Synchronous DRAM Architectures, Organizations, and Alternative Technologies

Prof. Bruce L. Jacob

Electrical & Computer Engineering Dept.
University of Maryland
College Park, MD 20742
http://www.ece.umd.edu/~blj/

December 10, 2002

1 DRAM TECHNOLOGY OVERVIEW

This section describes the structural organization of dynamic random-access memories (DRAMs), their operation, and the evolution of their design over time.

1.1 Basic Organization and Operation of a Conventional DRAM

DRAM is the "computer memory" that you order through the mail or purchase at Best Buy or CompUSA. It is what you put more of into your computer as an upgrade to improve the computer's performance. DRAM appears in personal computers (PCs) in the form shown in Figure 1—the figure shows a *memory module*, which is a small computer board ("printed circuit board") with a handful of chips attached to it. The eight black rectangles on the pictured module contain *DRAM chips*. Each DRAM chip contains one or more *memory arrays*, rectangular grids of storage cells with each cell holding one bit of data. Because the arrays are rectangular grids, it is useful to think of them in terms associated with typical grid-like structures—a good example of which is a Manhattan-like street layout with avenues running north-south and streets running east-west. When one wants to specify a rendezvous location in such a city, one simply designates the intersection of a street and an avenue, and the location is specified without ambiguity. Memory arrays are organized just like this, except whereas Manhattan is organized into *streets* and *avenues*, memory arrays are organized into *rows* and *columns*. A DRAM chip's memory array with the rows and columns indicated is pictured in Figure 2. By identifying the intersection of a row and a column, a computer's central processing unit (CPU) can access an individual storage cell inside a DRAM chip so as to read or write the data held there. This is accomplished by sending both a *row address* and a *column address* to the DRAM.

One way to characterize DRAMs is by the number of memory arrays inside them. Memory arrays within a memory chip can work in several different ways: they can act in unison, they can act completely independently, or they can act in a manner that is somewhere in between the other two. If the memory arrays are designed to act in unison, they operate as a unit, and the memory chip typically transmits or receives a number of bits equal to the number or arrays each time the memory controller accesses the DRAM. For example, in a simple organization, a x4 DRAM (pronounced "by four") indicates that the DRAM has at least four memory arrays and that a column width is 4 bits (each column read or write transmits 4 bits of data). Likewise, a x8 DRAM indicates that the DRAM has at least eight memory arrays and that a column width is 8 bits. Thus, in this x4 DRAM part, four arrays each read one data bit in unison, and the part sends out four bits of data each time the memory controller makes a column read request. If, on the other hand, the memory arrays inside the chip are completely independent, then they are typically referred to as "banks" and not "arrays." A memory controller can initialize[1] one bank while preparing to read data from another and even writing data to a third. Note that the two are not mutually exclusive: a DRAM can contain, for example, eight arrays organized into two x4 banks. A third mode of operation is the *interleaving* of independent banks; I will explain the concept of interleaving more

1. To "initialize" a bank is to make it ready for an upcoming read or write operation by *precharging* the columns in that bank to a specific voltage level.

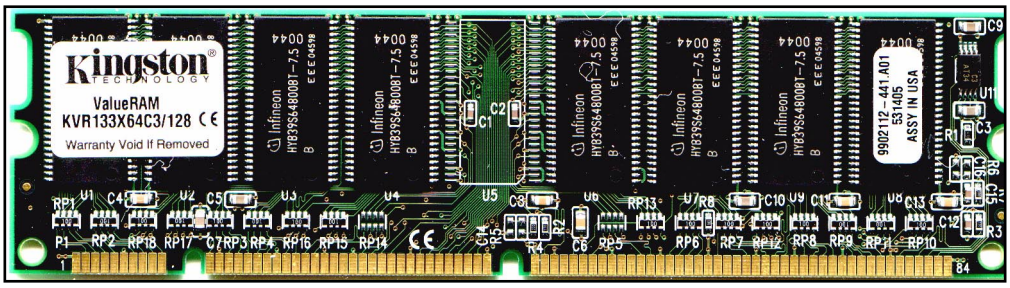


Figure 1: A Memory Module
A memory module is a computer board ("printed circuit board") with a handful of DRAM chips and associated circuitry attached to it. The picture is slightly larger than life-size.

Petitioner Lenovo (United States) Inc. - Ex. 1008

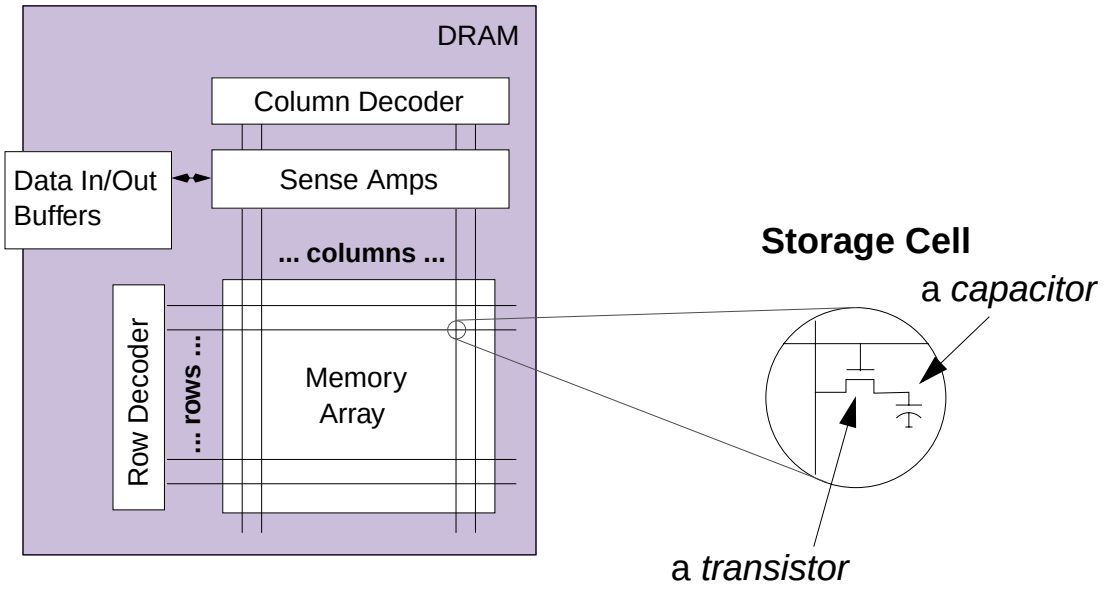


Figure 2: Basic Organization of DRAM Internals
The DRAM memory array is a grid of storage cells, where one bit of data is stored at each intersection of a *row* and a *column*.

fully in a later section on alternatives to dual edge clock technology, but for the present, using interleaving allows a x4 part to achieve the data bandwidth of a x8 part—that is, to double the data rate of the part (or triple, or quadruple, depending on the number of banks being interleaved). Interleaving multiple memory banks has been a popular method used to achieve fast memory busses using slow devices. The technique goes back at least to the mid-1960's [Anderson 1967; Thornton 1970].

The term "DRAM" stands for *dynamic random access memory*. It is characterized as "dynamic" primarily because the values held in the memory array's storage cells are represented by small electric charges that slowly leak out of the circuit over time—thus, the value held in a storage cell changes over time and is not static but dynamic. Because the electrical values represented in the storage cells dissipate over time, the values in the storage cells must be periodically *refreshed* to ensure valid data.

Figure 3 illustrates the DRAM's place in a typical PC. An individual DRAM device typically connects indirectly to the CPU through the north-bridge chipset. For purposes of this report, the primary component of the north-bridge chipset of interest is the memory controller, which serves as a liaison between the CPU and memory.

In the 1980's and 1990's, the conventional DRAM interface started to become a performance bottleneck in high-performance as well as desktop systems. The improvement in the speed and performance of CPUs was significantly outpacing the improvement in speed and performance of DRAM chips. As a consequence, the DRAM interface began to evolve, and a number of "revolutionary" proposals [Przybylski 1996] were made as well. In most cases, what was considered evolutionary or revolutionary was the proposed *interface*, or the mechanism by which the CPU accesses the DRAM. The DRAM core (i.e., what is pictured in Figure 2) remains essentially unchanged.

Every DRAM chip is equipped with *pins* (i.e., very short wires), each one of which connects the DRAM to one of many possible busses. Each bus is a group of wires that carry electrical signals; busses connect the CPU, memory controller, and DRAM chips. Pins are typically classified by the busses to which they connect; examples of different types of DRAM pins include address pins, data input and output pins, one or more clock input pins, and control pins (e.g., CAS and RAS input strobes[2], write-enable, chip-select, etc.).

The busses in a JEDEC-style organization are classified by their function and organization into data, address, control, and chip-select busses. There is a relatively wide data bus that transmits data to and from the DRAMs. This bus, in modern PC systems, is often 64 bits wide (64 bits equals eight bytes), and it can be much wider in high-performance systems. There is a dedicated address bus that carries row and column addresses to the DRAMs and has a width that grows with the physical storage on a DRAM device (typical widths today are about fifteen bits). A control bus is comprised of the row and column strobes, output enable, clock, clock enable, and other related signals. These signals are similar to the address bus signals in that they all connect from the memory controller to every DRAM in the system. Lastly, there is a chip-select network that connects from the memory controller to every DRAM in a *rank* (a separately addressable set of DRAMs). For example, a typical memory module (often called a "DIMM" for *dual in-line memory module*) can contain two ranks of DRAM devices. Thus, for every DIMM in the system there can be two separate chip-select networks, and thus the size of the chip-select "bus" scales with the maximum amount of physical memory in the system.

This last bus, the chip-select bus, is essential in a JEDEC-style memory system, as it enables the intended recipient of a memory request. A value is asserted on the chip-select bus at the time of a request (e.g., read or write). The chip-select bus contains a separate wire for every rank of DRAM in the system. The chip-select signal thus passes over a wire unique to each small set of DRAMs and enables or disables the DRAMs in that rank so that they, respectively, either handle the request currently on the bus or ignore the request currently on the bus. Thus, only the DRAMs to which the

2. A "strobe" is a signal that indicates to the recipient that another signal—e.g., data or command—is present and valid. An analogy would be to hold up one's hand while talking and lower it while silent. Another person can tell if you are talking (versus mumbling to yourself or talking in your sleep) by the status of your raised/lowered hand.

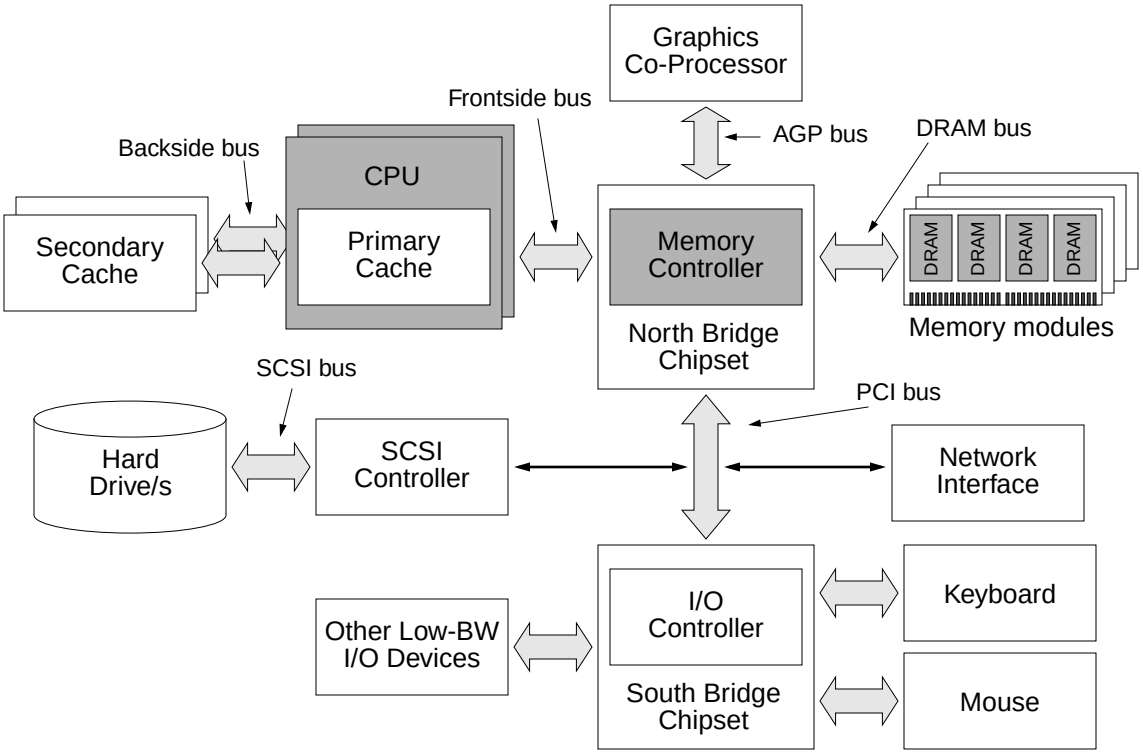
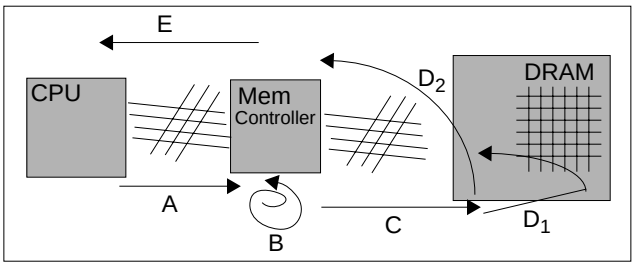


Figure 3: Typical PC Organization
The DRAM subsystem is one part of a relatively complex whole. This figure illustrates a 2-way multiprocessor, with each processor having its own dedicated secondary cache. The parts most relevant to this report are shaded in darker grey: the CPU, the memory controller, and the individual DRAMs.



A: Transaction request sent to Memory Controller
B: Transaction converted to Command Sequences
C: Command/s sent to DRAM
D_1: **RAS** — First command activates row (data driven into sense amps)
D_2: **CAS** — Later command/s send data from sense amps to memory controller
E: Transaction sent back to CPU

Figure 4: System Organization and the Steps Involved in a DRAM Read Operation
The CPU connects to the memory controller through some form of network or bus system, and the memory controller connects to the DRAM through another, usually different, network or bus system.

request is directed handle the request. Even though all DRAMs in the system are connected to the same address and control busses and could, in theory, all respond to the same request at the same time, the chip-select bus prevents this from happening.

Figure 4 focuses attention on the CPU, memory controller, and DRAM device and illustrates the steps involved in a DRAM request. The CPU connects to the memory controller through some form of network or bus system. The memory controller connects to the DRAM through some (other) form of network or bus system. The memory controller acts as a liaison between the CPU and DRAM, so that the CPU does not need to know the details of the DRAM's operation. The CPU presents requests to the memory controller that the memory controller then satisfies. The CPU connects to potentially many memory controllers at once; alternatively, many CPUs could

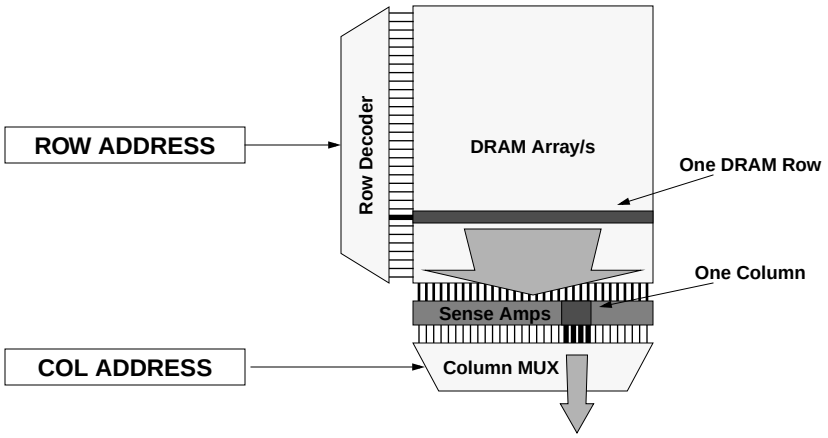


Figure 5: The Multi-Phase DRAM-Access Protocol
The row access drives a DRAM row into the sense amps. The column address drives a subset of the DRAM row onto the bus (e.g., 4 bits).

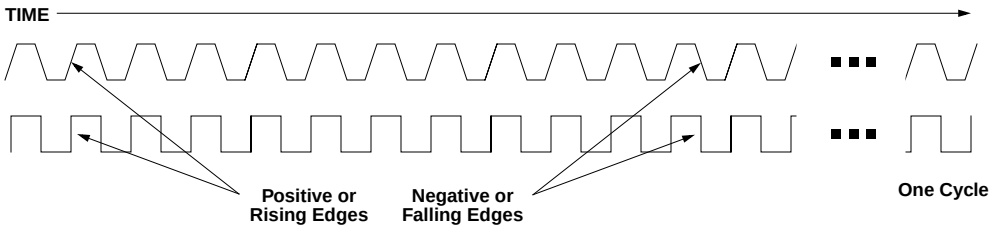


Figure 6: Example Clock Signals
Clocks are typically shown as square waves (bottom) or sort-of square waves (top). They repeat *ad infinitum* and the repeating shape is called a *clock cycle*. The two clocks pictured above have the same frequency—the number of cycles in a given time period.

be connected to the same memory controller. The simplest case, a uniprocessor system, is illustrated in the figure.

As mentioned, Figure 4 illustrates the steps of a typical DRAM read operation. The CPU sends a request (comprising a data address and a *read* command) over the bus that connects it to the memory controller. The memory controller is responsible for handling the RAS and CAS signals that, respectively, instruct the DRAM to *activate* or *open* a row and then to *read* a column within the activated row. The row activation and column read for a single DRAM device are shown in Figure 5.

The first step in handling a read request is for the memory controller to decompose the provided data address into components that identify the appropriate rank within the memory system, the bank within that rank[3], and the row and column inside the identified bank. The components identifying the row and column are called the *row address* and the *column address*. The bank identifier is typically one or more address bits. The rank number ends up causing a chip-select signal to be sent out over a single one of the separate chip-select lines.

Assuming the appropriate bank has been precharged, the second step is to activate the appropriate row inside the identified rank and bank, by setting the chip-select signal to activate the set of DRAMs comprising desired bank, sending the row address and bank identifier over the address bus, and signaling the DRAM's RAS pin (*row address strobe*). This tells the DRAM to send an entire row of data (thousands of bits) into the DRAM's sense amplifiers (circuits that detect and amplify the tiny logic signals represented by the electric charges in the row's storage cells).

The third step is to read the column bits, a subset of the data within the row, by setting the chip-select signal to activate the set of DRAMs comprising the desired bank[4], sending the column address and bank identifier over the address bus, and signaling the DRAM's CAS pin (*column address strobe*). This causes only a few select bits[5] in the sense amplifiers to be connected to the output drivers, where they will be driven onto the data bus and eventually will travel back to the CPU.

Most computer systems have a special signal that acts much like a heartbeat and is called the *clock*. A clock transmits a continuous signal with regular intervals of "high" and "low" values. It is usually illustrated as a square wave or semi-square wave with each period identical to the next, as shown in Figure 6. The upward portion of the square wave is called the *positive* or *rising edge* of the clock, and the downward portion of the square wave is called the *negative* or *falling edge* of the clock. The primary clock in a computer system is called the system clock or global clock, and it typically resides on the motherboard (the printed circuit board that contains the CPU and memory bus). The system clock drives the CPU and memory con-

3. The number of banks within a rank is usually equal to the number of banks within a single DRAM device. In the case of SDRAM, for example, there are two banks in a DRAM device, and so a bank identifier is a single bit sent over the address bus at the time of a command.

4. This step is necessary for SDRAMs; it is not performed for older, asynchronous DRAMs (it is subsumed by the earlier chip-select accompanying RAS).

5. One bit in a "x1" DRAM, two bits in a "x2" DRAM, four bits in a "x4" DRAM, etc.

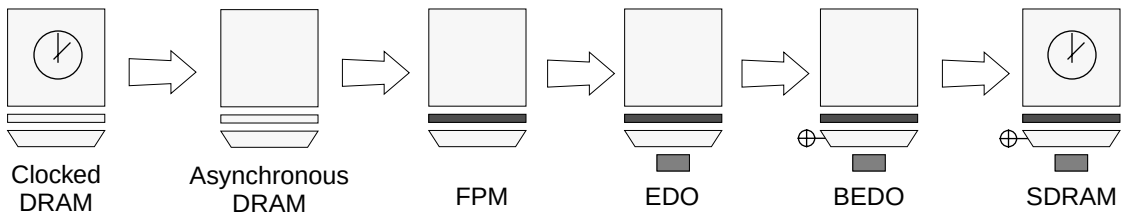


Figure 7: Evolution of the DRAM Architecture
Each step along DRAM's evolution has been incremental. Original designs were clocked; in the mid-1970's the clock disappeared; fast page mode (FPM) kept the sense amplifiers active; extended data-out (EDO) added a latch; burst EDO (BEDO) added an internal counter; SDRAM came full circle by reinstating a clock signal.

troller and many of the associated peripheral devices directly. If the clock drives the DRAMs directly, the DRAMs are called *synchronous DRAMs*. If the clock does not drive the DRAMs directly, the DRAMs are called *asynchronous DRAMs*. In a synchronous DRAM, operative steps internal to the DRAM happen in time with one or more edges of this clock. In an asynchronous DRAM, operative steps internal to the DRAM happen when the memory controller commands the DRAM to act, and those commands typically happen in time with one or more edges of the system clock.

1.2 Evolution of DRAM Technology

Since DRAM's inception, there have been numerous changes to the design. Figure 7 shows the evolution of the basic DRAM architecture from *clocked* to *asynchronous* to *fast page mode* (FPM) to *extended data-out* (EDO) to *burst-mode EDO* (BEDO) to *synchronous* (SDRAM). The changes have largely been structural in nature, have been relatively minor in terms of their implementation cost and have increased DRAM throughput significantly.

Compared to the asynchronous DRAM, FPM simply allows the row to remain open across multiple $\overline{\text{CAS}}$ commands, requiring very little additional circuitry. To this, EDO changes the output drivers to become output latches so that they hold the data valid on the bus for a longer period of time. To this, BEDO adds an internal counter that drives the address latch, so that the memory controller need not supply a new address to the DRAM on every $\overline{\text{CAS}}$ command if the desired address is simply one-off from the previous $\overline{\text{CAS}}$ command. Thus, in BEDO, the DRAM's column-select circuitry is driven from an internally generated signal, not an externally generated signal: the source of the control signal is close to the circuitry that it controls in space and therefore time, and this makes the timing of the circuit's activation more precise. Lastly, SDRAM takes this perspective one step further and drives all internal circuitry (row select, column select, data read-out) by a clock, as opposed to the $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$ strobes. The following paragraphs describe this evolution in more detail.

1.2.1 Clocked DRAM

The earliest DRAMs (1960's to mid 1970's, before standardization) were often clocked [Rhoden 2002, Sussman 2002, Padgett 1974]; DRAM commands were driven by a periodic clock signal. Figure 7 shows a stylized DRAM in terms of the memory array, the sense amplifiers, and the column multiplexer.

1.2.2 Asynchronous DRAM

In the mid-1970's, DRAMs moved to the asynchronous design with which most people are familiar. These DRAMs, like the clocked versions before them, require that every single access go through all of the steps described above. Even if the CPU wants to request the same data row that it previously requested, the entire process (*row activation* followed by *column read/write*) must be repeated. Figure 8 illustrates the timing for the asynchronous DRAM.

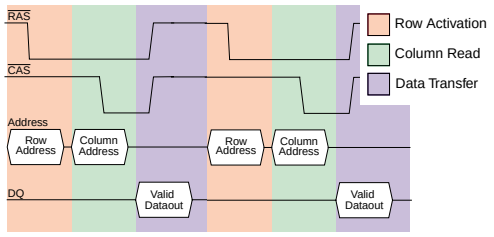


Figure 8: Read Timing for Conventional DRAM

1.2.3 Fast Page Mode DRAM (FPM DRAM)

Fast Page Mode is a version of asynchronous DRAM that permits the selected row to be held constant in the sense amplifiers over multiple column requests. The data from more than one column in the row can be accessed by having the memory controller send several column requests over the bus to the DRAM chip, each accompanied by a separate CAS pulse instructing the DRAM to read or write the corresponding column. Fast page mode improves performance by taking advantage of the fact the CPU in a computer system is statistically likely to want to access more than one bit from the same row. Fast page mode, therefore, improves performance by saving the time it would take to re-activate the row every time the CPU wants multiple data bits within the same row. Figure 9 gives the timing for FPM reads.

1.2.4 Extended Data Out DRAM (EDO DRAM)

The next generation of asynchronous DRAM to be produced was EDO DRAM. Extended Data Out DRAM adds a latch between the sense amplifiers and the output pins of the DRAM. This latch allows the data on the output drivers of the DRAM circuit to remain valid longer into the next clock phase (thus the name "extended data-out"). By permitting the memory array to begin precharging sooner, the addition of a latch allowed EDO DRAM to operate faster than FPM DRAM. EDO enabled the CPU to access memory at least 10 to 15% faster than with FPM [Kingston 2000, Cuppu et al. 1999, Cuppu et al. 2001]. Figure 10 gives the timing for an EDO read.

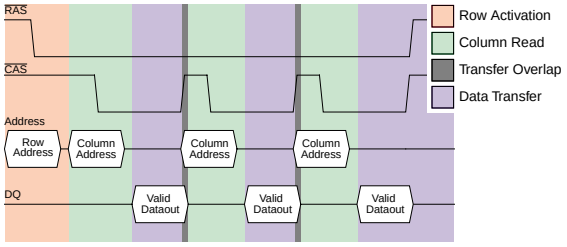


Figure 9: FPM Read Timing
Fast page mode allows the DRAM controller to hold a row constant and receive multiple columns in rapid succession.

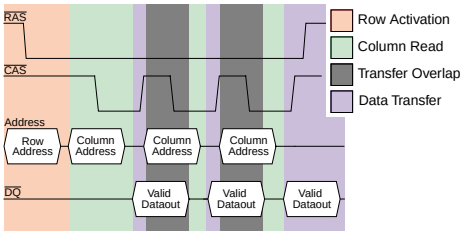


Figure 10: EDO Read Timing
The output latch in EDO DRAM allows more overlap between column access and data transfer than in FPM.

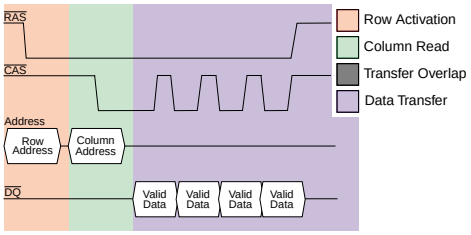


Figure 11: Burst EDO Read Timing
By driving the column-address latch from an internal counter rather than an external signal, the minimum cycle time for driving the output bus was reduced by roughly 30% over EDO.

1.2.5 Burst-Mode EDO DRAM (BEDO DRAM)

Although Burst EDO DRAM never reached the volume of production that EDO and SDRAM did, it was positioned to be the next generation DRAM after EDO [Micron 1995]. Burst EDO builds on EDO DRAM by adding the concept of "bursting" contiguous blocks of data from an activated row each time a new column address is sent to the DRAM chip. The memory controller toggles the CAS strobe by sending alternating high and low signals. With each toggle of the CAS strobe, the DRAM chip sends the next sequential bit of data onto the bus. By eliminating the need to send successive column addresses over the bus to drive a burst of data in response to each CPU request, Burst EDO eliminates a significant amount of timing uncertainty between successive addresses, thereby increasing the rate at which data can be read from the DRAM. In practice, BEDO reduced the minimum cycle time for driving the output bus by roughly 30% compared to EDO DRAM [Prince 2000], thereby increasing bandwidth proportionally. Figure 11 gives the timing for a Burst EDO read.

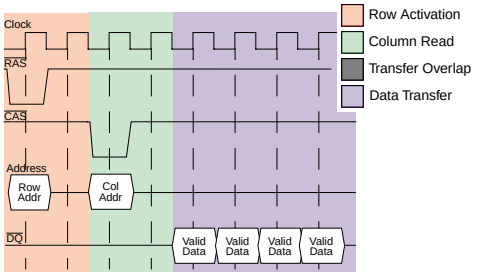


Figure 12: SDR SDRAM Read Operation Clock Diagram (CAS-2)

1.2.6 IBM's High-Speed Toggle Mode DRAM

IBM's High-Speed Toggle Mode ("toggle mode") is a high-speed DRAM interface designed and fabricated in the late 1980's and presented at the International Solid-State Circuits Conference in February 1990 [Kalter 1990a]. In September 1990, IBM presented toggle mode to JEDEC as an option for the next-generation DRAM architecture [minutes of JC-42.3 meeting 55, Jedec 13674–13676]. Toggle mode transmits data to and from a DRAM on both edges of a high-speed data strobe rather than transferring data on a single edge of the strobe. The strobe was very high speed for its day: Kalter reports a 10ns data cycle time—an effective 100MHz data rate—in 1990 [Kalter 1990b]. The term "toggle" is probably derived from its implementation: to obtain twice the normal data rate[6], one would toggle a signal pin which would cause the DRAM to toggle back and forth between two different (interleaved) output buffers, each of which would be pumping data out at half the speed of the strobe [Kalter 1990b]. As proposed to JEDEC, it offered burst lengths of 4 or 8 bits of data per memory access.

1.2.7 Synchronous DRAM (SDRAM)

Conventional, FPM, and EDO DRAM are controlled *asynchronously* by the memory controller; therefore, in theory the memory latency and data toggle rate can be some fractional number of CPU clock cycles[7]. More importantly, what makes the DRAM *asynchronous* is that the memory controller's $\overline{RAS}$ and $\overline{CAS}$ signals directly control latches internal to the DRAM, and those signals can arrive at the DRAM's pins at any time. An alternative is to make the DRAM interface *synchronous* such that requests can only arrive at regular intervals. This allows the latches internal to the DRAM to be controlled by an internal clock signal. The primary benefit of SDRAM, or synchronizing DRAM operation with the CPU clock, is that it improves the predictability of event timing—put simply, events such as the arrival of commands or the driving of output data either happen in time with the clock, or they do not happen. A timing diagram for synchronous DRAM is shown in Figure 12. Like BEDO DRAMs, SDRAMs support the concept of a burst mode; SDRAM devices have a programmable register that holds a burst length. The DRAM uses this to determine how many columns to output over successive cycles—SDRAM may therefore return many bytes over several cycles per request. One advantage of this is the

6. The term "normal" implies the data cycling at half the data-strobe rate.

7. In practice, this is not the case, as the memory controller and DRAM subsystem are driven by the system clock, which typically has a period that is an integral multiple of the CPU's clock.

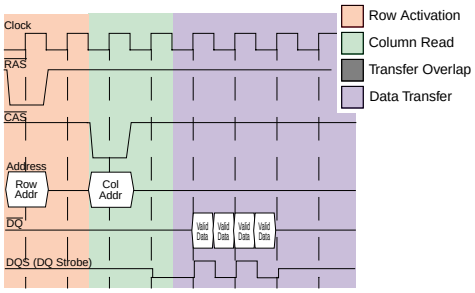


Figure 13: DDR SDRAM Read Timing (CAS-2)
DDR SDRAMs use both a clock and a source-synchronous data strobe (DQS) to achieve high data rates. DQS is used by the DRAM to sample incoming write data; it is typically ignored by the memory controller on DRAM reads.

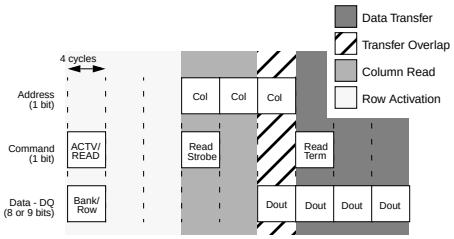


Figure 14: Concurrent RDRAM Read Operation
Rambus DRAMs transfer on both edges of a fast clock and use a 1-byte data bus multiplexed between data and addresses.

elimination of the timing signals (i.e., toggling $\overline{CAS}$) for each successive burst, which reduces the command bandwidth used. The underlying architecture of the SDRAM core is the same as in a conventional DRAM.

1.3 Contemporary DRAM Architectures

Since the appearance of SDRAM in the mid-1990's, there has been a large profusion of novel DRAM architectures proposed in an apparent attempt by DRAM manufacturers to make DRAM less of a commodity [Dipert 2000]. One reason for the profusion of competing designs is that we have apparently run out of the same sort of "free" ideas that drove earlier DRAM evolution. Since Burst EDO, there has been no architecture proposed that provides a 30% performance advantage at near-zero cost; all proposals have been relatively expensive. As Dipert suggests, there is no clear heads-above-the-rest winer yet because many schemes seem to lie along a linear relationship between additional cost of implementation and realized performance gain. Over time, the market will most likely decide the winner; those DRAM proposals that provide sub-linear performance gains relative to their implementation cost will be relegated to zero or near-zero market share.

1.3.1 Double Data Rate SDRAM (DDR SDRAM)

Double data rate (DDR) SDRAM is the modern equivalent of IBM's High-Speed Toggle Mode. DDR doubles the data bandwidth available from single data rate SDRAM by transferring data at both edges of the clock (i.e., both the rising edge and the falling edge), much like toggle mode's dual-edged clocking scheme. DDR DRAM are very similar to single data rate SDRAM in all other characteristics. They use the same signalling technology, the same interface specification, and the same pinouts on the DIMM carriers. However, DDR-DRAM's internal transfers from and to the DRAM array respectively read and write twice the number of bits as SDRAM. Figure 15 gives a timing diagram for a CAS-2 read operation.

1.3.2 Rambus DRAM (RDRAM, Concurrent RDRAM, and Direct RDRAM)

Rambus DRAM (RDRAM) is very different from traditional main memory. It uses a bus that is significantly narrower than the traditional bus, and, at least in its initial incarnation, it does not use dedicated address, control, data, and chip-select portions of the bus—instead, the bus is fully multiplexed, which means that address, control, data, and chip-select information all travel over the same set of electrical wires but at different times. The bus is one byte wide, runs at 250 Mhz, and transfers data on both clock edges to achieve a theoretical peak bandwidth of 500 Mbytes/s. Transactions occur on the bus using a split request/response protocol. Because the bus is multiplexed between address and data, only one transaction may use the bus during any 4 clock cycle period, referred to as an *octcycle*. The protocol uses packet transactions; first an address/control packet is driven, then the data. Different transactions can require different numbers of octcycles, depending on the transaction type, location of the data within the device, number of devices on the channel, etc.

Because of the bus's design—being a single bus and not comprised of separate segments dedicated to separate functions—only one transaction can use the bus during any given cycle; this limits the bus's potential *concurrency*, its ability to do multiple things simultaneously. Due to this limitation, the original RDRAM design was not considered well suited to the PC main-memory market [Przybylski 1996], and the interface was redesigned in the mid-1990's to support more concurrency. Specifically, with the introduction of "Concurrent RDRAM," the bus was divided into separate address, command, and data segments reminiscent of a JEDEC-style DRAM organization. The data segment of the bus remained one byte wide, and to this was added a one-bit address segment and a one-bit control segment. By having three separate, dedicated segments of the bus, one could perform potentially three separate, simultaneous actions on the bus. This divided & dedicated arrangement simplified transaction scheduling and increased performance over RDRAM accordingly. Figure 14 gives a timing diagram for a read transaction.

One of the few limitations to the "Concurrent" design was that the data bus sometimes carried a brief packet of address information, because the one-bit address bit was too narrow. This limitation has been removed in Rambus's latest DRAMs. The divided arrangement introduced in Concurrent RDRAM has been carried over into the most recent incarnation of RDRAM, called "Direct RDRAM," which increases the width of the data segment to two bytes, the width of the address segment to five bits, and the width of the control segment to three bits. These segments remain separate and dedicated—similar to a JEDEC-style organization—and the control and address segments are wide enough that the data segment of the bus never needs to carry anything but data, thereby increasing data throughput on the channel. Bus operating speeds have also changed over the years, and latest designs are roughly double the original speeds (500MHz bus frequency). Each half-row buffer in Direct RDRAM is shared between adjacent banks, which implies that adjacent banks cannot be active simultaneously. This organization has the result of increasing the row-buffer miss rate as compared to having one

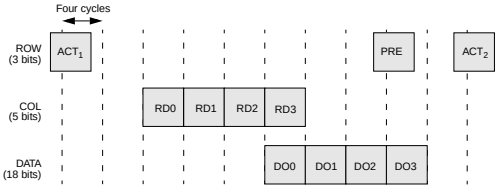


Figure 15: Direct Rambus Read Clock Diagram
Direct Rambus DRAMs transfer on both edges of a fast clock and use a 2-byte data bus dedicated to handling data only.

open row per bank, but it reduces the cost by reducing the die area occupied by the row buffers, compared to 16 full row buffers. Figure 15 gives a timing diagram for a read operation.

1.3.3 Virtual Channel Memory (VCDRAM)

Virtual Channel adds a substantial SRAM cache to the DRAM that is used to buffer large blocks of data (called *segments*) that might be needed in the future. The SRAM segment cache is managed explicitly by the memory controller. The design adds a new step in the DRAM access protocol: a row activate operation moves a page of data into the sense amps; "prefetch" and "restore" operations (data-read and data-write, respectively) move data between the sense amps and the SRAM segment cache one segment at a time; and column read or write operations move a column of data between the segment cache and the output buffers. The extra step adds latency to read and write operations, unless all of the data required by the application fits in the SRAM segment cache.

1.3.4 Enhanced SDRAM (ESDRAM)

Like EDO DRAM, ESDRAM adds an SRAM latch to the DRAM core, but whereas EDO added the latch after the column mux, ESDRAM adds it *before* the column mux. Therefore the latch is as wide as a DRAM page. Though expensive, the scheme allows for better overlap of activity: for instance, it allows row precharge to begin immediately without having to close out the row (it is still active in the SRAM latch). In addition, the scheme allows a write-around mechanism whereby an incoming write can proceed without the need to close out the currently active row. Such a feature is useful for writeback caches, where the data being written at any given time is not likely to be in the same row as data that is currently being read from the DRAM. Therefore, handling such a write delays future reads to the same row. In ESDRAM, future reads to the same row are not delayed.

1.3.5 MoSys 1T-SRAM

MoSys, i.e. Monolithic System Technology, has created a "1-transistor SRAM" (which is not really possible, but it makes for a catchy name). Their design wraps an SRAM interface around an extremely fast DRAM core to create an SRAM-compatible part that approaches the storage and power consumption characteristics of a DRAM while simultaneously approaching the access-time characteristics of an SRAM. The fast DRAM core is made up of a very large number of independent banks: decreasing the size of a bank makes its access time faster, but increasing the number of banks complicates the control circuitry (and therefore cost) and decreases the part's effective density. No other DRAM manufacturer has gone to the same extremes as MoSys to create a fast core, and thus the MoSys DRAM is the lowest-latency DRAM in existence. However, its density is low enough that OEMs have not yet used it in desktop systems in any significant volume. Its niche is high-speed embedded systems and game systems (e.g. Nintendo GameCube).

1.3.6 Reduced Latency DRAM (RLDRAM)

RLDRAM is a fast DRAM core that has no DIMM specification: it must be used in a direct-to-memory-controller environment (e.g. inhabiting a dedicated space on the motherboard). Its manufacturers suggest its use as an extremely large off-chip cache, probably at a lower spot in the memory hierarchy than any SRAM cache. Interfacing directly to the chip, as opposed to through a DIMM, decreases the potential for clock skew, thus the part's high speed interface.

1.3.7 Fast Cycle DRAM (FCRAM)

Fujitsu's FCRAM achieves a low-latency data access by segmenting the data array into sub-arrays, only one of which is driven during a row activate. This is similar to decreasing the size of an array, thus its effect on access time. The subset of the data array is specified by adding more bits to the row address, and therefore the mechanism is essentially putting part of the column address into the row activation (e.g. moving part of the column-select function into row activation). As opposed to RLDRAM, the part does have a DIMM specification, and it has the highest DIMM bandwidth available, in the DRAM parts surveyed.

The evolutionary changes made to the DRAM interface up to and including Burst EDO have been relatively inexpensive to implement, while the improvement in performance was significant: FPM was essentially free compared to the conventional design (required only a protocol change), EDO simply added a latch, BEDO added a counter and multiplexer. Each of these evolutionary changes added only a small amount of logic, yet each improved upon its predecessor by as much as 30% in terms of system performance [Cuppu et al. 1999, Cuppu et al. 2001]. Though SDRAM represented a more significant cost in implementation and offered no performance improvement over Burst EDO at the same clock speeds[8], the use of a strobe synchronized with the command and data signals (in this case, the strobe is the global clock signal) would allow the SDRAM protocol to be extended to higher switching speeds and thus higher data rates more easily than the earlier asynchronous DRAM interfaces such as fast page mode and EDO. Note that this benefit applies to any interface with a similar strobe signal, whether the interface is synchronous or asynchronous, and therefore an asynchronous burst-mode DRAM with this type of strobe could have scaled to higher switching speeds just as easily as SDRAM [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002]—witness the February 1990 presentation of a working 100MHz asynchronous burst-mode part from IBM, which used a dedicated pin to transfer the source-synchronous data strobe [Kalter 1990a/b].

Latter day advanced DRAM designs have abounded, largely because of the opportunity to appeal to markets asking for high performance [Dipert 2000] and because engineers have evidently run out of design ideas that echo those of the early-on evolution—that is, design ideas that are relatively inexpensive to implement and yet yield tremendous performance advantages. The DRAM industry tends to favor building the simplest design that achieves the desired benefits [Lee 2002, DuPreez 2002, Rhoden 2002]; currently, the dominant DRAM in the high-performance design arena is DDR.

8. By some estimates, Burst EDO actually had higher performance than SDRAM [Williams 2001].

DRAM Type	Data Bus Speed	Bus Width (per chip)	Peak BW (per Chip)	Peak BW (per DIMM)[a]	Latency (t_{RAC})
1T-SRAM	200	32	**800 MB/s**	*800 MB/s*	**10 ns**
PC133 SDRAM	133	16	266 MB/s	1.1 GB/s	30 ns
VCDRAM	133	16	266 MB/s	1.1 GB/s	45 ns
ESDRAM	166	16	332 MB/s	1.3 GB/s	**24 ns**
DRDRAM	533 * 2	16	**2.1 GB/s**	2.1 GB/s	40 ns
RLDRAM	300 * 2	32	**2.4 GB/s**	***2.4 GB/s***	25 ns
DDR 333	166 * 2	16	666 MB/s	**2.7 GB/s**	33 ns
FCRAM	200 * 2	16	**800 MB/s**	**3.2 GB/s**	**22 ns**

a. Data for DRAMs that are typically not found in DIMM organizations are italicized.

1.4 Comparison of Recent DRAMs

In the table above, some salient characteristics of a selection of today's DRAM architectures are presented, ordered by peak bandwidth coming off the DIMM (which, for high-performance server systems, is the most significant figure of merit). The "best" three DRAM architectures for each category are highlighted in bold type. Note that in the category of peak bandwidth per-chip there is a tie for third place. Note also that some DRAMs are not typically found in DIMM configurations and therefore have the same per-DIMM bandwidth as their per-chip bandwidth.

2 SALIENT FEATURES OF JEDEC'S SDRAM TECHNOLOGY

JEDEC SDRAMs use the traditional DRAM-system organization, described earlier and illustrated in Figure 16. There are four different busses, each classified by its function—a "memory bus" in this organization is actually comprised of separate (1) data, (2) address, (3) control, and (4) chip-select busses. Each of these busses is dedicated to handle only its designated function, except in a few instances—for example, when control information is sent over an otherwise unused address-bus wire. (1) The data bus is relatively wide: in modern PC systems, it is 64 bits wide, and it can be much wider in high-performance systems. (2) The width of the address bus grows with the number of bits stored in an individual DRAM device; typical address busses today are about 15 bits wide. (3) A control bus is comprised of the row and column strobes, output enable, clock, clock enable, and other similar signals that connect from the memory controller to every DRAM in the system. (4) Lastly, there is a chip-select network that uses one unique wire per DRAM rank in the system and thus scales with the maximum amount of physical memory in the system. Chip-select is used to enable ranks of DRAMs and thereby allow them to read commands off the bus and read/write data off/onto the bus.

The primary difference between SDRAMs and earlier asynchronous DRAMs is the presence in the system of a clock signal against which all actions (command and data transmissions) are timed. Whereas asynchronous DRAMs use the RAS and CAS signals as strobes—that is, the strobes directly cause the DRAM to sample addresses and/or data off the bus—SDRAMs instead use the clock as a strobe, and the RAS and CAS signals are simply commands that are themselves sampled off the bus in time with the clock strobe. The reason for timing transmissions with a regular (i.e., periodic) free-running clock instead of the less regular RAS and CAS strobes was to achieve higher dates more easily: when a regular strobe is used to time transmissions, timing uncertainties can be reduced, and therefore data rates can be increased.

Note that any regular timing signal could be used to achieve higher data rates in this way; a free-running clock is not necessary [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002]. In DDR SDRAMs, the clock is all but ignored in the data-transfer portion of write requests: the DRAM samples the incoming data with respect to not the clock but instead a separate, regular signal known as DQS [Lee 2002, Rhoden 2002, Macri 2002, Karabotsos 2002, Baker 2002]. The implication is that a free-running clock could be dispensed with entirely, and the result would be something very close to IBM's toggle-mode.

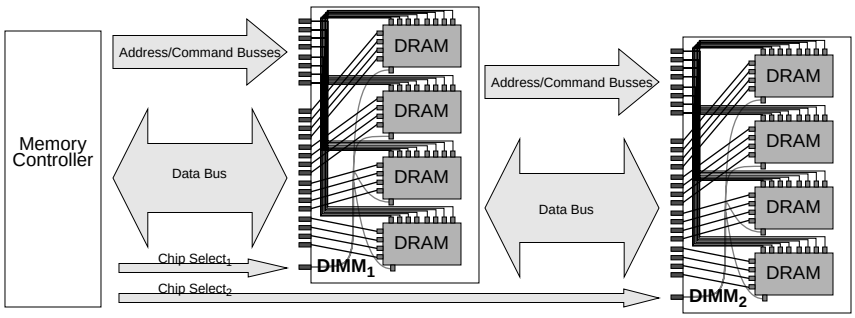


Figure 16: JEDEC-Style Memory Bus Organization
The figure shows a system of a memory controller and two memory modules with a 16-bit data bus and an 8-bit address and command bus.

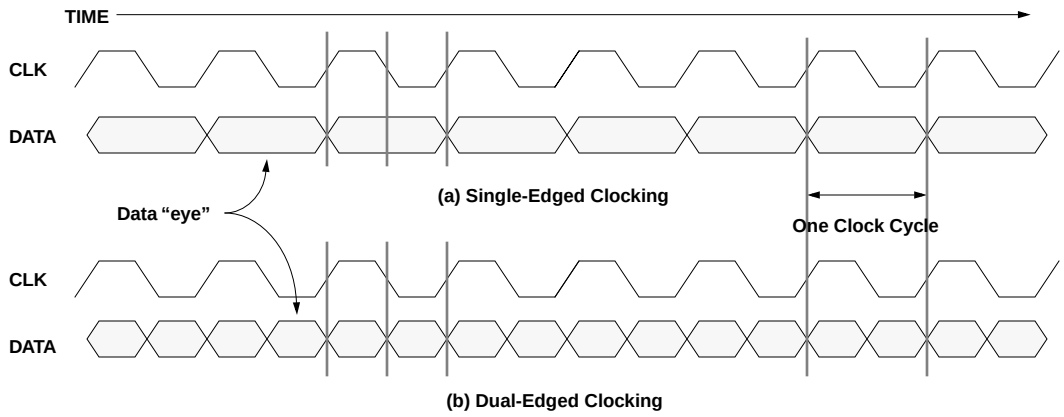


Figure 17: Running the Bus Clock at the Data Rate
The top diagram (a) illustrates a single-edged clocking scheme wherein the clock is twice the frequency of data transmission. The bottom diagram (b) illustrates a dual-edged clocking scheme in which the data transmission rate is equal to the clock frequency.

2.1 Single Data Rate SDRAM

Single data rate SDRAM use a *single-edged clock* to synchronize all information—that is, all transmissions on the various busses (control, address, data) begin in time with one edge of the system clock (as so happens, the rising edge). Because the transmissions on the various busses are ideally valid from one clock edge to the next, those signals are very likely to be valid during the other edge of the clock (the falling edge); consequently, that edge of the clock can be used to sample those signals.

SDRAMs have several features that were not present in earlier DRAM architectures: a burst length that is programmable and a CAS latency that is programmable.

2.1.1 Programmable Burst Length

Like BEDO DRAMs, SDRAMs use the concept of *bursting* data to improve bandwidth. Instead of using successive toggling of the CAS signal to burst out data, however, SDRAM chips only require CAS to be signalled once and, in response, transmit or receive in time with the toggling of the clock the number of bits indicated by a value held in a programmable mode register. Once an SDRAM receives a row address and a column address, it will burst the number of columns that correspond to the burst length value stored in the register. If the mode register is programmed for a burst length of four, for example, then the DRAM will automatically burst four columns of contiguous data onto the bus. This eliminates the need to toggle the CAS strobe to derive a burst of data in response to a CPU request. Consequently, the potential parallelism in the memory system increases (i.e., it improves) due to the reduced use of the command bus—the memory controller can issue other requests to other banks during those cycles that it otherwise would have been toggling CAS.

2.1.2 Programmable CAS Latency

The mode register also stores the CAS latency of an SDRAM chip. Latency is a measure of delay. CAS latency, as the name implies, refers to the number of clock cycles it takes for the SDRAM to return the data once it receives a CAS command. The ability to set the CAS latency to a desired value allows parts of different generations and fabricated in different process technologies (which would all otherwise have different performance characteristics) to behave identically. Thus, mixed-performance parts can easily be used in the same system and can even be integrated onto the same memory module.

2.2 Double Data Rate SDRAM

DDR SDRAM have several features that were not present in single data rate SDRAM architectures: dual-edged clocking and an on-chip delay-locked loop (DLL).

2.2.1 Dual-Edged Clocking

Double data rate SDRAMs, like regular SDRAMs, use a single-edged clock to synchronize control and address transmissions, but for data transmissions DDR DRAMs use a *dual-edged clock*—that is, some data bits are transmitted on the data bus in time with the rising edge of the system clock, and other bits are transmitted on the data bus in time with the falling edge of the system clock.

Figure 17 illustrates the difference, showing timing for two different clock arrangements. The top design is a more traditional arrangement wherein data is transferred only on the rising edge of the clock; the bottom uses a data rate that is twice the speed of the top design, and data is transferred on both the rising and falling edges of the clock. Such an arrangement was well known to engineers at the time. IBM had built DRAMs using this feature in the late 1980's and presented their results in the International Solid State Circuits Convention in February of 1990 [Kalter 1990a]. Reportedly, Digital Equipment Corp. had been experimenting with similar schemes in the late 1980's and early 1990's [Lee 2002, minutes of JC-42.3 meeting 58 Jedec 13958].

In a system that uses a single-edged clock to transfer data, there are two clock edges for every data "eye;" the data eye is framed on both ends by a clock edge, and a third clock edge is found somewhere in the middle of the data transmission (cf. Figure 17(a)). Thus, the clock signal can be used directly to perform two actions: to drive data onto the bus and to read data off the bus. Note that in a single-edged clocking scheme data is transmitted once per clock cycle.

By contrast, in a dual-edged clocking scheme, data is transmitted twice per clock cycle. This halves the number of clock edges available to drive data onto the bus and/or read data off the bus (cf. Figure 17(b)). The bottom diagram shows a clock running at the same rate as data transmission. Note that there is only one clock edge for every data "eye." The clock edges in a dual-edged scheme are either "edge-aligned" with the data or "center-aligned" with the data—this means that the clock can either drive the data onto the bus, or it can read the data off the bus, but it cannot do both, as it can in a single-edged scheme. In the figure, the clock is edge-aligned with the data.

The dual-edged clocking scheme by definition has fewer clock edges per data transmission that can be used to synchronize or perform functions. This means that some other mechanism must be introduced to get accurate timing for both driving data and sampling data—i.e., to compensate for the fact that there are fewer clock edges, a dual-edged signalling scheme needs an additional mechanism beyond the clock. For example, DDR SDRAM specifies along with the system clock a center-aligned data strobe that is provided by the memory controller on DRAM writes that the DRAM uses directly to sample incoming data. On DRAM reads, the data strobe is edge-aligned with the data and system clock; the memory controller is responsible for providing its own mechanism for generating a center-aligned edge. The strobe is called "DQS."

2.2.2 On-Chip Delay-Locked Loop

In DDR SDRAM, the on-chip DLL synchronizes the DRAM's outgoing data and DQS (data strobe) signals with the memory controller's global clock [JEDEC Standard 21-C, Section 3.11.6.6]. The DLL synchronizes those signals involved in DRAM reads, not those involved in DRAM writes; in the latter case, the DQS signal accompanying data sent from the memory controller on DRAM writes is synchronized with that data by the memory controller and is used by the DRAM directly to sample the data [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002]. The DLL circuit in a DDR DRAM thus ensures that data is transmitted by the DRAM in synchronization with the memory controller's clock signal so that the data arrives at the memory controller at expected times. The memory controller typically has two internal clocks, one in synch with the global clock, and another that is delayed 90° and used to sample data incoming from the DRAM. Because the DQS is in-phase with the data for read operations (unlike write operations), DQS cannot be used by the memory controller to sample the data directly. Instead, it is only used to ensure that the DRAM's outgoing DQS signal (and therefore data signals as well) are correctly aligned with the memory controller's clocks. The memory controller's 90° delayed clock is used to sample the incoming data, which is possible because the DRAM's DLL guarantees minimal skew between the global clock and outgoing read data. The following paragraphs provide a bit of background to explain the presence of this circuit in DDR SDRAMs:

Because DRAMs are usually external to the CPU, DRAM designers must be aware of the issues involved in propagating signals between chips. In chip-to-chip communications, the main limiting factor in building high-speed interfaces is the variability in the amount of time it takes a signal to propagate to its destination (usually referred to as the *uncertainty* of the signal's timing). The total uncertainty in a system is often the sum of the uncertainty in its constituent parts: e.g., each driver, each delay line, each logic block in a critical path adds a fixed amount of uncertainty to the total. This additive effect makes it very difficult to build high-speed interfaces for even small systems, because even very small fixed uncertainties that seem insignificant at low clock speeds become significant as the clock speed increases.

There exist numerous methods to decrease the uncertainty in a system, including sending a strobe signal along with the data (e.g., the DQS signal in DDR SDRAM or the clock signal in a source-synchronous interface), adding a phase-locked loop (PLL) or delay-locked loop (DLL) to the system, or matching the path lengths of signal traces so that the signals all arrive at the destination at (about) the same time. Many of the methods are complementary, that is, their effect upon reducing uncertainty is cumulative. Building systems that communicate at high frequencies is all about engineering methods to reduce uncertainty in the system.

The function of a PLL or DLL, in general, is to synchronize two periodic signals so that a certain fixed amount of phase-shift or apparent delay exists between them. The two are similar, and the terms are often used interchangeably. A DLL uses variable-delay circuitry to delay an existing periodic signal so that it is in synch with another signal; a PLL uses an oscillator to create a new periodic signal that is in synch with another signal. When a PLL/DLL is added to a communication interface, the result is a "closed-loop" system, which can, for example, measure and cancel the bulk of the uncertainty in both the transmitter and receiver circuits and align the incoming data strobe with the incoming data (see for example [Dally & Poulton 1998]).

Figure 18 shows how the DLL is used in DDR SDRAM, and it shows the effect that the DLL has upon the timing of the part. The net effect is to delay the output of the part (note that the output burst of the bottom configuration is shifted to the right, compared to the output burst of the top configuration). This delay is chosen to be sufficient to bring into alignment the output of the part with the system clock.

3 RAMBUS TECHNOLOGY

This section discusses Rambus's technology as described in their 1990 patent application number 07/510,898 (the '898 application) and describes how some of the technologies mentioned would be used in a Rambus-style memory organization as compared to a JEDEC-style memory organization.

3.1 Rambus's '898 Patent Application

This section describes Rambus's memory-system technology as elaborated in their '898 application; the section does not discuss later implementations of their technology such as Concurrent RDRAM and Direct RDRAM.

The most novel aspect of the Rambus memory organization, the aspect most likely to attract the reader's attention, and the aspect to which Rambus draws the most attention in the document, is the physical bus organization and operation: the bus's organization and protocol are more reminiscent of a computer network than a traditional memory bus. When the word "revolutionary" is used to describe the Rambus architecture, this is the aspect to which it applies.[9]

Figure 19 illustrates the "new bus" that Rambus describes in the '898 application. It juxtaposes Rambus's bus with a more traditional DRAM bus and thereby illustrates that these bus architectures are substantially different. As described earlier, the traditional memory bus is organized into four dedicated busses: (1) the data bus, (2) the address bus, (3) the command bus, and (4) the chip-select bus. In contrast, all of the information for the operation of the DRAM is carried over a single bus in Rambus's architecture; moreover, there is no separate chip-select network. In the specification, Rambus describes a narrow bus architecture over which command, address, and data information travel using a proprietary packetized protocol. There are no dedicated busses in the Rambus architecture described in the '898 application: in the Rambus bus organization, all addresses, commands, data, and chip-select information are sent on the same bus lines. This is why the organization is called "multiplexed." At different points in time, the same physical lines carry dissimilar classes of information.

Another novel aspect of the Rambus organization is its width: in Rambus's architecture, all of the information for the operation of the DRAM is carried over a very narrow bus. Whereas the bus in a tradi-

9. There is also significant discussion (and illustration) in the patent application describing Rambus's unorthodox/revolutionary proposed packaging technology, in which a DRAM's pins are all on one edge of the chip and in which the chips are inserted directly into a backplane-type arrangement individually, as opposed to being integrated onto a memory module.

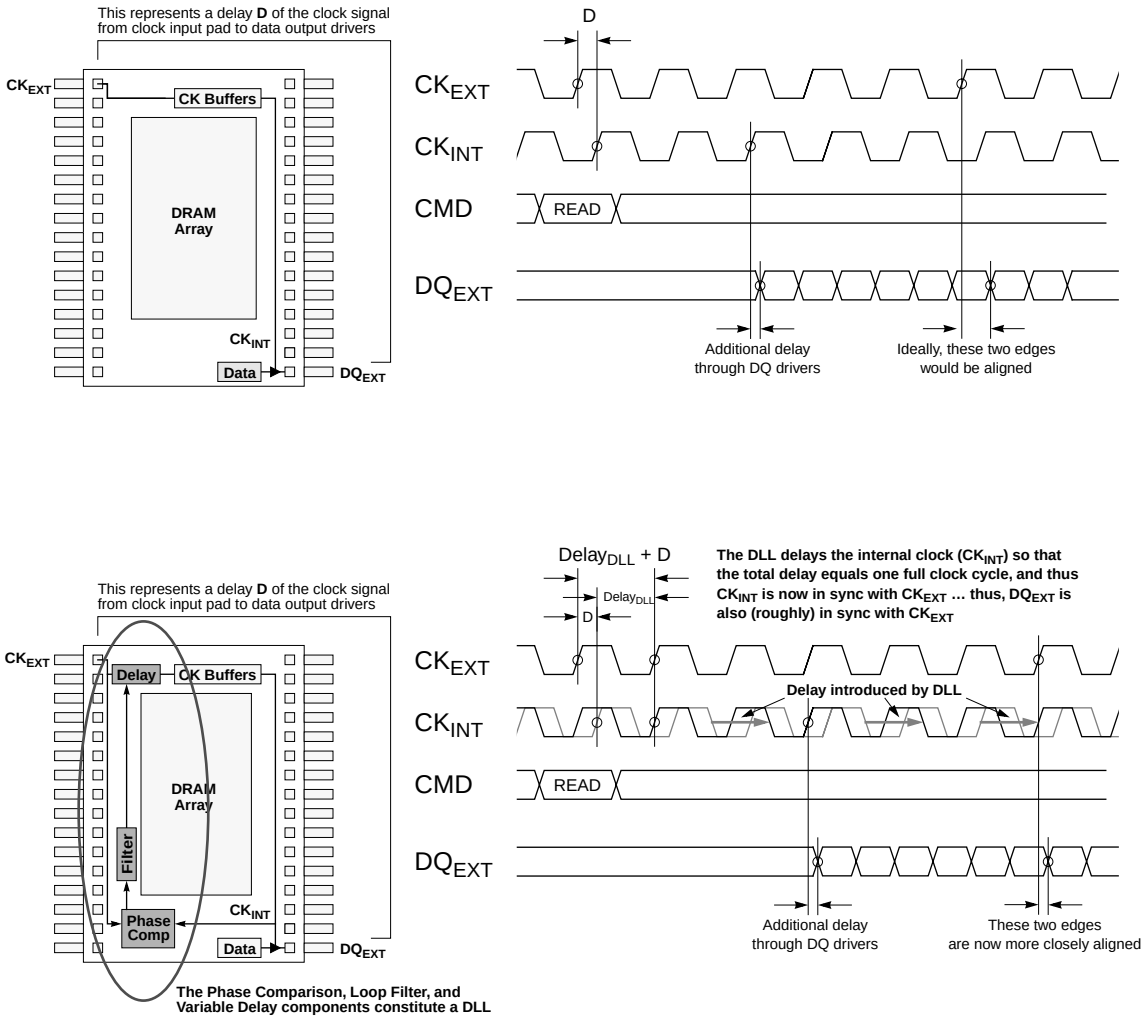
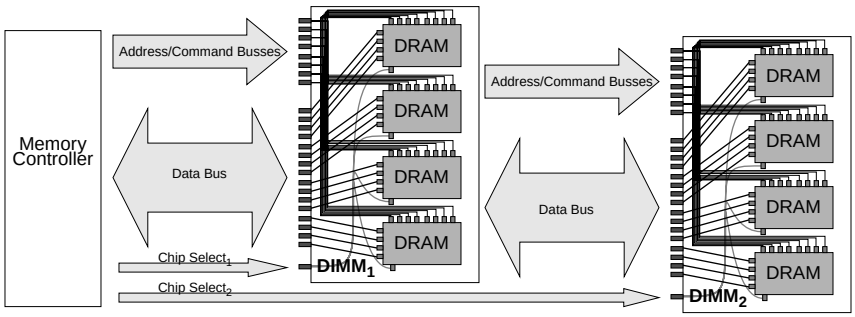


Figure 18: The Use of the DLL in DDR SDRAMs

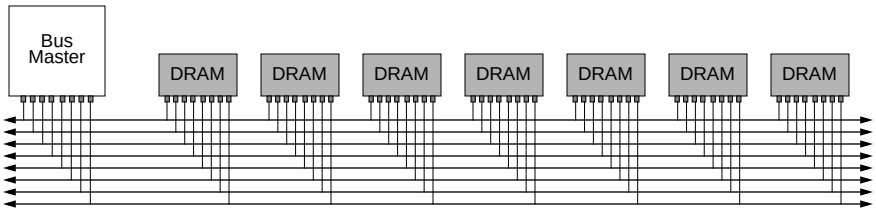
The top figure illustrates the behavior of a DDR SDRAM without a DLL: due to the inherent delays through clock receiver, multi-stage amplifiers, on-chip wires, output pads & bonding wires, output drivers, and other effects, the data output (as it appears from the perspective of the bus) occurs slightly delayed with respect to the system clock. The bottom figure illustrates the effect of adding the DLL: the DLL delays the incoming clock signal so that the output of the part is more closely aligned with the system clock. Note that this introduces extra latency into the behavior of the part.

tional system can use more than ninety (90) bus lines[10], the Rambus organization uses "substantially fewer bus lines than the number of bits in a single address." Given that a single physical address in the early 1990's was twenty to thirty (20–30) bits wide, this indicates a very narrow bus, indeed. In the Rambus specification, the example system uses a total of nine (9) lines to carry all necessary information, including addresses, commands, chip-select information, and data. Because the bus is narrower than a single data address, it takes many bus cycles to transmit a single command from a bus master (i.e., memory controller) to a DRAM device. The information is transmitted over an uninterrupted sequence of bus cycles and must obey a specified format in terms of both time and wire assignments. This is why the Rambus protocol is called "packetized," and it stands in contrast to a JEDEC-style organization in which the command and address busses are wide enough to transmit all address and command information in a single bus cycle[11].

As mentioned, the bus's protocol is also unusual and resembles a computer network more than a traditional memory bus. In an Internet-style computer network, for example, every packet contains the address of the recipient; a packet placed on the network is seen by every machine on the subnet, and every machine must look at the packet, decode the packet, and decide if the packet is destined for the machine itself or for some other machine. This requires every machine to have such decoding logic on board. In the Rambus memory organization, there is no chip-select network, and so there must be some other means to identify the recipient of a request packet. As in the computer network, a Rambus request packet contains (either explicitly or implicitly) the identity of the intended recipient, and every DRAM in the system has a unique identification number (each DRAM knows its own identity). As in the computer network, every DRAM must initially assume that a packet placed on the bus may be destined for it; every DRAM must receive and decode every packet placed on the bus, so that each DRAM can decide whether the packet is for the DRAM itself or for some other device on the bus. Not only does each DRAM require this level of "intelligence" to decode packets and decide if a particular packet is intended for it,



(a) Traditional Memory-Bus Organization: A "Wide-Bus" Architecture



(b) Rambus Memory-Bus Organization: A "Narrow-Bus" Architecture

Figure 19: Memory-Bus Organizations
The figure compares the organizations of a traditional memory bus and a Rambus-style organization. Figure (a) shows a system of a memory controller and two memory modules, with a 16-bit data bus and an 8-bit address and command bus. Figure (b) shows the Rambus organization with a bus master and 7 DRAM slave devices.

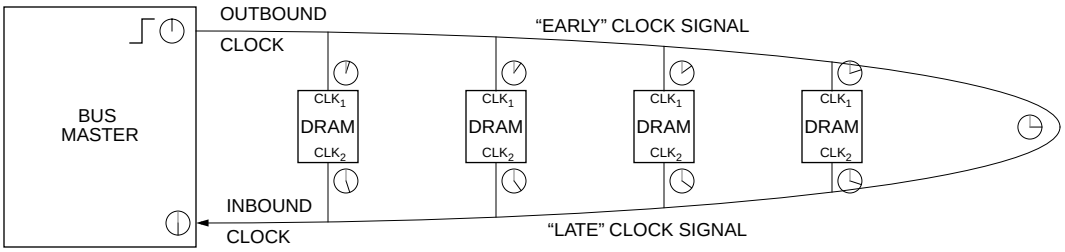


Figure 20: Rambus Clock Synchronization
The clock design in Rambus's 1990 patent application routes two clock signals into each DRAM device, and the path-lengths of the clock signals are matched so that the delay average of the two signals represents the time as seen by the midpoint or clock turnaround point.

but ultimately the implication is that each DRAM is in some sense an autonomous device—the DRAM is not "controlled;" rather, "requests" are made of it.

This last point illustrates another sense in which Rambus is a "revolutionary" architecture. Traditional DRAMs were simply marionettes. The Rambus architecture casts the DRAM as a semi-intelligent device capable of making decisions (e.g., determining whether a requested address is within range), which represents an unorthodox way of thinking in the early 1990's.

10. In a PC system, the data bus is 64 bits; the address bus can be 16 bits; there can be ten or more control lines, as the number of chip-select lines scales with the number of memory modules in the system.

11. A "bus cycle" is the period during which a bus transaction takes place, which may correspond to one clock cycle, or more than one clock cycle, or less than one clock cycle.

3.1.1 Low-Skew Clock Using Variable Delay Circuits

The clocking scheme of the Rambus system is designed to synchronize the internal clocks of the DRAM devices with a non-existent, ideal clock source, achieving low skew. Figure 20 illustrates the scheme. The memory controller sends out a global clock signal that is either turned around or reflected back, and each DRAM as well as the memory controller has two clock inputs, CLK_1 and CLK_2—the first being the "early" clock signal and the second being the "late" clock signal.

Because the signal paths between each DRAM have non-zero length, the global clock signal arrives at a slightly different time to each DRAM. The illustration shows this with small clock figures at each point representing the absolute time (as would be measured by the memory controller) that the clock pulse arrives at each point. This is one component contributing to clock skew, and clock skew traditionally causes problems for high-speed interfaces. However, if the clock path is symmetric—i.e., if each side of the clock's trace is path-length matched so that the distance to the turnaround point is

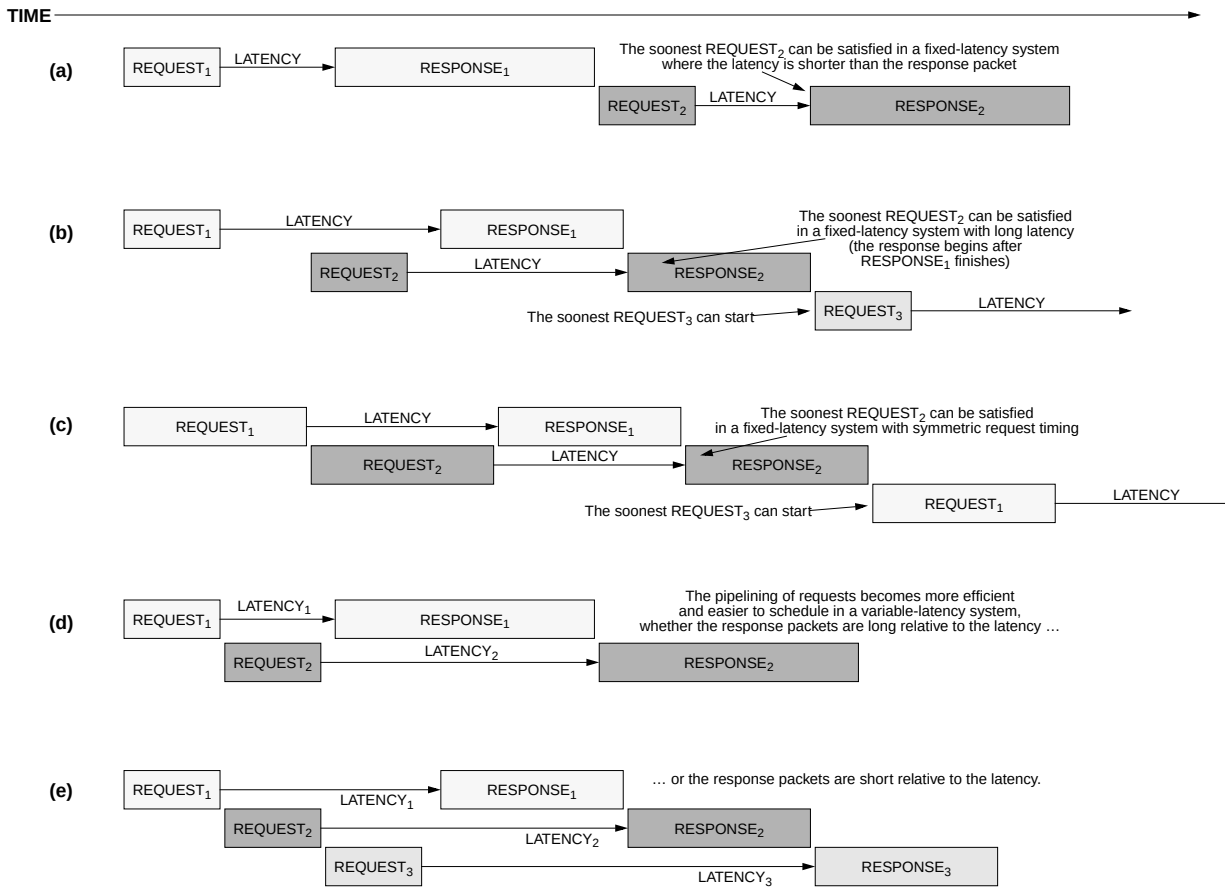


Figure 21: The Importance of Variable Request Latencies in Rambus Organization
Scheduling odd-sized request-response pairs on a multiplexed bus can be difficult to the point of yielding unacceptable performance.

equal from both CLK_1 and CLK_2 inputs—then the combination of the two clocks (CLK_1 and CLK_2) can be used to synthesize, at each device, a local clock edge that is in synch with an imaginary clock at the turnaround point. In the illustration, the memory controller sends a clock edge at 12:00 noon. That edge arrives at the first DRAM at 12:03; it arrives at the next DRAM at 12:06; the next at 12:09; and so on. It arrives at the turnaround point at 12:15 and begins to work its way back to the DRAM devices' CLK_2 inputs, finally arriving at the memory controller at 12:30. If at each point the device is able to find the average of the two clock arrival times (e.g., at the DRAM closest to the memory controller, find the average between 12:03 and 12:27), then each device is able to synthesize a clock that is synchronized with an ideal clock at the turnaround point; each device, including the memory controller, can synthesize a clock edge at 12:15, and so all devices can be "in synch" with an ideal clock generating an edge at 12:15. Note that, even though the figure shows the DRAMs evenly spaced with respect to one another, this is not necessary; all that is required is for the path length from CLK_1 to the turnaround point to be equal to the path length from CLK_2 to the turnaround point, for each device.

Rambus's specification includes on-chip variable-delay circuits (very similar to traditional delay-locked loops, or DLLs) to perform this clock-signal averaging. In other words, Rambus's on-chip "DLL" takes an "early" version and a "late" version of the same clock signal and finds the midpoint of the two signals. Provided that the wires making up the U-shaped clock on the motherboard (or wherever the wires are placed) are symmetric, this allows every DRAM in the system to have a clock signal that is synchronized with those of all other DRAMs.

3.1.2 Variable Request Latency

Rambus's 1990 patent application defines a mechanism that allows the memory controller to specify how long the DRAM should wait before handling a request. There are two parts to the description found in the specification. First, on each DRAM there is a set of registers, called *access-time registers*, that hold delay values. The DRAM uses these delay values to wait the indicated number of cycles before placing the requested data onto (or reading the associated data from) the system bus. The second part of the description is that the DRAM request packet specifies which of these registers to use for a delay value in responding to that particular request.

The patent application does not delve into the uses of this feature at all; the mechanism is presented simply, without justification or application. My (educated) guess is that the mechanism is absolutely essential to the successful operation of the design—having a variable request latency is not an afterthought or flight of whimsy. Without variable request latencies, the support of variable burst lengths, and indeed the support of any burst length other than one equal to the length of a request packet or one smaller than the request latency, cannot function even tolerably well. Figure 21 illustrates. The figure

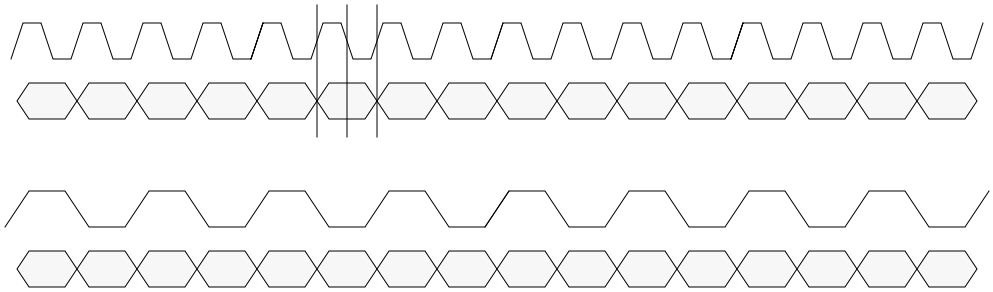


Figure 22: Running the Bus Clock at the Data Rate
Slowing down the clock so that it runs at the same speed as the data makes the clock easier to handle, but it reduces the number of clock edges available to do work: clock edges exist to drive the output circuitry, but no clock edge exists during the "eye" of the data to samepl the data.

shows what happens when a multiplexed, split transaction bus has back-to-back requests: if the request shapes are symmetric (e.g., the request and response packets are the same length, and the latency is slightly longer), or if the latency is long relative to the request and response packet lengths, then it is possible to pipeline requests and achieve good throughput—though neither scenario is optimal in bus efficiency. If the request packet and transaction latency are both short relative to the data transfer (a more optimal arrangement), later requests must be delayed until earlier requests finish, negating the value of having a split transaction bus in the first palce (cf. Figure 21(a)). This is the most likely arrangement: long data bursts are desirable, particularly if the target application is video processing or something similar; the potential problem is that these long data bursts necessarily generate asymmetric request-response shapes because the request packet should be as short as possible, dictated by the information in a request. If the DRAM supports variable request latencies, then the memory controller can pipeline requests, even those that have asymmetric shapes, and thus achieve good throughput despite the shape of the request.

3.1.3 Variable Block Size

Rambus's 1990 patent application defines a mechanism that allows the memory controller to specify how much data should be transferred for a read or write request. The request packet that the memory controller sends to the DRAM device specifies the data length in the request packet's *BlockSize* field. Possible data-length values range from 0 bytes to 1024 bytes (1 kilobyte). The amount of data that the DRAM sends out on the bus, therefore, is programmed at every transaction.

Like variable request latency, the variable-blocksize feature is necessitated by the design of the bus. To dynamically change the transaction length from request to request would likely have seemed novel to an engineer in the early 1990's. The unique features of Rambus's bus—its narrow width, multiplexed nature, and packet request protocol—pose unique scheduling demands on the bus. Variable blocksize is used to cope with the unique scheduling demands—it enables the use of Rambus's memory in many different engineering settings, and it helps to ensure that Rambus's bus is fully-utilized.

3.1.4 Running the Clock at the Data Rate

The design specifies a clock rate that can be half what one would normally expect in a simple, non-interleaved memory system. Figure 17 illustrates, showing timing for two different clock arrangements. The top design is a more traditional arrangement; the bottom uses a clock that is half the speed of the top design. The document, on page 48, lines 6-17, reads as follows:

> Clock distribution problems can be further reduced by using a bus clock and device clock rate equal to the bus cycle data rate divided by two, that is, the bus clock period is twice the bus cycle period. Thus a 500 MHz bus preferably uses a 250 MHz clock rate. This reduction in frequency provides two benefits. First it makes all signals on the bus have the same worst case data rates - data on a 500 MHz bus can only change every 2 ns. Second, clocking at half the bus cycle data rate makes the labeling of the odd and even bus cycles trivial, for example, by defining even cycles to be those when the internal device clock is 0 and odd cycles when the internal device clock is 1.

As the inventors claim, the primary reason for doing so is to reduce the number of clock transitions per second to be equal to the maximum number of data transitions per second. This becomes important as clock speeds increase, which is presumably why IBM's toggle mode uses the same technique. The second reason given is to simplify the decision of which edge to use to activate which receiver or driver (the labeling of "even/odd" cycles). Figure 10 in the specification illustrates the interleaved physical arrangement required to implement the clock-halving scheme: the two edges of the clock activate different input receivers at different times and cause different output data to be multiplexed to the output drivers at different times.

Note that halving the clock complicates receiving data at the DRAM end, because no clock edge exists during the "eye" of the data (this is noted in the figure), and therefore the DRAM does not know when to sample the incoming data—that is, assuming no additional help. This additional help would most likely be in the form of a PLL or DLL circuit to accurately delay the clock edge so that it would be 90° out of phase with the data and thus could be used to sample the data. Such a circuit would add complexity to the DRAM and would consequently add cost in terms of manufacturing, testing, and power consumption.

3.2 Use of Technologies in Rambus DRAM and JEDEC SDRAM

The following paragraphs compare briefly, for each of four technologies, how the technology is used in a JEDEC-style DRAM system and how it is used in a Rambus-style memory system as described in the '898 application.

3.2.1 Programmable CAS Latency

JEDEC's *programmable CAS latency* is used to allow each system vendor to optimize the performance of its systems. It is programmed at system initialization and, according to industry designers, it is never set again while the machine is running [Lee 2002, Baker 2002,

Kellogg 2002, Macri 2002, Ryan 2002, Rhoden 2002, Sussman 2002]. By contrast, with Rambus' *variable request latency*, the latency is programmed every time the CPU sends a new request to the DRAM, but the specification also leaves open the possibility that each access register could store two or more values held for each transaction type. Rambus's system has the potential (and I would argue the *need*) to change the latency at a request granularity—i.e., each request could specify a different latency than the previous request, and the specification has room for many different latency values to be programmed. Whereas the feature is a convenience in the JEDEC organization, it is a necessity in a Rambus organization.

3.2.2 Programmable Burst Length

JEDEC's *programmable burst length* is used to allow each system vendor to optimize the performance of its systems. It is programmed at system initialization and, according to industry designers, never set again while the machine is running [Lee 2002, Baker 2002, Kellogg 2002, Rhoden 2002, Sussman 2002]. By contrast, with Rambus' *variable block size*, the block size is programmed every time the CPU sends a new request to the DRAM. Whereas a JEDEC-style memory system can function efficiently if every column of data that is read out of or written into the DRAM is accompanied by a CAS signal, a Rambus-style memory system could not (as the command would occupy the same bus as the data, limiting data to less than half of the available bus cycles). Whereas the feature is a convenience in the JEDEC organization, it is a necessity in a Rambus organization.

3.2.3 Dual-Edged Clocking

Both JEDEC-style DRAMs and Rambus-style DRAMs use a clocking scheme that goes back at least to IBM's high-speed toggle-mode DRAMs, in which on-chip interleaving allows one to toggle back and forth between two buffers (e.g., on the rising and falling edges of a strobe signal) to achieve a data rate that is twice that possible by a single buffer alone. Both JEDEC-style DRAMs and Rambus-style DRAMs transfer data on both edges of a timing signal. In JEDEC-style DRAMs, the timing signal is an internal clock generated from the system clock and the DQS data strobe—a source-synchronous strobe that accompanies the data sent and is quiescent when there is no data on the bus. The data is edge-aligned with the system clock. In Rambus-style DRAMs, the timing signal is a synthesized internal clock signal in synch with no other clock source in the system and generated from two different phases of the U-shaped global clock (described above). The U-shaped clock is not source-synchronous and remains free running whether there is data on the bus or not. As opposed to the DDR clocking scheme, in the Rambus scheme the data is *not* aligned with either phase of the system clock at all—it is neither edge-aligned nor center-aligned. Furthermore, a different phase relationship exists between each Rambus DRAM's output data and the global clock's early and late signals, whereas DDR DRAMs strive to maintain the same phase relationship between each DRAM's output data and the system clock.

3.2.4 On-Chip PLL/DLL

JEDEC uses an *on-chip DLL* in their DDR SDRAMs to ensure that data being driven onto the data bus is aligned with the global clock signal. The DLL does this by delaying the DRAM's response to a read request just long enough that the data is driven at the same time the DRAM sees the next clock edge. Rambus uses an *on-chip variable delay* circuit to ensure that every DRAM in the system as well as the memory controller has a synchronized clock (i.e., they all believe that it is precisely 12:00 at the same time). The delay circuit does this by finding the midpoint in time between an "early" version and a "late" version (i.e., two different phases) of the same clock signal and thereby creates a synthesized internal clock signal in synch with no other clock in the system. This is a process that is significantly more complicated than a simple delay-locked loop, and thus Rambus's variable delay circuit is more complex than a simple DLL.

4 ALTERNATIVE TECHNOLOGIES

A large number of alternative technologies could achieve the same result as the technologies described; many of these alternatives are simply applications of long understood techniques to solving particular problems. This section discusses a sample of those mechanisms; it is not intended to be exhaustive but rather to be illustrative.

4.1 Programmable CAS Latency

The primary benefit of programmable CAS latency is flexibility: its presence in a DRAM allows a fast part to emulate the behavior of a slow part, which would enable an OEM or end consumer to intermingle parts from different generations (with different speeds) in the same system or intermingle same-generation parts from different manufacturers (which might have slightly different performance capabilities due to slight variations in process technologies) in the same DIMM [Lee 2002, Baker 2002, Kellogg 2002, Macri 2002, Ryan 2002, Rhoden 2002, Sussman 2002]. To illustrate, if a DRAM manufacturer has a part with a minimum 20ns CAS readout, the part could conceivably work as a CAS-2, 100MHz part (20ns requires 2 cycles at 100MHz), a CAS-3, 150MHz part (20ns requires 3 cycles at 150MHz), or a CAS-4, 200MHz part (20ns requires 4 cycles at 200MHz). Note that the part would never be able to make CAS-2 latencies at the higher bus speeds or CAS-3 at the highest bus speed (CAS-2 requires 13.33ns at 150MHz and 10ns at 200MHz; CAS-3 requires 15ns at 200MHz).

Alternatives that would have been available to the JEDEC community in the early to mid 1990's and would have been considered technically sound include the following:

- Use fixed-CAS-latency parts.
- Explicitly identify the CAS latency in the read or write command.
- Program CAS latency by blowing fuses on the DRAM.
- Scale CAS latency with clock frequency.
- Use an existing pin or a new, dedicated pin to identify the latency via two or more different voltage levels asserted by the memory controller.
- Stay with asynchronous DRAM (e.g., burst or toggle-mode EDO).

4.1.1 Use Fixed CAS Latency

JEDEC actually did consider using a fixed CAS latency instead of a programmable CAS latency or in addition to a programmable CAS latency. Fixing the CAS latency would simply mean that each individual SDRAM would operate with a single, predetermined latency. In response to a read request, the SDRAM would always output data after the predetermined number of clock cycles. There are two options that fall under fixing CAS latency: one option would be to define that all JEDEC-compliant DRAMs have the same latency (e.g., 2 cycles). Another option would be to define that each DRAM part would support one fixed latency (e.g., either 2 or 3 or 4), but different DRAMs would be allowed to have different latencies. Although using fixed latency would have reduced flexibility, it would have simplified testing considerably [Lee 2002], as only one mode of operation would need to be tested instead of two or more, and it would have simplified design slightly, as slightly less logic would be required on the chip. In a JEDEC-style bus organization, where the command, address, and data busses are all separate, the scheduling of requests is simplified if all requests have the same

latency. Thus, memory controllers typically set the CAS latency at initialization and never change it again while the machine is powered on; i.e., current systems use a *de facto* fixed CAS latency while the machine is running [Lee 2002, Baker 2002, Kellogg 2002, Macri 2002, Ryan 2002, Rhoden 2002, Sussman 2002]. Therefore, mandating a fixed latency at the DRAM level would have no discernible effect on system performance.

4.1.2 Explicitly Identify CAS Latency in the Command

One option would be to design the DRAM command set so that each *column read* command explicitly encodes the desired CAS latency. Instead of initializing the DRAM to use a latency of 2 or 3 cycles, and then presenting generic CAS commands, each of which would implicitly use a latency of 2 or 3 cycles, respectively, a memory controller could present commands to the SDRAM that would explicitly specify a desired latency. For example, the memory controller would use "CAS-2" read commands if it desired a latency of two, and it would use "CAS-3" read commands if it desired a latency of three.

Such a re-definition of the command set could be accomplished in different ways. One or more additional pins and bus lines could be added to carry the latency commands, or existing pins and bus lines could be used to carry the latency commands without the addition of new pins. There are several pins whose connections with the memory controller make up the "command bus" in a JEDEC-style DRAM system. These pins include RE (row enable, i.e., RAS), CE (column enable, i.e., CAS), W (write enable), CKE (clock enable), and DQM (DQ mask). Each plays a dedicated function—the CE pin is not toggled for row activations, the RE pin is not toggled for column reads or writes, the CKE pin is used primarily to put the DRAM to sleep, etc. Thus, the five pins are used to support a command set that contains less than 32 unique commands. A redefinition of the command set could detach these dedicated functions from the pins and instead homogenize the pins, turning them into a set of command pins that, taken together, can specify exactly 32 different commands, including *DQ Mask*, *Clock Disable*, *Row Enable*, *Bank Precharge*, *Column Write*, etc. This would make room for read commands that specify the desired CAS latency directly, e.g., *Column Read with Latency 2*, *Column Read with Latency 3*, etc. This would have the negative side-effect of limiting the simultaneous issuing of independent commands that is possible with the current command set (e.g., setting DQ Mask during the same cycle as issuing a *column write*), and if these special instances of simultaneous commands were viewed as truly valuable, one could solve the problem with additional pins.

4.1.3 Program CAS Latency Using Fuses

The CAS latency value could also be determined by the use of on-chip fuses. Modern DRAMs have numerous fuses on them that, when blown, enable redundant circuits, so that at test time a DRAM manufacturer can, for example, disable memory arrays that contain fabrication errors and in their place enable redundant arrays that are error-free; a typical DRAM has thousands of such fuses [O'Donnell 2002]. One could add one or more similar fuses that set different CAS latencies. This would enable a DRAM manufacturer to sell what are essentially fixed-latency parts, but because they could be programmed at the last possible moment, the DRAM manufacturer would not need separate stocks for parts with different latencies: one part would satisfy for multiple latencies, as the DRAM manufacturer could set the fuses appropriately right before shipping the parts. The manufacturer could also sell the parts with fuses intact to OEMs and embedded-systems manufacturers, who would likely have the technical savvy necessary to set the fuses properly, and who might want the ability to program the latency according to their own needs. This level of flexibility would be almost on par with the current level of flexibility, since in most systems the CAS latency is set once at system initialization and never set again [Lee 2002, Baker 2002, Kellogg 2002, Macri 2002, Ryan 2002, Rhoden 2002, Sussman 2002].

4.1.4 Scale CAS Latency with Clock Frequency

One option is for the DRAM to know the bus speed and deliver the fastest CAS latency available for that bus speed. The programming of the DRAM could be either explicit, in which case the memory controller tells the DRAM the bus speed using a command at system initialization, or implicit, in which case the DRAM senses the timing of the clock on the bus and "learns" the bus speed on its own. To return to the earlier example, if a DRAM part had an internal CAS readout latency of 20ns, it could deliver CAS-2 in a 100MHz/10ns bus environment, CAS-3 in a 133MHz/7.5ns bus environment, and CAS-4 in a 200MHz/5ns bus environment. This would satisfy most DRAM consumers, because OEMs and end-users usually want the fastest latency possible. However, without an additional mechanism in either the DRAM or memory controller, the scheme would not allow parts with different internal CAS readout latencies to be mixed within the same system. Disallowing mixed-latency systems would not cause undue confusion in the marketplace, because asynchronous parts in the 1980's and 1990's were sold as "70ns DRAMs" or "60ns DRAMs," etc.; the speed rating was explicit, so both OEMs and end-users were already accustomed to matching speed ratings in their systems. However, if support for mixed-latency systems was desired, one could always design the memory controller so that it determines at initialization the latency of each DIMM in its system[12] and accounts for any latency differences between DIMMs in its scheduling decisions.

4.1.5 Identify CAS Latency Through Pin Voltage Levels

An alternative to programming the CAS latency using a specific command is for the memory controller to specify a latency on a pin using one of several voltage levels, each level representing a different latency. The pin could be an existing pin (RAS or CAS) or could be an additional pin. Any of these approaches would have the same performance effect as the present mechanism. An additional pin would increase packaging costs and testing costs, but only slightly [Lee 2002, Macri 2002]. Note that, if the RAS pin is used to identify the CAS latency, this solution works well in concert with a similar alternative solution for implementing programmable burst length (see details below). Note also that JEDEC considered programming CAS latency and/or burst length through pin signals very early on [minutes of JC-42.3 meeting 60 (Attachment K) Jedec 14250, (Attachment L) Jedec 14254].

4.1.6 Use Asynchronous DRAM

Another alternative to programmable CAS latency is to use asynchronous parts instead of synchronous parts. In December 1991, JEDEC was facing the following choice: to continue to improve asynchronous memory design to meet the desired performance targets or develop synchronous memory standards. JEDEC could have chosen one or several available improvements to the existing asynchronous DRAM standards as an alternative. It could have developed burst EDO DRAM or toggle mode DRAM, which is a type of asynchronous memory with source-synchronous clocking. The advantage of an asynchronous design over a synchronous design is that it enables a smaller die area and, at least in the case of burst-mode EDO, potentially higher performance at the same bus speeds. Because the rising edge of CAS drives the data out onto the bus, this allows faster parts to emulate slower ones by increasing the CAS pulse delay.

12. One would probably not want to create a DIMM out of DRAM parts with different fixed CAS latencies.

4.2 Programmable Burst Length

In JEDEC-style DRAM systems, the ability to set the burst length to different values is a convenience for system designers that can result in better system performance. There are numerous subtle interactions between the chosen parameters of a DRAM system (such as request latency, burst length, bus organization, bus speed, and bus width), and these interactions can cause the resulting performance of that system to vary significantly, even for small changes in those parameters [Cuppu & Jacob 2001]. The ability for a system designer to fine-tune the burst length of the DRAMs affords the designer more flexibility in finding the optimal combination of parameters for his system and the expected workload for that system. In most systems the burst length is set once at system initialization and never set again [Lee 2002, Baker 2002, Kellogg 2002, Rhoden 2002, Sussman 2002].

Alternatives that would have been available to the JEDEC community in the early to mid 1990's and would have been considered technically sound include the following:

- Use a short fixed burst length.
- Explicitly identify the burst length in the read or write command.
- Program burst length by blowing fuses on the DRAM.
- Use a long fixed burst length coupled with the *burst-terminate* command.
- Use a burst-EDO style protocol where each CAS pulse toggles out a single column of data.
- Use an existing pin or a new, dedicated pin to identify a burst length via multiple voltage levels.

4.2.1 Use Fixed (Short) Burst Length

Fixing the burst length would simply mean that each individual SDRAM would operate with a single, predetermined burst length. In response to a read or write command, the SDRAM would always respond by driving/sampling the predetermined number of columns onto/off the bus. There are two options that fall under fixing burst length: one option would be to define that all JEDEC-compliant DRAMs have the same burst length (e.g., 4 columns). Another option would be to define that each DRAM part would support one fixed burst length (e.g., either 2 or 4 or 8), but different DRAMs would be allowed to have different burst lengths. Using fixed burst length would have simplified testing considerably, as only one mode of operation would need to be tested instead of two or more, and it would have simplified design slightly, as slightly less logic would be required on the chip.

Because of their design, PC systems do not access data from DRAMs a single byte at a time or even eight bytes at a time; typically, the minimum amount of data read from or written to the DRAM system is 32 or 64 bytes[13]. Thus, in PC systems, the burst length of each SDRAM is typically set at initialization to a value of bits that allows the system to send or receive 32 or 64 bytes of data every time the CPU accesses the memory. Consequently, the PC world would be able to tolerate a fixed burst length in the DRAM specification. Moreover, there is evidence that most JEDEC members could have been satisfied with a fixed burst length of four.

13. This corresponds to the size of the PC's cache block. A processor cache is a small, fast memory typically on the CPU. The purpose of a cache is to amortize the high cost of memory access over many bytes and to (hopefully) fetch data from the memory system before it is even needed, thus saving a costly memory access in the near future.

4.2.2 Explicitly Identify Burst Length in the Command

One option would be to design the DRAM command set so that each *column read* or *column write* command explicitly encodes the desired burst length. Instead of initializing the DRAM to use a burst length of 4 or 8, and then presenting generic CAS commands, each of which would implicitly use a burst length of 4 or 8, respectively, a memory controller could present commands that would explicitly use a specified burst length. For example, the memory controller would use "BL-4" read and write commands if it desired a burst length of four, and it would use "BL-8" read and write commands if it desired a burst length of eight.

Such a re-definition of the command set could be accomplished in different ways. One or more additional pins and bus lines could be added to carry the burst-length commands, or existing pins and bus lines could be used to carry the burst-length commands without the addition of new pins. There are several pins whose connections with the memory controller make up the "command bus" in a JEDEC-style DRAM system. These pins include RE (row enable, i.e., RAS), CE (column enable, i.e., CAS), W (write enable), CKE (clock enable), and DQM (DQ mask), and each plays a dedicated function—the CE pin is not toggled for row activations, the RE pin is not toggled for column reads or writes, the CKE pin is used primarily to put the DRAM to sleep, etc. Thus, the five pins are used to support a command set that contains less than 32 unique commands. A redefinition of the command set could detach these dedicated functions from the pins and instead homogenize the pins, turning them into a set of command pins that, taken together, can specify exactly 32 different commands, including *DQ Mask*, *Clock Disable*, *Row Enable*, *Bank Precharge*, *Column Write*, etc. This would make room for read and write commands that specify the desired burst length directly, e.g., *Column Read with Burst Length 4*, *Column Write with Burst Length 4*, *Column Read with Burst Length 8*, etc. This would have the negative side-effect of limiting the simultaneous issuing of independent commands that is possible with the current command set (e.g., setting DQ Mask during the same cycle as issuing a *column write*), and if these special instances of simultaneous commands were viewed as truly valuable, one could solve the problem with additional pins.

4.2.3 Program Burst Length Using Fuses

Modern DRAMs have numerous fuses on them that, when blown, enable redundant circuits, so that at test time a DRAM manufacturer can, for example, disable memory arrays that contain fabrication errors and in their place enable redundant arrays that are error-free; a typical DRAM has thousands of such fuses [O'Donnell 2002]. One could add one or more similar fuses that set different burst lengths. This would enable a DRAM manufacturer to sell what are essentially fixed-burst-length parts, but because they could be programmed at the last possible moment, the DRAM manufacturer would not need separate stocks for parts with different burst lengths: one part would satisfy for multiple values. The manufacturer could also sell the parts with fuses intact to OEMs and embedded-systems manufacturers, who would likely have the technical savvy necessary to set the fuses properly, and who might want the ability to program the burst length according to their own needs. This level of flexibility would be almost on par with the current level of flexibility, since in most systems the burst length is set once at system initialization and never set again [Lee 2002, Baker 2002, Kellogg 2002, Rhoden 2002, Sussman 2002].

4.2.4 Use Burst-Terminate Command

One can use a fixed burst length where the length is some number large enough to satisfy developers of systems that prefer large chunks of data and use the *burst terminate* command, also called *burst stop*, to halt data read out of the DRAM or signal the end of data written to the DRAM. The advantage of the scheme is that the

burst stop command is already part of the specification [JEDEC 1993], and bursts must also be interruptible by additional commands such as reads and writes. Therefore, an efficient pipeline of multiple column reads is already possible by simply sending multiple read commands spaced as closely as the desired burst length dictates (e.g., every four cycles to achieve a burst-length of four, even if the nominal burst length is longer). Each successive request would implicitly terminate the request immediately preceding it, and only the last burst in the pipeline would need to be terminated explicitly. The disadvantage of the scheme is that it would increase pressure slightly on the command bus: during a *burst terminate* command the memory controller would not be able to control any other bank on the same command bus. If the increase was determined to be significant, it could be alleviated by redesigning the memory controller to support multiple control busses (which is already supported by memory controllers).

4.2.5 Toggle Data-Out Using CAS

JEDEC could have adopted the burst-EDO style of bursting data, with each successive column read-out driven by toggling the CAS pin or holding it at a low voltage until the desired number of columns has been read out (e.g., holding it low for four cycles to read out four columns of data). This would require some modification to the specification, enabling the DRAM to distinguish between a CAS accompanied by a new column address and a CAS signifying a serial read-out based on the most recent column address received. There are numerous possibilities for solving this, including the following:

- Re-define the command set to have two different CAS commands: one for a new address, the other signifying a sequential read-out based on the most recent column address received.
- Add a pin to the command bus (which is usually thought of as comprising signals such as RAS, CAS, WE, and CKE) that is dedicated for the sequential read-out version of CAS. This would be similar to IBM's implementation of high-speed toggle mode, where the data strobe signal uses a different pin than the CAS signal [Kalter 1990a/b].
- Modify the DRAM to detect when the address bus has valid data, versus high impedance, and use this to identify when the CAS toggle represents a new column address.

The advantage of the second and third schemes, as compared to the first, is that they both allow the memory controller to control other DRAM banks simultaneously while the first bank is involved in data transfer.

Note that this alternative is just as valid for synchronous DRAMs as for asynchronous DRAMs.

4.2.6 Identify Burst Length Through Pin Voltage Levels

As in programmable CAS latency, it is possible to identify to the DRAM the desired burst length using one of several different voltage levels on a pin. That pin can be a new, dedicated pin, or the pin could be the RAS or CAS pins. In particular, using the CAS pin to identify burst length would complement the use of the RAS pin to identify CAS latency: at row select, the DRAM would be notified of the desired CAS latency, which would give the chip ample time (several cycles) to prepare internal buffers for CAS read-out, and then at column-select, the DRAM would be notified of the desired number of columns to transfer. Note that JEDEC considered programming CAS latency and/or burst length through pin signals very early on [minutes of
JC-42.3 meeting 60 (Attachment K) Jedec 14250, (Attachment L) Jedec 14254].

4.3 Dual-Edged Clocking

The advantage of using dual-edged clocking in a JEDEC-style DRAM bus organization is that it increases DRAM bandwidth without having to drive the clock any faster and without a commensurate increase in the clock signal's energy consumption. Alternatives that would have been available to the JEDEC community in the early to mid 1990's and would have been considered technically sound include the following:

- Use two or more interleaved memory banks on-chip and assign a different clock signal to each bank (e.g., use two or more out-of-phase clocks).
- Keep each DRAM single data rate and interleave banks on the module (DIMM).
- Increase the number of pins per DRAM.
- Increase the number of pins per module.
- Double the clock frequency.
- Use simultaneous bidirectional I/O drivers.
- Use asynchronous toggle mode.

4.3.1 Interleave On-Chip Banks

As mentioned earlier, interleaving multiple memory banks has been a popular method used to achieve high bandwidth memory busses using low-bandwidth devices. The technique goes back at least to the mid-1960's, where it was used in two of the highest performance (and, as it turns out, best documented) computers of the day: the IBM System/360 Model 91 [Anderson 1967] and Seymour Cray's Control Data 6600 [Thornton 1970]. In an interleaved memory system, the data bus uses a frequency that is faster than any one DRAM bank can support; the control circuitry toggles back and forth between multiple banks to achieve this data rate. For example, if a DRAM bank can produce a new chunk of data every 10ns, one can toggle back and forth between two banks to produce a new chunk every 5ns, or round-robin between four banks to produce a new chunk every 2.5ns, thereby effectively doubling or quadrupling the data rate achievable by any one bank. An alternative to using dual-edged clocking (e.g., to double or triple or even quadruple the memory bandwidth of SDRAM without using both edges of the clock to send/receive data) is to specify two, three, or four independent banks per DRAM (respectively) and assign each bank its own clock signal. The memory controller would send one request to the DRAM, and that request would be handed off to each bank in synch with the clock assigned to that bank. Thus each bank would receive its request slightly advanced or delayed with respect to the other banks. There are two ways to create the different clock signals:

- The different clock signals driving each of the different banks could be generated by the memory controller (or any entity outside the DRAM). Thus, if there were two interleaved banks, the memory controller would send two clock signals; if there were four interleaved banks, the memory controller would send four clock signals; and so on.
- The DRAM could receive one clock signal and delay and distribute it internally to the various banks on its own. Thus, if there were two interleaved banks, the DRAM would split the incoming clock signal two ways, delaying the second a half-phase; if there were four interleaved banks, the DRAM would split the incoming clock signal four ways, delaying the second clock one-quarter of a phase, delaying the third clock one-half of a phase, and delaying the last clock three quarters of a phase; and so on.

The first implementation would require extra DRAM pins (one CK pin for each bank); the latter would require on-DRAM clock generation and synchronization logic.

4.3.2 Interleave Banks on the Module

Instead of increasing the bandwidth of individual DRAMs, an argument could be made that the only place it matters is at the DIMM level. Therefore, one could take SDRAM parts and create a DDR DIMM specification where on-module circuitry takes a single incoming clock and interleaves two or more banks of DRAM—transparently, as far as the memory controller is concerned. Kentron's success [Kentron 2002] shows that this is certainly within our limits, even at very high data rates: they currently take DDR parts and interleave them transparently at the module level to achieve *quadruple data rate* DIMMs (see the website www.quadbandmemory.com for details).

4.3.3 Increase DRAM Data Width

To achieve higher bandwidth per DRAM, the trend in recent years has been not only to increase DRAM speed but also increase the number of data-out pins: x32 parts are now common. Every increase from x4 to x8 to x16 and so on doubles the DRAM's data rate. Doubling the data rate by doubling the data width of existing parts requires very little engineering know-how. It only requires the addition of more pins and doubling the width of the data bus to each individual DRAM. Note that the number of data pins has increased from x1 parts in late 1980's to x32 parts today [Przybylski 1996], while over the same time period the data rate has increased from 16MHz to the 166MHz clocks found in DDR-333 SDRAM today. JEDEC, therefore, could have simply put their weight more firmly behind this trend and increased bandwidth by increasing pin-out. The advantage of this approach is that it combines very well with the previous alternative—interleaving multiple banks at the module level—because doubling the data width of a DRAM decreases the number of DRAM parts needed to achieve the DIMM data width, effectively doubling the number of independent banks one can put on the DIMM. If the DRAM pin-out increases, then fewer DRAMs would be needed per DIMM to create the 64-bit data bus standard in PC-compatible systems. This would leave extra room on the DIMM for more DRAMs that could make up extra banks to implement an interleaved system.

4.3.4 Increase Module Data Width

As mentioned earlier, one could argue that the DIMM bandwidth is more important than the bandwidth of an individual DRAM. Therefore, one could simply increase the data width of the memory bus (the number of wires between the DIMMs and the memory controller) to increase bandwidth, without having to increase the clock speed of the individual DRAMs at all.

4.3.5 Double the Clock Frequency

An alternative to using a dual-edged clock is to use single-edged clock and simply speed up the clock. There are several advantages of an single-edged clocking scheme over a dual-edged clocking scheme (cf. Figure 17). One advantage illustrated in the figure is the existence of more clock edges to be used to drive data onto the bus and sample data off the bus. Another advantage is that the clock need not be as symmetric as it is in a dual-edged clocking scheme: a single-edged clock's duty cycle need not be 50%, as it needs to be for a dual-edged clock, and the rise and fall times (i.e., slew rates) need not match, as they need to for a dual-edged clock. Consequently, one can achieve the same speed clock in a single-edged scheme with much less effort than in a dual-edged clocking scheme. The disadvantage of this alternative is that it requires more engineering effort than simply widening the memory bus or increasing the number of data pins on a DRAM.

For perspective, increasing clock speed in a JEDEC DRAM requires less effort than in a Rambus DRAM because the speeds are much less aggressive. If the clock is currently being driven so fast that it looks like a sinusoidal or saw-tooth waveform, it cannot be driven any faster without lowering the voltage swing. If the clock has headroom—if there are long periods where it is "flat" and not rising or falling—then it can be driven faster. JEDEC-style bus organizations are more conservative than Rambus-style bus organizations in their clock speeds, so they have more clock headroom. Therefore, using a single-edge clocking scheme and speeding up the clock is more realistic an alternative for JEDEC DRAMs than for Rambus DRAMs.

4.3.6 Use Simultaneous Bidirectional I/O

Running the data at the same rate as the clock doubles a DRAM's bandwidth over previous, single-edged clock designs without an increase in the number of data pins. An alternative is to use simultaneous bidirectional I/O, in which it is possible to conduct a read from the DRAM and a write to the DRAM at exactly the same time—that is, both data values (the read data and the write data) are on the bus simultaneously, which is an effective doubling of the available bandwidth, and it does not require additional data pins. Such a scheme would require structural changes at the DRAM side to accommodate reading and writing simultaneously, and those changes would most likely resemble the ESDRAM design by Enhanced Memory Systems [ESDRAM 1998], which has been proposed for DDR II [Davis et al. 2000]. ESDRAM places a buffer after the sense amps that holds an entire DRAM row for reading and allows concurrent writes to the DRAM array—once the read data is in the buffer the sense amplifiers are no longer needed.

4.3.7 Use (Asynchronous) Toggle Mode

An alternative to double data rate is to use asynchronous DRAMs with toggle mode. As mentioned, researchers using toggle mode in the late 1980's in a laboratory setting had already achieved the same data rates in an asynchronous DRAM as would later be achieved in PC100 SDRAM. Toggle mode was evidently a competitive alternative at the time to synchronous DRAM. Presumably, this would require one or more additional pins on the DRAM package: for example, in IBM's implementation, the data was clocked out of the DRAM by a dedicated toggle pin separate from CAS [Kalter 1990a/b].

4.4 On-Chip PLL/DLL

DDR SDRAMs use an on-chip delay locked loop (DLL) circuit to ensure that the DRAM transmits its data and DQS signal to the memory controller as closely as possible to the next appropriate edge of the system clock. Its use is specified because the clock rates in DDR are high enough to warrant relatively strong methods for reducing the effects of dynamic changes in timing skew. Alternatives that would have been available to the JEDEC community in the early to mid 1990's and would have been considered technically sound include the following:

- Achieve high bandwidth using more DRAM pins or module pins, not clock frequency.
- Use a Vernier method to measure & account for dynamic changes in skew.
- Put the DLL on the memory controller.
- Use off-chip (on-module) DLLs.
- Use asynchronous DRAM, for example toggle mode or Burst EDO.

4.4.1 Go Wider, Not Faster

As the previous section indicates, there are numerous ways to achieve higher bandwidth at the DRAM, DIMM, or memory-system level, and many of the methods that achieve higher bandwidth are easier to implement than increasing clock speed. The use of a DLL is dictated only by the desired increase in clock speed; therefore one can forgo the use of an on-chip DLL by increasing DRAM or DIMM data width.

4.4.2 Vernier Mechanism

There are two main components to the uncertainty of signal propagation time: the first is a static component that is due to differences in process technologies, process variations, electrical loading of an unknown number of memory modules, etc. The second is a dynamic component that is typically due to temperature or voltage fluctuations. Some fluctuations change too fast to be handled effectively, but most of the fluctuations in voltage and temperature change over a relatively long time period: most fluctuations in voltage are associated with the 60Hz AC power cycle, which is a relatively long timespan, and most temperature fluctuations occur on the order of milliseconds or longer [Lee 2002, Macri 2002]. In many DDR systems, the static component is corrected by using a Vernier mechanism in the memory controller at system initialization—it is essentially a trial-and-error method in which the upper and lower limits of timing failure are found and the midpoint is used for signal transmission (analogous to setting the tracking on one's VCR by going forward and backward to the points of noise and then leaving the tracking set somewhere in between) [Lee 2002]. The dynamic component is then corrected by the on-chip DLL.

An alternative to the on-chip DLL is to perform recalibration of the Vernier correction often enough to account for the slower changes in voltage and temperature, as opposed to performing the timing correction only once at system initialization. Assuming that the most significant voltage fluctuations are associated with the 60Hz AC power supply and that most of the temperature fluctuations occur on the order of milliseconds or longer [Lee 2002, Macri 2002], recalibration would be needed roughly once or twice every millisecond. This would not likely impose a huge burden on performance; note that at present every row in a typical DRAM system must be refreshed once every 60ms, and the refresh cost (which amounts to refreshing a new row every couple dozen microseconds) only imposes a few percent overhead on the system [Cuppu et al 1999, Cuppu et al. 2001].

Note that JEDEC members suggest that the existing on-chip DLL will be insufficient at future speeds, and that a Vernier mechanism will be inevitable [Lee 2002, Macri 2002, Kellogg 2002].

4.4.3 Move the DLL onto the Memory Controller

Because the DLL is only used on the DRAM to synchronize outgoing data and DQS signals with the global clock for the benefit of the memory controller [Lee 2002, Rhoden 2002, Karabotsos 2002, Baker 2002, Macri 2002], it is possible to move that functionality onto the memory controller itself. The memory controller could maintain two clocks: the first, for example, could be synchronized with the global clock, and the second could be delayed 90°. The incoming DQS and data signals could be given a variable delay, the amount of delay controlled by a DLL/PLL on the memory controller so that the DQS signal would be in phase with the delayed clock. This arrangement could align the incoming data so that the eye would be centered on the global clock signal, which could be used to sample the data. The weakness of this scheme is that, as clock speeds increase to very high rates, the timing differences between different DIMMs would become significant. Therefore, each DIMM in the system would require a different phase shift between the memory controller's two clocks, which would imply that the memory controller would need to maintain a separate timing structure for each DIMM[14].

4.4.4 Move the DLL onto the DIMM

One alternative is to place the DLL on the DDR module instead of the DDR device itself. Sun has used this alternative for many years [Becker 2002, O'Donnell 2002, Prein 2002, Walker 2002]; moreover, it is similar in nature to the off-chip driver mechanism used by IBM in their high-speed toggle mode DRAM [Kalter 1990a/b]. As mentioned previously, the only function of the DLL on the DDR SDRAM is to synchronize the outgoing DQS and data signals with the global clock. This can be accomplished just as easily on the module, especially if the module is *buffered* or *registered* (which simply means that the module has local storage to hold the commands and addresses that are destined for the module's DRAMs). Note that it is presently common practice for engineers to disable DDR's on-chip DLL to achieve higher performance—the on-chip DLL is a convenience, not a necessity [Rhoden 2002, Kellogg 2002, Macri 2002]. A module-level DLL is perfectly workable; even if the data exiting the DRAM is completely out of synch with the global clock, the module can use its DLL to delay the clock/s, commands, and addresses so that the output of the DRAMs is in synch with the global clock—this might come at the expense of an extra CAS-latency cycle. A similar alternative was considered by JEDEC (cf. JC-42.3 meeting 78, March 20, 1996; JC-42.3 interim meeting, January 31, 1996; JC-42.5 meeting 21, September 15, 1994; JC-42.5 meeting 18, March 8, 1994).

5 REFERENCES

D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo. "The IBM System/360 Model 91: Machine philosophy and instruction-handling." *IBM Journal of Research and Development*, vol. 11, no. 1, pp. 8–24, January 1967.

V. Cuppu and B. Jacob. "Concurrency, latency, or system overhead: Which has the largest impact on uniprocessor DRAM-system performance?" In *Proc. 28th International Symposium on Computer Architecture (ISCA'01)*. Goteborg Sweden, June 2001.

V. Cuppu, B. Jacob, B. Davis, and T. Mudge. "A performance comparison of contemporary DRAM architectures." In *Proc. 26th Annual International Symposium on Computer Architecture (ISCA'99)*, Atlanta GA, May 1999, pp. 222–233.

V. Cuppu, B. Jacob, B. Davis, and T. Mudge. "High performance DRAMs in workstation environments." *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1133–1153. November 2001. (TC Special Issue on High-Performance Memory Systems)

W. Dally and J. Poulton. *Digital Systems Engineering*. Cambridge University Press, Cambridge UK, 1998.

B. Davis, T. Mudge, and B. Jacob. "The New DRAM Interfaces: SDRAM, RDRAM and Variants." In *High Performance Computing*, M. Valero, K. Joe, M. Kitsuregawa, and H. Tanaka, Editors, Vol. 1940 of *Lecture Notes In Computer Science*, pp. 26-31. Springer Publishing, Tokyo, Japan, 2000.

B. Dipert. "The slammin, jammin, DRAM scramble." *EDN*, vol. 2000, no. 2, pp. 68–82, January 2000.

ESDRAM. Enhanced SDRAM 1M x 16. Enhanced Memory Systems, Inc., http://www.edram.com/products/datasheets/16M_esdram0298a.pdf, 1998.

H. Kalter, J. Barth, J. Dilorenzo, C. Drake, J. Fifield, W. Hovis, G. Kelley, S. Lewis, J. Nickel, C. Stapper, and J. Yankosky, "A 50 ns 16 Mb DRAM

14. Most DLLs/PLLs require many cycles to lock onto a signal and create the correct phase shift. Forcing the memory controller to wait many cycles before switching from reading from one DIMM to reading from another would be an unsatisfactory situation. In general, DLLs respond faster than PLLs [Pricer 2002].

with a 10 ns data rate." In *37th IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 232–233. February 1990a. San Francisco CA.

H.L. Kalter, C.H. Stapper, J.E. Barth, Jr., J. DiLorenzo, C.E. Drake, J.A. Fifield, G.A. Kelley, Jr., S.C. Lewis, W.B. van der Hoeven, J.A. Yankosky. " A 50-ns 16-Mb DRAM with a 10-ns data rate and on-chip ECC." *IEEE Journal of Solid-State Circuits*, vol. 25, no. 5 , pp. 1118–1128, October 1990b.

C. W. Padgett and F. L. Newman. *Memory Clocking System*. United States Patent number 3,943,496. Submitted on September 9, 1974.

B. Prince. *High Performance Memories*. John Wiley and Sons, West Sussex, England, 2000.

S. Przybylski. *New DRAM Technologies: A Comprehensive Analysis of the New Architectures*. MicroDesign Resources, Sebastopol CA, 1996.

Rambus. 16/18Mbit & 64/72Mbit Concurrent RDRAM Data Sheet. Rambus, http://www.rambus.com/docs/Cnctds.pdf, 1998.

Rambus. Direct RDRAM 64/72-Mbit Data Sheet. Rambus, http://www.rambus.com/docs/64dDDS.pdf, 1998.

Rambus. Direct RDRAM 256/288-Mbit Data Sheet. Rambus, http://www.rambus.com/developer/downloads/rdram.256s.0060-1.1.book.pdf, 2000.

J. E. Thornton. *Design of a Computer: The Control Data 6600*. Scott, Foresman and Co., Glenview IL, 1970.

5.1 Persons Interviewed

Jacob Baker

Henry Becker

Jan DuPreez

Bob Goodman

Chris Karabotsos

Mark Kellogg

Terry Lee

Joe Macri

Art O'Donnell

Frank Prein

David Pricer

Kevin Ryan

Desi Rhoden

Howard Sussman

Alan Walker