr>
<td>A4</td>
<td>14</td>
<td>VSS</td>
</tr>
<tr>
<td>A5</td>
<td>15</td>
<td>DC3</td>
</tr>
<tr>
<td>A6</td>
<td>16</td>
<td>DC6</td>
</tr>
<tr>
<td>A7</td>
<td>17</td>
<td>DC5</td>
</tr>
<tr>
<td>A8</td>
<td>18</td>
<td>DQ4</td>
</tr>
<tr>
<td>A9</td>
<td>19</td>
<td>DC7</td>
</tr>
<tr>
<td>A10</td>
<td>20</td>
<td>E1</td>
</tr>
<tr>
<td>A11</td>
<td>21</td>
<td>E2</td>
</tr>
<tr>
<td>A12</td>
<td>22</td>
<td>W</td>
</tr>
<tr>
<td>A13</td>
<td>23</td>
<td>G</td>
</tr>
<tr>
<td>A14</td>
<td>24</td>
<td>A1</td>
</tr>
<tr>
<td>A15</td>
<td>25</td>
<td>A8</td>
</tr>
<tr>
<td>A16</td>
<td>26</td>
<td>A2</td>
</tr>
<tr>
<td>A17</td>
<td>27</td>
<td>A3</td>
</tr>
<tr>
<td>A18</td>
<td>28</td>
<td>VCC</td>
</tr>
<tr>
<td>A19</td>
<td>29</td>
<td>NC</td>
</tr>
</tbody>
</table>

Address

Data

DQ[7:0]
Reading an Asynchronous SRAM

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

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

- 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!

<table>
<thead>
<tr>
<th>Address Valid</th>
<th>Address setup time</th>
<th>Address hold time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E1 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( W )</td>
<td>Write pulse width</td>
<td></td>
</tr>
<tr>
<td>Data Valid</td>
<td>Data setup time</td>
<td>Data hold time</td>
</tr>
</tbody>
</table>

\( E2 \) and \( G \) are held high
**Sample Memory Interface Logic**

- Drive data bus only when clock is low
  - Ensures address are stable for writes
  - Prevents bus contention
  - Minimum clock period is twice memory access time

Write cycle

Read cycle

- **Clock/E1**
- **G**
- **W**

**Address**

- Address for write
- Address for read

**Data**

- Data for write
- Data read

Write occurs here, when E1 goes high

Data can be latched here

- **Drive data bus only**
- **Ensure address are stable for writes**
- **Prevent bus contention**
- **Minimum clock period is twice memory access time**

---

FPGA

- Clock
- Control (write, read, reset)

SRAM

- E2
- E1
- W
- G

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

---

- Drive data bus only when clock is low
- Ensures address are stable for writes
- Prevents bus contention
- Minimum clock period is twice memory access time
Multi-Cycle Read/Write
(less aggressive, recommended timing)

Control
(write, read, reset)

FSM

clk

W_b

G_b

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

SRAM

write states 1-3

write completes

read, address is stable

address/data stable

Data latched into FPGA

read states 1-3

write states 1-3
Simulation from Previous Slide

- Write states 1-3
- 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;
assign ext_data = data_oen ? int_data : 8'hz;
// Sequential always block for state assignment
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
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)

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Write</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Read</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Write</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Read</td>
</tr>
</tbody>
</table>

*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)

- Counter
  - Reset counter
  - Read address \(<\text{counter}>\)
  - Compare data read with 8-LSB’s of \(<\text{counter}>\)
    - Increment counter

- Comparator
  - \(<\text{counter}>\) = last address?
  - Matched?
    - Yes
      - Report success
    - No
      - Report failure

- SRAM
  - 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 \((n \mod 256)\)

- Enable memory test

- To normal memory interface

- Data
- Address
- Control

- \(<\text{counter}>\) = last address?
Clocking provides input synchronization and encourages more reliable operation at high speeds.

Synchronous SRAM Memories

- **Memory matrix**
- **Address Pins**
- **Data Pins**
- **Row Decoder**
- **Column Decoder**
- **Sense Amps/Drivers**
- **Write Logic**
- **Read Logic**
- **Write Enable Chip Enable**
- **Output Enable**

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

**long “flow-through” combinational path creates high CLK-Q delay**

**Address**

- \( A_1 \)
- \( A_2 \)
- \( A_3 \)
- \( A_4 \)
- \( A_5 \)

**Data**

- \( Q_1 \)
- \( Q_2 \)
- \( D_3 \)
- \( Q_4 \)
- \( D_5 \)
ZBT Eliminates the Wait State

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)

Courtesy Intel

[Rabaey03]
Interacting with Flash and (E)EPROM

- 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

Flash/EEPROM block diagram:

- Vcc (5V)
- Address
- Data
- Chip Enable
- Output Enable
- Write Enable
- FSM
- Programming voltage (12V)
- Charge pump
- EPROM omits FSM, charge pump, and write enable
Dynamic RAM (DRAM) Cell

**Dynamic RAM (DRAM) Cell**

**WL**  
**BL**  
**M1**  
**Cs**  
**CBL**  

**Write "1"**  
**Read "1"**  

**WL**  
**X**  
**GND**  
**BL**  
**VDD/2**  
**VDD/2**  

**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

[Image of DRAM cell with labels and circuit diagram]

**[Rabaey03]**

**Capacitor Insulator**  
**Cell Plate Si**  
**Si Substrate**  
**Refilling Poly**  
**Storage Node Poly**  
**2nd Field Oxide**
Asynchronous DRAM Operation

Clever manipulation of RAS and CAS after reads/writes provide more efficient modes: early-write, read-write, hidden-refresh, etc. (See datasheets for details)

- RAS-before-CAS for a read or write
  (Row and column addresses taken on falling edges of RAS and CAS)
- CAS-before-RAS for a refresh
Addressing with Memory Maps

- ‘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

---

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>0xBFFFF</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

- Address[12:0]
- Data[7:0]
- Write Enable (~W)
- Output Enable (~G)
- Bus Enable (~E1)

SRAM 2

- Address[12:0]
- Data[7:0]
- Write Enable (~W)
- Output Enable (~G)
- Bus Enable (~E1)

EPROM

- Address[12:0]
- Data[7:0]
- Write Enable (~W)
- Output Enable (~G)
- Bus Enable (~E1)

ADC

- Address[12:0]
- Data[7:0]
- Write Enable (~W)
- Output Enable (~G)
- Bus Enable (~E1)

---

Analysis:

- The ‘138 is a 3-to-8 decoder that maps 16-bit address space to 8, 13-bit segments.
- The upper 3-bits of address determine which chip (SRAM or EPROM) is enabled.

- SRAM-like interface is often used for peripherals, referred to as “memory mapped” peripherals.

Diagram:

- A diagram showing the connections and signals between the memory components (SRAM 1, SRAM 2, EPROM, ADC).
- The diagram includes address signals (Address[15:0]), data signals (Data[7:0]), enable signals (~W, ~G, ~E1), and power supply (+5V).

---

L7: 6.111 Spring 2005
Introductory Digital Systems Laboratory
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.
control signals such as *Write Enable* should be registered

a multi-cycle read/write is safer from a timing perspective than the single cycle read/write approach

it is a bad idea to enable two tri-states driving the bus at the same time

an SRAM does not need to be “refreshed” while a DRAM does

an EPROM/EEPROM/FLASH cell can hold its state even if the power supply is turned off

a synchronous memory can result in higher throughput