Memory Systems and Timing Analysis

Microprocessor systems use memory ICs to store programs and data. This lecture describes common semiconductor memory devices and how they are organized in microprocessor memory systems.
After this lecture you should be able to select the appropriate type of memory device for different applications, combine memory ICs to form memory arrays, and design address decoders.
Timing analysis is the process of verifying that the timing requirements of each chip in a circuit are met. Unless all timing requirements are met the circuit may fail to operate properly under some conditions.
After this lecture you should be able to draw a timing diagram for a simple circuit, derive expressions for a chip's timing requirements from the timing diagram and compute the margin for each requirement based on clock periods and the guaranteed responses of the other components.
Larger microcomputer systems use Dynamic RAM (DRAM) rather than Static RAM (SRAM) because of its lower cost per bit. DRAMs require more complex interface circuitry because of their multiplexed address buses and because of the need to refresh each memory cell periodically.
After this lecture you should be able to: (1) describe basic DRAM structure and terminology, (2) interface DRAMs to a CPU bus by multiplexing row and column address lines and forcing refresh cycles, and (3) justify the choice of DRAM or SRAM for a particular application.
Cache memory can greatly increase the processing speed of microprocessors whose CPU cycle time is significantly shorter than the memory access time. Most modern general-purpose microprocessors include some sort of cache memory. In this lecture we will introduce cache memory by looking at the design of a direct-mapped cache.

Memory Devices

ROM

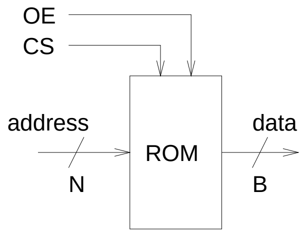
The simplest memory IC is a ROM (read-only memory). A ROM can be described as a combinational logic circuit that implements an arbitrary B-bit function of N bits. The N input bits are known as the *address* and the B output bits are the *data*. Such a device is referred to as a 2^N *by* B memory. For example, a 4096 by 8 (4kx8) ROM would have 12 address pin inputs and 8 data pin output.

Like any combinational logic circuit, a ROM can be described using a lookup table. The following table shows some of the contents (in hexadecimal) of a hypothetical byte-wide device:

address	data
0000	2E
0001	A3
0002	73
....	..
FFFF	D9

Exercise 69: What are the values of N and B for this device? When the values of the address inputs are 0002 (hex) what will be the values (in binary) of the outputs D_0 to D_7?

The following diagram shows the input and output pins on a typical read-only memory (ROM):

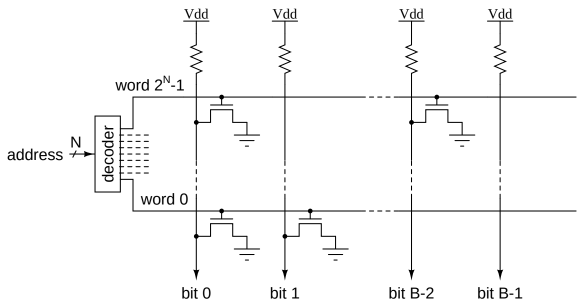


The address inputs are typically labeled A_0 to A_{N-1} and the data outputs D_0 to D_{B-1}. The CS (*chip select*) and OE (*output enable*) pins must be active for data to appear on the output.

Exercise 70: If we wanted to be able to connect the outputs of several memory chips in parallel, what "state" would the outputs have to be in when CS or OE were not asserted?

ROM Implementation

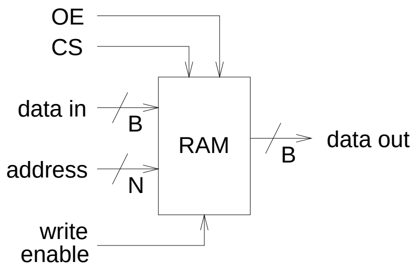
A ROM, like any other combinational logic circuit, could be implemented as a sum-of-products circuit. However, the large number of product terms needed (typically on the order of 2^{N-1}) make this approach impractical for large memories. Instead, an N-bit decoder is used to enable one of 2^N word lines. Each word line drives a number of transistors, each of which can can pull down one of the bit lines. By inserting or leaving out transistors, we can set the output values for each address.



Exercise 71: Assuming $N = 1$ and $B = 4$, what are the contents of the memory shown above?

RAM

A RAM (random-access memory) chip is a memory device that can be written as well as read. A RAM can be described as a sequential circuit in which N address inputs select one of 2^N of B-bit registers. The following diagram shows the essential pins on a (RAM):



During a *write* operation the CPU selects one set of B flip-flops by putting the desired address on the address pins and the data to be latched (stored) into the flip-flops on the data input pins. The write enable pin is used to latch the data into the flip-flops in the same way that a clock is used in a D flip-flop.

During a *read* operation a particular set of flip-flops is again selected by the address pins and the data previously stored in the flip-flops appears on the data output pins. In many cases a bidirectional data bus is used for both data input and data output.

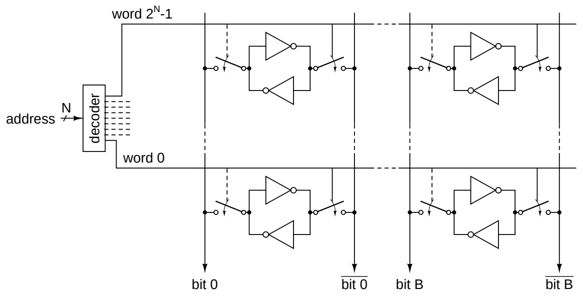
Note that the terms “read” and “write” are always described from the point of view of the CPU. For example, during a “read” cycle the CPU reads the memory contents (even though the memory chip is “writing” a value to it’s output pins).

As with a ROM, CS must be asserted for either operation and the OE pin must be asserted during a read operation.

RAM Implementation

A RAM can be implemented as a bank of flip-flops together with the associated decoders and multiplexers required to select the desired flip-flop output(s) and enable loading only the desired flip-flop input(s). A small RAM, such as a register file in a CPU, would be implemented this way.

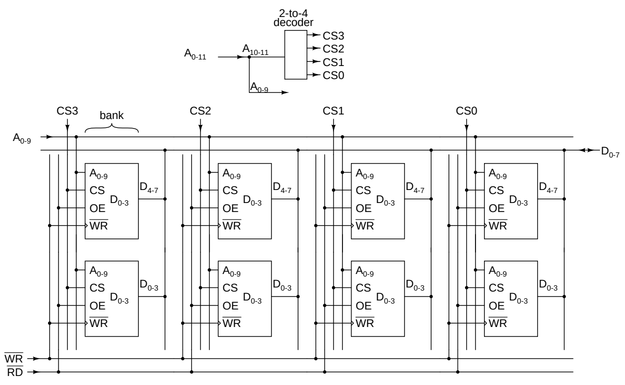
However, implementing large RAMs this way would be costly. Large RAMs use SR latches to store each bit of information. A decoder selects the set of latches to be read or written by enabling transmission gates that connect the complementary SR input/output nodes to circuits that can either alter or read the memory contents.



Combining Memory Chips

There are two ways in which memory chips can be combined: in parallel to increase the data bus width and in banks to increase the addressable memory size. If the width (the number of bits available in parallel) of the individual memory devices is less than what is required by the CPU, then memory devices can be combined in parallel by connecting their data bits to successive bits of the data bus. Multiple banks of these devices can then be combined to make available more memory than can be provided by one such set of ICs.

The following diagram shows an example of how memory chips can be combined to increase both the word size and the number of words available. In this example two 1kx4 RAMs are combined to form 1kx8 banks of memory and four such banks are combined to form a 4kx8 memory array. The 2-to-4 decoder selects one of four banks (using their CS inputs) according to the value of the 2 most significant address bits.

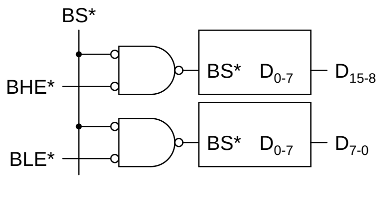


Exercise 72: Eight (8) 1Mx4 devices are to be connected to a CPU with a 16-bit data bus. How many address and data bits does each IC require? What is the total memory size in MBytes? Draw a block diagram showing the address and data bus connections to the different ICs. How many enables (used to drive CS pins) will be required from the address decoder?

Address Decoding

The memory space of a computer system can be considered to be an array of $2^{N_{cpu}}$ bytes where N_{cpu} is the number of CPU address bits. The memory (and possibly I/O devices) exist, or "are addressed," within this address space.

General-purpose processors use byte addressing - each address selects one byte. This means that not all of the CPU address lines may be explicitly available. For example, the 386SX processor has a 16-bit data bus. The least-significant bit of the address, A_0 is thus not visible. Instead, it is converted into BHE* and BLE*. This has to be taken into account in the design of the memory system (e.g. BHE* and BLE* are used as chip-select signals). For example, two of the above 8-bit-wide memory systems could be combined to create a 16-bit-wide memory using BHE* and BLE* select one or both bytes at a time:



Typically N_{cpu} is larger than the N for a memory device (the CPU can address more memory than can be supplied by one memory chip). For example, the 8088 has $N_{cpu} = 20$ address pins but we might want to use 32kB ($N = 15$) RAMs.

Combinational circuits called *address decoders* are used to enable a bank of memory devices when the address on the CPU bus lies in the desired range of 2^N addresses. The decoders' inputs are $N_{cpu} - N$ most-significant CPU bus address bits and its outputs are logic signals that enable the chips in one bank when the address falls within the desired address range. The two most common ways to implement address decoders are SSI (small scale integration) decoders and PLDs (programmable logic devices).

The number of words in a bank is always a power-of-two. Address decoders detect (decode) addresses that start on an address that is a multiple of this power-of-two. Therefore the address range can be written as a bit pattern that the decoder responds to. For example, a decoder for addresses from 2 0000H to 2 7FFFH ($N = 15$, 32 kBytes) would respond to addresses of the form `0010 0XXX XXXX XXXX XXXX` where the X's are "don't cares."

Exercise 73: If a decoded region spans 4 kBytes starting at address 1 0000H, what pattern of addresses will the memory respond to?

Decoder designs can be simplified by allowing the decoder to respond to multiple addresses. This is called *partial decoding*. In this case the decoder uses fewer than $N_{cpu} - N$ bits. For example, if the above decoder responded to addresses of the form `000X XXXX XXXX XXXX XXXX` the decoded region would extend from 0 0000H to 1 FFFFH (128kB) and the 32kB region would appear replicated in all four 32k blocks in that region. This partial decoding "wastes" part of the processor's address space because it is now unavailable for other devices. The advantage is that the extra "don't-care" bits results in a somewhat simpler implementation for the decoder.

Memory System Design

The design of a memory system involves the following steps:

1. determine the number of chips per bank by dividing the width of the CPU data bus by the width of the memory chips

2. determine the number of banks required from the total memory required and the memory provided by each bank

3. determine which CPU address lines will drive the memory chip address inputs and which CPU address lines will drive the bank-select decoder. This will be affected by the data bus size and whether full or partial decoding is used

SSI Decoders

A decoder such as the 74LS138 3-to-8 decoder can be used to divide up an address range.

Exercise 74: How would you connect a '138 to divide up the 8088's 20-bit (1 MByte) address space into eight 128-kByte regions?

PLD Decoders

Since PLDs can generate complex combinational functions they can be used to divide up a memory space into regions of different sizes.

Exercise 75: We want to design a PAL (a type of PLD) decoder that selects one 64kB region from 0 0000H to 0 FFFFH and one 256 kB region from C 0000 to F FFFFH out of the 8088's address space. How may input bits are required? How many outputs? Write the VHDL expressions for signals `sel1` and `sel2` assuming the address bits are declared as `a : in bit_vector (19 downto 0) ;`.

Memory Technologies

A wide variety of IC memory devices are available. They vary in terms of data permanence, power consumption, cost, capacity, and access time.

SRAM

Static Random-Access Memory. This is volatile read/write memory. Data is stored as the state of a flip-flop. The contents are lost when power is removed. CMOS devices have very low power consumption when not being accessed and can be used with battery or capacitor backup. Bipolar devices have higher power consumption but feature the shortest access times.

DRAM

Dynamic RAM. In dynamic RAM the data is stored as the charge on a capacitor. These devices have the lowest cost per bit for RAM. The contents are lost if memory locations are not accessed (refreshed) every few milliseconds. Capacity about 4 times that of same-generation SRAM.

Mask-Programmed ROM

Programmable Read-Only Memory. Non-volatile read-only memory. Data is stored as connections between gates. The memory contents are determined at time of manufacture. Lowest cost per bit of any memory but have large NRE ("non-refundable engineering") costs which makes them suitable only for large volume applications.

EPROM

Erasable Programmable Read-Only Memory. Field-programmable non-volatile ROM that can be erased by exposure to UV light. Data stored as charge on "floating gates." Typically have byte-wide organization. Not as fast or dense as RAM. OTP (one-time programmable) devices are less expensive since the packages don't have windows (and can't be erased).

EEPROM and Flash

Electrically-Erasable PROM. Non-volatile memory that can be written like RAMs. Write cycles are relatively long (hundreds of microseconds). Relatively small capacity. Limited to several thousand write cycles. Mostly used to store infrequently-changed configuration information.

Flash EEPROMs are similar to EEPROMs but whole sections of the chip must be erased and then re-written. This simplifies the design of each cell. This reduces cost and increases capacity.

Specialized Memories

FIFOs are memory devices that incorporate the logic required to implement first-in first-out (FIFO) queues. They are typically used as buffers between high-speed interfaces.

Video RAM is sometimes used in video display circuits. They have a second independent data output used that sequentially reads out a complete row of data from the memory array. This "dual ported" arrangement allows both the CPU and the video signal generator to access the RAM at the same time. This reduces contention and improves performance.

Serial EEPROMs are EEPROMs whose contents must be read or written one bit at a time from start to end. The serial interface reduces chip size, pin count and cost (e.g. a 128x8 EEPROM in an 8-pin DIP for $1). Typical application is storing configuration information. The slow serial access is not a drawback since the device is typically accessed only when the product is turned on (or off).

Exercise 76: What type of memory device(s) would likely be used in a popular PC for the following purposes: storage for the power-on boot code? the "CMOS" configuration memory? the main data/program RAM? the main memory cache? the boot program in a prototype of this PC? the video display memory?

What type of memory device(s) would likely be used for the following applications: non-volatile memory for user programs in a calculator? a font card for a laser printer? user-upgradeable storage for the firmware in modem? the mileage reading in a digital car odometer? a video game cartridge?

Timing Specifications

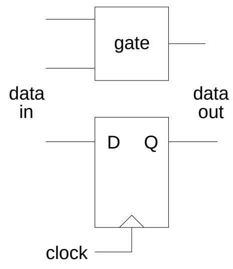
A chip's timing specifications are of two types: (1) timing requirements and (2) guaranteed responses. All timing specifications are measured between transitions from low to high (rising edge) or high to low (falling edge) – of a chip's inputs and outputs.

Guaranteed responses are delays between an edge on an input (or output) signal and the transition to the correct level on an *output* signal. A chip's manufacturer guarantees that this specification will always be true if the chip is operated within it's recommended limits. A typical guaranteed response is a propagation delay.

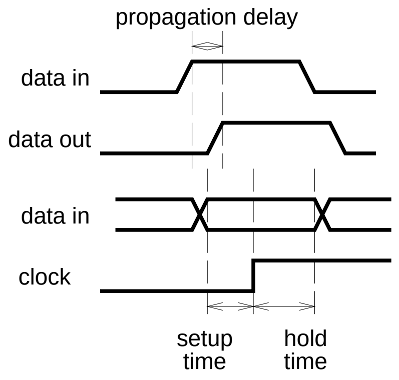
Timing requirements are the time relationships between a transition on an output (or input) signal and the transition to the correct level on an *input* signal. A chip's manufacturer guarantee the correct logical operation of the chip if all of the requirements are met. Typical examples of timing requirements are setup and hold times.

A simple rule to distinguish between these two is: Timing requirements are measured *to* an edge on an input signal while and guaranteed responses are measured *to* an edge on an output signal.

The diagram below shows the simplest examples of the two types of circuits: a logic gate (an example of a combinational circuit) and a D flip-flop (an example of a sequential circuit):



and the diagram below shows the three most common timing specifications:



The most common guaranteed response is the *propagation delay* which is the maximum delay between a change in the input and the correct value appearing at the output. A similar guaranteed response during a memory device read cycle is the *access time*. This is typically measured from the address, chip-select, or output-enable signals changing to when the data outputs become valid.

The most common timing requirements are the *setup time* and *hold time* which are the minimum durations that the data input to a flip-flop has to be at the desired value before and after the relevant clock edge. Setup and hold times also apply to memory device write cycles. These are typically measured between the address, data, or control signals changing to the edge of the write strobe that ends the write cycle.

Exercise 77: Which of the three basic specifications (delay, setup and hold times) would apply to a multiplexer? To a RAM chip? To a ROM?

Exercise 78: Draw timing diagrams for a ROM read cycle showing the address, CS*, OE*, and data signals and for a RAM write cycle showing the address, CS*, OE*, WR*, and data signals. Label the guaranteed responses and timing requirements.

During read cycles the data output by a memory device is loaded into a CPU register. Thus the CPU read cycle timing specifications usually include requirements similar to setup and hold times. Similarly, during write cycles the data output by the CPU is loaded into a memory device. Thus CPU write cycle timing specifications usually include guaranteed responses similar to propagation delays. Often the CPU manufacturer will quote timing requirements relative to clock edges rather than to read or write strobe edges.

In addition to the three fundamental specifications many chips may either require or guarantee a minimum/maximum *pulse width*s on certain signals and/or minimum/maximum *cycle time*s (waveform period) or frequencies.

Timing Diagram Conventions

Timing diagrams help to clarify the meanings of timing specifications by labeling the times between signal transitions (“edges”) using symbols from tables of timing specifications.

Some conventions used in timing diagrams are:

- high and low levels shown at the same time indicate the signal is not changing but can have either value (e.g. a data signal)

- shading between two levels indicates that the value is allowed to change during this time

- a line half-way between the two logic levels indicates that the signal is in high-impedance (“tri-state”) state

- arrows drawn between transitions on different signals show that one signal transition causes or affects another

- sloped transitions between levels allow references to the signal reaching a low (V_{OL}/V_{IL}) or high (V_{OH}/V_{IH}) value

It is important to understand that timing diagrams are *not* drawn to scale. This allows chips with different specifications to share the same timing diagram, allows small delays to be shown more clearly and also allows the same label on the diagram to refer to both maximum and minimum values. You can’t even rely on the timing diagram to show the order in which signal transitions will happen.

Timing Analysis

Timing analysis should be part of every digital system design. After a preliminary circuit design the designer must verify that all timing requirements for all devices will be met. It is not sufficient to build a prototype and demonstrate that it works properly since the actual timing characteristics will vary from chip to chip and as a function of temperature and supply voltage. If a timing analysis is not done before a design is put into production the consequences could be serious.

A timing analysis is most conveniently summarized in the form of a table. The first step in a timing analysis is to consult the data sheets for all of the devices in the circuit and prepare one table for each chip and for each possible type of cycle (read, write, etc). Each line in each table should list one timing *requirement* and it’s minimum or maximum value.

Next, draw timing diagrams that show the waveforms of the relevant signals with labels indicating the timing specifications. This may include signals generated by clocks, by the microprocessor, by memories and by interface circuits such as address decoders and buffers. Often there will be several timing diagrams for different parts of the circuit or for different sequences of signals (e.g. different bus cycles). The timing diagrams are then used to obtain expressions for each timing requirement in terms of the guaranteed timing responses of the other devices and, usually, clock periods.

The expressions are derived by expressing each requirement in terms of variables representing clock periods and other chips' guaranteed responses. These expressions are then entered into the table.

When values are substituted for the variables in the equations, a minimum (or possibly maximum) value is obtained for that requirement in that specific circuit. The difference between the computed requirement and the manufacturer's specified requirement is called the *margin*. For example, if a manufacturer specifies that a certain flip-flop requires a 10 ns setup time and in a particular circuit the setup time is guaranteed to be at least 50 ns then the margin is 40 ns.

On the other hand, if any of the margins are negative then the chip's timing requirements are not met and the design must be changed. Typical changes include:

- adding CPU wait states
- reducing the clock frequency
- registering signals to extend them
- using redundant logic gates to add small delays (poor practice)

Heres' an example of part of the timing analysis for a RAM chip:

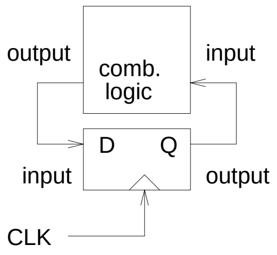
XYZ124-10 RAM write cycle

requirement			guaranteed		
name	symbol	value	expression	value	margin
setup time	t_{SU}	10	$= t_{CLK} - t_{PD}$	100-5 = 95	=85
…					

The first three columns are copied from the chip's spec sheet. The expression is obtained from inspection of the timing diagram. The value of the expression and the margin are computed.

Example

As a simple but complete example, consider a simple state machine where a combinational circuit computes the next state based on the current state and the input:



Exercise 79: Draw separate timing diagrams for the flip-flop and the combinational circuit. Assume the flip-flops require a minimum setup time, t_s, of 20ns and a minimum hold time, t_h of 0 ns. Assume the maximum clock-to-output propagation delay for the flip-flop is $t_{CO} = 5$ ns (again, with no minimum). Assume that the maximum propagation delay through the combinational logic circuit is guaranteed to be a maximum of $t_{PD} = 20$ ns, and there is no minimum for t_{PD}. Label the timing diagrams with each of these specifications.

Exercise 80: Draw a timing diagram for the complete circuit. It should include the clock CLK, the flip-flop's output, Q, and its input, D. Indicate cause–effect relationships between the signal edges using arrows.

Derive expressions for each timing requirement in terms of the clock period and guaranteed timing specifications for a clock frequency of 10 MHz. Substitute the actual values and compute the remaining margin. Will this circuit operate properly as far as timing is concerned? What if the hold time requirement was 5 ns?

Dynamic RAM

DRAM Structure

A description of the structure of a DRAM helps explain some of their unique features.

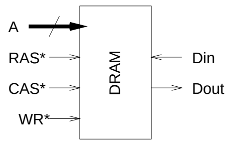
A typical DRAM memory is laid out as a square array of memory cells with an equal number of rows and columns. Each memory cell stores one bit. The bits are addressed by using half of the bits (the most significant half) to select a row (the *row address*) and the other half to select a column (the *column address*).

Exercise 81: How many rows and columns would there be in the memory cell array in a 16 M by 1 (16 Mbit) DRAM?

Address Multiplexing

In order to reduce the number of pins on the DRAM chip and thus reduce the size of the DRAM chip, most DRAM chips multiplex the row and column addresses onto the same set of pins.

Two strobes, RAS* (row address strobe) and CAS* (column address strobe) tell the chip which half of the address (row or column) is currently on the address pins. To further reduce the chip count the CAS* signal typically acts as an output enable when R/W* line is high (read) and the falling edge acts as a write strobe when R/W* is low (write).

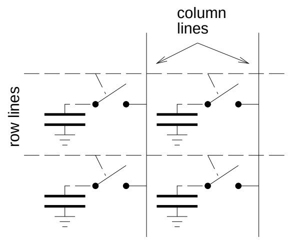


Exercise 82: How many address pins would be found on a typical 16 M x 1 DRAM?

DRAM Access and Refresh

To maximize the capacity of the DRAM, each memory cell is very simple – it consists of a capacitor and a FET switch. A DRAM memory cell is therefore much smaller than an SRAM cell which requires two inverters and two transmission gates.

The following diagram shows the structure of DRAM array using switches in place of transistors:

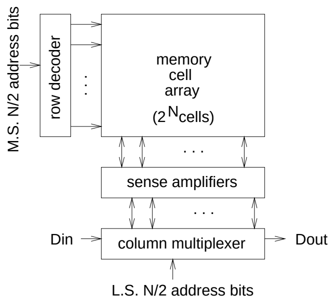


The *row* address drives a decoder which enables the appropriate *row-select* signal. This row-select line turns on all of the FET switches in that row and connects each the capacitors in the selected row to its *column* line. Note that each vertical column line is shared by all the memory cells in a column although only one capacitor is connected to the column at any time.

The charge stored in each memory cell capacitor is relatively small so each column line is connected to a "sense amplifier" which amplifies the voltage present on the line while RAS* is asserted.

When CAS* is asserted the output of the sense amplifiers is driven back onto the column lines and recharges the capacitors. Thus each memory access *refreshes* the contents of that row.

The column address also drives a multiplexer which selects one of the column lines and connects it to the output, thus reading out the value of a single bit. During a write, the value on the input over-rides the sense amplifier value for the addressed column and this stores new data into the desired memory cell.



DRAM Timing

In addition to the DRAM timing requirements of setup and hold times for the row and column addresses, DRAMs also require a minimum "pre-charge" time between the end of RAS* or CAS* and the start of the next cycle. This "recovery" time is needed to re-charge the storage capacitors. This pre-charge time extends the minimum cycle duration to considerably more than the access time. For example, a DRAM with a 60 ns access time may have a minimum cycle time of 100 ns.

Figure 1 shows timing diagrams and specifications for a typical DRAM. The sequence of operations required to read or write from a DRAM both start in the same way:

- set R/W* to the appropriate value and place the row address (the MS half of the address) on the address pins,
- wait the RAS* setup time, bring RAS* low, and wait for the RAS* hold time

In order to read from the DRAM:

- place the column address on the address pins, wait the CAS* setup time, bring CAS* low, and wait for the CAS* hold time

- wait until the access times from both CAS* and RAS* are met and then read the data from the data out pin

In order to write to a DRAM the sequence is similar except that during a write cycle the data is latched on the falling edge of CAS*:

- place the column address on the address pins, data on the data input pin, wait for the CAS* and data setup times, bring CAS* low, wait for the CAS* and data hold times.

At the end of either cycle we must then bring RAS* and CAS* high and wait the pre-charge (recovery) time before starting another cycle.

Exercise 83: Draw a timing diagram for a DRAM read cycle showing the address lines, RAS*, CAS*, WR* and the data pins. Show on the timing diagram the following specifications: address setup and hold times from RAS* and CAS* active, access times from RAS* and CAS* active, minimum "pre-charge" times (from RAS* or CAS* inactive), and the minimum cycle time (from RAS* to RAS*).

For a write cycle, show the setup and hold times for D_{in} from CAS* active.

Refresh

Since the DRAM storage capacitor discharges over time it must be refreshed periodically. The DRAM's structure ensures that all the memory cells in a row are refreshed every time that row is read. Therefore it is only necessary to periodically cycle through all of the *row* addresses to refresh all of the bits in the memory array.

The simplest technique to provide DRAM refresh is to include a device (such as a DMA controller or video display circuit) that accesses the RAM in such a way that all of the rows are accessed at least once during the minimum refresh time (typically every few tens of milliseconds). This is called *RAS*-only refresh* because it's not necessary to assert CAS* in order for the refresh operation to take place.

Another technique is to add a circuit that periodically forces a cycle in which CAS* is asserted before RAS*. This is called *CAS* before RAS* (CBR) refresh*. Modern DRAMs have an an internal refresh counter that cycles through the possible row values. On these DRAMs the CAS* before RAS* operation causes an internal row-refresh operation. The advantage of this type of refresh is that the refresh controller need only control RAS* and CAS*, it need not generate the refresh row addresses.

Exercise 84: Assume a microprocessor with a 200 ns memory cycle time is using a 1 MByte DRAM with a maximum refresh time of 10 ms. How many row addresses will have to be refreshed every 10ms? What is the approximate time between each refresh cycle? How many memory cycles are there per refresh cycle? What percentage of the memory accesses are "wasted" on refresh cycles?

DRAM versus SRAM

Since the SRAM devices require more circuitry per memory element they are more expensive (per bit) to produce and have lower density per chip. The typical ratio between DRAM and SRAM for the same chip (die) size is about 4 to 1.

The disadvantages of DRAMs are that they require additional control circuits to multiplex address lines and to handle refresh. If DRAMs are organized as bit-wide devices it is necessary to use a number of devices that is a multiple of the the data bus width (8, 16 or 32) in a system.

The use of large DRAM arrays in which the CPU address and data buses must drive many chips usually requires buffers for the address and data lines. Because of these reasons DRAMs are mainly used in systems that require large memories and SRAMs are mainly used in smaller systems.

The fastest RAM designs are static, so SRAMs are often used for high-speed memories such as cache or address translation tables.

CMOS SRAMs consume very little power when not being accessed so they are often used in low-power designs. With the use of battery (or a large capacitor) backup they can retain their contents for months. On the other hand, since DRAMs must be continuously refreshed, their power consumption cannot be reduced to very low values.

Due to the larger number of bits per chip and wider bus sizes, DRAMs are now being offered in nybble (4-bit) and larger organizations. The number of extra pins required to provide the additional data bits is less of a concern with modern high density packages such as QFP and SOIC.

Exercise 85: Consider a system using 16 Mbit X1 memories to design a memory array for a microprocessor system with a 32-bit data bus. What is the minimum amount of memory that could be provided using these devices?

EDO, FPM and SDRAM

Many DRAMS include an extended data out (EDO) feature, in which the output data is latched and held past the end of a read cycle and into the start of the next cycle in order to help satisfy CPU hold times.

Due to the need to charge the DRAM capacitors, the shortest practical cycle times are currently about 50 ns. If the CPU cycle time is less than this then wait states must be inserted for each memory access. This would limit the performance of processors with cycles times less than about 50 ns.

Instead of accessing the DRAM memory directly, these faster computers use fast SRAM memories that record ("cache") values as they are read from the main memory. When the CPU accesses a value that is already in the cache it can retrieve it much faster than it could from DRAM.

Instead of retrieving a single value from the main memory, cache controllers are designed to retrieve the values from several (e.g. 4) consecutive memory locations in a burst. Fast Page Mode (FPM) and Synchronous DRAM (SDRAM) memories provide fast access to consecutive memory locations and minimize the total time required for a burst read.

Since the contents of each row are read with each RAS* operation, it is possible to obtain the values of more than one column within a given row without having to re-read the row. Some DRAMs support such a "fast page mode" in which multiple CAS* cycles may be used to access several addresses within one row ("page") after one RAS* cycle. These accesses within one row are much faster than accesses to separate rows (e.g. 10 ns versus 50 ns).

It is also possible to design DRAMs to use multiple internal memory arrays, all of which are accessed in parallel. The arrays are assigned to consecutive memory locations and, for the same reasons as FPM DRAM, this allows faster access to consecutive memory locations.

High-speed DRAMs use a synchronous (clocked) interface. The cache/DRAM controller writes the desired staring memory location and word count into registers in the DRAM, waits a fixed number of clock cycles, and then reads one word per clock cycle.

Cache Memory

Locality of Reference

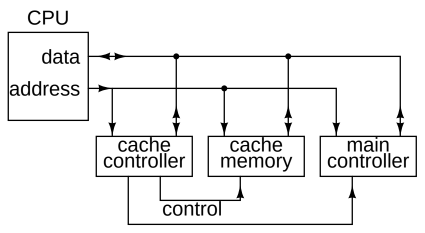
During the execution of most programs the CPU typically accesses a few memory locations very often. Furthermore, these locations are often close together. As an example, consider the following C function:

```
int strlen(char *p)
{
        int len=0 ;
        while (*p++ != 0 ) len++ ;
        return len ;
}
```

the variables `len` and `p` will be accessed repeatedly[1] as will the instructions within the loop.

To exploit this locality of reference we can use small high-speed memories to temporarily store the values of these often-used memory locations. If access to this *cache* memory can be done more quickly (i.e. with fewer wait states) than access to main memory we can reduce the overall execution time.

The structure of a cached memory system is shown below:



The CPU can read from the cache memory or the main memory and can write to the cache, main memory or both. The main complexity of a cache is designing the controller because it must keep track of which values in the cache are valid and what memory locations they represent.

Effects of Hit Rate and Access Times

If we define *k* as the ratio of the time required to retrieve a value from cache as opposed to from memory and *H* as the average fraction of time a memory

[1]In a proper system this function would be in-lined, and use only the variable 'p' which would be stored in a register, but the principle still applies anyways...

reference can be obtained from the cache, we can derive the speed improvement factor in memory access time due to the use of cache to be:

$$S = \frac{1}{1 - H(1-k)} \tag{1}$$

Note that H (hit rate) must be reasonably large for k to have an effect. Also note that k will depend not only on the memory access time but also on the length of the bus cycle. For example the bus cycle on a 68000 CPU at 8 MHz is long enough that no wait states are needed for main memory access. This means k is 1 and there is no advantage to using cache memory.

On the other hand consider a 50 MHz RISC processor that has a 20 ns bus cycle when reading from the cache and a main memory bus cycle of 100ns (k=0.2). Assuming the cache size and program structure result in a cache hit rate of 90%, then the use of the cache will speed up the processor by a factor of $1/(1-0.9(1-0.2)) = 3.6$.

Direct-Mapped Cache

The simplest type of cache memory organization (and the only one we will study) is the direct-mapped cache.

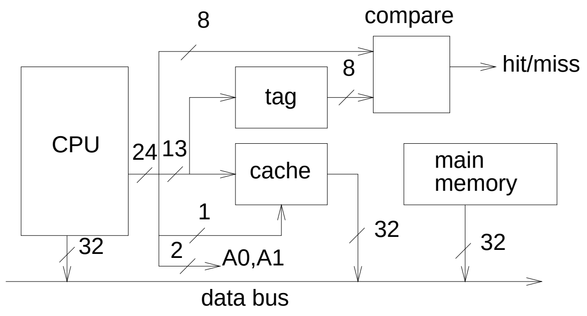
Both the main and cache memories are divided into *lines*. A line consists of a small number of contiguous bytes (e.g. 8). Lines are transferred between the cache and main memory as a complete unit. When a cache line is updated during a main memory read, the complete line must be updated. For example, if we are using 8-byte lines and 32-bit-wide memory the main memory accesses need to be done in pairs.

A direct-mapped cache that holds 2^N bytes uses the least significant 3 bits of the address (for 8-byte lines) to select a byte within a line, and the next least significant $N-3$ address lines to select a particular line. For example, a 64 kB cache with 8-byte lines would use the CPU address lines A15 to A3 to select a particular line from the cache.

When the CPU does a memory access, the cache control circuitry must decide whether the particular address is stored in the cache memory. If it is, then the CPU reads the cache, otherwise it must read the value from main memory. At the same time as a value is read from the main memory the appropriate cache line is updated.

The cache controller decides whether the particular line in the cache corresponding to that address is the desired one or not by using a lookup table stored in a "tag" RAM. The value stored in each entry in the tag RAM corresponds the remaining most-significant bits of the address that the line was read from.

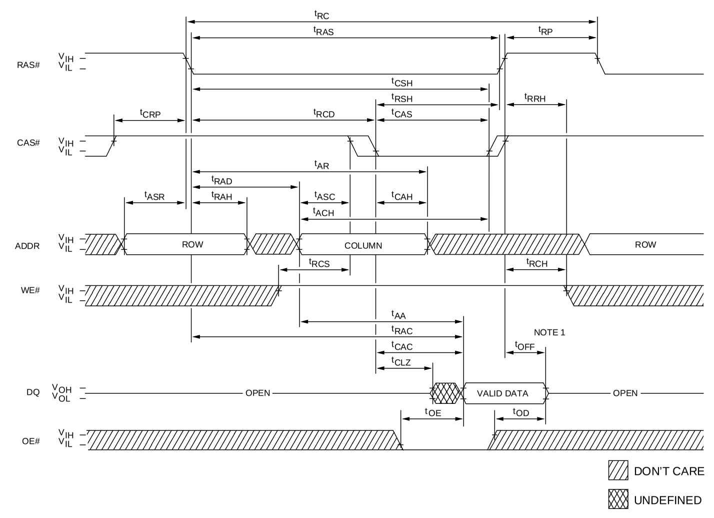
For example, a 64kB cache with 8-byte lines holds 8k lines. If the cache can support a main memory size of 16 MBytes (24 bits) then the tag RAM must store 8 bits for each line. The 24 address bits going to the cache are divided into 3-bits to select a byte from a line, 13 bits to select a line from the cache and 8 bits that are compared to the tag for that line). The 13 bit line select forms the address input to an 8Kx8 tag RAM and the 8 output bits are compared to the 8 most significant bits of the desired (24-bit) address.



Write-Back versus Write-Through

When the CPU writes a value to memory, the cache controller can either update both the main memory and the cache ("write through") or it can update just the cache ("write-back"). Although the latter is faster it means that we must keep track of cache lines that have been updated in cache but not in main memory. When we get a cache miss on a line we need to write that line back to main memory before it can be re-read from main memory. This means that the cache controller design is more complicated and requires a 1-bit memory (called a "dirty bit" memory, e.g. 8kx1) for each line in the cache.

READ CYCLE



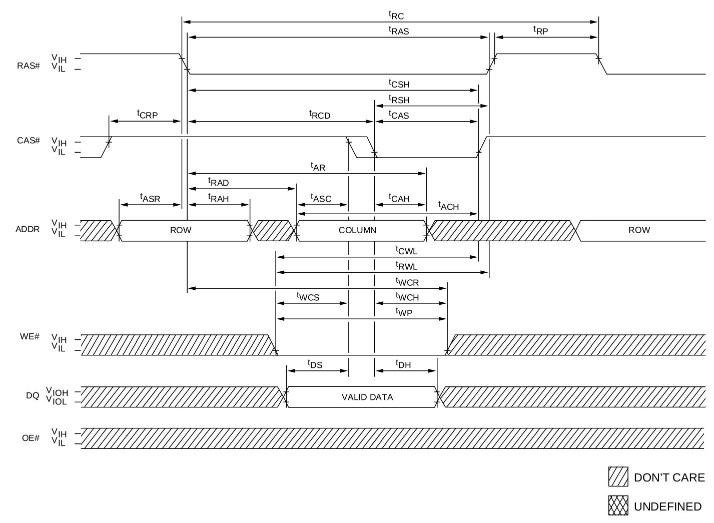
TIMING PARAMETERS

SYMBOL	-5 MIN	-5 MAX	-6 MIN	-6 MAX	UNITS
tAA		25		30	ns
tACH	12		15		ns
tAR	38		45		ns
tASC	0		0		ns
tASR	0		0		ns
tCAC		13		15	ns
tCAH	8		10		ns
tCAS	8	10,000	10	10,000	ns
tCLZ	0		0		ns
tCRP	5		5		ns
tCSH	38		45		ns
tOD	0	12	0	15	ns
tOE		12		15	ns

SYMBOL	-5 MIN	-5 MAX	-6 MIN	-6 MAX	UNITS
tOFF	0	12	0	15	ns
tRAC		50		60	ns
tRAD	9		12		ns
tRAH	9		10		ns
tRAS	50	10,000	60	10,000	ns
tRC	84		104		ns
tRCD	11		14		ns
tRCH	0		0		ns
tRCS	0		0		ns
tRP	30		40		ns
tRRH	0		0		ns
tRSH	13		15		ns

NOTE: 1. tOFF is referenced from rising edge of RAS# or CAS#, whichever occurs last.

EARLY WRITE CYCLE



TIMING PARAMETERS

SYMBOL	-5 MIN	-5 MAX	-6 MIN	-6 MAX	UNITS
tACH	12		15		ns
tAR	38		45		ns
tASC	0		0		ns
tASR	0		0		ns
tCAH	8		10		ns
tCAS	8	10,000	10	10,000	ns
tCRP	5		5		ns
tCSH	38		45		ns
tCWL	8		15		ns
tDH	8		10		ns
tDS	0		0		ns
tRAD	9		12		ns

SYMBOL	-5 MIN	-5 MAX	-6 MIN	-6 MAX	UNITS
tRAH	9		10		ns
tRAS	50	10,000	60	10,000	ns
tRC	84		104		ns
tRCD	11		14		ns
tRP	30		40		ns
tRSH	13		15		ns
tRWL	13		15		ns
tWCH	8		10		ns
tWCR	38		45		ns
tWCS	0		0		ns
tWP	5		5		ns

Figure 1: Typical DRAM specifications.