L7: Memory Basics and Timing

Acknowledgements:

- Nathan Ickes, Rex Min, Yun Wu


- Lecture notes prepared by Professor Anantha Chandrakasan
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
Memory Array Architecture

- **Row Decode**: $A_k, A_{k+1}, \ldots, A_{L-1}$
- **Word Line**: $2^{L-K}$
- **Bit Line**: Storage Cell
- **Column Decode**: $A_0, A_{K-1}$
- **Sense Amps/Driver**: Amplify swing to rail-to-rail amplitude
- **Selects appropriate word (i.e., multiplexor)**

$2^{L-K}$ row by $M \times 2^K$ column cell array

Input-Output (M bits)
Latch and Register Based Memory

Positive Latch

Negative Latch

Register Memory

- 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)
- Static Memory - 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 (\(\overline{E1}\) and \(E2\))
  - \(\overline{E1}\) must be low and \(E2\) must be high to enable the chip
- Write Enable (\(\overline{W}\))
  - When low (and chip is enabled), the values on the data bus are written to the location selected by the address bus
- Output Enable (\(\overline{G}\))
  - 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 Column

Pinout

L7: 6.111 Spring 2009
Introductory Digital Systems Laboratory
Reading an Asynchronous SRAM

- Read cycle begins when all enable signals ($\overline{E1}$, $E2$, $G$) are active.

- Data is valid after read access time:
  - Access time is indicated by full part number: $MCM6264CP-12 \rightarrow 12\text{ns}$

- Data bus is tristated shortly after $G$ 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 \( E_1 \) goes high (or \( E_2 \) 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!
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 occurs here, when $E_1$ goes high
Data can be latched here

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

SRAM

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

CLK
reset
write
read
W_b
G_b
address[3:0]
ext_address...
write_data[3:0]
data_oen
ext_data[3:0]
read_data[3:0]
state[2:0]
data_sample
address_load

write completes
read, address is stable
address/data stable
Data latched into FPGA
write states 1-3
read states 1-3
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

Now: 6000 ns
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;
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;

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
Verilog for Simple Multi-Cycle Access

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

- 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").

A long "flow-through" combinational path creates high CLK-Q delay.
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)

MacBook Air – 128GB solid state drive

Courtesy Apple
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
Dynamic RAM (DRAM) Cell

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

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 $V_{DD}$)

[Rabaey03]
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)

<table>
<thead>
<tr>
<th>Address</th>
<th>Row</th>
<th>Col</th>
</tr>
</thead>
</table>

- **RAS**
  - **RAS-before-CAS**: for a read or write
  - (Row and column addresses taken on falling edges of RAS and CAS)

- **CAS**
  - **CAS-before-RAS**: for a refresh
  - \( Q \) (data from RAM)
  - \( WE \) (set high/low before asserting CAS)

- **Data**
  - (Tristate)
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>0xBFFF</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>

---

**Diagram**

- ‘138 decoder schematic
- Address[15:0], Data[7:0] connections
- Output Enable, Write Enable, Bus Enable
- Memory map layout
- SRAM 1, SRAM 2, EPROM, ADC interfaces

---

L7: 6.111 Spring 2009 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.
You Should Understand Why…

- 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