LatticeCORE

Scatter-Gather Direct Memory Access Controller IP Core User Guide

March 2015
IPUG67_1.8

Table of Contents

Introduction

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
Core Requirements	FPGA Families Supported	LatticeECP3, LatticeXP2, ECP5			
Resource Utilization	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
Design Tool Support	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:

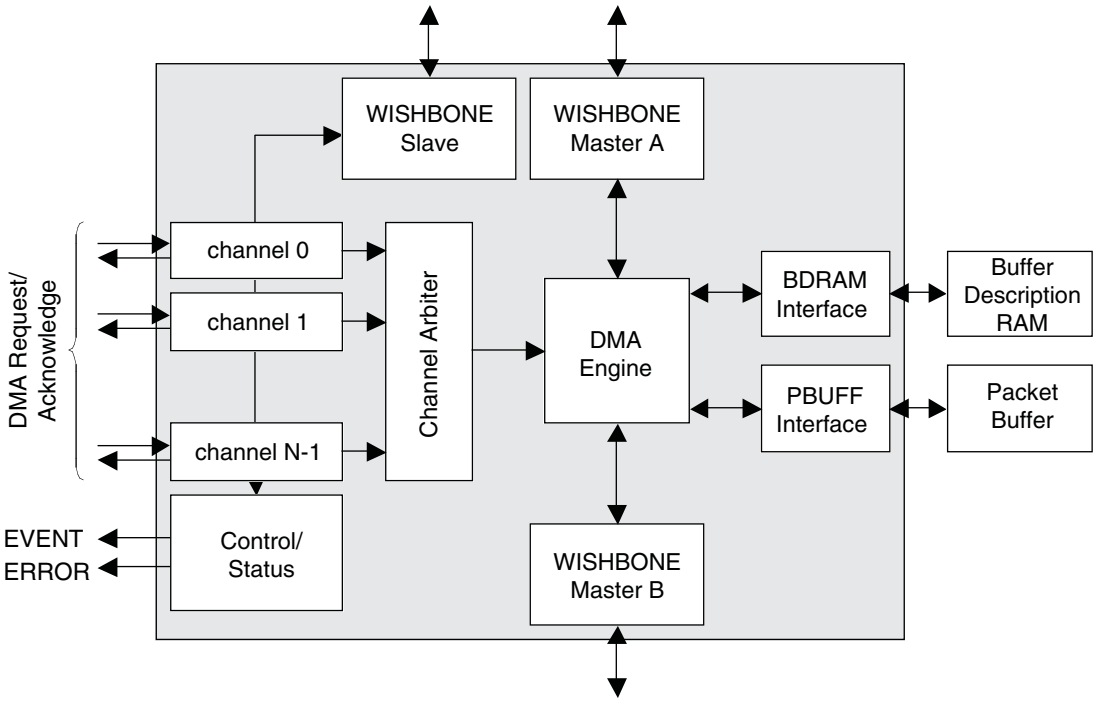
- A set of Buffer Descriptors describing the transfers associated with the channel
- Control and status registers for initiating/observing the transfer process
- An interface to allow the DMA engine access to the channel control and status
- An optional external DMA request/acknowledge signal pair for 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 auxctl 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

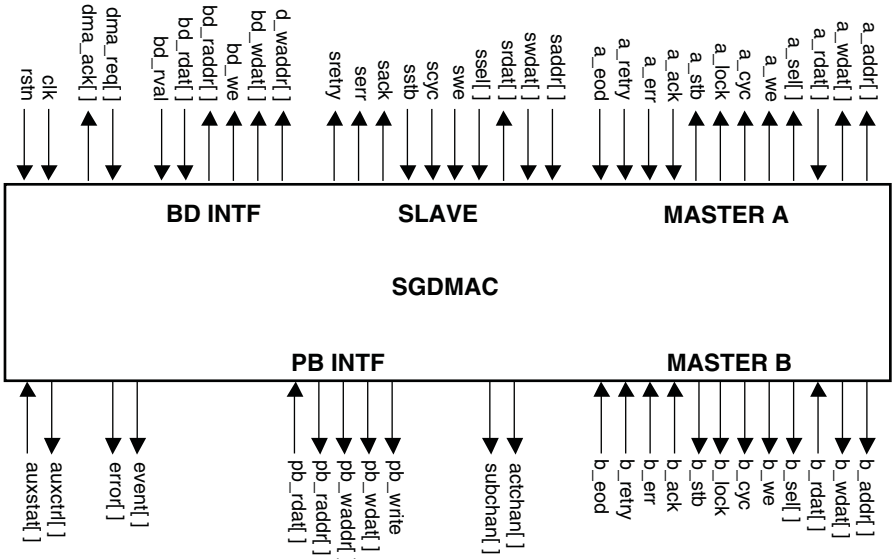


Table 2-1. Top-Level Port Definitions

Port	Size	I/O	Description
Global Signals			
clk	1	I	System clock
rstn	1	I	System wide asynchronous active-low reset signal.
A-Bus Master Signals			
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-Bus Master Signals			
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
Slave Signals			
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
Buffer Descriptor Memory Interface			
bd_waddr	BD_AWIDTH	O	Buffer descriptor write address

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

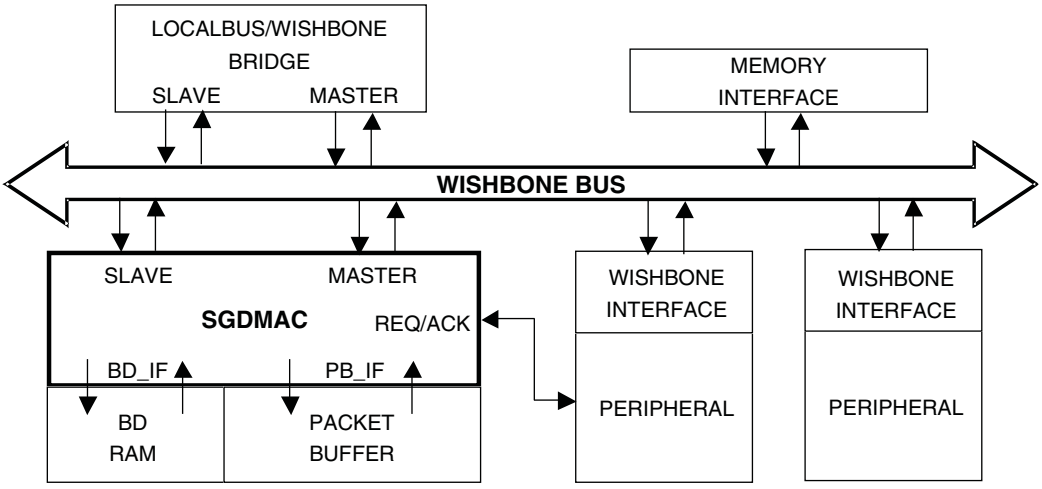
Port	Size	I/O	Description
bd_wdat	32	O	Buffer descriptor write data
bd_we	1	O	Buffer descriptor write enable, active high
bd_re	1	O	Buffer descriptor read request, active high
bd_raddr	BD_AWIDTH	O	Buffer descriptor read address
bd_rdat	32	I	Buffer descriptor read data
bd_rval	1	I	Buffer descriptor read data valid
bd_err	1	O	Buffer status check error
Packet Buffer Interface			
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
Interrupt and Control			
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

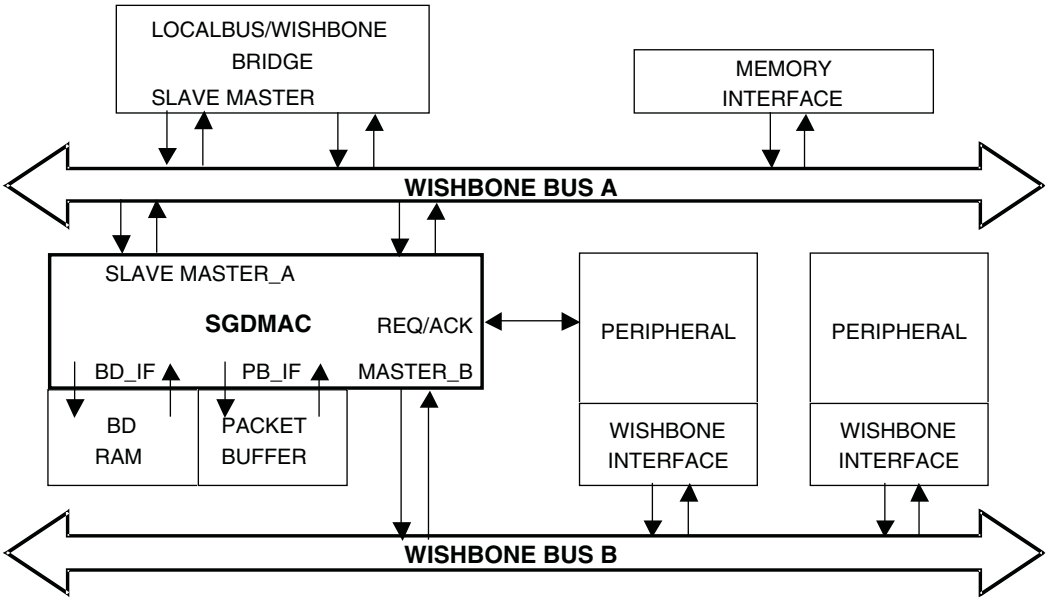
Packet Buffer, since all intra-bus transfers are implemented as split transactions. The SGDMAC Master competes with other bus masters for bus ownership. Peripherals are connected to the WISHBONE bus by their slave (and perhaps master) interfaces, and may also have request/acknowledge connections to the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

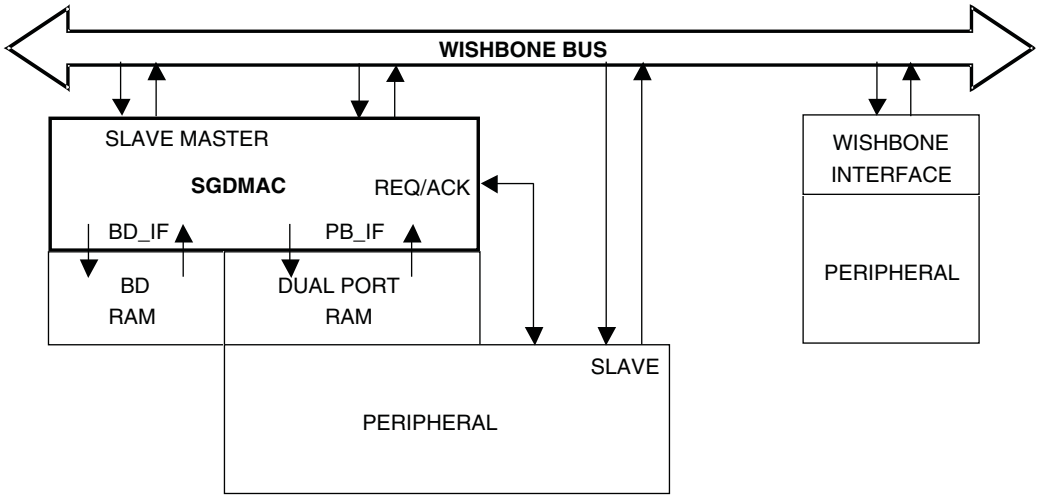
Distributed DMA

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

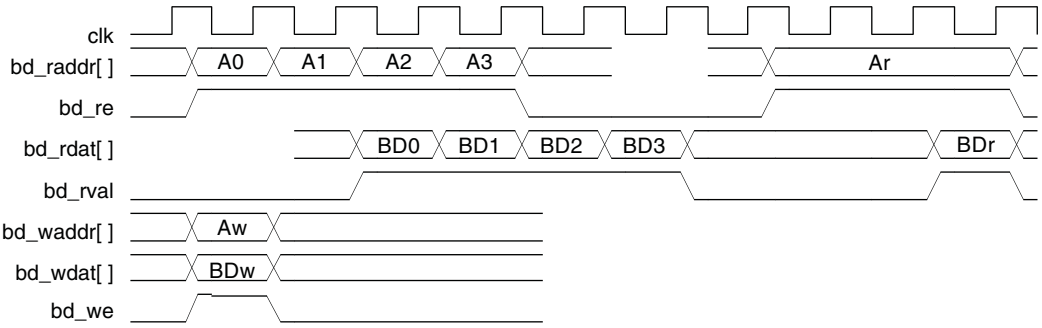
WISHBONE Interfaces

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

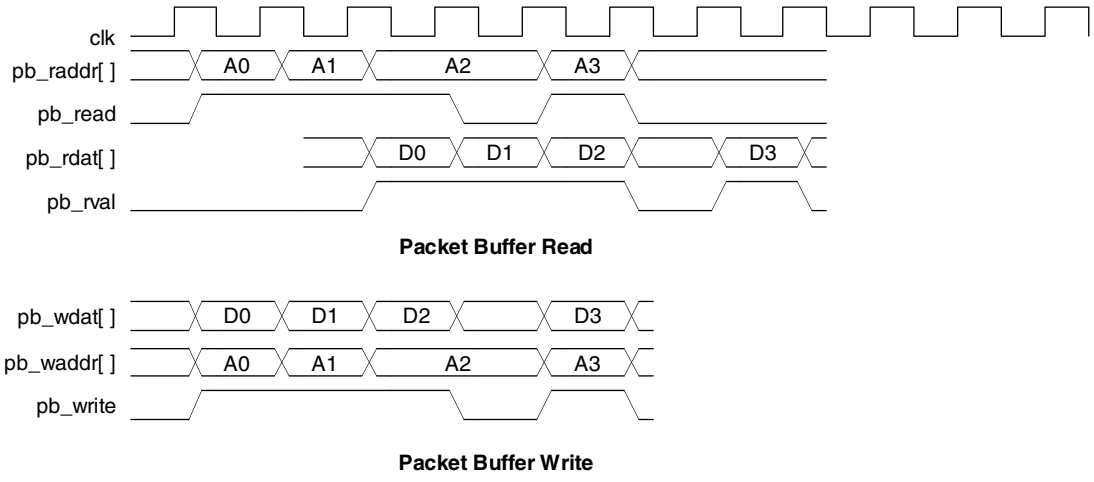


The SGDMAC core fetches a sequence of four 32-bit words from the user-provided Buffer Descriptor memory using a four clock-cycle burst. The active-high bd_re signal, accompanied by a read address on bd_raddr, signals the read request. The BD memory responds with an active-high bd_rval, accompanied by 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
Global Registers				
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
Channel N Registers				
CONTROLN	N<<5 + 200	32	RW	Control register
STATUSN	N<<5 + 204	32	RW	Status register
CURSRCN	N<<5 + 208	32	R	Current source register
CURDSTN	N<<5 + 20c	32	R	Current destination register
CURXFERCNT	N<<5 + 210	32	R	Current transfer count
PBOFFSETN	N<<5 + 214	32	RW	Packet buffer start address
Buffer Descriptor X				
CONFIG0X	X<<4 + 400	32	RW	Control register
CONFIG1X	X<<4 + 404	32	RW	Status register
SRC_ADDRX	X<<4 + 408	32	RW	Source address
DST_ADDRX	X<<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<<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<<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<<4 + 408)

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

Table 2-20. DST_ADDRESS - Buffer Descriptor Configuration 0 (address = bd<<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 >= 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
Buses		
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
Address Decoding		
Address Decode Size	0 to 24	0
Address Match Value	0 to FFFFFFFF	0
Channels		
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
Memory Interfaces		
Number of Buffer Descriptions	1 to 65536	256
Packet Buffer SIze	0 to 65536	4096
Generation Options		
Clock Frequency (MHz)		150
Behavioral Model		Enabled
Netlist (.ngo)		Enabled
Evaluation Directory		Enabled
Transfer Settings		
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

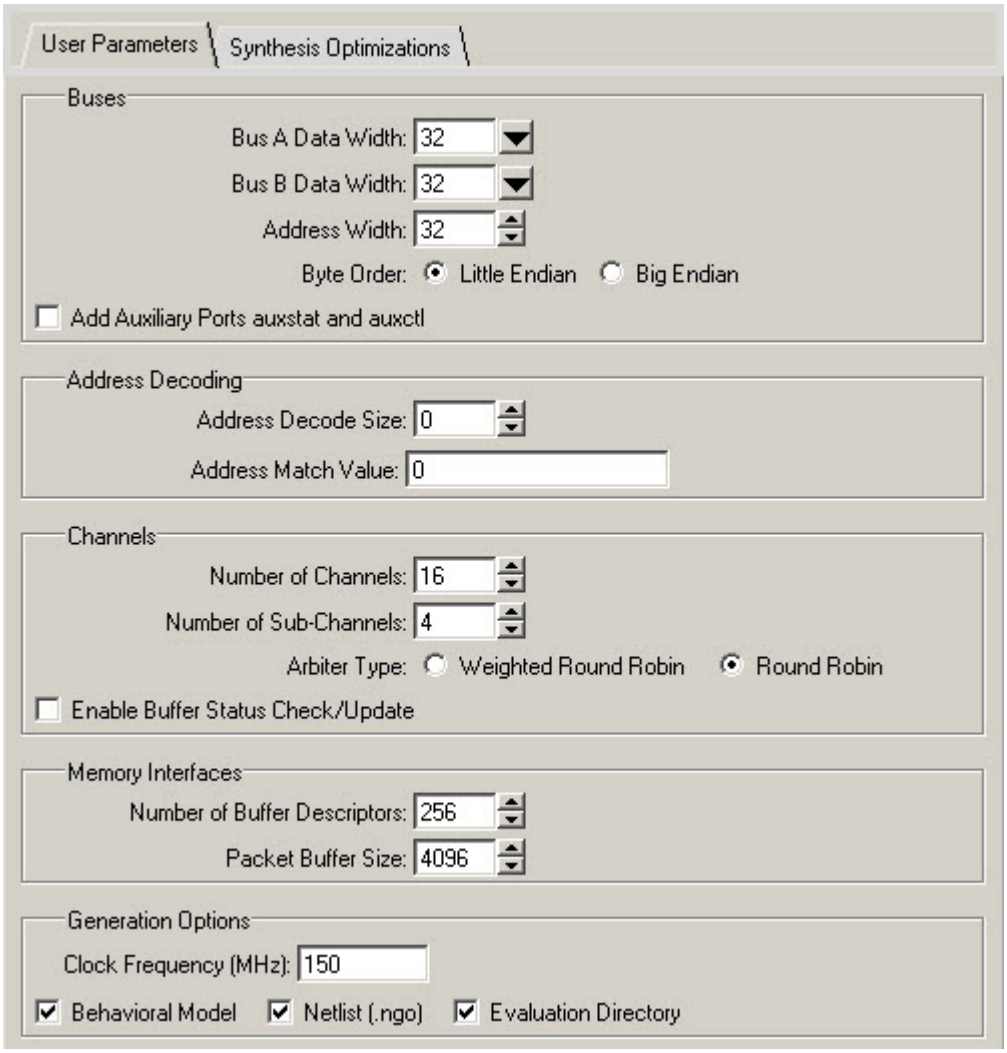
Table 3-1. SGDMAC Parameters (Continued)

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



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.

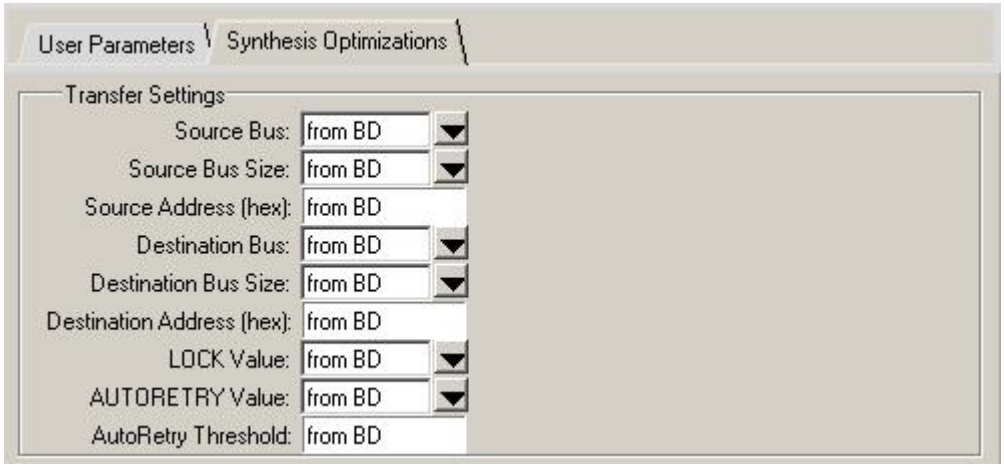
Synthesis Optimizations Tab

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



Transfer Settings

Source Bus
This option sets the source bus of the DMA transaction:

- A: Static, BusA
- B: Static, BusB
- PB: Packet Buffer
- From BD: Dynamic, refer to the BD register.

Source Bus Size
This option sets the source bus transaction data width:

- 8/16/32/64/128: Static, will be converted to log2(number_of_bytes) as 0/1/2/3/4
- From BD: Dynamic, refer to BD register

Destination Bus Size
This option sets the destination bus transaction data width:

- 8/16/32/64/128: Static, will be converted to log2(number_of_bytes) as 0/1/2/3/4
- From BD: Dynamic, refer to BD register

Source Address
This option sets the source starting address.

Destination Bus
This option sets the destination bus of the DMA transaction:

- A: Static, BusA
- B: Static, BusB
- PB: Static, Packet Buffer
- From BD: Dynamic, refer to the BD register.

Destination Bus Size
This option sets the destination bus transaction data width:

- 8/16/32/64/128: Static, will be converted to log2(number_of_bytes) as 0/1/2/3/4
- From BD: Dynamic, refer to BD register

Destination Address
This option sets the destination starting address.

LOCK Value
This option sets bus locking:

- 0: Static, Always off;
- 1: Static, Always on;
- From BD: Dynamic, refer to BD register

AUTORETRY Value
This option sets the channel to do autonomous retry:

- 0: Static, Always off;
- 1: Static, Always on;
- From BD: Dynamic, refer to BD register

AutoRetry Threshold
This option sets the number of autonomous retries.

IP Core Generation in IPexpress

This chapter provides information on how to generate the Lattice SGDMAC IP core using the Diamond software IPexpress tool, and how to include the core in a top-level design.

Licensing the IP Core

An IP core- and device-specific license is required to enable full, unrestricted use of the SGDMAC IP core in a complete, top-level design. Instructions on how to obtain licenses for Lattice IP cores are given at:

http://www.latticesemi.com/products/intellectualproperty/aboutip/isplevercoreonlinepurchas.cfm

Users may download and generate the SGDMAC IP core and fully evaluate the core through functional simulation and implementation (synthesis, map, place and route) without an IP license. The SGDMAC IP core also supports Lattice's IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited time (approximately four hours) without requiring an IP license. See the Hardware Evaluation section for further details. However, a license is required to enable timing simulation, to open the design in the Diamond EPIC tool, and to generate bitstreams that do not include the hardware evaluation timeout limitation.

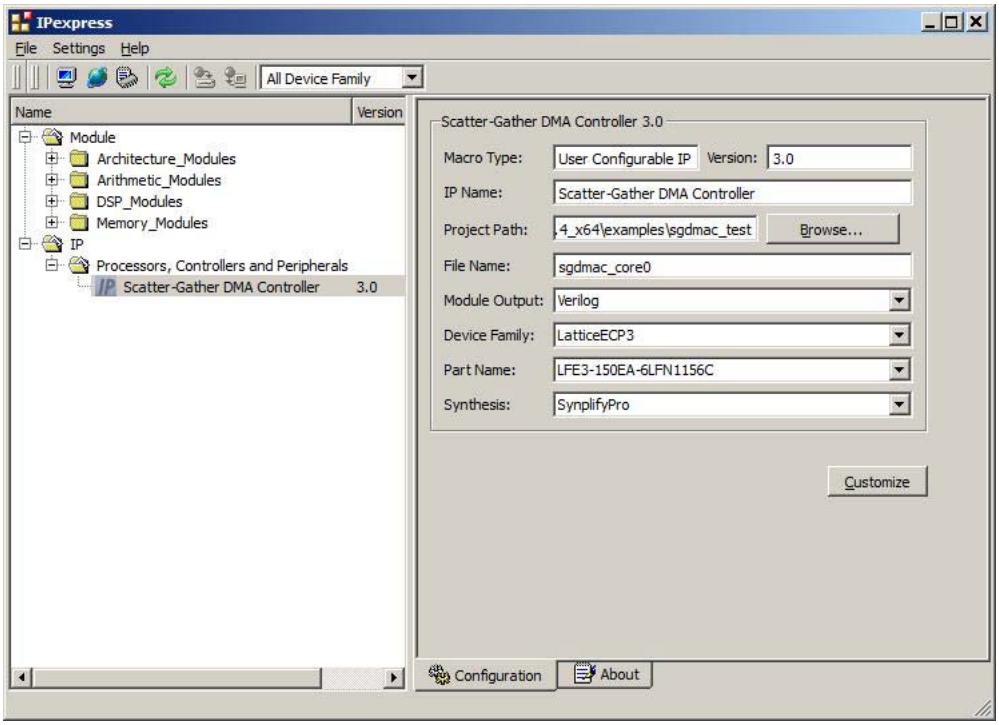
Getting Started

The SGDMAC IP core is available for download from the Lattice IP Server using the IPexpress tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the IPexpress GUI dialog box shown in Figure 4-1.

The IPexpress tool GUI dialog box for the SGDMAC IP core is shown in Figure 4-1. To generate a specific IP core configuration the user specifies:

- **Project Path** – Path to the directory where the generated IP files will be located.
- **File Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **(Diamond) Module Output** – Verilog or VHDL.
- **Device Family** – Device family to which IP is to be targeted (e.g. LatticeXP2, LatticeECP3, etc.). Only families that support the particular IP core are listed.
- **Part Name** – Specific targeted part within the selected device family..

Figure 4-1. IPexpress Dialog Box (Diamond Version)

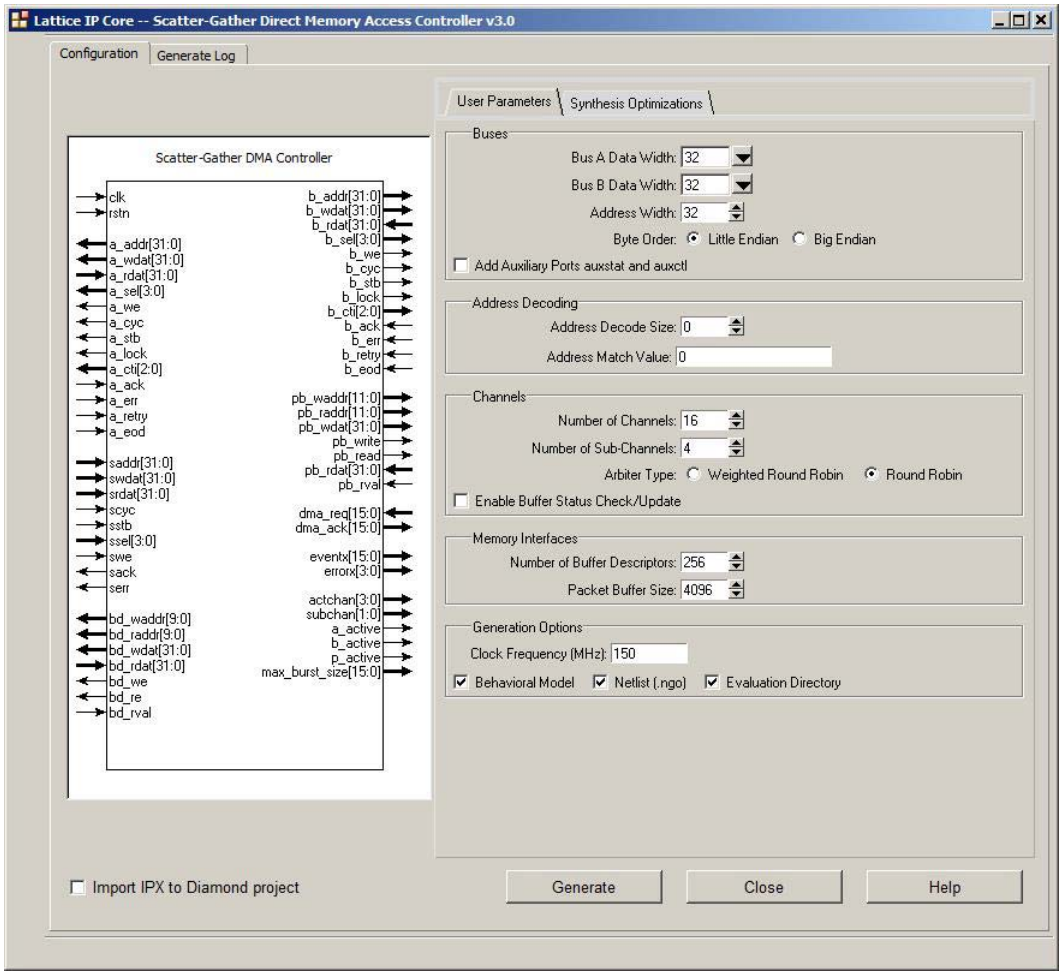


Note that if the IPexpress tool is called from within an existing project, Project Path, Module Output, Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration:

1. Click the **Customize** button in the IPexpress tool dialog box to display the SGDMAC IP core Configuration GUI, as shown in Figure 4-2.
2. Select the IP parameter options specific to your application. Refer to the Parameter Settings section for more information on the SGDMAC IP core parameter settings.

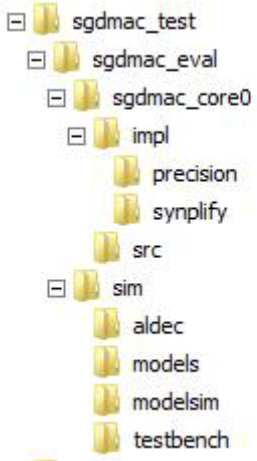
Figure 4-2. Configuration GUI (Diamond Version)



IPexpress-Created Files and Top Level Directory Structure

When the user clicks the **Generate** button in the IP Configuration dialog box, the IP core and supporting files are generated in the specified “Project Path” directory. The directory structure of the generated files is shown in Figure 4-3.

Figure 4-3. LatticeECP3 SGDMAC IP Core Directory Structure



The design flow for IP created with the IPexpress tool uses a post-synthesized module (NGO) for synthesis and a protected model for simulation. The post-synthesized module is customized and created during the IPexpress tool generation.

Table 4-1 provides a list of key files and directories created by the IPexpress tool and how they are used. The IPexpress tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the IPexpress tool.

Table 4-1. File List

File	Description
<*username*>.v	This file provides the SGDMAC core wrapper.
<*username*>_core.v	This file provides the SGDMAC core for simulation.
<*username*>_beh.v	This file provides a behavioral simulation model for the SGDMAC core.
<*username*>_core_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>_core.ngo	The ngo files provide the synthesized IP core.
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*username*>_top.[v,vhd]	This file provides a module which instantiates the SGDMAC core. This file can be easily modified for the user's instance of the SGDMAC core. This file is located in the **sgdmac_eval/**<*username*>**/src** directory.
generate_core.tcl	This file is created when GUI "Generate" button is clicked. This file may be run from the command line.
<*username*>_generate.log	This is the IPexpress scripts log file.
<*username*>_gen.log	This is the IPexpress IP generation log file

Simulation Evaluation

The sgdmac_eval directory produced by the SGDMAC generator contains two simulation (Aldec and ModelSim) subdirectories containing SGDMAC configuration and testbench files. The two directories are **sim/aldec** and **sim/modelsim**. The contents of the simulation evaluation subdirectories are as follows:

- sgdmac_eval.do – Simulation execution script
- transfer_tests.v – Test sequence commands in Verilog format
- modelsim.ini, modelsim.tcl – ModelSim only - execution support files
- wave.do – Waveform setup script

To run an Aldec Active-HDL simulation:

1. Invoke Active-HDL.
2. In Active-HDL, select **Tools->Execute macro**.
3. Browse to **sgdmac_eval\sim\aldec.**
4. Select sgdmac_eval.do.

To run a ModelSim simulation:

1. Invoke ModelSim.
2. In ModelSim, select **File -> Change Directory**.
3. Browse to **sgdmac_eval\sim\modelsim**.
4. Select **Tools -> TCL -> Execute Macro**.
5. Select sgdmac_eval.do.

Implementation Evaluation

The SGDMAC IPexpress Generator optionally creates an implementation evaluation directory **<*project_directory*>/sgdmac_eval/module_name/impl**. The module_name_eval file is a Diamond project file that will synthesize, map-place-and-route, and run timing analysis on the specified configuration of the SGDMAC core with dummy WISHBONE terminations to minimize I/O timing effects (in most applications the SGDMAC core will not interface directly with device I/O). The purpose of this evaluation directory is to provide users an estimate of area and maximum clock frequency for the specified configuration. Actual results in a user application will vary based primarily on routing constraints.

SGDMAC Core Implementation

Functional Simulation

The SGDMAC core generator produces a cycle-accurate Verilog behavioral model of the user-specified configuration of the SGDMAC core, module_name_beh.v. The generated module_name_inst.v file provides a template for instantiating the module in a Verilog netlist. The port connection list uses the port name as the default for the connection name. The size of the connection is provided. Users should edit these to connect the wires in their netlists. Only the ports that are valid for this specification configuration are included in the port list (all possible ports are listed in the example). An example of the module_name_inst.v file is shown below:

```
//============================================================================
// Verilog module generated by IPExpress    05/22/2007    10:14:24
// Filename: sgdmac_0_inst.v
// Copyright(c) 2005 Lattice Semiconductor Corporation. All rights reserved.
//============================================================================

//-----------------------------------------------------------------
// sgdmac_0 instance template
//-----------------------------------------------------------------

sgdmac_0 sgdmac_0_inst (
    .clk(clk),
    .rstn(rstn),
    .a_addr(sga_addr[31:0]),
    .a_wdat(sga_wdat[31:0]),
    .a_rdat(sga_rdat[31:0]),
    .a_sel(sga_sel[3:0]),
    .a_we(sga_we),
    .a_cyc(sga_cyc),
    .a_stb(sga_stb),
    .a_lock(sga_lock),
    .a_cti(sga_cti[2:0]),
    .a_ack(sga_ack),s
    .a_err(sga_err),
    .a_retry(sga_retry),
    .a_eod(sga_eod),
    .b_addr(sgb_addr[31:0]),
    .b_wdat(sgb_wdat[31:0]),
    .b_rdat(sgb_rdat[31:0]),
    .b_sel(sgb_sel[3:0]),
```

```
    .b_we(sgb_we),
    .b_cyc(sgb_cyc),
    .b_stb(sgb_stb),
    .b_lock(sgb_lock),
    .b_cti(sgb_cti[2:0]),
    .b_ack(sgb_ack),
    .b_err(sgb_err),
    .b_retry(sgb_retry),
    .b_eod(sgb_eod),
    .saddr(sgs_addr[31:0]),
    .swdat(sgs_wdat[31:0]),
    .srdat(sgs_rdat[31:0]),
    .scyc(sgs_cyc),
    .sstb(sgs_stb),
    .ssel(sgs_sel[3:0]),
    .swe(sgs_we),
    .sack(sgs_ack),
    .serr(sgs_err),
    .bd_waddr(bd_waddr[9:0]),
    .bd_raddr(bd_raddr[9:0]),
    .bd_wdat(bd_wdat[31:0]),
    .bd_rdat(bd_rdat[31:0]),
    .bd_we(bd_we),
    .bd_re(bd_re),
    .bd_rval(bd_rval),
    .bd_err(bd_err),
    .pb_waddr(pb_waddr[11:0]),
    .pb_raddr(pb_raddr[11:0]),
    .pb_wdat(pb_wdat[31:0]),
    .pb_rdat(pb_rdat[31:0]),
    .pb_write(pb_write),
    .pb_read(pb_read),
    .pb_rval(pb_rval),
    .dma_req(dma_req[15:0]),
    .dma_ack(dma_ack[15:0]),
    .eventx(eventx[15:0]),
    .errorx(errorx[15:0]),
    .actchan(actchan[3:0]),
    .subchan(subchan[2:0]),
    .auxctl(auxctl[15:0]),
    .auxstat(auxstat[15:0]
);
```

IP Core Implementation

The generated SGDMAC IP core is in a Lattice proprietary .ngo format, which is independent of the HDL used to capture the rest of the user's design. Instance templates and component (black box) definitions are generated in both Verilog and VHDL. The following steps are used in the implementation phase of the design process:

- Copy and paste the instance template (module_name_inst.v[hd]) into the user netlist where the SGDMAC IP core resides (not limited to user's top level).
- Edit the connection list as necessary to connect the SGDMAC IP core to the rest of the user design. *(Tip: If the wire names in the user design match the port names of the SGDMAC IP core, no editing is required. The file contents could be included in the design as is.)*
- Run the synthesis tool, making sure the blackbox definition file (module_name_bb.v or module_name_pkg.vhd) is included in the list of files to be compiled. The resulting netlist will contain a blackbox instantiation of the SGDMAC IP core.

- When running map, place, and route, make sure the module_name.ngo and the pmi_*.ngo files created by the generation process can be found by the Diamond software. This is accomplished either by copying the .ngo file(s) to the place-and-route working directory, or pointing to the directory where it resides (-p option if running ngdbuild from a script). In Diamond, set the directory where the module_name.ngo file resides by using one of the following methods:
  - In Diamond, set the **Macro Search Path** property under **Project > Active Strategy > Translate Design Settings**.
- If multiple identical IP cores are required in a single design, simply instantiate the same module multiple times with different connections. If multiple dissimilar IP cores are required, generate a different IP core for each type required using different module names, then instantiate each module type as required.

Things to consider:

No timing model exists for the SGDMAC IP core, so the synthesis tool will be unable to analyze the paths in and out of the core. The SGDMAC IP core generator provides registers on all its outputs and assumes that its inputs are driven by registers clocked with the SGDMAC IP core clock (clk).

The SGDMAC IP core netlist is not “lint-free” for all configurations, so warnings may be generated by the synthesis and map-place-route tools.

Hardware Evaluation

The SGDMAC IP core supports Lattice’s IP hardware evaluation capability, which makes it possible to create versions of the IP core that operate in hardware for a limited period of time (approximately four hours) without requiring the purchase of an IP license. It may also be used to evaluate the core in hardware in user-defined designs.

Enabling Hardware Evaluation in Diamond

Choose **Project > Active Strategy > Translate Design Settings**. The hardware evaluation capability may be enabled/disabled in the Strategy dialog box. It is enabled by default.

Updating/Regenerating the IP Core

By regenerating an IP core with the IPexpress tool, you can modify any of its settings including device type, design entry method, and any of the options specific to the IP core. Regenerating can be done to modify an existing IP core or to create a new but similar one.

Regenerating an IP Core in Diamond

To regenerate an IP core in Diamond:

1. In IPexpress, click the **Regenerate** button.
2. In the Regenerate view of IPexpress, choose the IPX source file of the module or IP you wish to regenerate.
3. IPexpress shows the current settings for the module or IP in the Source box. Make your new settings in the T**arget** box.
4. If you want to generate a new set of files in a new location, set the new location in the **IPX Target File** box. The base of the file name will be the base of all the new file names. The IPX Target File must end with an .ipx extension.
5. Click **Regenerate.** The module’s dialog box opens showing the current option settings.
6. In the dialog box, choose the desired options. To get information about the options, click **Help**. Also, check the About tab in IPexpress for links to technical notes and user guides. IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.

7. To import the module into your project, if it's not already there, select **Import IPX to Diamond Project** (not available in stand-alone mode).
8. Click **Generate**.
9. Check the Generate Log tab to check for warnings and error messages.
10. Click **Close**.

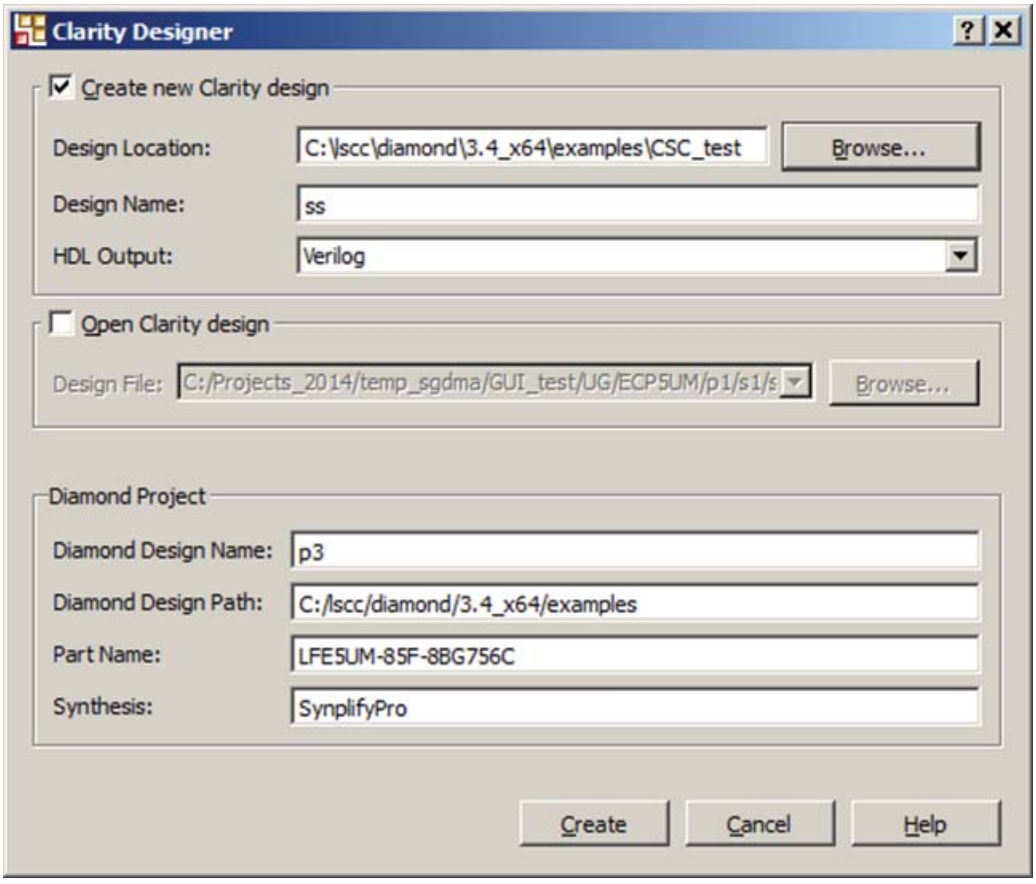
The IPexpress package file (.ipx) supported by Diamond holds references to all of the elements of the generated IP core required to support simulation, synthesis and implementation. The IP core may be included in a user's design by importing the .ipx file to the associated Diamond project. To change the option settings of a module or IP that is already in a design project, double-click the module's .ipx file in the File List view. This opens IPexpress and the module's dialog box showing the current option settings. Then go to step 6 above.

IP Core Generation in Clarity Designer

Getting Started

The first step in generating an IP Core in Clarity Designer is to start a project in Diamond software with the ECP5 device. Clicking the Clarity Designer button opens the Clarity Designer tool.

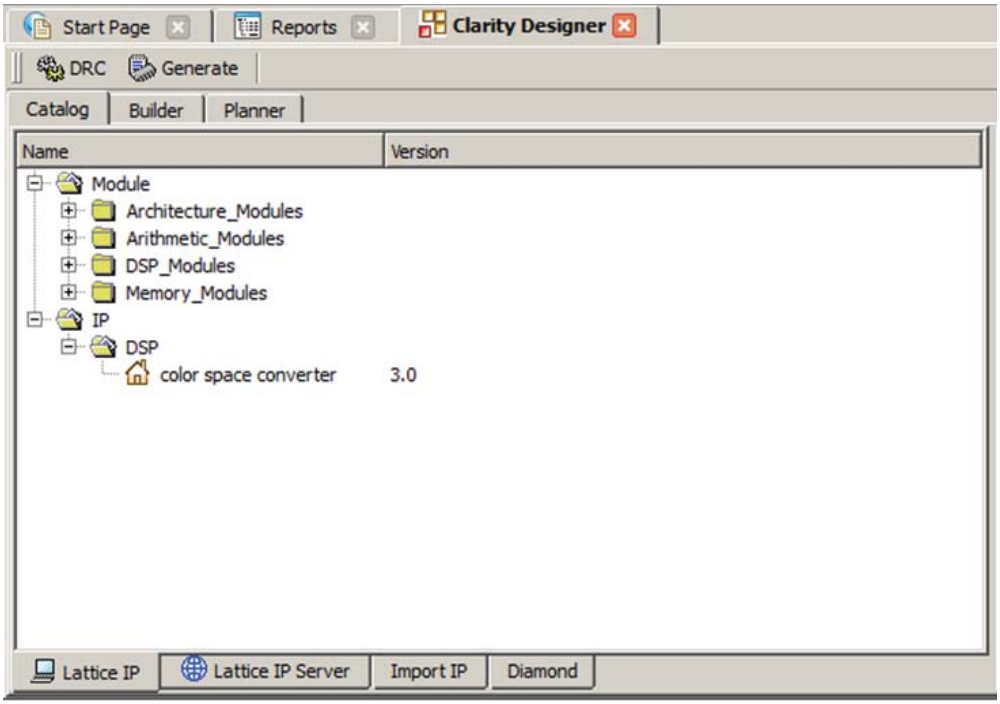
Figure 4-4. Starting a Project in Clarity Designer



As shown in Figure 4-4, you can create a new design or open an existed one. Specify the Design Location, Design Name and HDL Output format. Click **Create** to open the Clarity Designer main GUI window.

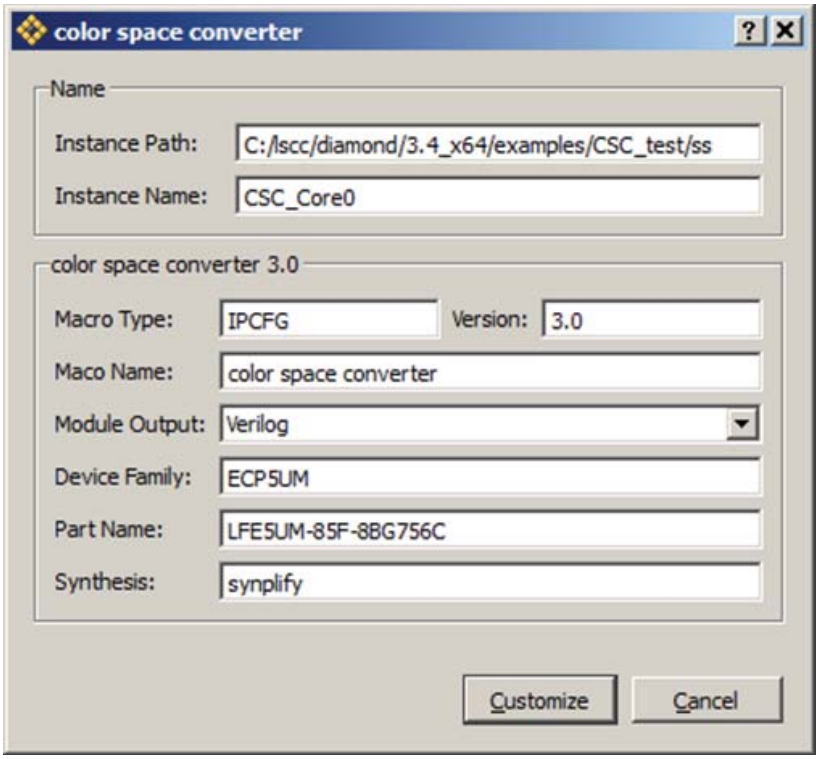
The SGDMAC IP core is available for download from the Lattice IP Server using the Clarity Designer tool. The IP files are automatically installed using ispUPDATE technology in any customer-specified directory. After the IP core has been installed, the IP core will be available in the Clarity Designer GUI Catalog window as shown in Figure 4-5.

Figure 4-5. Clarity Designer Catalog Window



Double-click the IP name to open a dialog box where you can choose configuration options, as shown in Figure 4-6.

Figure 4-6. Clarity Designer Dialog Box



To generate a specific IP core configuration the user specifies:

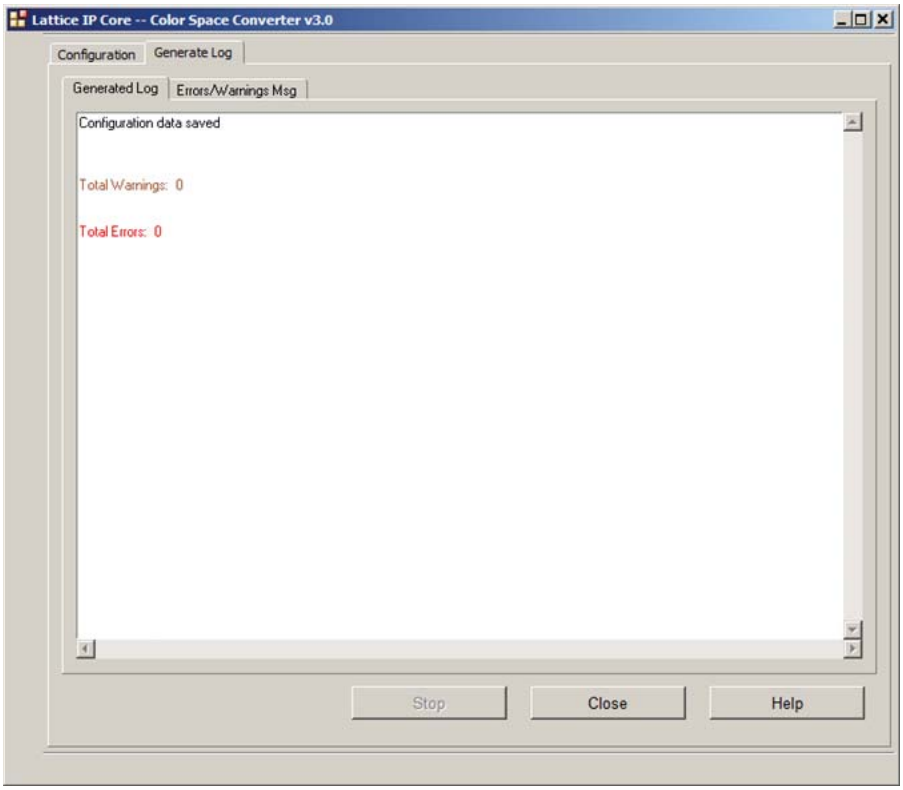
- **Instance Path** – Path to the directory where the generated IP files will be located.
- **Instance Name** – "username" designation given to the generated IP core and corresponding folders and files.
- **Module Output** – Verilog or VHDL.
- **Module Output** – Verilog HDL or VHDL.
- **Device Family** – Device family to which IP is to be targeted.
- **Part Name** – Specific targeted part within the selected device family.

Note that because the Clarity Designer tool must be called from within an existing project path, Module Output, Device Family and Part Name default to the specified project parameters. Refer to the IPexpress tool online help for further information.

To create a custom configuration:

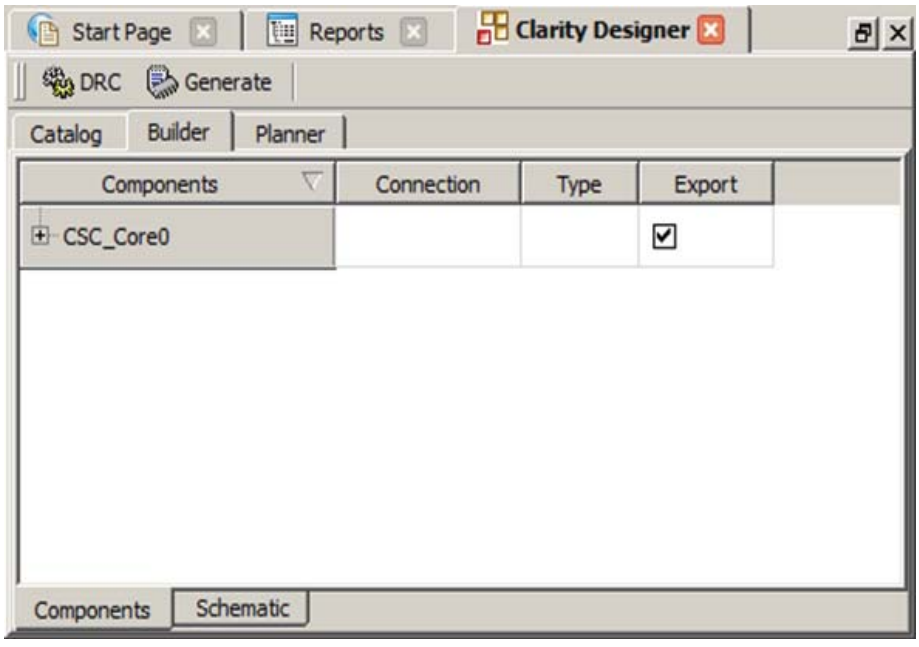
1. Click the **Customize** button in the Clarity Designer dialog box to display the SGDMAC IP core Configuration GUI, as shown in Figure 4-2.
2. Select the IP parameter options specific to your application. Refer to the Parameter Settings section for more information on the SGDMAC IP core parameter settings.
3. After setting the parameters, click **Configure**.
4. A dialog box, shown in Figure 4-7, displays logs, errors and warnings. Click **Close**.

Figure 4-7. Clarity Designer Generate Log Tab



5. The Clarity Designer Builder tab, shown in Figure 4-8, opens.

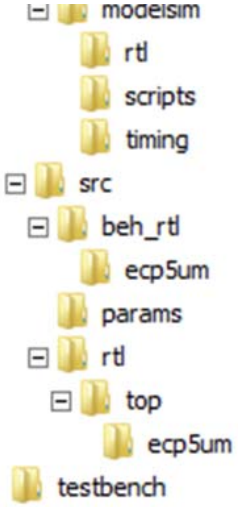
Figure 4-8. Clarity Designer Builder Tab



Clarity Designer Created Files and Top Level Directory Structure

The directory structure of the generated files is shown in Figure 4-9.

Figure 4-9. ECP5 SGDMAC IP Core Directory Structure



The design flow for IP created with the Clarity Designer tool uses post-synthesized modules (NGO) for synthesis and a protected model for simulation. The post-synthesized module are customized and created during the Clarity Designer tool generation.

Table 4-2 provides a list of key files and directories created by the Clarity Designer tool and how they are used. The Clarity Designer tool creates several files that are used throughout the design cycle. The names of most of the created files are customized to the user's module name specified in the Clarity Designer tool.

Table 4-2. File List

File	Description
<*username*>.v	This file provides the SGDMAC core wrapper.
<*username*>_core.v	This file provides the SGDMAC core for simulation.
<*username*>_beh.v	This file provides a behavioral simulation model for the SGDMAC core.
<*username*>_core_bb.v	This file provides the synthesis black box for the user's synthesis.
<*username*>_core.ngo	The ngo files provide the synthesized IP core.
<*username*>.lpc	This file contains the IPexpress tool options used to recreate or modify the core in the IPexpress tool.
<*username*>.ipx	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IPexpress tool (Diamond version only). The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP/Module generation GUI when an IP/Module is being re-generated.
<*username*>_top.[v,vhd]	This file provides a module which instantiates the SGDMAC core. This file can be easily modified for the user's instance of the SGDMAC core. This file is located in the **sgdmac_eval/**<*username*>**/src** directory.
generate_core.tcl	This file is created when GUI "Generate" button is pushed. This file may be run from command line.
<*username*>_generate.log	This is the IPexpress scripts log file.
<*username*>_gen.log	This is the IPexpress IP generation log file

Simulation Evaluation

Please refer to the Simulation Evaluation section for details.

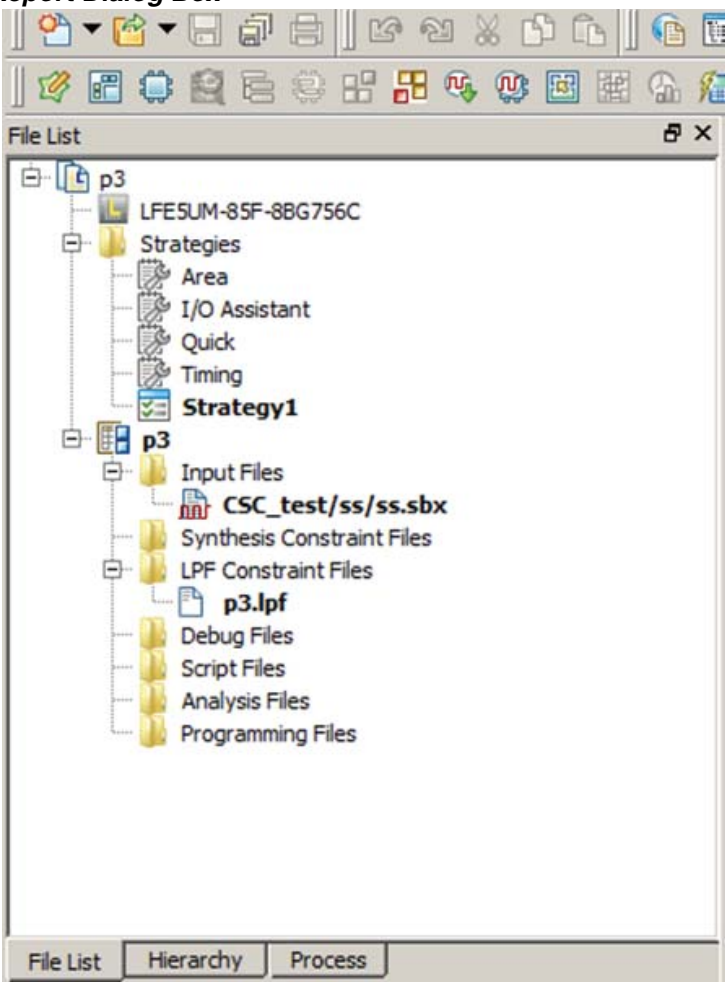
IP Core Implementation

After completing the Configuration step, click the **Generate** button, shown in Figure 4-8, to generate the Clarity Designer file (.sbx).

Clarity Designer (.sbx) files can be used in design projects such as an HDL file or an IPexpress generated (.ipx) file. A key difference between IPexpress generated files and Clarity Designer generated files is that the latter may contain not only a single block but multiple modules or IP blocks and may represent a subsystem. In IPexpress, the process generates a single module or IP. This is a one step process since an IPexpress file can only contain one module or IP. In Clarity Designer, saving a file is a separate step. Modules or IP are configured and multiple modules or IP can optionally be added within the same file. Additionally, since building and planning can also be done, saving the file and generating the blocks may be performed later.

After the Generate step is completed, the ".sbx" file is automatically added to current Diamond Project Input Files list as shown in Figure 4-10.

Figure 4-10. File List in Report Dialog Box



After this step, click **Process** at the bottom of window, then double-click **Place & Route Design** to Start PAR. This is similar to a standard Diamond PAR flow.

Regenerating/Recreating the IP Core

By *regenerating* an IP core with the Clarity Designer tool, you can modify any of the options specific to an existing IP instance. By *recreating* an IP core with Clarity Designer tool, you can create (and modify if needed) a new IP instance with an existing LPC/IPX configuration file.

Regenerating an IP Core in Clarity Designer Tool

To regenerate an IP core in Clarity Designer:

1. In the Clarity Designer Builder tab, right-click on the existing IP instance and choose **Config**.
2. In the module dialog box, choose the desired options.

   For more information about the options, click **Help**. You may also click the **About** tab in the Clarity Designer window for links to technical notes and user guides. The IP may come with additional information. As the options change, the schematic diagram of the module changes to show the I/O and the device resources the module will need.
3. Click **Configure**.

Recreating an IP Core in Clarity Designer Tool

To recreate an IP core in Clarity Designer:

1. In Clarity Designer click the Catalog tab.
2. Click the **Import IP** tab (at the bottom of the view).
3. Click **Browse**.
4. In the Open IPX File dialog box, browse to the .ipx or .lpc file of the module. Use the .ipx if it is available.
5. Click **Open**.
6. Type in a name for Target Instance. Note that this instance name should not be the same as any of the existing IP instances in the current Clarity Designer project.
7. Click **Import**. The module's dialog box opens.
8. In the dialog box, choose desired options.

   For more information about the options, click **Help**. You may also check the **About** tab in the Clarity Designer window for links to technical notes and user guides. The IP may come with additional information.

   As the options change, the schematic diagram of the module changes to show the ports and the device resources the module needs.
9. Click **Configure**.

Support Resources

This chapter contains information about Lattice Technical Support, additional references, and document revision history.

Lattice Technical Support

There are a number of ways to receive technical support.

E-mail Support

techsupport@latticesemi.com

Local Support

Contact your nearest Lattice sales office.

Internet

www.latticesemi.com

References

LatticeXP2

- HB1004, LatticeXP2 Family Handbook

LatticeECP3

- HB1009, LatticeECP3 Family Handbook

ECP5

- HB1012, ECP5 Family Handbook

Revision History

Date	Document Version	IP Version	Change Summary
March 2015	1.8	3.0	Updated Quick Facts section. Revised data in Table 1-1, Scatter-Gather DMA Controller IP Core Quick Facts.
			Updated IP Core Generation in Clarity Designer section. Changed “Customize” to “Configure” in the procedure for creating a custom configuration.
			Updated LatticeECP3 FPGAs, LatticeXP2 FPGAs, ECP5 LFE5U FPGAs and ECP5 LFE5UM FPGAs sections. — Revised data in Performance and Resource Utilization tables. — In Ordering Part Number, changed “ECP5U” and “ECP5UM” to “LFE5U” and “LFE5UM”.
April 2014	01.7	3.0asr	Added support for Diamond 3.2.
			Added support for ECP5 device family.
			Removed references to LatticeECP2M™ and LatticeSC™ device families.
			Updated corporate logo.
			Updated Technical Support information.
October 2010	01.6	2.5	Added support for Diamond software throughout.
July 2010	01.5	2.4	Divided document into chapters. Added table of contents.
			Added Quick Facts table in Chapter 1, “Introduction.”
			Added new content in Chapter 4, “IP Core Generation.”
November 2009	01.4	2.4	Updated for ispLEVER 8.0.
July 2009	01.3	2.1	Updated for SGDMAC core version 2.1, including revisions for LatticeECP3 support, addition of Buffer Status feature, Next BD feature, support for 64K buffer descriptors, and new optimization parameters.
December 2008	01.2	2.0	Updated IPexpress graphics and descriptions of GUI options.
			Added information to simulate with Aldec Active-HDL.
			Added information for licensing and purchase.
July 2007	01.1	1.1	Updated LatticeSC/M appendix and Parameter Settings for Standard Configurations table.
			Added support for LatticeECP2, LatticeECP2M/S and LatticeXP2 FPGA families.
June 2007	01.0	1.0	Initial release.

Resource Utilization

This appendix gives resource utilization information for Lattice FPGAs using the SGDMAC IP Core.

IPexpress is the Lattice IP configuration utility, and is included as a standard feature of the Diamond design tools. Details regarding the usage of IPexpress can be found in the IPexpress and Diamond help system. For more information on the Diamond or ispLEVER design tools, visit the Lattice web site at: www.latticesemi.com/software.

LatticeECP3 FPGAs

LatticeECP3 utilization data is shown in Table A-1. Table 1-1 lists the parameter settings used in deriving the utilization data shown from Table A-1 to Table A-4.

Table A-1. Performance and Resource Utilization[1]

Core Configuration	Device	Slices	LUTs	Registers	f_{MAX} (MHz)
Config1	LFE3-95EA-7FN672C	2670	4311	1932	145

1. Performance and utilization data are generated using an LFE3-95EA-7FN672C device with Lattice Diamond 3.4 software using Synopsys Synplify Pro for Lattice J-2014.09L. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeECP3 family.

Ordering Part Number

The OPN for all configurations of the Scatter-Gather DMA targeting LatticeECP3 devices is DMA-SG-E3-U1.

LatticeXP2 FPGAs

LatticeXP2 utilization data is shown in Table A-2.

Table A-2. Performance and Resource Utilization[1]

Core Configuration	Device	Slices	LUTs	Registers	f_{MAX} (MHz)
Config2	LFXP2-40E-6F672C	2139	3443	1355	120

1. Performance and utilization data are generated using an LFXP2-40E-6F672C device with Lattice Diamond 3.4 software using Synopsys Synplify Pro for Lattice J-2014.09L. Performance may vary when using a different software version or targeting a different device density or speed grade within the LatticeXP2 family.

Ordering Part Number

The OPN for all configurations of the Scatter-Gather DMA targeting LatticeXP2 devices is DMA-SG-X2-U1.

ECP5 LFE5U FPGAs

LFE5U utilization data is shown in Table A-3.

Table A-3. Performance and Resource Utilization[1]

Core Configuration	Device	Slices	LUTs	Registers	f_{MAX} (MHz)
Config3	LFE5U-85F-8BG756C	2570	4049	1637	160

1. Performance and utilization data are generated using anLFE5U-85F-8MG756C device with Lattice Diamond 3.4 software using Synopsys Synplify Pro for Lattice J-2014.09L. Performance may vary when using a different software version or targeting a different device density or speed grade within the ECP device family.

Ordering Part Number

The OPN for all configurations of the Scatter-Gather DMA targeting LFE5U devices is DMA-SG-E5-U/DMA-SG-E5-UT.

ECP5 LFE5UM FPGAs

LFE5UM utilization data is shown in Table A-4.

Table A-4. Performance and Resource Utilization[1]

Core Configuration	Device	Slices	LUTs	Registers	f_{MAX} (MHz)
Config4	LFE5UM-85F-8BG756C	1998	3222	1265	165

1. Performance and utilization data are generated using an LFE5UM-85F-8BG756C device with Lattice Diamond 3.4 software using Synopsys Synplify Pro for Lattice J-2014.09L. Performance may vary when using a different software version or targeting a different device density or speed grade within the ECP5 device family.

Ordering Part Number

The OPN for all configurations of the Scatter-Gather DMA targeting LFE5UM devices is DMA-SG-E5-U/DMA-SG-E5-UT.