It is important that you understand how to increase the efficiency and bandwidth of the memory controller when you design any external memory interface.

The following topics discuss 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.

Controller efficiency is expressed as:

\[
\text{Efficiency} = \frac{\text{number of active cycles of data transfer}}{\text{total number of cycles}}
\]

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

**Note:** 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.
  \[
  \text{Efficiency} = \frac{1}{(1 + \text{CAS} + \text{command latency})} = \frac{1}{(1+1.5+9)} = \frac{1}{11.5} = 8.6\%
  \]

- **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.
  \[
  \text{Efficiency} = \frac{4}{(4 + \text{CAS} + \text{command latency})} = \frac{4}{(4+1.5+9)} = \frac{1}{14.5} = 27.5\%
  \]
Factors Affecting Efficiency

The two main factors that affect controller efficiency are the interface standard specified by the memory vendor, and the way that 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<sub>RCD</sub> 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<sub>RCD</sub> 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<sub>RCD</sub> 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 data sheet, these components require the following values:
    - t<sub>RP</sub> = 12 ns, 3 clock cycles for a 200-MHz operation (5 ns period for 200 MHz)
    - t<sub>RFC</sub> = 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 x 5 ns/7.8 us) of the total efficiency.

Bank Management Efficiency

The following figures show examples of how the bank management operations affect controller efficiency.

The first figure 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.

The following figure illustrates a read-after-write operation. The controller changes the row address after the write-to-read from a different row.
The following sequence of events describes the above figure:

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.

The following figure 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.
The following sequence of events describes the above figure:

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 the second figure above, 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 the first figure above.

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.

The following figure shows an example of changing the row in the same bank.
The following sequence of events describes the above figure:

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 tools and methods:

- DDR2 SDRAM Controller
- Auto-Precharge Commands
- Additive Latency
- Bank Interleaving
- Command Queue Look-Ahead Depth
- 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 a given read or write command is the last access to the currently opened row.
The memory device automatically closes or auto-precharges the page that is currently being accessed, so that the next access to the same bank is faster. The Auto-Precharge command is useful when you want to perform fast random memory accesses.

The Timer Bank Pool (TBP) block supports the dynamic page policy, where depending on user input on local autoprecharge input would keep a page open or close. In a closed-page policy, a page is always closed after it is accessed with auto-precharge command. When the data pattern consists of repeated reads or writes to addresses not within the same page, the optimal system achieves the maximum efficiency allowed by continuous page miss limited access. Efficiency losses are limited to those associated with activating and refreshing. An efficiency of 10-20% should be expected for this closed-page policy.

In an open-page policy, the page remains open after it is accessed for incoming commands. When the data pattern consists of repeated reads or writes to sequential addresses within the same page, the optimal system can achieve 100% efficiency for page-open transactions (ignoring the effects of periodic refreshes, which typically consume around 2-3% of total efficiency), with minimum latency for highest priority single transactions.

If you turn on Enable Auto-Precharge Control, you can instruct the controller to issue an autoprecharge read or write command. The next time you access that bank, the access will be faster because the controller does not have to precharge the bank before activating the row that you want to access.

The controller-derived autoprecharge logic evaluates the pending commands in the command buffer and determines the most efficient autoprecharge operation to perform. The autoprecharge logic can reorder commands if necessary. When all TBP are occupied due to tracking an open page, TBP uses a scheme called on-demand flush, where it stops tracking a page to create space for an incoming command.

The following figure compares auto-precharge with and without look-ahead support.

**Figure 13-4: Comparison With and Without Look-ahead Auto-Precharge**

Without using the look-ahead auto-precharge feature, the controller must precharge to close and then open the row before the write or read burst for every row change. When using the look-ahead precharge feature, the controller decides whether to do auto-precharge read/write by evaluating the incoming command; subsequent reads or writes to same bank/different row will require only an activate command.
As shown in the preceding figure, the controller performs an auto-precharge for the write command to bank 0 at cycle 1. The controller detects that the next write at cycle 13 is to a different row in bank 0, and hence saves 2 data cycles.

The following efficiency results apply to the above figure:

**Table 13-1: Comparative Efficiencies With and Without Look-Ahead Auto-Precharge Feature**

<table>
<thead>
<tr>
<th></th>
<th>Without Look-ahead Auto-precharge</th>
<th>With Look-ahead Auto-precharge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active cycles of data transfer</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Total number of cycles</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>Approximate efficiency</td>
<td>84%</td>
<td>94%</td>
</tr>
</tbody>
</table>

The look-ahead auto-precharge used increases efficiency by approximately 10%.

The following figure shows how you can improve controller efficiency using the auto-precharge command.

**Figure 13-5: Improving Efficiency Using Auto-Precharge Command**

The following sequence of events describes the above figure:

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 the "data Transfer" topic).

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.

The following figure shows an example of additive latency.

**Figure 13-6: Additive Latency—Read**

The following sequence of events describes the above figure:

1. The controller issues a read or write command before the \( t_{RCD} \) (MIN) requirement—additive latency less than or equal to \( t_{RCD} \) (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.

Read latency = additive latency + CAS latency

Write latency = additive latency + CAS latency - \( t_{CK} \)

**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.

**Note:** Page size refers to the minimum number of column locations on any row that you access with a single activate command. For example: For a 512Mb x8 DDR3 SDRAM with 1024 column locations (column address A[9:0]), page size = 1024 columns x 8 = 8192 bits = 8192/8 bytes = 1024 bytes (1 KB)

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

Interleaving allows banks of the SDRAM device to alternate their background operations and access cycles. One bank undergoes its precharge/activate cycle while another is being accessed. By alternating banks, the controller improves its performance by masking the precharge/activate 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.

The following figure shows how you can use interleaving to increase bandwidth.

**Figure 13-7: Using Interleaving to Increase Bandwidth**

The Altera controller supports three interleaving options:

**Chip-Bank-Row-Col** – This is a noninterleaved option. Select this option to improve efficiency with random traffic.

**Chip-Row-Bank-Col** – This option uses bank interleaving without chip select interleaving. Select this option to improve efficiency with sequential traffic, by spreading smaller data structures across all banks in a chip.

**Row-Chip-Bank-Col** - This option uses bank interleaving with chip select interleaving. Select this option to improve efficiency with sequential traffic and multiple chip selects. This option allows smaller data structures to spread across multiple banks and chips.

Bank interleaving is a fixed pattern of data transactions, enabling best-case bandwidth and latency, and allowing for sufficient interleaved transactions between opening banks to completely hide $t_{RC}$. An optimal system can achieve 100% efficiency for bank interleave transactions with 8 banks. A system with less than 8 banks is unlikely to achieve 100%.

**Command Queue Look-Ahead Depth**

The command queue look-ahead depth value determines the number of read or write requests that the look-ahead bank management logic examines. The command queue look-ahead depth value also determines how many open pages the High-Performance Controller II (HPC II) can track.
For example, if you set the command queue look-ahead depth value to 4, the HPC II controller can track 4 open pages. In a 4-bank interleaving case, HPCII will receive repeated commands with addresses of bank A, bank B, bank C, and bank D. To receive the next set of commands, the controller issues a precharge command to exit the current page and then issues an activate command to track the new incoming page, leading to a drop in efficiency.

Figure 13-8: Simulation with Command Queue Look-ahead Depth of 4

With the command queue look-ahead depth set to 8, the controller can track 8 open pages and overall efficiency is much improved relative to a command queue look-ahead value of 4.

Figure 13-9: Simulation with Command Queue Look-ahead Depth of 8

There is a trade-off between efficiency and resource usage. Higher command queue look-ahead values are likely to increase bank management efficiency, but at the cost of higher resource usage. Smaller command queue look-ahead values may be less efficient, but also consume fewer resources. Also, a command queue look-ahead value greater than 4 may cause timing violations for interfaces approaching their maximum frequency.

Note: If you set Command Queue Look-ahead depth to a value greater than 4, you may not be able to run the interface at maximum frequency.

To achieve an optimized balance of controller efficiency versus resource usage and frequency, you must understand your traffic patterns. You should simulate your design with a variety of controller settings to observe the results of different settings.

Note: User-selectable Command Queue Look-ahead depth is available only when using the soft memory controller. For the hard memory controller, the Command Queue Look-ahead depth value is hard-coded to 8.

Additive Latency and Bank Interleaving

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

The following figure shows an example of bank interleaving in a read operation without additive latency. The example uses DDR2 SDRAM bank interleave reads with CAS latency of 4, and burst length of 4.
The following sequence of events describes the above figure:

1. The controller issues an activate command to open the bank, which activates bank x and the row in it.
2. After t\textsubscript{RCD} time, the controller issues a read with auto-precharge command to the specified bank.
3. Bank y receives an activate command after t\textsubscript{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.

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

The following figure shows an example of bank interleaving in a read operation with additive latency. The example uses DDR2 SDRAM bank interleave reads with additive latency of 3, CAS latency of 4, and burst length of 4. In this configuration, the controller issues back-to-back activate and read with auto-precharge commands.

The following sequence of events describes the above figure:

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\textsubscript{RCD} time.
3. The controller executes the read with auto-precharge command t\textsubscript{RCD} time later on the rising edge T4.
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 efficiency results in the two preceding figures:

- DDR2 SDRAM bank interleave reads with no additive latency, CAS latency of 4, and burst length of 4 (first figure),
  
  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 (second figure),
  
  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%.

**Note:** 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.

**Note:** If you enable the auto-precharge control, you must ensure that the average periodic refresh requirement is met, because the controller does not issue any refreshes until you instruct it to.

**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 are converted to clock cycles and rounded up.

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

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

- For half rate implementation ($t_{CK} = 20$ 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 \( t_{RCD} = 20 \, \text{ns} \) and running the interface at 133 MHz, you get the following results:

- For full-rate implementation (\( t_{Ck} = 7.5 \, \text{ns} \)):
  \[ t_{RCD} \text{ convert to clock cycle} = \frac{20}{7.5} = 2.66, \text{ rounded up to 3 clock cycles or 22.5 ns} \]

- For half rate implementation (\( t_{Ck} = 15 \, \text{ns} \)):
  \[ t_{RCD} \text{ convert to clock cycle} = \frac{20}{15} = 1.33, \text{ rounded up to 2 clock cycles or 30 ns} \]

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)
  \[ \text{Efficiency} = \frac{\text{number of active cycles of data transfer}}{\text{total number of cycles}} \]
  \[ \text{Efficiency} = \frac{1}{1 + \text{CAS} + \text{command latency}} = \frac{1}{1 + 1.5 + 9} = \frac{1}{11.5} = 8.6\% \text{ or approximately 9\%} \]

- Burst length of 8 (4 memory clock cycles of data transfer or 2 half-rate local clock cycles)
  \[ \text{Efficiency} = \frac{\text{number of active cycles of data transfer}}{\text{total number of cycles}} \]
  \[ \text{Efficiency} = \frac{2}{2 + \text{CAS} + \text{command latency}} = \frac{2}{2 + 1.5 + 9} = \frac{2}{12.5} = 16\% \]

**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 the second figure in the "Bank Management Efficiency" topic 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 can decrease by as much as 3–15%, 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.

**Data Reordering**

Data reordering and command reordering can both contribute towards achieving controller efficiency.
The Data Reordering feature allows the single-port memory controller to change the order of read and write commands to achieve highest efficiency. You can enable data reordering by turning on **Enable Reordering** on the **Controller Settings** tab of the parameter editor.

In the soft memory controller, inter-bank data reordering serves to minimize bus turnaround time by optimizing the ordering of read and write commands going to different banks; commands going to the same bank address are not reordered.

**Figure 13-12: Data Reordering for Minimum Bus Turnaround**

![Data Reordering Diagram](image)

In the hard memory controller, inter-row data reordering serves to minimize $t_{RC}$ by reordering commands going to different bank and row addresses; command going to the same bank and row address are not reordered. Inter-row data reordering inherits the minimum bus turnaround time benefit from inter-bank data reordering.

**Figure 13-13: Data Reordering for Minimum $t_{RC}$**

![Data Reordering Diagram](image)

**Starvation Control**

The controller implements a starvation counter to ensure that lower-priority requests are not forgotten as higher-priority requests are reordered for efficiency.

In starvation control, a counter is incremented for every command served. You can set a starvation limit, to ensure that a waiting command is served immediately upon the starvation counter reaching the specified limit.

For example, if you set a starvation limit of 10, a lower-priority command will be treated as high priority and served immediately, after ten other commands are served before it.

**Command Reordering**

Data reordering and command reordering can both contribute towards achieving controller efficiency.
DDR protocols are naturally inefficient, because commands are fetched and processed sequentially. The DDRx command and DQ bus are not fully utilized as few potential cycles are wasted and degrading the efficiency.

The command reordering feature, or look-ahead bank management feature, allows the controller to issue bank management commands early based on incoming patterns, so that when the command reaches the memory interface, the desired page in memory is already open.

The command cycles during the $t_{RCD}$ period are idle and the bank-management commands are issued to next access banks. When the controller is serving the next command, the bank is already precharged. The command queue look-ahead depth is configurable from 1-16, to specify how many read or write requests the look-ahead bank management logic examines. With the look-ahead command queue, if consecutive write or read requests are to a sequential address with same row, same bank, and column incremental by 1, the controller merges the write or read requests at the memory transaction into a single burst.

Figure 13-14: Comparison With and Without Look-Ahead Bank Management Feature

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Command</th>
<th>Data</th>
<th>Cycle</th>
<th>Command</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACT</td>
<td></td>
<td>1</td>
<td>ACT</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NOP</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>NOP</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>READ</td>
<td></td>
<td>4</td>
<td>READ</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>DATA0 (Burst 0, Burst 1)</td>
<td></td>
<td>5</td>
<td>ACT</td>
<td>DATA0 (Burst 0, Burst 1)</td>
</tr>
<tr>
<td>6</td>
<td>DATA0 (Burst 2, Burst 3)</td>
<td></td>
<td>6</td>
<td>NOP</td>
<td>DATA0 (Burst 2, Burst 3)</td>
</tr>
<tr>
<td>7</td>
<td>ACT</td>
<td></td>
<td>7</td>
<td>ACT</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>DATA0 (Burst 4, Burst 5)</td>
<td></td>
<td>8</td>
<td>READ</td>
<td>DATA0 (Burst 4, Burst 5)</td>
</tr>
<tr>
<td>9</td>
<td>NOP</td>
<td></td>
<td>9</td>
<td>NOP</td>
<td>DATA0 (Burst 6, Burst 7)</td>
</tr>
<tr>
<td>10</td>
<td>READ</td>
<td></td>
<td>10</td>
<td>NOP</td>
<td>DATA0 (Burst 6, Burst 7)</td>
</tr>
<tr>
<td>11</td>
<td>DATA0 (Burst 0, Burst 1)</td>
<td></td>
<td>11</td>
<td>NOP</td>
<td>DATA0 (Burst 0, Burst 1)</td>
</tr>
<tr>
<td>12</td>
<td>DATA0 (Burst 2, Burst 3)</td>
<td></td>
<td>12</td>
<td>READ</td>
<td>DATA0 (Burst 2, Burst 3)</td>
</tr>
<tr>
<td>13</td>
<td>ACT</td>
<td></td>
<td>13</td>
<td>ACT</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>DATA0 (Burst 4, Burst 5)</td>
<td></td>
<td>14</td>
<td>NOP</td>
<td>DATA0 (Burst 4, Burst 5)</td>
</tr>
<tr>
<td>15</td>
<td>NOP</td>
<td></td>
<td>15</td>
<td>NOP</td>
<td>DATA0 (Burst 5, Burst 7)</td>
</tr>
<tr>
<td>16</td>
<td>READ</td>
<td></td>
<td>16</td>
<td>NOP</td>
<td>DATA0 (Burst 5, Burst 7)</td>
</tr>
<tr>
<td>17</td>
<td>DATA0 (Burst 0, Burst 1)</td>
<td></td>
<td>17</td>
<td>NOP</td>
<td>DATA0 (Burst 0, Burst 1)</td>
</tr>
<tr>
<td>18</td>
<td>DATA0 (Burst 2, Burst 3)</td>
<td></td>
<td>18</td>
<td>NOP</td>
<td>DATA0 (Burst 2, Burst 3)</td>
</tr>
<tr>
<td>19</td>
<td>DATA0 (Burst 4, Burst 5)</td>
<td></td>
<td>19</td>
<td>NOP</td>
<td>DATA0 (Burst 4, Burst 5)</td>
</tr>
<tr>
<td>20</td>
<td>DATA0 (Burst 6, Burst 7)</td>
<td></td>
<td>20</td>
<td>NOP</td>
<td>DATA0 (Burst 6, Burst 7)</td>
</tr>
</tbody>
</table>

Compare the following efficiency results for the above figure:

**Table 13-2: Efficiency Results for Above Figure**

<table>
<thead>
<tr>
<th></th>
<th>Without Look-ahead Bank Management</th>
<th>With Look-ahead Bank Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active cycles of data transfer</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Total number of cycles</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>Approximate efficiency</td>
<td>60%</td>
<td>75%</td>
</tr>
</tbody>
</table>
In the above table, the use of look-ahead bank management increases efficiency by 15%. The bank look-ahead pattern verifies that the system is able to completely hide the bank precharge and activation for specific sequences in which the minimum number of page-open transactions are placed between transactions to closed pages to allow bank look-ahead to occur just in time for the closed pages. An optimal system would completely hide bank activation and precharge performance penalties for the bank look-ahead traffic pattern and achieve 100% efficiency, ignoring refresh.

**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:

\[
\text{Bandwidth} = \text{data width (bits)} \times \text{data transfer rate (1/s)} \times \text{efficiency}
\]

Data rate transfer (1/s) = 2 × frequency of operation (4 × 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:

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

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.

For information on best-case and worst-case efficiency scenarios, refer to the white paper, *The Efficiency of the DDR & DDR2 SDRAM Controller Compiler*.

**Related Information**

*The Efficiency of the DDR & DDR2 SDRAM Controller Compiler*

**Efficiency Monitor**

The Efficiency Monitor and Protocol Checker measures traffic efficiency on the Avalon interface between the traffic generator and the controller, and checks that the Avalon protocol is not violated.

The Protocol Checker monitors the controller’s Avalon slave interface for any illegal commands presented to it by any master; it does not monitor the legality of the controller’s Avalon outputs.
To enable efficiency measurements to be performed on the controller Avalon interface through UniPHY External Memory Interface Toolkit, you need to check on Enable the Efficiency Monitor and Protocol Checker on the Controller Avalon Interface.

Note: The efficiency monitor does not take refreshes into account.

The Efficiency Monitor counts the number of cycles of command transfers and wait times for the controller interface and provides an Avalon slave port to allow access to this data. The efficiency monitor has an internal 32-bit counter for accessing transactions; its status can be any of the following:

- Not Running
- Not Running: Waiting for pattern start
- Running
- Not Running: Counter Saturation

For example, once the counter saturates the efficiency monitor stops because it can no longer track transactions. In the summary panel, this appears as Not Running: Counter Saturation.

The debug toolkit summarizes efficiency monitor statistics as follows:

- **Efficiency Monitor Cycle Count** – counts cycles from first command/start until $2^{32}$ or a stop request
- **Transfer Count** – counts any data transfer cycle, read or write
- **Write Count** – counts how many writes requested, including those during a burst
- **Read Count** – counts how many reads requested (just commands)
- **Read Burst counter** – counts how many reads requested (total burst requests)
- **Non-Transfer Cycle Waitrequest Count** – counts Non Transfer Cycles due to slave wait request high. A Non-Transfer Cycle is a cycle during which no read data is received and no write data is accepted on the Avalon interface.
- **Non-Transfer Cycle No readdatavalid Count** – counts Non Transfer Cycles due to slave not having read data
- **Non-Transfer Cycle Master Write Idle Count** – counts Non Transfer Cycles due master not issuing command or pause in write burst
- **Non-Transfer Cycle Master Idle Count** – counts Non Transfer Cycles due master not issuing command anytime
- **System Efficiency** – The percentage of all Avalon-MM cycles where the interface is transferring data. Refreshes and idle time are not taken into consideration when calculating efficiency.
• **System Efficiency (During user access)** – Tracks the efficiency when transactions are occurring, which is a reflection on waitrequest. It is defined as: \( \frac{\text{Transfer Count}}{\text{(Efficiency Monitor Cycle Count - Non-Transfer Cycle Master Idle Count)}} \)

• **Minimum Read Latency** – The lowest of all read latencies, which is measured by time between a read command is being accepted by the Controller till the first beat of read data is presented to the driver.

• **Maximum Read Latency** – The highest of all read latencies, which is measured by time between a read command is being accepted by the Controller till the first beat of read data is presented to the driver.

• **Average Read Latency** – The average of all read latencies, which is measured by time between a read command is being accepted by the Controller till the first beat of read data is presented to the driver.