# Computer Architecture

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

September 5, 2025

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a "grey'ed out" header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

- Problem: an $n$-bit register based on latches (resp. flip-flops) is limited in that
  1. each latch (resp. flip-flop) in the register needs a relatively large number of transistors, which limits the viable capacity (i.e., $n$), and
  2. the register is not addressable, i.e.,
     - an **address** (or **index**) allows dynamic rather than static reference to some stored datum, so
     - by analogy, in a C program

Listing

```
1  int A0, A1, A2, A3;
2
3  A0 = 0;
4  A1 = 0;
5  A2 = 0;
6  A3 = 0;
```

Listing

```
1  int A[ 4 ];
2
3  A[ 0 ] = 0;
4  A[ 1 ] = 0;
5  A[ 2 ] = 0;
6  A[ 3 ] = 0;
```

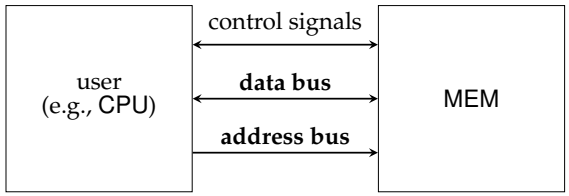
we *have* the left-hand side, but we *want* the right-hand side.

Notes:





- Solution: a **memory** component, i.e.,




such that

- MEM has a capacity of $n = 2^{n'}$ addressable words, and
- each such word is $w$ bits (where $n \gg w$).

Notes:

- Agenda:
  1. memory *cells*,
  2. memory *devices*,

  noting there are various ways to classify memories, e.g.,
  1. volatility:
     - **volatile**, meaning the content is lost when the component is powered-off, or
     - **non-volatile**, meaning the content is retained even after the component is powered-off.
  2. interface type:
     - **synchronous**, where a clock or pre-determined timing information synchronises steps, or
     - **asynchronous**, where a protocol synchronises steps.
  3. access type:
     - random versus constrained (e.g., sequential) access to content,
     - **Random Access Memory (RAM)** which we can read from *and* write to, and
     - **Read Only Memory (ROM)** which, as suggested by the name, supports reads only.
  4. ...

  but we'll focus exclusively on a volatile, synchronous RAM.

Notes:



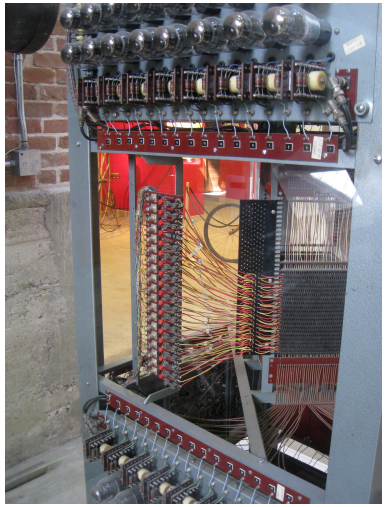
## An Aside: some history

- The EDSAC used **delay line** memory, where the rough idea is:
  - Each "line" is a tube of mercury (or something else in which sound waves propagate fairly slowly).
  - Put a speaker at one end to store sound waves into the line, and a microphone at the other to read them out.
  - Values are stored in the sense the corresponding waves take time to propagate; when they get to one end they are either replaced or fed back into the other.
- This is **sequential access** (cf. **random access**): you need to *wait* for the data you want to appear!

https://www.cl.cam.ac.uk/Relics/jpegs/delay_lines.jpg

Notes:

## An Aside: some history



- The Whirlwind used **magnetic-core** memory, where the rough idea is:
  - The memory is a matrix of small magnetic rings, or "cores", which can be magnetically polarised to store values.
  - Wires are threaded through the cores to control them, i.e., to store or read values.
  - The magnetic polarisation is retained, so core memory is non-volatile!
- You might still hear main memory termed **core memory** (cf. **core dump**) which is a throw-back to this technology.

https://en.wikipedia.org/wiki/File:Project_Whirlwind_-_core_memory,_circa_1951_-_detail_1.JPG



Notes:

## Part 1: memory cells (1)

### Comparison

**Static RAM (SRAM)** is

- manufacturable in lower densities (i.e., smaller capacity),
- more expensive to manufacture,
- fast(er) access time (resp. lower access latency),
- easy(er) to interface with,
- ideal for latency-optimised contexts, e.g., as cache memory.

### Comparison

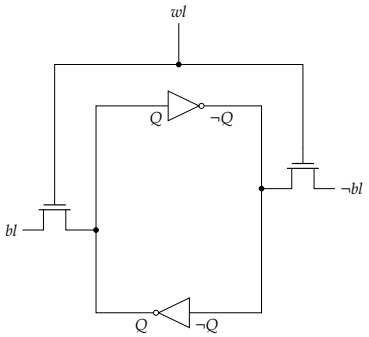
**Dynamic RAM (DRAM)** is

- manufacturable in higher densities (i.e., larger capacity),
- less expensive to manufacture,
- slow(er) access time (resp. higher access latency),
- hard(er) to interface with,
- ideal for capacity-optimised contexts, e.g., as main memory.

Notes:

# Part 1: memory cells (2)
## An SRAM cell

### Circuit




- Idea:
  - internally, the cell is essentially two NOT gates,
  - $bl$ and $\neg bl$ are the **bit lines** (via which the state is accessed),
  - $wl$ is the **word line** (which controls access to the state),
  - a "6T SRAM cell" requires 6 transistors (cf. $\sim 20$ or so for a D-type latch).

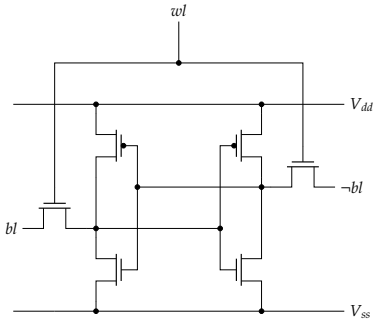
Notes:

- The initial NOT-based circuit might look odd, but clearly has two stable states: either $Q = 1$ and $\neg Q = 0$, or $Q = 0$ and $\neg Q = 1$. The transistors re-enforce each other; the state is maintained as long as the cell is powered-on.
- To read the cell we pre-charge $bl = 1$ and $\neg bl = 1$ then set $wl = 1$, after which $\neg bl$ (resp. $bl$) is discharged if state is 1 (resp. 0). To write $x$ into the cell we pre-charge $bl = x$ and $\neg bl = \neg x$ then set $wl = 1$, after which the state matches $x$.
- The pre-charging steps are managed by extra **bit line conditioning**, the detail of which we ignore.



Part 1: memory cells (2)
An SRAM cell

Circuit




- Idea:
  - internally, the cell is essentially two NOT gates,
  - $bl$ and $\neg bl$ are the **bit lines** (via which the state is accessed),
  - $wl$ is the **word line** (which controls access to the state),
  - a "6T SRAM cell" requires 6 transistors (cf. $\sim 20$ or so for a D-type latch).

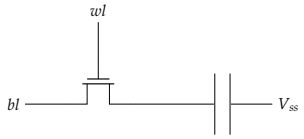
Notes:

- The initial NOT-based circuit might look odd, but clearly has two stable states: either $Q = 1$ and $\neg Q = 0$, or $Q = 0$ and $\neg Q = 1$. The transistors re-enforce each other; the state is maintained as long as the cell is powered-on.
- To read the cell we pre-charge $bl = 1$ and $\neg bl = 1$ then set $wl = 1$, after which $\neg bl$ (resp. $bl$) is discharged if state is 1 (resp. 0). To write $x$ into the cell we pre-charge $bl = x$ and $\neg bl = \neg x$ then set $wl = 1$, after which the state matches $x$.
- The pre-charging steps are managed by extra **bit line conditioning**, the detail of which we ignore.

© Daniel Page ⟨cdsapp@cs.bris.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

# Part 1: memory cells (3)
## A DRAM cell

### Circuit




- Idea:
  - internally, the cell is essentially one one transistor and one capacitor,
  - *bl* is the **bit line** (via which the state is accessed),
  - *wl* is the **word line** (which controls access to the state),
  - the capacitor
    1. discharges and charges (relatively) slowly,
    2. discharges when the cell is read, *and* also over time even if it's not read; this implies a need to **refresh** it.

Notes:

- To read the cell we set $wl = 1$, after which current flows (resp. does not flow) on $bl$ if the capacitor is charged (resp. not charged) meaning the state is 1 (resp. 0). To write $x$ into the cell we set $bl = x$ then $wl = 1$, after which the capacitor charges (resp. discharges) and the state matches $x$.
- The capacitor holds a tiny charge: this must be amplified to use as a driver in whatever circuit uses the cell.



# Part 2: memory cells ⇝ memory devices (1)

- Concept: a **memory device** is constructed from (roughly) three components
  1. a **memory array** (or matrix) of replicated cells with
     - $r$ rows, and
     - $c$ columns

     meaning a $(r \cdot c)$-cell capacity,
  2. a **row decoder** which given an address (de)activates associated cells in that row, and
  3. a **column decoder** which given an address (de)selects associated cells in that column

  plus additional logic to allow use (depending on cell type), e.g.,
  1. **bit line conditioning** to, e.g., ensure the bit lines are strong enough to be effective, and
  2. **sense amplifiers** to, e.g., ensure output from the array is usable.

Notes:

- (Typical, or exemplar) design: an **SRAM device**.
  1. interface:
     - auxiliary pin(s) for power and so on,
     - $D$, a single 1-bit **data pin**,
     - $A$, a collection of $n'$ **address pins** where $A_i$ is the $i$-th such pin,
     - a **Chip Select (CS)** pin, which enables the device,
     - a **Output Enable (OE)** pin, which signals the device is being read from,
     - a **Write Enable (WE)** pin, which signals the device is being written to.

Notes:
- In some cases, $D$ might be split into two separate $D_{in}$ and $D_{out}$ pins dedicated to input and output respectively.



Part 2: memory cells ⇝ memory devices (2)
An SRAM device: design

- (Typical, or exemplar) design: an **SRAM device**.

  2. usage:

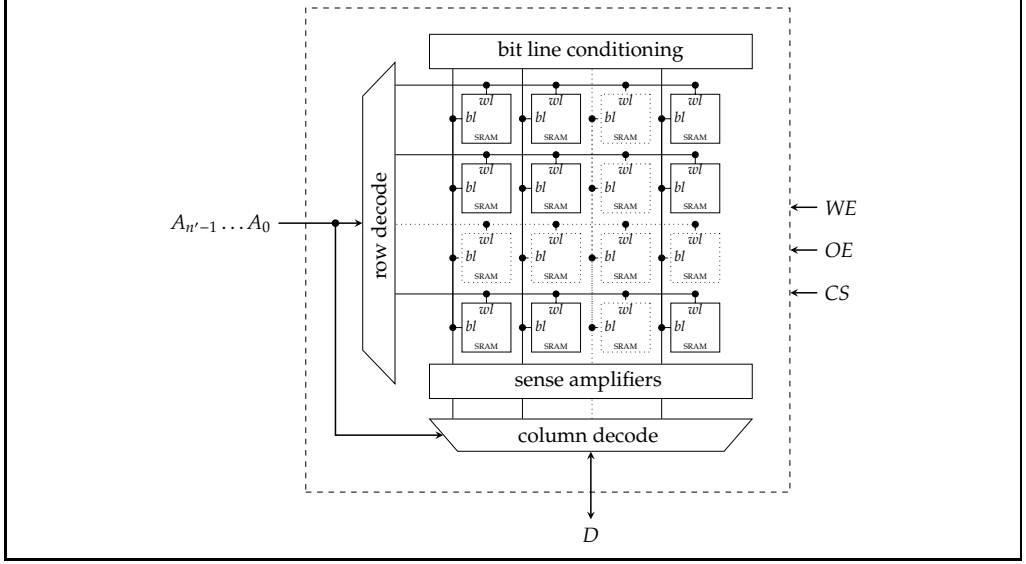
| Algorithm (SRAM-Read) |
| --- |
| Having performed the following steps<br>▶ drive the address onto $A$,<br>▶ set $WE$ = **false**, $OE$ = **true** and $CS$ = **true**,<br>1-bit of data is read and made available on $D$, then we set $CS$ = **false**. |

| Algorithm (SRAM-Write) |
| --- |
| Having performed the following steps<br>▶ drive the data onto $D$,<br>▶ drive the address onto $A$,<br>▶ set $WE$ = **true**, $OE$ = **false** and $CS$ = **true**,<br>1-bit of data is written, then we set $CS$ = **false**. |

Notes:
- In some cases, $D$ might be split into two separate $D_{in}$ and $D_{out}$ pins dedicated to input and output respectively.

© Daniel Page ⟨cdsapple@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

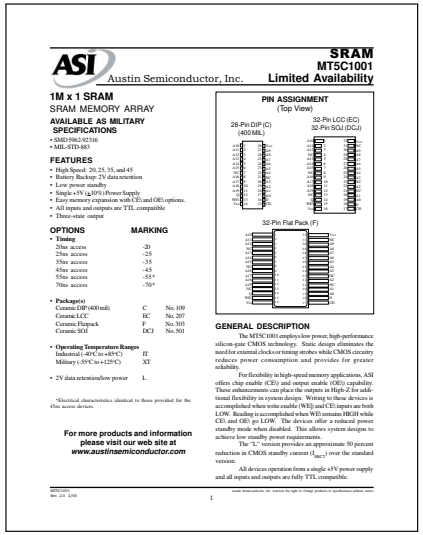
## Example (an $n$-cell SRAM device, for $n = 2^{n'}$)




Notes:







ASI Austin Semiconductor, Inc. — SRAM MT5C1001 Limited Availability

**1M x 1 SRAM**
SRAM MEMORY ARRAY

**AVAILABLE AS MILITARY SPECIFICATIONS**
- SMD 5962-92316
- MIL-STD-883

**FEATURES**
- High Speed: 20, 25, 35, and 45
- Battery Backup: 2V data retention
- Low power standby
- Single +5V (±10%) Power Supply
- Easy memory expansion with CE\ and OE\ options.
- All inputs and outputs are TTL compatible
- Three-state output

**OPTIONS** / **MARKING**
- **Timing**: 20ns access -20; 25ns access -25; 35ns access -35; 45ns access -45; 55ns access -55*; 70ns access -70*
- **Package(s)**: Ceramic DIP (600 mil) C No. 109; Ceramic LCC EC No. 207; Ceramic Flatpack F No. 103; Ceramic SOJ DCJ No. 501
- **Operating Temperature Ranges**: Industrial (-40°C to +85°C) IT; Military (-55°C to +125°C) XT
- 2V data retention/low power L

*Electrical characteristics identical to those provided for the 45ns access devices.

**For more products and information please visit our web site at www.austinsemiconductor.com**

PIN ASSIGNMENT (Top View)
28-Pin DIP (C) (400 MIL)
32-Pin LCC (EC)
32-Pin SOJ (DCJ)
32-Pin Flat Pack (F)

**GENERAL DESCRIPTION**

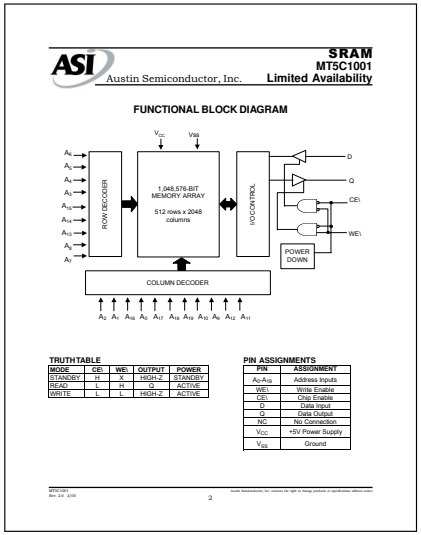
The MT5C1001 employs low-power, high-performance silicon-gate CMOS technology. Static design eliminates the need for external clocks or timing strobes while CMOS circuitry reduces power consumption and provides for greater reliability.

For flexibility in high-speed memory applications, ASI offers chip enable (CE\) and output enable (OE\) capability. These enhancements can place the outputs in High-Z for additional flexibility in system design. Writing to these devices is accomplished when write enable (WE\) and CE\ inputs are both LOW. Reading is accomplished when WE\ remains HIGH while CE\ and OE\ go LOW. The devices offer a reduced power standby mode when disabled. This allows system designs to achieve low standby power requirements.

The "L" version provides an approximate 50 percent reduction in CMOS standby current ($I_{SB2}$) over the standard version.

All devices operation from a single +5V power supply and all inputs and outputs are fully TTL compatible.





ASI Austin Semiconductor, Inc. — SRAM MT5C1001 Limited Availability

**FUNCTIONAL BLOCK DIAGRAM**




**TRUTH TABLE**

| MODE | CE\ | WE\ | OUTPUT | POWER |
|---|---|---|---|---|
| STANDBY | H | X | HIGH-Z | STANDBY |
| READ | L | H | Q | ACTIVE |
| WRITE | L | L | HIGH-Z | ACTIVE |

**PIN ASSIGNMENTS**

| PIN | ASSIGNMENT |
|---|---|
| $A_0$-$A_{10}$ | Address Inputs |
| WE\ | Write Enable |
| CE\ | Chip Enable |
| D | Data Input |
| Q | Data Output |
| NC | No Connection |
| $V_{CC}$ | +5V Power Supply |
| $V_{SS}$ | Ground |



Notes:

- (Typical, or exemplar) design: a **DRAM device**.
  1. interface:
     - auxiliary pin(s) for power and so on,
     - $D$, a single 1-bit **data pin**,
     - $A$, a collection of $n'/2$ **address pins** where $A_i$ is the $i$-th such pin,
     - a **Chip Select (CS)** pin, which enables the device,
     - a **Output Enable (OE)** pin, which signals the device is being read from,
     - a **Write Enable (WE)** pin, which signals the device is being written to,
     - a **Row Address Strobe (RAS)**, which controls the row buffer, and
     - a **Column Address Strobe (CAS)**, which controls the column buffer.

Notes:
- In some cases, $D$ might be split into two separate $D_{in}$ and $D_{out}$ pins dedicated to input and output respectively.
- An address is basically just an unsigned integer: if you have an $n'$-bit address, you can therefore use it to identify any one of $n = 2^{n'}$ addressable elements. An SRAM device will typically have $n'$ address pins, so one can provide the address in a single step. An DRAM device will typically have less than $n'$ address pins, so one must provide the address in multiple steps: here we see the use of $\frac{n'}{2}$ address pins, for example, which are used to provide an $n'$-bit address in 2 steps. The rationale for the latter is to cope with density. A DRAM device will typically has many addressable elements, but there is a physical limit on how many address pins are reasonable: by multiplexing, or sharing them, this issue is mitigated to some extent.
  Note that saying "typical" here is an important caveat, because it is not easy to give one definitive answer: there *are* other approaches and exceptions, for example which might be valid, even if less useful.



Part 2: memory cells ⇝ memory devices (5)
A DRAM device: interface

- (Typical, or exemplar) design: a **DRAM device**.

  2. usage:

**Algorithm (DRAM-Read)**

Having performed the following steps
- drive the row address onto $A$,
- set $RAS$ = **true** to latch row address,
- drive the column address onto $A$,
- set $CAS$ = **true** to latch column address,
- set $WE$ = **false**, $OE$ = **true** and $CS$ = **true**,

1-bit of data is read and made available on $D$, then we set $CS = RAS = CAS =$ **false**.

**Algorithm (DRAM-Write)**

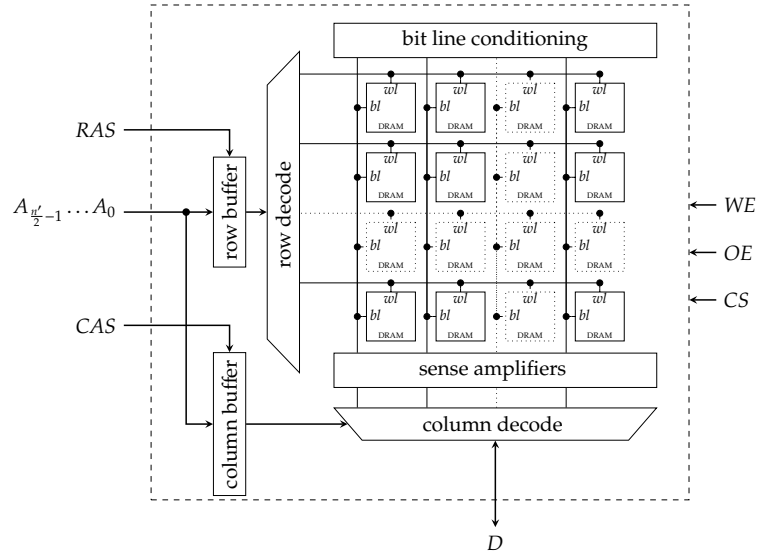
Having performed the following steps
- drive the data onto $D$,
- drive the row address onto $A$,
- set $RAS$ = **true** to latch row address,
- drive the column address onto $A$,
- set $CAS$ = **true** to latch column address,
- set $WE$ = **false**, $OE$ = **true** and $CS$ = **true**,

1-bit of data is written, then we set $CS = RAS = CAS =$ **false**.

Notes:
- In some cases, $D$ might be split into two separate $D_{in}$ and $D_{out}$ pins dedicated to input and output respectively.
- An address is basically just an unsigned integer: if you have an $n'$-bit address, you can therefore use it to identify any one of $n = 2^{n'}$ addressable elements. An SRAM device will typically have $n'$ address pins, so one can provide the address in a single step. An DRAM device will typically have less than $n'$ address pins, so one must provide the address in multiple steps: here we see the use of $\frac{n'}{2}$ address pins, for example, which are used to provide an $n'$-bit address in 2 steps. The rationale for the latter is to cope with density. A DRAM device will typically has many addressable elements, but there is a physical limit on how many address pins are reasonable: by multiplexing, or sharing them, this issue is mitigated to some extent.
  Note that saying "typical" here is an important caveat, because it is not easy to give one definitive answer: there *are* other approaches and exceptions, for example which might be valid, even if less useful.

© Daniel Page ⟨cs-ca@bristol.ac.uk⟩
Computer Architecture
University of BRISTOL
git # b282dbb9 @ 2025-09-03

## Example (a $n$-cell DRAM memory device, for $n = 2^{n'}$)

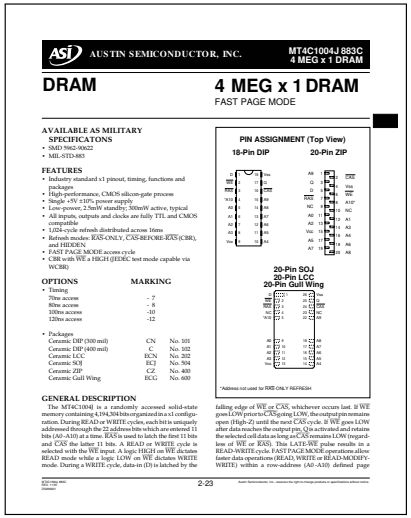



Notes:

- We also need some logic, which is not shown here, to refresh the cells At a high level, the idea is that to cope with decay of content, we periodically read







ASI AUSTIN SEMICONDUCTOR, INC. MT4C1004J 883C 4 MEG x 1 DRAM

**DRAM** **4 MEG x 1 DRAM**
FAST PAGE MODE

**AVAILABLE AS MILITARY SPECIFICATONS**
- SMD 5962-90622
- MIL-STD-883

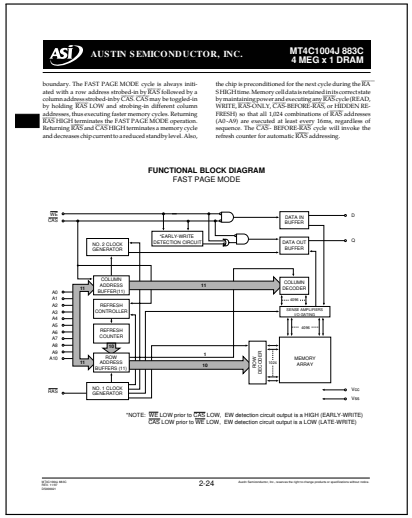
**FEATURES**
- Industry standard x1 pinout, timing, functions and packages
- High-performance, CMOS silicon-gate process
- Single +5V ±10% power supply
- Low-power, 2.5mW standby; 300mW active, typical
- All inputs, outputs and clocks are fully TTL and CMOS compatible
- 1,024-cycle refresh distributed across 16ms
- Refresh modes: RAS-ONLY, CAS-BEFORE-RAS (CBR), and HIDDEN
- FAST PAGE MODE access cycle
- CBR with WE a HIGH (JEDEC test mode capable via WCBR)

**OPTIONS** **MARKING**
- Timing
  - 70ns access - 7
  - 80ns access - 8
  - 100ns access -10
  - 120ns access -12
- Packages
  - Ceramic DIP (300 mil) CN No. 101
  - Ceramic DIP (400 mil) C No. 102
  - Ceramic LCC ECN No. 202
  - Ceramic SOJ ECJ No. 504
  - Ceramic ZIP CZ No. 400
  - Ceramic Gull Wing ECG No. 600

**PIN ASSIGNMENT (Top View)**
18-Pin DIP 20-Pin ZIP
20-Pin SOJ
20-Pin LCC
20-Pin Gull Wing
*Address not used for RAS-ONLY REFRESH

**GENERAL DESCRIPTION**
The MT4C1004J is a randomly accessed solid-state memory containing 4,194,304 bits organized in a x1 configuration. During READ or WRITE cycles, each bit is uniquely addressed through the 22 address bits which are entered 11 bits (A0–A10) at a time. RAS is used to latch the first 11 bits and CAS the latter 11 bits. A READ or WRITE cycle is selected with the WE input. A logic HIGH on WE dictates READ mode while a logic LOW on WE dictates WRITE mode. During a WRITE cycle, data-in (D) is latched by the falling edge of WE or CAS, whichever occurs last. If WE goes LOW prior to CAS going LOW, the output pin remains open (High-Z) until the next CAS cycle. If WE goes LOW after data reaches the output pin, Q is activated and retains the selected cell data as long as CAS remains LOW (regardless of WE or RAS). This LATE-WE pulse results in a READ-WRITE cycle. FAST PAGE MODE operations allow faster data operations (READ, WRITE or READ-MODIFY-WRITE) within a row-address (A0–A10) defined page





ASI AUSTIN SEMICONDUCTOR, INC. MT4C1004J 883C 4 MEG x 1 DRAM

boundary. The FAST PAGE MODE cycle is always initiated with a row address strobed-in by RAS followed by a column address strobed-in by CAS. CAS may be toggled-in by holding RAS LOW and strobing-in different column addresses, thus executing faster memory cycles. Returning RAS HIGH terminates the FAST PAGE MODE operation. Returning RAS and CAS HIGH terminates a memory cycle and decreases chip current to a reduced standby level. Also, the chip is preconditioned for the next cycle during the RAS HIGH time. Memory cell data is retained in its correct state by maintaining power and executing any RAS cycle (READ, WRITE, RAS-ONLY, CAS-BEFORE-RAS, or HIDDEN REFRESH) so that all 1,024 combinations of RAS addresses (A0–A9) are executed at least every 16ms, regardless of sequence. The CAS- BEFORE-RAS cycle will invoke the refresh counter for automatic RAS addressing.

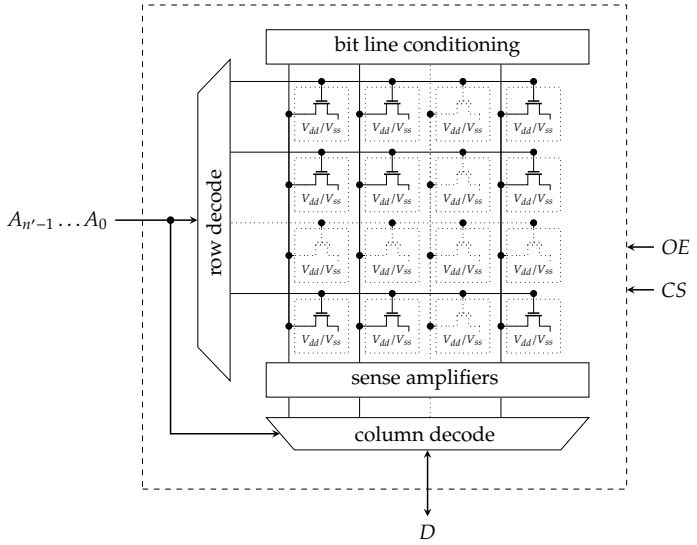
**FUNCTIONAL BLOCK DIAGRAM**
FAST PAGE MODE

*NOTE: WE LOW prior to CAS LOW. EW detection circuit output is a HIGH (EARLY-WRITE)
CAS LOW prior to WE LOW. EW detection circuit output is a LOW (LATE-WRITE)



Notes:

## Example (an $n$-cell ROM device, for $n = 2^{n'}$)




Notes:



## Part 2: memory cells ⇝ memory devices (9)

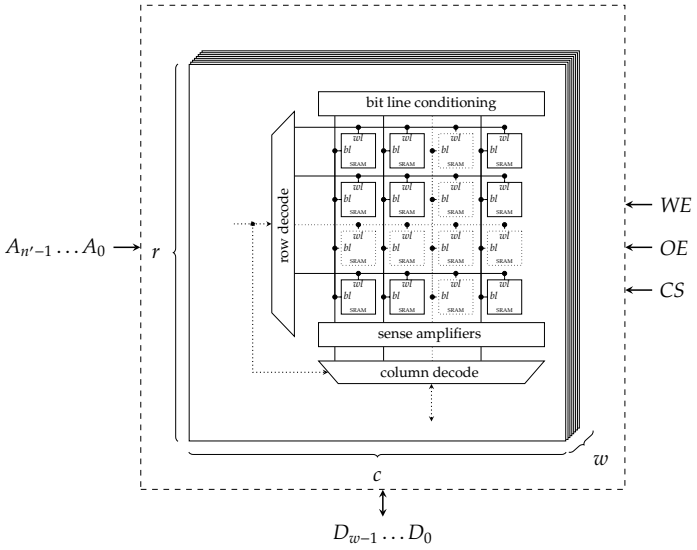
- Concept:
  - *externally*, the configuration of a device is described as something like

    $$\delta \times \omega \times \beta$$

    (plus maybe some timing information) where
    - $\delta$ relates to capacity, usually measured in (large multiples of) bits,
    - $\omega$ describes the width of words, measured in bits, and
    - $\beta$ is the number of internal **logical banks**.
  - *internally*, this implies some organisational choices: for example,
    1. for $\omega > 1$, we replicate the memory device internally to give $\omega$ arrays (each copy relates to one bit of a $\omega$-bit word),
    2. for the arrays, $r$ and $c$ can be selected to match physical requirements (e.g., to get "square" or "thin" arrays), and
    3. for $\beta > 1$, each array is split into logical **banks**.

Notes:

## Example (from a 1-bit to $w$-bit SRAM device via replication)

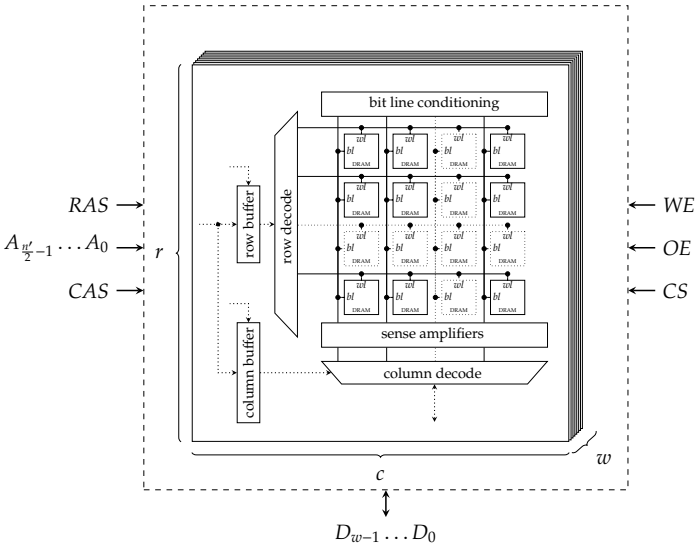



Notes:





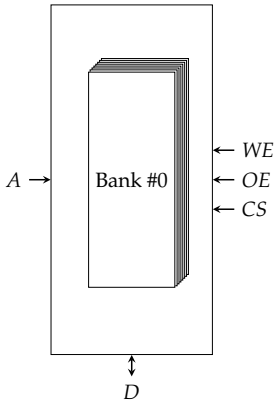
## Example (from a 1-bit to $w$-bit DRAM device via replication)




Notes:

## Example (a 512Mbit, 8-bit Device)

- A 64Mbit × 8 × 1 internal configuration:
  - The total capacity is 64Mbit · 8 · 1 = 512Mbit.
  - There is $\beta = 1$ logical bank: the bank consists of $\omega = 8$ arrays, each of which has $\delta = r \cdot c = 64 \cdot 1024 \cdot 1024$ cells.
  - Address $x$ refers to cell $x$ of each array; for example, $x = 42_{(10)}$ refers to cell 42 of each array, and therefore to an 8-bit word overall.




Notes:



Part 2: memory cells ⇝ memory devices (12)

## Example (a 512Mbit, 8-bit Device)

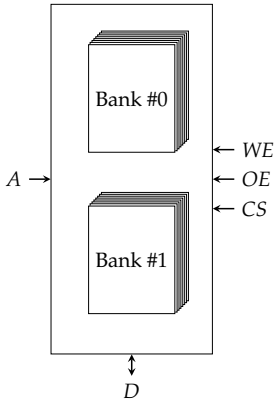
- A 32Mbit × 8 × 2 internal configuration:
  - The total capacity is 32Mbit · 8 · 2 = 512Mbit.
  - There are $\beta = 2$ logical banks: each bank consists of $\omega = 8$ arrays, each of which has $\delta = r \cdot c = 32 \cdot 1024 \cdot 1024$ cells.
  - Address $x$ refers to

$$x \mapsto \begin{cases} \text{bank} & : \quad x \bmod \beta \\ \text{cell} & : \quad x \operatorname{div} \beta \end{cases}$$

of each array; for example,

  - $x = 42_{(10)}$ refers to cell 21 of each array in bank 0, while
  - $x = 43_{(10)}$ refers to cell 21 of each array in bank 1,

each case therefore referring to 8-bit word overall.




Notes:

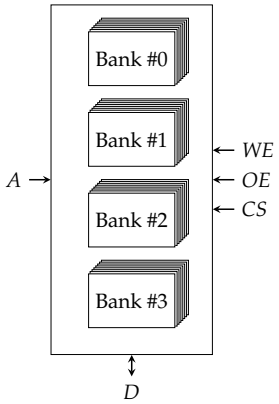
## Example (a 512Mbit, 8-bit Device)

- A 16Mbit × 8 × 4 internal configuration:
  - The total capacity is 16Mbit · 8 · 4 = 512Mbit.
  - There are $\beta = 4$ logical banks: each bank consists of $\omega = 8$ arrays, each of which has $\delta = r \cdot c = 16 \cdot 1024 \cdot 1024$ cells.
  - Address $x$ refers to

$$x \mapsto \begin{cases} \text{bank} & : \quad x \bmod \beta \\ \text{cell} & : \quad x \operatorname{div} \beta \end{cases}$$

  of each array; for example,
  - $x = 42_{(10)}$ refers to cell 10 of each array in bank 2, while
  - $x = 43_{(10)}$ refers to cell 21 of each array in bank 3,

  each case therefore referring to 8-bit word overall.




Notes:





- Concept: a **memory module** is essentially the combination of
  1. one or more memory devices, plus
  2. an interface which controls access.

  with two (physical) package types dominating:
  1. **Single Inline Memory Module (SIMM)**, which is (roughly) 1-sided, has less pins and a narrower word size, and
  2. **Dual Inline Memory Module (DIMM)**, which is (roughly) 2-sided, has more pins and a wider word size

  and are capable of housing different device types (e.g., EDO *or* SDRAM devices).

Notes:

- Concept:
  - *externally*, the configuration of a module is described as something like

$$\Delta \times \Omega$$

    (plus maybe some timing information) where
    - $\Delta$ relates to capacity, usually measured in (large multiplies of) bytes, and
    - $\Omega$ describes the width of words, measured in bits.
  - *internally*, this implies some organisational choices: for example,
    1. usually $\Omega > \omega$ so the module is "filled" using multiple devices to form one **physical bank**, and
    2. depending on the module type, the devices are organised into one or more **ranks**.

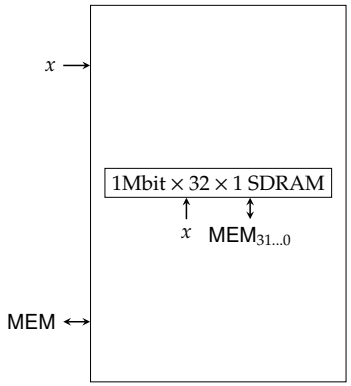
Notes:





## Example (a 4MB, 32-bit SIMM)

- A 4MB × 32, 1-device configuration:
  - The device has a capacity of $1\text{Mbit} \cdot 32 \cdot 1 = 32\text{Mbit} = 4\text{MB}$, arranged into 32-bit words.
  - The bank, which has a 32-bit data-path, is filled by the single device: address $x$ refers to the word MEM.



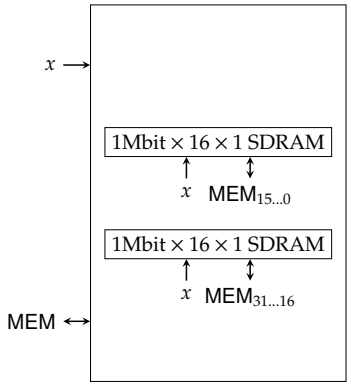

Notes:

## Example (a 4MB, 32-bit SIMM)

- A 4MB × 32, 2-device configuration:
  - Each device has a capacity of $1\text{Mbit} \cdot 16 \cdot 1 = 16\text{Mbit} = 2\text{MB}$, arranged into 16-bit words.
  - Since there are 2 devices, the total capacity is $2 \cdot 2\text{MB} = 4\text{MB}$.
  - The bank, which has a 32-bit data-path, is filled by merging the devices: address $x$ refers to

$$x \mapsto \begin{cases} \text{bits} & 15 \ldots 0 & \text{of MEM from device 0} \\ \text{bits} & 31 \ldots 16 & \text{of MEM from device 1} \end{cases}$$

  merging each 16-bit part into the word MEM.






Notes:

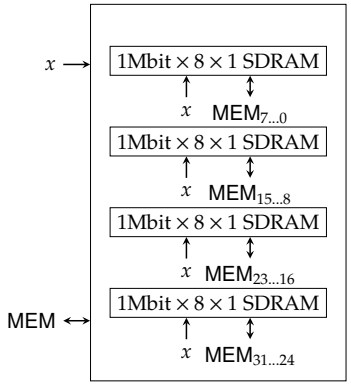
Part 3: memory devices ⇝ memory modules (3)

## Example (a 4MB, 32-bit SIMM)

- A 4MB × 32, 4-device configuration:
  - Each device has a capacity of $1\text{Mbit} \cdot 8 \cdot 1 = 8\text{Mbit} = 1\text{MB}$, arranged into 8-bit words.
  - Since there are 4 devices, the total capacity is $4 \cdot 1\text{MB} = 4\text{MB}$.
  - The bank, which has a 32-bit data-path, is filled by merging the devices: address $x$ refers to

$$x \mapsto \begin{cases} \text{bits} & 7 \ldots 0 & \text{of MEM from device 0} \\ \text{bits} & 15 \ldots 8 & \text{of MEM from device 1} \\ \text{bits} & 23 \ldots 16 & \text{of MEM from device 2} \\ \text{bits} & 31 \ldots 24 & \text{of MEM from device 3} \end{cases}$$

  merging each 8-bit part into the word MEM.






Notes:

## Conclusions

- Take away points:
  1. The initial goal was an $n$-element memory of $w$-bit words; the final solution is motivated by divide-and-conquer, i.e.,
     - 1.1 one or more channels, each backed by
     - 1.2 one or more physical banks, each composed from
     - 1.3 one or more devices, each composed from
     - 1.4 one or more logical banks, of
     - 1.5 one or more arrays, of
     - 1.6 many cells
  2. The major complication is a large range of increasingly detailed options:
     - lots of parameters mean lots of potential trade-offs (e.g., between size, speed and power consumption),
     - need to take care of detail: there are so many cells, any minor change can have major consequences!
  3. Even so, there is just one key concept: we have some cells, and however they are organised we just need to identify and use the right cells given some address.



Notes:

## Additional Reading


- *Wikipedia: Computer Memory*. URL: https://en.wikipedia.org/wiki/Category:Computer_memory.
- D. Page. "Chapter 8: Memory and storage". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009.
- A.S. Tanenbaum and T. Austin. "Section 3.3.5: Memory chips". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.
- A.S. Tanenbaum and T. Austin. "Section 3.3.6: RAMs and ROMs". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012.
- W. Stallings. "Chapter 5: Internal memory". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013.




Notes:

# References


[1] *Wikipedia: Computer Memory*. URL: https://en.wikipedia.org/wiki/Category:Computer_memory (see p. 67).

[2] D. Page. "Chapter 8: Memory and storage". In: *A Practical Introduction to Computer Architecture*. 1st ed. Springer, 2009 (see p. 67).

[3] W. Stallings. "Chapter 5: Internal memory". In: *Computer Organisation and Architecture*. 9th ed. Prentice Hall, 2013 (see p. 67).

[4] A.S. Tanenbaum and T. Austin. "Section 3.3.5: Memory chips". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 67).

[5] A.S. Tanenbaum and T. Austin. "Section 3.3.6: RAMs and ROMs". In: *Structured Computer Organisation*. 6th ed. Prentice Hall, 2012 (see p. 67).


Notes: