DMA on Demand
Operating Guide

797 North Grove Rd, Suite 101
Richardson, TX 75081
Phone: (972) 671-9570
www.redrapids.com

Red Rapids reserves the right to alter product specifications or discontinue any product without notice. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment. This product is not designed, authorized, or warranted for use in a life-support system or other critical application.

All trademark and registered trademarks are the property of their respective owners.

Table of Contents

List of Figures

1.0 Introduction

1.1 Contents and Structure

This manual describes the operation of the DMA on Demand function that is embedded in all Red Rapids PCI adapter products. High speed DMA transactions are critical to sustaining the data rate required by wideband signal acquisition and generation products.

The latest product documentation and software is available for download from the Red Rapids web site (www.redrapids.com).

1.2 Conventions

This manual uses the following conventions:

☞	Text in this format highlights useful or important information.

!	Text shown in this format is a warning. It describes a situation that could potentially damage your equipment. Please read each warning carefully.

The following are some of the acronyms used in this manual.

- **API** Application Programming Interface
- **CPU** Central Processing Unit
- **DMA** Direct Memory Access
- **FIFO** First-in / First-out

1.3 Revision History

Version	Date	Description
R00	3/21/2014	Initial release.

2.0 Background

It would be impossible to sustain high data rate transfers through computer peripherals if the central processing unit (CPU) had to manage these transactions. It is far more efficient to allow direct memory access (DMA) so that data transfers are orchestrated by the attached device, independent of the CPU or any software intervention. This can be a difficult concept to grasp. Software applications typically have no exposure to physical memory addresses, those hardware details are abstracted to a virtual address space. Fortunately, the driver supplied with the device acts as a bridge between the physical and virtual domains.

It helps to understand some basic computer architecture concepts to appreciate the power and limitations of DMA transfers. Figure 2-1 illustrates a very basic computer architecture consisting of a CPU, main memory, and bus connected devices. The challenge is to move data between the devices and main memory without passing through the CPU. The memory-mapped address space of the system is flat, eliminating any potential ambiguity about source or destination locations.

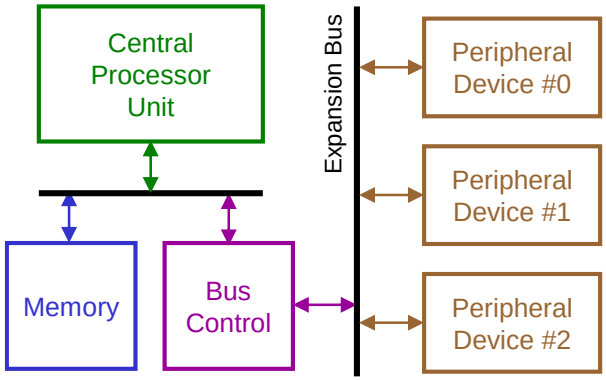

Figure 2-1 Basic Computer Architecture

To initiate a DMA transfer, the device must become a bus master. In other words, the device must make a hardware request to assume control of the bus. When control of the bus is granted, the device needs only a source or destination address to complete a read or write transfer. It is important to note that the source or destination does not have to be located in main memory. It can be any address in the system memory map, making it possible to perform DMA transfers between devices.

DMA transfers typically involve more than just a single word of data. It is much more efficient to transfer a large block of data in a single burst to mitigate the overhead incurred each time the bus is requested. A burst transfer always begins with a single source or destination address. The length of the transfer is accommodated by simply accessing subsequent addresses in sequence from the original source or destination address. This is a very important point since it implies that the device must have access to a continuous segment of addresses to complete a burst.

2.1 Device Driver

The device driver plays a key role in the DMA process. Before a DMA transaction to main memory can begin, a protected segment of address space must be allocated by the operating system. Imagine what would happen if a device began writing to unprotected memory space. The operating system would assume that those addresses are available for normal operation and the presence of foreign data would be interpreted as a hardware fault.

Application software must call the driver to allocate the desired DMA page size in main memory. The driver responds with two addresses for each page of memory allocated. The first is a physical address required by the device to perform transfers over the bus. The second is a virtual address that can be used by software to access the contents of the buffer from the application.

Keep in mind that the entire block must occupy contiguous memory. So it is possible that the operating system may not even have a large enough block of contiguous memory available to accommodate the request. Some operating systems even place a limit on the maximum contiguous address space that can be allocated.

2.2 Scatter Gather

Many devices employ a scatter gather technique to link a series of DMA pages in host memory. The complete DMA buffer is scattered across several pages and then gathered by the application code through indexing. This technique usually employs a DMA descriptor that is also stored in host memory to convey the size of each DMA page and the pointers necessary to index the pages as a single circular buffer as shown in Figure 2-2. The advantage of this approach is that there is no limit to the number of DMA pages that can be created so the size of each page can be kept small. This reduces the chance that the operating system will fail to find the requested contiguous segment.

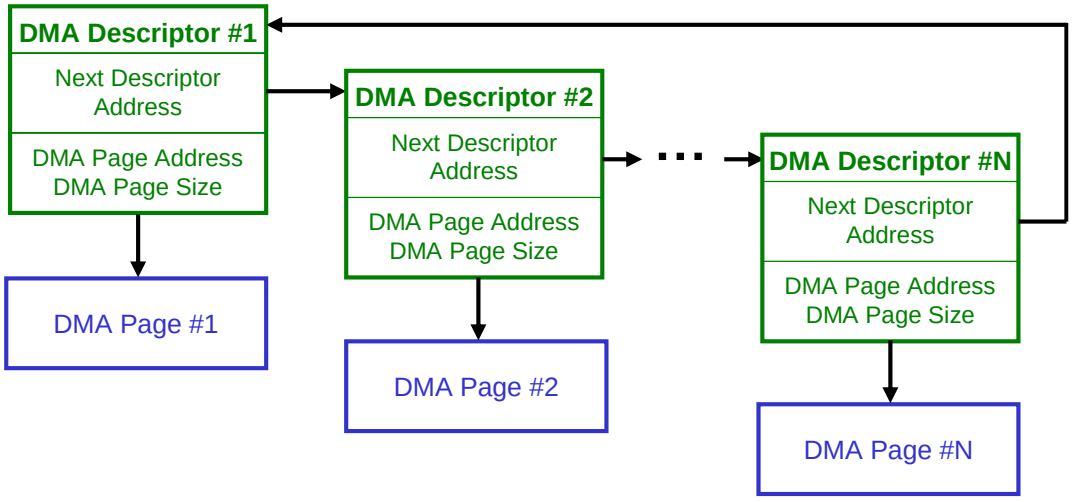

Figure 2-2 Scatter Gather DMA Indexing

The device must access each page descriptor to obtain the source or destination of the transaction. The device must be told where to find the first descriptor, it can then operate autonomously since each descriptor points to the next.

2.3 Status Reporting

There needs to be some mechanism for a device to inform the software application that a DMA buffer requires service. This notification must occur in a timely matter to avoid losing the information as the device continuously cycles through all of the available blocks. As bus master, the device is the sole source of current status regarding the DMA progress. There are two generally accepted techniques to convey these updates, interrupts and polling.

Interrupts can be generated by a device to inform the CPU of status updates or errors that require attention. In the case of DMA transfers, an interrupt can be issued when some

number of blocks require service. Of course the interrupt must be identified by the CPU and routed to the appropriate service routine. Unfortunately, that process takes time and can be unpredictable without real-time extensions to the operating system. High rate data transfers may require very large buffers to help overcome this latency.

The alternative to interrupts is polling by the application software. In other words, the application simply queries the device for status on some regular interval. This is a less elegant approach since the polling operation consumes CPU and bus resources when there may be no change in status. However, the impact can be minimized by smart scheduling of the polling interval based on the expected DMA buffer refresh rate.

2.4 Write Transaction

A bus master DMA write transaction occurs when data is transferred from the device to main memory. As bus master, the device is essentially writing data into the designated buffer. Once a write transaction is initiated, it must be able to complete uninterrupted through the length of the burst. As shown in Figure 2-3, a hardware FIFO in the device is usually employed to ensure that there are no gaps in the data. The FIFO depth needs to be greater than the length of a single burst so that the device can continue to accept new input data while waiting for access to the bus.

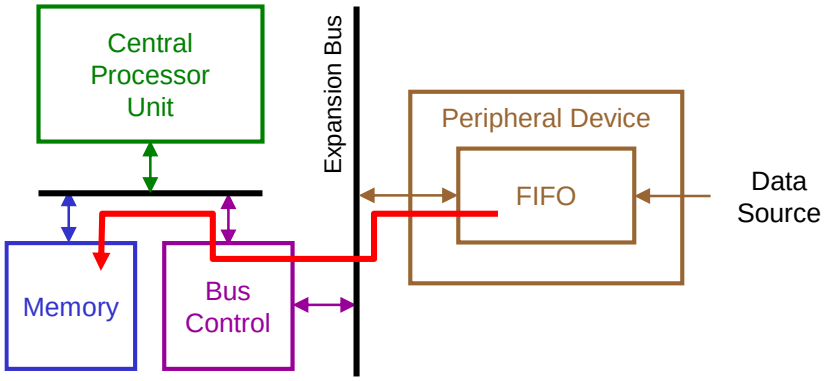

Figure 2-3 Bus Master DMA Write Transaction

It is important to note that increasing the size of the FIFO can overcome the latency incurred negotiating acces to the bus, but it will not compensate for a lack of sufficient bus bandwidth. If new data is arriving at the FIFO at a rate faster than the bus can sustain, adding FIFO capacity simply delays the inevitable overflow that will occur. If the bus is just slow to grant access between transfers, a large FIFO provides storage for the new arriving data.

2.5 Read Transaction

A bus master DMA read transaction occurs when data is transferred from main memory to the device. As bus master, the device is essentially reading data from the designated buffer. Once a read transaction is initiated, it must be able to complete uninterrupted through the length of the burst. Some type of memory in the device is usually used to store the incoming data stream. As shown in Figure 2-4, a FIFO is frequently selected to fulfill this role. The FIFO depth needs to be greater than the length of a single burst so that the device does not run out of data samples to process while waiting for the next DMA transaction.

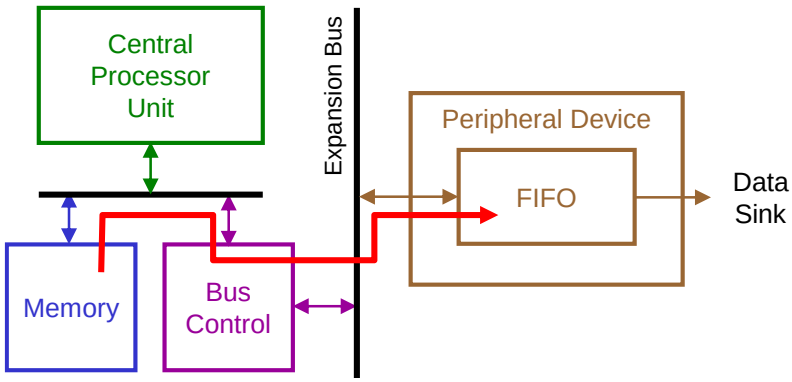

Figure 2-4 Bus Master DMA Read Transaction

It is important to note that increasing the size of the FIFO can overcome the latency incurred negotiating acces to the bus, but it will not compensate for a lack of sufficient bus bandwidth. If new data is removed from the FIFO at a rate faster than the bus can sustain, adding FIFO capacity simply delays the inevitable underflow that will occur. If the bus is simply slow to grant access between transfers, a large FIFO acts as a buffer to supply samples to the device.

2.6 Multi-Channel

It is possible for a device to operate more than one DMA channel to memory. This is a convenient approach to keep unique data streams separated in memory. Each DMA channel operates independently with a unique set of pages allocated by the host. There is really no limit to the number of channels that can be operated by a single device, but the complexity of tracking the progress of multiple channels introduces practical considerations.

3.0 DMA on Demand

Most Red Rapids products are capable of generating or consuming data at extremely high rates. It is impractical to rely on a DMA descriptor stored in host memory to meet the demands of real-time signal acquisition or generation when there is no way to predict how quickly the host may grant access to this key information. Instead, each device includes on-board descriptor storage for up to 1024 pages per DMA buffer. This strategy allows the device to initiate a DMA transaction immediately when the need arises (DMA on Demand) while still providing enough descriptors to keep individual page size manageable.

The software API defines the size of a DMA page in relation to a burst size and burst count as shown in Figure 32. The burst size is the number of bytes that will be transferred in an individual DMA transaction. The burst count is the number of these transactions that can be performed on a single page. Consequently, the page size in bytes is simply the burst size multiplied by the burst count. Each DMA page is 64-bit aligned, so the burst size must be a multiple of eight bytes.

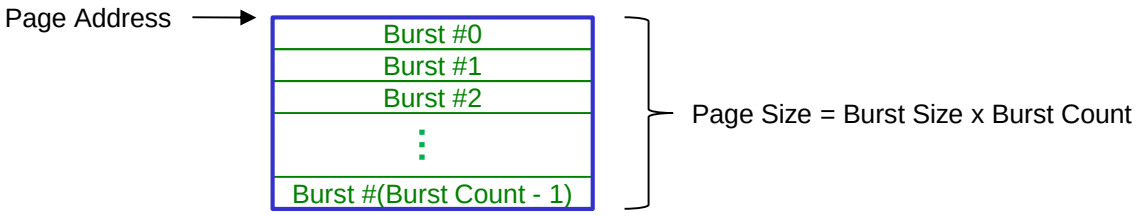

Figure 3-1 DMA Page Size Parameters

The burst size and burst count can vary with each page, offering maximum flexibility in creating the overall DMA buffer. In addition to the size parameters, there are two flags that can be set in each page descriptor. The “final page” flag indicates that the current page is the last in the circular buffer. This allows an application to use less than the full 1024 pages available to the hardware. A marker flag is set when the controller has completed pending operations on the designated page.

The DMA on Demand controller continuously sequences through all of the specified addresses, creating a circular buffer as shown in Figure 3-2. The operation is identical to traditional scatter gather techniques with the exception that DMA descriptor information is stored on the device.

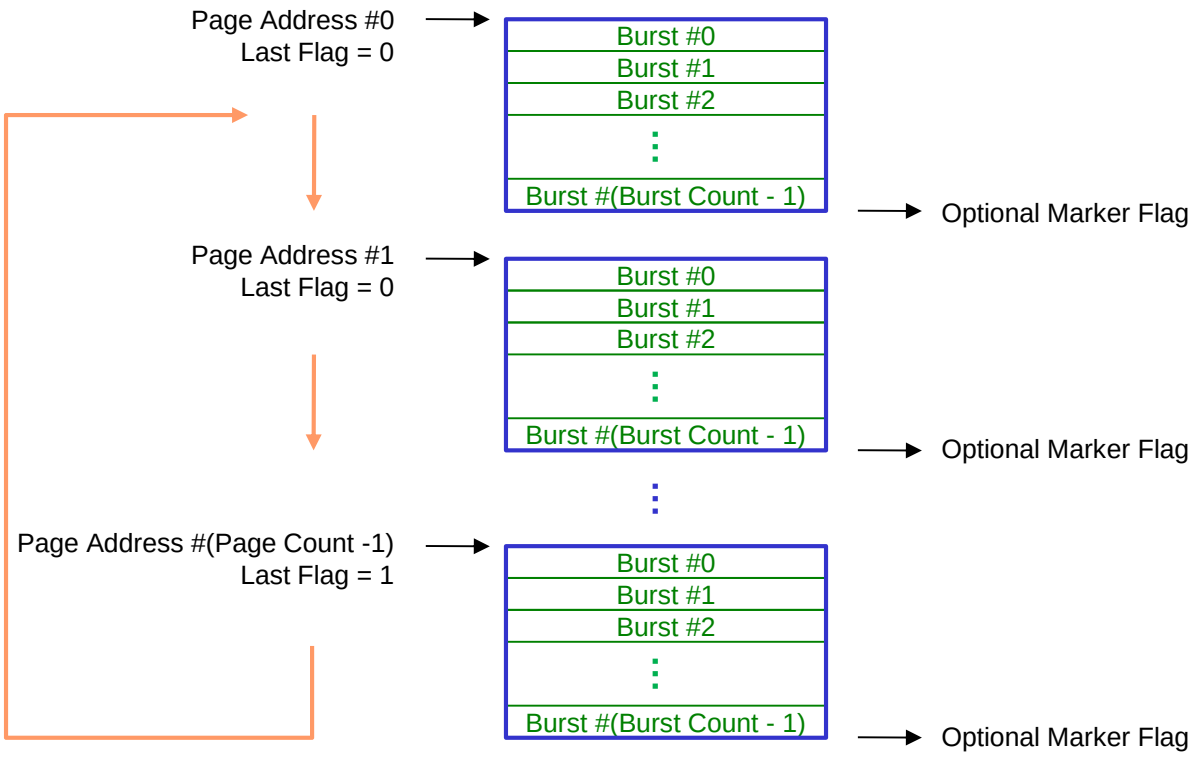

Figure 3-2 Scatter Gather DMA Sequence

The DMA on Demand controller also provides status information that can be accessed through registers in the reference designs. The current page and current burst status can be used to track DMA progress without interrupts or used to query the current state when an interrupt is received by the application.

3.1 Burst Size

The burst size setting is simply the number of bytes that will be transferred by the device in each DMA bus transaction. It must be set to a multiple of eight bytes to keep the buffer 64-bit aligned. The default value of 4096 bytes should be sufficient in most applications. There is a common misconception that a burst will proceed uninterrupted from start to finish. In reality, the host computer is very likely to break up the burst to accommodate resource limitations inherent to the hardware. The maximum burst size allowed is 16,376 bytes (2^{14}-8).

Keep in mind that the burst size will impact the effectiveness of the device FIFO. The device cannot initiate a DMA transaction until the FIFO can accommodate a complete burst. As the burst size increases, a greater fraction of the FIFO must be committed to a single transaction. The primary purpose of the FIFO is to act as a data buffer between DMA transactions, so larger burst sizes reduce this buffering capacity.

Consider the case of a device that is streaming data from an external source to main memory. If the FIFO is 16 kbytes deep and the burst size is set to 4 kbytes, then there is 12 kbytes of buffering available to hold incoming samples while the device waits for access to the bus. If the burst size is increased to 8 kbytes, then the buffer is reduced to 8 kbytes. This size of the buffer equates directly to time if the incoming data stream is continous.

3.2 Burst Count

The burst count defines the size of each page in relation to the burst size:

Page Size (bytes) = Burst Count x Burst Size (bytes)

Each DMA page must occupy contiguous address space in main memory. The descriptor provides the starting address of each page, leaving the DMA on Demand controller the responsibility for sequencing through the remaining addresses within a page. As mentioned earlier, it can become impossible for the operating system to allocate large contiguous space as other processes consume resources and fragment the memory.

Performance does not suffer switching between pages, so it is best to select a greater number of small pages over fewer large pages. A burst count of 128 will produce a half megabtye page size when the burst size is set to 4 kbytes. Very few applications will require anything larger.

3.3 Page Count

The page count defines the size of the overall DMA buffer in relation to the page size:

Buffer Size (bytes) = Page Count x Page Size (bytes)

If the maximum page count of 1024 is applied to the half megabyte page size calculated in the previous section, a half gigabyte buffer will be created. The adapter driver imposes a maximum DMA buffer size of 512 Mbytes on 32-bit operating systems and 1 Gbyte on 64-bit operating systems.

3.4 Marker Flag

The marker flag is used to set a hardware register status bit that can be queried by software to track DMA progress. The status bit can also be programmed to issue an interrupt when it is set. There is one marker status bit per DMA channel.

The number of pages between makers will determine how quickly the application must respond to DMA service requests. Consider the case of a device writing data to the DMA buffer at 200 Mbytes/sec. It will take 2.6 msec to fill a half megabtye page. If the marker flags are set every five pages, then the application software will have 13 msec to respond to each marker.

It is generally bad practice to set a single marker at the last page of a DMA buffer. This leaves the application virtually no time to service the buffer before the device cycles back to the first page. A classic "ping pong" buffer can be created by setting just two markers; one midway through the buffer and the other on the last page. This approach is sufficient in many applications. However, there may be other factors that make it convenient to set the markers more frequently. For instance, three markers per buffer make it possible to recover if any one marker is missed by the application.

3.5 Controller Status

The application software can query the device for the current page and current burst that the DMA controller is processing. These values are zero-based, so the first page and first burst are assigned an index of zero. Subsequent bursts are indexed sequentially up to the final burst that is assigned an index of Burst Count minus one. Likewise, pages are indexed sequentially up to a final index of Page Count minus one.

The current page should be retrieved each time a marker is detected to verify that pages are not missed. It is also possible to write an application that just polls the page and burst status directly, completely ignoring markers.

Operational status is also available from the DMA controller. The application can query the device to determine whether a channel is currently enabled for DMA transfers and whether there is an active DMA transaction in progress.

3.6 Controller Operation

The DMA controller emerges from reset ready to process the first burst in the first page of the DMA buffer. More specifically, the current burst and current page index are set to zero. Two actions are necessary before the first transaction can begin. First, the controller must be enabled. The enable command simply arms the controller, it cannot initiate a transaction. The second action, a DMA request, informs the controller that the hadware is prepared to process a complete burst of data. It is this command that instructs the controller to begin a transaction. The controller will continuously sequence through pages in the buffer until it has been disabled and any pending transactions are complete.

The controller can be programmed to either maintain the current burst and current page state when it is disabled, or reset both indexes to zero. The desired response usually depends on the structure of the application software. It may be more convenient to begin processing data from the start of the buffer each time the controller is enabled rather than continue from where the previous transaction concluded. The initialization command is used to make the selection.

4.0 FIFO Operation

The DMA controller typically operates in tandem with a FIFO to process bursts of data. The FIFO issues a transaction request to the DMA controller when there is enough capacity available to process a complete burst. However, it is unrealistic to expect every operating scenario to end on a precise burst boundary. There may be external triggers or other practical considerations that result in data residue left in the FIFO. The disposition of this residue is managed through FIFO commands.

As a simple example, consider the case of an analog to digital converter (ADC) that is triggered externally to capture a snapshot of data on a repeating interval. This scenario could apply to a radar or TDM communications waveform. The start trigger begins streaming samples into an empty FIFO. The FIFO will send a transaction request to the DMA controller as soon as enough samples are collected to complete a burst. This process will continue until the stop trigger is detected. At that point, the DMA controller will continue to drain the FIFO until there are no longer enough sample remaining to complete a burst. However, that does not mean that all of the samples collected have been extracted from the FIFO. There may be only a few sample remaining or enough to fall just short of completing another burst. It is up to the application to determine the fate of those remaining samples.

The same problem exists for a digital to analog (DAC) converter that consumes samples from a DMA buffer. The DMA controller must first prime the FIFO with data before the start trigger arrives so that samples are immediately available to the converter. The DMA controller will continuously fill the FIFO as long as there is space available to accept another burst. When the stop trigger shuts off the converter, the remaining samples will be stranded in the FIFO.

4.1 FIFO Reset

Resetting the FIFO is the easiest way to deal with data residue, but it may not always be a viable option. Clearing the FIFO guarantees that no two independent data segments can ever occupy the same burst in the DMA buffer. This eliminates the task of counting samples to determine where one data segment ends and another begins.

Resetting the FIFO obviously results in the loss of data, but this problem can be overcome by simply processing more samples than needed. Allowing just one extra burst of data will guarantee that all of the required samples are transferred by the DMA controller.

4.2 FIFO Hold

Holding the FIFO data contents will simply delay residue processing until the channel becomes active again.

If the FIFO is processing DMA write transactions, the residue will sit idle in the FIFO until enough new samples are collected to complete a DMA burst. This will not occur until the channel begins filling the FIFO again. Keep in mind that the residue will share a DMA burst with data that may be associated with an entirely different event.

If the FIFO is processing DMA read transactions, the residue will sit idle in the FIFO until the channel is activated. Once active, the residue will be the first data to exit the FIFO followed by samples that get loaded with fresh DMA burst data.

4.3 FIFO Flush

Flushing the FIFO forces out the residue by temporarily extending the channel active duration.

If the FIFO is processing DMA write transactions, a DMA burst will be initiated even though there is not enough data in the FIFO to fill the burst. The resulting transfer will first remove the residue and then pad the remainder of the burst with a repeating pattern of the last 64 bit value showing at the FIFO output. The application should only process the amount of data expected in the DMA buffer and disregard the rest.

If the FIFO is processing DMA read transactions, the channel will simply remain active until the FIFO is empty. This approach effectively extends channel activity beyond the specified stop event, which may not be practical.