Lancero:
PCI ExpressTarget Bridge and Scatter-Gather DMA Engine for Linux Systems and Altera FPGA Devices with PCIe

User Manual V1.11

9-1510 Woodcock St.
London, ON Canada N5H 5S1
www.microtronix.com

Document Information

This user guide provides basic information about using the Microtronix **Lancero: PCI Express Scatter-Gather DAM Engine** (PN: 6274-XX-XX) for Linux systems. A detailed revision history of this document is provided in section 12.

Date	Description
April 2010	Initial Release – Version 1.0
November 2011	Version 1.11

How to Contact Microtronix

E-mail

Sales Information: sales@microtronix.com
Support Information: support@microtronix.com

Website

General Website: http://www.microtronix.com
FTP Upload Site: http://microtronix.leapfile.com

Phone Numbers

General: (001) 519-690-0091
Fax: (001) 519-690-0092

Typographic Conventions

Path/Filename	A path/filename
`[SOPC Builder]$ <cmd>`	A command that should be run from within the Cygwin Environment.
`Code`	Sample code.
↵	Indicates that there is no break between the current line and the next line.

1 Contents

2 Introduction

Lancero offers a feature-complete solution for data exchange over PCI Express between Linux CPU systems and Altera FPGA devices. These features are summarized in chapter 2. Short explanations of common terminology are in chapter 3.

When using Lancero the user does not need knowledge of the complexities of PCI Express nor Linux device driver details. The interfaces of Lancero are simple and standard. Chapter 4 gives an overview of Lancero and its interfaces. The Lancero IP core is explained in chapter 5 and the Lancero Linux device driver is explained in chapter 6.

Lancero scales from low resource systems to high bandwidth systems due to a modular and high-performant IP core implementation. Two default modules are available that suit most systems. Chapter 7 describes the low resource Lancero Target module. Chapter 8 describes the high-performance Lancero SGDMA module.

A reference design is included with Lancero. Chapter 9 lists the Lancero deliverables and a step-by-step guide for getting started with the reference design.

Chapter 10 describes how to simulate the Lancero IP core using a driver for the Altera bus functional model of the PCI Express root port.

Chapter 11 describes the history of changes to both Lancero and this manual.

Refer to chapter 12 for support and contact information.

3 Features

3.1 Features

- Easily connect your logic to PCI Express.
- You do not have to deal with PCI Express protocol details.
- You do not have to deal with Linux device driver details.
- Support for all Altera devices with hard IP PCI Express core(s).
- Avalon Memory Mapped bus(es) for connecting your custom logic.
- Configurable width for the SGDMA buses; 32-bit, 64-bit or 128-bit.
- PCIe x1 Gen 1 up to x8 Gen 2 scaling from low-cost to high-performance.
- Avalon burst mode support for SGDMA buses.
- 32-bit Avalon Memory Mapped bus for control applications.
- Simply use `open`, `read` and `write` functions in Linux to perform SGDMA. No complex custom, proprietary or `ioctl`() interfaces.
- Driver supports `mmap` for direct, memory mapped access from your application to the target bus peripheral to remove the overhead of system calls.
- Scatter Gather DMA (SGDMA) transfers directly from and to user applications. No need to allocate a physically contiguous range inside or outside the kernel. A virtual memory buffer of Linux user applications, simply allocated with `malloc`, suffices.
- Zero-copy CPU overhead for transfers. No in-kernel data buffers or copies.
- Optimized for minimal CPU overhead for descriptor management.
- Supports asynchronous and multi-threaded I/O to completely remove inter-transfer latencies.

3.2 Supported Altera Devices

- Altera Cyclone IV GX
- Altera Stratix IV GT & GX
- Altera Arria II GX
- Altera Hardcopy IV GX

3.3 Supported Linux Systems

Embedded, PC and server Linux systems. Support for all architectures. The following architectures were specificly tested:

- ARM (Marvell Kirkwood)
- PowerPC (Freescale MPC8315E, 8536E, P1020, AMCC PPC460EX)
- x86 (Intel i5, i7, Core 2 Duo, Atom).

Most (recent) host distributions are supported. The device driver can be compiled or made available as a binary against any recent (2009 or later) Linux kernel.

3.4 Supported Windows Systems

Windows 7, Vista, XP for 32-bit and 64-bit is available in an upcoming update.

4 Terminology

This manual introduces a few terms that are specific to Lancero and which are explained below. A number of terms are used from the PCI Express specifications. These are listed first.

4.1 PCI Express terminology

downstream, upstream completion, initiator, root complex, end point, BAR, TLP
Refer to PCI Express documentation.

4.2 Lancero terminology

PCIe
short for PCI Express

API
Application Programming Interface.

Avalon
An on-chip local bus specification from Altera.

Avalon Memory-Mapped (Avalon-MM)
An on-chip address/data based bus for memory-mapped masters and slaves implementing host or peripheral functions.

Avalon streaming interface (Avalon-ST)
An on-chip FPGA interface that supports an unidirectional flow of data, including multiplexed streams and packets.

SGDMA
Scatter Gather Direct Memory Access; where an end-point device autonomously initiates data transfers to/from memory on other devices. The transfer may be scattered/gathered to/from physically disjunct memory addresses.

FPGA address
An address on one of the local Avalon memory-mapped buses on the local device, such as the Avalon target bus or an SGDMA-attached Avalon data bus.

PCIe address
An address in a memory address space that is outside the FPGA, on the PCIe bus. For example, the PCIe bus address that corresponds with memory in the Root Complex is a remote address.

target bridge
A transparent bridge that translates PCIe memory accesses to the device, into corresponding accesses on the Avalon target bus

target bus
An Avalon memory mapped 32-bit bus that is the cental target bus for controlling Lancero and user logic through the PCIe bus.

data bus
An Avalon memory-mapped 64-bit bus that is a data bus for high performance SGDMA data transfers.

read engine
The logic that performs the SGDMA transfer from the PCIe addresses to the on-chip Avalon data bus. The naming 'read' is in the context of the SGDMA engine, meaning the engine present in the endpoint reads from the host memory over PCIe.

write engine
The logic that performs the SGDMA transfer to the PCIe addresses from the on-chip Avalon data bus. The naming 'write' is in the context of the SGDMA engine, meaning the engine present in the endpoint writes to the host memory over PCIe.

shared data bus
Each SGDMA engine offers one bus master interface.

The masters can share one bus or have dedicated seperate buses. The default Lancero configuration has one read engine and one write engine that have a seperate bus interface.

5 Overview

5.1 Introduction

This chapter gives a high-level overview of Lancero and how it can be used into your applications. Further chapters explain the Lancero components in more detail.

5.2 Application

The target application of Lancero is any system where a CPU and FPGA exchange data over a PCI Express (PCIe) connection. To the user, the PCIe connection is transparent when using Lancero, as shown in figure 1.

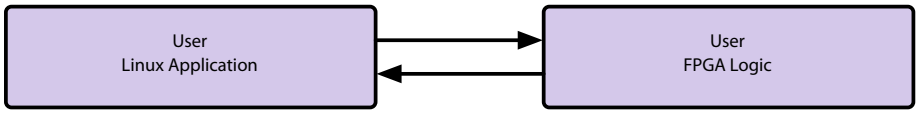


Fig. 1 Lancero implements a transparent PCI Express interconnect between user application and FPGA logic.

The developers using Lancero do not need knowledge of PCI Express nor Linux device driver details.

5.3 Lancero

Lancero provides an FPGA PCIe target bridge for setting control and reading statuses of logic blocks and small data transfers. Additionally, one or more FPGA Scatter Gather Direct Memory Access engines are provided for hardware assisted high speed data transfers for maximum performance on the PCIe bus. A corresponding Lancero Linux device driver controls the Lancero IP core transparently.

5.4 Interfaces

Your Linux application and FPGA logic are connected with each other through standard interfaces provided by Lancero. Your application opens, reads and writes to POSIX character devices. Your logic is attached as slaves on the Avalon buses.

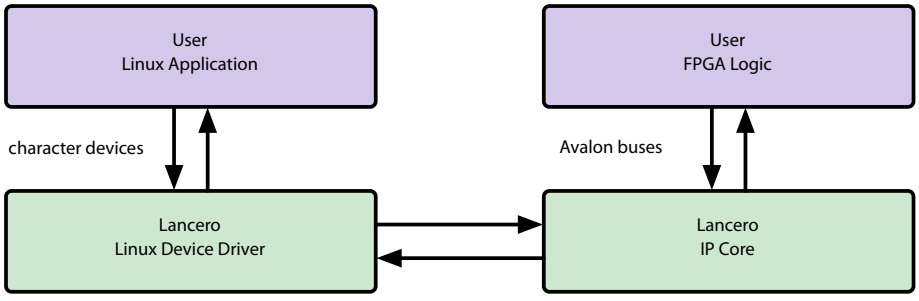


Fig. 2 Lancero uses standardized interfaces: POSIX character devices in software and Avalon memory mapped buses for the FPGA logic.

5.5 PCI Express

Lancero uses the hardware functions in your CPU and FPGA which implement the lower layers of the PCIe protocol in hardware.

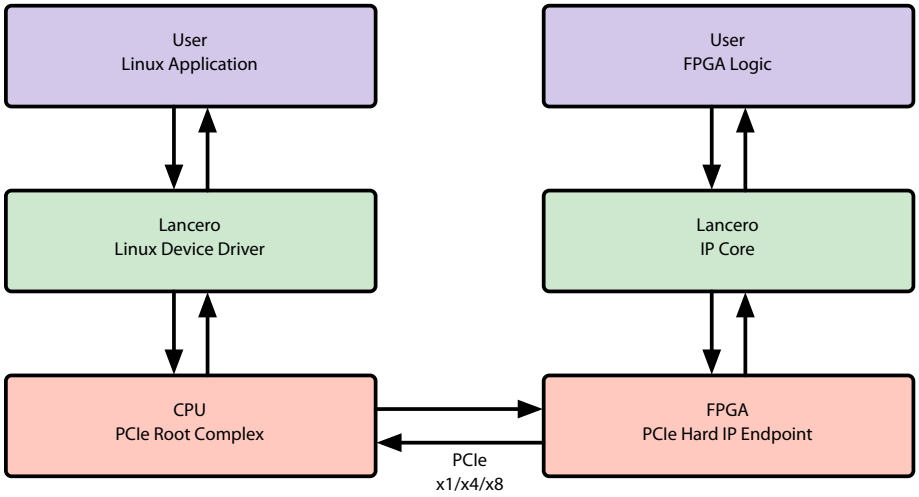


Fig. 3 The lower levels of PCIe, including SerDes and tranceiver blocks are implemented in hardware in your CPU and FPGA.

Performance of the system can be easily scaled by choosing the appropriate numbers of PCIe lanes (x1, x4 or x8), optionally using PCIe Gen 2.

5.6 Transparent Connection

The Lancero Linux device driver provides one character device interface for the target bus, and one character device interface for SGDMA.

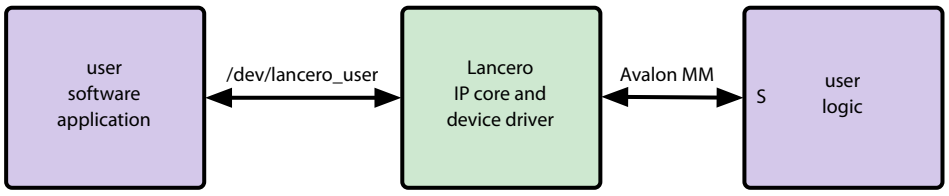


Fig. 4 The user application accesses the user logic through character devices that maps one-to-one to its corresponding Avalon bus.

Any `read`, `write` or `seek` on a character device corresponds with bus read, bus write or changing the address offset on the bus. For SGDMA buses, read and writes are performed as bursts.

The target bus is a memory mapped 32-bit Avalon bus. The data buses are 32, 64 or 128-bits memory mapped Avalon buses with bursting masters.

6 Lancero IP core

6.1 SOPC module

The Lancero IP core is delivered as a ready-to-use module for SOPC Builder. Two modules configurations are available; Target and SGDMA.

6.1.1 Target Module

The Target Module configuration is ideal for small PCIe end points which just require simple register access and where cost and resources are limited. This includes an interrupt controller.

6.1.2 SGDMA Module Read/Write

The SGDMA Module configuration additionally provides Scatter-Gather Direct Memory Access engines for applications where high data rates are required. This also includes the target bridge and interrupt controller.

6.1.3 SGDMA Module Read

For applications where the data flow is towards the FPGA endpoint only. This also includes the target bridge and interrupt controller.

6.1.4 SGDMA Module Write Only

For applications where the data flow is towards the CPU host only. This also includes the target bridge and interrupt controller.

6.2 Resources

Table 1 shows the performance and resource utilization of Lancero:

Device	Configuration	Logic	Memory (M9K)	fmax (MHz)
Cyclone IV GX (x1/x4)	Lancero Bridge	1352 LEs	11	>125
	Lancero SGDMA write	3388 LEs	18	
	Lancero SGDMA read	3927 LEs	21	
	Lancero SGDMA both	5905 LEs	28	
Arria II GX or Stratix IV GX (x1/x4)	Lancero Bridge	592 ALMs	9	>125
	Lancero SGDMA write	1491 ALMs	18	
	Lancero SGDMA read	1722 ALMs	20	
	Lancero Scatter Gather	2594 ALMs	27	
Arria II GX or Stratix IV GX (x8)	Lancero Bridge	680 ALMs	17	>250
	Lancero SGDMA write	1649 ALMs	32	
	Lancero SGDMA read	1942 ALMs	32	
	Lancero Scatter Gather	2834 ALMs	43	

Lancero supports a full speed interconnect with the PCIe IP core at 125 MHz even on the slowest speed grade Cyclone IV.

6.3 SOPC Block diagram

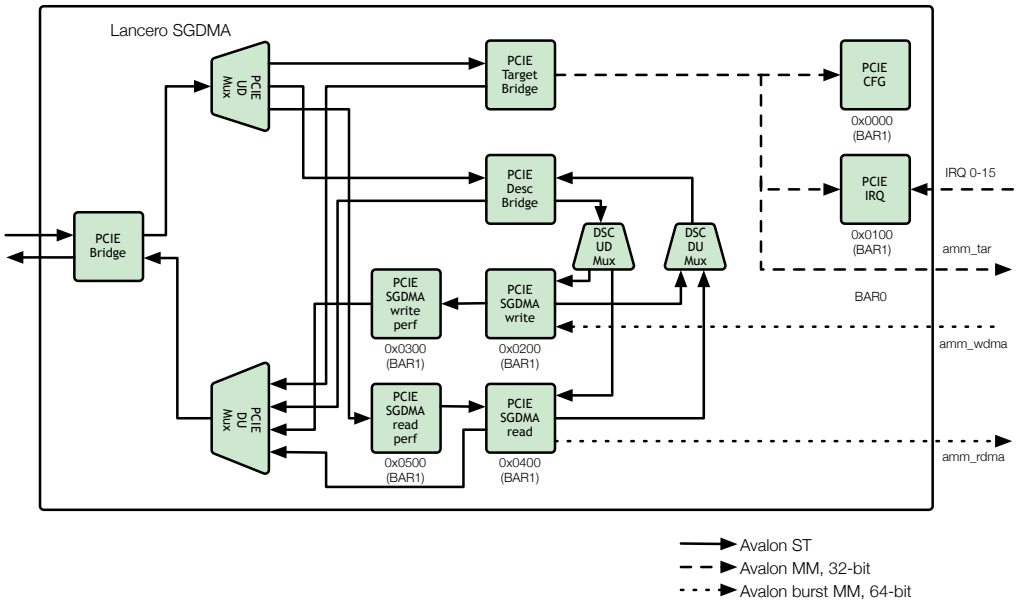


Fig. 5 Lancero block diagram showing the internal SOPC components of the Lancero Scatter-Gather configuration.

Lancero is a modular system which can be used with the system-on-programmable-chip (SOPC) Builder to construct your application logic.

6.4 PCIe Bridge

The PCIe bridge is the interface between the Lancero modules and the PCIe subsystem. It connects directly to the Altera PCI Express Hard IP core using the 64-bit wide Avalon streaming interface (Avalon ST).

6.5 Target Bridge

Single read and write requests from the PCIe host are translated to Avalon memory mapped bus transactions by the target bridge. The target bridge is master on the 32-bit memory-mapped Avalon bus. It supports byte, half-word and word transactions with natural alignment.

Additional Lancero components and/or user custom logic are connected to the target bus as Avalon slaves for purposes of control and status.

This offers full compatibility with the extending range of Avalon and SOPC building blocks.

6.6 Scatter Gather Direct Memory Access Engines

The SGDMA engines perform all the work to achieve high-speed direct memory access to the host (or other remote devices if a PCIe switch is used). The engines are slaves on the target bus. They can be instructed to perform one or more transfers by setting a few control registers. After the engine starts it works autonomously until completion.

Data is transfered between the PCIe host and a slave component on a 32, 64 or 128-bit Avalon Memory-Mapped bus connected to the engine. Each engine is an Avalon master on its bus and has burst support.

Each data bus has its own address space.

The amm_rdma and amm_wdma data buses can be interconnected to form a single bus if this suits the application.

6.6.1 Features

- Avalon 32, 64, or 128-bit memory mapped bus for data transfers.
- Burst support on the Avalon bus.
- Unlimited number of transfers in the descriptor list.
- Each transfer can be from 8 bytes up to ~2 GB (2^{31} – 8 bytes).
- New transfers can be added in-flight.
- Transfer descriptors are prefetched resulting in no idle latency.

6.7 Descriptor Lists

The transfer work performed by the SGDMA engines is described in a list of transfers called a descriptor list. Each entry in this list describes a memory content transfer between consecutive addresses in both the local and remote address space.

Descriptor lists are created by the initiator of a SGDMA transfer in the initiator memory, which is remote to the FPGA.

6.7.1 Features

- Unlimited descriptor list size.
- Read-only specification for a clear separation of work and result. No write overhead is involved during descriptor processing other than the interrupt message on completion.

6.8 Descriptor Processor

The descriptor processor assists the SGDMA engine. It is internal to Lancero and has no accessible interfaces. It is described here for completeness.

The descriptor processor fetches the descriptor list items from remote memory.

It is controlled by its associated SGDMA engine using a memory mapped Avalon interconnect where the engine is the master and the list processor is the slave.

6.8.1 Features

- Prefetches descriptors so that the SGDMA engine is never idle during processing the transfers in the list.
- Supports prefetching of up to sixteen descriptors in a single request.

6.9 Interrupt Controller

The interrupt controller signals the root complex about events by meaning of a PCIe Message Signaled Interrupt (MSI) or a Legacy Interrupt.

The interrupt controller accepts up to sixteen interrupt sources. Each of the Lancero SGDMA engines is a source. Additional custom user logic (either SOPC or external) interrupt sources can be connected to the controller.

6.9.1 Features

- Enable mask
- Event Mask
- Request mask
- Sixteen interrupt sources from user SOPC components.

7 Linux device driver

7.1 Overview

Your Linux application uses the Lancero kernel device driver, which interfaces with the target module and SGDMA engines in the FPGA logic.

The Linux software developer or user does not need any device driver programming expertise or knowledge, as the device driver fully abstracts the functionality in very clean POSIX interfaces.

Simple `open`, `read`, `write`, `close` function calls can be used to perform high-performance SGDMA transfers between user application buffers and SOPC slave components. The application buffer can be allocated using `malloc`. The FPGA bus is an Avalon Memory-Mapped bus which has slaves such as memories and peripherals connected to it.

7.2 Scather/Gather DMA to user buffers

Lancero can perform large SGDMA transfers directly to and from user application buffers, without in-kernel copies needed with regular DMA, and without the requirement to reserve a physical portion of memory out-of-kernel or in-kernel and without the requirement to perform a double-copy from kernel buffer to user space application buffer.

A large transfer from/to the virtual memory space of a Linux applications typically results in a large number of smaller scattered transfers from/to physical memory. Scattered refers to the fact that the physical memory locations of the data to be transferred are non-adjacent.

The scatter/gather approach of Lancero allows to perform such transfers with minimal overhead.

7.3 Character devices

The following diagram focusses on the Linux details of how an application accesses the hardware through character devices. These present themselves as filenames in the filesystem and are accessible as-if they were files.

The Lancero Linux device driver offers the following character device exists:

- user character device for access to user SOPC components
- control character device for controlling Lancero components
- events character device for waiting for interrupt events
- SGDMA character device for high performance transfers

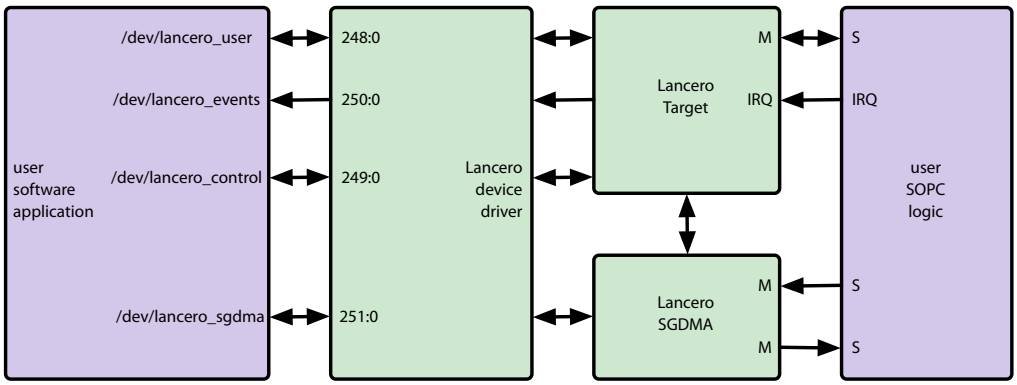


Fig. 6 The relationship between character devices, major and minor device numbers and Avalon Memory-Mapped buses and interrupts.

Each character device appears in the filesystem as a named device node; the name can be chosen by the user or a user application such as 'udev'.

In turn, the device node refers to a device major and minor number, chosen by the operating system, the device driver or the user through a device driver option.

7.4 First time setup

Make sure the FPGA is configured **before** the bootloader and/or Linux runs.

From the Linux command line shell, use the `lspci` command from the `pcitools` package to verify that your Linux system has recognized the PCIe end point. You should see the vendor and product ID you have chosen in the Altera PCIe Compiler MegaWizard. By default these are 0x1172 resp. 0x0004.

You may need to reset the system if the FPGA was configured after booting Linux and does not show up in the list of PCI devices.

7.4.1 Load the device driver

The `lancero.ko` file is the Linux device driver ('ko' is for kernel object).

```
sudo insmod ./lancero.ko
```

Use the `dmesg` command to see the output of the device driver and note the major minor numbers for the character devices if this is your first setup. For example:

```
lancero_user = 251:0
lancero_control = 250:0
lancero_events = 249:0
lancero_sgdma = 248:0
```

7.4.2 Create the filesystem device nodes

The device nodes allows your applications to access the character devices as if it were files. On a persistent storage device for the root filesystem this needs to be done once. Using the example above:

```
sudo mknod -m 0666 /dev/lancero_user c 251 0
sudo mknod -m 0666 /dev/lancero_control c 250 0
sudo mknod -m 0666 /dev/lancero_events c 249 0
sudo mknod -m 0666 /dev/lancero_sgdma c 248 0
```

248-251 are the major number and 0 is the minor number assigned to the device driver, which can be seen from the output of `dmesg` after loading the driver module (see above).

7.4.3 Static device node

In a typical system you want control over the major and minor device node numbers. This can be achieved by adding the major= parameter to the driver, with the matching creation of device nodes is (verify with the `dmesg` command):

```
sudo insmod ./lancero.ko major=130
sudo mknod -m 0666 /dev/lancero_user c 130 0
sudo mknod -m 0666 /dev/lancero_control c 130 1
sudo mknod -m 0666 /dev/lancero_events c 130 2
sudo mknod -m 0666 /dev/lancero_sgdma c 130 3
```

7.4.4 Managed dynamic device node

As an alternative, most Linux system allow managed on-demand creation of device nodes through a user daemon such as `udev`.

7.5 Interfacing with the device driver

The Lancero device driver uses a standard POSIX compliant interface using the `open`, `read`, `write`, `seek` and `close` calls, which act on the Lancero character devices.

Additionally, the target bus character devices supports `mmap`.

For advanced use, the SGDMA bus characters device support the asynchronous variants of `read`, `write`. Currently, this requires the use of the `libaio` library.

7.5.1 Function call summary

```
ssize_t read(int fildes, void *buf, size_t nbyte);
ssize_t write(int fildes, const void *buf, size_t nbyte);
off_t lseek(int fildes, off_t offset, int whence);
```

Variant function calls:

```
ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
               off_t offset);
```

```
ssize_t readv(int fildes, const struct iovec *iov, int iovcnt);
ssize_t writev(int fildes, const struct iovec *iov, int iovcnt);
```

Advanced use:

```
void *mmap(void *addr, size_t len, int prot, int flags,
       int fildes, off_t off);
```

7.6 Interfacing with the target bus

7.6.1 Read / write size

One, two and four byte (`nbytes`) read and write access is supported.

7.6.2 Alignment

All accesses must be naturally aligned. When using read and write calls the file offset (`offset`) must be a multiple of the size (`nbytes`).

When using mmap() the user application can directly access the target bus using memory access. The developer and user application are responsible for checking correct alignment.

7.6.3 Return codes

`-EPROTO` indicates misaligned access or invalid length.

7.7 Interfacing with the event bus

7.7.1 Read only

The event (virtual) bus is read-only. Only four byte (32-bit) reads are supported (`nbytes=4`). A read blocks until the interrupt events register has changed since the last read.

7.8 Interfacing with the SGDMA bus

7.8.1 Read / write size

The smallest read / write size is equal to the DMA Avalon bus data width, , which is configured in the Lancero IP core and can be found by inquiring the Lancero Configuration Inspector.

7.8.2 Alignment

All read/write accesses must be naturally aligned to the DMA Avalon bus data width, thus the file offset (`offset`) and buffer address (`buf`) and size (`nbytes`) both must be a multiple of the data bus width.

7.8.3 Return codes

`-EPROTO` indicates misaligned access or invalid length.
`-EIOCBQUEUED` indicates succesful queueing of the asynchronous request.

8 Target Module

8.1 Overview

The target module provides easy access to user SOPC component registers through single read/write operations. Additionally it provides an interrupt controller. The block diagram is shown in figure 7.

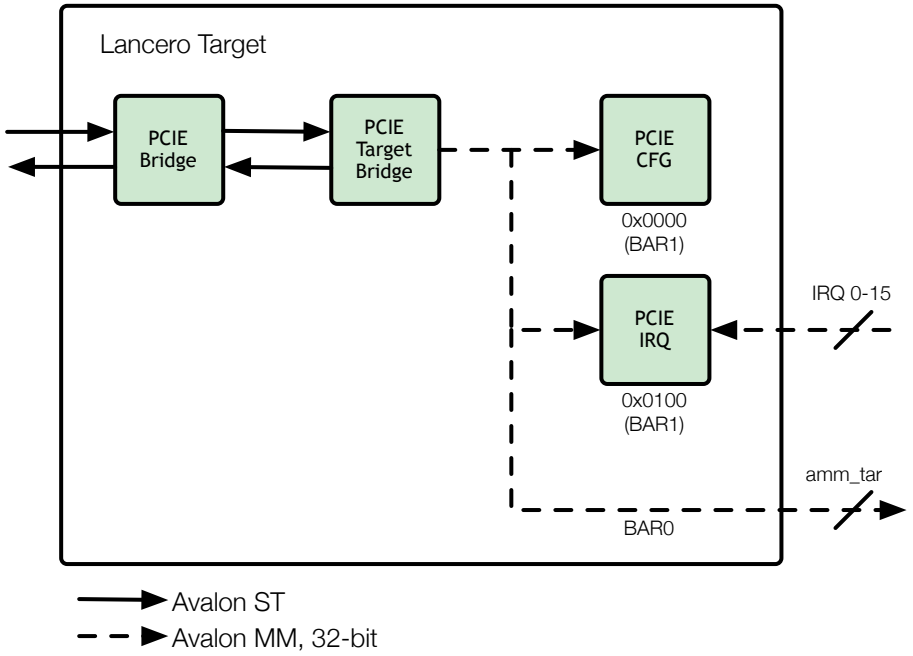


Fig. 7 The target module passes 8, 16 and 32-bit accesses from PCIe to the Avalon Memory-Mapped user target bus 'amm_tar'. It handles up to sixteen user interrupt sources.

8.2 SOPC component

In SOPC Builder, the Target Module appears as follows.

Use	Con...	Module Name	Description	Clock	Base	End
☑		⊟ **lancero_target**	Lancero PCIe Target			
		amm_tar	Avalon Memory Mapped Master	**pcie_clk**	IRQ 0	IRQ 15

Fig. 8 The Lancero Target Module as it appears in SOPC Builder

The target bridge is master on the Avalon Memory-Mapped bus. The user can connect SOPC slave components to the 'amm_tar' user target bus. Up to sixteen slave interrupt sources can be connected; IRQ 0 through 15.

8.3 PCIe Bridge

The PCIe bridge interfaces between the hard IP PCIe core and Lancero components.

8.4 Target Bridge

The target bridge transparently maps read and write accesses from the PCIe host into corresponding accesses on the local Avalon Memory-Mapped target bus.

The target bridge has two BAR regions. Lancero components are mapped in BAR1 and user SOPC components are mapped in BAR0.

8.5 BAR0 Address Map

The BAR0 address map fully depends on the user SOPC components.

8.6 User Character Device

The user character device is used to access the user SOPC components in the BAR0 address space connected on the 'amm_tar' bus.

The target control character device accepts 1, 2 or 4-byte `read` and `write` calls on 1,2 or 4-byte aligned positions respectively.

8.6.1 Application example; write to a user SOPC component register

```
uint32_t value = 0x00084321;
uint32_t register_address = 0x8000;
int fd = open("/dev/lancero_user", O_RDWR);
int rc = pwrite(fd, &value, 4, register_address);
close(fd);
```

Two variables are initialized with the value and register address to be set. In the open call, the user character device that corresponds with the target user bus on the FPGA is opened. A file descriptor is returned. You can now access this device like a file. With the `pwrite` call, a positional write of the 4-byte value 0x00084321 is performed to address 0x00008000 on the Avalon target bus.

8.6.2 Mapping the user bus in memory

Additionally, the user character device supports `mmap`, through which the target bus can be mapped into the application virtual memory area. The application can then use memory access to read and write through the target bus. This method has less system overhead than the read/write() approach. However, the user is responsible for proper accesses regarding address range, size and alignment.

8.7 BAR1 Address Map

The base addresses for the Lancero components in BAR1 are as follows.

Description	Target Bus Base Address (–End Address)
Configuration	0x000 (–0x0FF)
Interrupt Controller	0x100 (–0x1FF)

Single byte (8 bit), half-word (16 bit) and word (32 bit) memory read and write accesses are supported. These are transparently translated into similar accesses on the

target bus. The accesses must use natural alignment. For example: a 16-bit access must be on an even byte address.

8.8 Control Character Device

The control character device is used to acces the Lancero components on BAR1. The same access restrictions apply as with the user character device (see above).

8.9 Configuration Inspector

The configuration inspector is used to query the Lancero module configuration.

8.9.1 Register Map

The configuration inspector is located at base address 0x0 and has the following registers for identification and control:

Byte Offset	Name	RC access	EP access	Description
0x00	identifier	R	R	Identifier and Version
0x04	bus_dev_fn	R	W	Bus, device and function
0x08	payload	R	W	Payload
0x0C	maxread	R	W	Maximum read request (bytes)
0x10	lancero_id	R	W	Lancero System Identifier
0x14	msi	R	W	MSI
0x18	pcie_width	R	W	PCIe Hard IP Avalon ST width
0x1c	wdma width	R	W	Write DMA Avalon MM width
0x20	rdma width	R	W	Read DMA Avalon MM width

All registers support 32-bit (PCI DWORD) access only.

8.9.2 Identifier and Version Register

This read-only fixed-value register allows the device driver to verify that the configuration inspector component is available, and which version it is.

Identifier and Version Register		
31:24	23:8	7:0
Reserved	Identifier	Version

Identifier Value	Description
0xB200	Configuration Inspector Identifier
other values	Reserved

The version value indicates the configuration inspector module version. Some registers are only available since a specific version.

Version Value	Description
0x01	Configuration Inspector version 1
0x02	Configuration Inspector version 2
other values	Reserved

8.9.3 Bus Device Function register

Bit	Name	RC access	Description
31:16	Reserved	R	Reserved
15:8	bus	R	Bus number
7:3	device	R	Device number
2:0	function	RW	Function number

8.9.4 Payload register

The maximum PCI Express payload is configurable in both the PCI Express Megawizard and the Lancero SOPC component. The actual payload used is determined by the host system and it can be read by this register.

Bit	Name	RC access	Description
31:13	Reserved	R	Reserved
12:0	payload	R	PCIe TLP payload size in bytes

8.9.5 Maximum Read Request register

Bit	Name	RC access	Description
31:13	Reserved	R	Reserved
12:0	maxread	R	PCIe Maximum Read Request in bytes

8.9.6 Lancero System Identifier register

Indicates the Lancero system in use:

Bit	Name	RC access	Description
31:16	Reserved	R	Reserved
15:0	lancero_id	R	Lancero System Identifier

Lancero System Identifier Value	Description
0xFF01	Lancero Target
0xFF02	Lancero SGDMA
0xFF03	Lancero SGDMA Write Only
other values	Reserved

8.9.7 Message Signaled Interrupt (MSI) register

This register appeared in version 2 of the Lancero IP Specification.

Bit	Name	RC access	Description
31:1	Reserved	R	Reserved
0	msien	R	1 = MSI Enabled, 0= Legacy Interrupting Enabled

The use of Message Signaled Interrupts is configured using PCI Express configuration registers.

8.9.8 PCIe IP Avalon Width register

This register appeared in version 2 of the Lancero IP Specification.

Bit	Name	RC access	Description
31:2	Reserved	R	Reserved
1	pcie_128	R	128 bits PCIe Avalon width
0	pcie_64	R	64 bits PCIe Avalon width

Indicates the data bus width between Lancero and the PCIe IPs.

8.9.9 Write DMA Avalon Width register

This register appeared in version 2 of the Lancero IP Specification.

Bit	Name	RC access	Description
31:5	Reserved	R	Reserved
4	wdma_256	R	256 bits write DMA Avalon bus data width
3	wdma_128	R	128 bits write DMA Avalon bus data width
2	wdma_64	R	64 bits write DMA Avalon bus data width
1	wdma_32	R	32 bits write DMA Avalon bus data width
0	wdma_en	R	write DMA is enabled and available

Indicates the data bus width available to the user from the Lancero IP core.

8.9.10 Read DMA Avalon Width register

This register appeared in version 2 of the Lancero IP Specification.

Bit	Name	RC access	Description
31:5	Reserved	R	Reserved
4	rdma_256	R	256 bits read DMA Avalon bus data width
3	rdma_128	R	128 bits read DMA Avalon bus data width
2	rdma_64	R	64 bits read DMA Avalon bus data width
1	rdma_32	R	32 bits read DMA Avalon bus data width
0	rdma_en	R	read DMA is enabled and available

Indicates the data bus width available to the user from the Lancero IP core.

8.10 Interrupt Controller

Whenever the hardware needs to signal the software it will raise an interrupt request. Software then needs to determine the cause of the interrupt and determine which action to take and when.

The interrupt controller has 24 interrupt sources. The upper eight (23-16) are internal to Lancero. The others interrupts (15-0) are available for user SOPC com-

ponents. Whenever an interrupt occurs and is enabled in the enable register, a PCIe interrupt is sent.

8.10.1 Register Map

The interrupt controller is located at base address 0x100 and has the following registers for identification and control:

Byte Offset	Name	RC access	EP access	Description
0x00	identifier	R	R	Identifier and Version
0x04	enable	RW	R	Enable
0x08	request	R	W	Request
0x0C	pending	R	W	Event Pending

All registers support 32-bit (PCI DWORD) access only.

8.10.2 Identifier and Version Register

This read-only fixed-value register allows the device driver to verify that the interrupt controller component is available, and which version it is.

Identifier and Version Register		
31:24	23:8	7:0
Reserved	Identifier	Version

Identifier Value	Description
0xB100	Interrupt Controller Identifier
other values	Reserved

The version value indicates the hardware interface specification version.

Version Value	Description
0x01	Interrupt Controller version 1
0x02	Interrupt Controller version 2
other values	Reserved

8.10.3 Interrupt Enable Register Bits

When the interrupt enable is set and a rising edge occurs on the SOPC interrupt line, a PCIe interrupt is sent.

Bit	Name	RC access	Description
31:24	Reserved	R	Reserved
23	IE23	RW	Interrupt Enable 23, internal to Lancero
…	…	…	…
16	IE16	RW	Interrupt Enable 16, internal to Lancero
15	IE15	RW	Interrupt Enable 15 (SOPC interrupt 15)
0	IE0	RW	Interrupt Enable 0 (SOPC interrupt 0)

8.10.4 Interrupt Request Register Bits

IR0 through IR23 indicate the current interrupt source lines ANDed with the enable mask. The interrupts must be cleared by deasserting the source.

Bit	Name	RC access	Description
31:24	Reserved	R	Reserved
23	IE23	RW	Interrupt Request 23, internal to Lancero
…	…	…	…
16	IE16	RW	Interrupt Request 16, internal to Lancero
15	IE15	RW	Interrupt Request 15 (SOPC interrupt 15)
0	IE0	RW	Interrupt Enable 0 (SOPC interrupt 0)

8.10.5 Interrupt Event Pending Register Bits

EP0 through EP23 indicate the current interrupt source lines indicating pending events. The events must be cleared by removing the event cause condition in the source component.

Bit	Name	RC access	Description
31:24	Reserved	R	Reserved
23	EP23	R	Interrupt Event Pending 23, internal to Lancero
…	…	…	…
16	EP16	R	Interrupt Event Pending 16, internal to Lancero
15	EP15	R	Interrupt Event Pending 15 (SOPC interrupt 15)
0	EP0	R	Interrupt Event Pending 0 (SOPC interrupt 0)

The interrupt events pending register first appeared in version 2 of the interrupt controller.

8.11 Interrupt Event Character Device

The event character device is a convenience interface; it allows the application to perform a blocking read waiting for an interrupt event to occur, without polling the interrupt request register of the interrupt controller. The character device accepts only 4-byte `read` calls.

Whenever the character device is `read`, the read blocks until the interrupt request bit vector was changed since the last `read`.

The user application is responsible for clearing the pending interrupts by removing the interrupt cause condition in the respective Avalon slaves.

9 SGDMA Module

9.1 Overview

The SGDMA module consists of the target module components (see section 7) and independent read and write SGDMA engines described in this section.

The engines connect to user SOPC components through 32, 64 or 128-bits Avalon Memory-Mapped buses and are intended for high bandwidth data transfers.

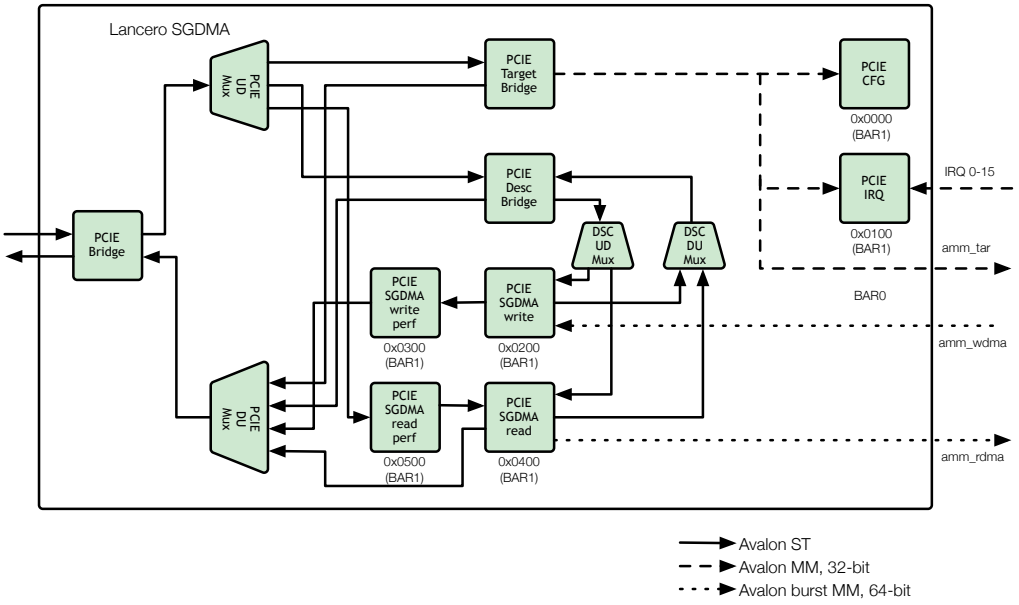


Fig. 9 Lancero block diagram showing the internal SOPC components of the Lancero SGDMA module.

9.2 SOPC component

In SOPC Builder, the SGDMA Module appears as follows.

Use	Connections	Module Name	Description
✔		⊟ **lancero_sgdma**	Lancero PCIe SGDMA
	×—	amm_tar	Avalon Memory Mapped Master
	×—	amm_wdma	Avalon Memory Mapped Master
	×—	amm_rdma	Avalon Memory Mapped Master

Fig. 10 The Lancero SGDMA Module as it appears in SOPC Builder

The target bridge is master on the 32-bit Avalon Memory-Mapped target bus called ‘amm_tar’, which is intended for control/status and small data transfers. The user can connect SOPC slave components to this bus. Up to sixteen slave interrupt sources can be connected; IRQ 0 through 15. See chapter 7 for more information.

The SOPC component can be configured by double-clicking it. The configuration GUI will appear as shown in figure 11. The PCIe Data Width, Maximum Payload and Maxium Read Request must match the PCIe Hard IP Megawizard settings.

The SGDMA (Avalon Bus) Data Widths are user configurable.

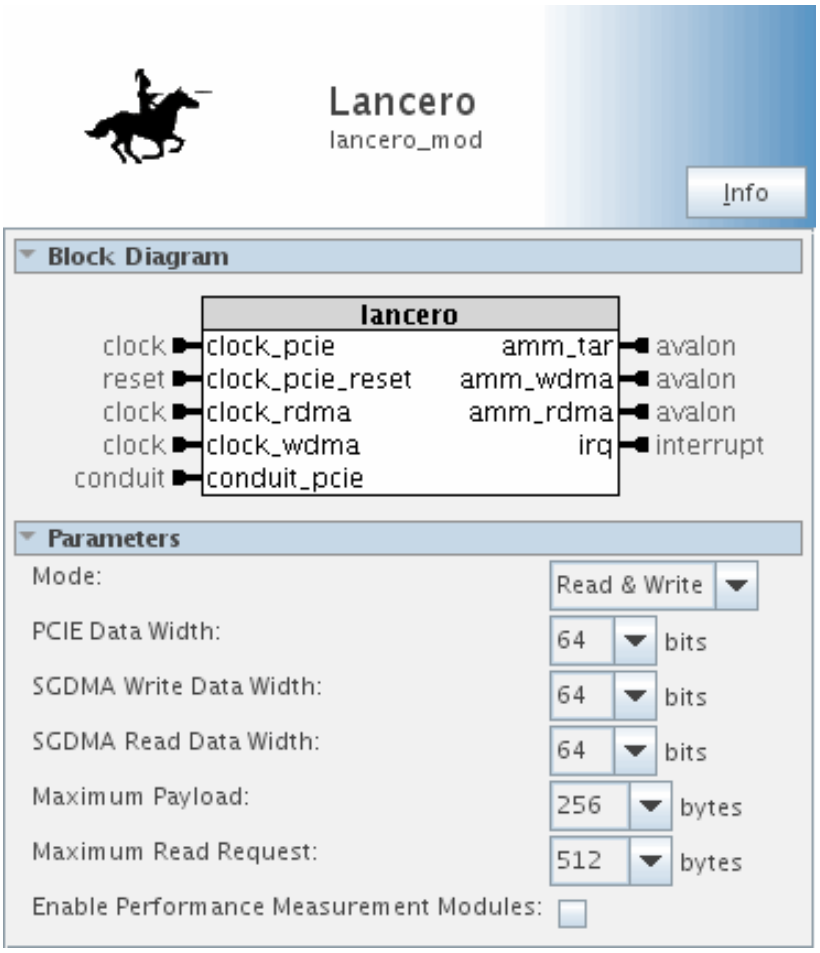


Fig. 11 The Lancero configuration GUI in SOPC Builder

The SGDMA write engine has a 32, 64 or 128-bits Avalon Memory-Mapped bus called 'amm_wdma' which performs writes to the PCIe host. The user can connect SOPC slave components to this bus for high performance data transfers. The bus supports burst tranfers. The write engine is controlled through the Target Bridge.

The SGDMA read engine has a 64-bit Avalon Memory-Mapped bus called 'amm_rdma' which performs reads from the PCIe host. The user can connect SOPC slave components to this bus for high performance data transfers. The bus supports burst tranfers. The read engine is controlled through the Target Bridge.

9.3 BAR1 Address Map

The Lancero components, including the SGDMA engines, are controlled through the target bus using BAR1. The base addresses for the Lancero components are:

Description	Target Bus Base Address (–End Address)
Configuration	0x000 (–0x0FF)
Interrupt Controller	0x100 (–0x1FF)
SGDMA Write Engine	0x200 (–0x2FF)
SGDMA Read Engine	0x400 (–0x4FF)

For the register map of the configuration controller and the interrupt controller see section 7.

9.4 SGDMA Read and Write Engine

An SGDMA engine transfers data between the local Avalon Memory-Mapped bus and the remote memory in the PCIe host.

The transfers are specified in a transfer descriptor list which are constructed by (driver) software in PCIe host memory. After this list is available, the PCIe host address of this list is set in an engine control register. The engine is then started.

The engine will autonomously prefetch the descriptors and perform the specified transfers. During its operation new transfers can be added to the list.

9.4.1 Write Engine

The write engine transfers data from the user SOPC component(s) on its Avalon Memory-Mapped bus to the PCIe host.

The write engine performs burst reads on the Avalon bus and performs (posted) memory writes on the PCIe bus. It

9.4.2 Read Engine

The read engine transfers data from the PCIe host to the user SOPC component(s) on the 64-bit Avalon Memory-Mapped bus.

The read engine is an initiator of (non-posted) memory reads on the PCIe bus. It performs burst writes on the Avalon data bus.

9.5 SGDMA Character Device

The SGDMA character device is used to perform high-speed SGDMA transfers between the PCIe host and the user SOPC components on the amm_rdma and amm_wdma buses.

9.5.1 Standard I/O

The SGDMA character device accepts `read` and `write` requests for any size which is a multiple of its data bus width, and to/from addresses that are aligned with its data bus width. Each I/O request results in one or more SGDMA transfers. After

the transfer completes, the `read` and `write` returns the number of bytes transferred, or a negative error code in case of errors.

The `read` and `write` engines work independently; `read` requests are handled in order of submission and `write` request are handled in order of submission.

When using the normal (blocking, synchronous) I/O request the SGDMA engines will be idle during setup and completion of the transfers. In order to achieve maximum performance, see the advanced I/O topic below.

9.5.2 Example: Transfer data from FPGA to application buffer using SGDMA

```
#define BUF_SIZE (1024 * 1024)
char *buffer = (char *)malloc(BUF_SIZE);
int fd = open("/dev/lancero_sgdma", O_RDWR);
int rc = read(fd, buffer, BUF_SIZE);
close(fd);
```

Using `malloc` a buffer of one Megabyte is allocated. Note that this buffer appears as one contiguous block in the application virtual address space but very likely lies scattered throughout the physical memory attached to the processor.

In the `open` call, the character device that corresponds with the SGDMA data bus on the FPGA is opened. A file descriptor is returned. You can now access this device like a file.

With the `read` call, one Megabyte is requested to be transferred from the device to the application buffer. During this call, the device driver will prepare and then initiate the transfer in hardware. The processor will be freed to run other tasks on your Linux system, while the driver waits for the hardware to finish. After the hardware finishes the transfer, the application continues and "transfer completed." is printed.

9.5.3 Advanced I/O

Multiple threads or applications can submit I/O requests simultaneously and transparently.

Additionally, the character device supports asynchronous `read` and `write` requests, which requires using the Linux `libaio` library.

When performing overlapping I/O by using multiple threads or asynchronous I/O requests, the transfers are handled back-to-back by the Lancero Linux driver and IP core, achieving maximum performance.

9.6 SGDMA Engine Register Map

Each SGDMA engine is controlled through its Avalon slave interface on the target bus.

Note: The Lancero Linux device driver hides the control from the user. The information below is needed only if the user is developing custom drivers or software to control the SGDMA engines.

The SGDMA engine control interface consists of seven 32-bit registers:

Byte Offset	Name	RC access	EP access	Description
0x00	identifier	R	R	Identifier and Version
0x04	status	R	W	Status
0x08	control	R/W	R	Control
0x0C	first_desc	R/W	R	First descriptor bus address
0x10	first_desc_adjacent	R/W	R	Number of descriptors in memory adjacent after the first descriptor.
0x14	completed_desc_count	R	W	Number of completed descriptors, updated by the engine after completion of each descriptor in the list.
0x18	completed_desc_bytes	R	W	Number of bytes transferred of the transfer related with the descriptor the engine is processing, updated by the engine during the transfer.
0x1c	Reserved	R/W	R	Must be set to zero (0).

All registers support 32-bit (PCI DWORD) access only.

The first_desc and first_desc_adjacent registers must be written before starting the engine.

9.6.1 Identifier and Version Register

This read-only fixed-value register allows the device driver to verify that the SGDMA engine component is available, and which version it is.

Identifier and Version Register		
31:24	23:8	7:0
Reserved	Identifier	Version

The identifier value indicates the variant of SGDMA engine, read or write.

Identifier Value	Description
0xC200	SGDMA Read Engine with Avalon Memory Master Bus
0xC100	SGDMA Write Engine with Avalon Memory Master Bus
others	Reserved

The version value indicates the hardware interface specification version

Version Value	Description
0x01	SGDMA Engine version 1
0x02	SGDMA Engine version 2
others	Reserved

9.6.2 Status Register

The status register indicates the current state of the SGDMA engine. The bits are set and reset by the SGDMA engine and are read-only.

Bit	Name	RC access	Description
0	BUSY	R	One (1) if the SGDMA engine is busy, zero (0) when it is idle.
1	DESCRIPTOR_ STOPPED	R	Reset (0) on setting RUN_STOP (1). Set (1) after the engine completed a descriptor with the STOP bit set.
2	DESCRIPTOR_ COMPLETED	R	Reset (0) on setting RUN_STOP (1). Set (1) after the engine completed a descriptor.
3	RESERVED	R	Reserved
4	MAGIC_ STOPPED	R	Reset (0) on setting RUN_STOP (1). Set (1) when the engine encountered a descriptor with invalid magic and stopped. This indicates memory corruption and/or a software bug.
5	FETCH_ STOPPED	R	Reset (0) on setting RUN_STOP (1). Set (1) when the engine could not fetch a descriptor due to a PCI Express completion timeout. This indicates a hardware problem or software misconfiguration.
6	IDLE_STOPPED	R	Reset (0) on setting RUN_STOP (1). Set (1) when the engine stopped after resetting RUN_STOP (0).
7	PAYLOAD_ MISMATCH	R	(Re)set during PCIe configuration. Set (1) when the system payload is larger than Lancero was configured for in SOPC Builder. This might decrease write engine performance.
8	MAXREAD_ MISMATCH	R	(Re)set during PCIe configuration. Set (1) when the system maxread is larger than Lancero was configured for in SOPC Builder. This might result in read engine data corruption.
9	NONALIGNED_ STOPPED	R	Reset (0) on setting RUN_STOP (1). Set (1) when the transfer is not aligned properly.
10-31	Reserved	R	Reserved

Bits 7 through 9 was introduced in version 2 of the SGDMA engine. They are zero in older versions.

9.6.3 Control Register

The status register indicates the current state of the SGDMA engine. The bits are set and reset by the SGDMA engine and are read-only.

Bit	Name	RC access	EP access	Description
0	RUN_STOP	RW	R	Set to one (1) to start the SGDMA engine. Reset to zero (0) to stop it; if it was busy it will complete the current descriptor. Flush earlier writes when accessing this register, and flush after.
1	IE_DESCRIPTOR_STOPPED	RW	R	Set to one (1) to interrupt after the engine stopped after completing a descriptor which has the STOP bit set.
2	IE_DESCRIPTOR_COMPLETED	RW	R	Set to one (1) to interrupt after the engine completed a descriptor which has the IR_DESCRIPTOR_COMPLETED control bit set.
3	RESERVED	RW	R	Reserved
4	IE_MAGIC_STOPPED	RW	R	Set to one (1) to interrupt when the engine encountered a descriptor thas invalid magic and therefore stopped.
5	IE_FETCH_STOPPED	RW	R	Set to one (1) to interrupt when the engine could not fetch a descriptor due to a PCI Express completion timeout.
6	IE_IDLE_STOPPED	RW	R	Set to one (1) to interrupt when the engine stopped due to RUN_STOP being reset (0).
7-8	Reserved	R	R	Reserved
9	IE_NONALIGNED_STOPPED			Set to one (1) to interrupt after the engine stopped due to encountering a nonaligned transfer.
10-31	Reserved	R	R	Reserved

Bit 9 was introduced in version 2 of the SGDMA engine. It is zero in older versions.
It is adviced that all interrupt enable flags are set when an SGDMA engine is started. This ensures that any reason why the engine stopped (expectedly or unexpectedly) can be noticed in the device driver.

9.6.4 First Descriptor Register

The first_desc register must be set with the address of the first descriptor in host memory, before the SGDMA engine is started.

9.6.5 First Descriptor Adjacent Register

The first_desc_adj register must be set to 0. (In future releases, this allows prefetching multiple descriptors which are contiguous in memory.)

9.6.6 Descriptor Completed Register

The desc_completed register is read-only and holds the number of descriptors that have been completed by the engine. It is set to zero on the engine BUSY rising edge,and is incremented after each descriptor that was completed. The value remains stable after the engine BUSY flag falling edge.

9.7 SGDMA Descriptor

Each descriptor contains the information for a transfer of a memory area that is contiguous in both the Avalon bus and remote PCIe memory address spaces.

A descriptor has the address of the next descriptor so that a linked list of descriptors can be processed by the SGDMA engine. Each descriptor has flags to control the engine when it completes processing the descriptor.

Byte Offset	Fields			
	31:16	15:12	11:8	7:0
0x00	magic 0xAD4B	reserved (4 bits)	extra_ adjacent	control
0x04	length [31:3]			
0x08	fpga_addr [31:3]			
0x0C	fpga_addr_pad			
0x10	pcie_addr [31:3]			
0x14	pcie_addr_pad			
0x18	next_addr [31:2]			
0x1C	next_addr_pad			

9.7.1 Descriptor Fields

The length and address fields describe a single memory transfer. The control field controls per-transfer actions the engine should take.

A wrong value in the magic field protects against software bugs or memory corruption. The engine will stop with the status MAGIC_STOPPED flag set.

The next_addr field indicates where the next descriptor is and the extra_adjacent helps to prefetch multiple descriptors in one request.

Byte Offset	Name	RC access	EP access	Description
0x00	control	R/W	R	Descriptor control flags.
0x01	extra_adjacent	R/W	R	Number of descriptors (0-15) that are adjacent in memory after the descriptor at next_addr.
0x02	reserved	R/W	R	Reserved. Must be set to zero.

Byte Offset	Name	RC access	EP access	Description
0x03	magic	R/W	R	Must be set to 0xAD4B. On mismatch the engine will stop.
0x04	length	R/W	R	Length of the transfer in bytes. This must be a positive multiple of the SGDMA data bus width.
0x08	fpga_addr	R/W	R	The local Avalon bus start address of the transfer.
0x0C	fpga_addr_pad	R/W	R	Reserved. Must be set to zero.
0x10	pcie_addr_lo	R/W	R	The remote PCIe bus start address of the transfer. This address can relate to memory in the PCIe Root Complex or another End Point. The upper 32 bits of the address is set in pcie_addr_hi, this is 0 for 32-bit addresses.
0x14	pcie_addr_hi	R/W	R	
0x18	next_addr	R/W	R	The remote PCIe bus start address of the next descriptor in the descriptor list, or zero (0) if this is the last descriptor in the list. The address can relate to memory in the PCI Express Root Complex or another End Point.
0x1C	next_addr_pad	R/W	R	Reserved. Must be set to zero.

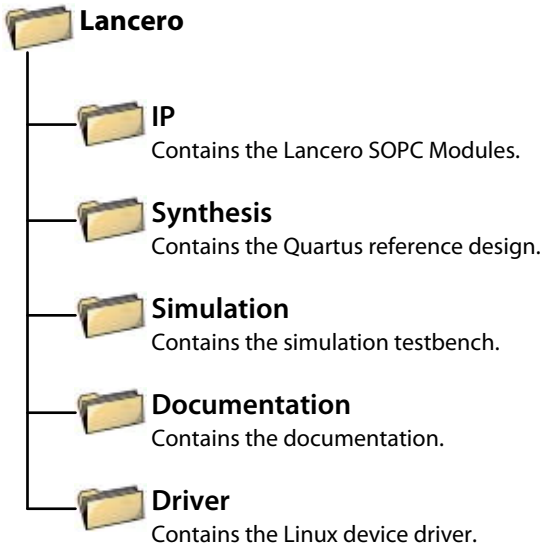
9.7.2 Descriptor Control Flags

Bit	Name	Description
0	STOP	Set to (1) to stop the engine when it completed this descriptor.
1	IR_DESCRIPTOR_COMPLETED	Set to (1) to interrupt after the engine completed this descriptor. This requires the global IE_DESCRIPTOR_COMPLETED control flag set in the SGDMA control register.
2	Reserved	Reserved. Must be set to zero.
3	FREEZE_FPGA_ADDR	Set to (1) to not increment fpga_addr during the transfer, typically used for FIFOs. Note that if your SOPC slave component is non-bursting, the SOPC builder will insert a bursting adapter resulting in seeing the burst count incrementing.
others	Reserved	Reserved. Must be set to zero.

10 Quick Start Reference Designs

10.1 Deliverables

Lancero is delivered as a file archive that must be extracted to your workstation. It contains the following:



10.2 Reference Designs

The reference designs are located in the synthesis directory. The reference designs have the following hierarchical design structure:

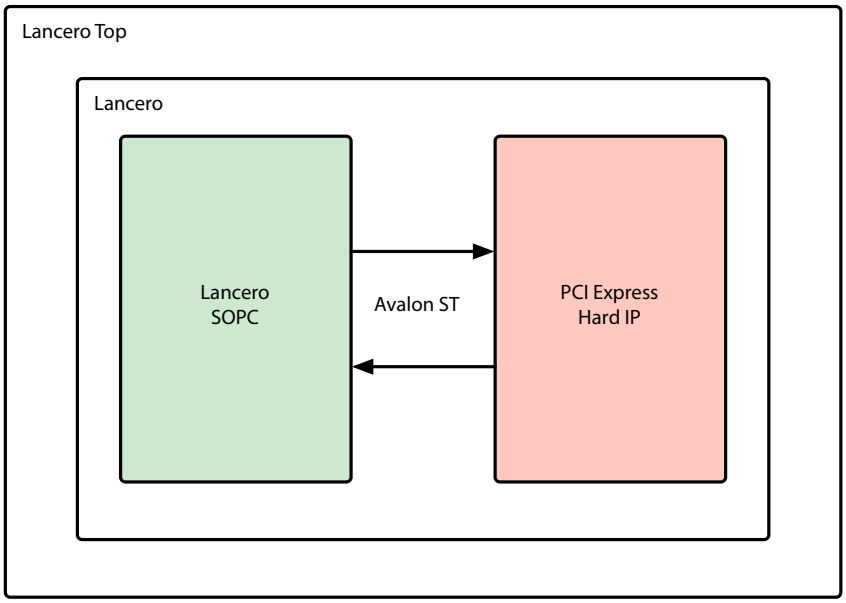


Fig. 12 Top level design of the Lancero reference design.

The following designs are readily available:

Name	Description
alt_a2gx_dex_x1	Altera Arria II GX development board single lane
alt_a2gx_dex_x4	Altera Arria II GX development board four lanes
alt_a2gx_dex_x4	Altera Arria II GX development board eight lanes
alt_c4gx_str_x1	Altera Cyclone IV GX starters kit single lane
terasic_de4	Terasic DE4 board eight lanes

Following these step-by-step instructions helps you generate a Lancero reference project:

- Add the Lancero OpenCore license to the Quartus license file.
- Start Quartus.
- Open a Lancero project from the 'synthesis' directory. Multiple projects are available. For the Arria II GX development board x4 reference design open the `lancero_alt_a2gx_dev_x4.qpf` file in the `alt_a2gx_dev_x4` directory. For the Cyclone IV GX development kit open the `lancero_alt_c4gx_str_x1.qpf` file in the `alt_c4gx_str_x1` directory.

The first step is to generate the PCI Express Hard IP core:
- Open the MegaWizard (Tools —› MegaWizard).
- Select the option 'Edit an existing megafunction variation' and click 'Next'.
- Select the file '`pcie_hip.v`' and click 'Next'.
- Click 'Finish' to generate the core.
- After the PCI Express compiler has finished, click 'Exit' to leave the Wizard.

The second step is to generate the SOPC Builder system:
- Start the SOPC Builder (Tools —› SOPC Builder).
- Initially the Lancero library must be added to the IP search path, this can be found in Tools —› Options. Add the directory 'IP'.
- Click on 'Finish' to leave the options menu.
- Click on 'Generate' to create the SOPC system.
- After the system was succesfully generated, click on 'Exit' to leave the SOPC Builder.

The project can now be built.
- Compile the project (Processing —› Start Compilation).
- Optionally program the board (Tools —› Programmer).

10.3 User Components

The reference design has some user SOPC components attached to the target bus and SGDMA buses.

10.4 On-chip memory attached to target bus

A small on-chip RAM slave is or can be attached to the target bus amm_tar.

10.5 Write tester attached to SGDMA write bus

The SGDMA amm_wdma bus has a write tester attached to it. This is an SOPC Avalon Memory-Mapped slave component. When reading from any location in its address space, the write tester returns incremental 32-bit sequence numbers, starting at zero. An internal counter keeps state.

The purpose of the write tester is to provide the write SGDMA engine with deterministic data which can then be tested by the PCIe host after DMA has finished.

10.5.1 Register Map

The write tester is located at base address 0x1000 and has the following registers for identification and control:

Byte Offset	Name	RC access	EP access	Description
0x00	identifier	R	R	Identifier and Version
0x04	control	RW	R	Control

All registers support 32-bit (PCI DWORD) access only.

10.5.2 Identifier and Version Register

This read-only fixed-value register allows the software to inquire the write tester component.

Identifier and Version Register		
31:24	23:8	7:0
Reserved	Identifier	Version

Identifier Value	Description
0xD100	Write Tester Identifier
other values	Reserved

The version value indicates the hardware interface specification version.

Version Value	Description
0x01	Write Tester version 1
other values	Reserved

10.5.3 Control Register

Bit	Name	RC access	Description
0	CLR	R/W	Write one (1) to clear the 32-bit data generator counter. The bit is cleared by the tester when the command has been accepted.

10.6 Read tester attached to SGDMA read bus

The SGDMA amm_rdma bus has a read tester attached to it. This is an SOPC Avalon Memory-Mapped slave component. When writing to any location in its

address space, the read tester will compare the written data against an incremental 32-bit data counter, starting at zero and incrementing with each 32-bit write. On mismatch a status bit is set and an interrupt is raised.

The purpose of the read tester is to verify deterministic data is correctly transferred through the read SGDMA engine. The PCIe host is responsible for resetting the read tester and preparing the data that is transferred to contain incremental 32-bit numbers.

10.6.1 Register Map

The read tester is located at base address 0x2000 and has the following registers for identification and control:

Byte Offset	Name	RC access	EP access	Description
0x00	identifier	R	R	Identifier and Version
0x04	control	RW	R	Control
0x08	status	R	W	Status

All registers support 32-bit (PCI DWORD) access only.

10.6.2 Identifier and Version Register

This read-only fixed-value register allows the software to inquire the read tester component.

Identifier and Version Register		
31:24	23:8	7:0
Reserved	Identifier	Version

Identifier Value	Description
0xD200	Read Tester Identifier
other values	Reserved

The version value indicates the hardware interface specification version.

Version Value	Description
0x01	Read Tester version 1
other values	Reserved

10.6.3 Control Register

Bit	Name	RC access	Description
0	CLR	R/W	Write one (1) to clear the 32-bit data receptor counter and ERROR status. The bit is cleared by the tester when the command has been accepted.
1	IE	R/W	When this bit is set, an interrupt is generated when a data mismatch occurs.

10.6.4 Status Register

Bit	Name	RC access	Description
0	ERROR	R	One (1) when written data does not match the 32-bit data receptor counter. The bit can be cleared through the control register.

The status register indicates the current state of the read tester.

11 Simulation

11.1 Testbench

The ModelSim simulation testbench is provided in the directory simulation. Figure 10 shows the testbench top level module. The testbench is derived from the Altera PCI Express testbench, generated by the Altera PCI Express compiler. For more information about the testbench and root port bus functional model (BFM) please refer to chapter 7 of the Altera PCI Express compiler user guide.

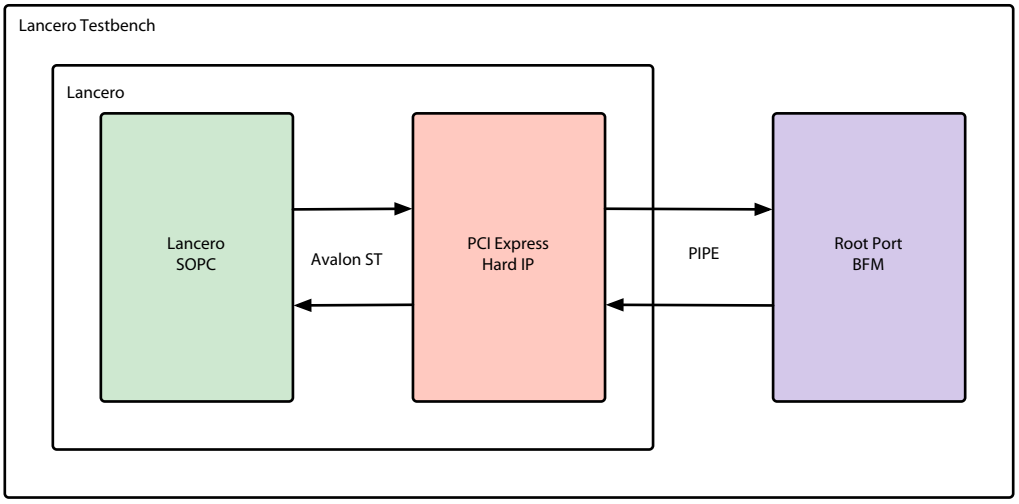


Fig. 13 Overview of the testbench used for simulation.

The module `altpcie_bfm_driver` generates transactions to the root port BFM in order to test the Lancero system. The driver performs a single read, single write, SGDMA read and SGDMA write test.

To perform a simulation using Modelsim execute the `run.do` script (`vsim -do run.do`). The wave script (`wave.do`) adds the Lancero signals of interest to the ModelSim wave viewer.

The Lancero modules and testbench are written in (System)Verilog. To simulate the testbench a ModelSim Verilog license is required. A mixed language license is required, when other custom VHDL modules are added to the Lancero testbench.

11.2 OpenCorePlus License

The Altera OpenCorePlus evaluation method allows to validate and evaluate Lancero in real hardware before purchasing a full license. OpenCorePlus hardware evaluation supports the tethered operation mode for which a connection between the board and the Quartus programmer is required. The OpenCorePlus files are located in the OpenCore directory. Please contact Logic & More about the OpenCorePlus possibilities.

12 Revision histories

12.1 Lancero IP Core revision

History of major changes to the Lancero IP core specification:

Revision	Change description
1.0	Original document
1.1	Target bridge now support 8-bit and 16-bit access
1.8	64-bit PCIe addressing to support 36-bits (PAE) and 64-bits hosts
2.0	eight-lane PCIe support, 128-bits Avalon ST interconnect with PCIe IP

12.2 Manual revisions

History of changes to this manual:

Revision	Change description	Lancero revision
1.0-1.2	draft revisions	1.0
1.2	first release	1.0
1.3	extended documentation of the IP core internals.	1.1
1.4	fixed some typo fixes, used 0666 permissions mknod	1.1
1.5	mention address mapping for interrupt controller	1.1
1.6	Target and SGDMA modules, BAR0 and BAR1	1.1
1.7	Added description of reference design user components added pcie_addr_hi for 64-bit SGDMA host support.	1.8
1.8	Internal release	1.8
1.9	Added interrupt control event pending, corrected write/read tester register description.	1.8
1.10	Mentioned the effect of the bursting adapter for non-bursting slaves in combination with the FREEZE_FPGA_ADDR descriptor control flag.	1.8
1.10	Added eight-lanes and 128-bit PCIe support, configurable DMA Avalon bus data widths (32 to 256 bits).	2.0
1.11	Added the Lancero SOPC configuration GUI window. Mention the available reference designs. Removed the now redundant step for reconfig.v generation.	2.0

13 Support

13.1 Support

For technical support, information requests and updates surf to:

http://www.logicandmore.com/

or send an e-mail to lancero@logicandmore.com.