



XSA Board SDRAM Controller

July 12, 2005 (Version 1.4)

Application Note by D. Vanden Bout

Summary

This application note describes how to use the XSA Board SDRAM controller in both pipelined and non-pipelined modes.

SDRAM Controller Features

The XSA Board synchronous DRAM (SDRAM) controller core accepts simple read and write requests on the host-side and generates the timed waveforms required to perform these operations on the SDRAM. With pipelining enabled, read and write operations within a row of the SDRAM can be dispatched almost every clock cycle. The controller also manages the refresh operations needed to keep the SDRAM data valid, and will place the SDRAM in a self-refresh mode so data is retained even if the controller ceases operation.

SDRAM Controller Interface

Generic Parameters

Several generic parameters affect the operation of the SDRAM controller:

FREQ: This parameter sets the operating frequency of the master clock input.

CLK_DIV: This parameter sets the divisor for the master clock. The resulting clock is used for the SDRAM controller and is also output on the clk1x output. The stepped-down frequency is used to determine the widths of the timers that sequence the controller operations. Also, the stepped-down frequency must be 25 MHz or higher in order to use DLLs to synchronize operations with the external SDRAM and to generate the stepped-down clock. For frequencies less than 25 MHz, the controller cannot use the DLLs and instead adjusts the clocking so the controller and external SDRAM trigger on opposite edges of the clock. In this case, the clock is not divided and keeps the same frequency as the

master clock. Valid values for CLK_DIV are 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 8.0 and 16.0.

PIPE_EN: This parameter enables pipelined operations in the SDRAM controller.

MULTIPLE_ACTIVE_ROWS: Setting this Boolean parameter to true enables circuitry that keeps track of the active row in each bank of the SDRAM. Setting this parameter to false disables the circuitry so that only the active row in the currently accessed bank is recorded.

MAX_NOP: This parameter sets the number of consecutive clock cycles without a read or write operation after which the SDRAM controller will place the external SDRAM into self-refresh mode.

DATA_WIDTH: This parameter sets the width of the host-side and SDRAM-side data buses.

NROWS, NCOLS: These parameters set the number of rows of data in each bank of the external SDRAM and the number of columns in each row. Each column of a row contains a single word of data that is DATA_WIDTH-bits wide. The values of these parameters for the various XSA Boards are shown below.

Board	NROWS	NCOLS
XSA-50	4096	256
XSA-100	4096	512
XSA-200	8192	512
XSA-3S1000	8192	512

HADDR_WIDTH: This parameter sets the width of the host-side address bus.

I/O Ports

The interface for the core is shown in Figure 1. The functions of the I/O signals are as follows:

clk: This is the master clock input. The clock from the oscillator on the XSA Board enters the FPGA through a global clock input pin and drives this input.

sclk: This output is derived from the master clock and it drives the clock input of the external SDRAM.

sclkfb: This input is a copy of the SDRAM clock signal with delays added by its passage out of the FPGA to the SDRAM and back into the FPGA through a global clock input. The SDRAM controller will use a DLL to compensate for these delays if the frequency of operation is 25 MHz or higher.

clk1x: Host-side logic circuitry in the FPGA can be clocked with this output signal that is derived from the master clock. This clock has a frequency of $FREQ/CLK_DIV$ if that frequency is 25 MHz or higher. Otherwise this clock has a frequency of $FREQ$.

clk2x: This is a clock-doubled version of the master clock if $FREQ/CLK_DIV$ is 25 MHz or higher. Otherwise, $clk2x$ is just a replica of the master clock.

lock: This signal goes high when $clk1x$ is synchronized with the master clock.

bufclk: This is just a globally buffered version of the master clock. It is available even when $clk1x$ has not locked to the master clock.

rst: This is an active-high, asynchronous reset for the internal logic of the SDRAM controller. The reset also causes the controller to initialize the SDRAM for use.

rd: This active-high input initiates a read of a single word from the SDRAM. It is sampled on the rising clock edge and must be held high until the $opBegun$ signal indicates the read process has started. The read control must be lowered before the next rising clock edge after the $done$ signal goes high or else another read operation will start.

wr: This active-high input initiates a write of a single word from the SDRAM. It is sampled on the rising clock edge and must be held high until the $opBegun$ signal indicates the write process has started. The write control must be lowered before the next rising clock edge after the $done$ signal goes high or else another write operation will start.

opBegun: This synchronous output signal goes high to indicate the initiation of a read or write operation.

earlyOpBegun: This asynchronous output signal goes high during the clock cycle immediately preceding the activation of the $opBegun$ signal.

rdPending: This synchronous output signal goes high if there are any read operations in the pipeline that have not yet delivered their data from the SDRAM.

done: This synchronous output signal goes high to indicate the completion of the currently active read or write operation. It remains high for a single clock cycle.

rdDone: This synchronous output signal goes high to indicate the completion of a pipelined read operation. It remains high for a single clock cycle.

hAddr: The address of the SDRAM word that is to be read or written is passed through this input bus. The address value must be held stable until the $opBegun$ signal goes high. The two most-significant bits of $hAddr$ correspond to the bank address bits of the SDRAM, the next $\log_2(NROWS)$ bits of $hAddr$ correspond to the row address within that bank, and the least-significant $\log_2(NCOLS)$ bits correspond to the column address within that row.

hDIn: The data to be written to the SDRAM enters through this input bus. The data value must be held stable until the $opBegun$ signal goes high.

hDOut: The data read from the SDRAM comes out on this bus. This data must be latched by the host-side logic on the rising clock edge after the $done$ or $rdDone$ signal goes high.

cke: This output drives the clock-enable input of the SDRAM.

cs_n: This output drives the chip-select of the SDRAM.

ras_n: This output drives the RAS input of the SDRAM.

cas_n: This output drives the CAS input of the SDRAM.

we_n: This output drives the write-enable input of the SDRAM.

ba: This two-bit bus selects one of the four banks of memory in the SDRAM.

sAddr: The row and column address fields for the SDRAM memory location are output on this bus.

sData: The data word to be written to SDRAM exits the FPGA on this bus during write operations, and data from the SDRAM enters the FPGA on this bus during read operations.

dqmh: This output drives the SDRAM input that controls the drivers for the upper half of the data bus during read operations.

dqml: This output drives the SDRAM input that controls the drivers for the lower half of the data bus during read operations.

status: The current state of the SDRAM controller is made available on this four-bit bus. This is used only for diagnostic purposes.

Using the SDRAM Controller

Non-Pipelined Read Operation

The timing waveforms for a non-pipelined read operation are shown in Figure 2. This example assumes the read operation accesses a memory location in the currently active bank and row of the SDRAM. The sequence of actions is:

Cycle 1: The SDRAM address is applied and the read control signal is driven high. If the SDRAM controller is able to begin the read operation, then the earlyOpBegun signal goes high. The address and read control must be held stable at least until the next rising edge after earlyOpBegun goes high.

Cycle 2: The earlyOpBegun signal goes low and the opBegun signal goes high. The column address is output on the pins that go to the SDRAM chip and the SDRAM control signals are set to initiate a read operation.

Cycle 3: The SDRAM initiates a read of the given column address on the rising clock edge.

Cycle 4: The SDRAM waits for the data to arrive from the given address.

Cycle 5: The data from the SDRAM arrives sometime during this cycle and is guaranteed to be stable by the end of the cycle.

Cycle 6: The data from the SDRAM is clocked into a register on the rising clock edge. The done signal goes high to signal the host-side logic that the data is available on the hDOut bus. The read control must be lowered before the next rising clock edge or else another read operation will be initiated.

Cycle 7: The done signal goes low again but the data on hDOut remains stable until another read operation is completed.

In the previous sequence of actions, it was assumed the read operation was initiated as soon as the read control signal was asserted. There are several cases when the SDRAM controller delays the initiation of the read operation:

When a row is being refreshed. In this case, SDRAM controller completes the row refresh operation. Then the SDRAM banks are precharged and the bank and row containing the given address are activated. Then the read operation can progress as described above.

When the given address is not in the currently active bank or row of the SDRAM. In this case, the controller precharges the SDRAM banks and the bank and row containing the given address is activated. Then the read operation can progress as described above.

When the initiation of the read operation is delayed, the earlyOpBegun and opBegun signals are held low until the read operation is actually initiated.

Non-Pipelined Write Operation

The timing waveforms for a non-pipelined write operation are shown in Figure 3. This example assumes the write operation accesses a memory location in the currently active bank and row of the SDRAM. The sequence of actions is:

Cycle 1: The SDRAM address and the data to be stored there are applied and the write control signal is driven high. If the SDRAM controller is able to begin the write operation, then the earlyOpBegun signal goes high. The address, data and write control must be held stable at least until the next rising edge after earlyOpBegun goes high.

Cycle 2: The earlyOpBegun signal goes low and the opBegun signal goes high. The data and the column address are output on the pins that go to the SDRAM chip and the SDRAM control signals are set to initiate the write operation. The done signal goes high because the SDRAM controller is effectively done at

this point since the SDRAM can complete the write operation on its own.

Cycle 3: On the rising clock edge the SDRAM latches the address and data and initiates a write operation. The output drivers on the data bus are disabled to free the SDRAM data bus.

Cycles 4 and 5: The SDRAM continues its internal operations to write the data into the given address.

In the previous sequence of actions, it was assumed the write operation was initiated as soon as the write control signal was asserted. There are several cases when the SDRAM controller delays the initiation of the write operation:

When a row is being refreshed. In this case, SDRAM controller completes the row refresh operation. Then the SDRAM banks are precharged and the bank and row containing the given address are activated. Then the write operation can progress as described above.

When the given address is not in the currently active bank or row of the SDRAM. In this case, the controller precharges the SDRAM banks and the bank and row containing the given address is activated. Then the write operation can progress as described above.

When a read operation is already in progress. In this case, the SDRAM controller completes the read operation already in progress. Then the write operation is initiated along with any preceding precharge and activation steps that are needed.

Pipelined Read Operation

The timing waveforms for three pipelined read operations are shown in Figure 4. New read operations enter the pipeline by keeping the rd signal high while supplying a new address each time the earlyOpBegun or opBegun signal indicates the previous read has started. This example assumes all three read operations access memory locations in the currently active bank and row of the SDRAM. If a read to an address in another bank or row occurs, the SDRAM controller will complete any reads currently in progress and then activate the new bank and row before raising the earlyOpBegun and opBegun signals.

Pipelined Write Operation

The timing waveforms for a three pipelined write operations are shown in Figure 5. New write operations enter the pipeline by keeping the rd signal high while supplying a new address each time the earlyOpBegun or opBegun signal indicates the previous write has started. This example assumes all three write operations are to memory locations in the currently active bank and row of the SDRAM. If a write to an address in another bank or row occurs, the SDRAM controller will complete any writes currently in progress and then activate the new bank and row before raising the earlyOpBegun and opBegun signals.

SDRAM Controller Application

A simple memory tester application is used to demonstrate the use of the SDRAM controller core. The memory tester performs the following functions:

- It initializes a pseudo-random number generator (RNG) with a known seed value.
- It writes a sequence of random numbers throughout a range of memory addresses.
- It re-initializes the RNG with the seed value.
- It reads back the contents of memory and compares it to the sequence from the RNG. Any mismatch indicates an error reading or writing the memory.

The source files that describe this application are:

common.vhd: Some functions and definitions useful in many applications are provided in this file.

memtest.vhd: The memory tester state machine is described in this file.

randgen.vhd: This file contains the RNG used by the memory tester.

sdramcntl.vhd: This file describes the core state machine of the SDRAM controller.

xsasdrmcntl.vhd: This file creates a wrapper around the SDRAM controller core to customize it for the XSA Board.

test_board_core.vhd: This file combines the generic memory tester and the SDRAM controller core to make the complete SDRAM tester.

test_board.vhd: This file instantiates the test_board_core module to create an SDRAM tester for a particular model of XSA Board.

test_board.ucf: This file contains the pin assignments for mapping the SDRAM tester to a particular board.

test_board.npl: This file tells the Xilinx WebPACK tools how to combine the source files to create the SDRAM tester application for a particular model of XSA Board.

Once the memory tester application is compiled and downloaded into the XSA Board, you should see the sequence 2→3→0 displayed on the LED. (If the memory tester detects an error in the SDRAM, then you will see 2→3→E.)

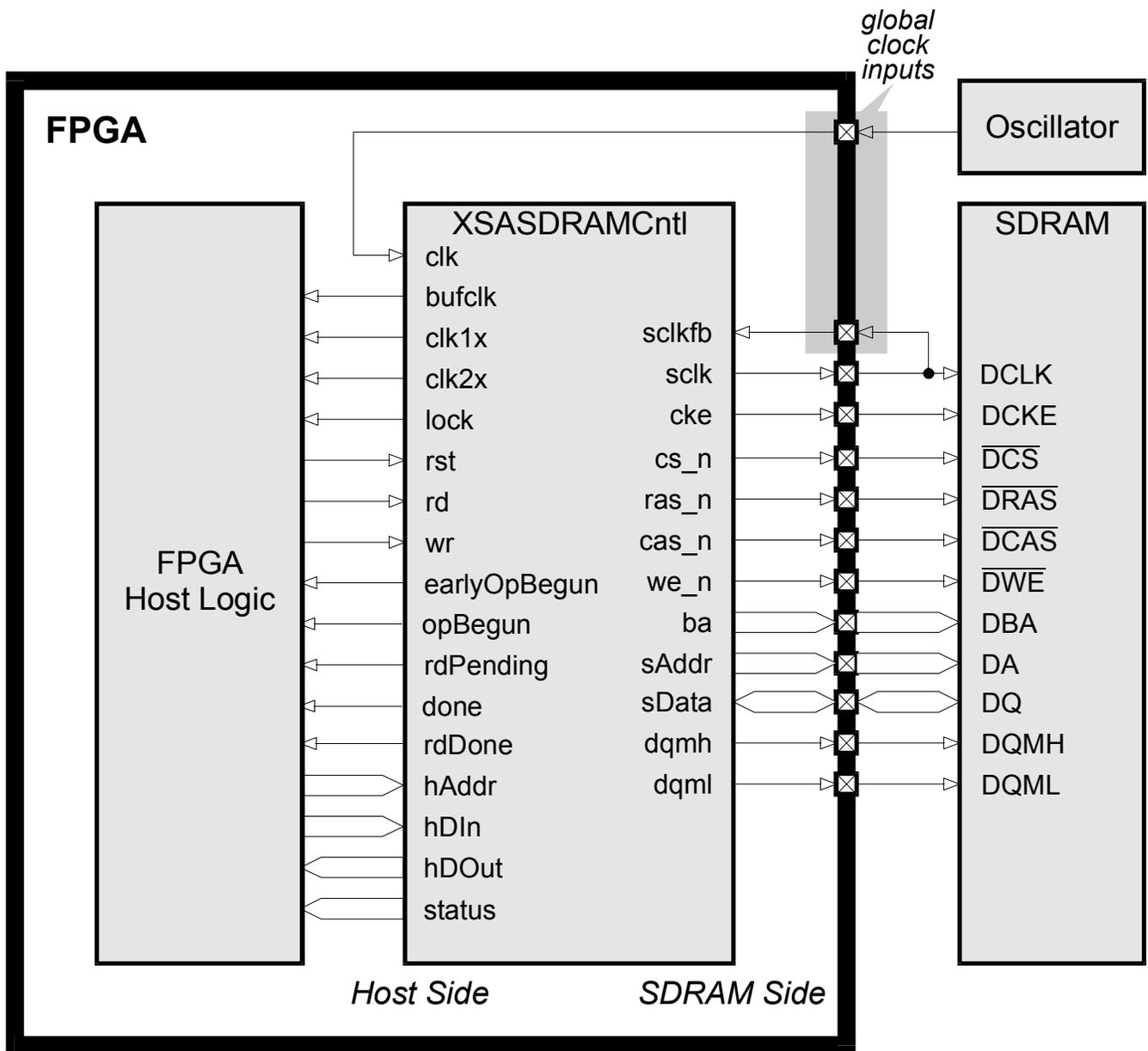


Figure 1: XSA Board SDRAM controller interfaces.

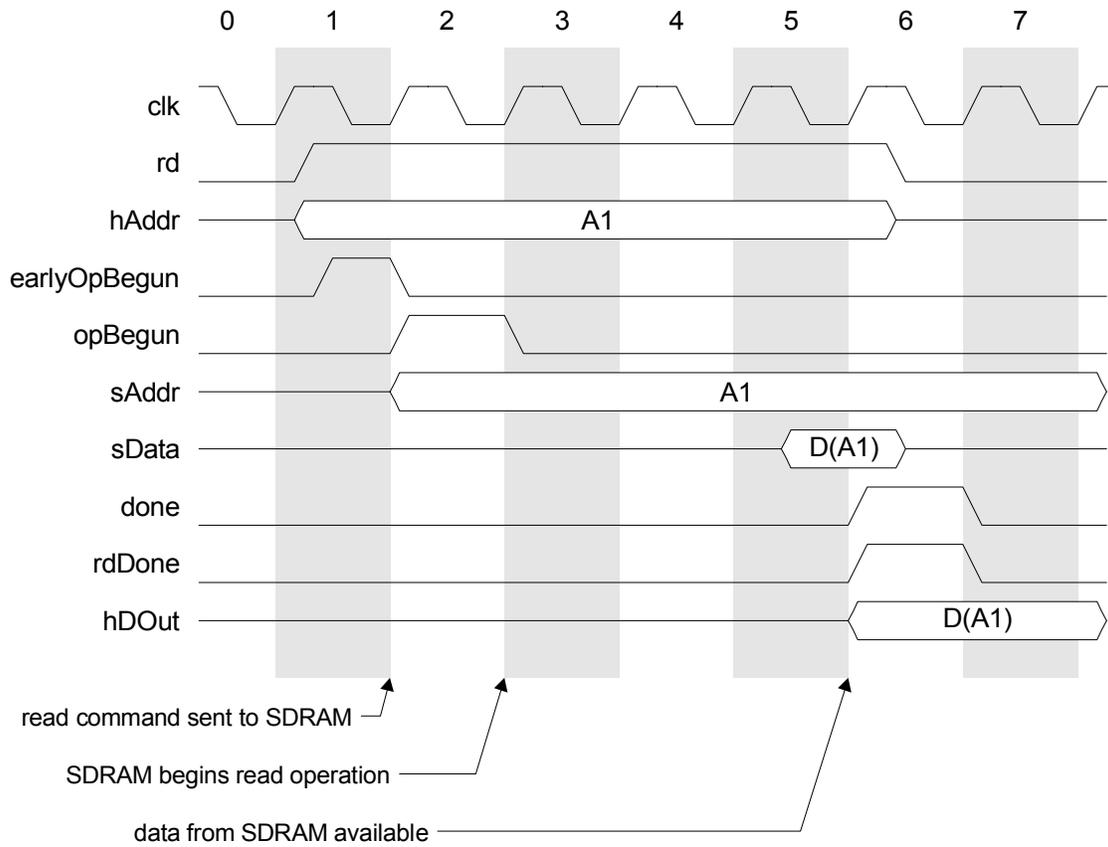


Figure 2: SDRAM controller non-pipelined read operation.

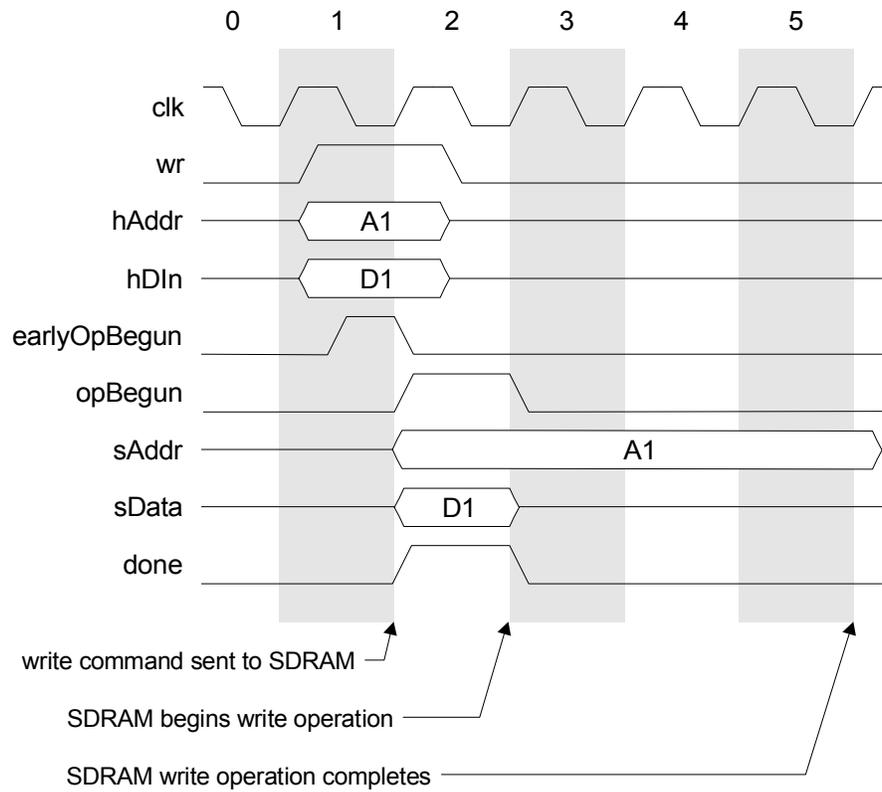


Figure 3: SDRAM controller non-pipelined write operation.

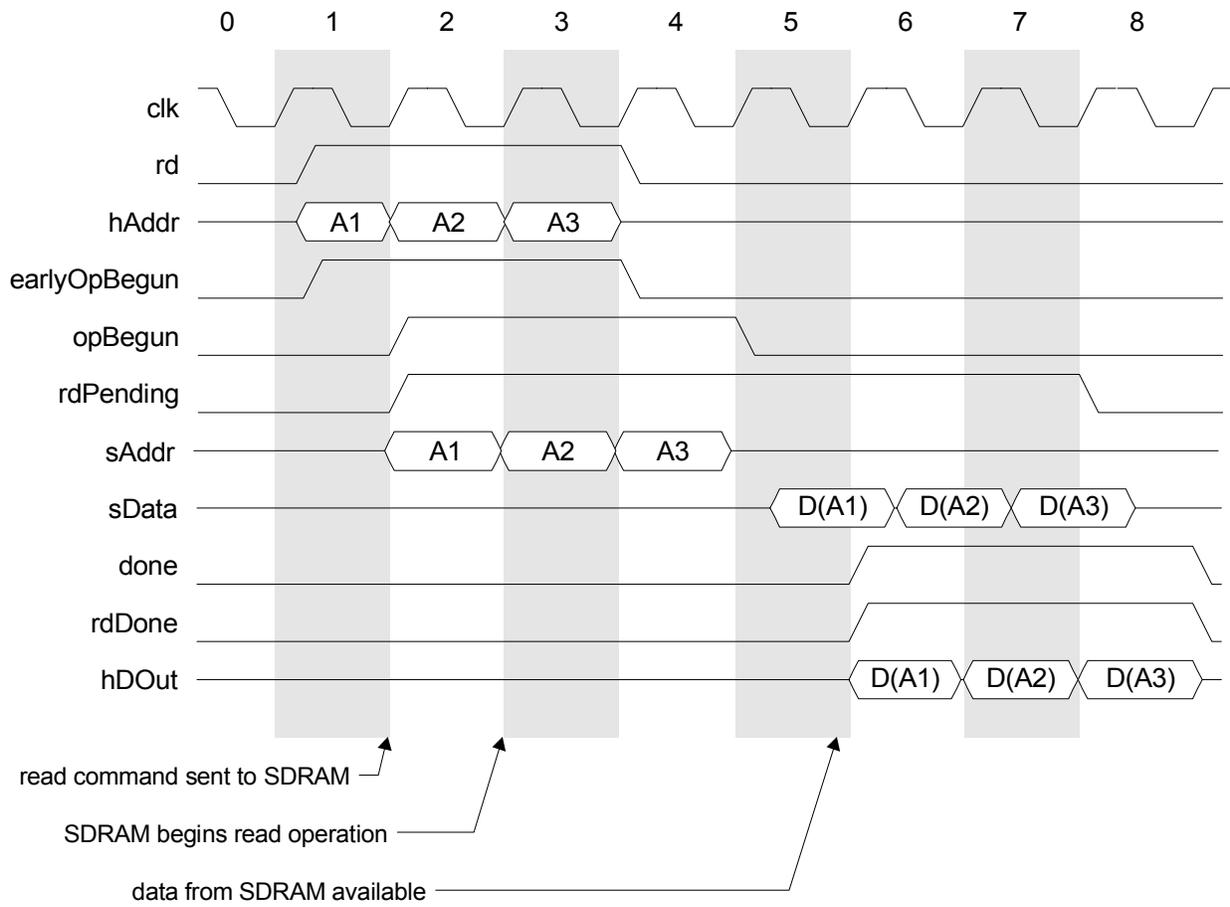


Figure 4: SDRAM controller pipelined read operations.

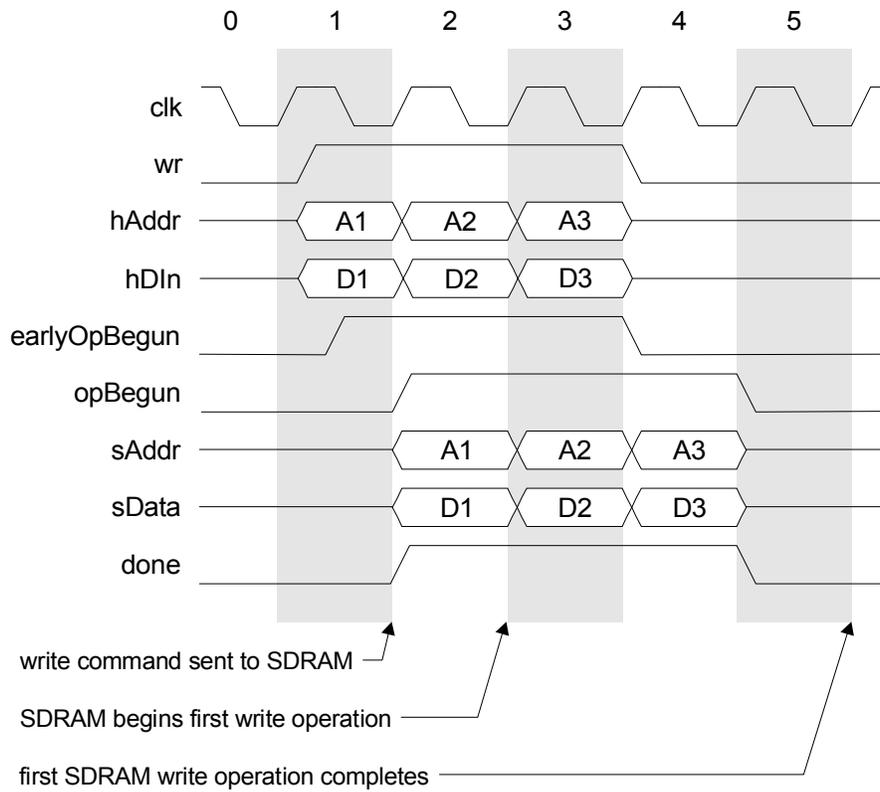


Figure 5: SDRAM controller pipelined write operations.

Revision	Date	Comments
1.4	07/12/05	Updated for SDRAM controller Version 1.4.
1.3	06/27/05	Updated for SDRAM controller Version 1.3.
1.2	04/22/04	Updated for SDRAM controller Version 1.2.
1.1	03/17/04	Updated for SDRAM controller Version 1.1.
1.0	09/27/01	Initial version.