

Analyzing and implementing SDRAM and SGRAM controllers

CHRISTIAN GREEN, MOSYS INC

Today's graphics engines, communications chip sets, and μ Ps are running faster than ever. High-speed core internal clock rates now exceed 300 MHz and are rising quickly. However, slow external-memory accesses, especially those involving DRAM, can greatly restrict overall system performance. Fortunately, higher performance synchronous memories are now available. Although these devices offer potentially much higher bandwidth, obtaining optimum performance from them requires a controller that takes advantage of their enhanced capabilities.

Synchronous DRAMs (SDRAMs), including synchronous graphics RAMs (SGRAMs), differ from their asynchronous DRAM counterparts, such as fast-page-mode and extended-data-out (EDO) RAM, in more ways than the simple presence or absence of a clock input. First, the row-address-strobe (RAS)/activate operation is independent from precharge. Most asynchronous DRAMs precharge whenever you drive the RAS input high. With SDRAMs, however, precharge is an explicit and separate command from RAS/activate to allow multiple memory accesses on the same row. Leaving a row activated is called *RAS parking*.

The burst features of SDRAMs generate their column addresses using an on-chip counter. Asynchronous DRAMs, on the other hand, require an explicit address that the memory controller supplies for each access. Also, SDRAMs include two or four internal banks. Each bank is an independent memory that you can activate and precharge separate from the other banks. This independent operation allows the controller to transfer data to or from one bank during another bank's precharge and activate latencies, maximizing performance (see box "DRAM basics").

SGRAMs differ from SDRAMs in two key areas. First, SGRAMs support a

Designing your own synchronous-DRAM controller lets you tune its cost, complexity, and performance to your application needs. You've got lots of options, so research the trade-offs before proceeding.

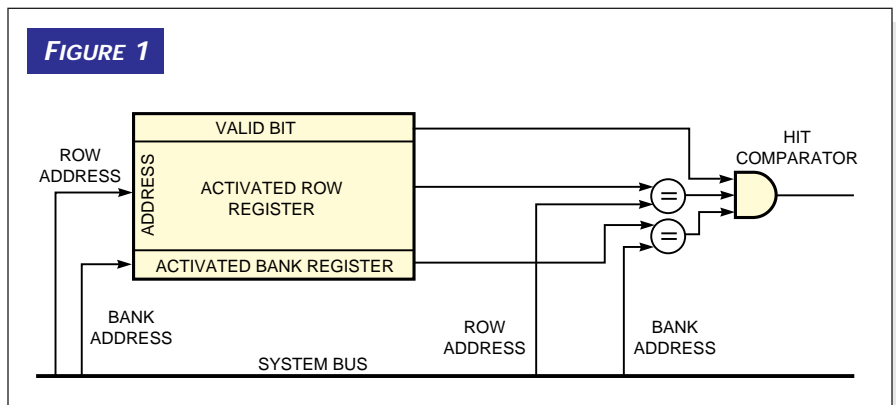
block-write command that fills many addresses with one value in a single operation. SGRAMs also offer a write-per-bit function that allows selective write of a portion of a word. Block write and write per bit are especially valuable in graphics applications, but any design that needs these

functions can benefit from SGRAM. If you tie the Device Special Function (DSF) pin on an SGRAM to ground, the device behaves identical to an SDRAM.

Memory-controller architectures

The simplest SDRAM controller, an *autoprecharge* controller, issues a precharge command after every memory access. The chief advantage of the autoprecharge controller is its simplicity. However, subsequent accesses must issue an activate command before reading or writing, even when this next access is on the same row. The bandwidth lost by this approach may not be too severe if one or more of the following conditions exists:

- Each access's burst is very long. In this case, the overhead of issuing commands becomes a small portion of the total cycle length because many data cycles occur during each command cycle.



The single-comparator controller architecture's main advantage is its simplicity.

SYNCHRONOUS DRAM CONTROLLERS

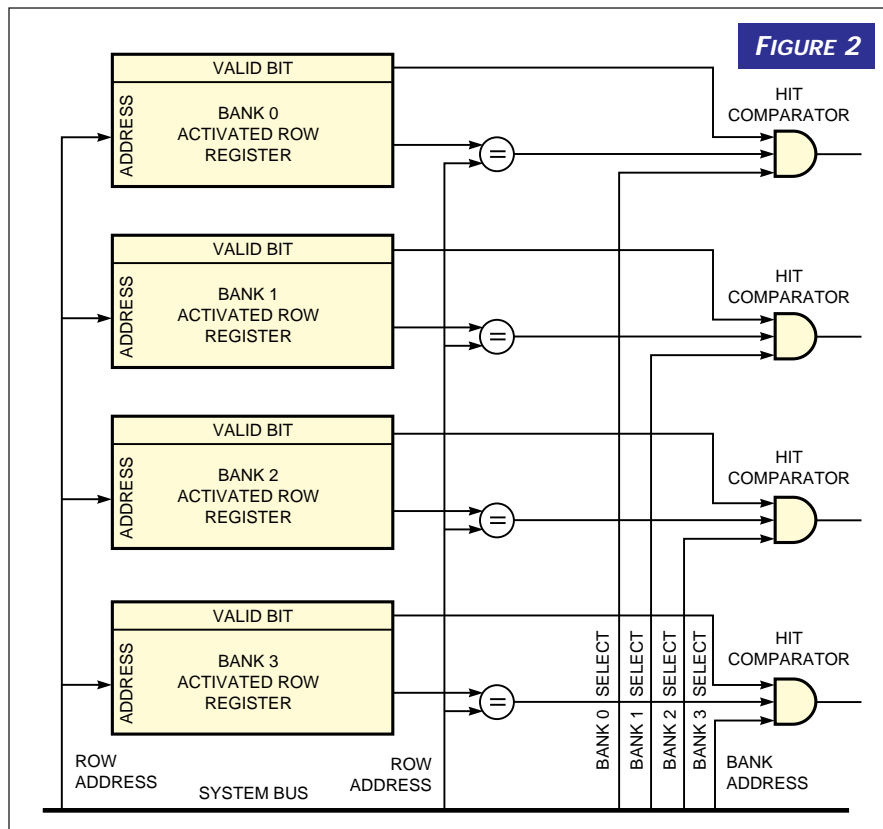
- The accesses are very random. (In other words, nearly every access is to a different row than was the previous access.) In this case, even a more advanced controller would frequently need to execute a precharge/activate sequence.
- The controller can hide the precharge overhead by interleaving accesses with another bank. For instance, Bank B can be undergoing a precharge/activate sequence while Bank A is bursting data. At the end of Bank A's burst, Bank A can precharge and activate while bank B is bursting data.

If you decide to use an auto-precharge controller, you'll find that the easiest way to perform the precharge after each access is to use the write-with-autoprecharge and read-with-autoprecharge commands.

A *single-comparator* controller contains one register that can map to any row in any bank (Figure 1). This controller leaves the most recently used bank activated between accesses. The register contains the bank and row address last activated. When an incoming task requests a memory access, the controller looks at the register to see whether the row the task is addressing is the row currently activated. If the rows are the same, the controller simply issues a read or write command.

If a bank miss occurs with the valid bit set, the controller must activate the bank to access *and* precharge the last bank activated. However, because these two commands are to different banks, the controller can interleave their latencies. For example, the controller can activate the bank to access and in the next clock cycle precharge the last bank accessed. This interleaving greatly improves the probability that the new bank will be ready for activation the next time the task attempts to access it.

A row miss (but a bank hit) that occurs with the valid bit



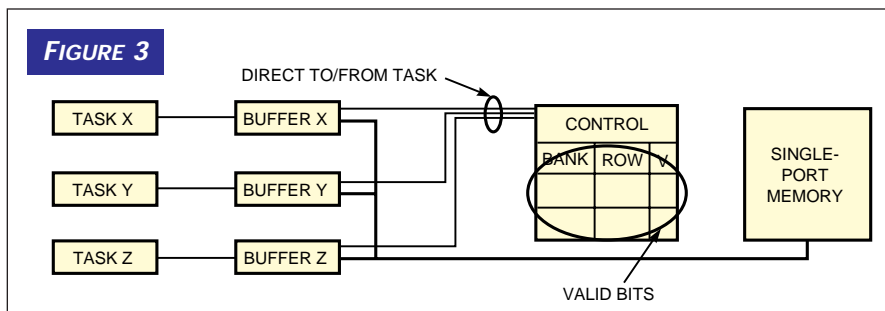
The comparator-per-bank controller architecture targets systems with multiple concurrent tasks.

set means that the task is trying to access an address in the same bank but in a different row from the last access. Therefore, the controller must precharge *and* activate the same bank before the access can take place. This scenario results in the biggest negative performance impact, which you can reduce only if the memory controller can interleave access to another memory chip in parallel with the precharge and activate operations.

At initial power-up and immediately after a refresh command, the data in the bank/row register's contents may be invalid. You must also implement a valid/invalid bit so that the controller knows whether the bank the register points to is precharged or activated. If a task requests an access when all banks are precharged, then the controller must perform an activate operation before the read or write.

The single-comparator controller is optimal for applications with only one or a few tasks, those in which task identification is impossible, and those in which the tasks have locality of reference (a high probability that accessed locations will be close to those of previous accesses).

A *comparator-per-bank* controller



The register-per-task controller architecture provides maximum flexibility in memory-access patterns.

SYNCHRONOUS DRAM CONTROLLERS

integrates a register for each internal memory bank (Figure 2). In this controller, every bank is left activated between accesses. Each bank's register contains its activated row address. During an access, the controller looks at the register corresponding to the bank to be accessed to see whether the row the task is addressing is the same as the row currently activated. If the compare indicates a row miss, the controller must execute a *precharge-on-miss* policy; that is, a pre-charge/activate sequence before the read or write operation.

Just as with a single-comparator architecture, the data in the register may be invalid after initial power-up and immediately after the register has issued a refresh command, so you must implement a valid/invalid bit. An application that uses four 256k×32-bit SGRAMs or SDRAMs on a 64-bit bus

(4 Mbytes total) will contain four banks. Thus, the memory-controller design requires four tags.

Whenever a miss occurs with the valid bit set, the memory controller must precede the activate command with a precharge, increasing the lead-off latency. Thus, this design performs poorly if the tasks exhibit extremely poor locality of reference. The autoprecharge architecture is a better and simpler choice in these situations.

The comparator-per-bank design is optimal for applications with many tasks, those in which task identification is impossible, and those in which the tasks have locality of reference. This scenario is typical of main memory in PCs, workstations, or embedded processors in which the tasks are software processes.

If the controller can identify accesses from different tasks

DRAM BASICS

To write data into the memory, the DRAM controller must first address a row by providing a row address. The row address then directs the word line connecting the storage capacitors to the appropriate bit lines (Figure A).

At the beginning of the datapath is a driver with the desired write data on its input. Assume that the cell or bit that originally contained a logic one must now be written to a logic zero. By making the driver more powerful—using larger transistors—than the cross-coupled latch (also called a “sense amplifier”), the input zero forces the sense amplifier to change state.

To read a cell, the column-address decoder selects a path through the transmission-gate selectors. However, instead of a driver on the other end of the path, a receiver buffers the state of the sense amplifier and drives it off-chip.

A precharge causes switches on the bit line to close, short-circuiting both sides of the sense amplifier. This short circuit forces the bit-line voltage to midway between logic zero and one. The array is then ready for an activate operation.

The charge in each capacitor slowly decays when idle. Whenever the memory controller performs a row-address-strobe (RAS)/activate operation, the cell content refreshes. Synchronous DRAMs also support an explicit refresh command that refreshes one row in the array whenever the memory controller executes the command.

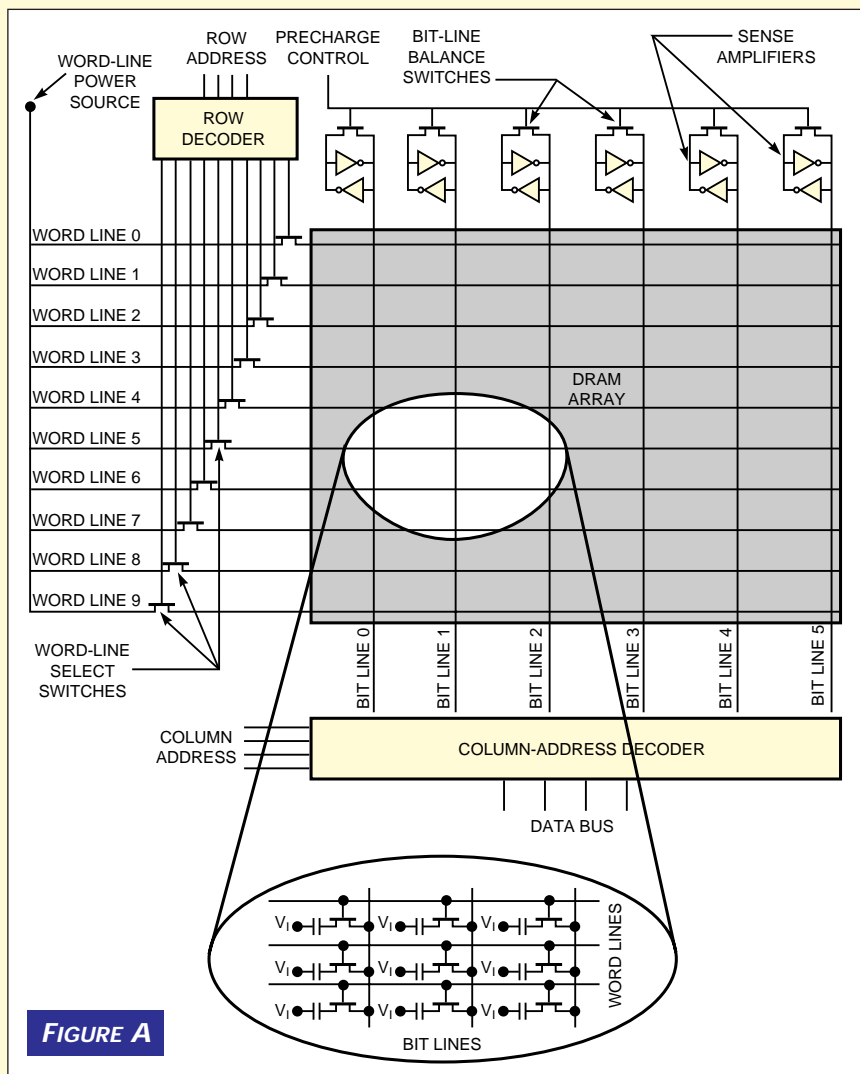


FIGURE A

DRAM accesses travel a multistep internal signal path.

SYNCHRONOUS DRAM CONTROLLERS

and knows their access patterns, then the *register-per-task* architecture performs better than the comparator-per-bank alternative (Figure 3). A register-per-task controller is appropriate for applications that contain few easily identifiable tasks, such as graphics traffic, transfers to and from local mass storage and the LAN connection, and processor cache-line fills.

The objective of the register-per-task architecture is to use the knowledge of each task's unique behavior to improve performance via selective optimizations. A comprehensive description of this architecture is impossible because the strategy is built around the special cases of the tasks. For example, a standard DRAM autoprecharge access best serves low-bandwidth clients with truly random-access patterns. On the other hand, keeping separate tags for both the source and the destination registers best serves graphics engines.

The difficulty in implementing the register-per-task architecture lies in handling bank contention. If two tasks are trying to access the same bank, then the controller tends to "thrash" the memory with incompatible protocols and

repeated precharge and activation cycles, negatively impacting performance. To solve this problem, the controller must

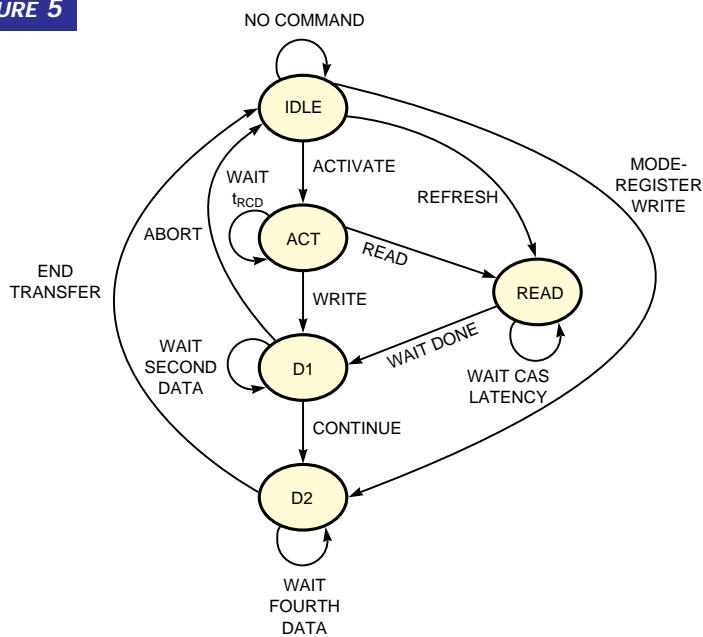
use a well-designed scheduling algorithm that minimizes costly precharge/activate sequences and still satisfies the latency requirements of each task.

Handling memory refresh

SDRAMs provide a refresh command to keep charge in each DRAM bit capacitor from decaying, which would cause loss of stored data. Before issuing the refresh command, the memory controller must precharge all banks in the device. Internally, the memory contains a register with the address of the next row to refresh. When the controller issues a refresh command, the device activates the row in the register, immediately precharges it, and increments the register. The refresh command must execute every 15.6 μ sec.

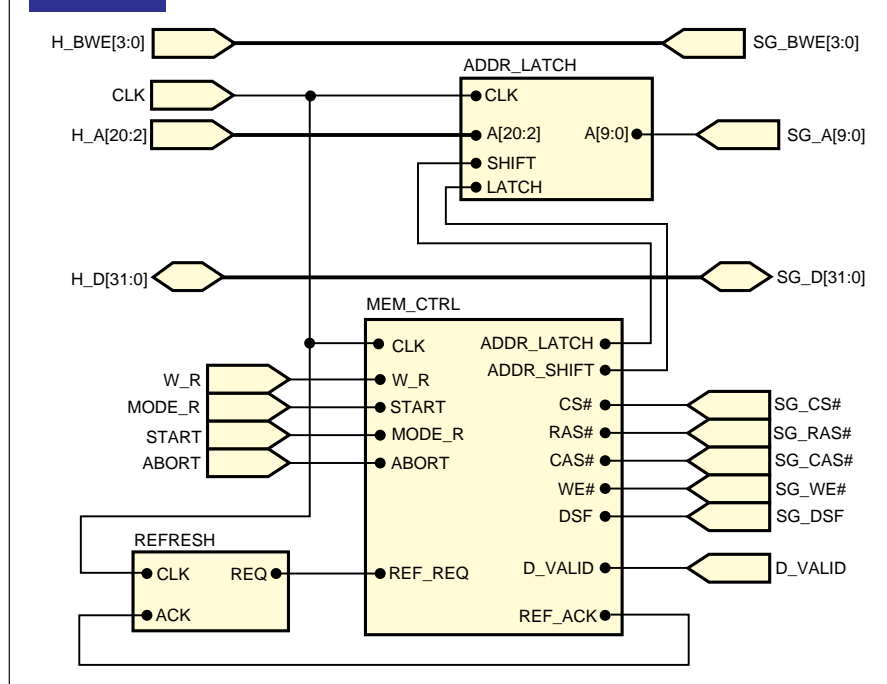
Refresh implementation in the autoprecharge architecture is easy because all the banks are in a precharged condition between memory accesses. The controller needs only to insert the refresh command between memory accesses. In the register-per-task architecture, the memory controller can

FIGURE 5



An autoprecharge controller requires a minimal number of valid states.

FIGURE 4



The memory-controller block is the most adaptable and, therefore, the most complex function to design.

SYNCHRONOUS DRAM CONTROLLERS

simply treat refresh as another task, which will request a refresh every 15.6 μ sec.

The controller must take into account the state of the memory in the single-comparator and comparator-per-bank architectures, because the controller must precharge all banks before issuing a refresh command. If a bank's register bit is invalid, the bank is already precharged, and the controller can immediately issue a refresh command. If the register bit indicates that the bank is activated, the controller must precharge the bank before issuing a refresh command.

The autoprecharge architecture shown in **Figure 4** is com-

pact and can operate at a high clock speed, and you can implement it even in a small to medium CPLD or FPGA. This basic controller can serve as a basis for designing a more robust controller for specific applications. It works with a fixed RAS-to-column-address-strobe (CAS) (t_{RCD}) of 2, a fixed CAS-to-data (CL or t_{CL}) of 2, and a fixed burst length of 4. The following discussion focuses on the implementation of the memory-control block. Although the controller is intended to work as an interface to SGRAM, you can adapt it to work with SDRAM by not implementing the DSF pin. (You can download the VHDL source code for the reference design

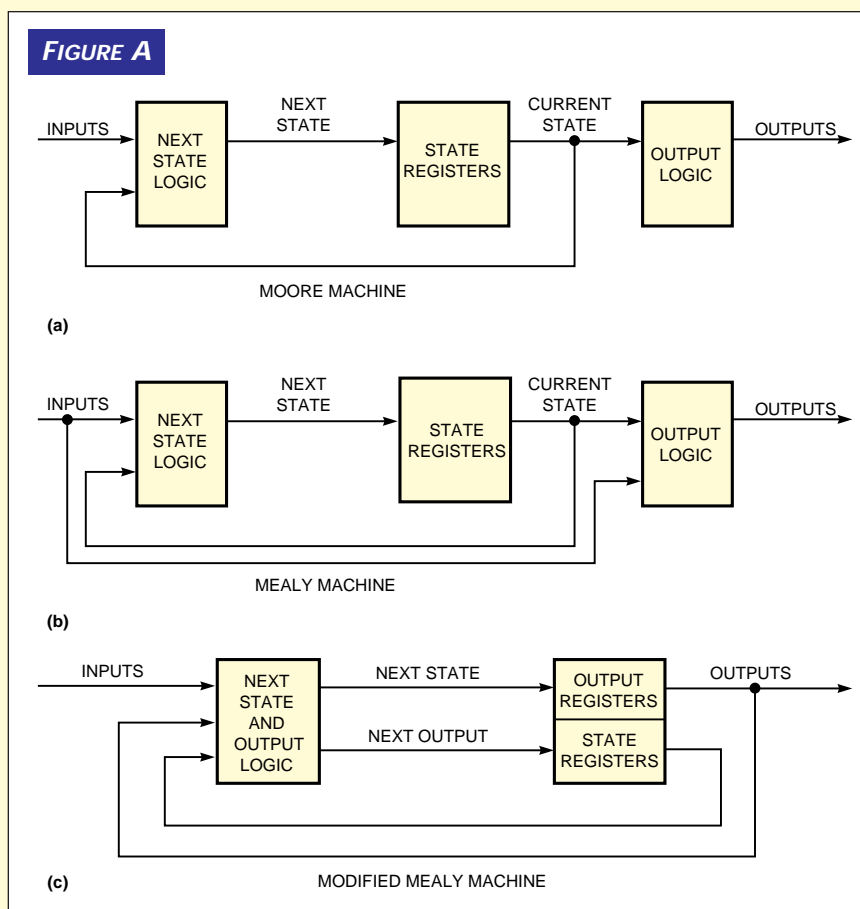
STATE-MACHINE THEORY

You are probably familiar with Mealy and Moore state machines. The reference controller described in this article, however, uses a Modified Mealy-state-machine architecture, which consumes slightly more logic resources than either a Mealy or a Moore state machine but delivers the best functional properties of both (**Figure A**).

The Moore machine's outputs depend only on the current state, so asynchronous inputs do not affect it. One limitation of Moore machines is that each state can have only one pre-defined output. Another limitation is that the combinatorial logic on the machine outputs causes an asynchronous propagation delay in addition to the clock-to-output registered delay.

The Mealy machine's outputs depend on both the current state and the current asynchronous inputs. One key advantage of Mealy machines is that each state transition can have a different output. This flexibility allows a given design to fit into a state machine with fewer states. The main drawback of Mealy machines is that asynchronous inputs can cause logic glitches and metastability problems.

A Modified Mealy machine is essentially a Mealy machine with registered state and outputs. This combination eliminates the output decoding problems of the Moore machine and the asynchronous switching of the Mealy machine. The primary downside is that each output must be registered. You can think of the outputs as an extension of the state registers. Two significant advantages of the Modified Mealy architecture are that the outputs appear



Moore (a), Mealy (b), and Modified Mealy (c) architectures provide alternative methods of implementing state machines.

quickly after the clock edge, because there is only a clock-to-output registered delay, and that the output timing is fixed and predictable.

The memory controller is one of the most critical parts of an ASIC or programmable-logic design. This fact makes Modified Mealy machines ideal

for memory controllers, even though they consume more resources than other types of state machines. On a logic device with tens or hundreds of thousands of gates, you do not miss the extra 50 gates it takes to use this kind of state machine. You do, however, notice the increased performance.

SYNCHRONOUS DRAM CONTROLLERS

from EDN's Web site, www.ednmag.com. In the Registered User area, go into the Software Center and click on "Analyzing and implementing SDRAM and SGRAM controllers.")

The controller uses the following signals:

- **START:** When high, this input tells the controller to start a memory-access cycle. Data reads and writes have a lower priority than do refresh requests or mode register writes.
- **REF_REQ:** With this input asserted, the controller executes a refresh command.
- **MODE_R:** With this input asserted, the controller executes a mode-register write.
- **R_W:** The controller samples this input with START asserted to denote a read or write cycle. A logic high denotes a read.
- **RESET:** This asynchronous input forces the state machine to an idle state.
- **ABORT:** With this input asserted during a write, the memory controller sends a burst stop command to the SGRAM. ABORT's primary purpose is to allow the controller to perform a write of less than four words.
- **CMD_OUT[4:0]:** This 5-bit output bus controls the SGRAM's CS, RAS, CAS, write-enable (WE), and DSF inputs.
- **REF_ACK_OUT:** This output goes high for a cycle after the memory controller accepts a refresh command. It acts as a feedback for the refresh task so that it can deassert its request.
- **D_VALID:** This output is high for read operations during data transfers.
- **ADDR_LATCH_OUT:** This output latches the address bus in response to assertion of the START signal.
- **ADDR_SHIFT_OUT:** This output causes the address latch to implement an 8-bit shift-left operation, necessary because the SGRAM accepts the row and column addresses at different points during the operation.
- **DIR:** This output goes high during read cycles. It can control a bidirectional buffer between the host and the memory. Not all systems need this signal.

The controller design supports read, write, write-mode-register, and refresh commands. The system can also abort any read or write command. **Figure 5** shows the controller state machine, and **Figure 6** shows timing diagrams for the states the controller goes through in response to different commands. The internal wait flip-flop, used

as a flag, causes the state machine to stay in some states for two clocks (see **box** "State-machine theory").

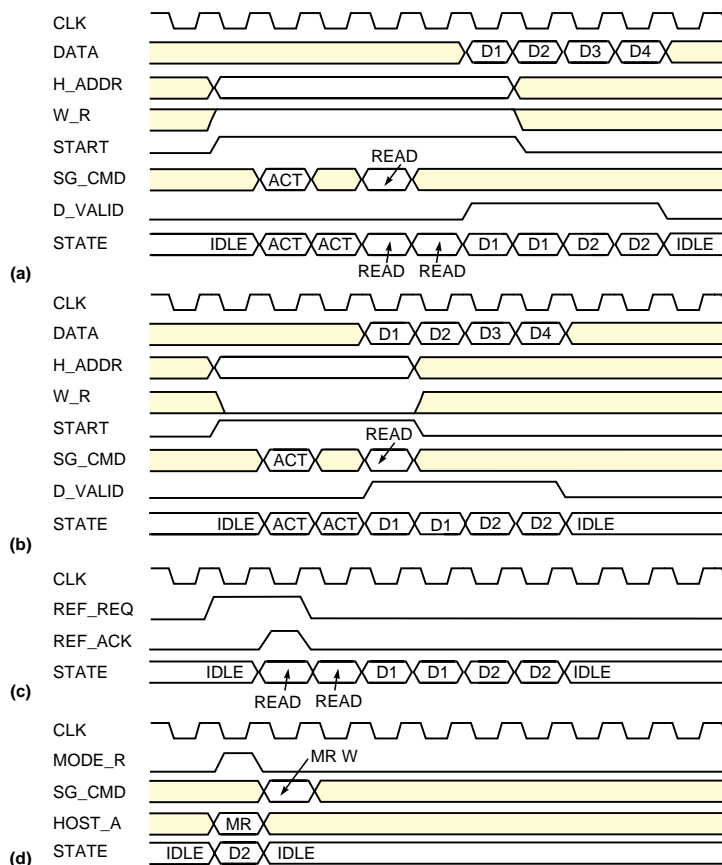
State transitions occur as follows:

READ: IDLE -> ACT -> CMD -> D1 -> D2 -> IDLE
 WRITE: IDLE -> ACT -> CMD -> D1 -> D2 -> IDLE
 REFRESH: IDLE -> CMD -> D1 -> D2 -> IDLE
 MODE REG WRITE: IDLE -> D2 -> IDLE

If the ABORT input goes high during states D1 or D2, the controller issues a burst-stop command to the SGRAM and immediately returns to the IDLE state. To correctly interface to this memory controller, the host should support the following requirements:

- Deassert its read or write command and transfer data on the bus when the controller asserts D_VALID. During a normal write, this scenario occurs during the second clock after the command; for a normal read, it occurs during the fourth clock after the command. However, the refresh task has priority over any host accesses. During refresh, the controller ignores any host commands. Therefore, the host must wait for the D_VALID acknowl-

FIGURE 6



State transitions result in read (a), write (b), refresh (c), and write-mode-register (d) waveforms.

SYNCHRONOUS DRAM CONTROLLERS

edgment before removing its request.


- This controller supports a CAS latency of only two clocks and a burst length of four cycles. Because this may not be the SGRAM's default configuration, the host's first operation should be a mode-register-write to set the correct value.

You should consider two main factors when evaluating the net throughput of an SDRAM device. The first factor is the minimum cycle time; vendors often market their SDRAMs using this parameter. Minimum cycle time tells how fast data can be read or written during a burst.

The second factor is the latency values of the device, which determine how much idle time is on the bus between data bursts. For example, CAS-to-data latency is the number of clocks before valid data appears on the bus after issuing a read command. CAS-to-data is perhaps the single most important latency value because it occurs on every read, regardless of whether the access is a row hit.

Additional latencies, such as precharge-to-activate (t_{RP}) and activate-to-read (t_{RCD}), are also important. These latencies occur whenever the controller accesses a different row from the previous access. For standard SDRAM, a four-word read burst preceded by a precharge/activate sequence takes 12 clocks to complete. With a higher performance SDRAM, the same read cycle takes only nine clocks. This latency decrease results in a net bandwidth increase of 133 to 237

Mbytes/sec (a 34% improvement) without any change in clock speed.

You can clearly see the benefits of using low-latency SGRAMs for graphics applications at the system level, with increased performance of memory-intensive applications, such as large screen-to-screen transfers and 3-D. Overall WinBench 3D scores can increase by 5 to 10%. The performance of memory-intensive 3-D operations that frequently switch rows, such as drawing large triangles that use Gouraud shading, can increase by as much as 35%. 

Author's biography

Christian Green is an application engineer with MoSys Inc (Sunnyvale, CA). He graduated from the University of California—Davis in 1994 with a bachelor's degree in computer engineering, and before working at MoSys he spent two years at Apple Computer (Cupertino, CA). Away from the office, he enjoys making pottery and restoring autos.

VOTE

Please use the Information Retrieval Service card to rate this article (circle one):

High Interest
590

Medium Interest
591

Low Interest
592