



# GN412x FlexDMA Sequencer

---

## Design Guide

# Revision History

Version	ECR	Date	Changes and / or Modifications
3	151924	May 2009	Added documents in the listing.
2	151759	April 2009	Renamed document; formerly called: Gennum FlexDMA Sequencer User Manual. Added list of related documentation.
1.0	151505	March 2009	New document

## Contents

1. Related Documentation .....	3
2. Overview.....	4
2.1 Nomenclature.....	5
2.2 Physical Attachment Layer .....	5
2.3 Application Attachment Layer .....	5
2.3.1 Target Controller .....	6
2.3.2 FlexDMA DMA Controller.....	6
2.4 L2P DMA Master.....	6
2.5 P2L DMA Master.....	7
2.6 VDMA Sequencer.....	7
3. VDMA Sequencer.....	9
3.1 VDMA Sequencer Operation.....	9
3.2 VDMA Descriptor Format .....	10
3.3 VDMA Register Descriptions.....	16
3.4 Byte Handling.....	20
3.5 Example VDMA Programming.....	21
3.5.1 Scatter/Gather List Processing .....	21
4. VDMA Architecture.....	23
4.1 VDMA Hardware Interface.....	23
4.1.1 EVENT Interface .....	23
4.2 VDMA Internal Memory Map .....	23

# 1. RELATED DOCUMENTATION

## **Industry Standards:**

- PCI Express Base Specification Revision 1.1, PCI-SIG, March 28, 2005

## **Gennum's Documentation:**

- GN4124 x4 Lane PCI Express to Local Bus Bridge Data Sheet, Document ID: 48407
- GN4121 x1 Lane PCI Express to Local Bus Bridge Data Sheet, Document ID: 51539
- GN412x PCI Express Family Reference Manual, Document ID: 52624
- GN4124 Gullwing RDK User Guide, Document ID: 50932
- GN4121 Gullwing-x1 RDK User Guide, Document ID: 52162
- GN412x RDK Software Design Guide, Document ID: 51859
- GN412x FPGA IP Hardware Design Guide, Document ID: 51860

## 2. OVERVIEW

The FlexDMA IP core is a highly flexible and scalable DMA controller provided as part of the GN412x local bus IP core. FlexDMA enables up to 512 DMA channels to be accommodated. There may be a small number of high bandwidth channels or a large number of lower bandwidth channels. The limitation is the available bandwidth of the GN412x, host system (root complex), and any intermediate switches.

This document describes the operation of the VDMA Sequencer. The sequencer acts as a programmable scheduler to control the state of 2 concurrent raw DMA engines (the L2P and P2L DMA master blocks). Programmability of the VDMA sequencer, its instruction set and registers, is described in this document. Additional documents describe application use model for the VDMA sequencer. Also, source code is available for several drivers that use the VDMA sequencer. This is available through:

[www.gennum.com/mygennum](http://www.gennum.com/mygennum)

The FlexDMA IP core is licensed and royalty free when used with the GN412x.

**Figure 2-1: Gennum Local Bus IP Core with FlexDMA**

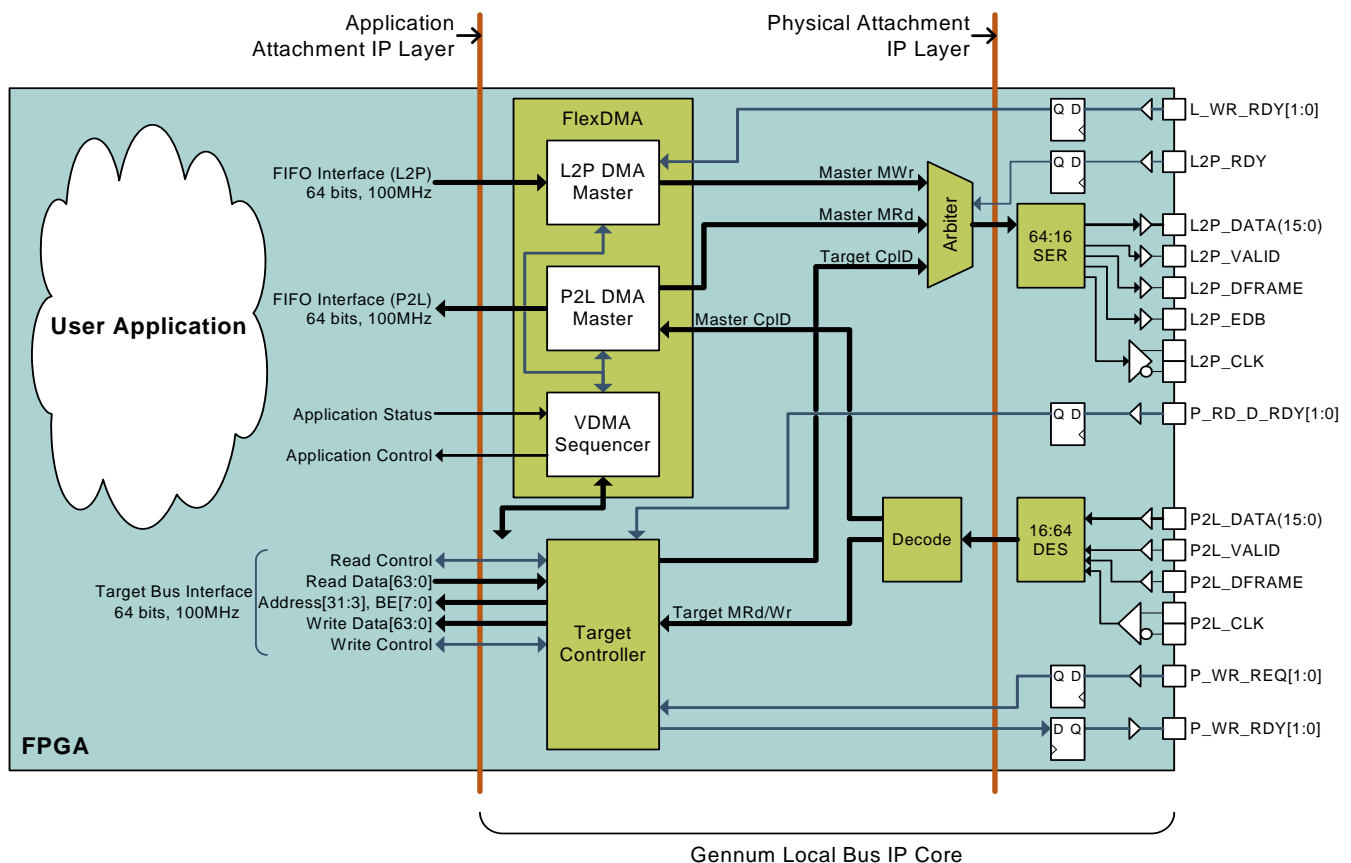


Figure 2-1 outlines the main functions of the GN412x local bus IP core. The IP is layered as:

- Physical Attachment Layer: provides the interface to the GN412x taking the high-speed, narrow external buses and widening them out for use inside the FPGA
- Application Attachment Layer: provides a target interface and the FlexDMA master mode DMA block

## 2.1 Nomenclature

The following terms are used throughout this document as defined below:

P2L	PCIe-to-Local Bus direction: P2L data flow involves bus traffic originating in the host system and moving via the P2L bus to the FPGA.
L2P	Local Bus-to-PCIe direction: L2P data flow involves bus traffic originating in the FPGA and moving via the L2P bus to the GN412x towards the host system via PCI Express.
Raw DMA	A fundamental DMA controller that moves data in either the P2L or L2P direction.
Master Mode Target Mode	There are two modes of data transfer through the GN412x: Master mode is where the FPGA attached to the local bus is initiating a transfer by sending a read request packet or write packet to the GN412x. Target mode operation is where a PCI Express agent is initiating the transfer and the GN412x and/or FPGA are responding.
PCIe	PCI Express
DW	Double Word (32-bit word)
DDR	Double Data Rate: refers to clocking scheme where data is valid on both the rising and falling edge of the clock.
SDR	Single Data Rate: refers to clocking scheme where data is valid only on the rising edge of the clock.

## 2.2 Physical Attachment Layer

The interface immediately attached to the high-speed dual data rate SSTL local interface on the GN412x is called the Physical Attachment Layer. It converts between the narrow high-speed FPGA IO and provides a slower, wider interface to the Application Attachment Layer.

## 2.3 Application Attachment Layer

The Application Attachment Layer converts the transmit/receive interface of the local bus into a form that is more convenient for a typical application. A typical application has a set of configuration and control registers that are used to set up and control the

transfer of data using master mode DMA. That is, the endpoint device achieves high MB/s of data throughput by reading and writing data directly from/to the hosts memory. The host CPU will perform setup through the “target” interface on the endpoint. However, setup is typically not bandwidth intensive. Master mode DMA is used to do the “heavy lifting”.

Both a master and target controller reside in the Application Attachment Layer of the Local Bus IP.

### 2.3.1 Target Controller

When an external PCIe agent<sup>1</sup> (typically the host system) performs a read or write to the GN412x and that falls within the address space defined by BAR0 or BAR2<sup>2</sup>, then this will generate a read/write request on the local bus. The local bus target controller handles these transactions.

### 2.3.2 FlexDMA DMA Controller

Three main blocks are present in the FlexDMA core as depicted in [Figure 2-1](#):

- L2P DMA Master: Local-to-PCIe raw DMA engine
- P2L DMA Master: PCIe-to-Local raw DMA engine
- VDMA Sequencer: manages and virtualizes the raw DMA engines

## 2.4 L2P DMA Master

The L2P DMA Master is a simple DMA engine that performs all transfers where the source data is from the application layer and the destination of the data is the host system.

The L2P DMA Master takes data from the application layer as a FIFO stream, or as described later, multiple FIFO streams. That is, the FlexDMA function doesn't provide an address to the application layer. Rather it is designed to draw data sequentially from a source which is typically a FIFO. If the source requires an address, then it is up to the application layer to provide an address counter.

Data is taken from the application layer and packetized by the L2P DMA Master as write cycles. The activity of the DMA engine is controlled by the VDMA sequencer.

- 
1. An external PCIe agent is typically going to be the host system/root complex. However, since PCIe supports peer-to-peer traffic an external agent may also be another endpoint device capable of addressing the GN412x by performing read/write cycles that fall within the BAR space decoded by the GN412x. Throughout this document, when host system is referred to it may also extend to any PCIe device in the entire memory space that can act as a master/target to the GN412x.
  2. When an address matches the range defined by a Base Address Register (BAR) that is sometimes referred to as a “bar hit”.

## 2.5 P2L DMA Master

The P2L DMA Master performs all transfers where the source data is the host system and the data is destined for the local application layer. In order for data to flow from the host into the GN412x bridge and back to the P2L DMA master, an L2P request is sent to the host. Consequently, P2L DMA does generate small request packets on the L2P bus. However, for large transfer block sizes, the traffic added to the L2P bus is minimal.

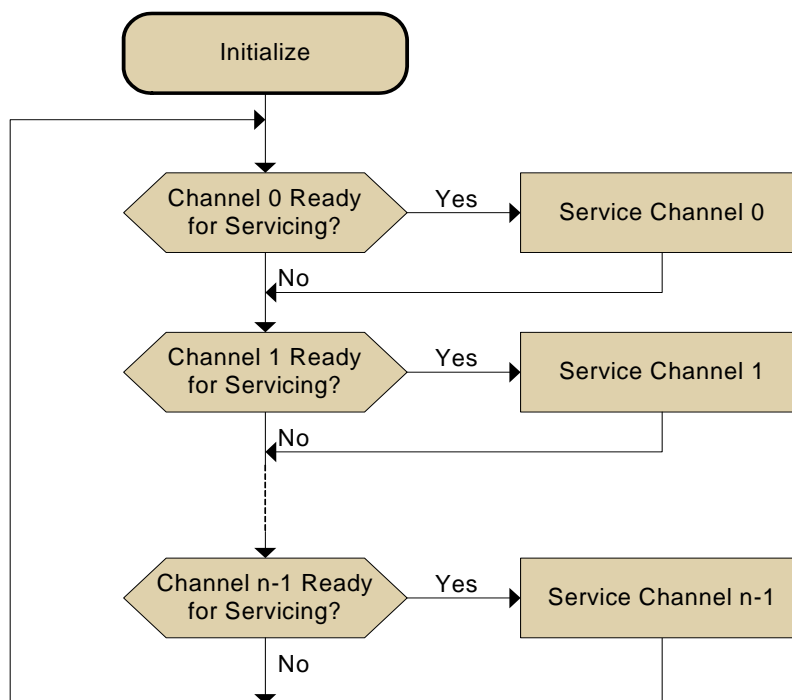
If performance (throughput) is important, it is imperative to take advantage of the full duplex nature of the PCI express interconnect. For this reason, the L2P and P2L DMA engines are separate and concurrent functions.

## 2.6 VDMA Sequencer

Both L2P and P2L DMA are controlled by a micro-coded sequencer. The VDMA sequencer is a very simple RISC processor with an instruction set optimized for DMA control. Its function is to handle an arbitrary number of application sources/sinks of data and keep the raw DMA engines as busy as possible by acting as an intelligent scheduler.

The typical simplified use model for VDMA is depicted in [Figure 2-2](#).

**Figure 2-2: Typical VDMA Polling Sequence Flow Chart**



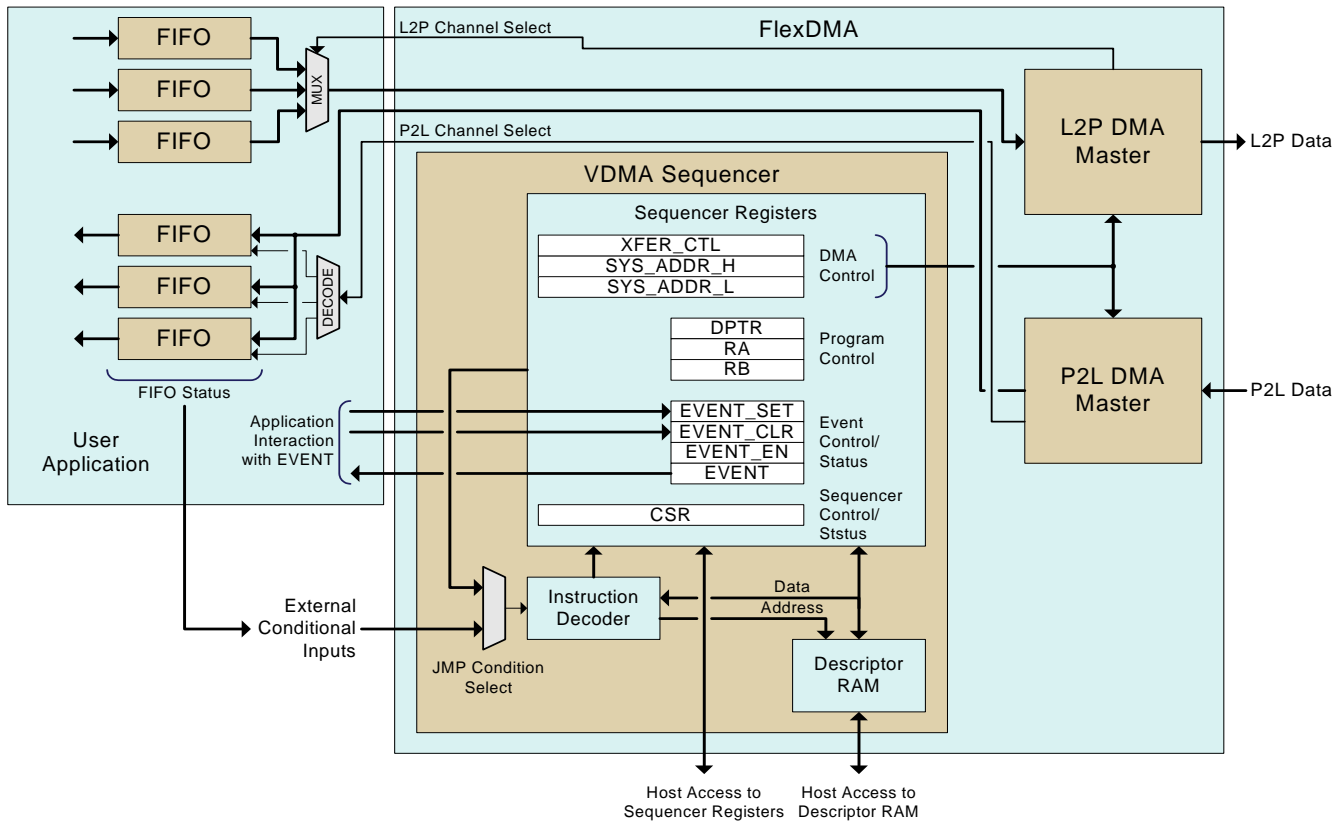
The sequencer is typically iterating a loop and polling for application data sources and/or sinks that are ready to be serviced (i.e. are ready for DMA transfer to happen). The L2P and P2L DMA engines handle only small transfer sizes of 4KB maximum. Larger DMA transfers may be assembled by the sequencer as a loop of smaller transfers. Consequently, VDMA has loop counters and indexing.

Another reason that the L2P and P2L DMA engines handle only up to 4KB transfer sizes is that typical operating systems use 4KB as a page size for virtual memory. Consequently, when an application (operating in virtual address space) asks the operating system to allocate a large buffer in memory, the physical layout of the buffer may be “scattered” around physical memory in blocks of 4KB or less.

Since hardware DMA devices always operate on physical addressing, they must be able to “scatter” and/or “gather” their transfers according to a scatter gather list (SG list) created by the operating system during memory allocation. The looping and indexing capability of the VDMA sequencer enables complex SG lists to be processed.

# 3. VDMA SEQUENCER

Figure 3-1: VDMA Controller Block Diagram



## 3.1 VDMA Sequencer Operation

FlexDMA engine has a sequencer that controls the raw DMA engines (L2P, P2L DMA masters). The sequencer is effectively a simple processor that executes 32 bit instructions stored in on-chip SRAM (referred to as descriptor memory). Sequencer instructions, referred to as descriptors, define a sequence of DMA operations. Descriptors may have nested loops that allow complex repeating sequences for processing scatter/gather lists.

Descriptor memory is mapped into the BAR0 address space and can be accessed there by the host. Consequently, the host can load a set of descriptors into descriptor memory and then initiate the VDMA sequencer to begin processing.

Descriptor memory may also be written by the VDMA engines based on the instructions embedded in the descriptors. This may be used to provide feedback to driver software for synchronization.

## 3.2 VDMA Descriptor Format

This section describes the format of descriptor entries. Some fields of the descriptor instructions are unused as denoted by a gray and blank field. For future compatibility, all unused fields should be set to 0x0.

Descriptor instruction types are indicated by the top 4 bits of the instruction. They are:

**Table 3-1: Instruction Encoding**

Encoding				Instruction
31	30	31	30	
0	0	0	0	NOP
0	0	0	1	JMP
0	0	1	0	LOAD_RA, LOAD_RB
0	0	1	1	-
0	1	0	0	LOAD_SYS_ADDR
0	1	0	1	STORE_SYS_ADDR
0	1	1	0	ADD_SYS_ADDR
0	1	1	1	-
1	0	0	0	SIG_EVENT
1	0	0	1	WAIT_EVENT
1	0	1	0	STORE_RA, STORE_RB
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	ADD_SYS_ADDR_I
1	1	1	1	LOAD_XFER_CTL

### NOP

No Operation

NOP																																
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0																													

Executes without affecting the internal state of the VDMA sequencer except for the VDMA\_DPTR register which is incremented by 1 to point to the next instruction.

Usage : VDMA\_NOP ( )

### LOAD\_SYS\_ADDR

Load system address register with indexing and an offset. Loads VDMA\_SYS\_ADDR\_L/VDMA\_SYS\_ADDR\_H from the pair of 32 bit descriptor entries at R+ADDRESS where R is either RA (R="10") or RB (R="11") register. In the case of R="00", then ADDRESS is used without an index register being added. Note that system addresses are always 64 bit addresses. They may be stored on any 32 bit boundary in descriptor memory. Note that although the VDMA sequencer can address up to 64K words of descriptor memory, this much memory is typically not implemented. Consequently, addresses that exceed the amount of physical memory will alias back.

For example: for a 4K word descriptor memory, ADDRESS=0x0000 would also be mapped into locations, 0x1000, 0x2000, and so on.

LOAD_SYS_ADDR																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0			R										ADDRESS															

Usage: `VDMA_LOAD_SYS_ADDR(R, ADDR)`

### STORE\_SYS\_ADDR

Store the system address register into descriptor memory: saves VDMA\_SYS\_ADDR\_L and VDMA\_SYS\_ADDR\_H into descriptor memory at R+ADDRESS (VDMA\_SYS\_ADDR\_L) and R+ADDRESS+1 (VDMA\_SYS\_ADDR\_H). The same addressing rules used for LOAD\_SYS\_ADDR are used for STORE\_SYS\_ADDR.

STORE_SYS_ADDR																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1			R										ADDRESS															

Usage: `VDMA_STORE_SYS_ADDR(R, ADDR)`

### ADD\_SYS\_ADDR

Adds the 2's complement value of DATA (signed extended to 32 bits) to the value of the VDMA\_SYS\_ADDR\_L register. This instruction can be used to step through the destination buffer for a series of DMA operations. When DATA is equal the number of bytes per DMA transfer, then the buffer is read/written contiguously. When DATA is greater than the number of bytes per DMA transfer, then this is equivalent to accessing a rectangular window inside the system buffer.

ADD_SYS_ADDR																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0													DATA															

Usage: `VDMA_ADD_SYS_ADDR(DATA)`

### ADD\_SYS\_ADDR\_I

Adds the 32 bit, 2's complement value of the contents located at ADDRESS to the value of the VDMA\_SYS\_ADDR\_L register. ADDRESS is a pointer inside the descriptor memory. This instruction can be used to step through the destination buffer for a series of DMA operations. When DATA is equal the number of bytes per DMA transfer, then the buffer is read/written contiguously. When DATA is greater than the number of bytes per DMA transfer, then this is equivalent to accessing a rectangular window inside the system buffer.

ADD_SYS_ADDR_I																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0													ADDRESS															

Usage: `VDMA_ADD_SYS_ADDR_I ( ADDR )`

## LOAD\_XFER\_CTL

Load the VDMA\_XFER\_CTL Register using indexing with an offset and initiate a DMA transfer. VDMA\_XFER\_CTL is loaded with the contents of the descriptor RAM at the address calculated as R+ADDRESS where R is the contents of either the RA (R="10") or RB (R="11") register. In the case of R="00", then ADDRESS is used without an index register being added.

LOAD_XFER_CTL																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1			R										ADDRESS															

Fields:

R Register select for address offset:

R	Description
"00"	0
"01"	Undefined
"10"	Uses contents of RA as an offset
"11"	Uses contents of RB as an offset

ADDRESS The absolute address of where VDMA\_XFER\_CTL is to be loaded from, in the case of R="00". In the case of R="10" or R="11" then VDMA\_XFER\_CTL is loaded with the value stored at ADDRESS + RA (R="10") or ADDRESS + RB (R="11").

Usage: `VDMA_LOAD_XFER_CTL ( R, ADDR )`

## LOAD\_RA LOAD\_RB

VDMA\_RA/VDMA\_RB is loaded with the contents of the descriptor memory (the lower 16 bits are used) at location ADDRESS.

This instruction is used to maintain loops and also to provide indexing for indexed instructions.

LOAD_RA																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0											ADDRESS															

LOAD_RB																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1											ADDRESS															

Usage: `VDMA_LOAD_RA ( ADDR )`

**ADD\_RA**  
**ADD\_RB**

Data from the lower 16 bits of descriptor memory at location ADDRESS is added (2's complement) to the contents of VDMA\_RA/VDMA\_RB and stored back to VDMA\_RA/VDMA\_RB. This provides a way to add/subtract an offset to VDMA\_RA/VDMA\_RB.

This instruction is used to maintain loops and also to provide indexing for indexed instructions.

ADD_RA																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	0		1									ADDRESS															

ADD_RB																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	1		1									ADDRESS															

Usage : VDMA\_ADD\_RA ( ADDR )

**STORE\_RA**  
**STORE\_RB**

Store the VDMA\_RA/VDMA\_RB registers:

STORE_RA																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0			1	0									ADDRESS															

STORE_RB																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0			1	1									ADDRESS															

VDMA\_RA/VDMA\_RB is stored to the descriptor memory at location ADDRESS.

The STORE\_RA/STORE\_RB instructions are useful when VDMA\_RA/VDMA\_RB are oversubscribed. When they are needed for another context, the previous context may be stored and then loaded back later. This is useful for complex multi-channel DMA scenarios.

Usage : VDMA\_STORE\_RA ( ADDR )

## JMP

Conditional Jump: generalized conditional jump instruction. If the condition is true then the branch address is loaded into the descriptor pointer register (VDMA\_DPTR). This instruction is used to create loops of DMA instructions.

JMP																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	C				CONDITION_SELECT									ADDRESS														

Where:

- **C** is the condition as given below [Table 3-2](#).
- **CONDITION\_SELECT** selects one of up to 256 conditions as specified below in [Table 3-3](#).
- **ADDRESS** is the value that will be placed into VDMA\_DPTR if the branch is taken

**Table 3-2: Condition Codes**

Instruction Bits				Condition	Mnemonic
27	26	25	24		
0	0	0	0	Branch when RA!=0	VDMA_RA_NEQZ
1	0	0	0	Branch when RA=0	VDMA_RA_EQZ
0	0	0	1	Branch when RB!=0	VDMA_RB_NEQZ
1	0	0	1	Branch when RB=0	VDMA_RB_EQZ
0	0	1	0	branch never	VDMA_NEVER
1	0	1	0	Branch always	VDMA_ALWAYS
0	0	1	1	'C' flag equals 0. 'C' is a bit in the VDMA_XFER_CTL register.	VDMA_C_LO
1	0	1	1	'C' flag equals 1.	VDMA_C_HI
0	1	0	0	P2L DMA command queue not full	VDMA_PDM_CMD_QUEUE_FULL_LO
1	1	0	0	P2L DMA command queue full	VDMA_PDM_CMD_QUEUE_FULL_HI
0	1	0	1	L2P DMA command queue not full	VDMA_LDM_CMD_QUEUE_FULL_LO
1	1	0	1	L2P DMA command queue full	VDMA_LDM_CMD_QUEUE_FULL_HI
0	1	1	X	Condition selected by the CONDITION_SELECT field ( <a href="#">Table 3-3</a> ) is low	VDMA_EXT_COND_LO
1	1	1	X	Condition selected by the CONDITION_SELECT field ( <a href="#">Table 3-3</a> ) is high	VDMA_EXT_COND_HI

**Table 3-3: CONDITION\_SELECT Mapping**

CONDITION_SELECT	Condition
0x00	The State of the VDMA_EVENT register EVENT(0)
0x01	The State of the VDMA_EVENT register EVENT(1)
0x02	The State of the VDMA_EVENT register EVENT(2)
0x03	The State of the VDMA_EVENT register EVENT(3)
0x04	The State of the VDMA_EVENT register EVENT(4)
0x05	The State of the VDMA_EVENT register EVENT(5)
0x06	The State of the VDMA_EVENT register EVENT(6)
0x07	The State of the VDMA_EVENT register EVENT(7)
0x08	The State of the VDMA_EVENT register EVENT(8)
0x09	The State of the VDMA_EVENT register EVENT(9)



- When A=1 (assert EVENT bits), then all bits enabled through EVENT\_ENABLE will cause the corresponding EVENT bits in the VDMA\_EVENT register to be asserted (set).
- When A=0 (de-assert EVENT bits), then all EVENT bits enabled through EVENT\_ENABLE are cleared.
- The EVENT\_ENABLE bits are used to selectively assert/de-assert the event bits of the VDMA\_EVENT register. EVENT\_ENABLE bits that contain '0' have no effect on the EVENT bits in the VDMA\_EVENT register. EVENT\_ENABLE bits that contain '1' will set/clear (according to the state of the A bit) the corresponding EVENT bits in the VDMA\_EVENT register.

Usage: VDMA\_SIG\_EVENT(S, A, EVENT\_EN)

## WAIT\_EVENT

Wait for a specific state of the EVENT register.

WAIT_EVENT																															
31	31	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	P				EVENT_ENABLE(11:0)												EVENT_STATE(11:0)											

This instruction stalls execution of the DMA descriptors until the following condition is TRUE:

$((\text{VDMA\_EVENT.EVENT} \text{ xnor } \text{EVENT\_STATE}) \text{ or not } \text{EVENT\_ENABLE}) == 0\text{xFFF}$

Where “xnor” is the bitwise exclusive nor operation, and “or not” are performed as bitwise operations.

The WAIT\_EVENT instruction can only be used with the low 12 bits of the VDMA\_EVENT register.

Usage: VDMA\_WAIT\_EVENT(EVENT\_EN, EVENT\_STATE)

## 3.3 VDMA Register Descriptions

This section describes the VDMA registers. Some of the registers can be accessed from PCI Express address space, most can be manipulated by descriptor instructions.

When VDMA\_CSRx.IPR='1', then most of the VDMA registers are going to be affected by the operation of the VDMA instruction execution and cannot be written by the host processor. They are:

- VDMA\_SYS\_ADDR\_L, VDMA\_SYS\_ADDR\_H
- VDMA\_DPTR
- VDMA\_XFER\_CTL
- VDMA\_RA, VDMA\_RB

VDMA\_EVENT and VDMA\_CSR are always accessible by the host.

**Table 3-4: VDMA Register Summary**

Name	Access		Description
	Host	Sequencer	
VDMA_SYS_ADDR_L	-	LOAD_SYS_ADDR, ADD_SYS_ADDR	VDMA System Address High
VDMA_SYS_ADDR_H	-	LOAD_SYS_ADDR	VDMA System Address High
VDMA_DPTR	RW when idle <sup>1</sup>	JMP and auto-incremented	VDMA Descriptor Pointer
VDMA_XFER_CTL	-	LOAD_XFER_CTL	VDMA Transfer Control
VDMA_RA	-	LOAD_RA	VDMA Register A (RA)
VDMA_RB	-	LOAD_RB	VDMA Register B (RB)
VDMA_EVENT_SET	W	VDMA_EVENT_SET	VDMA Event Set Register
VDMA_EVENT_CLR	W	VDMA_EVENT_CLR	VDMA Event Clear Register
VDMA_EVENT_EN	RW	-	VDMA Event Enable Register
VDMA_EVENT	R	SIG_EVENT	VDMA Event Status Register
VDMA_CSR	RW	SIG_EVENT	VDMA Control/Status Register

1. When VDMA\_CSR.IPR = '1' (DMA is in progress) then VDMA\_DPTR cannot be written by the host and read back state will be unknown. When VDMA\_CSR.IPR = '0' (DMA is idle) then VDMA\_DPTR is RW from the host. Host write to VDMA\_XFER\_CTL when VDMA\_CSR.IPR = '0' (VDMA is idle) would not initiate a DMA operation.

### VDMA\_SYS\_ADDR\_L

### VDMA System Address Low

Bits	Mnemonic	Type	Reset	Description
31:0	ADDRESS	RW	0	<b>Low Address:</b> Low 32 bits of the system address (bits 31:0). This register is modified by the ADD_SYS_ADDR instruction and auto incrementing is not done. When VDMA_SYS_ADDR_L wraps a 32 bit count (0xFFFFFFFF + 1) it wraps back to 0 and does not carry through to the VDMA_SYS_ADDR_H register.

### VDMA\_SYS\_ADDR\_H

### VDMA System Address High

Bits	Mnemonic	Type	Reset	Description
31:0	ADDRESS	RW	0	<b>High Address:</b> Upper 32 bits of the system address (bits 63:32)

### VDMA\_DPTR

### VDMA Descriptor Pointer

Bits	Mnemonic	Type	Reset	Description
31:16	-	R	0	Reserved
15:0	ADDRESS	RW	0	<b>Descriptor Word Address:</b> Address of the next 32 bit descriptor instruction.

### VDMA\_XFER\_CTL

### VDMA Transfer Control

Bits	Mnemonic	Type	Reset	Description
31	C	RW	0	<b>Condition Flag:</b> can be used as a condition for JMP instructions.
30:24	-	R	0	Reserved

Bits	Mnemonic	Type	Reset	Description
23:16	STREAM_ID	RW	0	<b>Stream Identifier:</b> Selects the stream for the DMA operation. The use of this field is determined by the application layer and can be used to select a source/destination channel in the application layer. For L2P DMA, (DIRECTION='0'), the stream ID is sent to the application layer using ports ldm_app_stream_id[7:0]. For P2L DMA, (DIRECTION='1'), the stream ID is sent to the application layer using ports pdm_app_stream_id[7:0]. Stream ID of 0 is reserved. No DMA operation is initiated when the ID is 0.
15:14	-	R	0	Reserved
13	DIRECTION	RW	0	<b>DMA Direction:</b> Selects the direction of data transfer. 0 = <b>Local-to-PCIe:</b> Data is transferred from the application layer and then written to system address space or local memory. This option invokes the L2P DMA master. 1 = <b>PCIe-to-Local Direction:</b> Data is transferred from system address space or local memory to the application layer. This option invokes the P2L DMA master.
12	-	R	0	Reserved
11:0	CNT	RW	0	<b>Transfer Count:</b> Transfer count (bytes). A value of zero indicates 4096 bytes.

## VDMA\_RA

VDMA Register A

## VDMA\_RB

VDMA Register B

The VDMA\_RA and VDMA\_RB registers are used as loop control counters for nesting descriptors.

Bits	Mnemonic	Type	Reset	Description
31:16	-	R	0	Reserved
15:0	D	RW	0	Data:

## VDMA\_EVENT\_SET

VDMA Event Set Register

The VDMA\_EVENT\_SET register is used to cause bits in the VDMA\_EVENT register to be set ('1').

Bits	Mnemonic	Type	Reset	Description
31:16	-	R	0	Reserved
15:0	EVENT	W	0	<b>Set EVENT register bits:</b> When a '1' is programmed into one or more of the EVENT bits, the corresponding VDMA_EVENT.EVENT bit(s) is (are) set. Individual VDMA_EVENT_SET.EVENT bits to '0' has no effect on the operation of the VDMA controller.

## VDMA\_EVENT\_CLR

## VDMA Event Clear Register

The VDMA\_EVENT\_CLR register is used to cause bits in the VDMA\_EVENT register to be cleared ('0').

Bits	Mnemonic	Type	Reset	Description
31:16	-	R	0	Reserved
15:0	EVENT	W	0	<b>Clear EVENT register bits:</b> When a '1' is programmed into one or more of the EVENT bits, the corresponding VDMA_EVENT.EVENT bit(s) is (are) cleared. Individual VDMA_EVENT_CLR.EVENT bits to '0' has no effect on the operation of the VDMA controller.

## VDMA\_EVENT\_EN

## VDMA Event Enable Register

The VDMA\_EVENT\_EN register is used to enable EVENT bits for forwarding to the external hardware (outside of VDMA sequencer). This register has no effect on the operation of the VDMA\_EVENT register.

Bits	Mnemonic	Type	Reset	Description
31:16	-	R	0	Reserved
15:0	EVENT_EN	RW	0	<b>EVENT Enable:</b> VDMA_EVENT_EN.EVENT is bitwise ANDed with VDMA_EVENT.EVENT and then the result is forwarded to the external interrupt controller. EVENT_EN doesn't affect the read back status of VDMA_EVENT.EVENT.

## VDMA\_EVENT

## VDMA Event Status Register

The VDMA\_EVENT register is used together with the SIG\_EVENT instruction to provide a mechanism for communication to external hardware. Typically signals generated by the VDMA\_EVENT register are connected to the interrupt controller. VDMA\_EVENT can be manipulated by descriptors using the SIG\_EVENT instruction, by the host using the VDMA\_EVENT\_SET, VDMA\_EVENT\_CLR registers, and by external hardware through the event interface signals on VDMA sequencer.

Bits	Mnemonic	Type	Reset	Description
31:16	-	R	0	Reserved
15:0	EVENT	R	0	<b>EVENT register bits:</b> These bits are read only. They may be set/cleared through SIG_EVENT instructions or by writing the VDMA_EVENT_SET or VDMA_EVENT_CLR registers.

The VDMA\_CSR register controls the operation of the VDMA controller. VDMA\_CSR is accessible from the host only.

Bits	Mnemonic	Type	Reset	Description
31:2	-	R	0	<b>Reserved</b>
1	HALT	RW	0	<p><b>Halt a DMA In Progress:</b> Controls the run state of the VDMA controller and also acts as a status bit.</p> <p>Write Operation: When this bit is written to '1', and IPR='1', the VDMA will stop processing descriptors as soon as the currently executing instruction completes. Afterwards, the descriptor address register, VDMA_DPTR, will be pointing to the very next instruction to be executed. This allows descriptor processing to be easily continued from where it left off by simply writing the IPR bit to '1'. Writing '1' to HALT when IPR='0' has no effect. Writing '0' to HALT has no effect. See the description below for the case that HALT and IPR are both written '1' at the same time. HALT is cleared by hardware when IPR='0'.</p> <p>Read Operation: When this bit is read as '1', then a halt to DMA operation is pending and once the current instruction is complete, HALT and IPR will be cleared by hardware. When IPR='0', then there is no pending halt of DMA operations either because a previously posted halt has completed or because the VDMA sequencer is idle.</p>
0	IPR	RW	0	<p><b>DMA In Progress:</b> Controls the run state of the VDMA controller and also acts as a status bit. IPR is set by the host and cleared through the SIG_EVENT instruction or by writing the HALT bit.</p> <p>Write Operation: When this bit is written to '1', the VDMA begins processing descriptors at the current VDMA_DPTR descriptor address. Writing '0' has no effect. When IPR and HALT are both written to '1' together, then this has the effect of single stepping the VDMA descriptor processing.</p> <p>Read Operation: When this bit is read as '0', the VDMA is idle and not performing any DMA operations. When IPR='1', then VDMA is processing descriptor instructions.</p>

### 3.4 Byte Handling

The FlexDMA core provides a 64 bit interface for attachment to the User Application. In order to minimize FPGA resources and since not all applications require it, the FlexDMA core doesn't provide byte alignment shifter hardware. However, the core does provide information via the Application Attachment Layer interface to allow the User Application to implement arbitrary byte alignment.

Byte alignment hardware is required when the DMA source and destination are not aligned. For example, consider a DMA in the L2P direction with the following attributes:

- The source data is from a 64 FIFO
- The size of the DMA transfer is to be a multiple of 8 bytes (64 bits)
- The destination of the data is always on an even 64 bit boundary (destination address bits 2:0 are always zero)

In the example above, there is never a requirement to shift the data because everything is 64 bit aligned. However, consider the case where:

- The source data is from a 64 FIFO
- The size of the DMA transfer is to be a multiple of 8 bytes (64 bits)

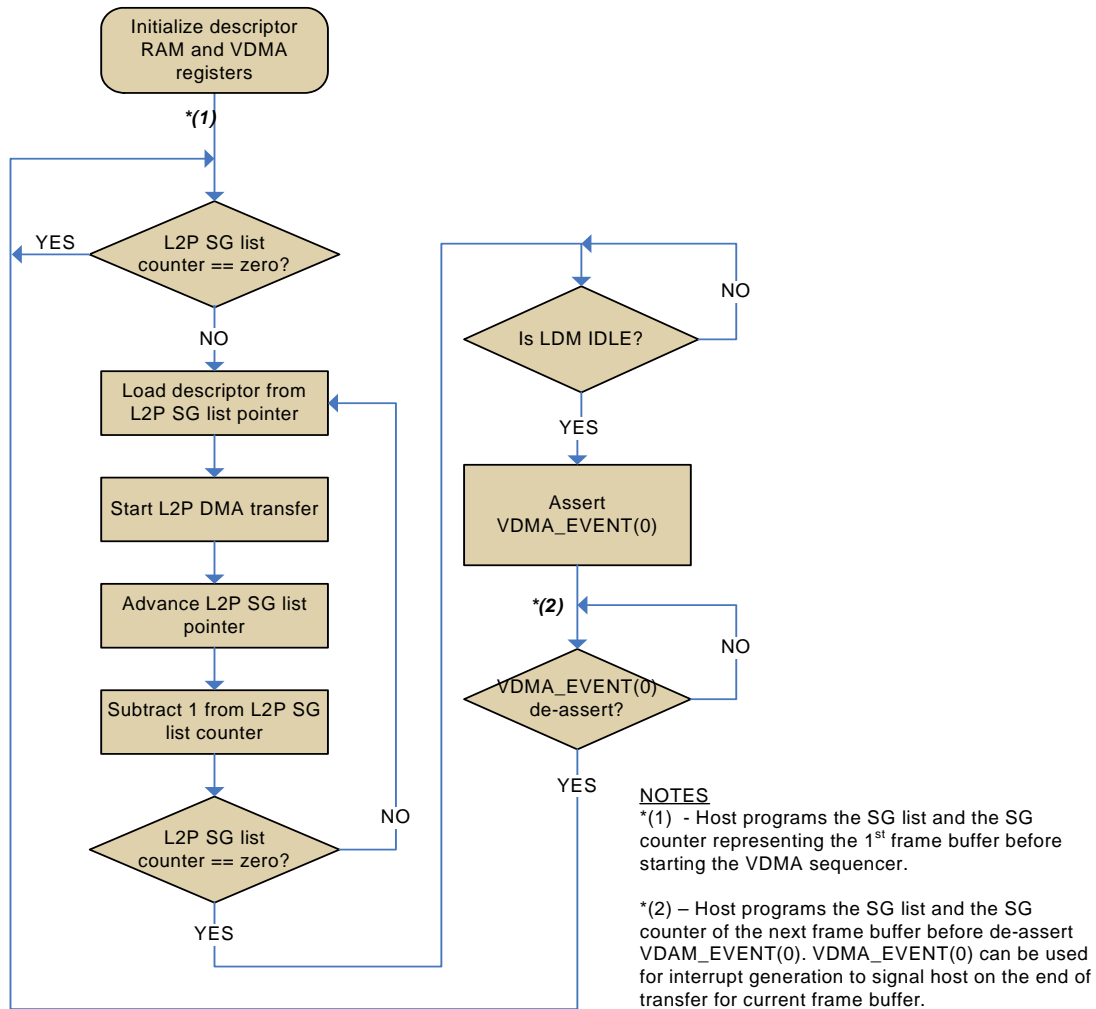
- The destination of the data may be on any byte boundary

In this case, a hardware shifter is required in order to align the source data from the FIFO with the destination. The interface to the Application Attachment Layer provides a set of first/last byte lane enables in order to allow the application layer to properly manage non-aligned transfers.

### 3.5 Example VDMA Programming

This section will describe some pseudo code for performing a video DMA.

**Figure 3-2: Example Scatter/Gather DMA**



#### 3.5.1 Scatter/Gather List Processing

When processing scatter/gather lists, it is desirable to access the list as an array in descriptor memory. For this purpose the indexed instructions are to be used. The following listing is an example of how this may be used to process a scatter/gather list.

```

DWORD    Sequencer
Address  Instructions
-----
0x0000    ..... //Initialization code

0x0010    VDMA_LOAD_RA(0x32) //Load RA with L2P SG list counter
0x0011    VDMA_JUMP(VDMA_RA_EQZ,0,0x10) //If RA != 0 process the L2P SG list
0x0012    VDMA_LOAD_RB(0x33) //Load RB with L2P SG list pointer
0x0013    VDMA_LOAD_SYS_ADDR(VDMA_RB,0) //Load system address from SG entry
0x0014    VDMA_LOAD_XFER_CTL(VDMA_RB,2) //Start L2P DMA
0x0015    VDMA_ADD_RA(0x31) //Subtract 1 from L2P SG list counter
0x0016    VDMA_ADD_RB(0x30) //Advance L2P SG list pointer
0x0017    VDMA_JUMP(VDMA_RA_NEQZ,0,0x13) //If RA != 0 process the next SG entry
0x0018    VDMA_STORE_RA(0x32) //Update L2P SG list counter in memory
0x0019    VDMA_JUMP(VDMA_EXT_COND_LO,LDM_IDLE,0x19) //Do not proceed until LDM becomes idle
0x001A    VDMA_SIG_EVENT(0,0x1,0x1) //Assert VDMA_EVENT(0)
0x001B    VDMA_JUMP(VDMA_EXT_COND_HI,0x1,0x1B) //Do not proceed until VDMA_EVENT(0) is low
0x001C    VDMA_JUMP(VDMA_ALWAYS,0,0x10) //Continue process the next frame

.....

0x0030    CONSTANT 0x3 //Size of SG entry
0x0031    CONSTANT 0xFFFFFFFF //Constant -1
0x0032    CONSTANT 0x0 //L2P SG list counter
0x0033    CONSTANT 0x40 //L2P SG list pointer

.....

0x0040    CONSTANT 0x----- //1st SG entry of frame buffer, VDMA_SYS_ADDR_L
0x0041    CONSTANT 0x----- //1st SG entry of frame buffer, VDMA_SYS_ADDR_H
0x0042    CONSTNAT 0x----- //1st SG entry of frame buffer, VDMA_XFER_CTL

```

The format of the descriptor is given in the following table:

**Table 3-5: Scatter/Gather Data Structure**

Offset		Description
VDMA	Host	
0x00	0x00	System (host) address low 32 bits
0x01	0x04	System (host) address high 32 bits
0x02	0x08	VDMA_XFER_CTL value

Before the host enables descriptor processing (IPR bit in VDMA\_CSRx), the entire range of descriptors should be set to desired values or unused entries cleared. As VDMA processes descriptors, it will write back zero into the SG entry. This allows the host driver to determine where in the list VDMA has completed.

Completed SG list entries can be re-used by the host when it detects completed entries. The host will write the descriptor by first writing the system address and then the VDMA\_XFER\_CTL value. This ensures that a DMA cannot be launched before the system address is in place.

## 4. VDMA ARCHITECTURE

This section describes non-system specific aspects of the VDMA engine.

### 4.1 VDMA Hardware Interface

#### 4.1.1 EVENT Interface

The EVENT interface provides a set of signals to allow the VDMA\_EVENTx register to interact with external hardware.

EVENTo: out std\_ulogic\_vector(N\_EVENT-1 downto 0); --State of the EVENT register  
EVENT\_SETi: in std\_ulogic\_vector(N\_EVENT-1 downto 0); --Allows setting of the EVENT register  
EVENT\_CLRi: in std\_ulogic\_vector(N\_EVENT-1 downto 0); --Allows clearing of the EVENT register

A unit pulse on EVENT\_SETi or EVENT\_CLRi will set/clear EVENTo.

### 4.2 VDMA Internal Memory Map

PCI BAR0 will be used to provide a memory mapped region for all internal resources of the VDMA that are accessible from the host. This includes:

- VDMA\_EVENT, VDMA\_EVENT\_SET, VDMA\_EVENT\_CLR, VDMA\_EVENT\_EN
- VDMA\_CSR
- VDMA\_DPTR

**Table 4-1: BAR0 Memory Map**

Offset	Mode	Register
0x00000	-	-
0x00004	-	-
0x00008	W	VDMA_EVENT_SET
0x0000C	W	VDMA_EVENT_CLR
0x00010	R	VDMA_EVENT
0x00014	RW	VDMA_EVENT_EN
0x00018	RW	VDMA_SYS_ADDR_L
0x0001C	RW	VDMA_SYS_ADDR_L

**Table 4-1: BAR0 Memory Map (Continued)**

Offset	Mode	Register
0x00020	RW	VDMA_DPTR
0x00024	RW	VDMA_XFER_CTL
0x00028	RW	VDMA_RA
0x0002C	RW	VDMA_RA
0x00030	RW	VDMA_CSR
0x4000   0x5FFF	RW	Descriptor RAM

---

**DOCUMENT IDENTIFICATION  
DESIGN GUIDE**

Information relating to this product and the application or design described herein is believed to be reliable, however such information is provided as a guide only and Gennum assumes no liability for any errors in this document, or for the application or design described herein. Gennum reserves the right to make changes to the product or this document at any time without notice.

**CAUTION**

ELECTROSTATIC SENSITIVE DEVICES

DO NOT OPEN PACKAGES OR HANDLE EXCEPT AT A  
STATIC-FREE WORKSTATION

---

**GENNUM CORPORATE HEADQUARTERS**

4281 Harvester Road, Burlington, Ontario L7L 5M4 Canada

Phone: +1 (905) 632-2996

E-mail: [corporate@gennum.com](mailto:corporate@gennum.com)

Fax: +1 (905) 632-2055

[www.gennum.com](http://www.gennum.com)

---

**OTTAWA**232 Herzberg Road, Suite 101  
Kanata, Ontario K2K 2A1  
Canada

Phone: +1 (613) 270-0458

Fax: +1 (613) 270-0429

**CALGARY**3553 - 31st St. N.W., Suite 210  
Calgary, Alberta T2L 2K7  
Canada

Phone: +1 (403) 284-2672

**UNITED KINGDOM**North Building, Walden Court  
Parsonage Lane,  
Bishop's Stortford Hertfordshire, CM23 5DB  
United Kingdom

Phone: +44 1279 714170

Fax: +44 1279 714171

**INDIA**#208(A), Nirmala Plaza,  
Airport Road, Forest Park Square  
Bhubaneswar 751009  
India

Phone: +91 (674) 653-4815

Fax: +91 (674) 259-5733

**SNOWBUSH IP - A DIVISION OF GENNUM**439 University Ave. Suite 1700  
Toronto, Ontario M5G 1Y8  
Canada

Phone: +1 (416) 925-5643

Fax: +1 (416) 925-0581

E-mail: [sales@snowbush.com](mailto:sales@snowbush.com)Web Site: <http://www.snowbush.com>**MEXICO**288-A Paseo de Maravillas  
Jesus Ma., Aguascalientes  
Mexico 20900

Phone: +1 (416) 848-0328

**JAPAN KK**Shinjuku Green Tower Building 27F  
6-14-1, Nishi Shinjuku  
Shinjuku-ku, Tokyo, 160-0023  
Japan

Phone: +81 (03) 3349-5501

Fax: +81 (03) 3349-5505

E-mail: [gennum-japan@gennum.com](mailto:gennum-japan@gennum.com)Web Site: <http://www.gennum.co.jp>**TAIWAN**6F-4, No.51, Sec.2, Keelung Rd.  
Sinyi District, Taipei City 11502  
Taiwan R.O.C.

Phone: (886) 2-8732-8879

Fax: (886) 2-8732-8870

E-mail: [gennum-taiwan@gennum.com](mailto:gennum-taiwan@gennum.com)**GERMANY**Hainbuchenstraße 2  
80935 Muenchen (Munich), Germany

Phone: +49-89-35831696

Fax: +49-89-35804653

E-mail: [gennum-germany@gennum.com](mailto:gennum-germany@gennum.com)**NORTH AMERICA WESTERN REGION**Bayshore Plaza  
2107 N 1st Street, Suite #300  
San Jose, CA 95131  
United States

Phone: +1 (408) 392-9454

Fax: +1 (408) 392-9427

E-mail: [naw\\_sales@gennum.com](mailto:naw_sales@gennum.com)**NORTH AMERICA EASTERN REGION**4281 Harvester Road  
Burlington, Ontario L7L 5M4  
Canada

Phone: +1 (905) 632-2996

Fax: +1 (905) 632-2055

E-mail: [nae\\_sales@gennum.com](mailto:nae_sales@gennum.com)**KOREA**8F Jinnex Lakeview Bldg.  
65-2, Bangidong, Songpagu  
Seoul, Korea 138-828

Phone: +82-2-414-2991

Fax: +82-2-414-2998

E-mail: [gennum-korea@gennum.com](mailto:gennum-korea@gennum.com)

---

Gennum Corporation assumes no liability for any errors or omissions in this document, or for the use of the circuits or devices described herein. The sale of the circuit or device described herein does not imply any patent license, and Gennum makes no representation that the circuit or device is free from patent infringement.

All other trademarks mentioned are the properties of their respective owners.

PCIe and PCI Express mark are registered trademarks and/or service marks of PCI-SIG.

GENNUM and the Gennum logo are registered trademarks of Gennum Corporation.

© Copyright 2009 Gennum Corporation. All rights reserved.

[www.gennum.com](http://www.gennum.com)