

Understanding how to increase the efficiency and bandwidth of the memory controller is important when you design any external memory interface. This section discusses factors that affect controller efficiency and ways to increase the efficiency of the controller.

Controller Efficiency

Controller efficiency varies depending on data transaction. The best way to determine the efficiency of the controller is to simulate the memory controller for your specific design.

You express controller efficiency as:

Efficiency = number of active cycles of data transfer / total number of cycles

The total number of cycles includes the number of cycles required to issue commands or other requests.



You calculate the number of active cycles of data transfer in terms of local clock cycles. For example, if the number of active cycles of data transfer is 2 memory clock cycles, you convert that to the local clock cycle which is 1.

The following cases are based on a DDR2 SDRAM high-performance controller design targeting a Stratix® IV device that has a CAS latency of 3, and burst length of 4 on the memory side (2 cycles of data transfer), with accessed bank and row in the memory device already open. The Stratix IV device has a command latency of 9 cycles in half-rate mode. The `local_ready` signal is high.

- Case 1: The controller performs individual reads.

$$\text{Efficiency} = 1 / (1 + \text{CAS} + \text{command latency}) = 1 / (1 + 1.5 + 9) = 1 / 11.5 = 8.6\%$$

- Case 2: The controller performs 4 back to back reads.

In this case, the number of data transfer active cycles is 8. The CAS latency is only counted once because the data coming back after the first read is continuous. Only the CAS latency for the first read has an impact on efficiency. The command latency is also counted once because the back to back read commands use the same bank and row.

$$\text{Efficiency} = 4 / (4 + \text{CAS} + \text{command latency}) = 4 / (4 + 1.5 + 9) = 1 / 14.5 = 27.5\%$$

Factors Affecting Efficiency

The two main factors that affect controller efficiency are the interface standard specified by the memory vendor, and the way you transfer data.

The following sections discuss these two factors in detail.

Interface Standard

Complying with certain interface standard specifications affects controller efficiency. When interfacing the memory with the DDR2 or DDR3 SDRAM controllers, you must follow certain timing specifications and perform the following bank management operations:

- Activate

Before you issue any read (RD) or write (WR) commands to a bank within a DDR2 SDRAM device, you must open a row in that bank using the activate (ACT) command. After you open a row, you can issue a read or write command to that row based on the t_{RCD} specification. Reading or writing to a closed row has negative impact on the efficiency as the controller has to first activate that row and then wait until t_{RCD} time to perform a read or write.

- Precharge

To open a different row in the same bank, you must issue a precharge (PCH) command. The precharge command deactivates the open row in a particular bank or the open row in all banks. Switching a row has a negative impact on the efficiency as you must first precharge the open row, then activate the next row and wait t_{RCD} time to perform any read or write operation to the row.

- Device CAS latency

The higher the CAS latency, the less efficient an individual access. The memory device has its own read latency, which is about 12 ns to 20 ns regardless of the actual frequency of the operation. The higher the operating frequency, the longer the CAS latency is in number of cycles.

- Refresh

A refresh, in terms of cycles, consists of the precharge command and the waiting period for the auto refresh. Based on the memory datasheet, these components require the following values:

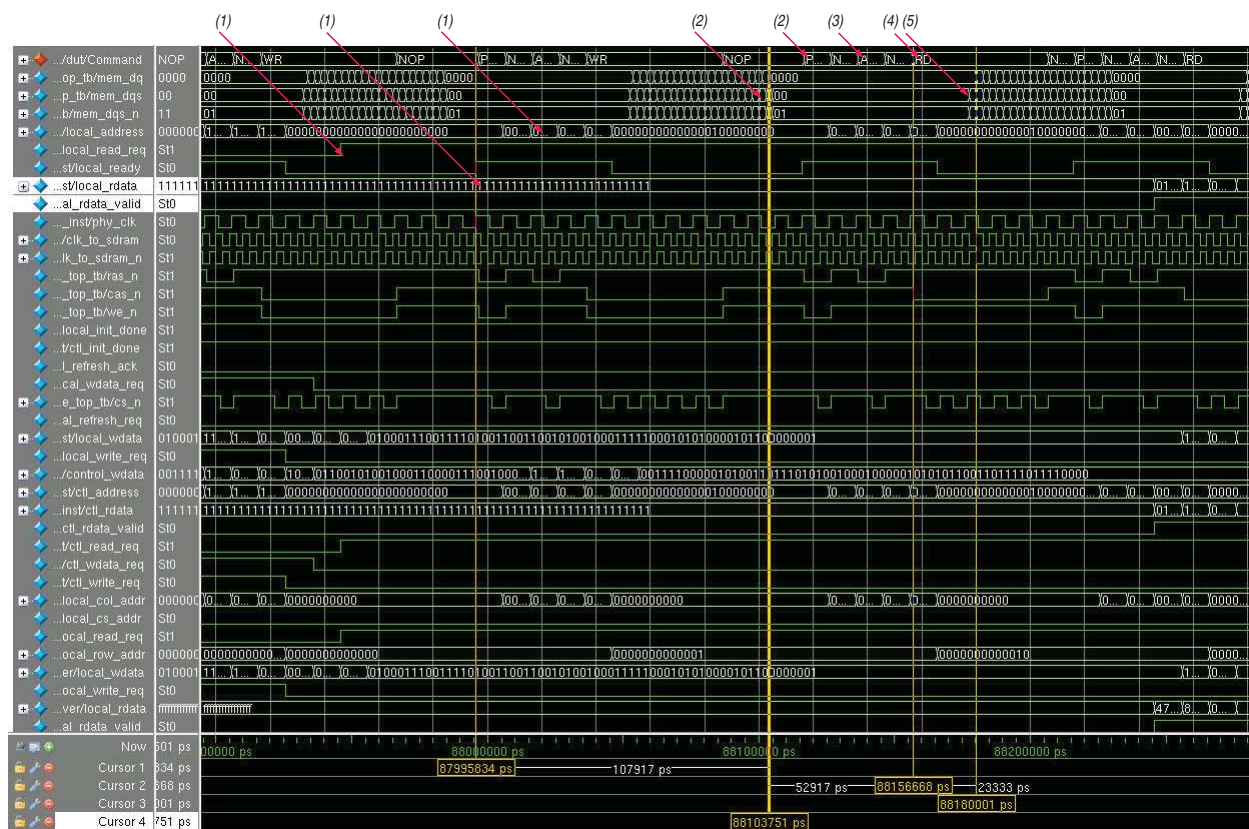
- $t_{\text{RP}} = 12$ ns, 3 clock cycles for a 200-MHz operation (5 ns period for 200 MHz)
- $t_{\text{RFC}} = 75$ ns, 15 clock cycles for a 200-MHz operation.

Based on this calculation, a refresh pauses read or write operations for 18 clock cycles. So, at 200 MHz, you lose 1.15% (18×5 ns / 7.8 us) of the total efficiency.

Figure 13-1 and Figure 13-2 show some examples of how the bank management operations affect controller efficiency. Figure 13-1 shows a read operation in which you have to change a row in a bank. This figure shows how CAS latency and precharge and activate commands affect efficiency.

Figure 13-1 illustrates a read-after-write operation. The controller changes the row address after the write-to-read from a different row.

Figure 13-1. Read Operation—Changing A Row in A Bank



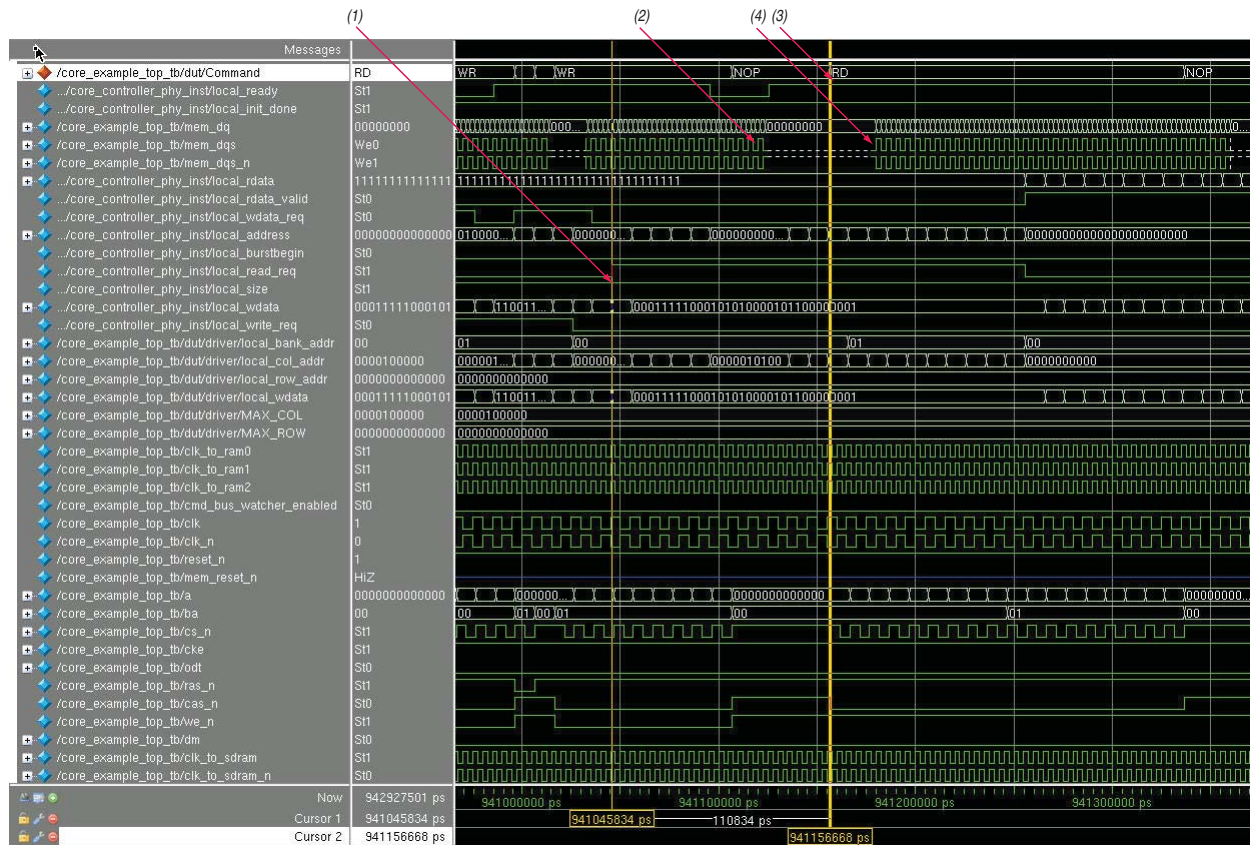
The following sequence of events describes Figure 13-1:

1. The `local_read_req` signal goes high, and when the `local_ready` signal goes high, the controller accepts the read request along with the address.
2. After the memory receives the last write data, the row changes for read. Now you require a precharge command to close the row opened for write. The controller waits for t_{WR} time (3 memory clock cycles) to give the precharge command after the memory receives the last write data.
3. After the controller issues the precharge command, it must wait for t_{RP} time to issue an activate command to open a row.
4. After the controller gives the activate command to activate the row, it needs to wait t_{RCD} time to issue a read command.
5. After the memory receives the read command, it takes the memory some time to provide the data on the pin. This time is known as CAS latency, which is 3 memory clock cycles in this case.

For this particular case, you need approximately 17 local clock cycles to issue a read command to the memory. Because the row in the bank changes, the read operation takes a longer time, as the controller has to issue the precharge and activate commands first. You do not have to take into account t_{WTR} for this case because the precharge and activate operations already exceeded t_{WTR} time.

Figure 13-2 shows the case where you use the same the row and bank address when the controller switches from write to read. In this case, the read command latency is reduced.

Figure 13-2. Changing From Write to Read—Same Row and Bank Address



The following sequence of events describes Figure 13-2:

1. The local_read_req signal goes high and the local_ready signal is high already. The controller accepts the read request along with the address.
2. When switching from write to read, the controller has to wait t_{WTR} time before it gives a read command to the memory.
3. The SDRAM device receives the read command.
4. After the SDRAM device receives the read command, it takes some time to give the data on the pin. This time is called CAS latency, which is 3 memory clock cycles in this case.

For the case illustrated in Figure 13-2, you need approximately 11 local clock cycles to issue a read command to the memory. Because the row in the bank remains the same, the controller does not have to issue the precharge and activate commands, which speeds up the read operation and in turn results in a better efficiency compared to the case in Figure 13-1.

Similarly, if you do not switch between read and write often, the efficiency of your controller improves significantly.

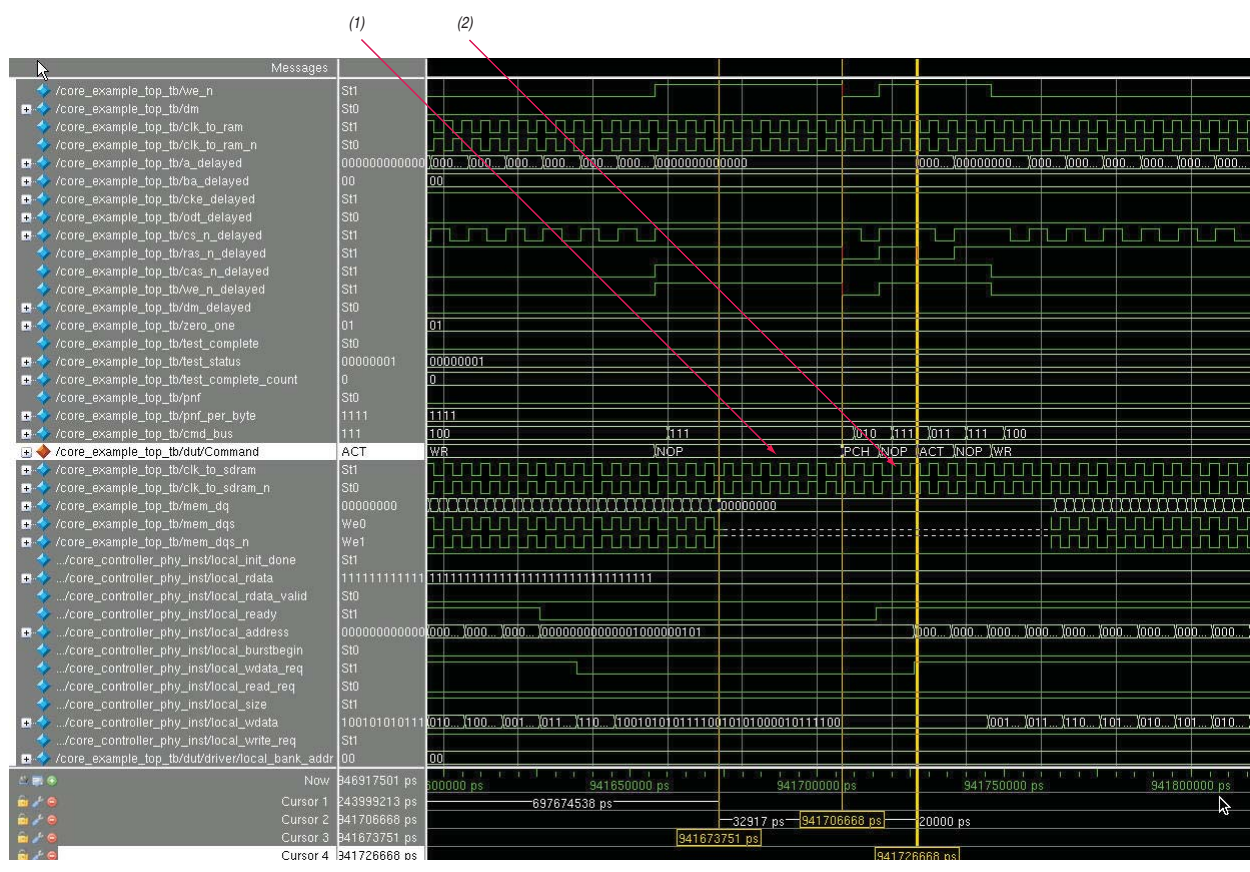
Data Transfer

The following methods of data transfer reduce the efficiency of your controller:

- Performing individual read or write accesses is less efficient.
- Switching between read and write operation has a negative impact on the efficiency of the controller.
- Performing read or write operations from different rows within a bank or in a different bank—if the bank and a row you are accessing is not already open—also affects the efficiency of your controller.

Figure 13-3 shows an example of changing the row in the same bank.

Figure 13-3. Changing Row in the Same Bank



The following sequence of events describes [Figure 13-3](#):

1. You have to wait t_{WR} time before giving the precharge command
2. You then wait t_{RP} time to give the activate command.

Ways to Improve Efficiency

To improve the efficiency of your controller, you can use the following methods:

- [DDR2 SDRAM Controller](#)
- [Auto-Precharge Commands](#)
- [Additive Latency](#)
- [Bank Interleaving](#)
- [Additive Latency and Bank Interleaving](#)
- [User-Controlled Refresh](#)
- [Frequency of Operation](#)
- [Burst Length](#)
- [Series of Reads or Writes](#)

The following sections discuss these methods in detail.

DDR2 SDRAM Controller

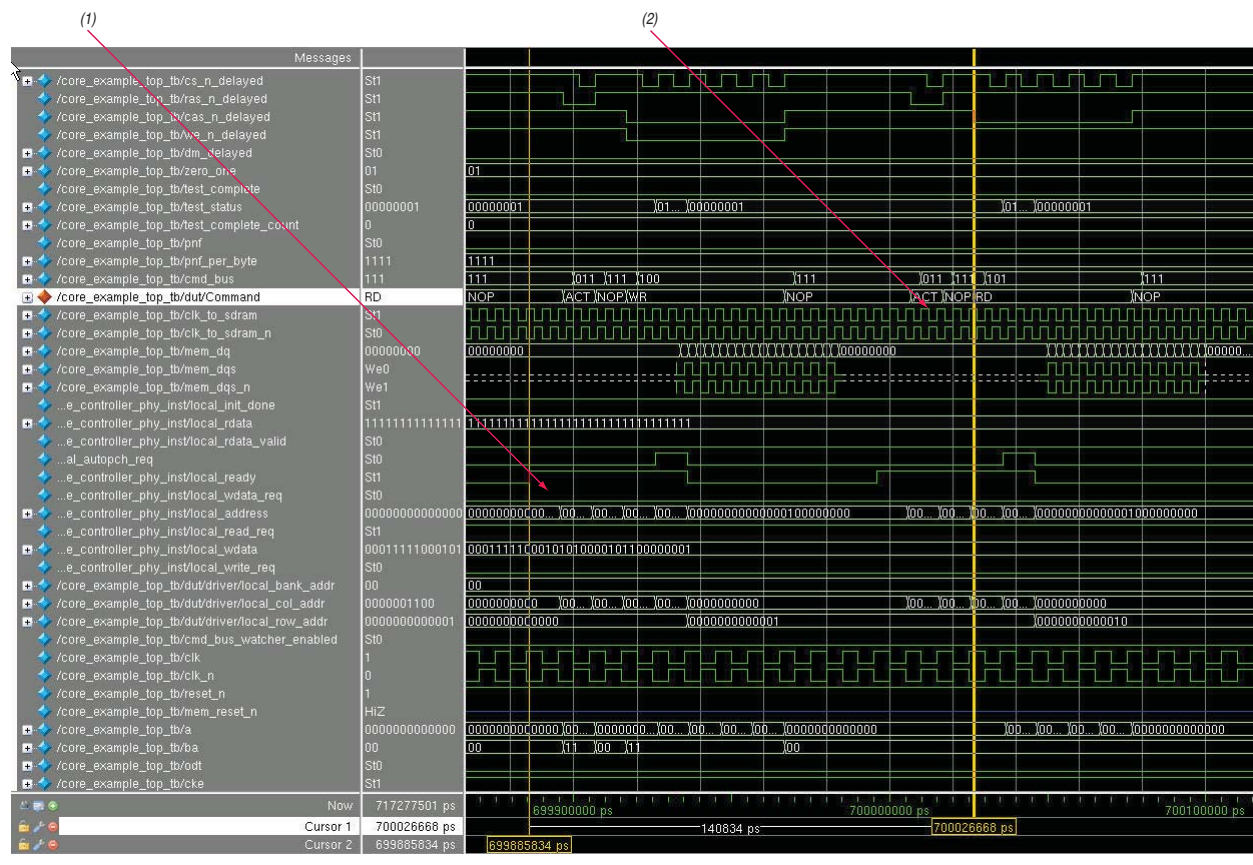
The DDR2 SDRAM controller maintains up to eight open banks; one row in each bank is open at a time. Maintaining more banks at one time helps avoid bank management commands. Ensure that you do not change a row in a bank frequently, because changing the row in a bank causes the bank to close and reopen to open another row in that bank.

Auto-Precharge Commands

The auto-precharge read and write commands allow you to indicate to the memory device that this read or write command is the last access to the currently opened row. The memory device automatically closes or auto-precharges the page it is currently accessing, so that the next access to the same bank is quicker. This command is useful when performing fast random memory accesses.

Figure 13-4 shows how you can improve controller efficiency using the auto-precharge command.

Figure 13-4. Improving Efficiency Using Auto-Precharge Command



The following sequence of events describes Figure 13-4:

1. The controller accepts a read request from the local side as soon as the local_ready signal goes high.
2. The controller gives the activate command and then gives the read command. The read command latency is approximately 14 clock cycles for this case as compared to the similar case with no auto precharge which had approximately 17 clock cycles of latency (described in Figure 13-3).

When using the auto-precharge option, note the following guidelines:

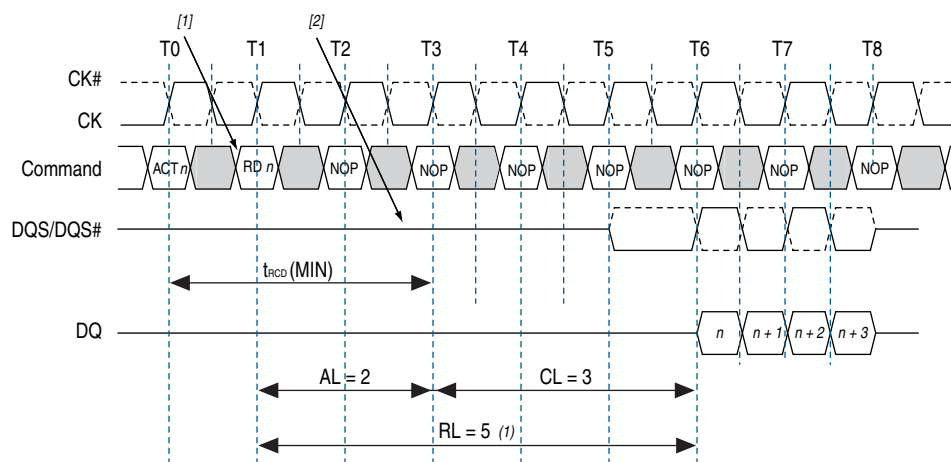
- Use the auto-precharge command if you know the controller is issuing the next read or write to a particular bank and a different row.
- Auto-precharge does not improve efficiency if you auto-precharge a row and immediately reopen it.

Additive Latency

Additive latency increases the efficiency of the command and data bus for sustainable bandwidths. You may issue the commands externally but the device holds the commands internally for the duration of additive latency before executing, to improve the system scheduling. The delay helps to avoid collision on the command bus and gaps in data input or output bursts. Additive latency allows the controller to issue the row and column address commands—activate, and read or write—in consecutive clock cycles, so that the controller need not hold the column address for several (t_{RCD}) cycles. This gap between the activate and the read or write command can cause bubbles in the data stream.

Figure 13-5 shows an example of additive latency.

Figure 13-5. Additive Latency—Read



The following sequence of events describes Figure 13-5:

1. The controller issues a read or write command before the $t_{\text{RCD}}(\text{MIN})$ requirement—additive latency less than or equal to $t_{\text{RCD}}(\text{MIN})$.
2. The controller holds the read or write command for the time defined by additive latency before issuing it internally to the SDRAM device.

$$\text{Read latency} = \text{additive latency} + \text{CAS latency}$$

$$\text{Write latency} = \text{additive latency} + \text{CAS latency} - t_{\text{CK}}$$

Bank Interleaving

You can use bank interleaving to sustain bus efficiency when the controller misses a page, and that page is in a different bank.



Page size refers to the minimum number of column locations on any row that you access with a single activate command.

Without interleaving, the controller sends the address to the SDRAM device, receives the data requested, and then waits for the SDRAM device to refresh before initiating the next data transaction, thus wasting several clock cycles.

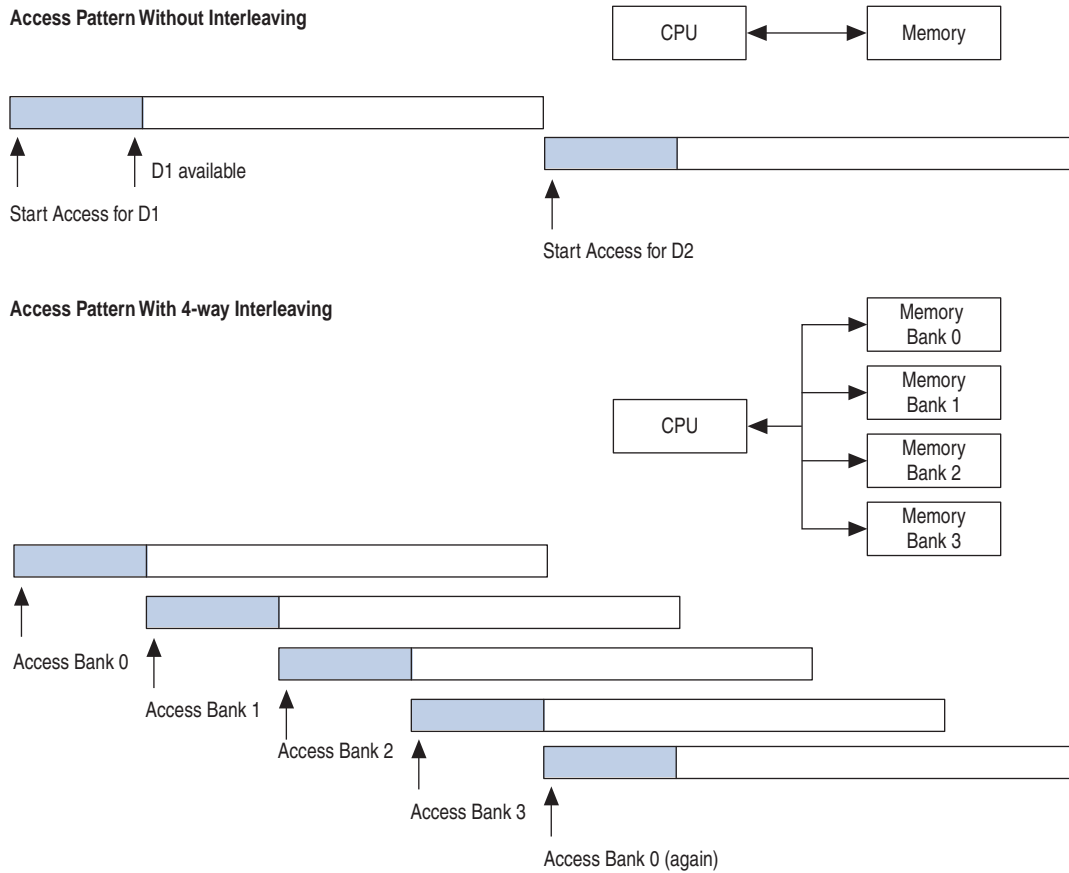
Interleaving allows banks of the SDRAM device to alternate their refresh and access cycles. One bank undergoes its refresh cycle while another is being accessed. By alternating banks, the controller improves its performance by masking the refresh time of each bank. If there are four banks in the system, the controller can ideally send one data request to each of the banks in consecutive clock cycles.

For example, in the first clock cycle, the CPU sends an address to Bank 0, and then sends the next address to Bank 1 in the second clock cycle, before sending the third and fourth addresses to Banks 2 and 3 in the third and fourth clock cycles respectively. The sequence is as follows:

1. Controller sends address 0 to Bank 0.
2. Controller sends address 1 to Bank 1 and receives data 0 from Bank 0.
3. Controller sends address 2 to Bank 2 and receives data 1 from Bank 1.
4. Controller sends address 3 to Bank 3 and receives data 2 from Bank 2.
5. Controller receives data 3 from Bank 3.

Figure 13-6 shows how you can use interleaving to increase bandwidth.

Figure 13-6. Using Interleaving to Increase Bandwidth



Additive Latency and Bank Interleaving

Using additive latency together with bank interleaving increases the bandwidth of the controller.

Figure 13-7 shows an example of bank interleaving in a read operation without additive latency.

Figure 13-7. Bank Interleaving—Without Additive Latency

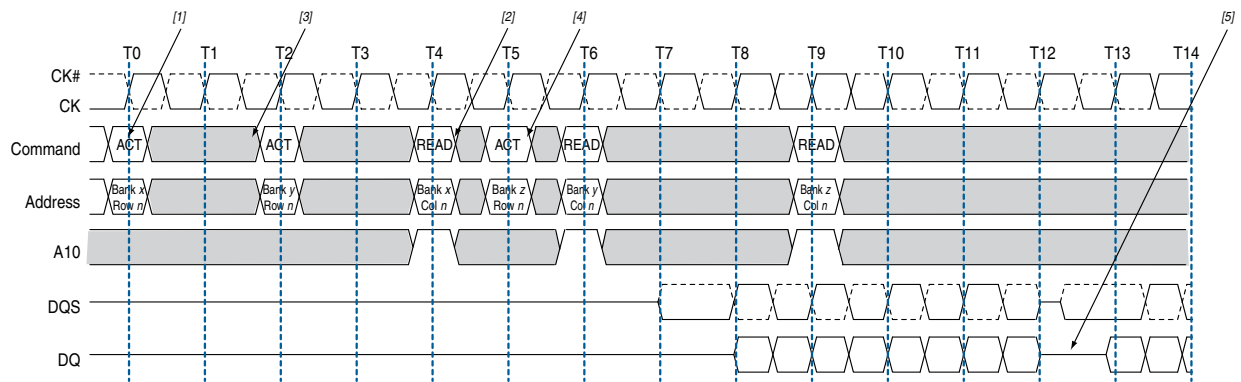


Figure 13-7 illustrates an example of DDR2 SDRAM bank interleave reads with CAS latency of 4, and burst length of 4.

The following sequence of events describes Figure 13-7:

1. The controller issues an activate command to open the bank, which activates bank *x* and the row in it.
2. After t_{RCD} time, the controller issues a read with auto-precharge command to the specified bank.
3. Bank *y* receives an activate command after t_{RRD} time.
4. The controller cannot issue an activate command to bank *z* at its optimal location because it must wait for bank *x* to receive the read with auto-precharge command, thus delaying the activate command for one clock cycle.
5. The delay in activate command causes a gap in the output data from the DDR2 SDRAM device.



If you use additive latency of 1, the latency affects only read commands and not the timing for write commands.

Figure 13-8 shows an example of bank interleaving in a read operation with additive latency. In this configuration, the controller issues back-to-back activate and read with auto-precharge commands.

Figure 13-8. Bank Interleaving—With Additive Latency

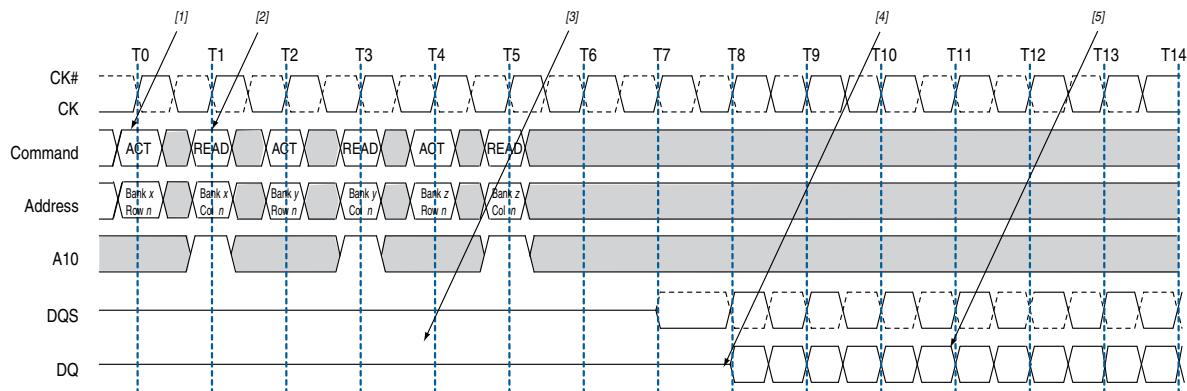


Figure 13-8 illustrates an example of a DDR2 SDRAM bank interleave reads with additive latency of 3, CAS latency of 4, and burst length of 4.

The following sequence of events describes Figure 13-8:

1. The controller issues an activate command to bank x .
2. The controller issues a read with auto precharge command to bank x right after the activate command, before waiting for the t_{RCD} time.
3. The controller executes the read with auto-precharge command t_{RCD} time later on the rising edge T_4 .
4. 4 cycles of CAS latency later, the SDRAM device issues the data on the data bus.
5. For burst length of 4, you need 2 cycles for data transfer. With 2 clocks of giving activate and read with auto-precharge commands, you get a continuous flow of output data.

Compare the following efficiency results in Figure 13-7 and Figure 13-8:

- DDR2 SDRAM bank interleave reads with no additive latency, CAS latency of 4, and burst length of 4 (Figure 13-7),

Number of active cycles of data transfer = 6.

Total number of cycles = 15

Efficiency = 40%


- DDR2 SDRAM bank interleave reads with additive latency of 3, CAS latency of 4, and burst length of 4 (Figure 13-8),

Number of active cycles of data transfer = 6.

Total number of cycles = 14

Efficiency = approximately 43%

The interleaving reads used with additive latency increases efficiency by approximately 3%.

 Additive latency improves the efficiency of back-to-back interleaved reads or writes, but not individual random reads or writes.

User-Controlled Refresh

The requirement to periodically refresh memory contents is normally handled by the memory controller; however, the **User Controlled Refresh** option allows you to determine when memory refresh occurs. With specific knowledge of traffic patterns, you can time the refresh operations so that they do not interrupt read or write operations, thus improving efficiency.

Frequency of Operation

Certain frequencies of operation give you the best possible latency based on the memory parameters. The memory parameters you specify through the parameter editor in the MegaWizard™ Plug-In Manager are converted to clock cycles and rounded up.

If you are using a memory device that has $t_{\text{RCD}} = 20$ ns and running the interface at 100 MHz, you get the following results:

- For full-rate implementation ($t_{\text{Ck}} = 10$ ns):
 t_{RCD} convert to clock cycle = $20/10 = 2$.
- For half rate implementation ($t_{\text{Ck}} = 20$ ns):
 t_{RCD} convert to clock cycle = $20/20 = 1$

This frequency and parameter combination is not easy to find because there are many memory parameters and frequencies for the memory device and the controller to run. Memory device parameters are optimal for the speed at which the device is designed to run, so you should run the device at that speed.

In most cases, the frequency and parameter combination is not optimal. If you are using a memory device that has $t_{\text{RCD}} = 20$ ns and running the interface at 133 MHz, you get the following results:

- For full-rate implementation ($t_{\text{Ck}} = 7.5$ ns):
 t_{RCD} convert to clock cycle = $20/7.5 = 2.66$, rounded up to 3 clock cycles or 22.5 ns.
- For half rate implementation ($t_{\text{Ck}} = 15$ ns):
 t_{RCD} convert to clock cycle = $20/15 = 1.33$, rounded up to 2 clock cycles or 30 ns.

There is no latency difference for this frequency and parameter combination.

Burst Length

Burst length affects the efficiency of the controller. A burst length of 8 provides more cycles of data transfer, compared to a burst length of 4.

For a half-rate design that has a command latency of 9 half-rate clock cycles, and a CAS latency of 3 memory clock cycles or 1.5 half rate local clock cycles, the efficiency is 9% for burst length of 4, and 16% for burst length of 8.

- Burst length of 4 (2 memory clock cycles of data transfer or 1 half-rate local clock cycle)

Efficiency = number of active cycles of data transfer / total number of cycles

Efficiency = $1/(1 + \text{CAS} + \text{command latency}) = 1/(1 + 1.5 + 9) = 1/11.5 = 8.6\%$ or approximately 9%

- Burst length of 8 (4 memory clock cycles of data transfer or 2 half-rate local clock cycles)

Efficiency = number of active cycles of data transfer/total number of cycles

Efficiency = $2/(2 + \text{CAS} + \text{command latency}) = 2/(2 + 1.5 + 9) = 2/12.5 = 16\%$

Series of Reads or Writes

Performing a series of reads or writes from the same bank and row increases controller efficiency.

The case shown in [Figure 13-2 on page 13-4](#) demonstrates that a read performed from the same row takes only 14.5 clock cycles to transfer data, making the controller 27% efficient.

Do not perform random reads or random writes. When you perform reads and writes to random locations, the operations require row and bank changes. To change banks, the controller must precharge the previous bank and activate the row in the new bank. Even if you change the row in the same bank, the controller has to close the bank (precharge) and reopen it again just to open a new row (activate). Because of the precharge and activate commands, efficiency decreases as the controller needs more time to issue a read or write.

If you must perform a random read or write, use additive latency and bank interleaving to increase efficiency.

Controller efficiency depends on the method of data transfer between the memory device and the FPGA, the memory standards specified by the memory device vendor, and the type of memory controller.

Bandwidth

Bandwidth depends on the efficiency of the memory controller controlling the data transfer to and from the memory device.

You can express bandwidth as follows:

Bandwidth = data width (bits) × data transfer rate (1/s) × efficiency

Data rate transfer (1/s) = $2 \times$ frequency of operation ($4 \times$ for QDR SRAM interfaces)

The following example shows the bandwidth calculation for a 16-bit interface that has 70% efficiency and runs at 200 MHz frequency:

Bandwidth = $16 \text{ bits} \times 2 \text{ clock edges} \times 200 \text{ MHz} \times 70\% = 4.48 \text{ Gbps}$.

DRAM typically has an efficiency of around 70%, but when you use the Altera® memory controller efficiency can vary from 10 to 92%.

In QDR II+ or QDR II SRAM the IP implements two separate unidirectional write and read data buses, so the data transfer rate is four times the clock rate. The data transfer rate for a 400-MHz interface is 1,600 Mbps. The efficiency is the percentage of time the data bus is transferring data. It is dependent on the type of memory. For example, in a QDR II+ or QDR II SRAM interface with separate write and read ports, the efficiency is 100% when there is an equal number of read and write operations on these memory interfaces.

Document Revision History

Table 13-1 lists the revision history for this document.

Table 13-1. Document Revision History

Date	Version	Changes
November 2011	2.0	Reorganized optimizing the controller information into an individual chapter.
June 2011	1.0	Initial release.