



Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!



## Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832

Email & Skype: info@chipsmall.com Web: www.chipsmall.com

Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China





Lattice**CORE**

## Scatter-Gather Direct Memory Access Controller IP Core User Guide

---

---

<b>Chapter 1. Introduction .....</b>	<b>4</b>
Quick Facts .....	4
Features .....	4
<b>Chapter 2. Functional Description .....</b>	<b>5</b>
Key Concepts .....	5
Block Diagram .....	6
WISHBONE Interfaces .....	6
Control and Status .....	6
Channel Arbiter .....	7
BDRAM Interface .....	7
PBUFF Interface .....	7
DMA Engine .....	7
Buffer Status Mode .....	8
AUXCTRL and AUXSTAT .....	9
Primary I/O .....	9
System Configurations .....	11
Interface Descriptions .....	13
Registers and Memory .....	15
Transaction Scenarios .....	18
Requirements and Guidelines .....	20
<b>Chapter 3. Parameter Settings .....</b>	<b>22</b>
User Parameters Tab .....	23
Buses .....	23
Address Decoding .....	24
Channels .....	24
Memory Interfaces .....	25
Generation Options .....	25
Synthesis Optimizations Tab .....	25
Transfer Settings .....	26
<b>Chapter 4. IP Core Generation .....</b>	<b>28</b>
IP Core Generation in IPexpress .....	28
Licensing the IP Core .....	28
Getting Started .....	28
IPexpress-Created Files and Top Level Directory Structure .....	30
Simulation Evaluation .....	31
Implementation Evaluation .....	32
SGDMAC Core Implementation .....	32
IP Core Implementation .....	33
Hardware Evaluation .....	34
Updating/Regenerating the IP Core .....	34
IP Core Generation in Clarity Designer .....	35
Getting Started .....	35
Clarity Designer Created Files and Top Level Directory Structure .....	39
Simulation Evaluation .....	39
IP Core Implementation .....	40
Regenerating/Recreating the IP Core .....	41
Regenerating an IP Core in Clarity Designer Tool .....	41
Recreating an IP Core in Clarity Designer Tool .....	41

---

<b>Chapter 5. Support Resources .....</b>	<b>42</b>
Lattice Technical Support.....	42
E-mail Support .....	42
Local Support .....	42
Internet .....	42
References.....	42
LatticeXP2.....	42
LatticeECP3 .....	42
ECP5.....	42
Revision History .....	43
<b>Appendix A. Resource Utilization .....</b>	<b>44</b>
LatticeECP3 FPGAs.....	44
Ordering Part Number.....	44
LatticeXP2 FPGAs .....	44
Ordering Part Number.....	44
ECP5 LFE5U FPGAs .....	44
Ordering Part Number.....	44
ECP5 LFE5UM FPGAs .....	45
Ordering Part Number.....	45

This user guide describes the Scatter-Gather Direct Memory Access Controller (SGDMAC) IP core for the ECP5™, LatticeECP3™ and LatticeXP2™ families of devices. The Lattice SGDMAC core implements a configurable, multi-channel, WISHBONE-compliant DMA controller with scatter-gather capability. Directions for specifying the IP core’s configuration, including it in a user’s design, and directions for simulation and synthesis are provided in this user’s guide.

## Quick Facts

Table 1-1 gives quick facts about the Scatter-Gather DMA Controller IP core.

**Table 1-1. Scatter-Gather DMA Controller IP Core Quick Facts**

		SGDMAC IP Configuration			
		16 Channel, Dual-bus	4 channel, Dual-bus	8 channel, Dual-bus	4 channel, Dual-bus
<b>Core Requirements</b>	FPGA Families Supported	LatticeECP3, LatticeXP2, ECP5			
<b>Resource Utilization</b>	Targeted Device	LFE3-95EA-7FN672C	LFXP2-40E-6F672C	LFE5U-85F-8BG756C	LFE5UM-85F-8BG756C
	Data Path Width	32	32	32/64	32/8
	LUTs	4311	3443	4049	3222
	Slices	2670	2139	2570	1998
	Registers	1932	1355	1637	1265
	FMAX (MHz)	145	120	160	165
<b>Design Tool Support</b>	Lattice Implementation	Lattice Diamond® 3.4			
	Synthesis	Synopsys® Synplify Pro® for Lattice J-2014.09L			
		Mentor Graphics® Precision® RTL			
	Simulation	Aldec® Active-HDL™ 9.3 SPI Lattice Edition			
Mentor Graphics® ModelSim® SE 6.6e or later					

## Features

- Supports up to 16 physical channels
- Up to 8 sub-channels per physical channel
- Four priority levels using round-robin arbitration (weighted or simple)
- WISHBONE bus widths from 8 to 128 bits
- Simple DMA, split transfers, scatter-gather
- Direct interface to external RAM for packet buffering
- Autonomous and hardware-directed retry
- Supports WISHBONE burst and classic-cycle transfers
- Supports centralized and distributed DMA control architectures

# Functional Description

---

This chapter provides a functional description of the Scatter-Gather DMA Controller core.

## Key Concepts

**Direct Memory Access (DMA)** is a technique for transferring blocks of data between system memory and peripherals without a processor (e.g., system CPU) having to be involved in each transfer. DMA not only offloads a system's processing elements, but can transfer data at much higher rates than processor reads and writes.

**Scatter-Gather DMA** provides data transfers from one non-contiguous block of memory to another by means of a series of smaller contiguous-block transfers.

**Buffer Descriptors** hold the necessary control information for data transfers:

- Source and destination buses and addresses
- Amount of data to be transferred and maximum burst size
- Addressing modes, bus sizes, transaction types, retry options, etc.

Buffer descriptors may be chained together to provide scatter-gather capability.

A **DMA Channel** consists of:

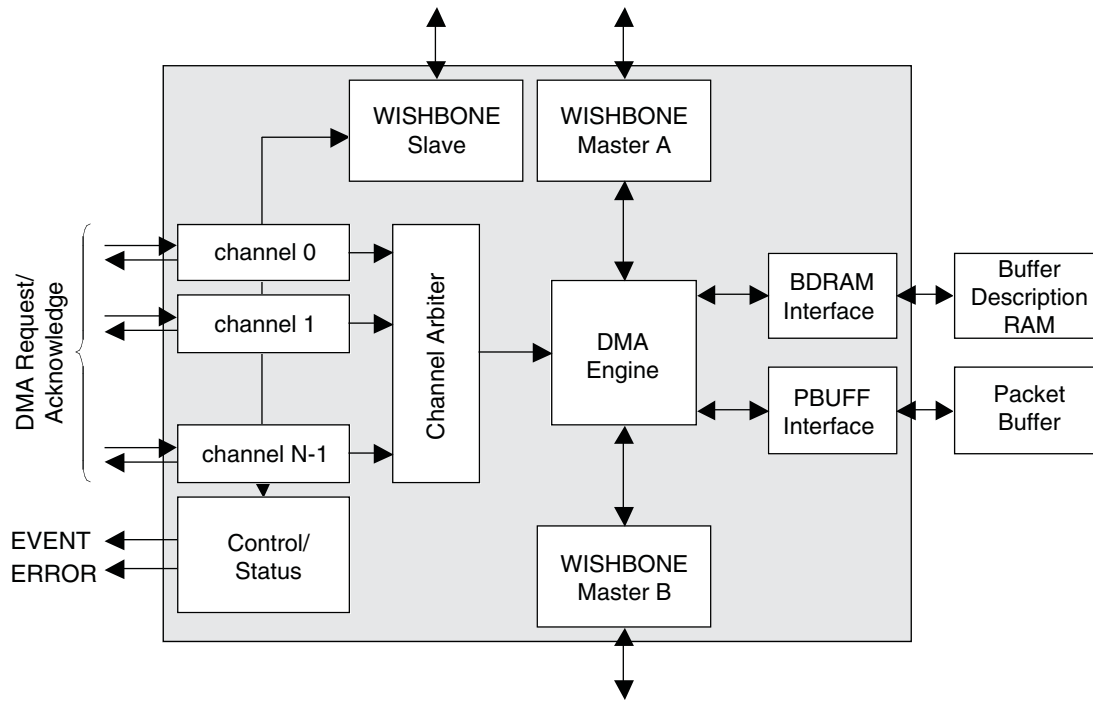
- A set of Buffer Descriptors describing the transfers associated with the channel
- Control and status registers for initiating/observing the transfer process
- An interface to allow the DMA engine access to the channel control and status
- An optional external DMA request/acknowledge signal pair for hardware initiated transfers
- A signal for indicating a pending DMA request to the DMA controller's arbiter and engine

The SGDMAC core provides DMA transfers of data between WISHBONE bus slaves for up to 16 physical DMA channels.

## Block Diagram

The high-level architecture of the Scatter-Gather DMA Controller is shown in Figure 2-1.

**Figure 2-1. SGDMAC Block Diagram**



## WISHBONE Interfaces

The SGDMAC core provides a single WISHBONE slave for accessing registers and memory within the core itself. The slave does full or partial address decoding depending on the `FULL_ADDR_SIZE` parameter. If greater than zero, the upper `FULL_ADDR_SIZE` bits must match the `FULL_ADDR` parameter value. If `FULL_ADDR_SIZE` is zero, the slave address range is being decoding externally, and an active-high `scyc` input indicates a valid cycle.

The core may be configured with either one or two WISHBONE masters. The bus masters are controlled by the DMA Engine. Each master is capable of interacting with both burst-capable and non-burst slaves. Full or partial width bus interactions are allowed (configured in the buffer descriptor).

## Control and Status

Channel control and status registers are accessible through the WISHBONE slave. The registers contain control and current state information for each channel (up to 16 channels). Register details are provided in the Registers and Memory section of this document. Only the registers required for `NUM_CHAN` channels are implemented in the core.

The control and status block also handles external DMA request and acknowledge signals. Each channel control and status register is connected to a single pair of request/acknowledge signals. DMA requests may be generated by hardware via the request signal or by writing the request bit in the channel control and status register.

The control and status block also contains two interrupt registers per channel: one for event interrupts (such as transfer complete notification), the other for errors. Interrupt source registers hold the event until cleared through the slave interface.

## Channel Arbiter

The channel arbiter determines which channel DMA request will be serviced next. For weighted round-robin arbitration, the arbiter consists of four round-robin arbiters, one for each of four priority levels. Each DMA channel makes an appearance on one of the round-robin arbiters as determined by a “priority group” field in its control register. Each of the four round-robin arbiters will handle NUM\_CHAN channels.

The four round-robin arbiters feed a fifth weighted-share arbiter. Each priority group receives a share of the arbiter’s attention that is proportional to the value (0 is lowest, 15 is highest) entered in its “share” control register field. The weights correspond to numbers of transactions without regard to the total amount of data transferred.

Simple round-robin arbitration employs a single NUM\_CHAN-wide round-robin arbiter.

Once the transaction on the active channel is under way, the arbiter is released to choose the next active channel. This arbiter look-ahead feature minimizes the transaction startup latency.

The Global Arbiter control register provides the ability to mask a channel from vying for active channel status. Masking a channel, in effect, freezes a channel in its current state.

## BDRAM Interface

Buffer descriptors are held in an external dual port RAM. The BDRAM interface provides independent read and write access. Reads require a data valid signal to be returned from the BD memory, since the read latency is unknown. Writes require no acknowledgement. The BDRAM is read-write accessible via the WISHBONE slave.

Each channel buffer descriptor head pointer is set in the channel control and status register. Buffer descriptor integrity is the responsibility of the software; the hardware does no checking.

## PBUFF Interface

The SGDMAC core optionally provides an interface to an external packet buffer. The interface is a simple memory interface with separate address and data buses for reads and writes. The Packet Buffer may serve as the source or destination for DMA transactions. The contents of Packet Buffer memory are accessible through DMA transfers, not directly accessible through the SGDMAC WISHBONE slave interface.

## DMA Engine

The DMA Engine uses information stored in the channel buffer descriptors and channel control registers to control the operation of the WISHBONE bus masters. The DMA engine supports the following transactions:

### Simple DMA

Simple DMA transfers are block copies of data from the source address to the destination address. These may be intra- or inter-bus transfers. The active channel remains the active channel until the entire transfer is complete.

### Multi-Burst Transfers

Large blocks of data may have to be split into bursts to avoid bus-hogging by individual channels or priority groups. The maximum burst size is specified by a field in the buffer descriptor. The DMA engine transfers the burst, then tells the channel and channel arbiter that the burst has been transferred. The channel then must compete for attention according to the usual arbitration scheme. When all bursts have been transferred, the channel and channel arbiter are notified, and normal operation resumes. Inter-bus bursts are always locked.

### Multi-Descriptor Transfers

Scatter-Gather operation is implemented using multi-descriptor transfers. A bit in the buffer descriptor tells the DMA engine whether the descriptor is the last in a series. Each descriptor has its own transfer size, burst size, source and destination addresses. When the current burst is complete, the DMA engine fetches the next buffer descriptor, if there is one. The arbiter’s look-ahead feature minimizes the time required between descriptors.

### Split Transactions



Any of the transaction types discussed above may be implemented as split transactions. Split transaction bursts occur in two steps: source to packet buffer, and packet buffer to destination. The advantage offered by split transactions is that each step only occupies one bus. The channel logic maintains the to-packet-buffer/from-packet-buffer sequence. The software subsystem is responsible for ensuring buffer offsets and burst sizes are set appropriately to prevent channel data overlap.

### Direct Packet Buffer Transactions

The Packet Buffer may be selected as the source or destination for simple DMA transactions by setting the SRC\_BUS or DST\_BUS field in the buffer descriptor. When the packet buffer is the source (or destination), the corresponding address in the buffer descriptor is ignored; the channel packet buffer offset is used instead.

### Delayed Transactions

Delayed transactions occur when a source or destination WISHBONE slave signals a retry in response to the access request from the SGDMAC. Depending on the state of the RETRY bit in the buffer descriptor, the SGDMAC either: if RETRY is set, relinquishes control and re-attempts to become the active channel; or, relinquishes control, clears its DMA\_REQUEST bit, and waits for the delayed peripheral to activate the channel's dma request input. It's the responsibility of the peripheral to activate the request only when it can respond without retry. The number of retries is determined by the NUM\_RETRY field in the channel control and status register. Exceeding NUM\_RETRY results in an error. See the sections on Autonomous Retry and Hardware Retry below.

### EOD Transactions

For some transactions, the data source may not have available the number of bytes designated by the buffer descriptor. The SGDMAC core uses a WISHBONE data tag (user-defined set of signals synchronous with the WISHBONE slave output data) to signal EOD to the master requesting the data. In response to the EOD tag, the WISHBONE master terminates the cycle and signals the DMA engine. The DMA engine signals transfer complete to the channel logic and arbiter.

### Errored Transactions

The scope of error checking by the SGDMAC core is limited to its field of view. For example, address values, transfer sizes, buffer descriptor memory allocation and alignment, and packet buffer overlap are conditions that the core (by design) lacks the information to detect and address. It is the responsibility of the configuration software to ensure control information integrity.

The core is able to detect bus errors and retry errors. Transactions that encounter errors freeze the active channel state. The channel will not vie to become the active channel again until the errors have been cleared and the channel reset (by disabling and enabling it).

### Freezing Channels

Channels may be prevented from vying for active channel status by way of the GARBITER.CHARBMSK bits. A channel with its corresponding CHARBMSK bit set has its arbiter request bit masked. Because the channel never becomes the active channel, it remains in its current state. Note that this differs from disabling a channel, which returns the channel logic to its initialization state.

### Bus Locking

Setting the LOCK bit in the buffer descriptor causes the WISHBONE master to request a locked transfer. Inter-bus, non-split transfers are always locked. Intra-bus and split transfers are locked only if LOCK is set.

### Buffer Status Mode

Setting the BUFFER\_STATUS user parameter to '1' provides logic for checking and updating buffer availability. In this mode, the CONFIG0.BD\_STATUS\_EN bit enables status checking. If '1', the DMA engine checks CONFIG0.BD\_STATUS. A '1' indicates that the requested buffer is available: the transfer proceeds as usual, and the dma\_engine clears CONFIG0.BD\_STATUS upon completion of the transfer. If CONFIG0.BD\_STATUS is '0', the requested buffer is unavailable, the channel's buffer descriptor error bit is set, and the transfer is terminated. If CONFIG0.BD\_STATUS\_EN is '0', the buffer status check is skipped, and the transfer proceeds as usual.

## AUXCTRL and AUXSTAT

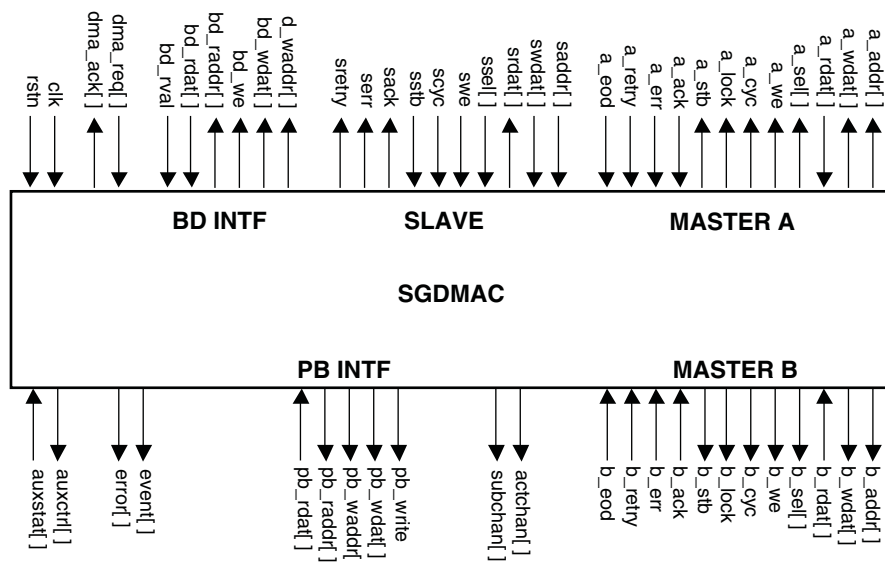
The optional auxctrl and auxstat ports provide auxiliary read (auxstat) and write (auxctrl) capability. The read-only auxstat port operates in pass-through mode - that is, there are no registers associated with the port. The write-only auxctrl port is only active during a slave write, zeros otherwise.

**Memory Interfaces:** The “Number of Buffer Descriptors” option configures the size of the BD (buffer descriptor) memory read and write address buses. There are four 32-bit words per buffer descriptor. 256 is the maximum number of descriptors allowed. The “Packet Buffer Size” option is the number of bytes in the external packet buffer. This item sets the packet buffer address bus sizes. The data bus is always the size of the largest WISHBONE bus.

## Primary I/O

The top-level interface diagram is shown in Figure 2-2 and a brief description of the signals is given in Table 2-1.

**Figure 2-2. SGDMAC Core Primary I/O**



**Table 2-1. Top-Level Port Definitions**

Port	Size	I/O	Description
<b>Global Signals</b>			
clk	1	I	System clock
rstn	1	I	System wide asynchronous active-low reset signal.
<b>A-Bus Master Signals</b>			
a_addr	AWIDTH	O	A-Bus master address output
a_wdat	DWIDTHA	O	A-Bus master write data
a_rdat	DWIDTHA	I	A-Bus master read data
a_sel	DWIDTHA/8	O	A-Bus master byte selects, active high
a_we	1	O	A-Bus master write enable output, active high
a_cyc	1	O	A-Bus master valid transfer cycle output, active high
a_lock	1	O	A-Bus master lock request to bus arbiter
a_stb	1	O	A-Bus master data strobe output, active high
a_cti	3	O	A-Bus master cycle type identifier, active high
a_ack	1	I	A-Bus master acknowledge, active high
a_err	1	I	A-Bus master error acknowledge, active high
a_retry	1	I	A-Bus master retry, active high
a_eod	1	I	A-Bus master end-of-data flag
<b>B-Bus Master Signals</b>			
b_addr	AWIDTH	O	B-Bus master address output
b_wdat	DWIDTHB	O	B-Bus master write data
b_rdat	DWIDTHB	I	B-Bus master read data
b_sel	DWIDTHB/8	O	B-Bus master byte selects, active high
b_we	1	O	B-Bus master write enable output, active high
b_cyc	1	O	B-Bus master valid transfer cycle output, active high
b_lock	1	O	B-Bus master lock request to bus arbiter
b_stb	1	O	B-Bus master data strobe output, active high
b_cti	3	O	B-Bus master cycle type identifier, active high
b_ack	1	I	B-Bus master acknowledge, active high
b_err	1	I	B-Bus master error acknowledge, active high
b_retry	1	I	B-Bus master retry, active high
b_eod	1	I	B-Bus master end-of-data flag
<b>Slave Signals</b>			
saddr	AWIDTH	I	Slave address input
swdat	32	I	Slave write data
srdat	32	O	Slave read data
ssel	4	I	Slave byte selects, active high
swe	1	I	Slave write enable input, active high
scyc	1	I	Slave valid transfer cycle input, active high
sstb	1	I	Slave data strobe input, active high
sack	1	O	Slave acknowledge, active high
serr	1	O	Slave error acknowledge, active high
sretry	1	O	Slave retry, active high
<b>Buffer Descriptor Memory Interface</b>			
bd_waddr	BD_AWIDTH	O	Buffer descriptor write address

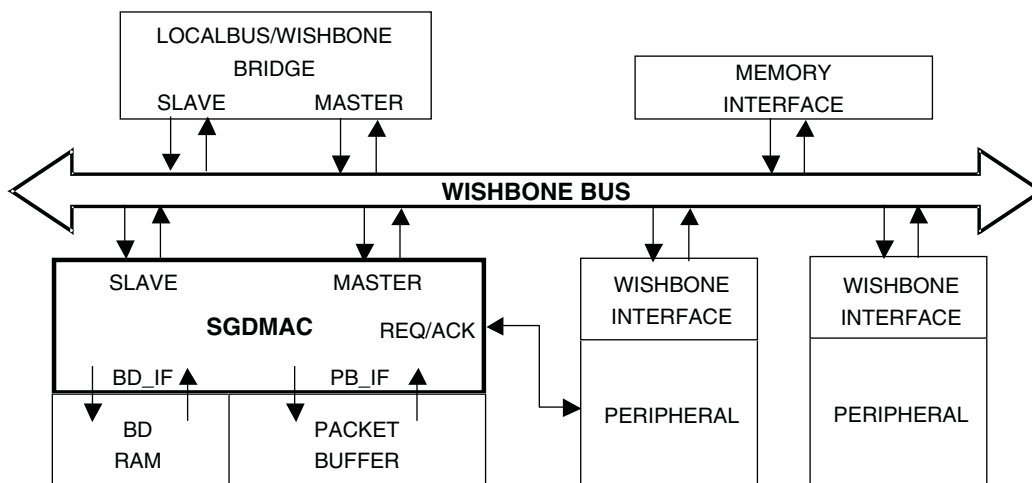
**Table 2-1. Top-Level Port Definitions (Continued)**

Port	Size	I/O	Description
bd_wdat	32	O	Buffer descriptor write data
bd_we	1	O	Buffer descriptor write enable, active high
bd_re	1	O	Buffer descriptor read request, active high
bd_raddr	BD_AWIDTH	O	Buffer descriptor read address
bd_rdat	32	I	Buffer descriptor read data
bd_rval	1	I	Buffer descriptor read data valid
bd_err	1	O	Buffer status check error
<b>Packet Buffer Interface</b>			
pb_write	1	O	Packet buffer write enable, active high
pb_wdat	DWIDTHA	O	Packet buffer write data
pb_waddr	PB_AWIDTH	O	Packet buffer write address
pb_raddr	PB_AWIDTH	O	Packet buffer read request, active high
pb_rdat	DWIDTHA	I	Packet buffer read data
pb_rval	1	I	Packet buffer read data valid
<b>Interrupt and Control</b>			
dma_req[]	NUM_CHAN	I	DMA requests, active high
dma_ack[]	NUM_CHAN	O	DMA acknowledge, active high
event[]	NUM_CHAN	O	Event interrupts
error[]	NUM_CHAN	O	Error interrupts
actchan[]	CWIDTH	O	Active channel number
subchan[]	SUBWIDTH	O	Sub-channel value
auxctrl[]	16	O	Auxiliary control outputs
auxstat[]	16	I	Auxiliary control inputs

## System Configurations

### Single-WISHBONE

A typical single-WISHBONE configuration is illustrated in Figure 2-3.

**Figure 2-3. SGDMAC in a Single-WISHBONE System**


The SGDMAC core Master and Slave interfaces are connected to the WISHBONE bus. The SLAVE data width is always 32 bits. The MASTER data width should be the full width of the bus. Single-bus configurations require a

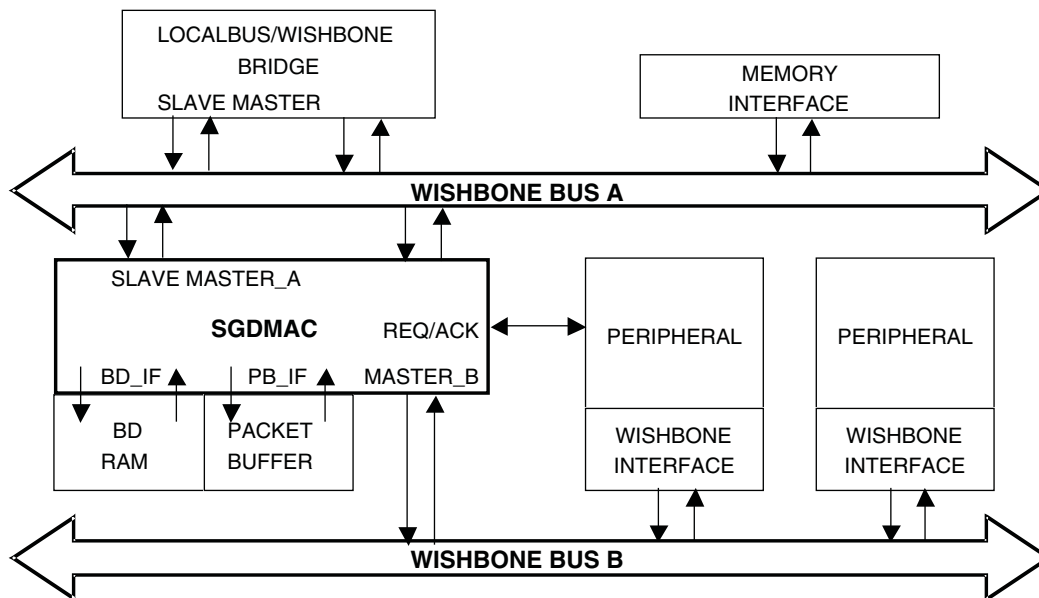
Packet Buffer, since all intra-bus transfers are implemented as split transactions. The SGDMAC Master competes with other bus masters for bus ownership. Peripherals are connected to the WISHBONE bus by their slave (and perhaps master) interfaces, and may also have request/acknowledge connections to the SGDMAC to allow peripheral-initiated transfers.

Peripherals need not occupy the full bus width; if not, they should be connected to the low-order WISHBONE data signals for both Big- and Little-Endian systems (for example, an eight-bit port would connect to D0-D7). Transfers between slaves with dissimilar port widths are handled by the SGDMAC core. For example, for a data transfer from an 8-bit peripheral to a 32-bit peripheral, the read portion of the transfer would occur 8 bits at a time, the write portion 32 bits at a time. Transfers to and from WISHBONE slaves should always utilize their full bus widths, both to achieve the greatest throughput and to avoid confusion about which portion of the bus should be active (especially in Big-Endian systems).

**Dual-WISHBONE**

A typical dual-WISHBONE configuration is illustrated in Figure 2-4.

**Figure 2-4. SGDMAC in a Dual-WISHBONE System**



Higher throughput may be achieved by adding a second WISHBONE bus. A dual-bus SGDMAC core performs both intra- and inter-bus transfers. The same bus-width considerations apply for dual-bus configurations as for single-bus. The two WISHBONE buses need not be of the same topology; for example, one may be a shared bus and the other a crossbar switch. They do, however, need to have the same Endian-ness.

Inter-bus transfers do not require split transactions. Unlike intra-bus transfers, inter-bus transfers do not use Packet Buffer resources (in fact, the Packet Buffer is optional if only inter-bus transfers are required). A small FIFO in the SGDMAC core provides temporary storage for assembly/disassembly of data transferred from bus to bus. For non-split transfers, both buses are owned by the SGDMAC for the duration of each burst. Split transactions, on the other hand, transfer a full burst of data from the source bus to the Packet buffer, then from the Packet Buffer to the destination bus, and only occupy one bus at a time. Bus traffic characteristics will determine whether split or non-split transactions provide the greatest overall throughput.

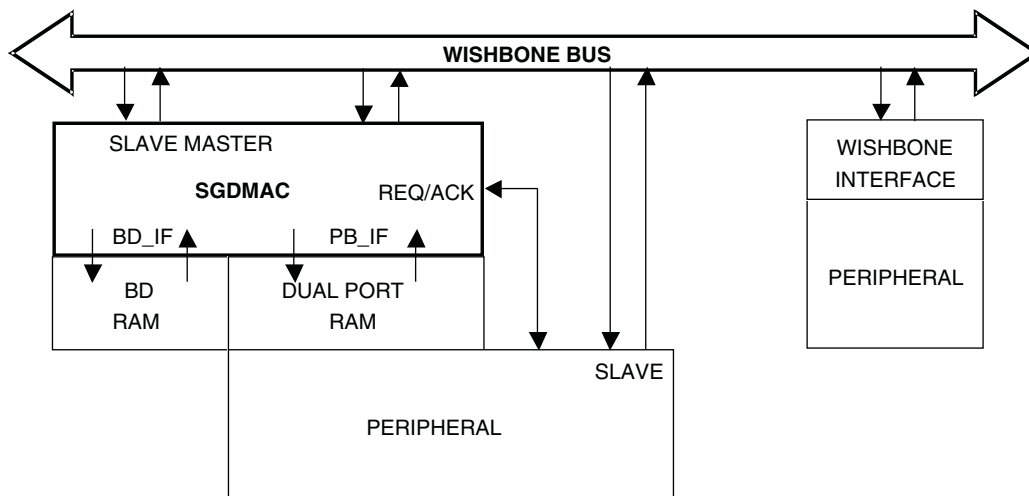
**Distributed DMA**

The single- and dual-bus configurations above are examples of centralized DMA control: a single controller handles the DMA transfers for a set of WISHBONE peripherals. For some systems, a better solution might be to distribute the DMA function to the bus clients themselves, allowing them to initiate and accept transfers to and from other clients. This approach offers the following advantages:

- Reduces bus traffic. In a centralized DMA system, two bus transactions are required for each datum moved. With distributed DMA, most data transfers require only one bus transaction.
- The number of available DMA channels grows with the addition of each DMA-capable peripheral.
- Supports full-interconnect bus topologies (crossbar switch, for example) for higher throughput. Multiple bus master-slave pairs can exchange data simultaneously.

The SGDMAC core provides an easy way to add DMA capability to peripheral devices by using the Packet Buffer interface and the request/acknowledge ports. A possible implementation of a DMA-enabled peripheral is illustrated in [Figure 2-5](#).

**Figure 2-5. DMA-Capable Peripheral with SGDMAC Core**



The SGDMAC core in this example is used in single-bus mode, although dual-bus mode would also work (the peripheral could transfer data between itself and either bus). A small buffer descriptor RAM provides only those descriptors needed by the peripheral. A dual-port RAM attached to the core's Packet Buffer interface provides the data path, and the request/acknowledge signals allow the peripheral to initiate transfers and recognize transfer completion. The core's slave interface may be connected to the wishbone bus for channel and buffer descriptor setup or, for smart peripherals, there might be a direct connection between the peripheral and the core's slave port.

## Interface Descriptions

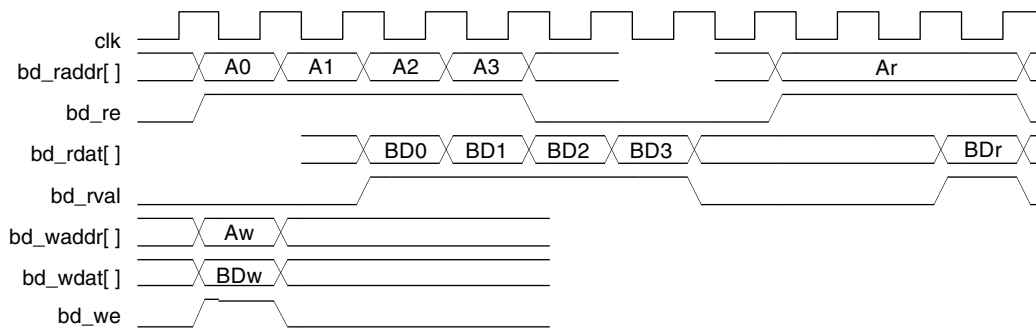
### WISHBONE Interfaces

All WISHBONE interfaces are compliant with WISHBONE Specification B.3. The Cycle Indicator tags are used for burst transactions. They are sourced by the WISHBONE masters and valid when the strobe signal (stb) is active. The SGDMAC WISHBONE master interfaces also support an End-Of-Data tag. Buffer Descriptor Interface. The eod signal may be returned by bus slaves and is valid when the acknowledge (ack) signal is active.

### Buffer Descriptor Memory Interface

The interface to the buffer descriptor RAM uses a simple synchronous handshake for reads and writes, as shown in [Figure 2-6](#).

**Figure 2-6. Buffer Descriptor Interface Timing**

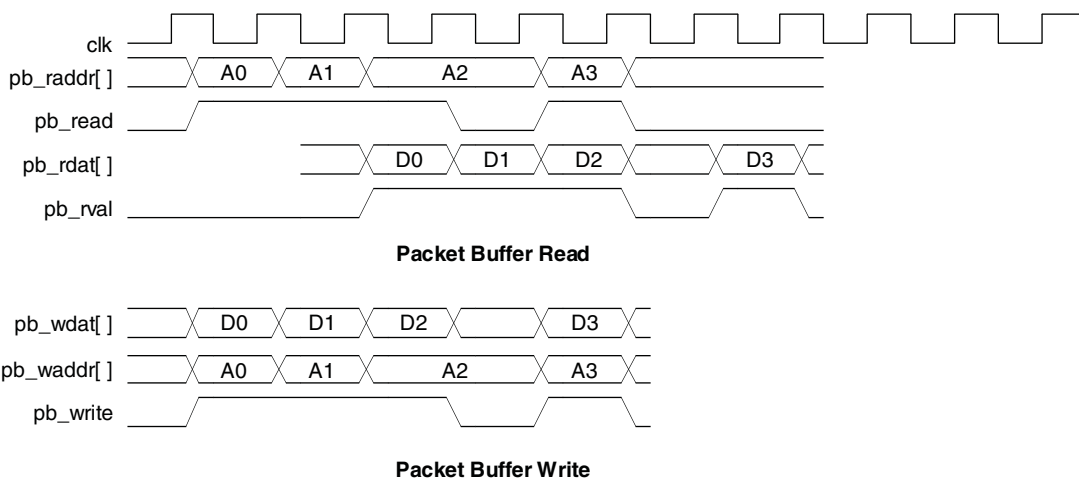


The SGDMAC core fetches a sequence of four 32-bit words from the user-provided Buffer Descriptor memory using a four clock-cycle burst. The active-high `bd_re` signal, accompanied by a read address on `bd_raddr`, signals the read request. The BD memory responds with an active-high `bd_rval`, accompanied by buffer descriptor data. The request-to-valid-data interval is under the control of the BD memory. WISHBONE slave initiated reads are presented to the Buffer Descriptor memory as active high `bd_re` and `bd_raddr`, both of which are held until the BD memory responds with an active-high `bd_rval` and corresponding data. BD memory writes are presented as a single clock-cycle assertion of `bd_we` accompanied by write address `bd_waddr` and write data `bd_wdat`. The SGDMAC core assumes that the BD memory accepts the write, so no acknowledge signal is required.

### Packet Buffer Interface

The interface to the external Packet Buffer memory also uses a simple handshake to transfer data, as shown in Figure 2-7.

**Figure 2-7. Packet Buffer Interface Timing**



The `pb_read` signal is asserted along with the read address. The PB memory responds some number of clock-cycles later with `pb_rval` and the read data.

*Note: Because the PB memory's read latency is unknown to the core, the SGDMAC may perform more reads than necessary to complete the burst. The data from these superfluous reads are unused, and the next burst will begin at the appropriate PB read address. This should note pose a problem for Packet Buffers implemented as normal RAMs, but prevents the use of Packet Buffers implemented as FIFOs.*

For packet buffer writes, the `pb_write` signal is asserted with the write address and data. The SGDMAC core assumes that the PB memory can accept the write, so no acknowledge signal is required.

**Registers and Memory**
**Table 2-2. Registers and Memory**

Name	Addr (hex)	Width	Access	Description
<b>Global Registers</b>				
IPID	0	32	R	IP identification register
IPVER	4	32	R	IP version register
GCONTROL	8	32	RW	Global control register
GSTATUS	c	32	RW	Global status register
GEVENT	10	32	RC1	Global channel event register and mask
GERROR	14	32	RW	Global channel error register and mask
GARBITER	18	32	RW	Global arbiter control register
GAUX	1c	32	RW	Auxiliary inputs and outputs
<b>Channel N Registers</b>				
CONTROLN	$N \ll 5 + 200$	32	RW	Control register
STATUSN	$N \ll 5 + 204$	32	RW	Status register
CURSRCN	$N \ll 5 + 208$	32	R	Current source register
CURDSTN	$N \ll 5 + 20c$	32	R	Current destination register
CURXFERCNT	$N \ll 5 + 210$	32	R	Current transfer count
PBOFFSETN	$N \ll 5 + 214$	32	RW	Packet buffer start address
<b>Buffer Descriptor X</b>				
CONFIG0X	$X \ll 4 + 400$	32	RW	Control register
CONFIG1X	$X \ll 4 + 404$	32	RW	Status register
SRC_ADDRX	$X \ll 4 + 408$	32	RW	Source address
DST_ADDRX	$X \ll 4 + 40c$	32	RW	Destination address

**Global Registers**
**Table 2-3. IPID - IP Identification Register (address = 0)**

Field Name	Bits	Access	Description
VENDORID	31:16	R	Vendor ID (1204)
IPNUM	15:0	R	IP number (TBD)

**Table 2-4. IPVER - IP Version Register (address = 4)**

Field Name	Bits	Access	Description
MAJOR	31:24	R	Major version number
MINOR	23:16	R	Minor version number
NUMCHAN	15:12	R	Number of channels supported by this configuration
unused	11		
NUMSUB	10:8	R	Number of sub-channels supported by this configuration
CAPABLE	7:0	R	Capability bits 0: B-Bus present (0-no, 1-yes) 1: Packet Buffer present (0-no, 1-yes) 2: Endian-ness (1-Big, 2-Little)



**Table 2-5. GCONTROL - Global Control Register (address = 8)**

Field Name	Bits	Access	Description
CHENABLE	15:0	RW	Channel enable bits
CHMASK	31:16	RW	Channel request mask - '1' masks channel's DMA request input

**Table 2-6. GSTATUS - Global Status Register (address = c)**

Field Name	Bits	Access	Description
CHACTIVE	15:0	R	Each bit is a copy of the corresponding channel's REQUEST bit
AENABLE	29	RW	'1' enables A-Bus Master operation, '0' resets
BENABLE	30	RW	'1' enables B-Bus Master operation, '0' resets
GENABLE	31	RW	'0' holds all SGDMAC machines in their initialization state; '1' enables normal operation.

**Table 2-7. GEVENT - Global Event Register (address = 10)**

Field Name	Bits	Access	Description
CHEVENT	15:0	R	Each bit is a copy of the corresponding channel's XFERCOMP bit
CHEVMSK	31:16	RW	Masks the channels' event outputs

**Table 2-8. GERROR - Global Error Summary and Mask (address = 14)**

Field Name	Bits	Access	Description
CHERR	15:0	R	Summary error bits for each channel. Clears when channel errors are cleared (or masked)
CHERRMSK	31:16	RW	Masks the channels' summary error bit outputs

**Table 2-9. GARBITER - Global Arbiter Control (address = 18)**

Field Name	Bits	Access	Description
SHARE0	3:0	RW	Priority group 0 fair share value (0-15)
SHARE1	7:4	RW	Priority group 1 fair share value
SHARE2	11:8	RW	Priority group 2 fair share value
SHARE3	15:12	RW	Priority group 3 fair share value
CHARBMSK	31:16	RW	Masks channel logic requests for service to arbiter

**Table 2-10. GAUX - Global Auxiliary Control and Status Register (address = 1c)**

Field Name	Bits	Access	Description
AUXCTRL	15:0	W	Drives auxctrl outputs during slave write only, '0's otherwise.
AUXSTAT	31:16	R	Reflects value of auxstat inputs

### Per Channel Registers

**Table 2-11. CONTROLx - Channel Control Register (address = channel<<5 + 200)**

Field Name	Bits	Access	Description
PRIGRP	7:6	RW	Priority group
ERRMASK	15:8	RW	Error mask - '1' masks error output, error still registered
BDBASE	31:16	RW	Buffer descriptor base index

**Table 2-12. Channel Status Register (address = channel<<5 + 204)**

Field Name	Bits	Access	Description
ENABLED	0	R	Copy of channel enable in global control register
REQUEST	1	RW	DMA request active on this channel. Set by external DMA request signal or SW set on write to 1. Cleared by channel when transfer complete.
XFERCOMP	2	R	'1' indicates current transfer complete
EOD	3	R	'1' indicates End-Of-Data condition encountered
CLRCOMP	4	W	'1' clears XFERCOMP and EOD
unused	6:5		
RTRYCNT	11:7	R	Retry count
STATE	15:12	R	Channel logic machine's current state (values TBD)
ERRORS	23:16	RC1	Channel errors (clear on write to 1) 16: bus error 17: local address out of range 18: timeout error 19: illegal retry

**Table 2-13. CURSRCx - Current Source Address (address = channel<<5 + 208)**

Field Name	Bits	Access	Description
SRCADDR	31:0	R	Current source address

**Table 2-14. CURDSTx - Current Destination Address (address = channel<<5 + 20c)**

Field Name	Bits	Access	Description
DSTADDR	31:0	R	Current destination address

**Table 2-15. CURXFERCNTx - Current Transfer Count (address = channel<<5 + 210)**

Field Name	Bits	Access	Description
CNT	15:0	R	Number of bytes transferred so far
CURR_BD	31:16	R	Current BD pointer

**Table 2-16. PBOFFSETx - Packet Buffer Offset for Channel x (address = channel<<5 + 214)**

Field Name	Bits	Access	Description
OFFSET	31:0	RW	Packet Buffer start address

## Buffer Descriptors

**Table 2-17. CONFIG0 - Buffer Descriptor Configuration 0 (address = bd<<4 + 400)**

Field Name	Bits	Access	Description
EOL	0	RW	End of BD sequence
SPLIT	1	RW	Split transaction
LOCK	2	RW	Bus locking
AUTORETRY	3	RW	Channel does autonomous retry
RETRYTHRESH	7:4	RW	Retry threshold for this transaction
SRC_BUS	9:8	RW	Source bus (00=A, 01=B, 10=PB)
SRCBUS_SIZE	12:10	RW	Source bus transaction data width (log2(number_of_bytes))
SRCINCR	14:13	RW	Source address increment mode (00-none, 01-linear, 10-loop)
unused	15		
DST_BUS	17:16	RW	Destination bus (00=A, 01=B, 10=PB)
DSTBUS_SIZE	20:18	RW	Destination bus transaction data width (log2(number_of_bytes))

**Table 2-17. CONFIG0 - Buffer Descriptor Configuration 0 (address =  $bd \ll 4 + 400$ ) (Continued)**

Field Name	Bits	Access	Description
DSTINCR	22:21	RW	Destination address increment mode (00-none, 01-linear, 10-loop)
unused	23		
SUBCHAN	26:24	RW	Sub-channel value
unused	28:27		
BD_NEXT	29	RW	If '1', start next transaction at next BD, otherwise BDBASE.
BD_STATUS	30	RW	BD status – '1' means buffer available.
BD_STATUS_EN	31	RW	'1' enables buffer status behavior for this BD.

**Table 2-18. CONFIG1 - Buffer Descriptor Configuration 1 (address =  $bd \ll 4 + 404$ )**

Field Name	Bits	Access	Description
XFER_SIZE	15:0	RW	Total transfer size in bytes
BURST_SIZE	31:16	RW	Maximum burst size in bytes

**Table 2-19. SRC\_ADDRESS - Source Address (address =  $bd \ll 4 + 408$ )**

Field Name	Bits	Access	Description
SRC	31:0	RW	Source address

**Table 2-20. DST\_ADDRESS - Buffer Descriptor Configuration 0 (address =  $bd \ll 4 + 40c$ )**

Field Name	Bits	Access	Description
DST	31:0	RW	Destination address

## Transaction Scenarios

### Initialization

**Channel Initialization:** Channel initialization occurs when a channel is disabled. A disabled channel has its STATUS register's REQUEST, XFERCOMP, and ERRORS fields cleared. CONTROL register fields PRIGRP and BDBASE retain their current values, and ERRMASK is set to all '1's. Before enabling a channel, its PRIGRP and BDBASE fields must be set to valid values.

To reinitialize a channel, simply disable it via its corresponding CHENABLE bit in the Global Control Register (GCONTROL), set its control fields to their appropriate values, then enable it.

**Global Initialization:** Power-up and GSR resets have the following effect on the global registers:

- IP identification and version registers have fixed logic values and are unaffected by reset.
- In GCONTROL, CHENABLE is all zeros, disabling all channels, and CHMASK is all '1's, masking all request inputs.
- Since all channels are disabled, the CHACTIVE bits in GSTATUS will be '0'. AENABLE, BENABLE, and GENABLE initialize to '0'.
- Since all channels are disabled, the CHEVENT bits in GEVENT will be '0'. The CHEVMSK bits will all be '1', masking the event outputs.
- Since all channels are disabled, the CHERR bits in GERROR will be '0'. The CHERRMSK bits initialize to '1', masking the error outputs.
- In GARBITER, all SHAREx fields initialize to zero. All CHARBMSK bits init to zero (allows all channels' requests to arbiter).

### The Overall Initialization Sequence

- Set up channel control for channels to be enabled, including buffer descriptors.
- Set up arbiter properties.
- Enable WISHBONE masters.
- Enable SGDMAC core.
- Enable channels.

### Simple DMA

- Set the channel's head buffer descriptor values with EOL bit set to '1', BURST\_SIZE  $\geq$  XFER\_SIZE.
- Set the STATUSx.REQUEST bit to '1'. (Alternatively, the DMA request input may be unmasked and the transaction initiated by a '1' on the request input.)
- DMA engine signals WISHBONE master(s) to request bus access. Masters have bus control until transfer complete.
- Upon completion, channel logic clears STATUSx.REQUEST, sets STATUSx.XFERCOMP, and signals DMA acknowledge to the requesting peripheral.
- Transfer completion handling may be: fire-and-forget; poll STATUSx.XFERCOMP for a '1', then write CONTROLx.CLRCOMP to '1' to clear STATUSx.XFERCOMP; or respond to channel event interrupt by writing CONTROLx.CLRCOMP to '1' to clear STATUSx.XFERCOMP. Hardware initiated requests are terminated by dma\_ack[x]. Withdrawing dma\_req[x] clears both STATUSx.XFERCOMP and dma\_ack[x].

### Multi-Burst Transfers

- Same as simple DMA, except BURST\_SIZE  $<$  XFER\_SIZE in head buffer descriptor.
- DMA engine transfers BURST\_SIZE data, then signals channel that burst is complete. Channel requests servicing for bursts until XFER\_SIZE has been transferred.
- Transfer completion is the same as simple DMA

### Multiple-Descriptor Transfers

- Same as simple DMA, except multiple buffer descriptors are set up in sequence. Only final descriptor has EOL set to '1'. Each buffer descriptor has its own XFER\_SIZE and BURST\_SIZE.
- DMA engine completes each transfer as in multi-burst case. Channel logic increments STATUSx.CURR\_BD.
- Transfer completion is the same.

### Split Transactions

- Same as any of the above, except SPLIT bit in descriptor is set.
- DMA engine signals source WISHBONE master only, data for first burst is transferred to packet buffer.
- DMA engine signals burst complete to arbiter, which reselects active channel.
- When split transaction channel is serviced again, DMA engine signals destination WISHBONE master only, data for first burst is transferred from packet buffer. Burst complete sent to arbiter.
- Repeat sequence for all bursts until destination half of final burst is complete.
- Transfer completion as usual

**Packet Buffer as Source or Destination**

- Set channel packet buffer offsets so that channel space is large enough to hold XFER\_SIZE
- Same as any other non-split transaction, but set SRC\_BUS (or DST\_BUS) field in buffer descriptor to select Packet Buffer.

**Autonomous Retry**

- During any burst transfer, a WISHBONE master may receive a retry indication during bus arbitration. If AUTORETRY is set in descriptor, follow this sequence.
- DMA engine signals retry to channel and arbiter. Arbiter moves to next active channel. Channel logic increments retry count.
- When retry channel is again serviced, proceed as before. If retry count exceeds RETRYTHRESH (in descriptor), signal error and freeze channel logic.
- Otherwise, continue until no retry occurs.

**Hardware Retry**

- If AUTORETRY is not set, channel responds to retry signal from DMA engine by deactivating its STATUSx.REQUEST bit.
- '1' on DMA request input for this channel starts the process beginning with this burst.
- Continue as before.

**EOD**

- During any burst in a transaction, the source bus WISHBONE master receives an End-Of-Data flag. The WISHBONE master relinquishes control of the bus and signals EOD to the DMA engine.
- The DMA engine terminates the entire transaction as usual, even in the middle of a sequence of bursts, and sets the STATUSx.EOD to a '1', in addition to setting STATUSx.XFERCOMP.
- When CONTROLx.CLRCOMP is written to '1', STATUSx.EOD is cleared along with STATUSx.XFERCOMP.

**Buffer Status Checking**

- In Buffer Status mode, if status checking is enabled for the requested buffer (CONFIG0.BD\_STATUS\_EN is '1') and the buffer is marked as unavailable (CONFIG0.BD\_STATUS is '0'), the DMA engine immediately terminates the transfer by setting STATUSx.XFERCOMP and clearing STATUSx.REQUEST. STATUSx.ERRORs[4] is set, indicating a buffer descriptor error.
- The SGDMAC core takes no autonomous action to recover from a buffer descriptor error.

**Requirements and Guidelines****Transfer and Burst Sizes**

- Both transfer and burst sizes must be integer multiples of the largest bus size. (Reason: the internal FIFO is the width of the largest bus, and it is read and written with full-width data words only.)
- For multiple buffer descriptor transfers (scatter-gather mode), each descriptor has its own transfer and burst sizes. The SGDMAC core has no information about the total transfer size for multi-descriptor transactions.
- Internal address, transfer, and burst counters only increment in their lower 16 bits. This imposes two restrictions: (a) Transfer and burst sizes are limited to 64K bytes; (b) in address auto-increment mode, the address must not cross a 64K boundary.

- For transfers to/from the packet buffer, transfer size must be less than or equal to the space allocated for the channel in packet buffer memory.
- For split transactions and intra-bus transfers, burst size must be less than or equal to the space allocated for the channel in packet buffer memory.

**WISHBONE Connections**

- The SGDMAC WISHBONE master ports should occupy the full width of the buses to which they're attached.
- WISHBONE bus slaves may have narrower data ports than the bus to which they are attached. In this case, they must be connected to the lower bus data bits for both Big- and Little-Endian systems.

**Sources and Destinations**

- Intra-bus transfers are always implemented as split transactions. This is done autonomously by the core, so setting the split bit in the buffer descriptor is not required.
- Do not do split transactions with the Packet Buffer as source or destination (one of the transfer steps will be PB-to-PB).
- Packet buffer source and destination addresses always start at the channel's packet buffer offset. The source (or destination) address in the buffer descriptor is ignored.

**Channels and Sub-Channels**

- The actchan and subchan output ports of the SGDMAC core carry the active channel number and sub-channel number for the duration of each burst, and may be useful for interfaces outside the scope of the standard WISHBONE/SGDMAC interaction. For example, they could be used as WISHBONE data tags. Or, the sub-channel value could be used to form part of the address into Packet Buffer memory to provide sub-channel granularity.

# Parameter Settings

The IPexpress™ tool is used to create IP and architectural modules in the Diamond software for the LatticeECP3 and LatticeXP2 device families. Refer to the [IP Core Generation](#) section for a description on how to generate the IP. Clarity Designer tool is used to create IP in the Diamond software for the ECP5 device family. Refer to [IP Core Generation](#) section for more information.

Table 3-1 provides the list of user configurable parameters for the SGDMAC IP core. The parameter settings are specified using the SGDMAC IP core Configuration GUI in IPexpress or Clarity Designer.

**Table 3-1. SGDMAC Parameters**

Parameters	Range/Options	Default Value
<b>Buses</b>		
Bus A Data Width	8, 16, 32, 64, 128	32
Bus B Data Width	0, 8, 16, 32, 64, 128	32
Address Width	16-32	32
Byte Order	0 or 1	Little Endian
Add Auxiliary Ports auxstat and ausctl	0=Unselected 1=Selected	Not selected
<b>Address Decoding</b>		
Address Decode Size	0 to 24	0
Address Match Value	0 to FFFFFFFF	0
<b>Channels</b>		
Number of Channels	1 to 16	16
Number of Sub-Channels	0 to 8	4
Arbiter Type	0 or 1	Simple Round-robin (value: 0)
Enable Buffer Status Check/Update	0 = Disabled 1 = Enabled	Disabled
<b>Memory Interfaces</b>		
Number of Buffer Descriptions	1 to 65536	256
Packet Buffer Size	0 to 65536	4096
<b>Generation Options</b>		
Clock Frequency (MHz)		150
Behavioral Model		Enabled
Netlist (.ngo)		Enabled
Evaluation Directory		Enabled
<b>Transfer Settings</b>		
Source Bus	A=0 B=1 C=2	From Buffer Descriptor
Source Bus Size	8, 16, 32, 64, 128	From Buffer Descriptor
Source Address	0 to FFFFFFFF	From Buffer Descriptor
Destination Bus	A=0 B=1 C=2	From Buffer Descriptor
Destination Bus Size	8, 16, 32, 64, 128	From Buffer Descriptor
Destination Address	0 to FFFFFFFF	From Buffer Descriptor

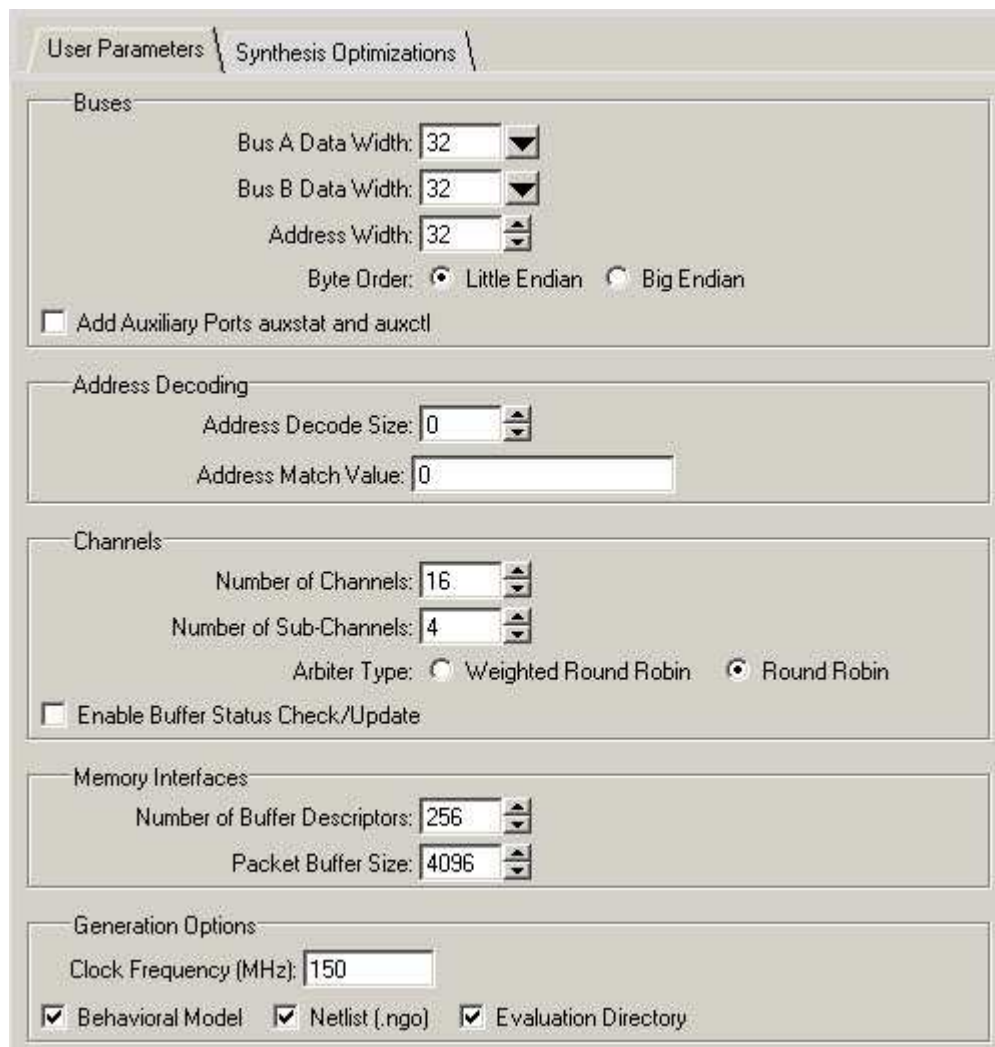
**Table 3-1. SGDMAC Parameters (Continued)**

Parameters	Range/Options	Default Value
LOCK Value	0=off 1=on	From Buffer Descriptor
AUTORETRY Value	0=off 1=on	From Buffer Descriptor
AutoRetry Threshold	0 to 15	From Buffer Descriptor

## User Parameters Tab

Figure 3-1 shows the contents of the User Parameters tab.

**Figure 3-1. User Parameters Tab**



The screenshot shows the 'User Parameters' tab with the following settings:

- Buses:**
  - Bus A Data Width: 32
  - Bus B Data Width: 32
  - Address Width: 32
  - Byte Order:  Little Endian  Big Endian
  - Add Auxiliary Ports auxstat and auxctl
- Address Decoding:**
  - Address Decode Size: 0
  - Address Match Value: 0
- Channels:**
  - Number of Channels: 16
  - Number of Sub-Channels: 4
  - Arbiter Type:  Weighted Round Robin  Round Robin
  - Enable Buffer Status Check/Update
- Memory Interfaces:**
  - Number of Buffer Descriptors: 256
  - Packet Buffer Size: 4096
- Generation Options:**
  - Clock Frequency (MHz): 150
  - Behavioral Model  Netlist (.ngo)  Evaluation Directory

## Buses

This section sets the data widths for the A and B buses and the common address width. Setting “Bus B Data Width” to zero results in a single-bus configuration. “Byte Order” applies to both buses. Checking the “Add Auxiliary Ports auxstat and auxctl” box adds the auxiliary ports to the core.



**Bus A Data Width**

This option sets the Bus A data width.

**Bus B Data Width**

This option sets the Bus B data width. 0 = single-bus configuration.

**Address Width**

This option sets the Address width.

**Byte Order**

This option sets the Byte order.

- 0=Little Endian
- 1=Big Endian

*Note: This parameter applies to both buses.*

**Add Auxiliary Ports auxstat and auxctl**

When checked, this options adds the auxiliary ports to the core.

**Address Decoding**

“Address Decode Size” sets the number of high-order address bits the WISHBONE slave will use for partial address decoding. “Address Match Value” sets the value the bits will be compared against. An “Address Decode Size” of zero means the WISHBONE bus is doing full address decoding and no address matching by the core.

**Address Decode Size**

This option sets the full address decode size.

*Note: a value of 0 means the WISHBONE bus is doing full addressing (no address matching is done by the core).*

**Address Match Value**

This option sets the full address decode match value.

**Channels**

The Channels section sets the number of channels and sub-channels (per channel). The Arbiter type choices are weighted and simple round-robin. Choose simple round-robin for a much more compact, higher performance arbitration module. The Enable Buffer Status Check/Update option provides the ability to provide circuitry for buffer status handling. The check/update is enabled/disabled per buffer descriptor).

**Number of Channels**

This option sets the number of DMA channels.

**Number of Sub-Channels**

This option sets the number of DMA sub-channels.

**Arbiter Type**

This option sets the determines weighted or simple round-robin arbitration:

- 0 = Round-robin is selected
- 1 = Weighted is selected

**Enable Buffer Status Check/Update**

This option provides circuitry for buffer status handling:

- 0 = Unselected
- 1 = Selected

## Memory Interfaces

The “Number of Buffer Descriptors” option configures the size of the BD (buffer descriptor) memory read and write address buses. There are four 32-bit words per buffer descriptor. 65536 is the maximum number of descriptors allowed. The “Packet Buffer Size” option is the number of bytes in the external packet buffer. This item sets the packet buffer address bus sizes. The data bus is always the size of the largest WISHBONE bus.

### Number of Buffer Descriptions

This option sets the number of buffer descriptors.

### Packet Buffer Size

This option sets the number of bytes in the external packet buffer.

## Generation Options

### Clock Frequency (MHz)

This option sets the fMAX setting for core synthesis and eval implementation map, place, and route (more on eval later).

### Behavioral Model

This option generates a configuration-specific non-synthesizeable behavioral model for the core.

### Netlist

This option synthesizes the user-specified core configuration in .ngo format.

### Evaluation Directory

This option creates a configuration-specific implementation directory where the user may map, place, and route the core to obtain performance and utilization results.

## Synthesis Optimizations Tab

Every data transfer performed by the SGDMAC is controlled by values normally held in a 4-word section of memory called a buffer descriptor. When a channel is chosen by the arbiter to be the active channel, the dma engine fetches these control values (also known as transaction parameters) from buffer descriptor memory.

Several of the buffer descriptor transaction parameters may be fixed values for some applications. Substantial logic savings are possible by setting these values at synthesis time rather than retrieving them from the buffer descriptor memory. Source and Destination address maybe set to hexadecimal values via their respective text entry field. An entry field is provided for retry threshold.

Figure 3-2 shows the contents of the Synthesis Optimization tab.

**Figure 3-2. Synthesis Optimization Tab**

