Contents

Chapter 1
Main memory

Chapter goals

In this chapter, we will cover the modern memory design and operations in memory chips and modules that enable efficient data transfer between the memory controller and so called DIMMS, i.e., memory modules used in modern computer systems. To understand the organization and operation of modern memory chips fully, we need to start with some fundamental digital building blocks. Then, we gradually build memory components, arrays, operations inside the memory chips, timings and the techniques to boost the performance of memory chips. At the end of the chapter, you should fully understand modern DDR SDRAM chips, the DDR memory technology, memory timings, DIMM modules and multi-channel architecture.
From this chapter, you should gain a basic understanding of the design and operation of computer memory and storage circuits including:

- Static memory circuits using the six-transistor cell
- Dynamic memory circuits including the one-transistor cell
- Sense amplifier circuits required to detect the information stored in the memory cells
- Sense amplifier circuits required to detect the information stored in the memory cells
- Overall DRAM memory chip organization

1.1 Introduction

We are now already familiar with CPUs and caches. In this chapter, we focus on the main memory used in modern computer systems as one in Figure 1.1. Figure 1.1 illustrates the memory hierarchy in the Intel i7-860 based system. Intel i7-860 is an out-of-order execution processor that includes four cores. The L1 and L2 caches

are separate for each core, while the L3 cache is shared among the cores on a chip. The L1 cache is the 32 KB, four-way set-associative cache. There are two L1 caches per core: instruction (I) and data (D) . The L2 cache is the 256 KB, eight-way set-associative cache. Finally, the L3 cache is the 8 MB, 16-way set-associative cache.

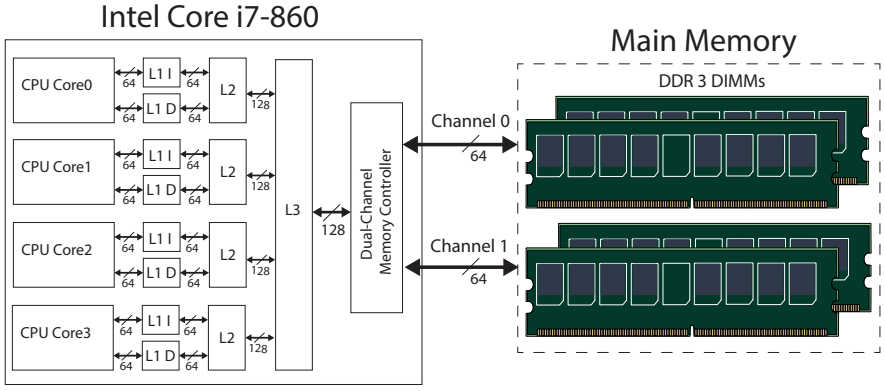


Fig. 1.1: Intel i7-860 memory hierarchy

A CPU core directly accesses only its L1 cache. If a hit in L1 occurs, the data is returned after an initial latency of 4 cycles. If the L1 cache misses, the L2 cache is accessed. If a hit in L2 occurs, the block of size 64B is returned after an initial latency of 10 cycles at a rate of 8 bytes per clock cycle. If the L2 cache misses, the L3 cache is accessed. If a hit occurs in L3, the 64-byte block is returned after an initial latency of 35 cycles at a rate of 16 bytes per clock. If L3 misses, memory access is initiated - the on-chip memory controller must get the block of size 64B from the main memory.

The main memory is implemented of DDR3 memory chips placed on the printed circuit boards called Dual In-Line Memory Module (DIMM). The memory controller on i7-800 supports two 64-bit memory channels. Each channel is used to access eight 8-bit memory chips placed on one side of DIMM (64 bits per access). Two 64-bit memory channels are used simultaneously as one 128-bit channel (since there is only one memory controller, and the same address of the missing block in L3 is sent on both channels) to fill the missing block in L3. Thus, the memory controller fills the 64-byte cache block at a rate of 16 bytes (124 bits) per memory clock cycle.

Have you struggled reading the description of the memory hierarchy in the Intel i7-860 based system? Don't worry, at the end of this chapter you should be able to understand it. Let us now begin our journey into the world of modern memory.

1.2 Basics of Digital Circuits: A Quick Review

Before looking under the hood of modern memory chips used in the computer systems, we should apprehend some basic concepts from digital electronics like MOS transistors used as logical switches and MOS inverters. The aim is to understand the operations in modern memory chips and not to fall into the physical equations of electronic circuits. Therefore, the description of the basic concepts of digital circuits will be significantly simplified.

The basic building block of all digital circuits is the MOS transistor. MOS is an acronym for Metal-Oxid-Semiconductor and indicates the manufacturing process used to make transistors. The MOS transistor has three terminals: gate (G), drain

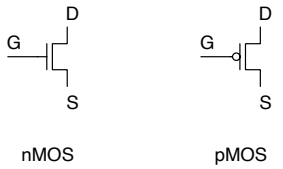


Fig. 1.2: nMOS nad pMOS transistor symbols.

(D) and source (S). The gate terminal is a control input: it controls the flow of electrical current between the source and drain terminals. There are two types of MOS transistors: nMOS and pMOS. Figure 1.2 shows the symbols of both types of MOS transistors. We will consider only the type of operation where MOS transistors act as logical switches.

1.2.1 MOS transistor as a switch

Consider first an nMOS transistor. If the gate terminal is grounded (logical 0), no current flows between drain and source. Hence, we say the transistor is OFF. If the gate voltage is high and corresponds to logic 1, a conducting path of electrons is formed from source to drain, and current can flow. We say the transistor is ON.

The reverse holds for a pMOS transistor. When the gate is at a positive voltage that corresponds to logic 1, no current flow, so the transistor is OFF. A sufficiently low gate voltage that corresponds to logic 0 forms a conducting path from source to drain, so the transistor is ON.

In summary, the gate of a MOS transistor controls the flow of current between the source and drain. Simplifying this to the extreme allows us to **view the MOS transistors as ON/OFF switches**. When the gate of an nMOS transistor is 1, the transistor is ON, and the current flow between source to drain. When the gate is 0, the nMOS transistor is OFF, and no current flows between source to drain. A pMOS

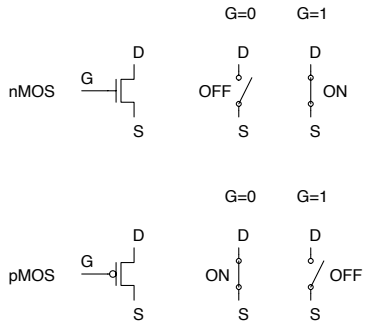


Fig. 1.3: Switch-level models of nMOS nad pMOS transistors.

transistor is just the opposite, being ON when the gate is low and OFF when the gate is high. Figure 1.3 illustrates this switch model.

1.2.2 CMOS inverter

The most straightforward logic gates that can be built using MOS transistors are an inverters. An inverter is built from two **complementary** MOS transistors, one nMOS, and one pMOS, hence the name *complementary MOS (CMOS) inverter*. Figure 1.4 shows the schematic and the switch-level model for a CMOS inverter or NOT gate using one nMOS transistor and one pMOS transistor. The bar at the top of the schematic indicates a supply voltage (Vdd), and the triangle at the bottom indicates the ground terminal (GND). The input IN connects both transistors' gates. When the input IN is 0, the nMOS transistor is OFF, and the pMOS transistor is ON. Thus, the output OUT is pulled to logic 1 because it is connected to Vdd through the pMOS transistor. Conversely, when IN is 1, the nMOS is ON, the pMOS is OFF, and OUT is pulled down to '0', because it is connected to GND through the nMOS transistor.

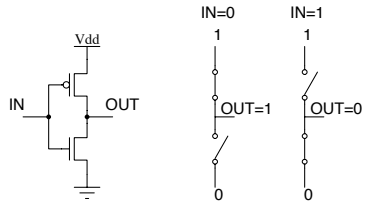


Fig. 1.4: CMOS inverter and its switch-level models.

1.2.3 Bistable element

Now, as we are familiar with MOS transistors and CMOS inverter, it is time to learn how we can store one bit of information in a MOS digital circuit, i.e., how to form a 1-bit storage (memory) cell using MOS transistors and inverters. The fundamental building block of memory is a **bistable element** - a logic element with two stable states. Figure 1.5 shows the bistable element composed of two inverters, I1 and I2. The inverters are cross-coupled, meaning that the input of I1 is the output of I2 and vice versa.

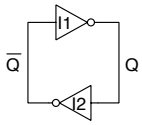


Fig. 1.5: A bistable element.

If $Q = 0$, I2 receives a FALSE input, so it produces a TRUE output on $\overline{Q}$. I1 receives a TRUE input, so it produces a FALSE output on Q. This is consistent with the original assumption that Q=0, so the circuit is in the stable state. If $Q = 1$, I2 receives a TRUE input, so it produces a FALSE output on $\overline{Q}$. I1 receives a FALSE input, so it produces a TRUE output on Q. This is consistent with the original assumption that Q=1, so the circuit is again in the stable state. Because the cross-coupled inverters have two stable states, 0 and 1, the circuit is said to be **bistable**. The state of the cross-coupled inverters is contained in one binary state variable, Q. Specifically, if $Q = 0$, it will remain 0 forever, and if $Q = 1$, it will remain 1 forever. Although the cross-coupled inverters can store a bit of information, they are not practical because the user has no inputs to control the state. So, we have to expand the bistable element with a circuitry, which provides inputs to control the value of the state variable. One such element that can accept the inputs to control the value stored in the bistable is a *static RAM cell*.

Summary: Transistors, Inverter and Bistable

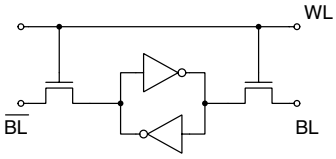
The gate of a MOS transistor controls the flow of current between the source and drain. Simplifying this to the extreme allows us to *view the MOS transistors as ON/OFF switches*.

An inverter is built from two *complementary* MOS transistors, one nMOS, and one pMOS.

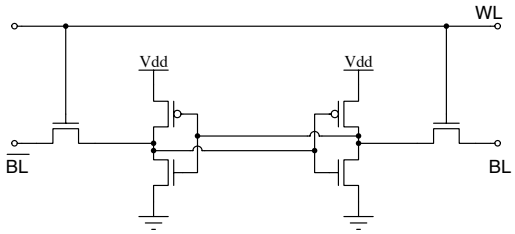
The fundamental building block of memory is a *bistable element*. It is composed of two cross-coupled inverters. It stores one bit of information.

1.3 SRAM cell

Static random-access memory (static RAM or SRAM) is a type of random-access memory (RAM) that uses a bistable element to store one bit of information. This is the type of memory used as the building block of most caches because of its superior performance over other memory structures, specifically DRAM, which we will cover later. SRAM is faster and more expensive than DRAM; it is typically used for CPU cache and registers while DRAM is used for a computer's main memory.



(a) A basic structure of a SRAM cell.



(b) 6-transistor SRAM cell.

Fig. 1.6: SRAM cell.

A typical SRAM cell is made up of six MOS transistors - two complementary pairs that form two cross-coupled inverters (bistable), and two *access* nMOS transistors that serve as a switch used to control the state of the bistable element during the read and write operations. Figure 1.6 shows an SRAM cell. Each bit in an SRAM cell is stored in the bistable element composed of four transistors that form two cross-coupled inverters. As we have already learned, this cross-coupled connection creates regenerative feedback that allows it to *store a single bit of data indefinitely* provided that power is supplied to the SRAM cell. The SRAM cell also has two bit lines that control both the input and output of the data from the cell. The first bit line (BL), holds the same value that is stored in the cell. The second bit line ($\overline{\text{BL}}$) holds the inverse of the value that is stored in the cell.

When the word line (WL) is not selected (WL=0), the cell is in standby mode. Setting the word line to a logic high enables the access nMOS transistors. This connects the cell with both bit lines and allows the cells to be read or written. The SRAM cell is read by asserting a WL and detecting the voltage difference at the bit lines BL and $\overline{\text{BL}}$. The SRAM cell is written by setting the content on the bit lines BL and $\overline{\text{BL}}$ and asserting the word line.

Due to the ability to store the information indefinitely and the high speed of SRAM cells, they are used to implement caches and registers in microprocessors. Furthermore, the main advantage of SRAM is that it uses the same fabrication process as the microprocessor core, simplifying the integration of cache and CPU registers onto the processor die. On the other hand, the main **disadvantages of SRAM cells are price, low density, and high operational power consumption**. These disadvantages prevent the usage of SDRAM cells in the main computer memory.

Since SRAM cells are not used to build the main memory, we will end up dealing with and learning about SRAM cells at this point, and we are now going to deep dive into DRAM cells. By contrast, DRAM typically uses a different process that is not optimal for logic circuits, making the integration of CPU logic and DRAM harder than the integration of CPU logic and SRAM. But DRAMs are smaller, cheaper, and consume less power, which makes them the better candidate for implementing the main memory.

Summary: SRAM cell

A *SRAM cell* uses a bistable element to store one bit of information. It is made up of a bistable and two access nMOS transistors that serve as a switch used to control the state of the bistable element during the read and write operations.

Due to the ability to store the information indefinitely and the high speed of SRAM cells, they are used to implement caches and registers in microprocessors.

1.4 DRAM cell

Dynamic Random Access Memory (DRAM) is the main memory used for all computers. To pack more bits per chip, a DRAM cell consists only of a single MOS transistor (T) and a storage capacitor (C) as shown in Figure 1.7. The data in the

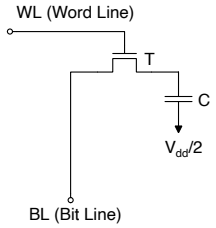


Fig. 1.7: A DRAM cell.

cell can be read or written through the bit line (BL) terminal. In contrast to SRAMs, DRAMs store their contents as a charge on a capacitor C. This way, the DRAM cell is substantially smaller than the SRAM cell. The transistor T acts as a switch between the storage capacitor and the bit line. The word line (WL) terminal is used to switch on/off the transistor T. Reading the bit from the DRAM cell discharges the capacitor and thus destroys the information. Even if we do not read the DRAM cell, the charge leaks from the capacitor because the cell transistor does not entirely disconnect the storage capacitor from the bit line. Even though the transistor is switched off, a tiny current flows from the capacitor to the bit line and discharges the capacitor. Therefore, the charge (information) must be refreshed several times each second. Hence the name *dynamic*.

1.4.1 Basic operation of DRAM

The transistor T acts as a switch between the storage capacitor C and the bit line BL. One node of the capacitor is connected to $V_{dd}/2$. The voltage across the capacitor is either $+V_{dd}/2$, if the capacitor stores "1", or $-V_{dd}/2$, if the capacitor stores "0". The charge stored in a capacitor is equal to capacitance times voltage across the capacitor:

$$Q = C \times V_{dd}/2 \; . \tag{1.1}$$

In a 90 nm DRAM process technology, the capacitance of a DRAM storage cell is 30 fF. If we assume $V_{dd} = 3.3V$, then

$$Q = 30fF \times 3.3V/2 = 34.5fC \; .$$

As you may recall from physics class, one electron equals to a charge of $1.6 \cdot 10^{-19}C$, thus the storage capacitor stores only 210000 electrons! Even though the transistor has a very high resistance when switched-off, the charge on the capacitor leaks away through switched off transistor in tens to hundreds of milliseconds. Storage cells should be regularly refreshed to avoid loss of data.

The data is written into a memory cell by placing the "1" or "0" charge into the storage capacitor. To write data into a cell, we first set the bit line to Vdd ("1") or to GND ("0") and assert the word line to connect the capacitor to the bit line. The storage capacitor then retains the stored charge after the word line is de-asserted, and the transistor is turned off. The electric charge on the storage capacitor slowly leaks off, so without intervention, the data on the chip would soon be lost. This capacitor will be accessed for either a new write, a read, or a refresh.

To read data from the cell, the bit line is first precharged to $V_{dd}/2$. The word line is then driven high to connect a cell's storage capacitor to its bit line. This causes the transistor to conduct, transferring charge from the storage cell to the connected bit line (if the stored value is "1") or from the connected bit-line to the storage cell (if the stored value is "0"). This process is depicted in Figure 1.14. In both cases, information stored in the DRAM cell is lost. Thus, reading from

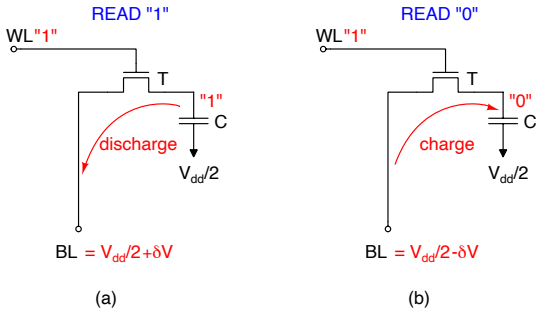


Fig. 1.8: Reading from a DRAM cell. (a) Reading "1" from a DRAM cell discharges the storage capacitor and slightly increases the voltage of the bit line. (b) Reading "0" from a DRAM cell charges the storage capacitor and slightly decreases the voltage of the bit line. In both cases, information is lost.

DRAM is a **destructive operation**. The bit lines are relatively long because they

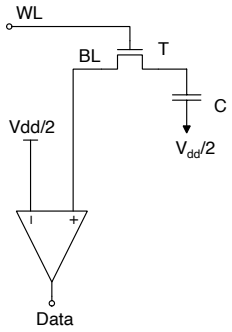


Fig. 1.9: A DRAM cell with a sense amplifier.

connect storage cells in all memory words, and they act as a capacitor with relatively high capacitance (the capacitance of the bit lines is ten times the capacitance of the storage capacitor). According to the charge-sharing equation (capacitive voltage divider), the voltage swing (the magnitude of a voltage difference) δV on the bit line during readout is

$$\delta V = \frac{V_{dd}}{2} \frac{C}{C + C_{BL}}, \tag{1.2}$$

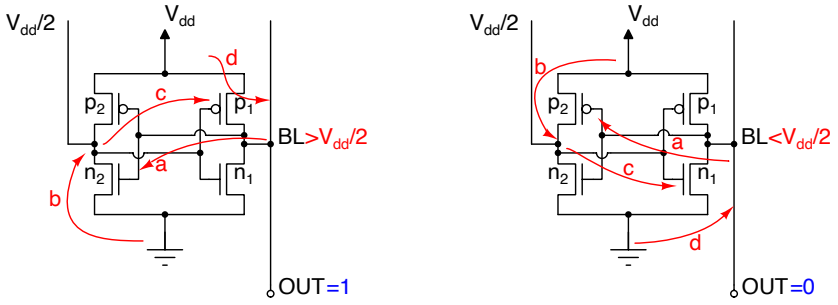
where C is the capacitance of the storage capacitor and C_{BL} is the capacitance of the bit line. If the capacitance of the bit line is ten times the capacitance of the storage capacitor and $V_{dd} = 3.3V$, the voltage difference δV on the bit line during the read operation is only 150 mV! When dealing with such tiny voltage swing,

correctly detecting the bit value is quite a challenge. Thus, we need a special circuit to sense this small voltage swing. Sensing is necessary to read the cell data properly. A special circuit used to detect the voltage swing and read the data is a **sense amplifier**.

To sense the voltage swing on the bit line, a sense amplifier is used, as presented in Fig 1.9. A sense amplifier has two inputs. One input is connected to the bit line, and the other input is tied to $V_{dd}/2$. The sense amplifier detects the voltage difference at its inputs and outputs 0 at the Data terminal if the voltage on the bit line is less than $V_{dd}/2$, or 1 otherwise.

1.4.2 Basic operation of sense amplifiers

A sense amplifier is a simple circuit made up of two cross-coupled CMOS inverters - so it is a SRAM cell. Figure 1.10 shows a sense amplifier built from cross-coupled CMOS inverters. Initially, the bit line (BL) is precharged to $V_{dd}/2$. During a read, the bit line changes its voltage by a small amount, δV. If the voltage of the bit line is higher than $V_{dd}/2$ (Figure 1.10a), the `n2` nMOS transistor begins to conduct and pulls the precharged line down to "0". This, in turn, causes the `p1` pMOS transistor to conduct. After a small delay, BL is pulled high, and OUT=1. On the other hand, if the voltage of the bit line is lower than $V_{dd}/2$ (Figure 1.10b), the (p2) pMOS transistor begins to conduct and pulls the precharged line up to "1". This, in turn, causes the `n1` nMOS transistor to conduct. After a small delay, BL is pulled down to "0", and OUT=0. The feedback that occurs from the cross-connected inverters



(a) Sensing "1". (b) Sensing "0".

Fig. 1.10: A simplified structure and operation of a sense amplifier. (a) Sensing "1". (b) Sensing "0".

thereby amplifies the small voltage difference between the BL and the precharged input reference until the bit line is entirely at the lowes or the highest voltage.

We have just learned that the main function of sense amplifiers is to sense the tiny voltage swing on the bit lines that occurs when an access transistor is turned on and a storage capacitor places its charge on the bit line. The second function of sense amplifiers is to restore the value of cells after the voltage on the bit lines is sensed. Recall that turning on the access transistor allows a storage capacitor to share its stored charge with the bit line. However, the process of sharing the charge from a storage cell discharges that storage cell. Thus, the information in the cell is lost and cannot be read again. But this information is stored in the sense amplifier, as the sense amplifier is a bistable circuit made up of two cross-coupled inverters. As such, it can store information as long the supply voltage is present. Consequently, after sensing, the sense amplifier is used to write back the the bit value to the storage cell. This operation is referred to as **(row) precharge**.

Summary: DRAM cell

Dynamic Random Access Memory (DRAM) is the main memory used for all computers. DRAMs store their contents as a charge on a capacitor. A DRAM cell consists only of a storage capacitor and a single nMOS transistor that acts as a switch between the storage capacitor and the bit line.

Reading from a DRAM cell is a *destructive operation*. Besides, the charge on the capacitor leaks away through switched off transistor in tens to hundreds of milliseconds. Thus DRAMs should be regularly *refreshed*.

A sense amplifier is a special circuit used to detect the tiny voltage swing on the bit line and read the data. The sense amplifier is also used to write back the bit value to the storage cell. This operation is referred to as *precharge*.

1.5 DRAM Arrays and DRAM Banks

DRAM is usually arranged in a rectangular **memory array** of storage cells organized into rows and columns. Figure 1.11 shows a simplified basic structure of a DRAM cell array containing R-by-C cells. **DRAM arrays usually contain many hundreds or thousands of cells in height and width**. The cells of a DRAM are accessed by a **row address** and a **column address**. The rows address lines (i.e., the word lines) are connected to the gates of the nMOS transistors, and the column lines are connected to the sense amplifiers.

The array size represents a trade-off between density and performance. Larger arrays contain more bits of information, but they also require longer word lines and bit lines. Longer word and bit lines have a higher capacitance. An array that contains thousands of cells in height and width has an order of magnitude higher capacitance on the bit line than in the cell, so the bit line voltage swing δV during a read is tiny,

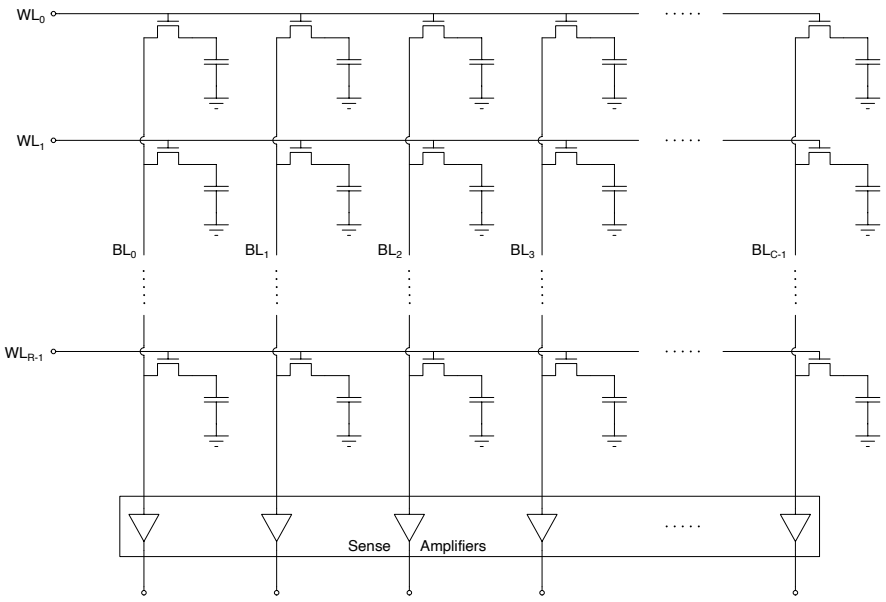


Fig. 1.11: A simplified structure of a DRAM array.

which is hard to detect. Besides, due to a higher capacitance, larger arrays are slow. A typical array size in a recent DRAM is 32K words (rows) by 1024 bits (columns).

A DRAM memory chip can have 4-16 DRAM arrays that are accessed simultaneously, and transmits or receives a number of bits equal to the number of arrays each time the memory controller accesses the DRAM. Each array provides a single bit to the output pin. DRAM chips are described as xN, where N refers to the number of memory arrays and output pins. For example, in a simple organization, a x8 DRAM (pronounced "by eight") indicates that the DRAM has at least eight memory arrays and that a column width is 8 bits (each column read or write access transmits 8 bits of data). This means that the DRAM transmits or receives eight bits each time the memory controller accesses the DRAM. A set of memory arrays accessed simultaneously is referred to as a **bank**.

Summary: DRAM Arrays and DRAM Banks

DRAM is arranged in a rectangular *memory array* of storage cells organized into rows and columns.

The cells of a DRAM are accessed by a *row address* and a *column address*.

A **bank** is a set of N memory arrays accessed simultaneously, forming an N-bit width column. Usually, there are 4, 8, or 16 DRAM arrays in a bank.

1.6 DRAM Chips

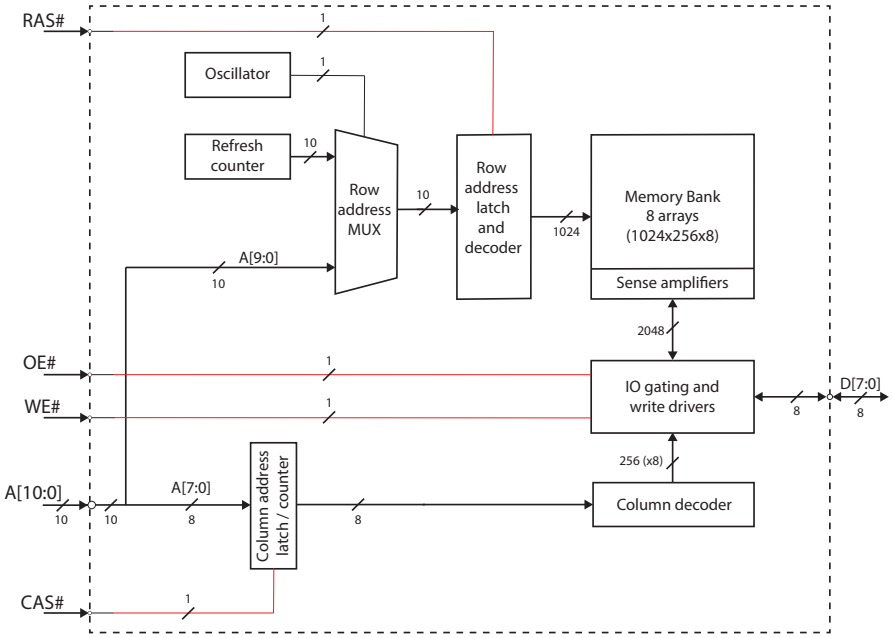


Fig. 1.12: Simplified structure of a 256K × 8-bit DRAM chip.

Figure 1.12 presents the basic structure of a DRAM chip. As we have learned, the DRAM memory is organized as a rectangular matrix of rows and columns. The DRAM chip in Figure 1.12 contains a bank of 8 arrays. Each array has 1024-by-256 storage cells. All arrays in a bank are accessed at the same time, so the DRAM chip in Figure 1.12 reads or transmits eight bits in a single access (D0 to D7). The components identifying the row and column are referred to as the **row address**

decoder and the **column selector**. The row address decoder is used to activate the appropriate word line from the given row address. The column selector is used to select the appropriate column from the given column address.

As the capacity of DRAMs is large, the DRAM chips would require a large number of address lines to address a row and a column. For example, to address a cell in a 32256-by-1024 array, we need 15 bits to select a word and 10 bits to select a column. Such a large number of address bits could be an issue. The solution is to **multiplex the address lines**. Firstly, the row address is applied to the address lines, then the column address follows. In such a way, the number of address pins is cut almost in half. The same holds for DRAM in Figure 1.12. Instead of having 18 address bits (10 for the row and 8 for the column), only 10 address bits are used. To indicate which of two addresses is currently on the bus, we need **two additional control signals**: the **row access strobe (RAS)** and the **column access strobe (CAS)**. When the RAS signal is activated, the address bits A0 to A9 are latched into the **row address latch**. Similarly, when the CAS signal is activated, the address bits A0 to A7 are latched into the **column address latch**.

Two more control signals are required to appropriate transfer data into and from a DRAM chip. The **write enable (WE)** signal is used to choose a read operation or a write operation. A low voltage level signifies that a write operation is desired; a high voltage level is used to choose a read operation. During a read operation, the **output enable (OE)** signal is used to prevent data from appearing at the output until needed. When OE is low, data appears at the data outputs as soon as it is available. OE is kept high during a write operation. Figure 1.13 illustrates a pinout diagram of a 256K $\times$ 8-bit DRAM from Figure 1.12.

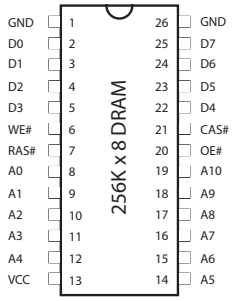


Fig. 1.13: 256K $\times$ 8-bit DRAM chip pinout.

Summary: DRAM Chips

DRAM chips contain at least one memory bank. The *row address decoder* is used to activate the appropriate word line from the given row address. The *column selector* is used to select the proper column from the given column address.

As the number of address bits required to select rows and columns can be quite large, the *address lines are multiplexed*. To indicate which of two addresses is currently on the bus, we need two additional control signals: the *row access strobe (RAS)* and the *column access strobe (CAS)*.

The *write enable (WE)* signal is used to choose a read or a write operation. During a read operation, the *output enable (OE)* signal is used to prevent data from appearing at the output until needed.

FALLACY: Memories (DRAMs) are physically organized as a liner vector of memory words.

It is a common and erroneous belief that memory is physically organized as a vector of memory words (and not as a rectangular array of rows and columns). Such an organization of memory would otherwise be ideal. A memory array would be just one long vector of memory cells, and there would be only one memory cell in a word. All memory cells would then be connected to the same bit-line. In that case, a DRAM array would contain R-by-1 memory cells. The memory with 8-bit words would then be composed of eight parallel R-by-1 memory arrays. In this case, the row address would already be the column address, because there would be only one column in a row. The memory addresses would not be multiplexed, and we would not need the RAS and CAS signals. Wouldn't that be great? However, it is physically impossible to make such memory because, in such memory, the bit lines would be extremely long and would have huge capacitance. The capacitance of such long bit lines would probably be several thousand times greater than the capacitance of the memory cells, and it would be impossible to detect a tiny voltage swing.

1.7 Basic DRAM operations and timings

The most challenging aspect when working with DRAMs is resolving the timing requirements. **DRAMs are generally asynchronous, responding to input signals whenever they occur**. As long as the signals are applied in the proper sequence,

with signal durations and delays between signals that meet the specified limits, the DRAM works properly. The following signals control the DRAM operations:

1. Row Address Strobe (RAS). RAS is active low. To enable RAS, a transition from a high voltage to a low voltage, is required. The voltage must remain low until RAS is no longer needed. During a complete memory cycle, there is a minimum amount of time that RAS must be active (t_{RAS}). There is a minimum amount of time that RAS must be inactive before activating it again, called the RAS precharge time (t_{RP}). t_{RP} tells us how fast the row can be precharged before we can engage another RAS.
2. Column Address Strobe (CAS). CAS is used to latch the column address and to initiate the read or write operation. It is active low. The memory specification lists the minimum amount of time CAS must remain active (t_{CAS}). For most memory operations, there is also a minimum amount of time that CAS must be inactive before activating it again, called the CAS precharge time (t_{CP}).
3. Write Enable (WE). The write enable signal is used to choose a read operation or a write operation. It is active low.
4. Output Enable (OE). It is active low. When OE is low during a read operation, data appears at the data outputs as soon as it is available. During a write operation, OE should be high.
5. Address. The addresses are used to select a memory location on the chip. The address pins on a memory device are used for both row and column selection (multiplexing).
6. Data In or Out. The data pins on the DRAM memory device are used for data input and output. During a write operation, data at data pins are stored in the selected memory cells. During a read operation, data from the selected memory cells appear at the data once access is complete, and OE is low.

1.7.1 Reading data from DRAM memory

To read the data from a DRAM memory cell, we must select the DRAM memory cell by applying its row and column addresses to the address input pins. The charge on the selected DRAM cell must then be sensed by the sense amplifier and sent to the data output (pins). In terms of timing, the following steps must occur:

1. The row address must be applied to the address input pins on the memory device before RAS goes low.
2. RAS must go from high to low and remain low for the prescribed amount of time (t_{RAS}). When RAS goes low, the **memory row** addressed by the row address **is open**, and the charge from the cells in the selected row starts to flow to the bit lines.

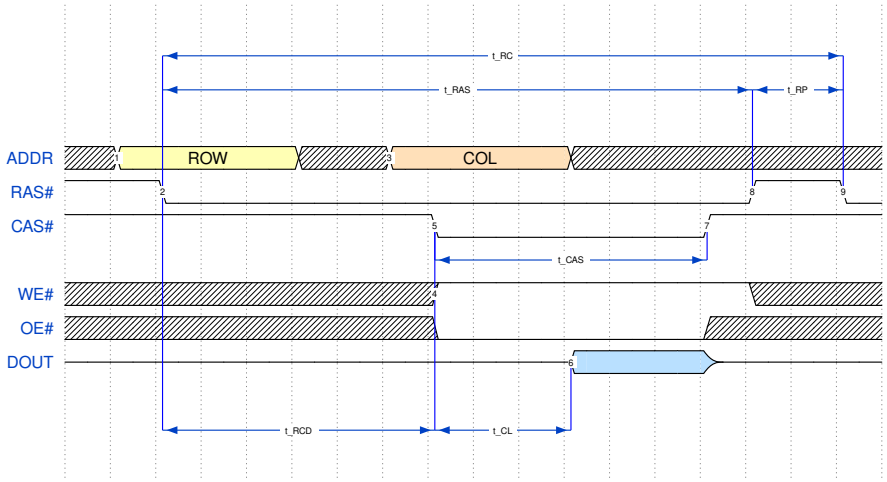


Fig. 1.14: Simplified DRAM read cycle.

3. The column address must be applied to the address input pins on the memory device before CAS goes low.
4. WE must be set high for a read operation to occur before the transition of CAS, and remain high after the transition of CAS.
5. Only after the prescribed amount of time (t_{RCD}), CAS must go from high to low and remain low for the prescribed amount of time (t_{CAS}). RAS-to-CAS delay (t_{RCD}) time ensures that the charge from the selected cells is on the bit lines and properly sensed by the sense amplifiers.
6. Data appears at the data output pins of the memory device. The time at which the data appears is called CAS latency (t_{CL}).
7. Before the read cycle can be considered complete, CAS and RAS must return to their inactive states. A new read or write access can start only after the prescribed amount of time (t_{RP}- Row Precharge).

The read access lasts for a **row cycle time** (t_{RC}):

$$t_{RC} = t_{RAS} + t_{RP} \ . \tag{1.3}$$

The row cycle time, t_{RC}, determines the minimum time a memory row takes to complete a full cycle, from row activation up to the precharging of the active row. This is an interval between accesses to different rows in a given set of DRAM arrays.

1.7.2 Writing data to DRAM memory

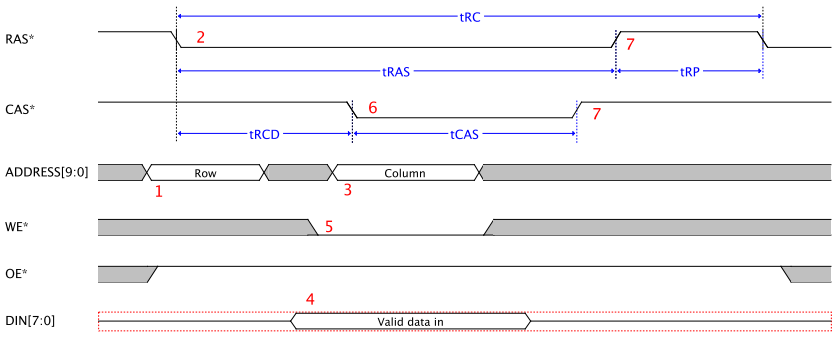


Fig. 1.15: Simplified DRAM write cycle.

To write to a DRAM memory cell, the row and column address for the DRAM cell must be selected, and data must be presented at the data input pins. The sense amplifier either charge the memory cell's capacitor or discharges it, depending on whether a 1 or 0 is to be stored. In terms of timing, the following steps must occur:

1. The row address must be applied to the address input pins on the memory device before RAS goes low.
2. RAS must go from high to low and remain low for the prescribed amount of time (t_{RAS}). When RAS goes low, the **memory row** addressed by the row address is open.
3. Data must be applied to the data input pins before CAS goes low.
4. The column address must be applied to the address input pins on the memory device after RAS goes low and before CAS goes low.
5. WE must be set low for a write operation to occur.
6. Only after the prescribed amount of time (t_{RCD}), CAS must switch from high to low and remain low for a prescribed amount of time (t_{CAS}).
7. Before the write cycle can be considered complete, CAS and RAS must return to their inactive states. A new read or write access can start only after the prescribed amount of time (t_{RP}).

The write access also lasts for a **row cycle time** (t_{RC}).

1.7.3 Refreshing the DRAM memory

Since DRAM memory cells are capacitors, the charge they contain can leak away over time. If the charge is lost, the data is lost! To prevent the loss of data, DRAMs must be refreshed, i.e., the charge on the individual memory cells must be restored. **DRAMs are refreshed one row at a time**. The frequency of refresh depends on the silicon technology used to manufacture the memory chip and the design of the memory cells. **Most of today's DRAMs require a refresh to occur every 64 ms**.

Reading or writing a memory cell has the effect of refreshing the selected cell because after read/write the entire row is precharged. Unfortunately, not all cells are read or written within 64 ms time frame. Hence, each row in the array must be accessed and restored during the refresh interval. The refresh cycles are distributed across the entire refresh interval of 64 ms in such a way that all rows are refreshed within the required interval. If, for example, a DRAM array has 4096 rows, every 15.6 microseconds a new row must be refreshed. At the end of the 64 ms interval, the process begins again.

DRAMs use an **internal oscillator** to determine the refresh frequency and a **counter** to keep track of which row is to be refreshed, and initiate the refresh periodically. Such an auto-initiated refresh is referred to as **self refresh**. To refresh one row of the memory array, the so-called **CAS-before-RAS refresh** is used. The following steps form the CAS-before-RAS refresh:

1. CAS must switch from high to low, while the WE signal remains in a high state (equivalent to read).
2. After the prescribed delay, RAS must switch from high to low.
3. The internal counter determines which row is to be refreshed and applies the row address at the address pins
4. After the required delay, CAS returns to a high level.
5. After the necessary delay, RAS returns to a high level.

Summary: DRAM Operations and Timings

DRAMs are asynchronous systems, responding to input signals whenever they occur. The DRAM will work properly, as long as the input signals are applied in the proper sequence, with signal durations and delays between signals that meet the specified limits.

Typical operations in DRAMs are: read, write, and refresh. All these operations are initiated and controlled by the prescribed sequence of input signals.

The read and write accesses last for a *row cycle time* (t_{RC}):

$$t_{RC} = t_{RAS} + t_{RP} \ .$$

DRAMs must be refreshed in order to prevent the loss of data. DRAMs are refreshed one row at a time. DRAMs use an *internal oscillator* to determine the refresh frequency and initiate a refresh and a *counter* to keep track of row to be refreshed. Such an auto-initiated refresh is referred to as *self refresh*. Self-refresh uses the so-called CAS-before-RAS sequence.

Summary: Important timings in DRAMs.

Name	Symbol	Description
Row Active Time	t_{RAS}	The minimum amount of time RAS is required to be active (low) to read or write to a memory location.
CAS latency	t_{CL}	This is the time interval it takes to read the first bit of memory from a DRAM with the correct row already open.
Row Address to Column Address Delay	t_{RCD}	The minimum time required between activating RAS and activating CAS . It is the time interval between row access and data ready at sense amplifiers.
Random Access Time	t_{RAC}	This is the time required to read any random memory cell. It is the time to read the first bit of memory from an DRAM without an active row. t_{RAC}= t_{RCD}+ t_{CL}.
Row Precharge Time	t_{RP}	After a successful data retrieval from the memory, the row that was used to access the data needs to be closed. This is the minimum amount of time that RAS must be inactive.
Row Cycle Time	t_{RC}	This is the time associated with single rad or write cycle. t_{RC}= t_{RAS}+ t_{RP}

1.8 Improving the performance of a DRAM chip

As mentioned earlier, one DRAM access is divided into row access and column access. Let's first look at how we read two consecutive columns from the same row

in classic DRAMs. A timing diagram for reading two consecutive columns, A and B, in the same row X is shown in the Figure 1.16. Although both columns are in the same row X, we have to repeat the entire reading cycle from Figure 1.14 to read each column. Wouldn't it be better to keep the entire row 'open' once the amplifiers

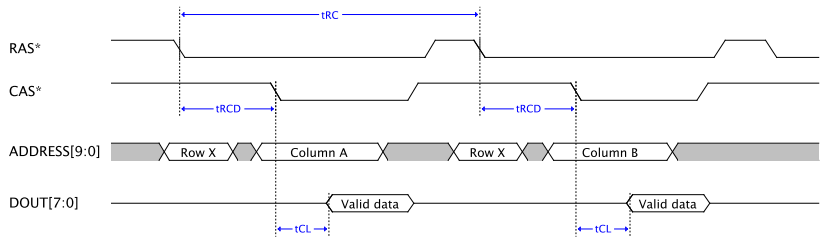


Fig. 1.16: Simplified read timing for two columns in the same row for conventional DRAM.

sensed all bits in that row? Actually, the sense amplifiers can act like a row buffer to keep the row data. That way, we don't have to access the row every time and then close it after reading each column. Exactly this solution was used for one of the first performance enhancements in DRAM memories. But wait, how often do we access two or more consecutive columns from the same row? Very often, indeed, due to **temporal and spatial locality**. All methods used to improve the performance of a DRAM chip and to decrease the access time rely on the ability to access all of the data stored in a row without having to initiate a completely new memory cycle.

1.8.1 Fast Page Mode DRAM

Fast Page Mode DRAM is a minor modification to the first-generation DRAMs that allows faster access to data in the same row. The performance of read and write accesses to a row was improved by avoiding the inefficiency of opening and precharging the same row repeatedly to access different columns in the same row. Fast Page Mode DRAM eliminates the need for a row address if data is located in the row previously accessed. In the Fast Page Mode DRAM, after a row has been opened by holding RAS low, the row bits are kept by the sense amplifiers, and multiple reads or writes could be performed to any of the columns in the open row. Each column access is initiated by asserting CAS and presenting a column address.

To read data using Fast Page Mode, we start a regular read operation by addressing the row (same steps 1 through 6 as in Figure 1.14). Once the row data is valid, we switch CAS high but leave RAS low. There is a minimum amount of time that CAS must be inactive, called the CAS precharge time (t_{CP}). When CAS has been inactive (high) for the required amount of time (t_{CP}), we repeat steps 3 through 6

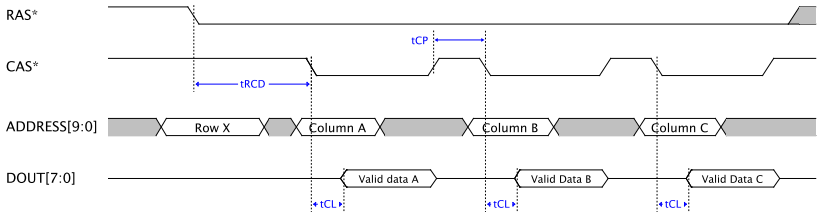


Fig. 1.17: Simplified read timing for two columns in the same row for conventional DRAM.

of the read operation from Figure 1.14. We can continue in this way until a new row address is required or the chip needs to be refreshed. Figure 1.17 is a simplified timing diagram that illustrates a Fast Page Mode read cycle.

Let's use an example to illustrate how fast page mode impacts the system's performance. In this example, we compare two scenarios: 4 memory accesses in the same row without fast page mode, and 4 memory accesses in the same row with fast page mode. We are assuming that t_{RC} is 70 ns, t_{RCD} is 20 ns, and t_{CL} is 15 ns. In the first scenario, the data from the fourth column will be available after $3 \cdot t_{RC} + t_{RCD} + t_{CL} = 245ns$. In the second scenario, we are also assuming that CAS should remain high for 5 ns before going down again (t_{CP} is 5 ns), and that data is kept valid for 20 ns. Now, the data from the fourth column will be available after $t_{RCD} + 3 \cdot (t_{CL} + 20 + t_{CP}) = 140ns$.

1.8.2 Extended Data Output DRAM

The second change to improve the performance is Extended Data Out (EDO) DRAM. EDO is very similar to FPM. The primary advantage of EDO DRAMs over FPM DRAMs is that the data outputs are not disabled when CAS goes high on the EDO DRAM, allowing the data from the current read cycle to be present at the outputs while the next read cycle begins, i.e., data is still present on the output pins, while CAS is changing and a new column address is latched. This allows a certain amount of overlap in operation (pipelining), resulting in faster access (cycle) time. Figure 1.18 is a complete timing diagram that illustrates an EDO mode read cycle. Let's now illustrate how EDO impacts the system's performance using the same example as before, i.e., 4 memory accesses in the same row with EDO. Assuming, that we keep the data valid for 20 ns, the data from the fourth column becomes available after $t_{RCD} + t_{CL} + 3 \cdot 20 = 95ns$.

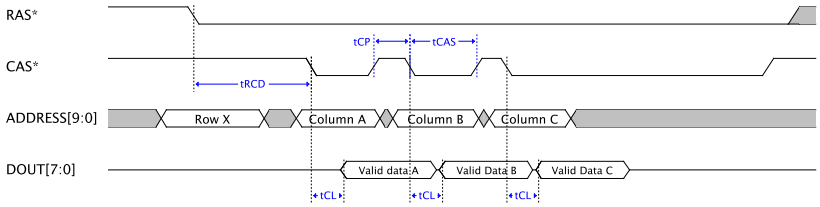


Fig. 1.18: Simplified read timing for two columns in the same row for conventional DRAM.

Summary: FPM and EDO DRAMs

Due to *temporal and spatial locality*, we often access two or more consecutive columns from the same row.

All methods used to improve the performance of a DRAM chip and to decrease the access time rely on the ability to access all of the data stored in a row without having to initiate a completely new memory cycle.

Fast Page Mode DRAM eliminates the need for a row address if data is located in the row previously accessed.

In *EDO DRAMs*, data is still present on the output pins, while CAS is changing, and a new column address is latched. This allows a certain amount of overlap in operation (pipelining), resulting in faster access time.

1.9 Synchronous DRAM

Originally, DRAMs that we have just covered and were produced from the early 1970s to early 1990s had an asynchronous interface, in which input control signals have a direct effect on internal functions. The **synchronous DRAM (SDRAM)** device represents a significant improvement over the DRAM devices. In particular, SDRAM devices differ from previous generations of DRAM devices in two significant ways:

1. the **clock signal** was added to the SDRAM device; hence the SDRAM device has a synchronous device interface, where commands instead of signals are used to control internal latches, and
2. SDRAM devices contain **multiple independent banks**.

Besides, SDRAMs typically also have a programmable **mode register** to hold the number of bytes requested, and hence can send many bytes over several cycles per request without sending any new addresses. This type of transfer is referred to as **burst mode**.

SDRAMs have the clock signal and all internal actions occur on its negative edge. As we have seen, in DRAM devices, the RAS, CAS, and WE signals from the memory controller directly control internal latches and input/output buffers, and these signals can arrive at the DRAM device's pins at any time. The DRAM devices then respond to the RAS, CAS, and WE signals as soon as possible. Contrary, in SDRAM devices, the RAS, CAS, and WE signals do not directly control internal latches and buffers. In SDRAM devices these signals form a command bus used to transmit **commands** to the internal state machine, which **executes the commands at the falling edge of the clock signal**. In this way, the control of internal latches and input/output buffers moved from the external memory controller into the state machine in the SDRAM device's control logic. The RAS, CAS and WE names were retained for signals on the command bus that transmits commands, although these specific signals no longer control latches and buffers that are internal to the SDRAM device.

The second feature that significantly differentiates the SDRAM device from the DRAM devices is that the SDRAM devices contain multiple banks. The presence of multiple, independent banks in each SDRAM device means that while one bank is busy with a row activation command or a precharge command, the memory controller can send a new command to a different bank. Multiple banks now enable the interleaving of memory requests to different banks in a single SDRAM device. SDRAM devices contain either 2, 4, or 8 independent banks. One to three bank address inputs (BA0, BA1, and BA2) determine which bank the command refers to.

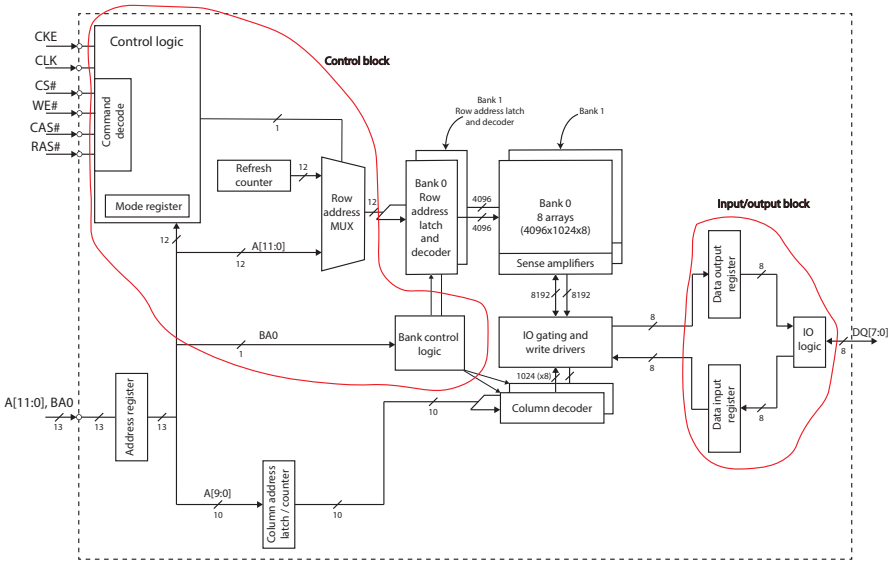


Fig. 1.19: Simplified block diagram of a SDRAM device with two banks.

1.9.1 Functional description

Figure 1.19 shows the simplified block diagram of an SDRAM device with two independent banks. The hash (#) beside a signal name denotes that the signal is active low. Each bank has its row address latch and decoder, its column decoder, and its sense amplifiers. Each bank in the SDRAM device in Figure 1.19 consists of eight DRAM arrays of size 4096-by-1024 bits. The address now consists of bank number (BA0), row address (A[11:0]), and a column address (A[9:0]).

In an SDRAM device, commands are decoded on the rising edge of the clock signal (CLK) and executed on the falling edge of CLK if the chip-select signal (CS) is active. The command is asserted on the command bus by the external memory controller . The command bus consists of WE, CAS, and RAS signals. All these signals are active low. Although the signal lines retain the function-specific names from DRAMs, they only form a command bus. Table 1.1 shows the command set of the SDRAM device and the input signal combinations on the command bus that designate the commands. The table also shows that as long as CS is not active, the SDRAM device ignores the signals on the command bus.

The control block in Figure 1.19 consists of control logic, a multiplexor to select a row address, a refresh counter and bank control logic. The refresh counter keeps track of the row to be refreshed. The multiplexor is used to select a row address to be transferred into the row address latch and decoder. The address is either an address coming from the refresh counter (in case the control logic performs a refresh cycle) or an address from the external address bus coming from the DRAM

Table 1.1: SDRAM commands.

Command	CS#	RAS#	CAS#	WE#	Address
COMMAND INHIBIT	H	X	X	X	X
NO OPERATION (NOP)	L	H	H	H	X
ACTIVE (select bank and activate row)	L	L	H	H	Bank/row
READ (select bank and column, and start READ burst)	L	H	L	H	Bank/col
WRITE (select bank and column, and start WRITE burst)	L	H	L	L	Bank/col
PRECHARGE (deactivate row in bank)	L	L	H	L	Bank/row
AUTO REFRESH	L	L	L	H	X
LOAD MODE REGISTER	L	L	L	L	Code

controller. Control logic contains a command decoder, a finite state machine that executes commands, and the mode register. The mode register is a programmable 10-bit register whose individual bits determine:

- CAS latency (CL). CL is t_{CL} rounded-up to the nearest number of clock cycles,
- the length of the burst transfer,
- and the order of memory words in the burst transfer.

The control logic receives a command from the command bus. Then, depending on the type of command and values contained in the respective fields of the mode register, the control logic performs specific sequences of operations to execute the command. These operations are performed by the internal state machine on successive clock cycles without requiring clock-by-clock control from the memory controller. Figure 1.20 illustrates a simplified state diagram of the internal state machine. After the initialization of the mode register, the internal state machine is in the Idle state with all banks and rows precharged. If no command is issued to SDRAM, the SDRAM chip will regularly perform the self-refresh. The internal counter drives the self-refresh operation. To start memory access, the memory controller should first issue the ACTIVE command. This will eventually open a row/bank, and the internal state machine waits in the Active state for additional commands. To read data, the memory controller should issue the READ command, and to write data into memory, the memory controller should issue the WRITE command. Then, the internal state machine enters the Read or Write state, and uses the column address and generates the appropriate internal signals to access the column. The READ or WRITE commands can be followed by any number of READ or WRITE commands or the PRECHARGE command can be issued to restore the data and close the open bank/row. After the precharge operation has been executed, the internal state machine will wait in the IDLE state.

For example, in the case of the ACTIVE command, the state machine passes the row address to the row address latch and decoder through the multiplexor. The address bit BA0 determines the bank, which will be accessed. The bank control block, which acts as a decoder, selects the appropriate row address latch and decoder, and the appropriate column decoder based on the BA0 bit. The selected row is then

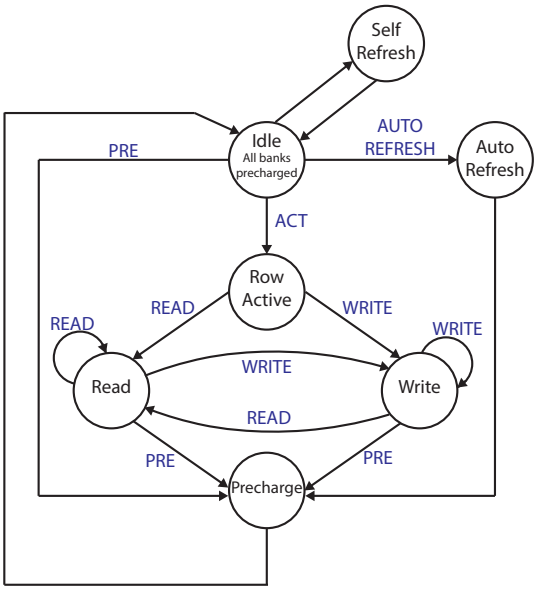


Fig. 1.20: Simplified state diagram of the internal state machine.

opened and its content is transferred into the sense amplifiers. In the case the memory controller asserts a READ command, the internal state machine drives the bank control logic, which selects the appropriate column decoder, based on the BA0 bit. The column decoder then selects the word from the sense amplifiers of the chosen bank. Each bank has its own column decoder - this feature is especially useful when interleaving transfers from two (or more) active banks. The SDRAM device in Figure 1.19 provides for two rows of the DRAM to be opened simultaneously. Memory accesses between two opened banks can be interleaved to hide RAS-to-CAS delay and row precharge time. When an address is firstly sent that designates a new bank, the row in that bank must be opened. But when subsequent access specifies the same row in an already open bank, the access can happen quickly, sending only the column address. This feature requires that each bank has its own row address latch, sense amplifiers and a column decoder. For example, while one row is accessed, the memory controller can send an ACTIVE command to a different bank and, in such a way, transfer a new row into the sense amplifiers. This row can than be read or written to without waiting for t_{RCD}. Later, we will learn how data is transferred to/from SDRAM chip and how the burst transfers and bank interleaving can speed up memory transactions.

Summary: SDRAMs

SDRAM devices have a synchronous device interface, where commands, instead of signals, are used to control internal latches.

In SDRAM devices, signals CAS, RAS, WE and CS form a *command bus* used to transmit *commands* to the *internal state machine*.

SDRAM devices contain multiple independent banks.

SDRAMs can transfer many columns over several cycles per request without sending any new addresses. This type of transfer is referred to as *burst mode*.

1.9.2 Basic operations and timings

Now that we are familiar with the basic functionality of SDRAMs, we are going to present four basic operations in SDRAMs: ACTIVE, READ, WRITE, and PRECHARGE.

1.9.2.1 Activate (open) row

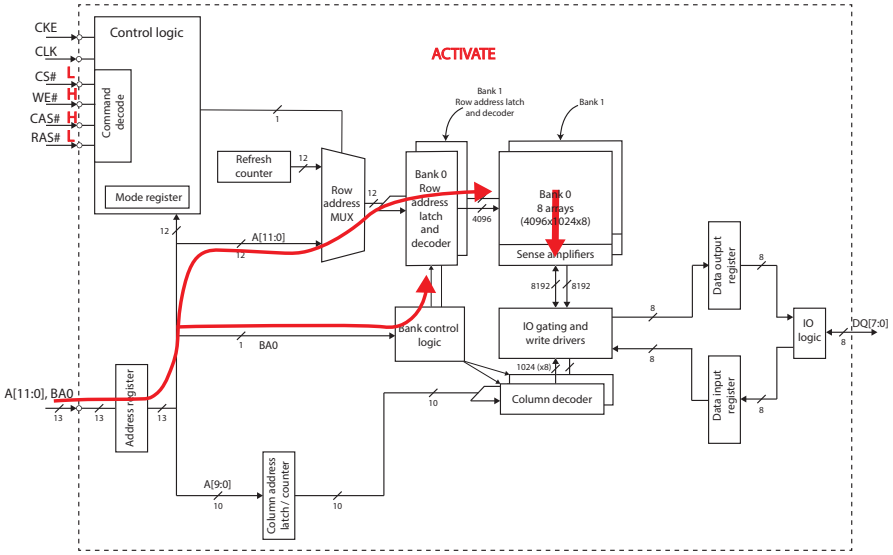


Fig. 1.21: The progression of the ACTIVE command.

Before any READ or WRITE commands can be issued to a bank within the SDRAM, a row in that bank must be opened. This is accomplished via the ACTIVE command. The purpose of the ACTIVE command is to open (activate) a row in a selected bank and move data from the DRAM arrays to the sense amplifiers of the open bank. Figure 1.21 illustrates the progression of the ACTIVE command. The address A11-A0 from the address bus is stored into the row address latch and decoder of the selected bank. The address bit BA0 selects the bank and its row address latch and decoder. Then, the entire row of data is read into the sense amplifiers. Similarly to DRAMs, two timings are associated with the ACTIVE command: *Row Address to Column Address Delay* (t_{RCD}) and *Row Active Time* t_{RAS}. t_{RCD} is the time it takes for the ACTIVE command to move data from the DRAM cell arrays to the sense amplifiers that hold the entire row of data. After t_{RCD}, a column read or write access commands can be issued to move data between the sense amplifiers and the memory controller through the input/output block and data bus (Figure 1.22). Row

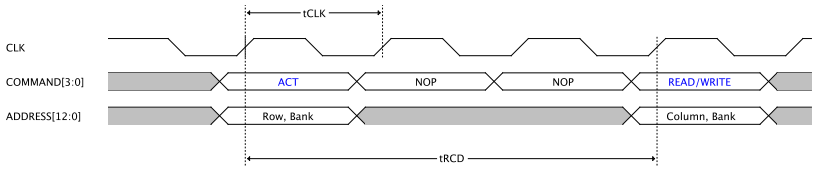


Fig. 1.22: Meeting t_{RCD}.

address to column address delay, t_{RCD}, should be divided by the clock period and rounded up to the nearest whole number to determine the earliest clock edge after the ACTIVE command on which a READ or WRITE command can be issued. For example, a t_{RCD} of 20ns with a 125 MHz clock (8ns period) results in 2.5 clock periods, rounded to 3. A subsequent ACTIVE command to a different row in the same bank can only be issued after the previous active row has been precharged.

Row active time, t_{RAS}, is the minimum amount of time that must elapse before the PRECHARGE command can be issued to the open row. t_{RAS} is also referred to as ACTIVE-to-PRECHARGE time.

1.9.2.2 Read

Figure 1.23 illustrates the progression of a column read command. A column read command moves data from the sense amplifiers of a selected bank to the memory controller through IO gating and write drivers and data output register. The address A[9:0] from the address bus is stored into the column address latch and column decoder of the selected bank. The address bit BA0 selects the bank and its column

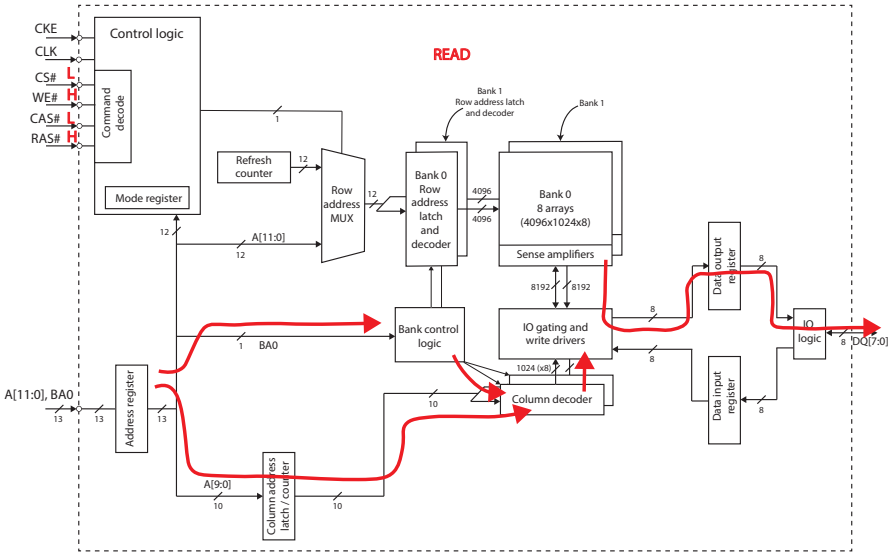


Fig. 1.23: The progression of the READ command.

decoder and sense amplifiers. Then, the selected 8-bit data is read from the sense amplifiers and output to DQ pins. There are two (timing) parameters associated with a column read command: CAS latency (CL) and burst length (BL).

CL is the time it takes for the SDRAM device to move the requested data from the sense amplifiers through IO gating and output register onto the data DQ bus. For SDRAMs, the **CAS latency (CL) is the delay, in clock cycles**, between the registration of a READ command and the availability of the output data. In modern SDRAMs, the CAS latency can be set to two or three clocks. If a READ command is registered at clock edge n, and the CL is m clocks, the data will be available by clock edge $n+m$. Now, we can combine the timing parameters, t_{RCD} and CL, to

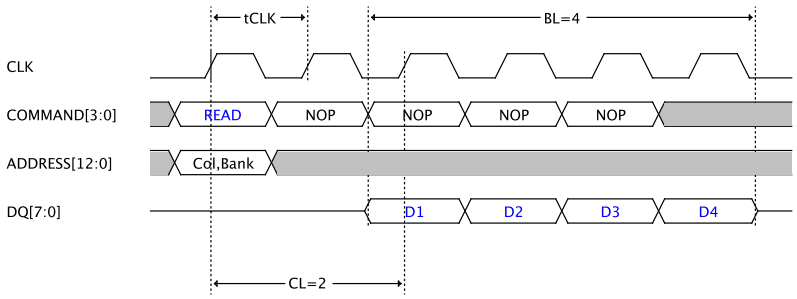


Fig. 1.24: The READ burst with CL=2 and BL=4.

form a **random access time** (t_{RAC}).

$$t_{RAC} = t_{RCD} + CL \tag{1.4}$$

Random access time, t_{RAC}, denotes the speed at which the SDRAM device can move data from the DRAM arrays into the memory controller.

Modern memory systems move data in relatively short bursts, and the burst length (BL) is programmable. The burst length determines the maximum number of column locations that can be accessed for a given READ or WRITE command. Typically, BL is 2, 4, or 8. Read bursts are initiated with a READ command, as shown in Figure 1.24. The starting column and bank addresses are provided with the READ command. During READ bursts, the valid data from the starting column address is available following the CAS latency after the READ command. Each subsequent data will be valid by the next positive clock edge. Upon completion of a burst, assuming no other commands have been initiated, the DQ signals will go to High-Z.

Data from a fixed-length READ burst can be followed immediately by data from a new READ or WRITE command. In such a way, a continuous flow of data can be maintained. SDRAM devices use a pipelined architecture, and therefore, a READ command can be initiated on any clock cycle following a READ command. The

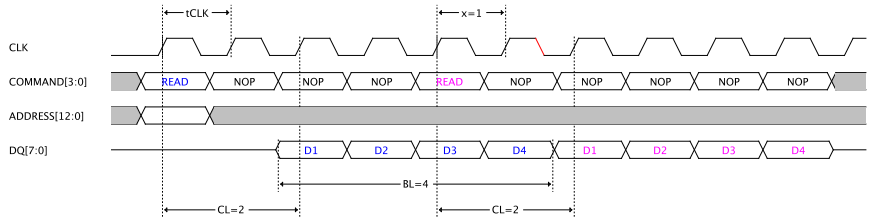


Fig. 1.25: Two consecutive READ bursts with CL=2 and BL=4.

new READ command should be issued x cycles before the clock edge at which the last desired data element is valid, where $x = CL - 1$. This is shown in Figure 1.25 for CL=2 and BL=4. Full-speed random read accesses can be performed to the same bank, or each subsequent READ can be performed to a different open bank (bank interleaving).

1.9.2.3 Write

Figure 1.26 illustrates the progression of the WRITE command. The WRITE command moves data from the DQs pins through IO gating and write drivers and data input register to the sense amplifiers of a selected bank. The column address A[9:0] from the address bus is stored into the column address latch and column decoder of

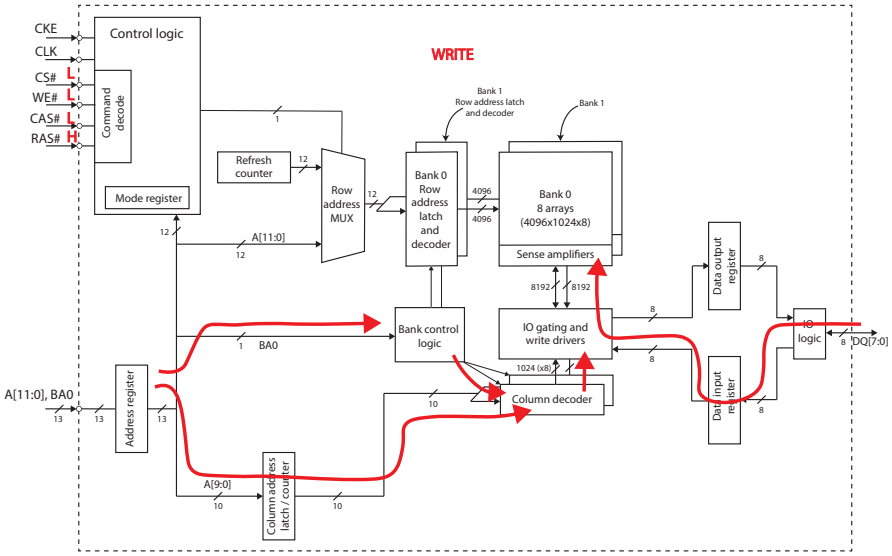


Fig. 1.26: The progression of the WRITE command.

the selected bank. The address bit BA0 selects the bank and its column decoder and sense amplifiers.

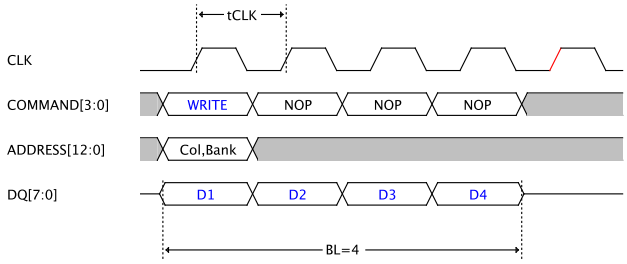


Fig. 1.27: The WRITE burst with BL=4.

Figure 1.27 shows a write burst with BL=4. The starting column and bank addresses are provided with the WRITE command, which initiates write bursts. During write bursts, the first valid data is registered coincident with the WRITE command. Subsequent data are registered on each successive positive clock edge. Upon com-

pletion of a fixed-length burst, assuming no other commands have been initiated, the DQ pins remain at High-Z, and any additional input data is ignored.

Data from a fixed-length WRITE burst can be followed immediately by data from a new READ or WRITE command. In such a way, a continuous flow of data can be maintained. Figure 1.28 shows two consecutive write bursts with BL=2.

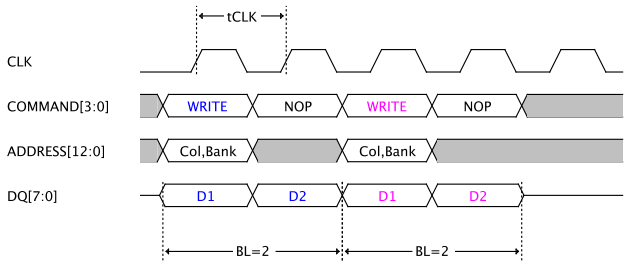


Fig. 1.28: Two consecutive WRITE bursts with BL=2.

1.9.2.4 Precharge

So far, we have seen that accessing data on a SDRAM device is a two-step process. First, the ACTIVE command opens a row in a selected bank and moves data from the DRAM cells in that row to the sense amplifiers. The data then remains in the sense amplifiers and can be transferred to or from SDRAM using the READ and WRITE commands. The PRECHARGE command is used to deactivate the open row in a particular bank or the open row in all banks - it restores data in the row, resets the sense amplifiers and the bit lines, and prepares the sense amplifiers for another row access. Figure 1.29 illustrates the progression of the PRECHARGE command. The address A[11:0] from the address bus is stored into the row address latch and decoder of the selected bank. The address bit BA0 selects the bank and its row address latch and decoder. Then the selected bank is precharged.

The timing parameter associated with the (row) PRECHARGE command is row precharge time,t_{RP}. The bank(s) will be available for a subsequent access row precharge time (t_{RP}) after the PRECHARGE command is issued. Recall that t_{RAS} is the minimum amount of time that the row should remain open before issuing the PRECHARGE command (i.e., ACTIVE-to-PRECHARGE time). Now, we can combine the timing parameters, t_{RP} and t_{RAS}, to form a **row cycle time** (t_{RC}):

$$t_{RC} = t_{RAS} + t_{RP} \tag{1.5}$$

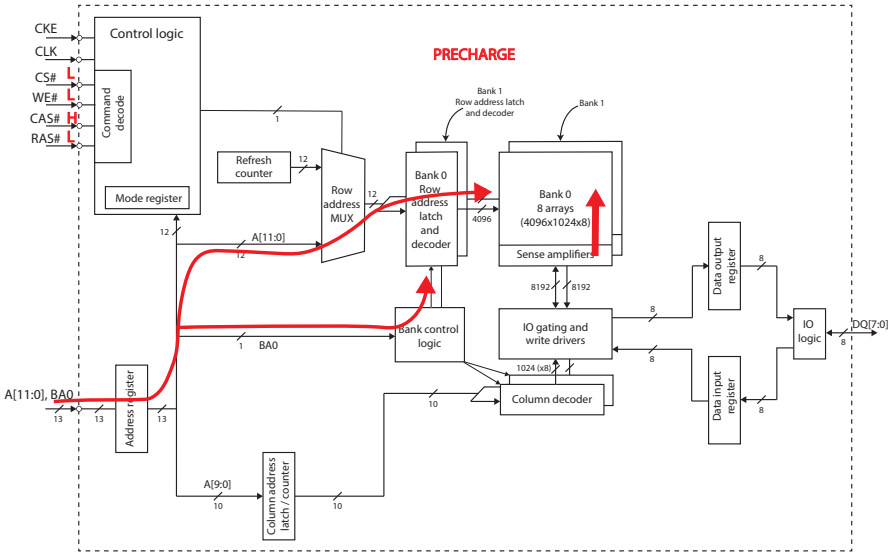


Fig. 1.29: The progression of the PRECHARGE command.

Row cycle time, t_{RC}, denotes the speed at which the SDRAM device can bring data from the DRAM arrays into the sense amplifiers, restore the data to the DRAM cells, and be ready for another ACTIVE command. t_{RC} is the fundamental limitation to the speed at which data may be retrieved from different rows within the same SDRAM bank.

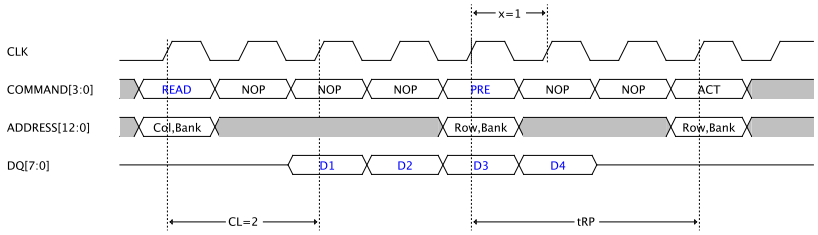


Fig. 1.30: READ to PRECHARGE.

A PRECHARGE command may follow a READ or WRITE burst to the same bank. In the case of PRECHARGE after READ, the PRECHARGE command should be issued $x = CL - 1$ cycles before the clock edge at which the last data element in a burst is valid. This is shown in Figure 1.30 for CL = 2. In the case of PRECHARGE after WRITE, the PRECHARGE command should be issued at

least one clock period after the positive clock edge at which the last input data is registered, regardless of frequency (Figure 1.31).

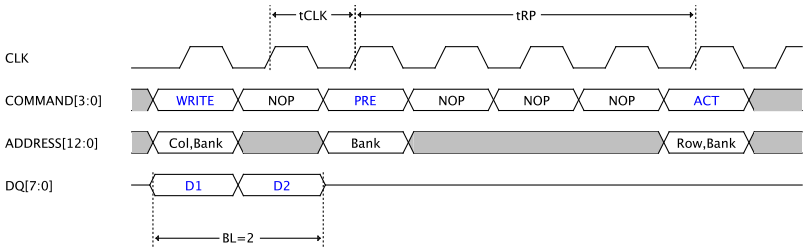


Fig. 1.31: WRITE to PRECHARGE.

Following the PRECHARGE command, a subsequent command to the same bank cannot be issued until t_{RP} is met. The disadvantage of the PRECHARGE command is that it requires that the command and address buses be available at the appropriate time to issue the command.

1.9.3 Case study: Using the STM32F Flexible Memory Controller to access SDRAM

The Flexible Memory Controller (FMC) found in STM32 microcontrollers consists of the following main blocks:

1. the interface to the CPU's Advanced High-performance Bus (AHB),
2. the NOR Flash/SRAM memory controller,
3. the SDRAM memory controller, and
4. NAND Flash controller.

The block diagram of the FMC is shown in Figure 1.32. The AHB interface allows the CPU (and other bus master peripherals) to access the external memories through the FMC controller. Two primary purposes of FMC are to translate transactions on the high-speed CPU bus (namely AHB bus) into the appropriate external protocol and to meet the access time requirements of the external memory devices.

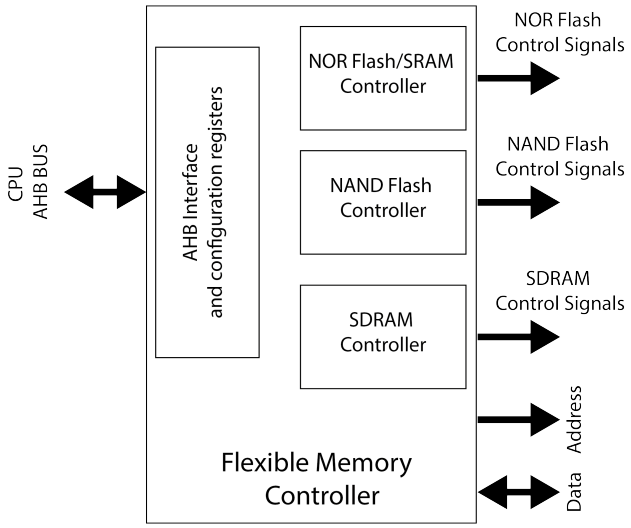


Fig. 1.32: FMC block diagram.

From the FMC (or microprocessor) point of view, the external memory is divided into six fixed-size regions of 256 Mbytes each, called banks (Figure 1.33). The first bank is used to address NOR Flash memory devices. The third bank is used to address NAND Flash memory devices. The last two banks are used to address two SDRAM devices (one device per bank). The address bit 28 on the AHB bus (internal AHB address line 28) selects one of the memory devices (banks). Let us focus only on the FMC SDRAM controller and an SDRAM device in the fifth bank.

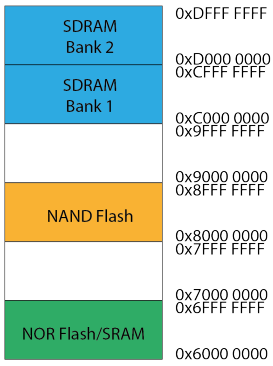


Fig. 1.33: Memory regions accessible from the FMC controller.

All external memories share the addresses, data and control signals with the controller, and each external device is accessed utilizing a unique chip-select signal. The FMC performs only one access at a time to an external device. Here, we will describe only the SDRAM controller and its use to interface a 128 Mbit SDRAM memory chip. All AHB transactions, in this case, translate into the SDRAM device protocol.

The FMC SDRAM controller supports SDRAM devices of up to 256 Mbytes. It can issue a 13-bit row address, an 11-bit column address, and a 2-bit bank address. The memory accesses can be 8-bit, 16-bit, and 32-bit. We will use Micron's 1 Meg x 32 x 4 banks MT48LC4M32B2 SDRAM chip, organized as 4096 rows x 256 columns x 32 bits per bank. Hence, the memory controller would issue a 12-bit row address, an 8-bit column address, and a 2-bit bank address.

The SDRAM controller in Figure 1.34 accepts single and burst read and write requests and translates them into single memory accesses. In both cases, the SDRAM controller keeps track of the active row in each bank to be able to perform consecutive read and write accesses. The FMC SDRAM controller comprises a read FIFO (6 lines x 32 bits). It is used to read data in advance - the memory controller anticipates READ commands to the open row if the RBURST bit is set in the FMC_SDCRx register and stores data in the FIFO. Two bits RPIPE[1:0] in the FMC_SDCRx register defines how much data will be anticipated and stored into the FIFO during the read access. If we set both RPIPE[1:0] bits to zero, four data will be anticipated during a single read access. The first read data will be transmitted to the AHB bus, and the other three will be stored in the read FIFO buffer. The read FIFO buffer stores a 14-bit address tag for each line to identify its content: 11 bits for the column address, 2 bits for the internal bank in the active row, and 1 bit for the SDRAM device. Each time a read request occurs, the SDRAM controller checks if the address matches one of the address tags in the read FIFO buffer. In such a case, data are directly read from the FIFO buffer. Otherwise, a new read command is issued to the SDRAM device, and new data is read to the FIFO buffer.

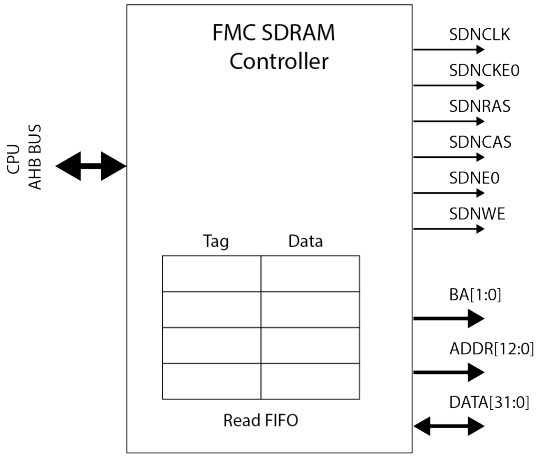


Fig. 1.34: FMC SDRAM Controller block diagram and signals.

The FMC SDRAM controller periodically issues auto-refresh commands to refresh the SDRAM. The programmer should initialize the internal counter value in the FMC_SDRTR. This value defines the number of memory clock cycles between two refresh cycles (refresh rate). When this counter reaches zero, the FMC SDRAM controller issues the auto-refresh command. If there is an ongoing memory access, the auto-refresh request is delayed until the memory access finishes; otherwise, the auto-refresh request takes precedence. If the memory access request occurs during an auto-refresh operation, the request is buffered and processed when the auto-refresh completes.

For our particular case, where the FMC SDRAM controller is used to access the MT48LC4M32B2 SDRAM chip, the 32-bit memory address from the AHB bus is mapped into the SDRAM address as presented in Figure 1.35. This figure illustrates how the 32-bit addresses issued by the CPU on the AHB bus map to the 26-bit addresses issued by the SDRAM controller to the SDRAM device.

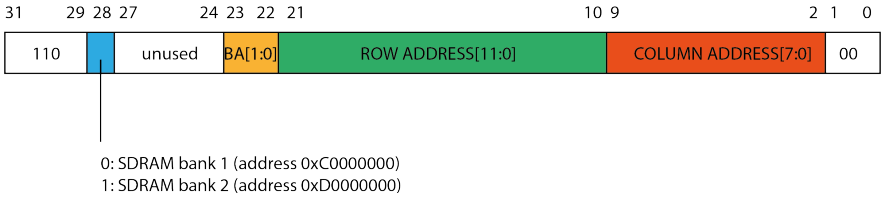


Fig. 1.35: Address maping for a 128-bit SDRAM (4096 rows x 256 columns x 4 banks x 32 bit).

In order to use the FMC SDRAM controller with an external SDRAM device residing in the SDRAM Bank 1, we should:

1. first, initialize the FMC SDRAM controller according to the used SDRAM device, and
2. secondly, initialize the SDRAM device.

The first step involves programming two FMC SDRAM controller configuration registers, SDRAM Control Register 1 (FMC_SDCR1) and SDRAM Timing Register 1 (FMC_SDTR1). The bits in FMC_SDCR1 (Figure 1.37) define the SDRAM clock period, CAS Latency, whether the FMC anticipates READ commands (burst read), data bus width and the internal organization of the SDRAM chip (rows, columns and banks).

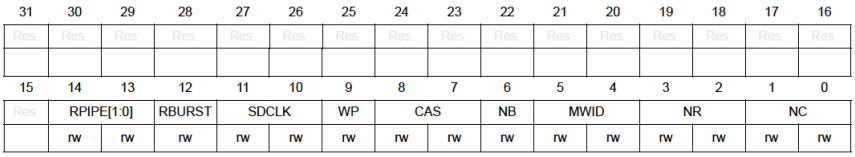


Fig. 1.36: Control register (FMC_SDCR).

The bits in FMC_SDTR1 define SDRAM timing parameters, e.g. RAS-to-CAS delay, row-precharge delay, etc. In order to correctly set the bits in these two registers, we should consult the datasheet for a particular SDRAM chip.

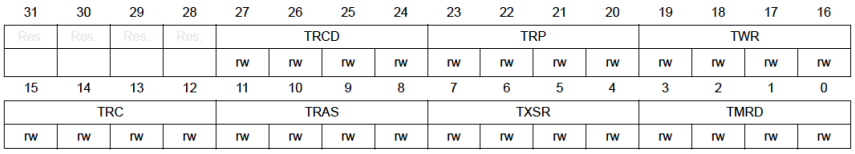


Fig. 1.37: Timing register (FMC_SDTR).

The second step initializes the SDRAM chip. During the SDRAM chip initialization, the FMC controller sends several predefined commands to the SDRAM chip. To send these commands, we should write them into the FMC SDRAM Command Mode Register (FMC_SDCMR). The required initialization steps are described in the datasheet for a particular SDRAM chip and involve the following:

1. providing stable CLOCK signal,
2. performing a PRECHARGE ALL command, which puts all rows in all banks into an idle state,

3. issuing several AUTO REFRESH commands
4. issuing several NOP commands before SDRAM is ready for access.

Instead of directly setting bits in the FMC SDRAM configuration registers, we will rather use the HAL library. The HAL library abstracts most of the FMC SDRAM controller hardware details. The FMC SDRAM controller is abstracted in HAL with the `SDRAM_HandleTypeDef` C structure. The two most important members of this structure are the C reference to `FMC_SDRAM_TypeDef Instance` structure and `FMC_SDRAM_InitTypeDef Init`. The `Instance` is a reference to the SDRAM registers (it holds the base address of the FMC SDRAM registers), while the `Init` structure allows for FMC SDRAM controller configuration. The `FMC_SDRAM_InitTypeDef Init` structure is defined as follows:

```
typedef struct
{
  uint32_t SDBank;
  uint32_t ColumnBitsNumber;
  uint32_t RowBitsNumber;
  uint32_t MemoryDataWidth;
  uint32_t InternalBankNumber;
  uint32_t CASLatency;
  uint32_t WriteProtection;
  uint32_t SDClockPeriod;
  uint32_t ReadBurst;
} FMC_SDRAM_InitTypeDef;
```

Listing 1.1: FMC SDRAM `FMC_SDRAM_InitTypeDef` C structure.

Let us briefly describe the elements of the `FMC_SDRAM_InitTypeDef Init` structure:

- `SDBank`: Specifies the SDRAM memory device that will be used (bank 1 or bank 2 according to Figure 1.33).
- `ColumnBitsNumber`: Defines the number of bits of the column address.
- `RowBitsNumber`: Defines the number of bits of the row address.
- `MemoryDataWidth`: Defines the memory device width.
- `InternalBankNumber`: Defines the number of the device's internal banks.
- `CASLatency`: Defines the SDRAM CAS latency in the number of memory clock cycles.
- `WriteProtection`: Enables/Disables the SDRAM device to be accessed in write mode.
- `SDClockPeriod`: Defines the SDRAM Clock Period for SDRAM devices. The SDRAM clock period can be HCLK/2 or HCLK/3, where HCLK is the clock period on the CPU's AHB bus.
- `ReadBurst`: Enables the SDRAM controller to anticipate the next read commands during the CAS latency and stores data in the Read FIFO.

Besides the `FMC_SDRAM_InitTypeDef` C structure, which abstracts the content of FMC_SDCR1 register, the `FMC_SDRAM_TimingTypeDef` C structure is used to abstract the content of FMC_SDTR1 register. It is defined as follows:

```
typedef struct
{
  uint32_t LoadToActiveDelay;
  uint32_t ExitSelfRefreshDelay;
  uint32_t SelfRefreshTime;
  uint32_t RowCycleDelay;
  uint32_t WriteRecoveryTime;
  uint32_t RPDelay;
  uint32_t RCDDelay;
} FMC_SDRAM_TimingTypeDef;
```

Listing 1.2: FMC SDRAM `FMC_SDRAM_TimingTypeDef` C structure.

The elements of the `FMC_SDRAM_TimingTypeDef` C structure are self-explanatory (they represent the particular timings for SDRAM chips), and there is no need to describe them.

The code Listing 1.3 shows the FMC SDRAM controller initialization.

```
uint8_t Init_SDRAM(void)
{
  static uint8_t sdramstatus = SDRAM_ERROR;
  /* SDRAM device configuration */
  sdramHand.Instance = FMC_SDRAM_DEVICE;

  /* Timing configuration for 100Mhz as SDRAM clock frequency
     (System clock is up to 200Mhz) */
  /* These parameters are from the MT48LC4M32B2 Data Sheet,
     Table 18 and Table 19 */
  sdramTiming.LoadToActiveDelay    = 2;   // t_MRD
  sdramTiming.ExitSelfRefreshDelay = 7;   // t_XSR
  sdramTiming.SelfRefreshTime      = 5;   // t_RAS
  sdramTiming.RowCycleDelay        = 7;   // t_RC
  sdramTiming.WriteRecoveryTime    = 2;   // t_WR
  sdramTiming.RPDelay              = 2;   // t_RP
  sdramTiming.RCDDelay             = 2;   // t_RCD


  sdramHand.Init.SDBank             = FMC_SDRAM_BANK1;
  sdramHand.Init.ColumnBitsNumber   = FMC_SDRAM_COLUMN_BITS_NUM_8;
  sdramHand.Init.RowBitsNumber      = FMC_SDRAM_ROW_BITS_NUM_12;
  sdramHand.Init.MemoryDataWidth    = FMC_SDRAM_MEM_BUS_WIDTH_32;
  sdramHand.Init.InternalBankNumber = FMC_SDRAM_INTERN_BANKS_NUM_4;
  sdramHand.Init.CASLatency         = FMC_SDRAM_CAS_LATENCY_3;
  sdramHand.Init.WriteProtection  = FMC_SDRAM_WRITE_PROTECTION_DISABLE;
  sdramHand.Init.SDClockPeriod      = FMC_SDRAM_CLOCK_PERIOD_2;
  sdramHand.Init.ReadBurst          = FMC_SDRAM_RBURST_ENABLE;
  sdramHand.Init.ReadPipeDelay      = FMC_SDRAM_RPIPE_DELAY_0;

  /* SDRAM controller initialization */

  if(HAL_SDRAM_Init(&sdramHand, &sdramTiming) != HAL_OK)
  {
    sdramstatus = SDRAM_ERROR;
  }
  else
  {
    sdramstatus = SDRAM_OK;
  }

  /* Once the FMC SDRAM Ctrl is initialized, we can access
     and initialize the SDRAM chip */
  /* SDRAM initialization sequence */
  SDRAM_Initialization_sequence(REFRESH_COUNT);
```

```
46
  return sdramstatus;
48 }
```

Listing 1.3: FMC SDRAM Controller initialization.

Firstly, we set the SDRAM timing parameters (in the FMC_SDTR1 register) considering the 100MHz SDRAM clock, then we set the SDRAM configuration (in the FMC_SDCR1 register). To initialize the FMC SDRAM controller (that is to copy the elements of both C structures into the appropriate fields of the FMC_SDCR1 and FMC_SDTR1 registers), we call the HAL function `HAL_SDRAM_Init(SDRAM_HandleTypeDef *hsdram, FMC_SDRAM_TimingTypeDef *Timing)`.

After the FMC SDRAM initialization, we should initialize the SDRAM chip. SDRAMs must be powered up and initialized in a predefined manner. This is a necessary step required to put all SDRAM rows in the idle state (precharge all rows) and prepare the SDRAM chip for accepting and executing the commands. The SDRAM initialization sequence is described in the SDRAM datasheet in detail. The code Listing 1.4 shows the FMC SDRAM chip initialization. Briefly, the initialization procedure contains four steps:

1. Enable the stable SDRAM clock.
2. Wait for at least 100us prior to issuing any command.
3. Perform a PRECHARGE ALL command.
4. Issue at least two AUTO REFRESH commands.
5. The SDRAM is now ready for mode register programming. Because the mode register will power up in an unknown state, it should be loaded with desired bit values prior to applying any operational command.

```
/**
2   * @brief  Init the SDRAM device.
  * SDRAMs must be initialized in a predefined manner. Operational ↩
    procedures
4   * other than those specified in the SDRAM Data Sheet may result in ↩
    undefined operation.
  * @param  RefreshCount: SDRAM refresh counter value
6   * @retval None
  */
8 void SDRAM_Initialization_sequence(uint32_t RefreshCount)
{
10   __IO uint32_t tmpmrd = 0;

12   /* Step 1: Configure a clock configuration enable command */
  sdramCmd.CommandMode            = FMC_SDRAM_CMD_CLK_ENABLE;
14   sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
  sdramCmd.AutoRefreshNumber      = 1;
16   sdramCmd.ModeRegisterDefinition = 0;

18

  /* Send the Clock Configuration Enable command to the target bank*/
20   /* The command is sent as soon as the Command MODE field in the
    CMR is written */
22   HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);

24   /*
```

```
    * Once the clock is stable, the SDRAM requires a 100us delay
    * prior to issuing any command
    */

  /* Step 2: Insert 100 us minimum delay */
  /* Inserted delay is equal to 1 ms due to systick time base unit */
  HAL_Delay(1);


  /*
   * Once the 100us delay has been satisfied, a PRECHARGE command
   * should be applied. All banks must then be precharged,
   * thereby placing the device in the all banks idle state.
   */
  /* Step 3: Configure a PALL (precharge all) command */
  sdramCmd.CommandMode            = FMC_SDRAM_CMD_PALL;
  sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
  sdramCmd.AutoRefreshNumber      = 1;
  sdramCmd.ModeRegisterDefinition = 0;

  /* Send the Precharge All command to the target bank */
  /* The command is sent as soon as the Command MODE field
     in the CMR is written */
  HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);

  /*
   * Once in the idle state, at least two AUTO REFRESH cycles must
   * be performed. If desired, more than two AUTO REFRESH
   * commands can be issued in the sequence.
   */
  /* Step 4: Configure an Auto Refresh command */
  sdramCmd.CommandMode            = FMC_SDRAM_CMD_AUTOREFRESH_MODE;
  sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
  sdramCmd.AutoRefreshNumber      = 8;
  sdramCmd.ModeRegisterDefinition = 0;

  /* Send the Auto-refresh commands to the target bank */
  /* The command is sent as soon as the Command MODE
     field in the CMR is written */
  HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);


  /*
   * The SDRAM is now ready for mode register programming.
   * Because the mode register will power up in an unknown state,
   * it should be loaded with desired bit values prior to
   * applying any operational command. Using the LMR command,
   * program the mode register.
   */
  /* Step 5: Program the external memory mode register */
  tmpmrd = (uint32_t)SDRAM_MODEREG_BURST_LENGTH_1          |\
                     SDRAM_MODEREG_BURST_TYPE_SEQUENTIAL   |\
                     SDRAM_MODEREG_CAS_LATENCY_3           |\
                     SDRAM_MODEREG_OPERATING_MODE_STANDARD |\
                     SDRAM_MODEREG_WRITEBURST_MODE_SINGLE;

  sdramCmd.CommandMode            = FMC_SDRAM_CMD_LOAD_MODE;
  sdramCmd.CommandTarget          = FMC_SDRAM_CMD_TARGET_BANK1;
  sdramCmd.AutoRefreshNumber      = 1;
  sdramCmd.ModeRegisterDefinition = tmpmrd;

  /* Send the Load Mode Register command to the target bank */
  /* The command is sent as soon as the Command MODE field in
     the CMR is written */
  HAL_SDRAM_SendCommand(&sdramHand, &sdramCmd, SDRAM_TIMEOUT);

  /*
```

```
92      * Wait for at least tMRD time. This is automatically performed by
        * the FMC SDRAM controller. At this point the DRAM is ready for
94      * any valid command.
        */
96
    /* Step 6: Set the refresh rate counter in Refresh Timer register */
98  /* This 13-bit field defines the refresh rate of the SDRAM device.
       It is expressed in number of memory clock cycles. */
100 HAL_SDRAM_ProgramRefreshRate(&sdramHand, RefreshCount);
}
```

Listing 1.4: SDRAM initialization sequence.

To enable the above procedure, the FMC SDRAM controller provides a special register called Command Mode register (FMC_SDCMR), illustrated in Figure 1.38. It contains four fields. The MODE field defines the command issued to

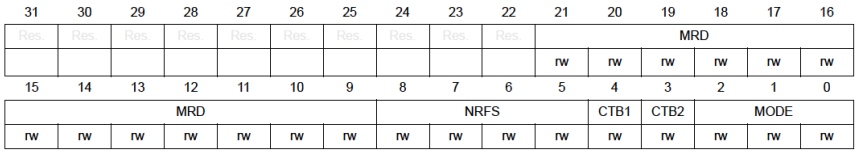


Fig. 1.38: Command Mode register (FMC_SDCMR).

the SDRAM chip. The possible commands are, for example, "CLK ENABLE", "PRECHARGE ALL", "AUTO REFRESH", and "LOAD MODE REGISTER". The CTB1 and CTB2 fields select the SDRAM chip to which the command is sent. As soon as the MODE field is written, the FMC SDRAM controller will issue the corresponding command to SDRAM chips selected by CTB1 and CTB2 command bits. The NRFS field defines how many consecutive Auto-refresh commands are issued in the fourth step of the initialization sequence, the MRD field contains the content that should be written to the SDRAM Mode Register. The mode register is a 12-bit special register inside the SDRAM chip and is used to define the specific mode of operation of the SDRAM. This definition includes the selection of a burst length (BL), a burst type, a CAS latency (CL), an operating mode and a write burst mode, as shown in Figure 1.39. The mode register is programmed from the FMC SDRAM controller via the "LOAD MODE REGISTER" command and retains the stored information until it is programmed again or the SDRAM device loses power.

The initialization of the SDRAM device is performed by sending a series of commands from the FMC_SDCMR register to the SDRAM device. Each command contains the actual instruction and its parameters. To facilitate the SDRAM chip initialization, HAL provides the `FMC_SDRAM_CommandTypeDef` C structure and `HAL_SDRAM_SendCommand` function. The `FMC_SDRAM_CommandTypeDef` C structure abstracts the content of FMC_SDCMR register and is defined as follows:

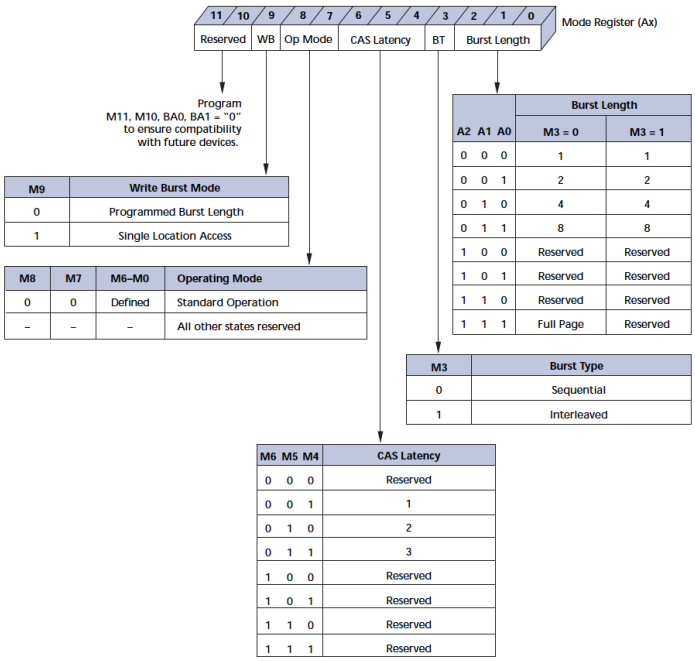


Fig. 1.39: SDRAM Mode Register.

```
typedef struct
{
  uint32_t CommandMode;
  uint32_t CommandTarget;
  uint32_t AutoRefreshNumber;
  uint32_t ModeRegisterDefinition;
} FMC_SDRAM_CommandTypeDef;
```

Listing 1.5: FMC SDRAM `FMC_SDRAM_CommandTypeDef` C structure.

Let us briefly describe the elements of the `FMC_SDRAM_CommandTypeDef Init` structure:

- `CommandMode`: Defines the command issued to the SDRAM device.
- `CommandTarget`: Defines which SDRAM device (1 or 2) the command will be issued to.
- `AutoRefreshNumber`: Defines the number of consecutive auto-refresh commands issued in auto-refresh mode.
- `ModeRegisterDefinition`: Defines the SDRAM Mode register content.

In order to send a command to the SDRAM device, we first fill the fields in the `FMC_SDRAM_CommandTypeDef Init` structure and then call the `HAL_SDRAM_SendCommand` function.

At the end of the SDRAM chip initialization, we set the auto-refresh period in the FMC SDRAM controller. The AUTO REFRESH command is used during the regular operation of the SDRAM to refresh its content. This command is nonpersistent, so it must be issued each time a refresh is required. If memory access is in progress, the auto-refresh request is delayed. The refresh controller inside the SDRAM chip generates the address of the row that should be refreshed. For example, the 128Mb SDRAM requires 4096 AUTO REFRESH commands every 64ms. To ensure that each row is refreshed according to this requirement, the SDRAM controller must issue an AUTO REFRESH command every 15.625us. The FMC SDRAM controller provides the Refresh Timer register (FMC_SDRTR). This register holds the 13-bit refresh rate in number of SDRAM clock cycles. This 13-bit field should be set immediately after the initialization of SDRAM. The 13-bit refresh rate is calculated as follows. As the SDRAM clock runs at 100 Mhz (10 ns period), 15.625 us equals 1562 SDRAM clock periods. We should subtract at least 20 SDRAM clock periods from this value to obtain a safe margin if an auto-refresh request occurs when a read request has been accepted. Hence, the 13-bit refresh rate in the FMC_SDRTR register corresponds to 1542.

To demonstrate the different scenarios when using the FMC SDRAM controller, we copy a matrix of size 64 rows times 256 columns from the external SDRAM to the internal SRAM. The elements of the matrix are 32-bit unsigned integers. In the first scenario (Listing 1.6), the matrix is accessed in row-major order, while in the second scenario (Listing 1.7), the matrix is accessed in column-major order. The constants `PA3_SDRAM_DEVICE_ADDR` and `SDRAM_COLS` in Listings 1.6 and 1.7 equal 0xC0008000 and 256, respectively. Hence, the matrix is read from the SDRAM startin at address 0xC000800.

```
void SDRAM_mat_row_access_test(void){
  volatile uint32_t address;

  for (int i = 0; i<MAT_ROWS; i++) {
    for(int j=0; j<SDRAM_COLS; j++) {
      address = PA3_SDRAM_DEVICE_ADDR + ((i*SDRAM_COLS + j)<<2);
      matrixB[i][j] = *(uint32_t*)address;
    }
  }
}
```

Listing 1.6: Read matrix from SDRAM in row-major order.

```
void SDRAM_mat_col_access_test(void){
  volatile uint32_t address;

    for (int i = 0; i<SDRAM_COLS; i++) {
    for(int j=0; j<MAT_ROWS; j++) {
      address = PA3_SDRAM_DEVICE_ADDR + ((j*SDRAM_COLS + i)<<2);
      matrixB[j][i] = *(uint32_t*)address;
    }
  }
}
```

Listing 1.7: Read matrix from SDRAM in column-major order.

Figure 1.40 illustrates one read issued from the CPU for the first scenario (row-major order access). The FMC SDRAM controller does not support SDRAM burst reads or writes (the only allowable burst length is 1). Instead, it supports burst reads on the CPUs AHB bus by utilizing the internal FIFO. Hence, it anticipates four READ commands to fill in the internal FIFO. The FIFO content is then transferred to the CPU using the AHB burst read of length 4.

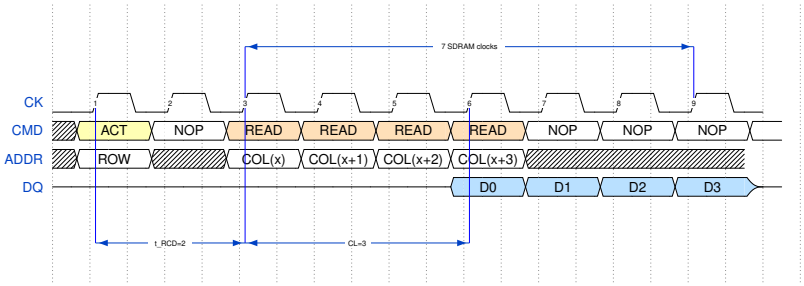


Fig. 1.40: Using row-major order to read a matrix, the SDRAM controller anticipates four consecutive READ command to the active SDRAM row for each read initiated from the CPU

In the second scenario, the matrix is accessed using column-major order. Figure 1.41 illustrates two consecutive reads issued from the CPU. As the CPU reads data from consecutive rows in each iteration, the CPU controller first reads four consecutive words from the active SRAM row and fills the internal FIFO, but it only returns one word to the CPU over the AHB bus. As the CPU starts another read from the next row, the SDRAM controller first precharges the active row. It then waits for two SDRAM clock periods (Row Precharge time) before activating the next row.

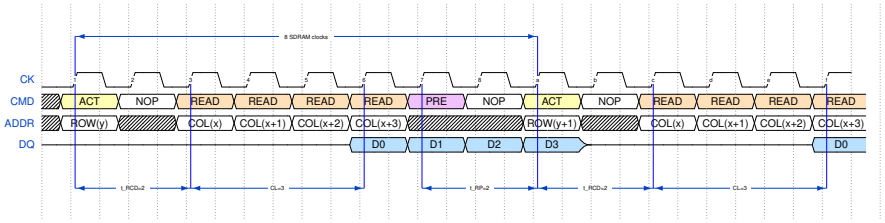


Fig. 1.41: Using column major order results in activating, reading and precharging an SDRAM row for every read issued from the CPU.

It is obvious that row-major order access is considerably faster than column-major order access. A rough estimate of the access time for row-major order access

considering an already open row is seven (7) SDRAM clock periods per four words. On the other side, a rough estimate of the access time for column-major order access is eight (8) SDRAM clock periods per word. Recall that only one word is transferred to the CPU, although the SDRAM controller anticipates four consecutive reads from the active row.

To assess the performance (speed) of the row-major and column-major matrix reads, we use the code in Listing 1.8. For each test, the code first sets the PC8 pin and reads the timer TIM3 counter value (this is the start of the test). After the test, we reset the PC8 pin and read the timer TIM3 counter value (this is the start of the test). By setting and resetting the PC8 pin, we can measure the duration of each test using an oscilloscope. The timer TIM3 runs at 1MHz (1 us resolution). Hence, we can estimate the duration of each test simply by reading the timer counter before and after the test.

```
// Row-major order access:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_mat_row_access_test();
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
if (timer_val_end > timer_val_start)
  elapsed_rows = timer_val_end - timer_val_start;
else
  elapsed_rows = timer_val_end + (65536-timer_val_start);

// Column-major order access:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_mat_col_access_test();
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
if (timer_val_end > timer_val_start)
  elapsed_cols = timer_val_end - timer_val_start;
else
  elapsed_cols = timer_val_end + (65536-timer_val_start);
```

Listing 1.8: Code used to test the speed of row-major and column-major matrix read from the SDRAM.

Figure 1.42 shows the oscilloscope trace for the signal on the GPIOC pin. It shows that the row-major order read lasts for about 2.3 ms, while the column-major order read lasts for about 10 ms. Using the timer counter, we estimate the duration of the row-major order read to 2365 us and the duration of the column-major order read to 9816 us. Both measurements show that the row-major order read is about four times faster than the column-major order read, which is in accordance with the rough estimation from figures 1.40 and 1.41.

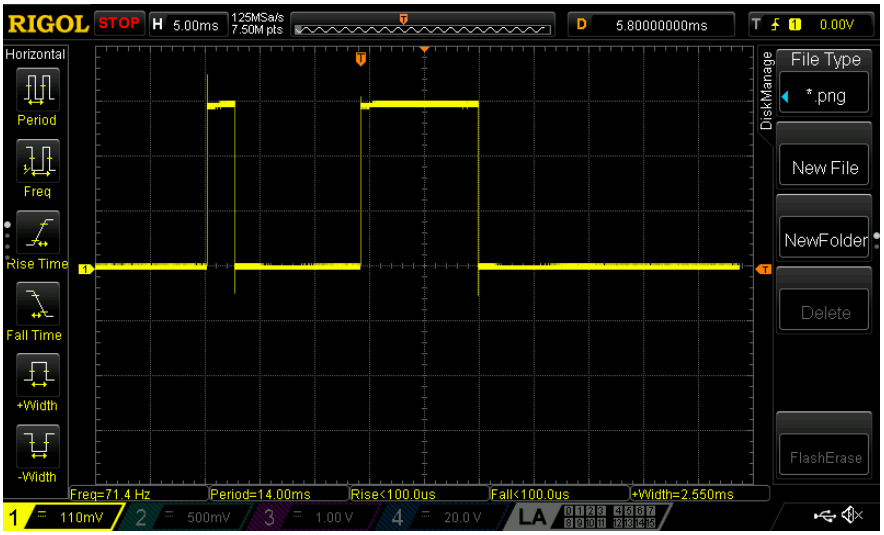


Fig. 1.42: Oscilloscope trace on the GPIOC pin 8. The row-major order matrix read lasts for about 2.5 ms while the column-major order matrix read lasts for more than 10 ms.

1.9.3.1 Using DMA to transfer data from an external SDRAM to the internal SRAM

Direct memory access (DMA) is used to provide high-speed data transfer between peripherals and memory and between memory and memory without any CPU action (except DMA controller initialization and DMA transfer request in case of memory-to-memory transfer). As already described in Section **??**, the DMA controller in STM32 microcontrollers (actually, there are two DMA controllers, DMA1 and DMA2, respectively) have 16 streams in total (8 for each DMA controller), each dedicated to managing memory access requests from one or more peripherals. Each stream can have up to 8 channels (requests) in total. Each DMA controller has an arbiter for handling the priority between DMA requests. According to the STM32F69I reference manual, the memory-to-memory mode in DMA is a mode that doesn't need any triggering request from a peripheral, and it will happen just after the stream enable bit is set. Also, according to the STM32F69I reference manual, only the DMA2 could handle memory-to-memory data transfers. The stream can be enabled just by setting the Enable bit (EN) in the DMA SxCR register. Then, the stream immediately fills the FIFO up to the threshold level. When the threshold level is reached, the FIFO contents are drained and stored in the destination.

Before using the DMA2 controller to transfer data from one memory region to another, we must configure (initialize) the DMA2 controller as described in Section **??**. When configuring the DMA controller we:

1. Select a stream that we wish to use. Any available stream in the DMA2 controller can be used for memory-to-memory transfers.
2. Select a channel; this is irrelevant for memory-to-memory transfers because a peripheral device does not trigger the DMA transfer through a channel. Instead, it is triggered by setting the EN bit in the DMA SxCR register.
3. Set a priority for a selected DMA stream.
4. Set the number of data to be transferred (it can be any value from 1 to 65535).
5. Set the source and destination transfer width (byte, half-word, word).
6. Set the source and destination addresses.
7. Select whether the source and destination addresses should be incremented during the transfer. For memory-to-memory transfers, both addresses should be incremented during the transfer.
8. Select whether the burst transfers of 4, 8 or 16 beats should be used during the transfer.

Programming DMA is relatively easy. Recall from Section **??** that each stream can be controlled using four registers: memory address register, peripheral address register, number of data register, and configuration register. Once set, DMA takes care of memory address increment without disturbing the CPU. Now that it is clear how the DMA works from a theoretical point of view, we can use the HAL library to configure and use a DMA controller. The HAL library abstracts most of the underlying hardware details. The DMA controller is abstracted in HAL with a C structure `DMA_HandleTypeDef`. Let us describe more in-depth only the two most important fields of this structure:

- `Instance`: this is the pointer to the DMA Stream descriptor we will use. For example, `DMA2_Stream1` indicates the first stream of DMA2. The stream descriptor is a C structure that contains all DMA stream registers. The reference to the Instance structure points to the actual peripheral address. For example, the `DMA2_Stream1` is defined in HAL as a pointer to the stream descriptor structure, and it holds the register base address for DMA2 Stream1 registers.
- `Init`: is an instance of the C structure `DMA_InitTypeDef`, which is used to configure the DMA Stream and channel.

`DMA_InitTypeDef` is defined in the following way:

```
typedef struct
{
  uint32_t Channel;
  uint32_t Direction;
  uint32_t PeriphInc;
  uint32_t MemInc;
  uint32_t PeriphDataAlignment;
  uint32_t MemDataAlignment;
  uint32_t Mode;
  uint32_t Priority;
  uint32_t FIFOMode;
  uint32_t FIFOThreshold;
  uint32_t MemBurst;
  uint32_t PeriphBurst;
}DMA_InitTypeDef;
```

Listing 1.9: DMA `DMA_InitTypeDef` C structure.

Let us briefly describe the C `DMA_InitTypeDef` structure:

- `Channel`: Specifies the channel used for the specified stream. It can assume the values DMA_CHANNEL_0, DMA_CHANNEL_1 up to DMA_CHANNEL_7. The peripherals are bound to streams and channels during the MCU design, so we should consult the datasheet for our microcontroller to see the stream/channel bound to the peripheral we want to use with DMA.
- `Direction`: Specifies if the data will be transferred from memory-to-peripheral, memory-to-memory or peripheral-to-memory.
- `PeriphInc`: Specifies whether the Peripheral address register should be incremented or not during the DMA transfer. Recall that a DMA controller has one peripheral port used to specify the address of the peripheral register involved in the DMA transfer. Since a DMA transfer usually involves several bytes, the DMA can be configured to increment the peripheral register for every transmitted byte.
- `MemInc`: Specifies whether the memory address register should be incremented or not during the DMA transfer.
- `PeriphDataAlignment`: Specifies the Peripheral data width. Transfer data sizes of the peripheral and memory are fully programmable through this field and the next one. The DMA controller is designed to automatically perform data alignment when source and destination data sizes differ.
- `MemDataAlignment`: Specifies the Memory data width.
- `Mode`: the DMA controller has two working modes: normal and circular. In normal mode, the DMA sends the specified amount of data from the source to the destination port and stops the activities. It must be re-activated again to do another transfer. In circular mode, it automatically resets the transfer counter at the end of transmission and starts transmitting again from the first byte of the source buffer.
- `Priority`: Specifies the software priority for the DMA Stream. The priority allows the internal arbiter in the DMA controller to rule concurrent requests.
- `FIFOMode`: Specifies if the stream uses the FIFO buffer. Recall that each stream has an independent 4-word (4 * 32 bits) FIFO. The FIFO temporarily stores data coming from the source before transmitting it to the destination. The FIFO introduces one important advantage: it reduces S(D)RAM access time by supporting burst transactions. The FMC SDRAM controller used in our case issues four READ commands in a row, thus reading four consecutive words from an active SDRAM row. Using the FIFO allows for storing these four words efficiently before they are sent to SRAM.
- `FIFOThreshold`: Specifies the FIFO threshold level. The FIFO will be drained to the destination when this threshold is achieved.
- `MemBurst`: Specifies the amount of data to be transferred to/from memory in a single non-interruptible transaction.

- `PeriphBurst`: Specifies the amount of data to be transferred to/from peripheral (or memory for mem-to-mem DMA transfers) in a single non-interruptible transaction.

All HAL functions related to DMA manipulation are designed so that they accept as the first parameter an instance of the C structure `DMA_HandleTypeDef`. To initialise the DMA Stream, we first set all desired parameters in the `DMA_InitTypeDef` structure and then use the HAL function `HAL_DMA_Init(DMA_HandleTypeDef *hdma)`. The following code illustrates configuring and initialising the DMA2 Stream 1 for memory-to-memory transfers using FIFO and burst of length 4:

```
HAL_StatusTypeDef DMA2_SDRAM_Config(DMA_HandleTypeDef* DmaHandle)
{
  /* Enable DMA2 clock */
  __HAL_RCC_DMA2_CLK_ENABLE();

  /* Select the DMA Stream to be used #*/
  DmaHandle->Instance = DMA2_Stream1;

  /* Set the DMA Parameters */
  /* DMA_CHANNEL_0                    */
  DmaHandle->Init.Channel = DMA_CHANNEL_0;
  /* M2M transfer mode                */
  DmaHandle->Init.Direction = DMA_MEMORY_TO_MEMORY;
  /* Peripheral increment mode Enable */
  DmaHandle->Init.PeriphInc = DMA_PINC_ENABLE;
  /* Memory increment mode Enable     */
  DmaHandle->Init.MemInc = DMA_MINC_ENABLE;
  /* Peripheral data alignment : Word */
  DmaHandle->Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD;
  /* memory data alignment : Word     */
  DmaHandle->Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
  /* Normal DMA mode                  */
  DmaHandle->Init.Mode = DMA_NORMAL;
  /* priority level : high            */
  DmaHandle->Init.Priority = DMA_PRIORITY_HIGH;
  /* FIFO mode enabled                */
  DmaHandle->Init.FIFOMode = DMA_FIFOMODE_ENABLE;
  /* FIFO threshold: full             */
  DmaHandle->Init.FIFOThreshold = DMA_FIFO_THRESHOLD_FULL;
  /* Memory burst                     */
  DmaHandle->Init.MemBurst = DMA_MBURST_INC4;
  /* Peripheral burst                 */
  DmaHandle->Init.PeriphBurst = DMA_PBURST_INC4;

  /* Initialize the DMA stream */
  if (HAL_DMA_Init(DmaHandle) != HAL_OK)
  {
    /* Initialization Error */
    return HAL_ERROR;
  }

  /* Configure NVIC for DMA transfer complete/error interrupts */
  HAL_NVIC_SetPriority(DMA2_Stream1_IRQn, 0, 0);

  /* Enable the DMA STREAM global Interrupt */
  HAL_NVIC_EnableIRQ(DMA2_Stream1_IRQn);

  return HAL_OK;
}
```

Listing 1.10: DMA2 Controller configuration and initialization.

In the above code, the DMA2 Stream 1 is configured to automatically increment the source and destination addresses for each transmitted word. Remember that we are transferring the block of contiguous memory words in the matrix from the external SDRAM to the matrix in the internal SRAM. Hence, the addresses in the source and destination block should increase for each word transferred. Also, we enable the FIFO, set the FIFO threshold level to full and enable the burst transfers of size 4. In such a way, the DMA controller will read four words from SDRAM to the FIFO and then transfer them to SRAM.

After DMA initialisation, we should set the priority for the DMA2 Stream 1 interrupt and enable the interrupt request generated by DMA2 Stream 1. The DMA2 Controller will then assert an interrupt request whenever the DMA 2 Stream 1 completes the DMA transfer. Hence, we should also implement the minimal interrupt handler for the DMA2 Stream 1 as follows:

```
void DMA2_Stream1_IRQHandler(void)
{
  /* Check the interrupt and clear flag */
  HAL_DMA_IRQHandler(&DMA2_SDRAM_Handle);
}
```

Listing 1.11: DMA2 Stream 1 Interrupt Handler.

Now we should create a DMA handle variable and initialize the DMA2 Stream 1 by passing reference to the DMA handle into the `DMA2_SDRAM_Config` function:

```

DMA_HandleTypeDef     DMA2_SDRAM_Handle;

...

// Configure DMA2 for SDRAM:
if (DMA2_SDRAM_Config(&DMA2_SDRAM_Handle) != HAL_OK) {
   Error_Handler();
}
```

Listing 1.12: DMA2 Stream 1 handle and its configuration.

To initiate the memory-to-memory DMA transfer, we use `HAL_DMA_Start(DMA_HandleTypeDef *hdma, uint32_t SrcAddress, uint32_t DstAddress, uint32_t DataLength)`. The arguments are the pointer to the DMA handle, the source address, the destination address and the number of data to transfer. This function sets the EN bit in DMA SxCR, which in turn triggers the DMA controller to start the transfer. Hence, to transfer a matrix from the external SDRAM to SRAM, we implement and use the following function:

```
void SDRAM_DMA_mat_row_access_test(void){
  volatile uint32_t address;

  for (int k = 0; k < N; k++)
  {
    HAL_DMA_Start(&DMA2_SDRAM_Handle,
                  (uint32_t) PA3_SDRAM_DEVICE_ADDR_RW,
```

```
                        (uint32_t) matrixB,
                        MAT_ROWS * SDRAM_COLS);
    HAL_DMA_PollForTransfer(&DMA2_SDRAM_Handle,
                                HAL_DMA_FULL_TRANSFER,
                                HAL_MAX_DELAY);
  }
}
```

Listing 1.13: Matrix transfer using DMA.

The function `HAL_DMA_PollForTransfer()` waits for DMA transfer to complete. Otherwise, the CPU would continue to execute the program and would not bother with DMA transfer (which would be the desired way), but in our case, we are going to measure the time required to transfer the matrix from SDRAM to SRAM using DMA; hence, we should wait for DMA to terminate. The function `HAL_DMA_PollForTransfer()` is used here for the sake of simplicity, but it is strongly recommended to use the DMA interrupt handler instead. Finally, we can add the DMA matrix transfer to a set of the previous performance tests in Listing 1.8 as follows:

```
// Row-major order access:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_mat_row_access_test();
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
if (timer_val_end > timer_val_start)
  elapsed_rows = timer_val_end - timer_val_start;
else
  elapsed_rows = timer_val_end + (65536-timer_val_start);

// Column-major order access:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_mat_col_access_test();
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
if (timer_val_end > timer_val_start)
  elapsed_cols = timer_val_end - timer_val_start;
else
  elapsed_cols = timer_val_end + (65536-timer_val_start);

// DMA transfer:
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_SET);
timer_val_start = __HAL_TIM_GET_COUNTER(&TIM3Handle);
SDRAM_DMA_mat_row_access_test();
timer_val_end = __HAL_TIM_GET_COUNTER(&TIM3Handle);
HAL_GPIO_WritePin(GPIOC, GPIO_PIN_8, GPIO_PIN_RESET);
if (timer_val_end > timer_val_start)
  elapsed_cols = timer_val_end - timer_val_start;
else
  elapsed_cols = timer_val_end + (65536-timer_val_start);
```

Listing 1.14: Code used to test the speed of row-major, column-major and DMA matrix read from the SDRAM.

When executing the DMA performance test, we observe from Figure 1.43 that the time required to transfer the matrix from the external SDRAM to the internal SRAM is only about 1500 us. Why is the DMA controller faster than the CPU, considering

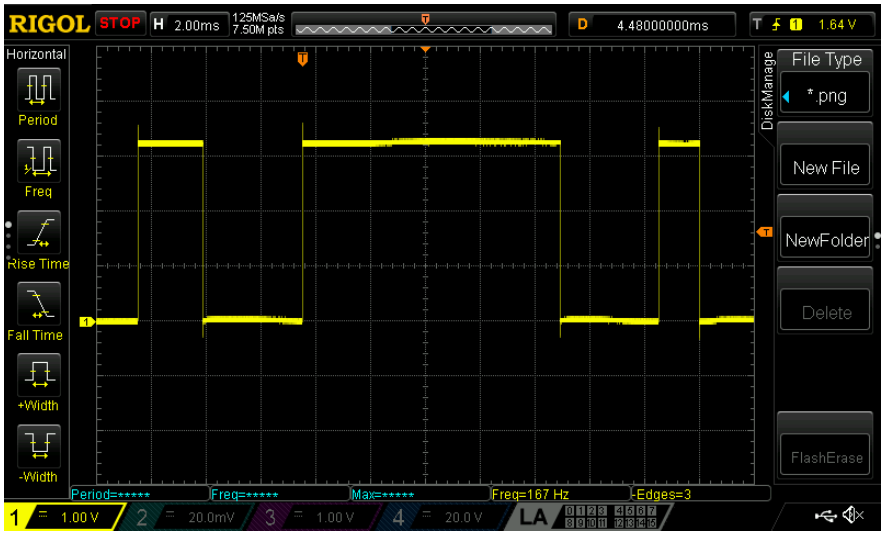


Fig. 1.43: Oscilloscope trace on the PC8 pin. The row-major order matrix read lasts for about 2.5 ms, the column-major order matrix read lasts forabout 10 ms while DMA transfer lasts for about 1.5 ms.

that the same amount of data is being transferred from/to the same devices in both cases?

```
J_LOOP:
  ; address = PA3_SDRAM_DEVICE_ADDR_RW  + ((i*SDRAM_COLS + j)<<2);
  add.w    r1, r3, #0            ; r1 <- r3
  ldr      r2, [pc, #60]         ; r2 <- 0xC0008000 (SDRAM address)
  add.w    r2, r2, r1, lsl #2    ; r2 <- r2+(r1*4) LOAD FROM SDRAM
  ; matrixB[i][j] = *(uint32_t*)address;
  ldr      r0, [r2, #0]          ; r0 <- M_SDRAM[r2]
  ldr      r2, [pc, #52]         ; r2 <- matB base address
  str.w    r0, [r2, r1, lsl #2]  ; matB[i][j] <- r0  STORE TO SRAM
  ; for(int j=0; j<SDRAM_COLS; j++) {
  adds     r3, #1                ; inc r3 (r3 holds j)
  cmp      r3, #255              ; if j <= 255
  ble.n    J_LOOP                ;       loop back
```

Listing 1.15: Assembly code corresponding to the instructions created by the compiler for the innermost loop in Listing 1.6. There are 11 instructions executed in each iteration of the innermost loop; hence 11 instructions are executed for transferring one word from SDRAM to SRAM. The first four instructions are used to calculate the address in SDRAM. Then, four instructions are used to read the word from SDRAM and write it to SRAM, and finally, the last three instructions increments the innermost loop counter, compare it to 255 and loop if not equal.

Well, the answer lies in the fact that the DMA controller does not execute instructions. For each word transferred, the CPU fetches the `LDR` instruction (load register

with word), executes it (it loads the data from SDRAM to an internal register), fetches the STR instruction (store register as word), and finally executes it (it stores the data from the internal register to SRAM). Besides LDR and STR instructions, in each loop iteration, the CPU executes a bunch of other instructions required to calculate the address in SDRAM, increment and compare the loop index, etc. (see Listing 1.15). The DMA controller only transfers data from SDRAM (in bursts!) and forwards them to SRAM (in bursts!) without fetching and executing the load-/store instructions! Besides offloading the CPU, this is another benefit of utilizing DMA controllers.

1.10 Double Data Rate SDRAM

How can we further speed-up memory transfers? The solution is to access two adjacent columns simultaneously with one READ/WRITE command. So, instead of reading/writing one 8-bit memory word (column), we can read/write two adjacent 8-bit memory words (columns). But with that solution, a new challenge arises. How to transfer two 8-bit words in the same amount of time as one 8-bit word? One solution would be to have a twice wider bus. Thus, instead of the 8-bit data bus (DQ[7:0]), the SDRAM device would have had a 16-bit data bus (DQ[5:0]). But this could be challenging because more wires mean more noise on the data bus and worse data/signal integrity. The second solution would be to have a twice faster bus. But this is also challenging because higher frequency means worse data/signal integrity and higher power consumption. The better solution is to **transfer data at both clock edges to double data bus bandwidth without a corresponding increase in clock frequency or in data bus width**.

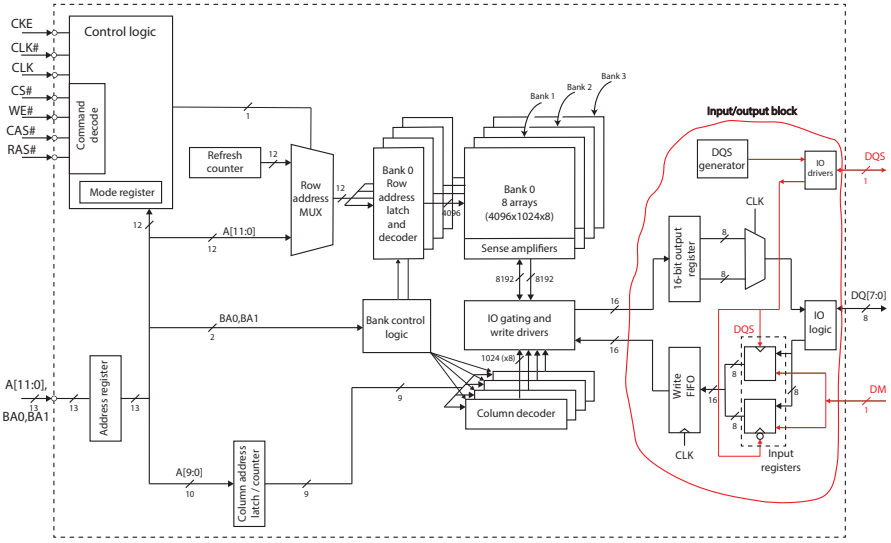


Fig. 1.44: Simplified block diagram of a DDR SDRAM device with four banks.

In SDRAM devices, each time a column read command is issued, the control logic determines the duration of the data burst, and each column is moved separately from the sense amplifiers through the I/O logic to the external data bus. However, the separate control of each column limits the operating data rate of the SDRAM device. **In Double Data Rate (DDR) SDRAM devices, two adjacent columns are moved in parallel from the sense amplifiers to the output data register**, and the data is then pipelined through a multiplexor to the external data bus. The feature

to access two columns at a time is referred to as **2N-prefetch**. Figure 1.44 illustrates the simplified block diagram of a DDR SDRAM device with four independent banks. We can see that the internal structure is similar to the internal structure of an SDRAM device except for the IO block. The memory arrays and banks used in DDR SDRAMs are the same as in SDRAMs. The name "double data rate" refers to the fact that a DDR SDRAM with a certain clock frequency achieves nearly twice the bandwidth of an SDRAM running at the same clock frequency, due to this double pumping. Double data rate SDRAM is a significant improvement of SDRAM. DDR SDRAMs have been used in computer systems' memory since 2001.

The main difference in the internal organization of DDR SDRAM over SDRAMs is an improved I/O block. The I/O block of an 8-bit DDR SDRAM device from Figure 1.44 now consists of a 16-bit output register, a 2/1 multiplexor, a DQS generator, two 8-bit input registers, a write FIFO and IO logic. Figure 1.44 shows that, in the case of the READ access, given the width of the external data bus (DQ) as 8-bit, 16 bits are moved from the sense amplifiers to the output register, and the 16 bits are then pipelined through the multiplexor to the external data pins. The clock signal controls the select input of the multiplexor. In the case of the WRITE access, two 8-bit data are stored successively (one after the other) in two 8-bit input registers and then transferred together into a 16-bit write FIFO. From there, data is transferred to the sense amplifiers through IO gating and write drivers. Besides, DDR SDRAMs have two new control signals: data strobe (DQS) and data mask (DM). In the following subsections, we are going to describe the operation of the IO block during the READ and WRITE accesses, and the role of DQS and DM in more detail.

The downside of the 2N-prefetch architecture means that short column bursts are no longer possible. In DDR SDRAM devices, a minimum burst length of 2 columns of data is accessed per column read command.

1.10.1 Functional description

The DDR SDRAM uses a double data rate architecture to achieve high-speed operation. The double data rate architecture is essentially a 2N-prefetch architecture with an I/O block designed to transfer two data words per clock cycle at the I/O pins. A single read or write access for the DDR SDRAM effectively consists of a single 2N-bit-wide, one clock cycle data transfer at the internal DRAM core, and two corresponding N-bit-wide, one-half clock cycle data transfers at the I/O pins.

The DDR SDRAM operates from a **differential clock** . Differential clock employs two complementary clock signals, CLK and CLK#. In general, a clock signal can be regarded as a binary signal whose duty cycle is nominally 50%. As we know, the clock signal is used to synchronize and capture data at its rising or falling edges. In DDR SDRAMs, data are synchronized and captured at both clock edges. But clocks are notoriously bad at having 50% duty cycles at high frequencies. As a rule of thumb, high frequency is generally considered to be above 100MHz. So, the reason for having two separate clocks is to allow for more precise alignment of

the rising edges of the clock with the data. The crossing of CLK going HIGH and CLK# going LOW is referred to as the positive edge of CLK. Commands (address and control signals) are registered at every positive edge of CLK.

Read and write accesses to the DDR SDRAM are burst oriented. Accesses start at a selected location and continue for the BL number of locations in a sequence. Similarly to SDRAMs, accesses begin with the registration of an ACTIVE command, which may then be followed by a READ or WRITE command. The address bits registered coincident with the ACTIVE command are used to select the bank and row to be accessed. The address bits registered coincident with the READ or WRITE command are used to select the bank and the starting column location for the burst access. The DDR SDRAM provides for programmable READ or WRITE burst lengths of 2, 4, or 8 locations.

1.10.1.1 Read

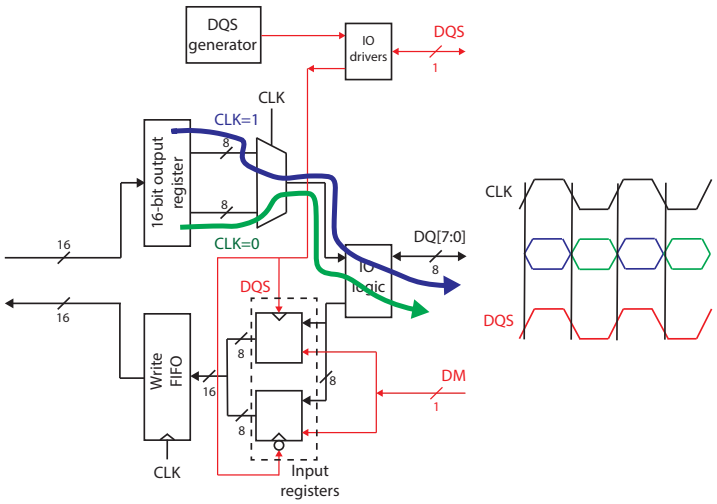


Fig. 1.45: Operation of the IO block during READ.

Figure 1.45 illustrates the operation of the I/O block during the READ access to an 8-bit DDR SDRAM. First, 16 bits (two adjacent 8-bit columns) are transferred from the sense amplifiers to the 16-bit output register as the consequence of the READ command. Then, when CLK is HIGH, the first 8-bit word is transferred through the multiplexor onto the I/O pins; when the CLK signal is LOW, the second 8-bit word is transferred through the multiplexor onto the IO pins. In such a way, two 8-bit words from the DRAM array are transferred in one clock cycle. A bidirectional data strobe (DQS) signal is transmitted, along with data, for use in data capture at the

memory controller. The DQS generator generates the DQS signal and synchronizes it with the memory controller's global clock. The **DQS signal is edge-aligned with data for READs**.

1.10.1.2 Write

Figure 1.46 illustrates the operation of the I/O block during the WRITE access to an 8-bit DDR SDRAM. Two 8-bit words are successively transferred from the data bus into the input registers.Two input registers form a DDR input pair. A bidirectional data strobe **(DQS) signal is now transmitted by the memory controller**, along with data, for use in data capture at DDR SDRAM. The first 8-bit word is captured into the first data input register at the positive edge od DQS, while the second 8-bit word is captured into the second input register at the negative edge of DQS. Hence, **input data is registered on both edges of DQS**, and **DQS signal is center-aligned with data for WRITEs**. Then, the 16-bit data is transferred into the write FIFO at the positive edge od the CLK signal and written to the sense amplifiers and the DRAM array during the PRECHARGE command.

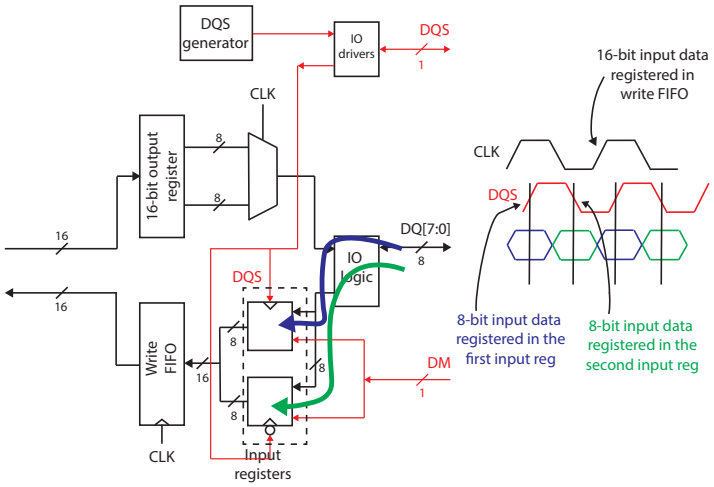


Fig. 1.46: Operation of the IO block during WRITE.

1.10.2 DDR SDRAM timing diagrams

1.10.2.1 Read bursts

Figure 1.47 shows the timing for a read burst with CL=2 and BL=4. During READ bursts, the valid data-out element from the starting column address is available following the CL after the READ command. Each subsequent data-out element is valid at the next positive or negative clock edge (i.e., at the next crossing of CLK and CLK#). DQS is driven by the DDR SDRAM along with output data. The initial LOW state on DQS is known as the *read preamble*; the LOW state coincident with the last data-out element is known as the *read postamble*. Upon completion of a read burst, assuming no other commands have been initiated, the DQ will go High-Z.

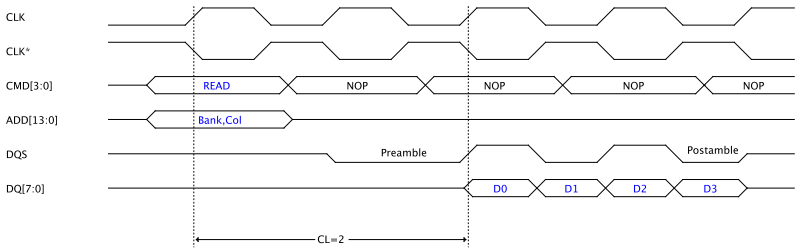


Fig. 1.47: The DDR READ burst with CL=2 and BL=4.

Data from any READ burst may be concatenated with data from a subsequent READ command. In such a way, a continuous flow of data can be maintained. The first data element from the new burst will follow the last element of a completed burst if the new READ command is issued x cycles after the first READ command, where x equals the number of desired data element pairs (pairs are required by the 2N-prefetch architecture). This is shown in Figure 1.47.

A PRECHARGE command may follow a READ burst to the same bank. The PRECHARGE command should be issued x cycles after the READ command, where x equals the number of desired data element pairs ($x = BL/2$). This is shown in Figure 1.49. Following the PRECHARGE command, a subsequent command to the same bank cannot be issued until both t_{RAS} and t_{RP} have been met.

1.10.2.2 Write bursts

Figure 1.50 shows the timing for a WRITE burst with BL=4. Input data appearing on the DQ is written to the memory array subject to the data mask (DM) input coin-

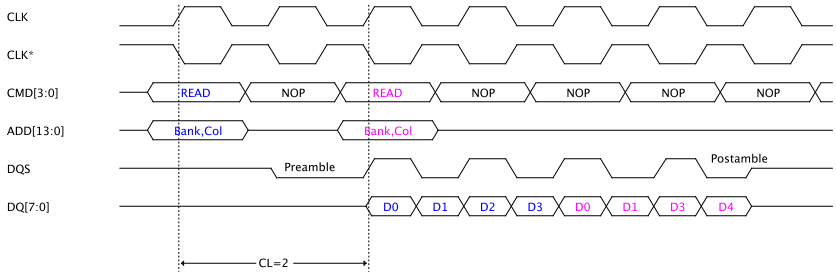


Fig. 1.48: Two consecutive DDR READ bursts with CL=2 and BL=4.

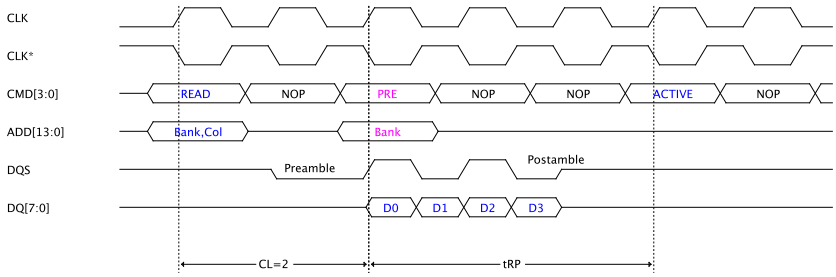


Fig. 1.49: DDR READ to PRECHARGE.

cident with the data. The DQS and DM signals are now transmitted by the memory controller, along with data. If the DM signal is registered LOW, the corresponding input data is written to memory. If the DM signal is registered HIGH, the corresponding input data is ignored, and a WRITE is not executed to that column location. During WRITE bursts, the first valid input data element is registered on the first rising edge of DQS following the WRITE command. Subsequent data elements are registered on the successive edges of DQS. The LOW state on DQS between the WRITE command and the first rising edge is known as the *write preamble*, and the LOW state on DQS following the last input data element is known as the *write postamble*. The first input data element following the WRITE command, along with its DQS, should be valid on the data bus one clock period after the WRITE command. Actually, most modern DDR SDRAMs specify this time between the WRITE command and the first corresponding rising edge of DQS from 75% to 125% of one clock cycle. In all of the WRITE diagrams, this time is one clock cycle.

Data for any WRITE burst may be concatenated with a subsequent WRITE command. The new WRITE command should be issued x cycles after the first WRITE

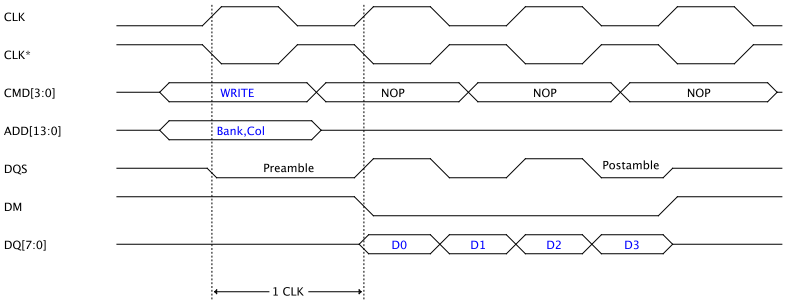


Fig. 1.50: The DDR WRITE burst with BL=4.

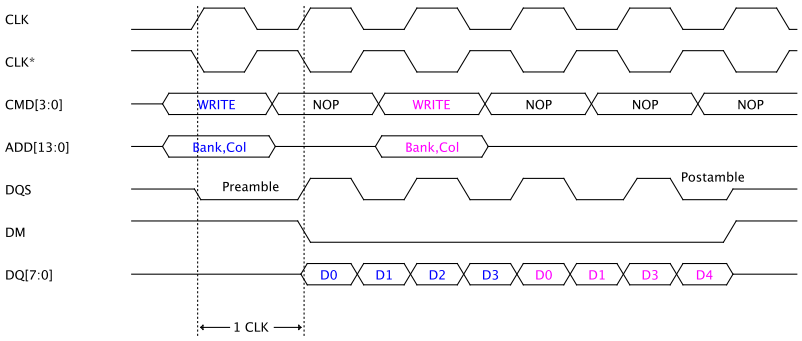


Fig. 1.51: Two DDR WRITE bursts with BL=4.

command, where x equals the number of desired data element pairs. Figure 1.51 illustrates two concatenated bursts with BL=4.

A PRECHARGE command to the same bank may follow a WRITE burst, as shown in Figure 1.52. There is a time period, **write recovery time** (t_{WR}), associated with the WRITE-to-PRECHARGE command sequence. Only the data-in pairs registered prior to the t_{WR} period are written to the internal array. After the PRECHARGE command, a subsequent command to the same bank cannot be issued until t_{RP} is met.

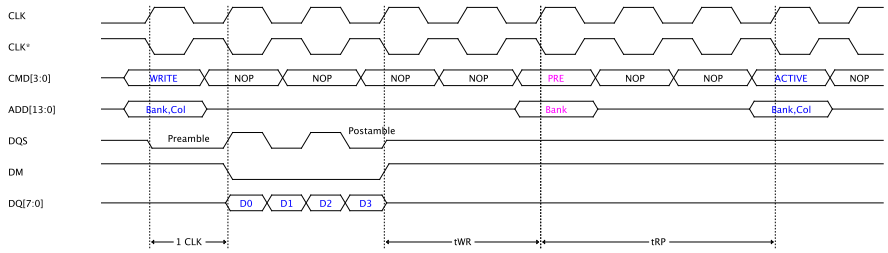


Fig. 1.52: DDR WRITE to PRECHARGE.

1.10.3 Address Mapping

Now that we are familiar with the basic operations in SDRAMs, we can move forward and see how an address from the CPU should be mapped into SDRAM's bank, row, and column address. The memory controller performs the address mapping. Let us suppose we are addressing a DDR SDRAM chip that consists of 8 banks, and each bank has eight DRAM arrays of size 4096 rows by 1024 columns. To address such a DDR SDRAM chip, we need 12 bits for the row address, three bits for the bank address, and 10 bits for the column address.

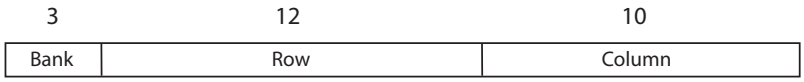


Fig. 1.53: Naive address mapping.

Figure 1.53 shows the naive way of an address mapping, where the top address bits are used to address the bank, the middle 14 bits are used to address the row, and the last 10 bits select the column. The main problem of such naive address mapping is that consecutive rows are in the same bank; hence, there is no bank interleaving. In the case of consecutive memory transfers consisting of more than one row, the currently open row should first be precharged before the new row is open.

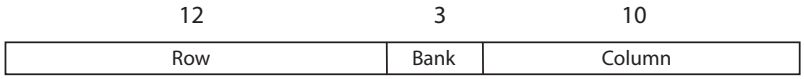


Fig. 1.54: Bank interleaving.

The better way of an address mapping would be to take advantage of bank interleaving, such that consecutive rows are in different banks. In this way, we can open a new row before the currently accessed row is precharged. We say that the precharge time is masked. Figure 1.54 shows the address mapping, where **bank interleaving** is used. Now, the top address bits select the row, while the middle address bits select the bank. Each time the end of a row is reached, the same row in a different bank is accessed.

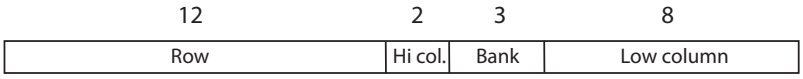


Fig. 1.55: Cache block interleaving.

The third way of an address mapping would be to take into account the cache memory. Typically, the cache block is of size 64 bytes. In reality, memory reads or writes are rarely random due to locality of reference. If a cache is used to support the locality of references, the CPU will access consecutive cache blocks. Hence, the cache misses will occur on the consecutive 64 bytes in memory. For example, if a cache block is stored in the last 64 bytes of a row, the cache miss on the next cache block would require to precharge the row and open a new one. In the case were consecutive cache blocks are stored in different banks, a row precharge would not be required. Thus it would be better to put consecutive cache blocks into different banks - this is **called cache block interleaving**. Figure shows the address mapping, where cache block interleaving is used. Now the column bits are split into two parts. Low column bits select the word within the cache block. The remaining hi column bits address the cache block in different banks.

1.10.4 Memory timings: a summary

So far, we have learned that each memory operation is associated with one or more memory timings that should be met in order to perform these operations correctly. Table 1.2 summarizes the most important memory timings.

CL, t_{RCD}, and $t_{RP}$are for most modern SDRAMs, typically around 13 ns, and have not changed significantly since the SDRAMs were first introduced. Actually, the DRAM cell and array process technologies have not significantly changed over the decades, and only the techniques to speed-up memory transfers have been (e.g. synchronous interface, bank interleaving, etc.). The next subsection covers the techniques to speed-up memory transfers in DDR SDRAMs.

Table 1.2: Summary of important timings in SDRAMs.

Name	Symbol	Description
CAS latency	CL	The number of cycles between sending a column address to the memory and the beginning of the data in response to a READ command. This is the number of cycles it takes to read the first bit of memory from a DRAM with the correct row already open. CL is an exact number that must be agreed on between the memory controller and the memory.
Row Address to Column Address Delay	t_{RCD}	The minimum number of clock cycles required between opening a row and issuing a READ/WRITE command. The time to read the first bit of memory from an SDRAM without an active row is t_{RCD}+ CL.
Row Precharge Time	t_{RP}	The minimum number of clock cycles required between issuing the precharge command and opening the next row. The time to read the first bit of memory from an SDRAM with the wrong row open is t_{RP}+ t_{RCD}+ CL.
Row Active Time	t_{RAS}	The minimum number of clock cycles required between a row active command and issuing the precharge command. This is the time needed to internally refresh the row, and overlaps with t_{RCD}. In SDRAM modules, it is usually t_{RCD}+ CL.

1.10.5 DDR Versions

To bust the performance od DDR SDRAMs, DDR SDRAMs have been further improved. Due to its nature (data is stored as a charge) and the process technology used to implement DRAM cells, the DRAM core (DRAM arrays) has not changed significantly over the decades, and its speed of operation remains relatively low. In SDRAMs, the clock rate used to transfer data on the data bus equals the clock rate used to transfer data between internal latches, sense amplifiers, and input/output data registers. The following improvements aim to speed-up memory transfers by employing larger prefetch or by increasing the frequency on the data bus (and not the frequency of the SDRAM core). These subsequent improved versions of DDR SDRAM are numbered sequentially: DDR2, DDR3, and DDR4.

DDR SDRAMs have 2N-prefetch, and the typical frequencies of the SDRAM core and the data bus are 133, 167, and 200 Mhz. In DDR2 SDRAM devices, the number of columns prefetched is 4. Hence, **DDR2 employs 4N-prefetch**. Besides, DDR2 internal clock runs at half the DDR2 external bus clock rate. DDR2 offers data bus clock rates of 266 MHz, 333 MHz, and 400 MHz. DDR2 also lowers power by dropping the voltage from 2.5 volts (DDR) to 1.8 volts. **DDR3 increased the prefetch to 8N**. DDR3 bus clock rate is 4 times faster than DDR3 internal clock rate. DDR3 also drops the voltage to 1.5 volts and has a maximum data-bus clock speed of 800 MHz. **DDR4 also employs 8N-prefetch** but drops the voltage to 1 to 1.2 volts and has a maximum data-bus clock rate of 1600 MHz. DDR4 bus clock rate is 4 times faster than DDR4 internal clock rate.

1.11 DIMM Modules

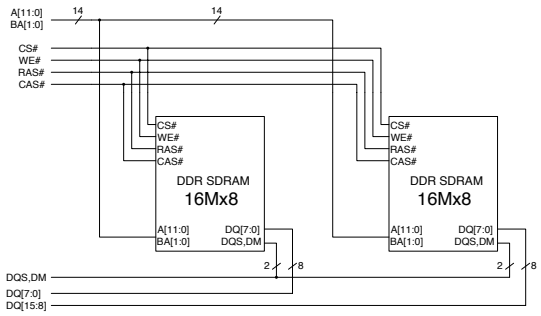


Fig. 1.56: A rank composed of two DRAM 16Mx8 chips.

The capacities of one DDR3 SDRAM chip are 1, 2, 4, and 8 Gbits, while the capacities of one DDR4 SDRAM chip are 4, 8, 16, and 32 Gbits. To increase memory capacity and bandwidth, we can connect two or more chips together, as illustrated in Figure 1.56. Each chip in Figure 1.56 is the DDR SDRAM chip from Figure 1.44, containing four banks, each of size 4096x1024x8 bits. Hence, one DDR SDRAM chip is of size 16Mx8 bits. Both chips in Figure 1.56 share the memory, the control (DQS and DM), and the command bus (CS#, WE#, RAS#, and CAS#); hence, both chips are accessed simultaneously. A set of DRAM chips connected to the same chip select (CS#) signal, which are therefore accessed simultaneously, is referred to as **a rank**. The chips in figure form a DDR SDRAM of size 16MX16 bits. Hence, connecting two DRAM chips as in Figure 1.56 we have increased the capacity and the data bus bandwidth, as now 16 data bits are transferred simultaneously.

We can further increase the size and the bandwidth od DRAM by connecting more than two chips in one rank. Figure 1.57 illustrates a rank composed of four DDR SDRAM chips of size 16Mx8 bits. Again, all four DDR SDRAM chips share the same CS# signal and are accessed simultaneously. The rank is of size 16Mx32 bits, as now 32 data bits are transferred simultaneously.

We can even form two independent ranks. In such a way, we can interleave the accesses to both ranks (similarly to bank interleaving) and mask latencies: while accessing one rank, we can activate a row in another rank or refresh another rank. Figure 1.58 illustrates two independent ranks, Ran0, and Rank1. For each rank, there is a separate CS# signal: CS0# for Rank 0, and CS0# for Rank 0. Now both ranks share the same data bus, as only one rank can be read or written at the same time.

In modern computer systems, DRAM chips are combined on a printed circuit board designed for use in personal computers, workstations, and servers. The memory chips are placed on both sides of the printed circuit board. Typically, there are eight (8) memory chips placed on one side of the printed circuit boards. A printed

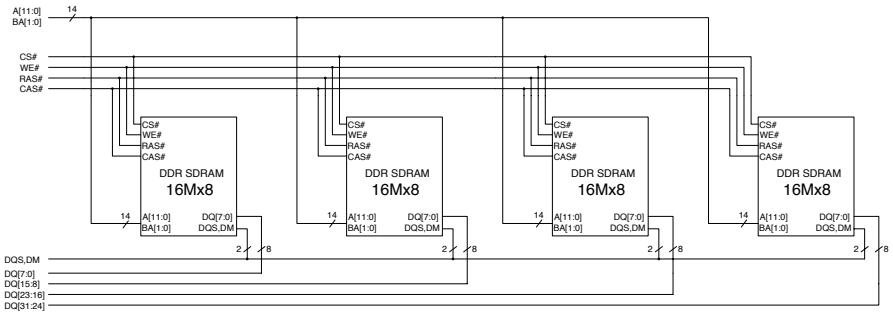


Fig. 1.57: A rank composed of four DRAM 16Mx8 chips.

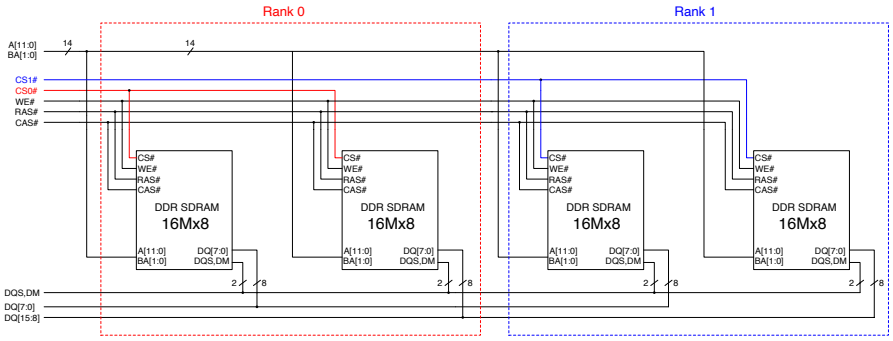


Fig. 1.58: Two ranks each containing two DRAM 16Mx8 chips.

circuit board containing memory chips on both sides is referred to as **dual in-line memory module (DIMM).** For instance, the 64-bit data bus for DIMM requires eight 8-bit chips, addressed in parallel. The DRAM chips on one side of the DIMM module form one rank: they share the same chip select (CS#) signal and are therefore accessed simultaneously. Figure 1.59 illustrates a DIMM module and its two ranks, Rank 0 and Rank 1. In practice, all DRAM chips on DIMM share all of the other command and control signals, and only the chip select pins for each rank are separate. Each side of a DIMM, containing eight 8-bit DRAM chips is one rank, and each rank has a 64-bit-wide data bus.

Manufacturers use the rather confusing labeling of SDRAM chips and DIMM modules. When DDR SDRAMs are packaged as DIMMs, they are confusingly labeled by the peak DIMM bandwidth. For example, when DDR SDRAMs with a clock frequency of 133 MHz are packed as a DIMM, the DIMM name becomes PC2100. The name comes from 133MHz x 2(DDR) x 8 bytes (eight 8-bit DRAM chips in a rank) equals 2100 MB/sec. Also, confusing names are used to label the DRAM chips. DRAM chips are labeled with the number of bits per second rather than their clock rate, so a 133 MHz DDR SDRAM chip is called a DDR266. Table

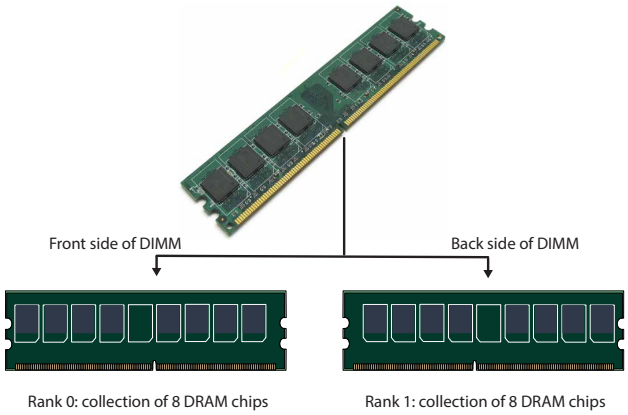


Fig. 1.59: A DIMM module.

Table 1.3: Comparison of DDR SDRAM generations and DIMMs.

Generation		Chip		Data bus		Timings		DIMM		
DRAM	**DRAM name**	**Clock (Mhz)**	**Prefetch**	**Clock (MHz)**	**MT/s**	**CL-t_{RCD}-t_{RP}**	**t_{CL} (ns)**	**MB/s**	**Voltage**	**DIMM name**
DDR	DDR-266	133		133	266	2.5-3-3	18.8	2128		PC-2100
DDR	DDR-300	150	2N	150	300			2400	2.5	PC-2400
DDR	DDR-400	200		200	400	3-3-3	15	3200		PC-3200
DDR2	DDR2-533	133		266	533	4-4-4		4264		PC2-4300
DDR2	DDR2-667	166	4N	333	667	5-5-5	15	5336	1.8	PC2-5300
DDR2	DDR2-800	200		400	800	6-6-6		6400		PC2-6400
DDR3	DDR3-1066	133		533	1066	7-7-7	13.12	8528		PC3-8500
DDR3	DDR3-1333	166	8N	666	1333	9-9-9	13.5	10664	1.5	PC3-10700
DDR3	DDR3-1600	200		800	1600	11-11-11	13.75	12800		PC3-12800
DDR4	DDR4-2400	300	8N	1200	2400	18-18-18	13.5	19200	1.2	PC4-19200
DDR4	DDR4-2666	333		1333	2666	20-20-20	13.6	21333		PC4-21333
DDR4	DDR4-3200	400		1600	3200	22-22-22	13.75	25600		PC4-25600

1.3 shows the relationships among internal and data-bus clock rates, prefetch, transfers per second per chip, chip names, DIMM bandwidth, DIMM supply voltage and and DIMM names.

DDR, DDR2, DDR3 and DDR4 memories are classified according to the maximum speed at which they can work, as well as their timings. The important memory timings of commercial memory chips are usually given as triple:

$$CL - t_{RCD} - t_{RP} \ ,$$

where CL, t_{RCD}, and t_{RP} are given in data-bus clock cycles. For example, a DDR3-1333 chip can be described as 9-9-9, meaning that CL equals nine bus clock cycles,

t_{RCD} equals nine bus clock cycles, and t_{RP} equals nine bus clock cycles. As the bus clock rate of a DDR3-1333 chip is 667MHz, all timings equal 13.5 ns.

1.11.1 Micron DDR4 DIMM module

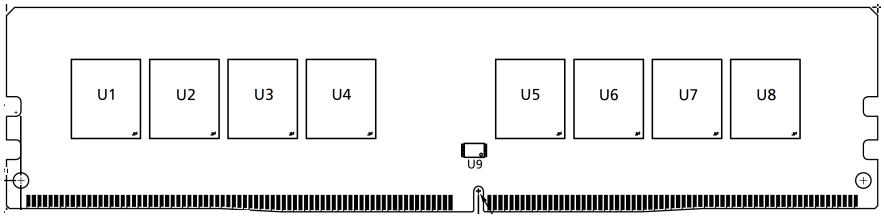


Fig. 1.60: 288-Pin Micron (1G x 64 bit) DDR4 SDRAM DIMM - Front side.

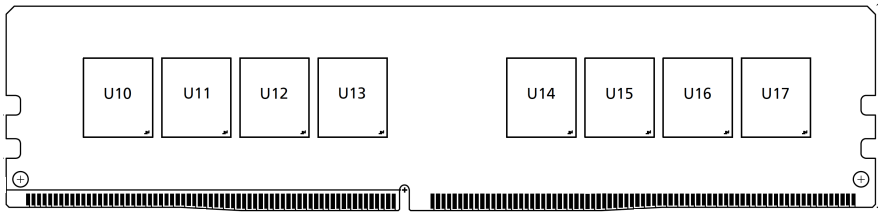


Fig. 1.61: 288-Pin Micron (1G x 64 bit) DDR4 SDRAM DIMM - Back side.

1.12 Memory channels

We have learned that multiple banks and multiple ranks enable concurrent DRAM accesses. Multiple ranks can be further used to form a **channel**, but only one rank can be activated at a time. **Multiple independent channels** serve the same purpose as multiple banks or ranks, but they are even better because they **have separate data buses**. In such a way, bus bandwidth is increased. The advantage of running two or four channels is that they will provide the same capacity as a larger single-channel, while at the same time doubling and quadrupling the amount of memory bandwidth. Of course, multiple channels bring a few disadvantages: more board wires and more

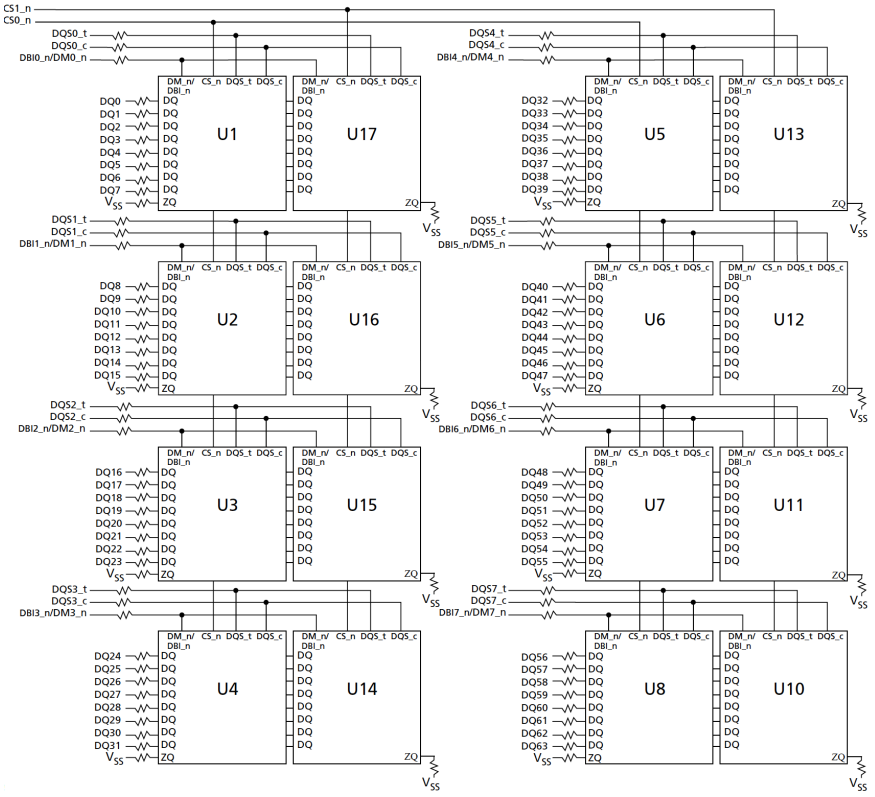


Fig. 1.62: 88-Pin Micron (1G x 64 bit) DDR4 SDRAM DIMM functional block diagram.

pins (on memory controller) are required. Multiple-channel architecture is a technology implemented on motherboards by the motherboard manufacturer and does not apply to memory modules. Also, a memory controller (which is a part of chipset) must support multiple-channel architecture. Theoretically, dual-channel configurations double the memory bandwidth when compared to single-channel configurations.

Figure 1.63 illustrates one channel formed from two ranks on the same DIMM module. Indeed, in multi-channel architectures, one channel is formed from at least one DIMM module. In today's desktop computers, up to two DIMM modules can be used to form one channel.

Most of today's computer systems support the dual channel configuration. Dual-channel-enabled memory controllers in a PC system architecture use two 64-bit data channels. For example, the Intel Core i7-800 series supported dual-channel configuration, as illustrated in Figure 1.64.

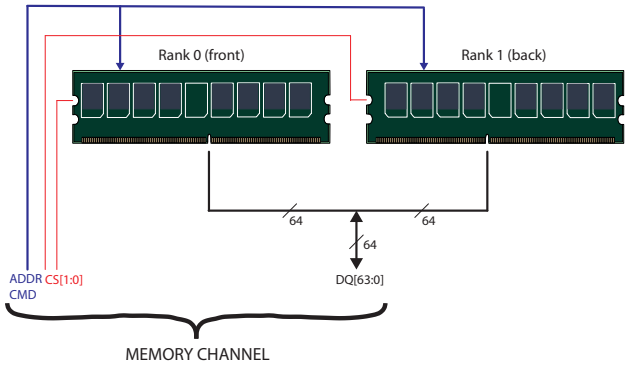


Fig. 1.63: A memory channel.

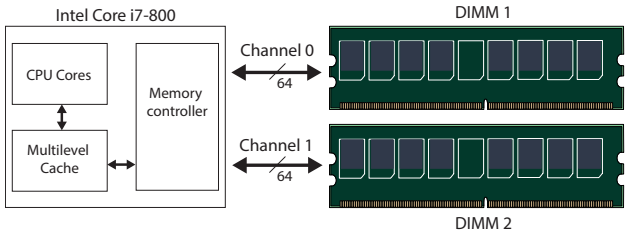


Fig. 1.64: A dual channel configuration supported by Intel Core i7-800. DIMM 1 and DIMM 2 should be identical in capacity, speed and CAS latency.

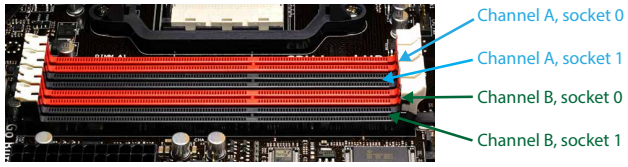


Fig. 1.65: Color codes of channels on PC motherboard.

Figure 1.65 shows a part of a motherboard that supports two memory channels. The motherboard has four DIMM sockets. To distinguish the channel's sockets on the motherboard, the sockets are color-coded. The motherboards use two colors. The colored pair of sockets is a dual channel set. A **matching pair of DIMMs** are two DIMMs that **are identical in capacity, speed, and CAS latency**. A matching pair should be used in both memory channels, i.e., a matching pair od DIMMs should be installed on the same color sockets. Another matching pair then goes in the remaining two sockets. Figure 1.66 shows two identical DIMM modules (a matching pair) inserted into the same-color sockets (red) forming two identical memory channels A

and B. Ideally, all DIMM modules should be identical in a system, or else we may end up with some memory being potentially downclocked to the lowest common denominator.

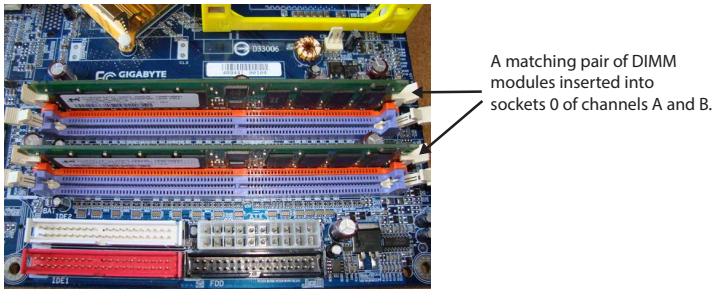


Fig. 1.66: A matching pair of DIMMs form two channels.

Intel Core i7-900 series DDR3 uses a triple-channel architecture, while modern high-end processors like the Intel Core i9 and AMD Ryzen Threadripper series support quad-channel memory. The quad-channel architecture can be used only when all four DIMM memory modules (or a multiple of four) are identical in capacity and speed and are placed in the same-color quad-channel sockets. When two DIMM memory modules are installed, the architecture will operate in a dual-channel mode; when three memory modules are installed, the architecture will operate in a triple-channel mode. On motherboards supporting quad-channel configuration, a similar color-coding scheme is used for dual-channel DIMM sockets. A same-color quadruple is a quad-channel set. A matching DIMM module quadruple (i.e., four DIMMs that are identical in capacity, speed, and CAS latency) should be installed on the same color sockets.

1.12.1 Case study: Intel i7-860 memory

At the beginning of the chapter, we have introduced the i7-860 and its memory hierarchy. This system is again illustrated in Figure 1.67. Now, we are going to describe the system with its real memory components and the case of an L3 miss.

The i7-860 supports up to two 64-bit memory channels, each consisting of a separate set of DDR3 1066/1333 DIMMs, and each of which can transfer in parallel. The i7-860 supports up to two DIMMs per channel and a total of up to 16 GB of memory.

In the case of L3 miss, both 64-bit memory channels are used simultaneously as one 128-bit channel (since there is only one memory controller, and the same address of the missing block in L3 is sent on both channels) to fill the missing block in L3. Using DDR3-1333 (DIMM PC3-10700), the i7-860 has a peak memory

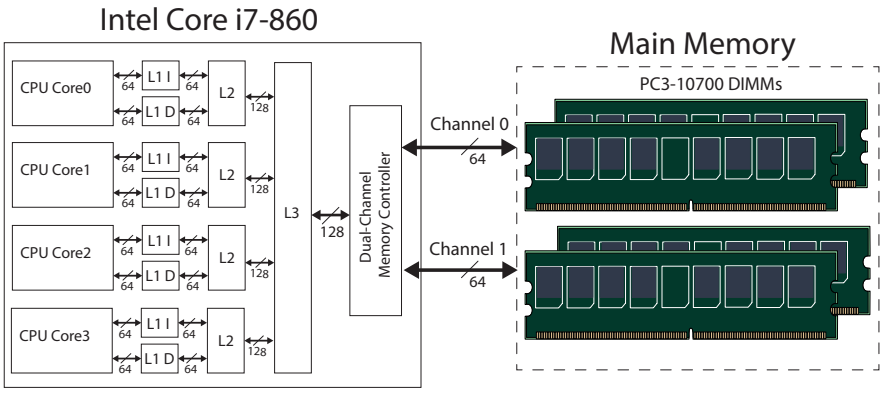


Fig. 1.67: Intel i7-860 memory.

bandwidth of just over 21 GB/sec. Thus, the memory controller fills the 64-byte cache block at a rate of 16 bytes (124 bits) per memory clock cycle.

If we assume the peak memory bandwidth, a 64-byte block is transferred at the rate of 21GB/s, which equals to 3 ns. Of course, we cannot assume that the missing block is transferred at the peak memory bandwidth. At best, we can assume that the row in SDRAMs, containing the missing block, is open. Thus, we have to add the CAS latency (CL), which equals 13.5 ns for DDR3-1333 chips. Thus, the missing block in L3 can be filled in 16.5 ns. The i7-860 runs at 2.8 GHz, which means that one CPU cycle equals 0.36 ns. Thus, the missing block in L3 will be available no prior than in 47 CPU cycles. In the case that the row containing the missing block is not open and all rows in that bank are precharged, we should add at least t_{RCD} to the above block access time. As t_{RCD} also equals 13.5 ns, the block is transferred in 29 ns or 81 CPU cycles. And finally, if we have to precharge a row before opening the row containing the missing block, the block will be transferred in 42.5 ns or 119 CPU cycles.

1.12.2 Case study: i9-9900K memory

Figure 1.68 illustrates the Intel i9-9900K system. Intel i7-9900K is an out-of-order execution processor that includes eight cores. The L1 and L2 caches are separate for each core, while the L3 cache is shared among the cores on a chip. The L1 cache is the 32 KB, eight-way set-associative cache. The L2 cache is the 256 KB, four-way set-associative cache. Finally, the L3 cache is the 16 MB, 16-way set-associative cache. The i9-9900K supports up to four 64-bit memory channels, each consisting of a separate set of DDR4-2666 DIMMs (PC4-21333), and each of which can transfer

in parallel, thus the peak memory bandwidth is 41.6 GB/s. The i9-9900K supports up to two DIMMs per channel and a total of up to 128 GB of memory.

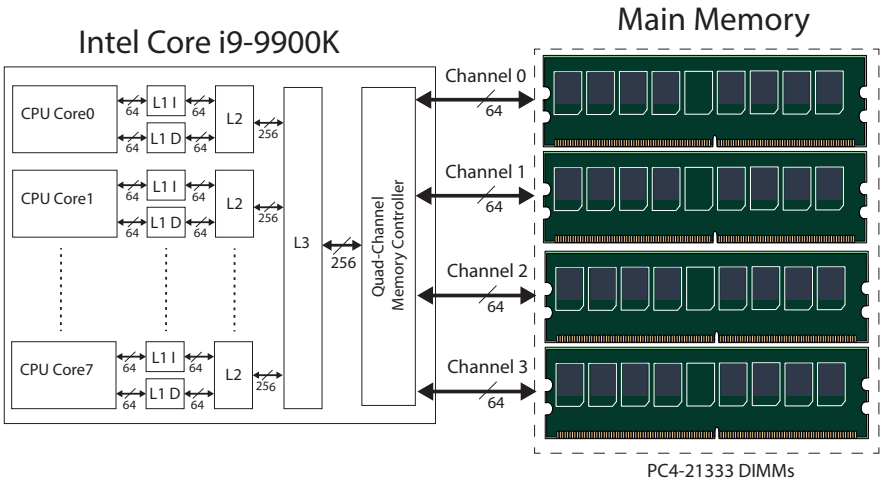


Fig. 1.68: Intel i9-9900K memory.

1.13 Bibliographical notes

TODO: The primary source of information including all details of DRAMs is the application note "Understanding DRAM Operation" [**?**] where basic asynchronous DRAM operation, including some of the most commonly used features for improving DRAM performance, is described.

the complete desreference guide is available.