Today: Caches and the Memory System

Static Memory: Used in cache designs.

Memory Hierarchy: Technology motivation for caching.
Static Memory Circuits

Dynamic Memory: Circuit remembers for a fraction of a second.

Static Memory: Circuit remembers as long as the power is on.

Non-volatile Memory: Circuit remembers for many years, even if power is off.
Preliminaries
Inverters: Building block for SRAM

\[ V_{in} \rightarrow V_{out} \]

Symbol for $V_{dd}$
Inverter: Die Cross Section

V_{\text{in}} \rightarrow \text{V}_{\text{out}}
Recall: Our simple inverter model ...

Correctly predicts logic output for simple static CMOS circuits.

Extensions to model subtler circuit families, or to predict timing, have not worked well ...

\[
\begin{array}{c|c}
\text{In} & \text{Out} \\
0 & 1 \\
1 & 0 \\
\end{array}
\]

pmFET. A switch. “On” if gate is grounded.

nmFET. A switch. “On” if gate is at Vdd.

Inverter

\[
\text{Out} = \overline{\text{In}}
\]

Circuit

\[
\begin{align*}
\text{PMOS} & \quad \text{Vdd} \\
\text{NMOS} & \quad \text{NM} \\
\text{Out} & = \begin{cases} 
\text{“1”} & \text{In} = \text{“0”} \\
\text{“0”} & \text{In} = \text{“1”} \\
\end{cases}
\end{align*}
\]
When the 0/1 model is too simple ...

We wire the output of the inverter to drive its input. What happens?

Logic simulators based on our too-simple model predict this circuit will oscillate! This prediction is incorrect.

In reality, $V_{in} = V_{out}$ settles to a stable value, defined as $V_{th}$, where nFET and pFET current match.

Can we figure out $V_{th}$ without solving tedious equations?
Graphical equation solving ...

Vth

\[ \begin{align*}
V_{\text{in}} &\rightarrow V_{\text{out}} \\
\downarrow I_{\text{sd}} &\downarrow I_{\text{ds}} \\
&= 25 \text{ nm}
\end{align*} \]

Intersection defines Vth

Note: Ignores second-order effects.

Recall: Graphs from power and energy lecture ...
Recall: Transistors as water valves

If electrons are water molecules, transistor strengths (W/L) are pipe diameters, and capacitors are buckets ...

A "on" p-FET fills up the capacitor with charge.

A "on" n-FET empties the bucket.
What happens when we break tie wire?

Small amounts of noise on Vin causes $I_{ds} > I_{sd}$ or $I_{sd} > I_{ds}$ ... and output bucket randomly fills and empties.

Result: $V_{out}$ randomly flips between logic 0 and logic 1.
SRAM

1971 state of the art.

Intel 2102, a 1kb, 1 MHz static RAM chip with 6000 nFETs transistors in a 10 μm process.
Recall DRAM cell: 1 T + 1 C

"Word Line"
"Row"

"Column"

"Row"

"Bit Line"

"Column"

Vdd

Word Line

"Bit Line"
Idea: Store each bit with its complement

Why?

We can use the redundant representation to compensate for noise and leakage.
Case #1: \( y = \text{Gnd}, \overline{y} = \text{Vdd} \ldots \)
Case #2: \( y = Vdd, \overline{y} = Gnd \) ...
Combine both cases to complete circuit

"Cross-coupled inverters"
SRAM Challenge #1: It’s so big!

SRAM area is 6-10X DRAM area, same generation ...

Cell has both transistor types

Capacitors are usually "parasitic" capacitance of wires and transistors.

More contacts, more devices, two bit lines ...

Vdd AND Gnd
Intel SRAM core cell (45 nm)

Word Lines

Bit Lines

Thursday, February 20, 14
Challenge #2: Writing is a “fight”

When word line goes high, bitlines “fight” with cell inverters to “flip the bit” -- must win quickly!
Solution: tune W/L of cell & driver transistors

Initial state

Initial state

Bitline drives Gnd

Bitline drives Vdd
Challenge #3: Preserving state on read

When word line goes high on read, cell inverters must drive large bitline capacitance quickly, to preserve state on its small cell capacitances.
SRAM array: like DRAM, but non-destructive

Architects specify number of rows and columns. Word and bit lines slow down as array grows larger!

For large SRAMs: Tile a small array, connect with muxes, decoders.
SRAM vs DRAM, pros and cons

- **DRAM** has a 6-10X density advantage at the same technology generation.

- **SRAM** advantages
  - [ ] **SRAM** has deterministic latency: its cells do not need to be refreshed.
  - **SRAM** is much faster: transistors drive bitlines on reads.
  - **SRAM** easy to design in logic fabrication process (and premium logic processes have SRAM add-ons)

Big win for **DRAM**
RAM Compilers

On average, 30% of a modern logic chip is SRAM, which is generated by RAM compilers.

Compile-time parameters set number of bits, aspect ratio, ports, etc.
Flip Flops Revisited
Recall: Static RAM cell (6 Transistors)

"Cross-coupled inverters"
Recall: Positive edge-triggered flip-flop

A flip-flop “samples” right before the edge, and then “holds” value.

Sampling circuit

Holds value

16 Transistors: Makes an SRAM look compact!

What do we get for the 10 extra transistors? Clocked logic semantics.
Sensing: When clock is low

A flip-flop “samples” right before the edge, and then “holds” value.

Sampling circuit

Holds value

Will capture new value on posedge.

Outputs last value captured.

\[ \text{clk} = 0 \]
\[ \text{clk}' = 1 \]
Capture: When clock goes high

A flip-flop "samples" right before the edge, and then "holds" value.

Sampling circuit

Holds value

\[ \text{clk} = 1 \quad \text{clk}' = 0 \]

Remembers value just captured.

Outputs value just captured.
Flip Flop delays: clk-to-Q? setup? hold?

CLK == 0
Sense D, but Q outputs old value.

CLK 0->1
Capture D, pass value to Q
From flip-flops to latches ...

Latch-based design: Break up the flip-flop circuit into two latch state elements. Then, add combinational logic between the latches.

Latches are good for making small memories. Saves half the area over using D flip-flops.
Break
The Memory Hierarchy
60% of the area of this CPU is devoted to SRAM cache.

But the role of cache in computer design has varied widely over time.
1977: DRAM faster than microprocessors

Apple ][ (1977)

CPU: 1000 ns
DRAM: 400 ns
Since then: Technology scaling ...

Circuit in **250 nm** technology (introduced in 2000)

Same circuit in **180 nm** technology (introduced in 2003)

- Each dimension 30% smaller.
- Area is 50% smaller

Logic circuits use smaller $C'$s, lower $V_{dd}$, and higher $k_n$ and $k_p$ to speed up clock rates.
DRAM scaled for more bits, not more MHz

Assume \( C_{\text{cell}} = 1 \, \text{fF} \)

Bit line may have 2000 nFET drains, assume bit line \( C \) of 100 fF, or 100*\( C_{\text{cell}} \).

\( C_{\text{cell}} \) holds \( Q = C_{\text{cell}} \times (V_{\text{dd}} - V_{\text{th}}) \)

When we dump this charge onto the bit line, what voltage do we see?

\[
dV = \frac{C_{\text{cell}} \times (V_{\text{dd}} - V_{\text{th}})}{100 \times C_{\text{cell}}} \\
   \approx \frac{V_{\text{dd}} - V_{\text{th}}}{100} \approx \text{tens of millivolts!}
\]

In practice, scale array to get a 60mV signal.
1980-2003, CPU speed outpaced DRAM ...

Q. How do architects address this gap?
A. Put smaller, faster "cache" memories between CPU and DRAM. Create a "memory hierarchy".

- CPU: 60% per yr, 2X in 1.5 yrs
- DRAM: 9% per yr, 2X in 10 yrs

Gap grew 50% per year

The power wall
Caches: Variable-latency memory ports

Data in upper memory returned with lower latency.

Data in lower level returned with higher latency.

From CPU

To CPU

Small, fast
Blk X

Large, slow
Blk Y

To Processor

From Processor

Upper Level Memory

Lower Level Memory

CLK

REQ

Addr

Wait

Data

Data Out

Address

Read Address
Avoid blocking by using a queue (a First-In, First-Out buffer, or FIFO) to communicate between two sub-systems.
Variable-latency port that doesn’t stall on a miss

CPU makes a request by placing the following items in Queue 1:

**CMD:** Read, write, etc ...

**MTYPE:** 8-bit, 16-bit, 32-bit, or 64-bit.

**TAG:** 9-bit number identifying the request.

**MADDR:** Memory address of first byte.

**STORE-DATA:** For stores, the data to store.
This cache is used in an ASPIRE CPU (Rocket)

When request is ready, cache places the following items in Queue 2:

**TAG:** Identity of the completed command.

**LOAD-DATA:** For loads, the requested data.

CPU saves info about requests, indexed by **TAG**.

Why use **TAG** approach? Multiple misses can proceed in parallel. Loads can return out of order.
Cache replaces data, instruction memory

Replace with Instruction Cache and Data Cache of DRAM main memory

IF (Fetch) → ID (Decode) → EX (ALU) → MEM → WB
Recall: Intel ARM XScale CPU (PocketPC)

![Diagram of Intel ARM XScale CPU](image)

- **32 KB Instruction Cache**
- **32 KB Data Cache**
- **180 nm process (introduced 2003)**
ARM CPU

32 KB instruction cache uses 3 million transistors. Typical miss rate: 1.5%

DRAM interface uses 61 pins that toggle at 100 MHz.
### 2005 Memory Hierarchy: Apple iMac G5

#### Managed by compiler
- L1 Inst
- L1 Data

#### Managed by hardware
- L2
- DRAM
- Disk

#### Managed by OS, hardware, application

<table>
<thead>
<tr>
<th></th>
<th>Reg</th>
<th>L1 Inst</th>
<th>L1 Data</th>
<th>L2</th>
<th>DRAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>1K</td>
<td>64K</td>
<td>32K</td>
<td>512K</td>
<td>256M</td>
<td>80G</td>
</tr>
<tr>
<td>Latency (cycles)</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>160</td>
<td>10M</td>
</tr>
</tbody>
</table>

**Goal:** Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access.
## Latency: A closer look

**Read latency:** Time to return first byte of a random access

<table>
<thead>
<tr>
<th>Size</th>
<th>Reg</th>
<th>L1 Inst</th>
<th>L1 Data</th>
<th>L2</th>
<th>DRAM</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>64K</td>
<td>32K</td>
<td>512K</td>
<td>256M</td>
<td>80G</td>
<td></td>
</tr>
<tr>
<td>Latency (cycles)</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>160</td>
<td>1E+07</td>
</tr>
<tr>
<td>Latency (sec)</td>
<td>0.6n</td>
<td>1.9n</td>
<td>1.9n</td>
<td>6.9n</td>
<td>100n</td>
<td>12.5m</td>
</tr>
<tr>
<td>Hz</td>
<td>1.6G</td>
<td>533M</td>
<td>533M</td>
<td>145M</td>
<td>10M</td>
<td>80</td>
</tr>
</tbody>
</table>

**Architect’s latency toolkit:**

1. **Parallelism.** Request data from N 1-bit-wide memories at the same time. Overlaps latency cost for all N bits. Provides N times the bandwidth. Requests to N memory banks (interleaving) have potential of N times the bandwidth.

2. **Pipeline memory.** If memory has N cycles of latency, issue a request each cycle, receive it N cycles later.
Recall: Adding pipeline stages to memory

Before we pipelined, slow! Only read behavior shown.

A7-A0: 8-bit read address

Can we add two pipeline stages?

Each register holds 32 bytes (256 bits)

OE --> Tri-state Q outputs!

Data output is 32 bits i.e. 4 bytes
Recall: Reading an entire row for later use

What if we want all of the 16384 bits?
In row access time (55 ns) we can do
22 transfers at 400 MT/s.
16-bit chip bus \( \rightarrow 22 \times 16 = 352 \) bits \(< 16384\)
Now the row access time looks fast!

Thus, push to faster DRAM interfaces

13-bit row address input

16384 columns

8192 rows

134 217 728 usable bits
(tester found good bits in bigger array)

16384 bits delivered by sense amps

Select requested bits, send off the chip

[Diagram of memory access and addressing]
Interleaving: Design the right interface to the 4 memory banks on the chip, so several row requests run in parallel.

Can also do other commands on banks concurrently.
Recall: Leveraging banks and row reads

(A) Without access scheduling (56 DRAM Cycles)

(B) With access scheduling (19 DRAM Cycles)

DRAM Operations:

P: bank precharge (3 cycle occupancy)
A: row activation (3 cycle occupancy)
C: column access (1 cycle occupancy)

From: Memory Access Scheduling
Set scheduling algorithms in gates ...

Figure 4. Memory access scheduler architecture.

Memory References

Bank 0 Pending References

<table>
<thead>
<tr>
<th>V</th>
<th>L/S</th>
<th>Row</th>
<th>Col</th>
<th>Data</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bank N Pending References

<table>
<thead>
<tr>
<th>V</th>
<th>L/S</th>
<th>Row</th>
<th>Col</th>
<th>Data</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory Access Scheduler Logic

Precharge\(_0\)

Row Arbiter\(_0\)

Column Arbiter

Row Arbiter\(_N\)

Precharge\(_N\)

Address Arbiter

DRAM Operations

From: Memory Access Scheduling
On Tuesday

Caches, part two ...

<table>
<thead>
<tr>
<th>T 2/25</th>
<th>Cache II</th>
<th>Chapter 2.3,6.</th>
</tr>
</thead>
</table>

Have a good weekend!