DMA in PCIe® Switches

Application Note AN-714

Notes

By Craig Hackney

Introduction

With the increases in PCIe® bus speeds and system performance requirements, the ability to utilize the PCIe bus bandwidth without adding additional load to the host CPU becomes increasingly more important. A simple PCIe GigE NIC solves this problem by integrating a simple DMA engine, allowing for maximum data throughput with minimum host CPU overhead. Of course, this DMA engine is dedicated to the NIC, which makes it useless when it comes to transferring data to or from other endpoints in the system. As PCIe-based system topologies become more popular, there is a growing need for a generic method to move large amounts of data quickly between the root complex and the endpoints, or between multiple endpoints. The solution to this problem is to integrate flexible DMA engines directly into the PCIe switch.

DMA Overview

The IDT PES32NT24G2 PCIe switch contains two high performance DMA engines, each supporting 2 independent channels. A DMA engine can serve as a Function of the partition's upstream port (shown in Figure 1), as a Function of a Non-transparent port (shown in Figure 2), or as a Function of an Upstream Port with a Non-transparent Port (shown in Figure 3).

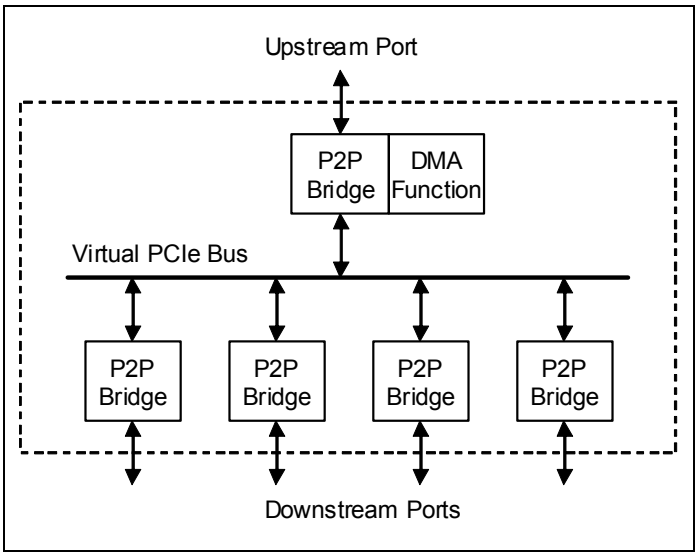

Figure 1 Upstream Port with DMA Function

Notes

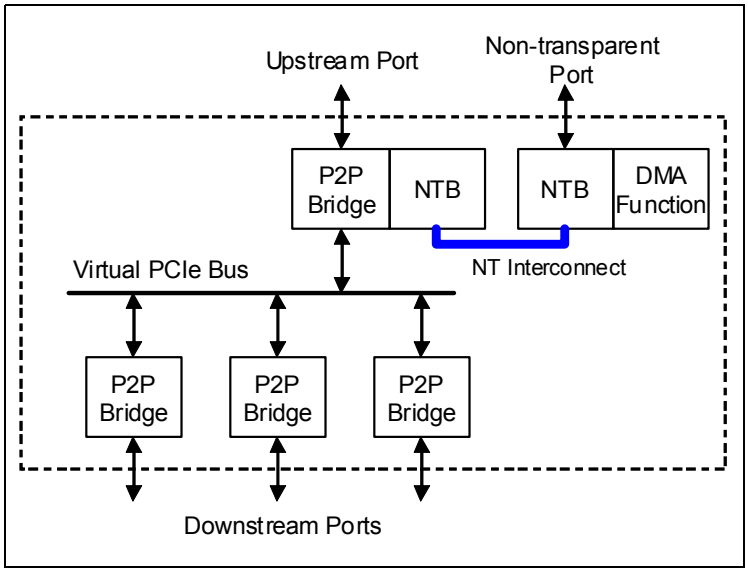

Figure 2 Non-transparent Port With DMA Function

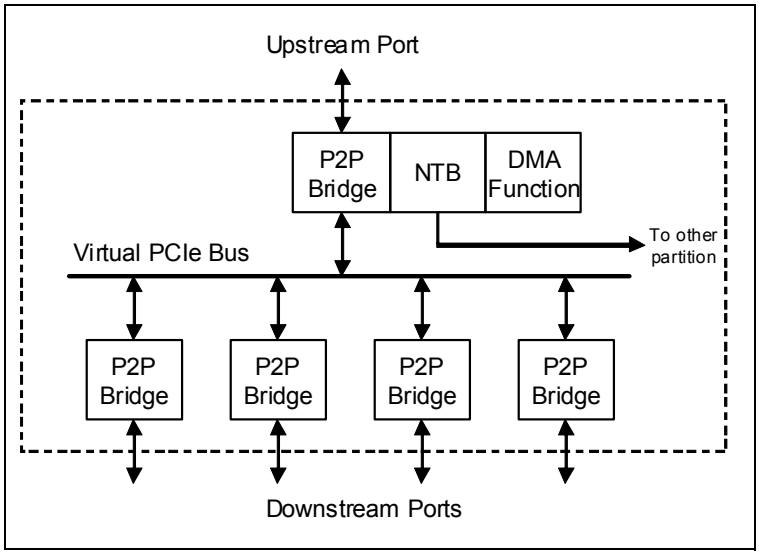

Figure 3 Upstream Port with Non-transparent Port and DMA Function

The DMA channels perform fly-by translation of the TLP's to reduce latency and increase performance over buffered approaches. Figure 4 illustrates a data transfer operation performed by a DMA channel. As shown in this figure, the DMA channel issues memory read request TLP's (MRd) to read data from the source memory (1). When the DMA receives a completion (Cpl) TLP for the read request (2) it transforms the completion into a memory write (MWr) TLP and issues it to the destination memory (3).

Notes

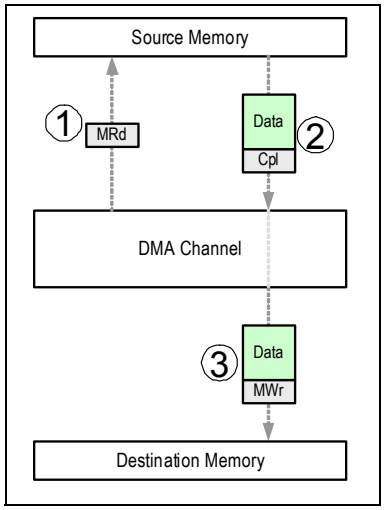

Figure 4 DMA Data Transfer

As depicted in Figure 5, the DMA supports data transfer within the root complex (1), between the root complex and endpoints (2), between endpoints (3), and within an endpoint (4), or between partitions via NTB as depicted in Figure 6.

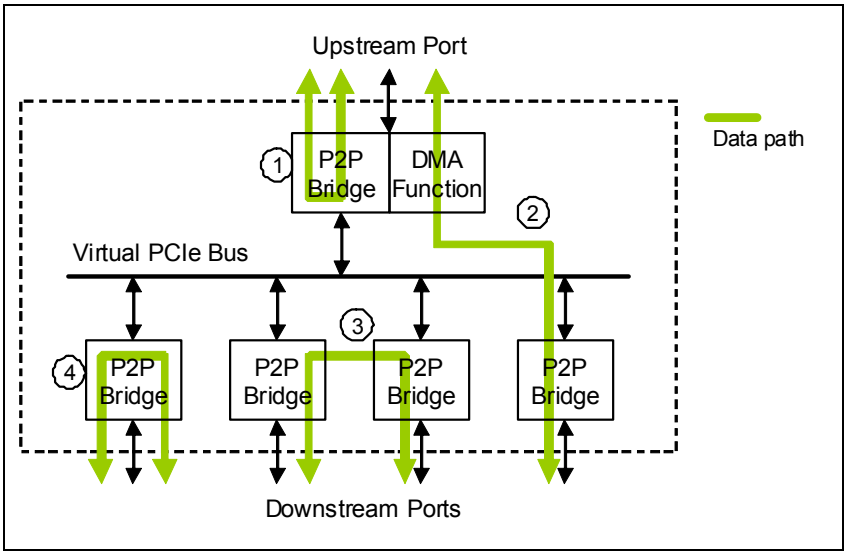

Figure 5 Supported DMA Datapaths

Notes

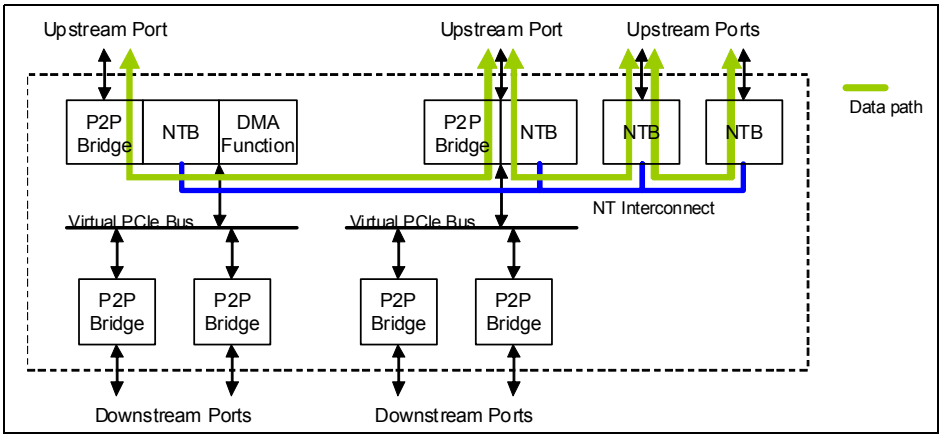

Figure 6 Inter-partition Datapath via NTB

Descriptors located above the upstream port, below a downstream port, or in another partition accessed through an NT endpoint are used to control the DMA channels; each DMA descriptor is 8 DWords in size and contains all of the information required by the DMA channel to perform a data transfer.

DMA Descriptor Types

There are three different types of descriptors used by the DMA: Data Transfer Descriptor, Stride Control Descriptor, and Immediate Data Transfer Descriptor.

A Data Transfer Descriptor, as depicted in Figure 7, is used to perform a normal transfer of data between a source and destination address. Table 1 describes the Data Transfer Descriptor fields.

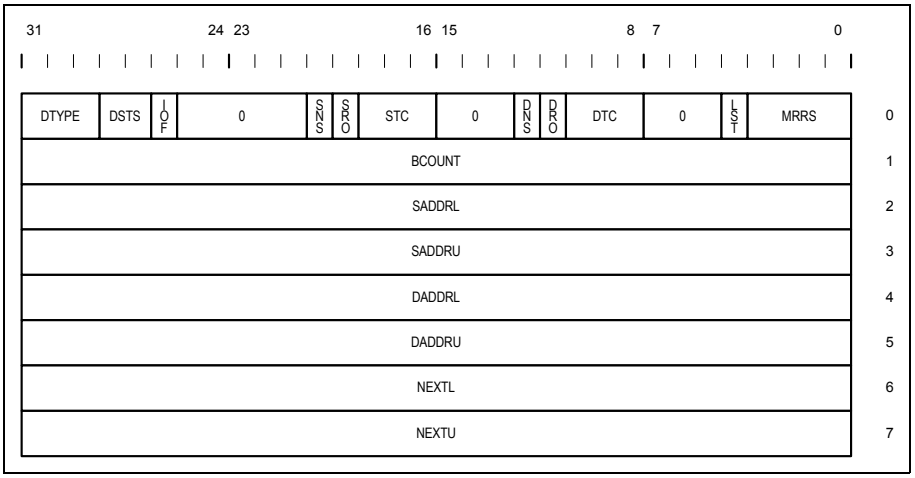

Figure 7 Data Transfer Descriptor

Notes

Field	DWord	Bit Position	Description
MRRS	0	3:0	**Maximum Read Request Size.** This field specifies the maximum DMA source read request size. 0000b - 1B 0001b - 2B 0010b - 4B 0011b - 8B 0100b - 16B 0101b - 32B 0110b - 64B 0111b - 128B 1000b - 256B 1001b - 512B 1010b - 1024B 1011b - 2048B 1100b - 4096B 1101b - Reserved 1110b - Reserved 1111b - Reserved
LST	0	4	**Last.** When this bit is set, it indicates that this descriptor is the last descriptor in the list. Setting this bit is equivalent to setting the next descriptor address fields (NEXTL and NEXTU) to zero.
DTC	0	10:8	**Destination Traffic Class.** This field specifies the traffic class used by destination TLPs.
DRO	0	11	**Destination Relaxed Ordering.** This field specifies the state of the relaxed ordering attribute in destination TLPs.
DNS	0	12	**Destination No Snoop**. This field specifies the state of the no snoop attribute in destination TLPs.
STC	0	18:16	**Source Traffic Class.** This field specifies the traffic class used by source TLPs.
SRO	0	19	**Source Relaxed Ordering.** This field specifies the state of the relaxed ordering attribute in source TLPs.
SNS	0	20	**Source No Snoop.** This field specifies the state of the no snoop attribute in source TLPs.
IOF	0	26	**Interrupt on Finished.** When this bit is set and the DMA controller normally finishes processing of the descriptor, then the F bit is set in the corresponding channel DMA Status (DMAxS) register.
DSTS	0	28:27	**Descriptor Status.** A non-zero value in this field indicates that the DMA controller has finished processing the operation associated with this descriptor. 0x0 - Unprocessed descriptor 0x1 - Descriptor processing completed normally (i.e., finished) 0x2 - Reserved 0x3 - Descriptor processing completed due to an error Other - Reserved

Table 1 Data Transfer Descriptor Fields (Page 1 of 2)

Notes

Field	DWord	Bit Position	Description
DTYPE	0	31:29	**Descriptor Type.** This field encodes the type of descriptor and must be set to 0x1 in data transfer DMA descriptors. 0x0 - Reserved 0x1 - Data transfer DMA descriptor 0x2 - Immediate data transfer DMA descriptor 0x3 - Stride control DMA descriptor Others - Reserved
BCOUNT	1	31:0	**Byte Count.** Total number of bytes to transfer.
SADDRL	2	31:0	**Source Address Lower.** Lower 32-bits of 64-bit source DMA address.
SADDRU	3	31:0	**Source Address Upper.** Upper 32-bits of 64-bit source DMA address.
DADDRL	4	31:0	**Destination Address Lower.** Lower 32-bits of 64-bit destination DMA address.
DADDRU	5	31:0	**Destination Address Upper.** Upper 32-bits of 64-bit destination DMA address.
NEXTL	6	31:0	**Next Descriptor Address Lower.** Lower 32-bits of 64-bit next descriptor DMA address. Descriptors must be word aligned thus the lower two bits must be zero.
NEXTU	7	31:0	**Next Descriptor Address Upper.** Upper 32-bits of 64-bit next descriptor DMA address.

Table 1 Data Transfer Descriptor Fields (Page 2 of 2)

The Stride Control Descriptor, depicted in Figure 8, is used to configure the DMA channel addressing modes. No data transfer is performed by this descriptor (refer to section DMA Addressing Modes on page 13 for more information). Table 2 describes the Stride Control Descriptor fields.

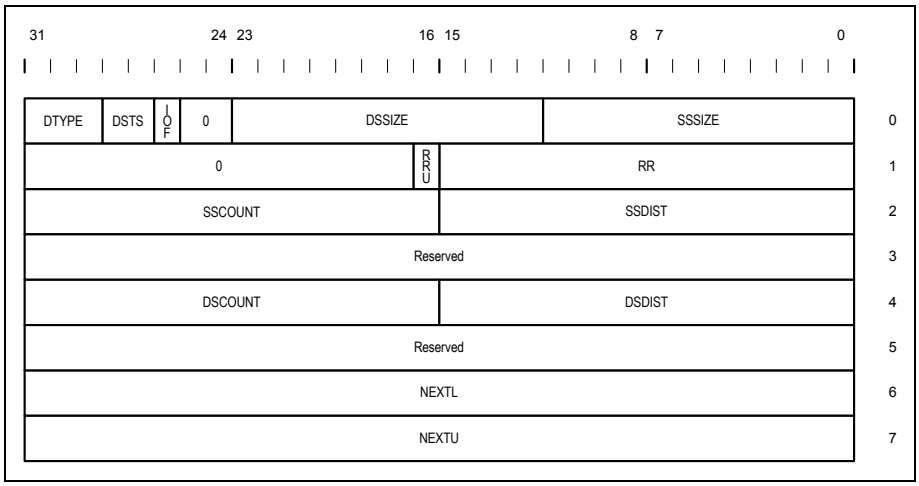

Figure 8 Stride Control Descriptor

Notes

Field	DWord	Bit Position	Description
SSSIZE	0	11:0	**Source Stride Size.** This field specifies the source stride in bytes. A value of zero indicates an infinite stride size.
DSSIZE	0	23:12	**Destination Stride Size.** This field specifies the destination stride in bytes. A value of zero indicates an infinite stride size.
IOF	0	26	**Interrupt on Finished.** When this bit is set and the DMA controller normally finishes processing of the descriptor, then the F bit is set in the corresponding channel DMA Status (DMAxS) register.
DSTS	0	28:27	**Descriptor Status.** A non-zero value in this field indicates that the DMA controller has finished processing the operation associated with this descriptor. 0x0 - Unprocessed descriptor 0x1 - Descriptor processing completed normally (i.e., finished) 0x2 - Reserved 0x3 - Descriptor processing completed due to an error Other - Reserved
DTYPE	0	31:29	**Descriptor Type.** This field encodes the type of descriptor and must be set to 0x3 in stride control DMA descriptors. 0x0 - Reserved 0x1 - Data transfer DMA descriptor 0x2 - Immediate data transfer DMA descriptor 0x3 - Stride control DMA descriptor Others - Reserved
RR	1	15:0	**Request Rate.** Controls the request rate at which the DMA channel issues data read requests. The value in this field is used when the Request Rate Update (RRU) field in this descriptor is set
RRU	1	16	**Request Rate Update.** When this bit is set, the DMA channel uses the value in the Request Rate (RR) field of this descriptor to update the DMA Channel Request Rate Control (DMACxRRCTL) register.
SSDIST	2	15:0	**Source Stride Distance.** This field contains the source stride distance in bytes. The value in this field is a signed number in two's complement notation.
SSCOUNT	2	31:16	**Source Stride Count.** This field contains the source stride count. The value in this field is an unsigned number. This field must be non-zero.
Reserved	3	31:0	Reserved.
DSDIST	4	15:0	**Destination Stride Distance.** This field contains the destination stride distance in bytes. The value in this field is a signed number in two's complement notation.
DSCOUNT	4	31:16	**Destination Stride Count.** This field contains the destination count. The value in this field is an unsigned number. This field must be non-zero.

Table 2 Stride Control Descriptor Fields (Page 1 of 2)

Notes

Field	DWord	Bit Position	Description
Reserved	5	31:0	Reserved.
NEXTL	6	31:0	**Next Descriptor Address Lower.** Lower 32-bits of 64-bit next descriptor DMA address. Descriptors must be word aligned thus the lower two bits must be zero.
NEXTU	7	31:0	**Next Descriptor Address Upper.** Upper 32-bits of 64-bit next descriptor DMA address.

Table 2 Stride Control Descriptor Fields (Page 2 of 2)

The Immediate Data Transfer Descriptor, depicted in Figure 9, is used to transfer up to 64-bits of immediate data contained within the DATAL/U fields to the destination address specified by the DADDRL/U fields. This type of DMA descriptor is particularly useful in signaling to an endpoint that a data transfer has completed. For example, a host needs to transfer data to an endpoint and then write to a memory mapped register on the endpoint to signal that the transfer is complete. Normally, the DMA would be setup to perform the data transfer and then interrupt the host. The ISR on the host would then signal to the endpoint that the transfer is complete. The latency between the DMA completing the transfer and the ISR signaling to the endpoint that the transfer is complete can be quite large. A good way to reduce this latency would be to have the DMA itself signal that the transfer is complete by writing to the endpoint's memory mapped register. The Immediate Data Transfer Descriptor solves these types of latency issues without the need to allocate and populate memory with the source data.

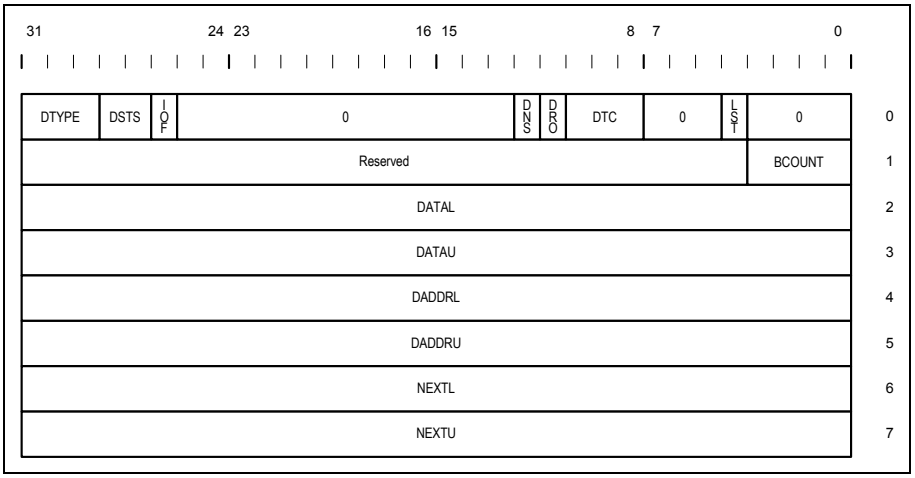

Figure 9 Immediate Data Transfer Descriptor

Notes

Table 3 describes the Immediate Data Transfer Descriptor fields.

Field	DWord	Bit Position	Description
LST	0	4	**Last.** When this bit is set, it indicates that this descriptor is the last descriptor in the list. Setting this bit is equivalent to setting the next descriptor address fields (NEXTL and NEXTU) to zero.
DTC	0	10:8	**Destination Traffic Class.** This field specifies the traffic class used by destination TLPs.
DRO	0	11	**Destination Relaxed Ordering.** This field specifies the state of the relaxed ordering attribute in destination TLPs.
DNS	0	12	**Destination No Snoop.** This field specifies the state of the no snoop attribute in destination TLPs.
IOF	0	26	**Interrupt on Finished.** When this bit is set and the DMA controller normally finishes processing of the descriptor, then the F bit is set in the corresponding channel DMA Status (DMAxS) register.
DSTS	0	28:27	**Descriptor Status.** A non-zero value in this field indicates that the DMA controller has finished processing the operation associated with this descriptor. 0x0 - Unprocessed descriptor 0x1 - Descriptor processing completed normally (i.e., finished) 0x2 - Reserved 0x3 - Descriptor processing completed due to an error Other - Reserved
DTYPE	0	31:29	**Descriptor Type.** This field encodes the type of descriptor and must be set to 0x2 in immediate data transfer DMA descriptors. 0x0 - Reserved 0x1 - Data transfer DMA descriptor 0x2 - Immediate data transfer DMA descriptor 0x3 - Stride control DMA descriptor Others - Reserved
BCOUNT	1	3:0	**Byte Count.** Total number of bytes to transfer. Immediate data transfer descriptors can transfer a maximum of 8 bytes. 0x1 - Transfer 1 byte 0x2 - Transfer 2 bytes 0x3 - Transfer 3 bytes 0x4 - Transfer 4 bytes 0x5 - Transfer 5 bytes 0x6 - Transfer 6 bytes 0x7 - Transfer 7 bytes 0x8 - Transfer 8 bytes Others - Reserved
DATAL	2	31:0	**Data Lower.** Data to be written into the destination device (first 4 bytes).
DATAU	3	31:0	**Data Upper.** Data to be written into the destination device (last 4 bytes).

Table 3 Immediate Data Transfer Descriptor Fields (Page 1 of 2)

Notes

Field	DWord	Bit Position	Description
DADDRL	4	31:0	**Destination Address Lower.** Lower 32-bits of 64-bit destination DMA address.
DADDRU	5	31:0	**Destination Address Upper.** Upper 32-bits of 64-bit destination DMA address.
NEXTL	6	31:0	**Next Descriptor Address Lower.** Lower 32-bits of 64-bit next descriptor DMA address. Descriptors must be word aligned thus the lower two bits must be zero.
NEXTU	7	31:0	**Next Descriptor Address Upper.** Upper 32-bits of 64-bit next descriptor DMA address.

Table 3 Immediate Data Transfer Descriptor Fields (Page 2 of 2)

Upon completion of a transfer, the DMA updates a status field in the associated descriptor and optionally interrupts the host. Each descriptor is capable of transferring up to 4GB of data with no restrictions on the alignment of the source or destination addresses. For example, the source address may reside on a byte boundary and the destination address may reside on a word boundary. If more than one transfer is required, descriptors may be linked together to form a list of descriptors. The DMACxDPRTL/H control registers (located in the DMA Functions PCIe configuration space) should be initialized with the address of the first descriptor in the list. The end of the list is signified by a descriptor with the LST bit set or its NEXTL/U fields set to zero; address zero is not a valid address for a descriptor. To reduce the latency associated with reading a descriptor and thus increasing performance, an active DMA channel may be configured to pre-fetch descriptors by writing to the DPREFETCH field in the DMACxCFG register.

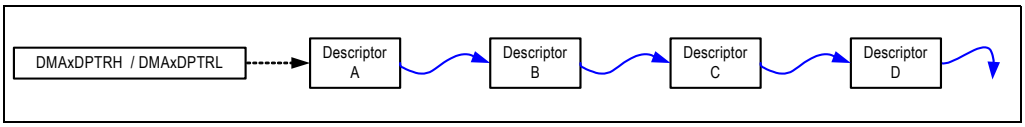

Figure 10 Typical DMA Descriptor List

In general, being able to process a single list of descriptors is useful, but in a typical system there will probably be a need for some type of descriptor queuing or at least the ability to append descriptors to a list. The DMA supports three methods to facilitate this: Descriptor List Chaining, Descriptor List Appending, and Descriptor Rings. All three methods are described in detail in the following sections.

Descriptor List Chaining

Without Descriptor List Chaining a DMA channel will cease descriptor processing once it reaches the last descriptor in a list. Descriptor List Chaining is enabled by clearing the DISNDPTRL/H bits in the DMACxCFG register. A new descriptor list is then chained by writing the address of the first descriptor in the list to the DMACxNDPTRL/H registers. When the DMA channel finishes processing the descriptor list pointed to by the DMACxDPTRL/H registers it will move the contents of the DMACxNDPTRL/H registers to the DMACxDPTRL/H registers and then begin to process the new descriptor list.

Notes

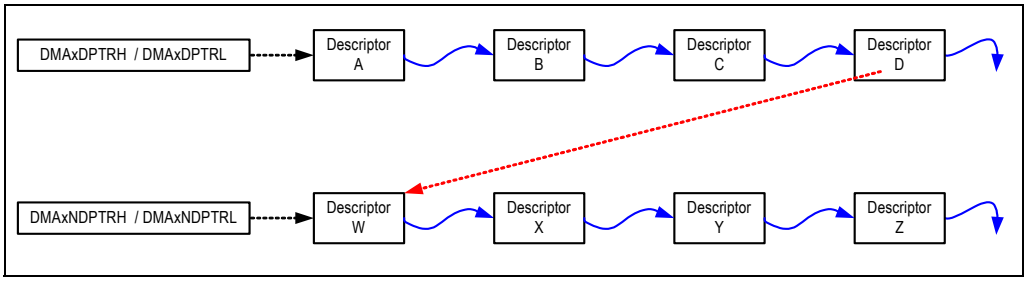

Figure 11 Descriptor List Chaining

As depicted in Figure 11, the DMACxDPTRL/H registers are programmed with the address of a descriptor list which the DMA channel begins to process. While processing this list, the address of a new descriptor list is written to the DMACxNDPTRL/H registers. Once the DMA channel has completed processing descriptors A, B, C, and D, it will automatically begin to process descriptors W, X, Y, and Z. If the DMA had already completed processing the initial descriptor list when the DMACxNDPTRL/H registers were written, the DMA immediately begins processing the new list. To avoid a race condition when writing 64-bit addresses to the DMACxNDPTRL/H registers, the initiation of descriptor processing as a side effect of writing to the DMACxNDPTRL/H registers may be disabled by setting the DISNDPTRH or DISNDPTRL bits in the DMACxCFG register.

This method of descriptor processing is useful when the DMA is used to perform multiple data transfers from multiple PCIe devices in a set order. Take, for example, an application which is required to periodically collect statistical information stored in multiple locations on multiple PCIe devices. The application can build multiple descriptor lists, one for each PCIe device, during initialization time, and then issue the first list to the DMA by writing its address to the DMACxNDPTRH/L registers. When the list completes, the last descriptor interrupts the host, which in turn will trigger the next descriptor list by writing its address to the DMACxNDPTRH/L registers. When the last descriptor list has been processed, the host would trigger a timer. When the timer expires, the host kicks off the first descriptor list again, and so on.

Descriptor List Appending

In order to use Descriptor List Appending, the DSCP field in the DMACxCFG register must be set to “process next descriptor”. For this method of descriptor processing the LST bit in the descriptor is not used to signify the last descriptor in the list and should be set to zero. Instead, the end of the descriptor list is signified by the NEXTU/L entries being set to zero. Appending to a descriptor list when the descriptor being appended is located below the 4GB boundary (i.e. within 32 bit address space) is a simple case of setting the NEXTL entry in the last descriptor of the list to point to the descriptor to be appended and then setting the RUN bit in the DMACxCTL register. If the new descriptor is appended and the RUN bit set in the DMACxCTL register before the DMA begins to process the last descriptor, the DMA continues to process descriptors, including the newly appended descriptor, in the usual way. If the DMA had already started to process or had finished processing the last descriptor before the new descriptor was appended, the setting of the RUN bit in the DMACxCTL register causes the last descriptor, with the updated NEXTL entry, to be re-fetched. The DMA will not re-process the last descriptor since its status will be *Descriptor Processing Completed Normally* and the DMA is configured to process the next descriptor in the list when a descriptor’s status is anything other than *Unprocessed Descriptor*. Instead, it will simply use the NEXTL entry in the descriptor to move on to the newly appended descriptor.

To avoid a race condition when appending a descriptor to a list when the descriptor to be appended resides above the 4GB boundary (i.e. in 64-bit address space), the DMA channel must first be suspended (by writing a 1 to the SUSPEND bit in the DMACxCTL register) before updating the NEXTL/H entries in the last descriptor of the list. Once the last descriptor has been updated, software should wait for the SUSPEND bit in the DMACxSTS register to be set before resuming the DMA by simultaneously writing a zero to the SUSPEND bit and a one to the RUN bit of the DMACxCTL register.

Notes

This method of descriptor processing is useful for applications which transfer unknown amounts of data. New descriptors can simply be dynamically allocated and appended to the descriptor list while processed descriptors are freed.

Descriptor Rings

In systems where the requirements on the DMA are known up front and performance is of the upmost importance, a better solution to descriptor appending may be to use a descriptor ring. When this method of descriptor processing is employed, a ring of descriptors is created during initialization using the NEXTL/H entries in the descriptors to maintain the ring and the LST bit in the descriptors is used to signify the end of the list. Initially, the LST bit will be set in all of the descriptors in the ring. As with Descriptor List Appending, the DSCP field in the DMACxCFG register must be set to “process next descriptor”.

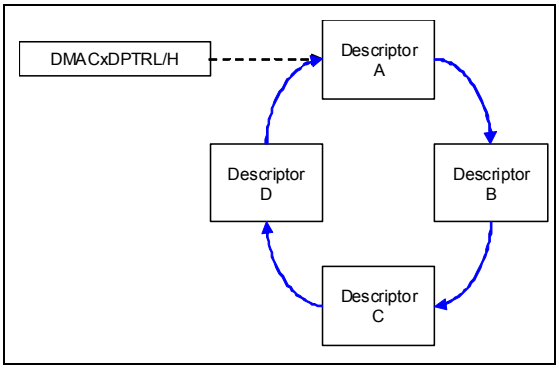

Figure 12 Descriptor Ring Example

Once the ring has been initialized, the address of the first descriptor in the ring, in this case descriptor A, is programmed into the DMACxDPTRL/H registers. Software will first program, in order, each descriptor in the ring, making sure to only set the LST bit in the last descriptor for the transfer, and then it will set the RUN bit in the DMACxCTL register. To append a descriptor, software simply programs the next descriptor in the ring, clears the LST bit in the previous descriptor, and sets the RUN bit in the DMACxCTL register. If the DMA had not yet processed the last descriptor of the transfer, it will continue as usual and include the newly appended descriptor. If the DMA had started to process the last descriptor, when processing is completed the DMA will begin to process the newly appended descriptor because the RUN bit was set in the DMACxCTL register. This approach does require the software to keep track of where the end of the ring is and where a new descriptor should be appended and to prevent a descriptor not yet processed by the DMA from being overwritten. However, there is no requirement for the dynamic allocation and freeing of descriptors, which in some cases could take quite some time.

A good example of this method of descriptor processing would be in CPU to CPU communications, as depicted in Figure 13. In this example, two separate CPU’s connected to two separate switches communicate with each other via NTB, utilizing a DMA engine in each switch to perform high speed data transfers.

Notes

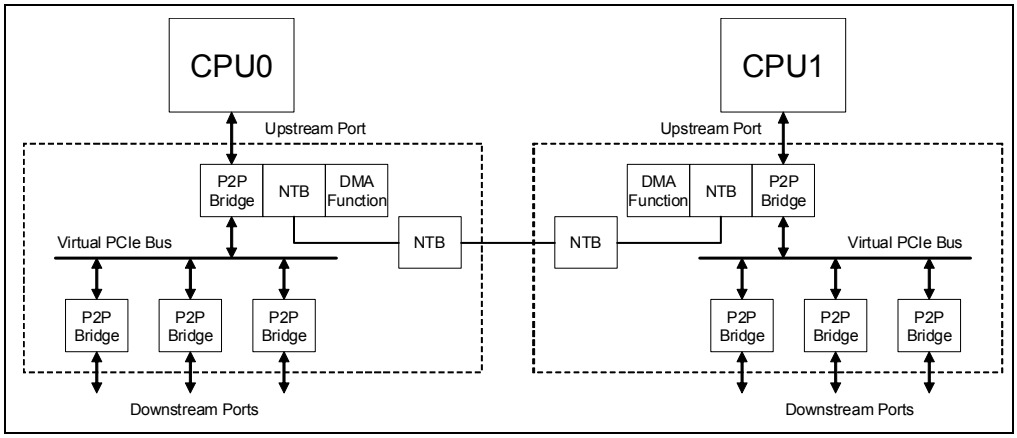

Figure 13 CPU to CPU Communications Utilizing DMA and NTB

DMA Addressing Modes

The DMA supports two types of addressing modes: Linear Address and Constant Addressing. Both of these addressing modes are described in the following sections.

Linear Addressing

The simplest form of DMA addressing is Linear Addressing where data is moved from an incrementing source address to an incrementing destination address. Figure 14 depicts a linear transfer of BCOUNT bytes from SADDR to DADDR within the PCIe memory address space.

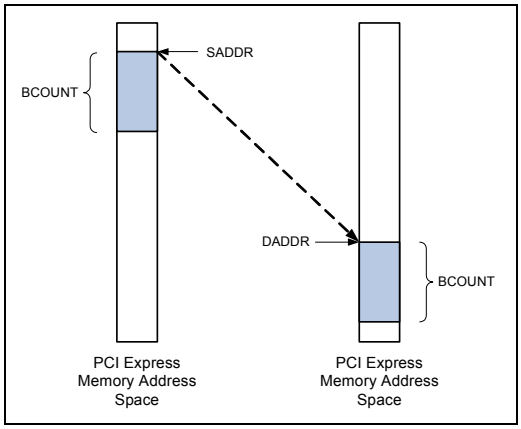

Figure 14 Linear Addressing

The DMA may be instructed to perform a linear addressing transfer by means of a Data Transfer Descriptor.

Constant Addressing

Constant Addressing refers to one or both the source or destination address remaining constant during a transfer. Figure 15 depicts a transfer of BCOUNT bytes from a constant source address (SADDR) to a linear destination address (DADDR).

Notes

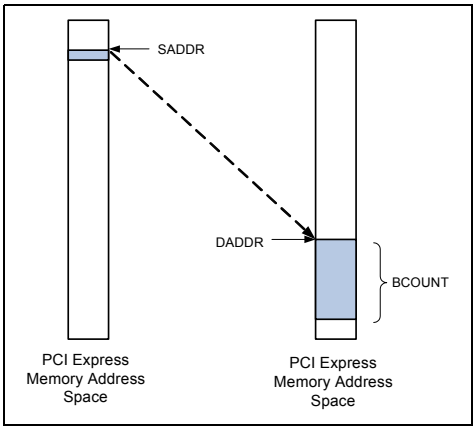

Figure 15 Constant Addressing

Before performing a constant address transfer, the DMA must first be configured by means of a Stride Control Descriptor. This type of addressing is useful when reading or writing data to a memory mapped FIFO of a PCIe endpoint.

Request Rate Control

By default, a DMA channel issues memory read requests as needed which provides the highest level of performance. However, this may cause congestion in systems with low memory bandwidths. To solve this issue and to allow for lower priority background DMA transfers, each DMA channel supports a Request Rate Control capability. Request Rate Control is enabled and controlled by the value in the Request Rate (RR) field of the DMA Channel Request Rate Control (DMACxRRCTL) register.

When Request Rate Control is enabled, a counter is loaded with the number of DWords requested when each memory read request (MRd) TLP is issued. While non-zero, the counter is decremented by the value indicated in the RR field of the DMACxRRCTL register, e.g., a value of 1 in the RR field indicates that the counter is decremented every 4ns while a value of 1000 indicates that the counter is decremented every 4 µs. A new memory read request will not be issued until the counter is zero.

Request Rate Control has no effect on descriptor read requests or data transfer memory write requests. It is possible to specify Request Rate Control information within a Stride Control Descriptor by specifying values for the RR and RRU fields.

DMA Multicast

DMA multicast allows the DMA to be configured to read data from a source location, and multicast this data to several destination locations. A DMA channel may be configured such that the destination address of a descriptor hits a multicast BAR aperture in the upstream port's PCI-to PCI bridge (i.e., transparent multicast) and/or in the NT function (non-transparent multicast). When this occurs, the posted TLP emitted by the DMA function is transferred to the upstream port's link (i.e., the port where the DMA function resides), as well as multicasted to the appropriate ports, as dictated by transparent and non-transparent multicast operation. Figure 16 depicts the path taken by a posted TLP emitted by the DMA that falls in the multicast BAR aperture of the upstream port's PCI-to-PCI bridge function. As shown in the figure, the TLP is transmitted on the upstream port's link, as well as multicasted to the appropriate downstream ports.

Notes

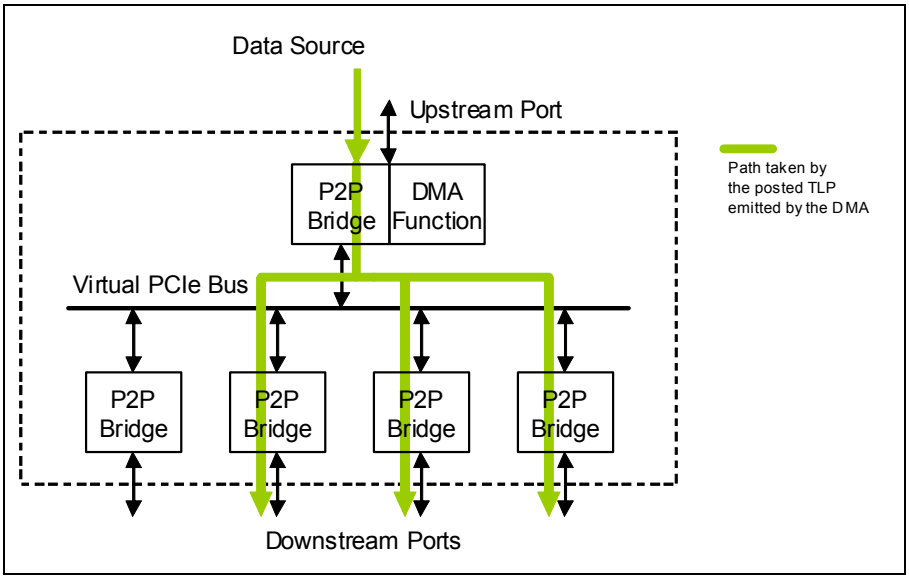

Figure 16 Path Taken by a TLP Emitted by the DMA When It Is Multicasted

Figure 17 shows a similar example, but this time the TLP emitted by the DMA is NT multicasted to ports in other partitions. As shown in the figure, the TLP is transmitted on the upstream port's link, as well as NT multicasted to ports in other partitions.

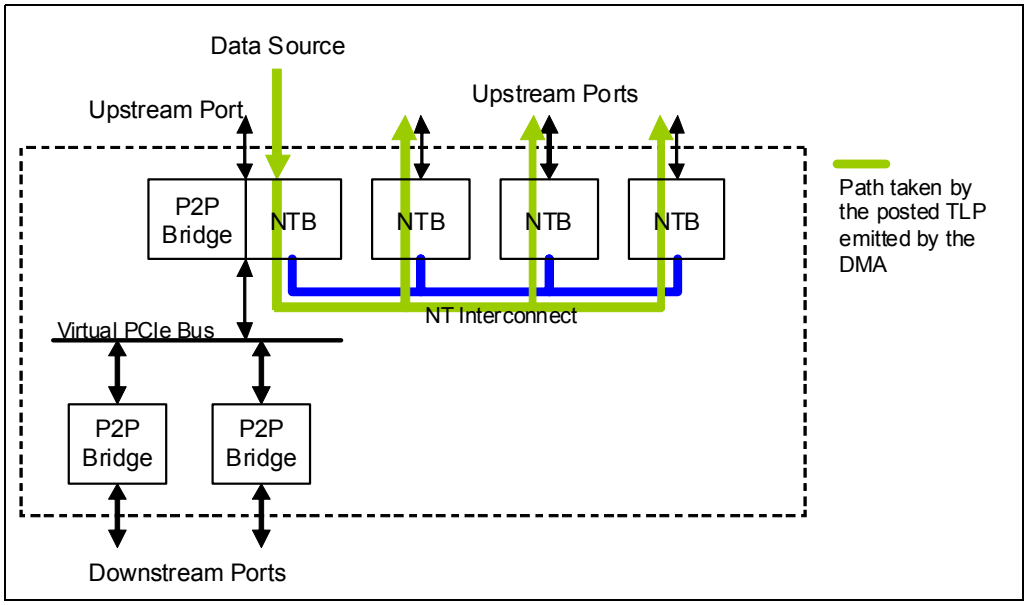

Figure 17 Path Taken by a TLP Emitted by the DMA When It Is NT Multicasted

Programming Examples

The following sections contain DMA programming examples. It is assumed that all registers contain their default values unless otherwise stated.

Notes

Linear Addressing Transfer

In this example, it is assumed that the DMA descriptor is located at address 0x100000 and will be transferring 0x1000 bytes of data from address 0x80000000 to address 0x10000000. First populate the 8 DWord Data Transfer Descriptor at address 0x100000 as depicted in Table 4.

DWord #	Value	Description
0	0x24000010	Control/Status ◆ Data transfer descriptor. ◆ Unprocessed descriptor. ◆ Interrupt on Finish. ◆ Last descriptor.
1	0x00001000	Byte count.
2	0x80000000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x10000000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00000000	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 4 Data Transfer Descriptor Example

The Control/Status word (DWord 0) of the descriptor is set to generate an interrupt on completion of the transfer. Ensure that the appropriate interrupt is enabled and has an ISR associated with it on the host, then do the following:

- Clear the FINISHED bit (bit 0) in the DMACxMSK register to allow the generation of an interrupt when the transfer completes.
- Write 0x100000 to the DMACxDPTRL register to set the address of the first descriptor.
- Begin the data transfer by setting the RUN bit in the DMACxCTL register to 1.

Upon completion of the data transfer the host will be interrupted, the DSTS field in the descriptor will be set to 0x1 (Descriptor processing completed normally), and the FINISHED bit (bit 0) will be set in the DMACxSTS register; this bit should be cleared, to acknowledge the receipt of the interrupt, by writing 1 to it.

To have the DMA transfer automatically start when the address of the descriptor is written to the DMACxDPTRL register, first clear the FINISHED bit (bit 0) in the DMACxMSK register and then do the following:

- Clear the DISDPTRL bit in the DMACxCTL register.
- Set the RUN bit in the DMACxCTL register to 1.
- Start the DMA transfer by writing the address of the descriptor (0x100000) to the DMACxDPRTL register.

Constant Address Transfer

A DMA channel must be configured in order to perform a constant address transfer. This configuration is performed by a Stride Control Descriptor and need only be performed once. In this example, two descriptors are used. The first, a Stride Control Descriptor, is used to configure the DMA channel. The second, a Data Transfer Descriptor, is used to perform the actual data transfer. It is assumed that the first descriptor is located at address 0x100000 and is directly followed by the second descriptor at address 0x100020. The DMA will be configured to transfer 0x1000 bytes of data from a constant source address (0x80000000) to a linear destination address (0x10000000), similar to that which is depicted in Figure 15. To begin, create the two descriptors located at addresses 0x100000 and 0x100020 as depicted in Tables 5 and 6.

Notes

DWord #	Value	Description
0	0xe4000004	Control/Status Stride control descriptor. Unprocessed descriptor. Interrupt on Finish. Source Stride Size (4).
1	0x00000000	Request Rate Update and Request Rate. Unused for this transfer.
2	0x0400FFFC	Source stride count (0x400) and source stride distance (-4).
3	0x00000000	Reserved.
4	0x00010000	Destination stride count (0x1).
5	0x00000000	Reserved.
6	0x00100020	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 5 First Descriptor of a Constant Address Transfer

DWord #	Value	Description
0	0x24000010	Control/Status Data transfer descriptor. Unprocessed descriptor. Interrupt on Finish. Last descriptor.
1	0x00001000	Byte count.
2	0x80000000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x10000000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00000000	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 6 Second Descriptor of a Constant Address Transfer

The Control/Status word (DWord 0) of the descriptors is set to generate an interrupt upon completion of descriptor processing, ensure that the appropriate interrupt is enabled and has an ISR associated with it on the host, then do the following:

- Clear the FINISHED bit (bit 0) in the DMACxMSK register to allow the generation of an interrupt as each descriptor is processed.
- Write 0x100000 to the DMACxDPTRL register to set the address of the first descriptor.
- Begin the data transfer by setting the RUN bit in the DMACxCTL register to 1.

The DMA will generate two interrupts, one upon completion of the first descriptor, which is used to configure the DMA constant addressing mode, and another upon completion of the second descriptor, which performs the actual data transfer. As each descriptor completes, the DSTS field within the descriptor will be set to 0x1, *Descriptor Processing Completed Normally*.

Notes

Descriptor List Chaining

The following example shows how to configure the DMA to perform simple linear transfers using the descriptor list chaining method. In this example, two lists of descriptors will be created. The first list transfers 0x1000 from address 0x80000000 to address 0x10000000 (described in Table 7) and then an additional 0x1000 bytes from address 0x80001000 to address 0x10001000 (described in Table 8)

.

DWord #	Value	Description
0	0x20000000	Control/Status Data transfer descriptor. Unprocessed descriptor.
1	0x00001000	Byte count.
2	0x80000000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x10000000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00100020	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 7 List 0, Descriptor 0

DWord #	Value	Description
0	0x24000010	Control/Status Data transfer descriptor. Unprocessed descriptor. Interrupt on Finish. Last Descriptor.
1	0x00001000	Byte count.
2	0x80001000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x10001000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00000000	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 8 List 0, Descriptor 1

As can be seen in Table 8, only one interrupt will be generated (signified by the 0x04000000 bit in the Control/Status DWord) when the last descriptor in the list completes. Ensure that the appropriate interrupt is enabled and has an ISR associated with it on the host, then do the following:

- Clear the FINISHED bit (bit 0) in the DMACxMSK register to allow the generation of an interrupts.
- Clear the DISNDPTRL bit in the DMACxCFG register to enable descriptor list chaining when the DMACxNDPTRL register is written.
- Set the RUN bit in the DMACxCTL register.
- Start the transfer of the first descriptor list by writing the address of the first descriptor in the list to the DMACxNDPTRL register.

Notes

DWord #	Value	Description
0	0x20000000	Control/Status Data transfer descriptor. Unprocessed descriptor.
1	0x00001000	Byte count.
2	0x80002000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x10002000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00100060	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 9 List 1, Descriptor 0

DWord #	Value	Description
0	0x24000010	Control/Status Data transfer descriptor. Unprocessed descriptor. Interrupt on Finish. Last Descriptor.
1	0x00001000	Byte count.
2	0x80003000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x10003000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00000000	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 10 List 1, Descriptor 1

Upon completion of the first descriptor list, the host will be interrupted.

- Clear the FINISHED bit (bit 0) bit in the DMACxSTS register by writing one to it.
- Start the transfer of the second descriptor list by writing the address of the list to the DMACx-NDPTR register.

The completion procedure for the second descriptor list will be identical to that of the first.

Descriptor List Appending

Before appending descriptors to a descriptor list, the DMA mush first be initialized with a initial (dummy) descriptor, depicted in Table 11.

- Set the DSCP field of the DMACxCFG register to 0x02 (process next descriptor).
- Program the address of the dummy descriptor into the DMACxDPTRL/H registers.
- If interrupts are to be generated upon completion of a transfer, clear the FINISHED bit (bit 0) in the DMACxMSK register and ensure that the appropriate interrupt is enabled and has an ISR associated with it on the host.
- Set the RUN bit in the DMACxCTL register.

Notes

DWord #	Value	Description
0	0x28000000	Control/Status Data transfer descriptor. Processed descriptor.
1	0x00000000	Byte count.
2	0x00000000	Source address (lower 32 bits).
3	0x00000000	Source address (upper 32 bits).
4	0x00000000	Destination address (lower 32 bits).
5	0x00000000	Destination address (upper 32 bits).
6	0x00000000	Next descriptor (lower 32 bits).
7	0x00000000	Next descriptor (upper 32 bits).

Table 11 Dummy Starting Descriptor

The DMA will attempt to process the Dummy Descriptor, but, because the status of the descriptor is not *Unprocessed Descriptor,* the DMA will attempt to process the next descriptor. Since the NEXTL/U fields of the dummy descriptor are 0x0, there is no next descriptor. So the DMA will halt descriptor processing. Additional descriptors may be appended by:

- Writing the address of the new descriptor into the NEXTL field of the last descriptor.
- Setting the RUN bit in the DMACxCTL register.

If the address of the descriptor being appended is above the 4GB boundary (i.e. 64-bit address space):

- Suspend the DMA by setting the SUSPEND bit in the DMACxCTL register.
- Wait for the SUSPEND bit to be set in the DMACxSTS register.
- Update the NEXTL/H entries in the last descriptor in the list.
- Resume the DMA by simultaneously writing a zero to the SUSPEND bit and a one to the RUN bit of the DMACxCTL register.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.