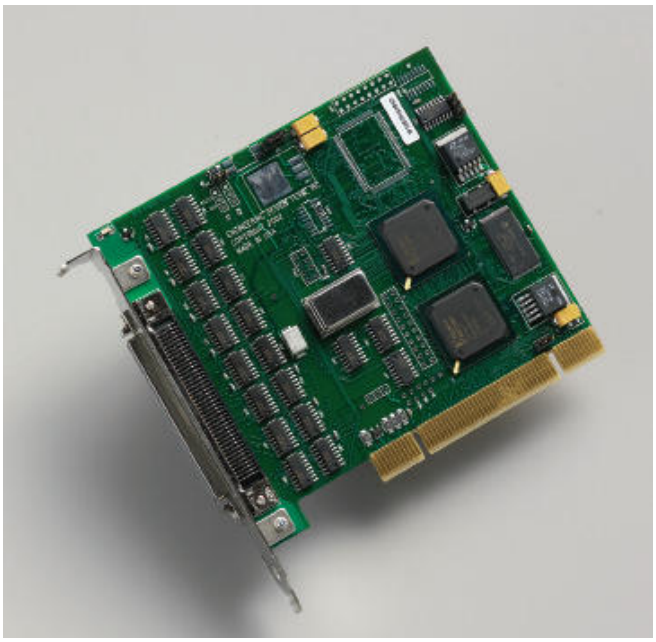
PCI CD/CDa

Configurable DMA Interface

for PCI Local Bus Computers



May 2007

008-00965-08

International Distributors

Sky Blue Microsystems GmbH
Geisenhausenerstr. 18
81379 Munich, Germany
+49 89 780 2970, info@skyblue.de
www.skyblue.de

In Great Britain:
Zerif Technologies Ltd.
Winnington House, 2 Woodberry Grove
Finchley, London N12 0DR
+44 115 855 7883, info@zerif.co.uk
www.zerif.co.uk

Contents

The PCI Bus Configurable DMA Interface 1
Related Manuals 2

About the DMA Interface 3

About the Software and Firmware 4
CDa FPGA Configuration Files 4
CD-20 and CD-60 FPGA Configuration Files 4
Software Initialization Files 4
The PCD Device Driver 5
FPGA Configuration Files 5
Software Initialization Files 5
Sample Applications and Utilities 6
Sample Applications 6
Utility Files 6
Testing Files 7
Building Applications 7

Configuring the PCI CD/CDa 8
Checking the PCI FPGA Firmware 8
Loading the UI FPGA Firmware and Configuring the PCI CD/CDa 9
Using Custom FPGA Configuration Files 9

Testing 10

Generating an Output Clock 11
PCI CDa 11
PCI CD 11

Hardware Interface Protocol 13
Electrical Interface 13
RS-422 14
LVDS 14
Signals 15
Timing 16
Connector Pinouts 18

Registers 19
Configuration Space 19
PCI Local Bus Addresses 21
Scatter-gather DMA 21
Main DMA Current Address Register 22
Main DMA Next Address Register 23
Main DMA Current Count and Control Register 23
Main DMA Next Count and Control Register 23

Scatter-gather DMA Current Address Register....24
Scatter-gather DMA Next Address Register....24
Scatter-gather DMA Current Count and Control Register....24
Scatter-gather DMA Next Count and Control Register....25
PLL Programming Register....26
Flash ROM Address Register....26
Flash ROM Data Register....27
PCI Interrupt and UI Xilinx Configuration Register....27
PCI Interrupt Status Register....28
UI Xilinx Data Register....28
UI Xilinx Registers....29
Command Register....29
Data Path Status Register....30
Funct Register....30
Stat Register....31
Stat Polarity Register....31
Direction Control Registers....32
Programmed I/O Low Register....33
Programmed I/O High Register....33
Interface Configuration Register....34
PCI CDa Registers....35
PLL Programming Register....35
PLL Divider Register....35
Output Data Valid Delay Register....35
LED Control Register....36

References....37

The PCI Bus Configurable DMA Interface

The Configurable DMA Interface (PCI CD/CDa) is a single-slot DMA input/output interface for PCI Bus-based computer systems. It is designed for continuous input or output between a user device and PCI Bus host memory. This interface is typically used to move data to or from a PCI Bus host computer to devices such as scanners, plotters, imaging devices, or research prototypes.

The basic configuration of the PCI CD/CDa uses a bidirectional single-channel 16-bit parallel protocol. In a 66 MHz PCI Bus slot, it can transfer up to 210 MB per second. It is available using either RS-422 or LVDS signal levels. A variety of other options are also available: a 16-channel synchronous serial protocol, a 4-channel synchronous serial protocol, a single channel synchronous serial protocol, or an 8-bit parallel protocol. Manuals describing these options are listed in Related Manuals on page 2.

The PCI CD-20 (RS-422) and the PCI CD-60 (LVDS) are older versions of the PCI CDa that use different versions of the DMA engine, but have many of the same capabilities. However, they operate only in a 33 MHz PCI Bus slot.

The PCI CD/CDa implements the DMA interface using two Xilinx field-programmable gate arrays, referred to as the PCI Xilinx and the UI (user interface) Xilinx.

- The *PCI Xilinx* communicates with the host computer over the PCI Bus. It transfers data between the board and the host computer, and loads its firmware on powerup from flash ROM.
- The UI Xilinx transfers data between the user device and the board, through the connector. The UI Xilinx may also process the data in some manner, depending on the application.

Each Xilinx must be loaded with the firmware specific to the chosen interface, and the firmware in one Xilinx must be compatible with the firmware in the other. The PCI CD/CDa comes configured for the interface you ordered; a list of the firmware files provided is available in About the Software and Firmware on page 3, and instructions for loading them, if necessary, are provided in Configuring the Board on page 7.

When data comes in from the user device, the UI Xilinx sends it to input and output FIFO buffers, which smooth data transfer between the PCI bus and the user device, as well as accommodating data during the transition from one DMA to the next. DMA transfers are queued in hardware, minimizing the amount of FIFO required. The PCI CD/CDa uses 4 KB input and output FIFOs; the PCI CDa implements these FIFOs internally in the UI Xilinx; the PCI CD-20 and CD-60 implement them externally.

Either the PCI CD/CDa interface or your own device can generate the receive or transmit timing, or each can generate its own transmit timing.

About the Software and Firmware

The PCI CD/CDa comes with firmware files to configure the two Xilinxes (having the extension `.bit`), a variety of utility applications, a firmware file to use for testing the board, and software initialization files (having the extension `.cfg`) to use to initialize the board for each configuration.

The PCI Xilinx firmware files are in the `flash` subdirectory of the EDT top-level distribution directory. UI Xilinx firmware files are in the `bitfiles` subdirectory of the EDT top-level distribution directory.

The following files are included:

CDa FPGA Configuration Files

`cda1.bit` PCI Xilinx configuration file for single-channel, 16-bit parallel input/output for RS-422 or LVDS.

Either of the two UI Xilinx configuration files can be used with the PCI Xilinx configuration file provided:

`pcda.bit` UI Xilinx configuration file for single-channel, 16-bit parallel input/output for RS-422 or LVDS.

`xtest.bit` UI Xilinx configuration file for running the test described in Testing on page 10

CD-20 and CD-60 FPGA Configuration Files

The PCI Xilinx must be loaded with the FPGA configuration file intended for its model:

`pcd20.bit` PCI Xilinx configuration file for 16-bit parallel input/output for the PCI CD-20 (RS-422).

`pcd60.bit` PCI Xilinx configuration file for 16-bit parallel input/output for the PCI CD-60 (LVDS).

Any of the UI Xilinx configuration files can be combined with either of the PCI Xilinx configuration files:

`pcd_src.bit` UI Xilinx configuration file for 16-bit parallel input/output, synchronized to the PCI CD source clock.

`pcd_looped.bit` UI Xilinx configuration file for 16-bit parallel input/output, synchronized to an external clock.

`xtest.bit` UI Xilinx configuration file for running the test described in Testing on page 10.

Software Initialization Files

Sample software initialization files for all board configurations are in the `pcd_config` subdirectory of the distribution directory.

Software initialization files are editable text files that you can customize for your own applications.

`pcda.cfg` Software initialization file for single-channel, 16-bit parallel input/output for RS-422 or LVDS, for the PCI CDa only.

`pcd_src.cfg` Software initialization file for 16-bit parallel input/output, synchronized to the PCI CD source clock, for the PCI CD-20 and CD-60 only.

`pcd_looped.cfg` Software initialization file for 16-bit parallel input/output, synchronized to an external clock, for the PCI CD-20 and CD-60 only.

FPGA Configuration Files

The main board implements the DMA interface using two field-programmable gate arrays (FPGAs), referred to as the PCI FPGA and the UI (user interface) FPGA:

- The *PCI FPGA* communicates with the host computer over the PCI Bus. It implements the DMA engine, which transfers data between the board and the host computer, and loads its firmware on powerup from flash ROM located on the main board.
- The *UI FPGA* transfers data between the user device and the PCI FPGA; in some instances, it also sends the data to the mezzanine board. The UI FPGA or mezzanine board may also process the data in some manner, depending on the application.
- In addition, some mezzanine boards also have an FPGA which must be loaded with the correct firmware.

FPGA configuration files define the firmware required for the PCI FPGA and the UI FPGA. The PCI FPGA firmware files are in the `flash` subdirectory of the EDT top-level distribution directory. UI FPGA firmware files are in the `bitfiles` subdirectory of the EDT top-level distribution directory. mezzanine board configuration files, if any, are in the mezzanine board-specific subdirectory.

Each FPGA must be loaded with the firmware specific to the chosen interface, and the firmware in one FPGA must be compatible with the firmware in the other. By default, the correct FPGA configuration file for the PCI FPGA is loaded at the factory. However, you'll need to load the required FPGA configuration file for the UI FPGA yourself.

The firmware files specific to your PCI CD/CDa are listed at the beginning of this section. Instructions for loading them are provided in Configuring the PCI CD/CDa.

The PCD Device Driver

The PCD device driver is the software running on the host that allows the host operating system to communicate with the PCI CD/CDa. The driver is loaded into the kernel upon installation, and thereafter runs as a kernel module. The driver name and subdirectory is specific to each supported operating system; the installation script handles those details for you, automatically installing the correct device driver in the correct operating system-specific manner.

Software Initialization Files

Software initialization files (having the extension `.cfg`) are editable text files that run like scripts to configure EDT boards so that they are ready to perform DMA. The commands in a software initialization file are defined in a C application named `initpcd`. When you invoke `initpcd`, you specify which software initialization file to use with the `-f` flag.

A typical software initialization file loads an FPGA configuration file into the UI FPGA and sets up various registers to prepare the board for DMA transfers. Some software initialization files may also load an FPGA configuration file into an FPGA residing on the mezzanine board.

A variety of software initialization files are included with the EDT software, at least one of which is customized for each main board or main board / mezzanine board combination — that is, each FPGA configuration file has a matching software initialization file. Software initialization files are located in the `pcd_config` subdirectory of the EDT top-level distribution directory. The software initialization files specific to your PCI CD/CDa are listed at the beginning of this section. Instructions for their use are provided in Configuring the PCI CD/CDa.

Commands defined in `initpcd` and typically found in software initialization files allow for specific FPGA configuration files to be loaded (for example, `bitfile:`), write specified hexadecimal values to specified registers (for example, `command_reg:`), enable or disable byte-swapping or short-swapping to accommodate different operating systems' requirements for bit ordering (for example, `byteswap:`), or invoke arbitrary commands (for example, `run_command:`). If the mezzanine board you're using has

an FPGA, you can load the appropriate firmware on it by running the command `mezzload`, which determines the appropriate firmware for the specific mezzanine board in your system.

For example:

```
bitfile: ssd16io.bit
command_reg: 0x08
byteswap: 1
run_command: set_ss_vco -F 1000000 2
run_command: mezzload
```

For complete usage details, enter `initpcd --help`.

C source for `initpcd` is included so that you can add your own commands, if you wish. You can then edit your own software initialization file to use your new commands and specify that `initpcd` use your new file when configuring your board. If you would like us to include your new software initialization commands in subsequent releases of `initpcd`, send mail to `tech@edt.com`.

Sample Applications and Utilities

Along with the driver, the FPGA configuration files, and the software initialization files, the software CD includes a number of applications and utilities that you can use to initialize and configure the board, access registers, or test the board. For many of these applications and utiilities, C source is also provided, so that you can use them as starting points to write your own applications. The most commonly useful are described below; see the README file for the complete list.

NOTE Software is updated regularly; the latest versions are available on our website at www.edt.com/software.html. We encourage you to use the latest versions for new installations. For existing applications, upgrade only if you have a specific reason to do so.

Sample Applications

`rd16`	Performs simple multichannel ring buffer input.
`wr16`	Performs simple multichannel ring buffer output.
`simple_read`	Performs DMA input without using ring buffers. Data is therefore subject to interruptions, depending on system performance.
`simple_write`	Performs DMA output without using ring buffers. Data is therefore subject to interruptions, depending on system performance.
`simple_getdata`	Serves as an example of a variety of DMA-related operations, including reading the data from the connector interface and writing it to a file, as well as measuring input rate.
`simple_putdata`	Serves as an example of a variety of DMA-related operations, including reading data from a file and writing it out to the connector interface.
`test_timeout`	Under normal operation, timeouts cancel DMA transfers. This application exemplifies giving notification when a timeout occurs, without canceling DMA
`set_ss_vco`	A utility for programming the output clock or clocks on the PCI CD/CDa to specific frequencies used by the UI FPGA for input and output.

Utility Files

`initpcd`	A utility for initializing and configuring the PCI CD/CDa.

`pdb` Utility application that enables interactive reading and writing of the PCI SS/GS UI FPGA registers.

Testing Files

A variety of files — C source, executables, and FPGA configuration files — are available to test the boards. Their uses are described in the documents listed under the heading Testing Procedures. They include at least:

`sslooptest` Tests most PCI SS- and PCI GS-based boards. Determines the board model and selects the loopback test to run, then runs it.

`xtest` Tests the PCI CD and CDa boards, and the single-channel DMA interface for the PCI SS and PCI GS main boards.

Building Applications

Executable and PCD source files are at the top level of the EDT PCD driver distribution directory. If you need to rebuild an application, therefore, run `make` in this directory.

Windows and Solaris users must install a C compiler. For Windows, we recommend the Microsoft Visual C compiler; for Solaris, the Sun WorkShop C compiler. Linux users can use the `gcc` compiler typically included with your Linux installation. If Solaris or Windows users wish to use `gcc`, contact `tech@edt.com`.

After you've built an application, use the `--help` command line option for a list of usage options and descriptions.

Configuring the Board

For your EDT board to operate as you require, it must be loaded with the appropriate FPGA configuration files for both FPGAs. The PCI FPGA is loaded from flash ROM, which is shipped from the factory already loaded with the appropriate FPGA configuration file; however, you must load the UI FPGA yourself.

Before loading the UI FPGA, however, you may wish to check the firmware in the PCI FPGA to ensure that it is correct and up-to-date.

Checking the PCI FPGA Firmware

When upgrading to a new device driver, or switching to a FPGA configuration file with special functionality, you may also need to reprogram the PCI interface flash PROM using `pciload`.

The following procedure applies to standard firmware only. If you are running a custom firmware file and need to update it, first get a custom firmware configuation file from EDT.

NOTE The presence of a newer version of the firmware with a new driver doesn't necessarily mean that the firmware must be updated; if a package contains a mandatory upgrade, it is prominently stated in the README file.

On UNIX systems, `pciload` is an application in the installation directory `/opt/EDTpcd`.

On Windows systems, double-click the Pcd Utilities icon to bring up a command shell in the installation directory `\EDT\Pcd`.

On Macintosh systems, `pciload` is an application in the installation directory `/Applications/EDT/pcd`.

To see currently installed and recognized EDT boards and drivers, enter:

```
pciload
```

The program outputs the date and revision number of the firmware in the PROM.

To compare the PCI FPGA firmware in the package with the one already loaded on the board, enter:

```
pciload verify
```

The program compares the firmware in the PROM against the firmware file in the installation directory. If they match, there's no need to upgrade the firmware. If they differ, you'll see error messages. This does not necessarily indicate a problem; if your application is operating correctly, you may not need to upgrade the firmware.

If you wish to update the standard firmware, enter:

```
pciload update
```

1. To upgrade or switch to a custom firmware file, enter:

   ```
   pciload firmware_filename
   ```

   replacing *`firmware_filename`* with the name of the PCI FPGA configuration file, with or without the `.bit` file extension.

NOTE If the host computer holds more than one board, you can specify the correct board to load with the optional *`unit_number`* argument (by default, 0 for the first or only board in a host):

```
pciload -u unit_number filename
```

2. At the prompt, press **Enter** to confirm the loading operation. (If the file date is older than the PROM ID date, you may need to press **Enter** twice.)

The board reloads the firmware from the PROM only during power-up, so after running `pciload`, the old firmware remains in the PCI FPGA until the system has power-cycled.

NOTE Updating the firmware requires cycling power, not simply rebooting.

For a list of all `pciload` options, enter:

```
pciload --help
```

Loading the UI FPGA Firmware and Configuring the PCI CD/CDa

The utility `initpcd` loads the UI FPGA configuration files, programs the registers, sets the clocks (if necessary), and gets the PCI CD/CDa mezzanine board ready to perform DMA. This utility takes, as an argument, a software initialization file, and then automatically runs the pertinent commands.

If you use `initpcd` to configure the PCI CD/CDa, your application can concern itself solely with performing DMA and other application-specific operations; it will therefore omit PCI CD/CDa-specific operations and be portable to other EDT boards that peform DMA.

To configure the PCI CD/CDa, enter:

```
initpcd -u unit_number -f pcd_config/filename.cfg
```

replacing `unit_number` with the number of the board (by default, 0), and replacing *`filename`* with one of the initialization files listed in About the Software and Firmware; for example:

```
initpcd -f pcda.cfg
```

NOTE Software initialization files are editable text files. If the files provided don't meet your needs, copy and modify the one that's closest to your required configuration, then run `initpcd` with your new file.

Using Custom FPGA Configuration Files

You can substitute your own FPGA configuration file, if necessary. If you wish to develop your own VHDL design, contact EDT. When you're done, be sure to create a new software initialization file for your new firmware file and update the `pcd_config` directory to include it.

Testing

When you run this test, the PCI CD/CDa is configured with the FPGA configuration file `xtest.bit`. For normal operation, reconfigure the board with `initpcd` after completing the test, as described in Loading the UI FPGA Firmware and Configuring the PCI CD/CDa on page 8, to reconfigure the board with the correct UI Xilinx configuration file.

To verify that installation was successful and that the PCI CD/CDa is operating correctly, run `xtest`. At a command prompt, enter:

```
xtest -i 4096
```

The number following `-i` specifies the number of bytes to test. The PCI CD returns test status information. The following is an example of proper behavior, although details will vary:

```
reading 4096 words
buf at 820000
testing dirreg at 4 4
testing dirout at 8 8
testing dirin at 8 c
testing ctlout at a a
testing ctlin at a e
Calling DMA read 8192 at 820000
return to do read:
read returned length 8192
Done.
checking data
4096 words 0 errors
buf 0 820000
buf 1 920000
reading 100 buffers of 1048576 bytes from unit 0 with 2 bufs
return to start: starting read at 820000
starting read at 920000
hit return to continue:
counter0 4628 2362958141 counter1 4628 3122437420 freq 0 266230000
dtime 759479279.000000 ticktime 266230000.000000
time is 2.852719 sec
36757077.671371 bytes/sec
```

Problems are indicated by the string `ERROR -` followed by an error message. If you do not see such a string in the output, then the test completed satisfactorily.

Generating an Output Clock

The PCI CD/CDa has a programmable frequency generator, allowing you to specify precisely the frequency at which to transmit data.

PCI CDa

The PCI CDa has one programmable clock with a range of 168 Hz – 100 MHz. Most frequencies between these extremes can be achieved with little error. (Some FPGA configuration files allow an extended range of up to 200 MHz.)

The programmable clock is implemented with an ICS307-02 clock generator followed by a 14-bit programmable divider. The reference clock to the ICS307-02 is 10.3861 MHz. For details on the ICS307-02, consult Integrated Circuit Systems.

NOTE The library routine `set_ss_vco.c` includes an example of setting the output clock frequency.

The following three library routines, documented in the EDT DMA Software Library, help compute the output clock frequency:

`edt_find_vco_frequency_ics307`
: Computes the PLL parameter for the ICS307 chip, based on an input clock frequency and a target frequency.

`edt_set_out_clk_ics307`
: Sets the frequency output using parameters computed by `edt_find_vco_frequency_ics307`.

`edt_set_frequency_ics307`
: A convenience function to to set the frequency on the desired channel by first calling `edt_find_vco_frequency_ics307` or `edt_find_vco_frequency_ics307_nodivide`, then `edt_set_out_clk_ics307`.

PCI CD

The output clock is generated from a phase-locked loop (PLL) oscillator, a reference crystal, and programmable dividers. Because each of these components has physical limits, it may not be possible to get exactly the frequency desired. To get the expected results, you need to understand how the clock generator operates. Figure 1 diagrams how the final value is generated, using the following notation:

f_{xtal} 10 MHz (PCI CD-20 or 30 MHz (PCI CD-60)

f_{ref} The PLL reference frequency must be between 200 KHz and 5.0 MHz.

f_{vco} The VCO output frequency must be between 50 MHz and 250 MHz.

f_{fback} The VCO varies f_{vco} until the feedback frequency matches the PLL reference frequency.

f_{xilinx} The input frequency into the high speed odd divider must be less than 100 MHz.

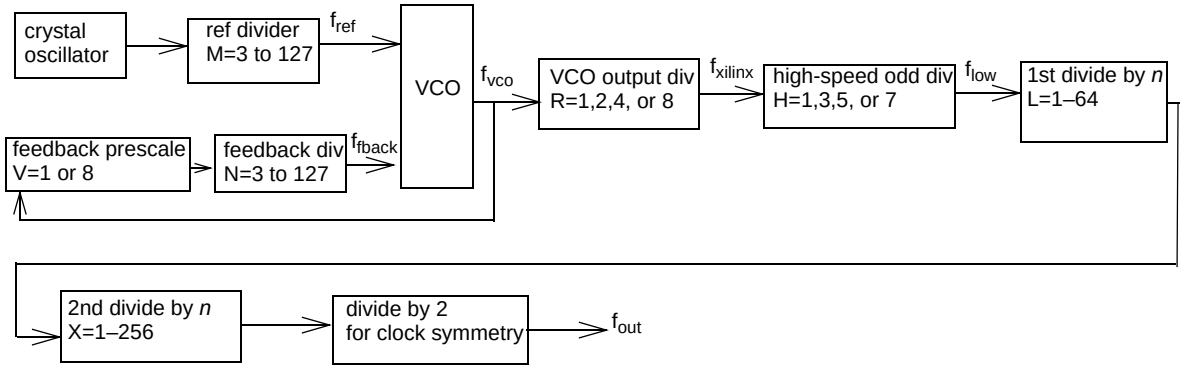
f_{low} The divide by *n* counter input frequency must be less than 30 MHz. If L and X are both set to 1, then frequencies to 100 MHz can be passed to the final divide-by-two block.

f_{out} This final divide by two assures a 50% output clock duty cycle.

The formula for calculating the output frequency is:

$f_{out} = (N * V * f_{xtal}) / (m * R * H * L * X * 2)$

Figure 1. Setting the Output Clock Frequency



For example, for an output clock of15 MHz using a PCI CD-20 (f_{xtal} = 10 MHz), the following numbers work, although they are not unique: N=60, V=1, M=10, R=2, H=1, L=1, X=1

For an output clock of 125 Hz in a PCI CD-20 (f_{xtal} = 10 MHz): N=50, V=1, M=10, R=8, H=5, L=50, X=100

The output clock has a wide range of values, but the frequency limitations at different stages limits the ultimate ability to exactly reach any specific frequency.

For example, the PLL reference frequency can be as low as 200 KHz, which would seem to allow steps of 200,000 in the VCO output. Unfortunately, since the maximum VCO output is 50 MHz and the *n*-programmable divider only goes to 127, the loop cannot lock unless V is set to 8, giving 1.6 MHz minimum steps. If, however, R is set to 8, we can get 200 KHz steps at f_{xilinx}. The lowest frequency in this case is at N=32 (6.4 MHz) to N=127 (25.4 MHz), in 200 KHz steps.

The following three library routines, documented in the EDT DMA Software Library, help compute the output clock frequency:

`edt_find_vco_frequency`
: Computes the phase-locked loop parameters necessary to match or approximate the supplied target frequency.

`edt_set_pll_clock`
: Sets the phase-locked loop circuit to the value computed by `edt_find_vco_frequency`. Includes debugging information.

`edt_set_out_clock`
: Sets the phase-locked loop circuit to the value computed by `edt_find_vco_frequency`. Does not include debugging information.

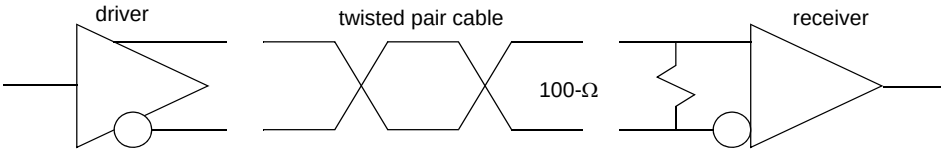
Hardware Interface Protocol

This section describes how to connect your device to a PCI CD/CDa interface, including the electrical characteristics of the signal, the signal descriptions, the timing specifications, and the connector pin-out.

Electrical Interface

The PCI CD/CDa uses differential data transmission to transmit data at very high rates over long distances through noisy environments. Differential transmission nullifies the effects of ground shifts and noise. These effects appear as common mode voltages on the transmission line and are rejected by the receiver. A typical balanced differential circuit is shown below.

Figure 2. Balanced Differential Circuit

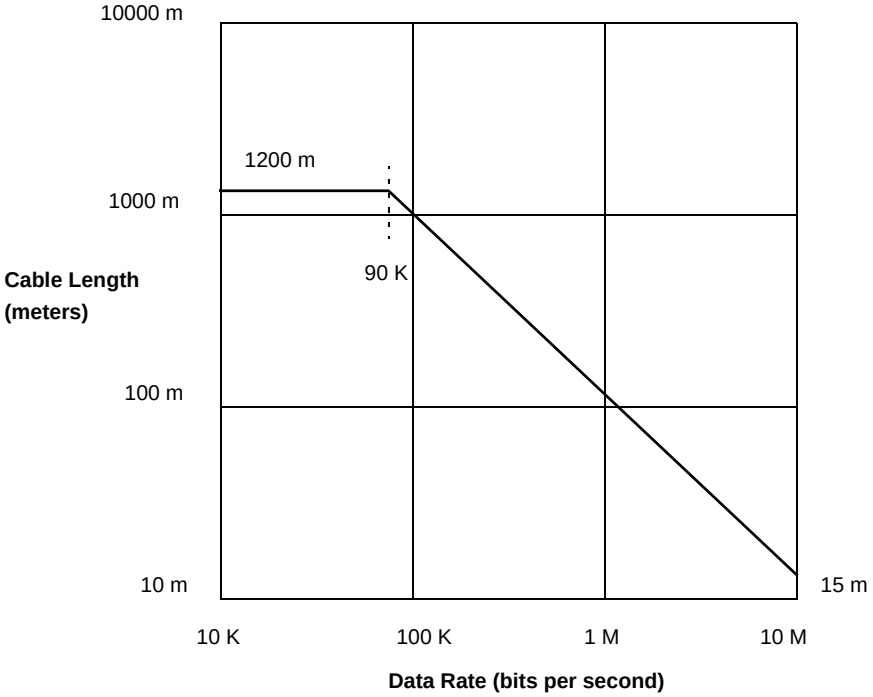


The interface is implemented with 32 signals (PCI CD) or 35 signals (PCI CDa), each implemented as a differential pair of wires.

RS-422

The PCI CD-20 and PCI CDa RS-422 DMA interface protocols use RS-422 signal levels. RS-422 is defined by the Electronic Industries Association to provide robust high-speed data transmission. It allows signaling rates up to 10 MHz for short cables of up to 40 feet (12 meters) and cables of up to 4000 feet (1219 meters) at 100 KHz, as shown in Figure 3:

Figure 3. RS-422 Data Signalling Rate and Cable Length



For further information, see the EIA RS-422-A standard, Electrical Characteristics of Balanced Voltage Digital Interface Circuits.

LVDS

The PCI CD-60 and PCI CDa LVDS DMA interface protocols use low-voltage differential signaling, a standard for faster signaling at lower power, compatible with IEEE 1596.3 SCI LVDS standard, and conforming to ANSI/TIA/EIA-644 LVDS standard.

For details, also see the LVDS Owner's Manual.

Signals

The hardware flow control protocol assumes that FIFO or memory buffers on both ends implement almost-full and almost-empty thresholds. Therefore, when a BNR (board not ready) or DNR (device not ready) signal is sent to the transmitting device, the receiver can still accommodate enough data to allow for cable delay and synchronization.

Table 1. Signals

Signal	I/O	Description
DAT(15:0)	I/O	Sixteen bidirectional data lines for DMA data.
STAT(3:0)	I	Four general-purpose control inputs. Any can be enabled to interrupt the PCI bus host.
FUNCT(3:0)	O	Four general-purpose program control outputs. Can be used to reset the user device or indicate DMA direction for bidirectional devices.
SENDT	O	Send Timing is a constant clock driven by the DMA interface that the user device can use (though it need not) to generate the receive timing. See Table 2 for timing specifications. The source for SENDT can be either: • the internal oscillator (the default when using `pcd_src.bit`); • the signal RXT when the PCI CD is using `pcd_looped.bit`, or the PCI CDa has set bit 1 (SELRXT) in the Interface Configuration Register; or • the PLL when the PCI CD is using `pcd_src.bit`, or the PCI CDa has set bit 7, PLLCLK, in the Funct Register.
RXT	I	Receive Timing is an input to the DMA interface. This is the clock used to synchronize input data and control signals. This signal can be equal to or less than the SENDT frequency of the board in use. It is best, although not required, that this signal is a continuous clock. Data clocked into the DMA interface must propagate through pipelining registers before it can be transferred into PCI bus memory. If the RXT clock stops, data is left in this pipe instead of being transferred to host memory.
TXT	O	Transmit Timing is an output from the DMA interface. TXT synchronizes the DMA output data and control signals. TXT is an inverted copy of SENDT, which provides maximum setup and hold on rising edge.
IDV	I	Input Data Valid is asserted by the device synchronous with RXT, to tell the DMA interface that data on the DAT(15:0) signals are valid and must be registered and transferred to the PCI bus memory. The DMA interface will accomplish this unless the BNR signal has been asserted for >32 IDV signals.
BNR	O	Bus Not Ready is asserted by the DMA interface synchronous with TXT when 32 bytes or fewer of data space remains for input data from the device. This warns you to stop the data transfer or prepare for overflow.
OUTPUT DISABLE	I	Disables the data outputs when more than one PCI CD board is connected to the same cable. In order to use this signal, you must set the Enable Output Control bit in the Stat Polarity Register. TTL-compatible. PCI CD only.
ODV	O	Output Data Valid is asserted by the DMA interface when it has placed valid output data on DAT(15:0). ODV is asserted synchronously with the TXT clock, and only if the DNR signal is not asserted.
DNR	I	Device Not Ready is asserted by the device synchronous with the RXT clock when the user device is about to run out of space for storing data from the DMA interface. The amount of overrun buffer required in the device varies according to the cable length. The PCI CD/CDa may produce four or more words of valid data after DNR is presented to the input pins.

Timing

The clock and data output timing is specified at the pins of the PCI CD/CDa connector.

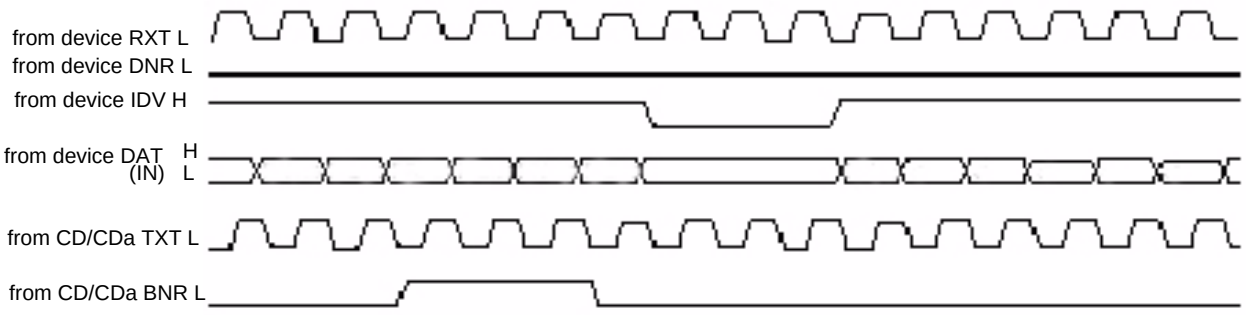
Table 2. Timing Specifications

	PCI CD-20	PCI CD-60	PCI CDa LVDS	PCI CDa RS-422
Clock frequency	10 MHz	30 MHz	40 MHz, or as programmed	10 MHz, or as programmed
Clock duty cycle	50% ±10 ns	50% ± 5 ns	50% ± 5 ns	50% ± 5 ns
Input minimum set-up time	20 ns	5 ns	5 ns	20 ns
Input minimum hold time	25 ns	6 ns	6 ns	25 ns
Output maximum propagation delay	20 ns	10 ns	10 ns	20 ns

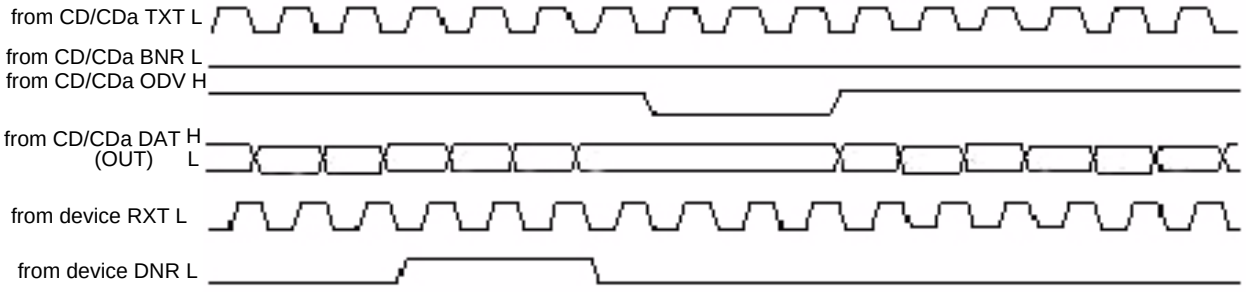
Figure 4 shows the PCI CD/CDa timing.

Figure 4. PCI CD/CDa Timing

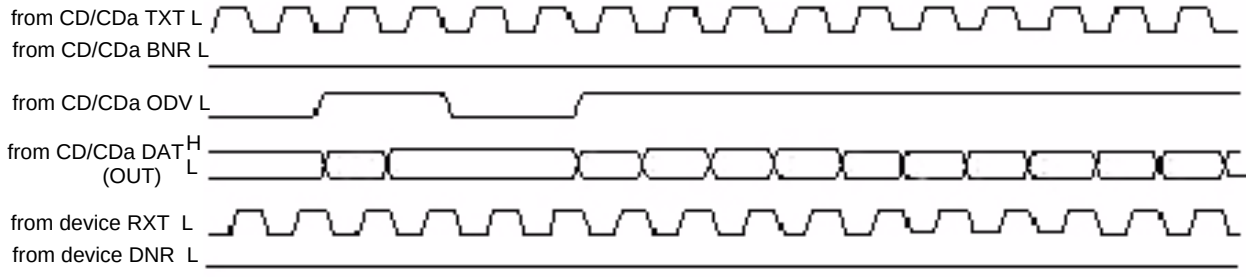
Typical DMA data read handshake — input FIFO gets almost full, then empties:



Typical DMA data write handshake — user device needs to hold data out:



Typical data output startup— user device is ready (DNR = false):



The number of ODV deassertions depends on the frequency of the TXT clock and the PCI Bus response of the host. To minimize or prevent ODV deassertions, align the memory buffer so that data is transferred from a 64-byte boundary. In that case, the first PCI Bus transfer is in burst mode.

Connector Pinouts

The following pinout diagrams describe the connection from the PCI CD/CDa board to the cable. The board uses a high-density 80-pin I/O connector (EDT part number 012-10026), with a straight-shielded backshell (EDT part number 013-10287) or right-angle backshell (EDT 013-00458).

NOTE Do not connect your own circuits to the unused pins, as they may be internally connected.

Table 3 applies to the PCI CD when loaded with either `pcd_src.bit` or `pcd_looped.bit`, or the PCI CDa when loaded with `pcda.bit`.

Table 3. PCI CD (pcd_src.bit, pcd_looped.bit), PCI CDa (pcda.bit) Connector Pinout

Pin	Signal	Pin	Signal
1	ground	41	ground
2	ground (spare on CDa)	42	ground (spare on CDa)
3	DAT4+	43	DAT0+
4	DAT4–	44	DAT0–
5	DAT5+	45	DAT1+
6	DAT5–	46	DAT1–
7	DAT6+	47	DAT2+
8	DAT6–	48	DAT2–
9	DAT7+	49	DAT3+
10	DAT7–	50	DAT3–
11	DAT12+	51	DAT8+
12	DAT12–	52	DAT8–
13	DAT13+	53	DAT9+
14	DAT13–	54	DAT9–
15	DAT14+	55	DAT10+
16	DAT14–	56	DAT10–
17	DAT15+	57	DAT11+
18	DAT15–	58	DAT11–
19	spare 0+	59	spare 0–
20	+5V	60	+5V
21	spare 1+	61	spare 1–
22	spare 2+	62	spare 2–
23	ground	63	ground
24	STAT0+	64	RXT+
25	STAT0–	65	RXT–
26	STAT1+	66	IDV+
27	STAT1–	67	IDV–
28	STAT2+	68	DNR+
29	STAT2–	69	DNR–
30	STAT3+	70	reserved+
31	STAT3–	71	reserved–
32	FUNCT0+	72	SENDT+
33	FUNCT0–	73	SENDT–
34	FUNCT1+	74	ODV+
35	FUNCT1–	75	ODV–
36	FUNCT2+	76	BNR+
37	FUNCT2–	77	BNR–
38	FUNCT3+	78	TXT+
39	FUNCT3–	79	TXT–
40	ground	80	ground

Registers

The PCI CD/CDa has two memory spaces: the memory-mapped registers and the configuration space. Expansion ROM and I/O space are not implemented. Applications can access the PCI CD/CDa registers through the DMA library routines especially `edt_reg_read()` and `edt_reg_write()`, using the symbolic names listed under "Access" for each register.

Configuration Space

The configuration space is a 64-byte portion of memory required to configure the PCI Local Bus and to handle errors. Its structure is specified by the PCI Local Bus specification. The structure as implemented for the PCI CD/CDa is as shown in Figure 5 and described below.

Figure 5. Configuration Space Addresses

Address	Bits 31 16	Bits 15 0
0x00	Device ID (CD-20=11, CD-60=13, CDa=44)	Vendor ID = 0x123D
0x04	Status *(see below)*	Command *(see below)*
0x08	Class Code = 0x088000	Revision ID = 0 *(will be updated)*
0x0C	BIST = 0x00 / Header Type= 0x00	Latency Timer *(set by OS)* / Cache Line Size *(set by OS)*
0x10	DMA Base Address Register *(set by OS)*	
0x14	UI Xilinx Memory-mapped I/O Base Address Register *(set by OS)*	
	not implemented	
0x3C	Max_Lat = 0x04 / Min_Gnt = 0x04	Interrupt Pin = 0x01 / Interrupt Line *(set by OS)*

Values for the status and command fields are shown in Tables 4 and 5. For complete descriptions of the bits in the status and command fields, see the *PCI Local Bus Specification*, available from:

PCI Special Interest Groupwww.pcisig.com

Table 4. Configuration Space Status Field Values

Bit	Name	Value	Bit	Name	Value
0–4	reserved	0	10	DEVSEL Timing	0
5	66 MHz Capable	1	11	Signaled Target Abort	implemented
6	UDF Supported	0	12	Received Target Abort	implemented
7	Fast Back-to-back Capable	0	13	Received Master Abort	implemented
8	Data Parity Error Detected	implemented	14	Signaled System Error	implemented
9	DEVSEL Timing	1	15	Detected Parity Error	implemented

Table 5. Configuration Space Command Field Values

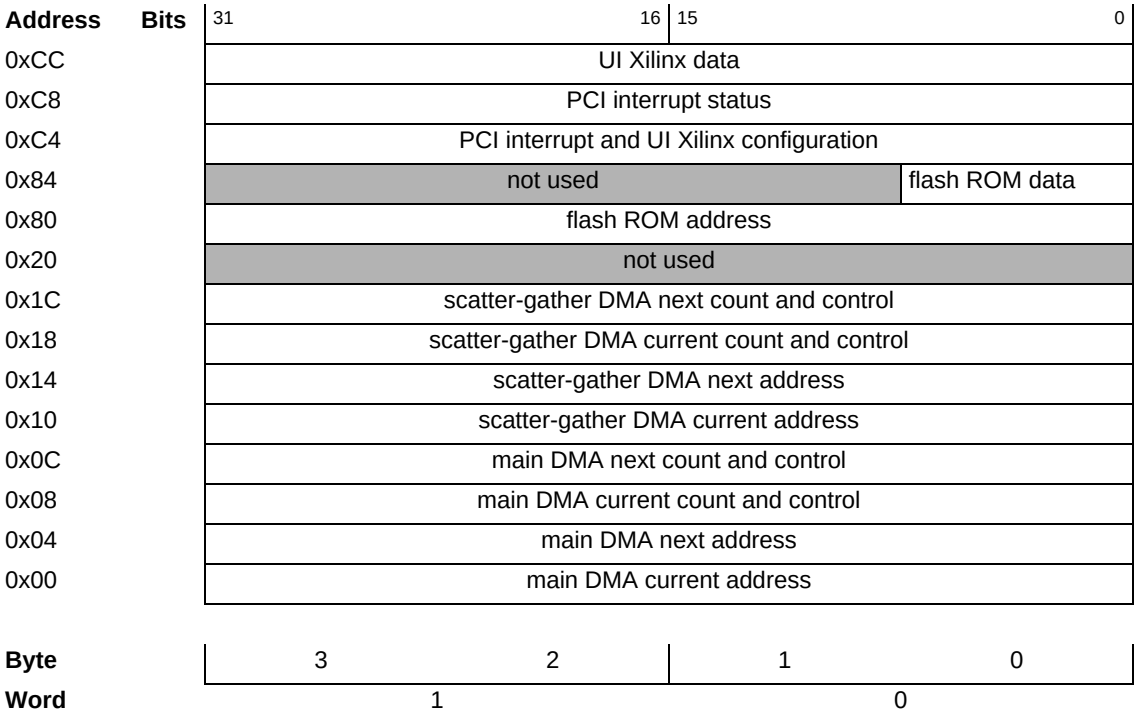
Bit	Name	Value	Bit	Name	Value
0	IO Space	0	6	Parity Error Response	implemented
1	Memory Space	implemented	7	Wait Cycle Control	0
2	Bus Master	implemented	8	SERR# Enable	implemented
3	Special Cycles	0	9	Fast Back-to-back Enable	implemented
4	Memory Write and Invalidate Enable	0	10–15	reserved	0
5	VGA Palette Snoop	0			

PCI Local Bus Addresses

Figure 6 describes the PCI CD/CDa interface registers in detail. The addresses listed in Figure 6 are offsets from the gate array boot ROM base addresses. This base address is initialized by the PCI Local Bus host operating system at boot time.

NOTE The addresses 0x80 and 0x84 are used by the *pciload* utility to update the gate array. User applications must not modify use these registers. Results of running *pciload* do not take effect until after the board has been turned off and then on again.

Figure 6. PCI Local Bus Addresses



Scatter-gather DMA

PCI Direct Memory Access (DMA) devices in Intel-based computers access memory using physical addresses. Because the operating system uses a memory manager to connect the user program to memory, memory pages that appear contiguous to the user program are actually scattered throughout physical memory. Because DMA accesses physical addresses, a DMA read operation must *gather* data from noncontiguous pages, and a write must *scatter* the data back to the appropriate pages. The PCI CD/CDa driver uses information from the operating system to accomplish this. The operating system passes the driver a list of the physical addresses for the user program memory pages. With this information, the driver builds a scatter-gather (SG) table, which the DMA device uses sequentially.

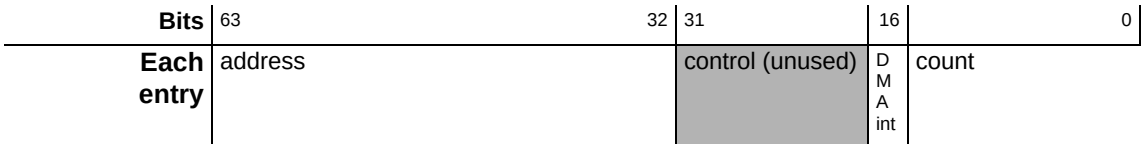
Most other PCI computers offer memory management for the PCI bus as well, so the operating system needs to pass only the address and count for DMA. The addresses appear contiguous to the PCI bus.

The scatter-gather DMA list is stored in memory. The scatter-gather DMA channel copies it as required into the main DMA registers. The format of the DMA list in memory is as follows (illustrated in Figure 7):

- Each page entry takes eight bytes. Therefore, the scatter-gather DMA count is always evenly divisible by eight.
- The first word consists of the 32-bit start address of a memory page.
- The most significant 16 bits of the second word contain control data.
- The least significant 16 bits of the second word contain the count.

Only bit 16 contains control information. When set to one, and when enabled by setting bit 28 of the Scatter-gather DMA Next Count and Control register, this bit causes the main DMA interrupt to be set when the marked page is complete.

Figure 7. Scatter-gather DMA List Format



All main DMA registers are read-only. Only the corresponding scatter-gather DMA registers must write to them. To initiate a DMA transfer, the driver performs the following general operations:

1. It sets up one or more scatter-gather DMA lists in host memory, using the format described above and illustrated in Figure 7.
2. It writes the address of the first entry in the list to the Scatter-gather DMA Next Address Register.
3. It writes the length of the scatter-gather DMA list to the Scatter-gather DMA Next Count and Control Register, setting the interrupts as you require. Setting bit 29 of this register to 1 starts the DMA.
4. If the DMA list is greater than one page, it loads the address of the first entry of the next page and its length, as described in steps 2 and 3, when bit 29 of theScatter-gather DMA Next Count and Control Register is asserted.

Main DMA Current Address Register

Size	32-bit
I/O	read-only
Address	0x00
Access	EDT_DMA_CUR_ADDR
Comment	This register is automatically copied from the main DMA next address register after main DMA completes.

Bit	Description
31–0	The address of the current DMA, or the last used address if no DMA is currently active.

Main DMA Next Address Register

Size	32-bit
I/O	read-only
Address	0x04
Access	EDT_DMA_NXT_ADDR
Comment	The scatter-gather DMA fills this register when required from the scatter-gather DMA list.

Bit	Description
31–0	Read the starting address of the next DMA.

Main DMA Current Count and Control Register

Size	32-bit
I/O	read-only
Address	0x08
Access	EDT_DMA_CUR_CNT
Comment	This register is automatically copied from the main DMA next count and control register after main DMA completes.

Bit	Description
31-16	Read-only versions of bits 31–16 of the Scatter-gather DMA current count and control register.
15–0	The number of words still to be transferred in the current DMA.

Main DMA Next Count and Control Register

Size	32-bit
I/O	read-only
Address	0x0C
Access	EDT_DMA_NXT_CNT
Comment	The scatter-gather DMA fills this register when required from the scatter-gather DMA list.

Bit	Description
31-16	Read-only versions of bits 31–16 of the Scatter-gather DMA next count and control register.
15–0	The number of words still to be transferred in the current DMA.

Scatter-gather DMA Current Address Register

Size	32-bit
I/O	read-only
Address	0x10
Access	EDT_SG_CUR_ADDR
Comment	This register is automatically copied from the scatter-gather DMA next address register when that register is valid and the current scatter-gather DMA completes.

Bit	Description
31–0	The address of the current DMA, or the last used address if no DMA is currently active.

Scatter-gather DMA Next Address Register

Size	32-bit
I/O	read-write
Address	0x14
Access	EDT_SG_NXT_ADDR
Comment	The driver software writes this register as described in step 2 of the list on page 21.

Bit	Description
31–0	The starting address of the next DMA.

Scatter-gather DMA Current Count and Control Register

Size	32-bit
I/O	read-only
Address	0x18
Access	EDT_SG_CUR_CNT
Comment	The driver software can read this register for debugging or to monitor DMA progress.

Bit	Description
31-16	Read-only versions of bits 31–16 of the Scatter-gather DMA next count and control register.
15–0	The number of words still to be transferred in the current DMA.

Scatter-gather DMA Next Count and Control Register

Size	32-bit
I/O	read-write
Address	0x1C
Access	EDT_SG_NXT_CNT
Comment	The driver software writes this register as described in step 3 in the list on page 21.

Bit	EDT_	Description
31	EN_RDY	Enable scatter-gather next empty interrupt. A value of 1 enables DMA_START (bit 29 of this register) to set DMA_INT (bit 12 of the Status register), thus causing an interrupt if the PCI_EN_INTR bit is set (bit 15 of the Main DMA Command and Configuration register). A value of 0 disables the DMA_START from causing an interrupt.
30	DMA_DONE	Read-only: a value of 0 indicates that a scatter-gather DMA transfer is currently in progress. A value of 1 indicates that the current scatter-gather DMA is complete.

Bit	EDT_	Description
29	DMA_START	Write a 1 to this bit to indicate that the values of this register and the SG DMA Next Address register are valid; this sets this bit to 0, indicating either that the copy is in progress, or that the device is waiting for the current DMA to complete. In either case, this register and the SG DMA Next Address register are not available for writing. Reading a value of 1 indicates that the SG DMA Next Count and SG DMA Next Address registers have been copied into the SG DMA Current Count and SG DMA Current Address registers and that the Next Count and Next Address registers are once more available for writing.
28	EN_MN_DONE	A value of 1 enables the main DMA page done interrupt (bit 18).
27	EN_SG_DONE	Enable scatter-gather DMA done interrupt. A value of 1 enables DMA_DONE (bit 30 of this register) to set DMA_INT (bit 12 of the Stat Register on page 30), thus causing an interrupt if the PCI_EN_INTR bit is set (bit 15 of the PCI Interrupt and UI Xilinx Configuration Register). A value of 0 disables the DMA_DONE from causing an interrupt.
26	DMA_ABORT	A value of 1 stops the DMA transfer in progress and cancels the next one, clearing bits 29 and 30. Always 0 when read.
25	DMA_MEM_RD	A value of 1 specifies a read operation; 0 specifies write.
24	BURST_EN	A value of 0 means bytes are written to memory as soon as they are received. A value of 1 means bytes are saved to write the most efficient number at once.
23	MN_DMA_DONE	Read only: a value of 1 indicates that the main DMA is not active.
22	MN_NXT_EMP	Read only: a value of 1 indicates that the main DMA next address and next count registers are empty.
21–19		Reserved for EDT internal use.
18	PG_INT	Read-only: a value of 1 indicates that the page interrupt is set (enabled by bit 28 of this register), and that the main DMA has completed transferring a page for which bit 16 (the page interrupt bit) was set in the scatter-gather DMA list (see Figure 7). If the PCI interrupt is enabled (bit 15 of the PCI Interrupt and UI Xilinx Configuration Register), this bit causes a PCI interrupt. Clear this bit by disabling the page done interrupt (bit 28 of this register).
17	CURPG_INT	Read-only: a value of 1 indicates that bit 16, the page interrupt bit, was set in the scatter-gather DMA list entry for the current main DMA page.
16	NXTPG_INT	Read-only: a value of 1 indicates that bit 16, the page interrupt bit, was set in the scatter-gather DMA list entry for the next main DMA page.
15–0		The number of bytes in the next scatter-gather DMA list.

PLL Programming Register

Size	8-bit
I/O	read-write
Address	0x20
Access	EDT_SS_PLL_CTL
Comment	**PCI CDa only.** The program `set_ss_vco` uses this register to program the serial interface of the four PLLs.

Bit	Name	Description
7	PLL_SCLK	Connected to all four PLL serial clock inputs.
6	PLL_DATA	Connected to all four PLL serial data inputs.
5–4		not used
3–0	PLL_STROBE	Connected to the strobe inputs of PLL 3–0, respectively.

Flash ROM Address Register

Size	32-bit
I/O	read-write
Address	0x80
Access	EDT_FLASHROM_ADDR
Comment	Use this register and the Flash ROM Data Register (below) to update the program the PCI Xilinx.

Bit	Description
31–25	Reserved for EDT internal use.
24	A value of 1 causes the data in the flash ROM data register to be written to the address specified by bits 0 through 23. A value of 0 reads the data.
23–0	Address of location in flash ROM that the next read or write will access.

Flash ROM Data Register

Size	32-bit
I/O	read-write
Address	0x84
Access	EDT_FLASHROM_DATA
Comment	Use this register and the Flash ROM Address Register (above) to update the program the PCI Xilinx.

Bit	Description
31–9	not used

Bit	Description
8	A read-only bit indicating the position of the jumper that enables access to the protected area of the ROM that contains the executable program. A value of 1 indicates that the board can load a new program.
7–0	The new program to load into flash ROM with a write operation (specified by setting bit A24 in the flash ROM address register), or the data that was read (specified by clearing bit A24 in the flash ROM address register).

PCI Interrupt and UI Xilinx Configuration Register

Size 32-bit

I/O read-write

Address 0xC4

Access EDT_REMOTE_OFFSET

Bit	EDT_	Description
31–22		not used
21	RMT_STATE	UI Xilinx INIT pin state. This bit is read-only.
20	RMT_DONE	UI Xilinx DONE pin.
19	RMT_PROG	UI Xilinx PROG pin.
18	RMT_INIT	UI Xilinx INIT pin.
17	EN_CCLK	Enable one configuration clock cycle to UI Xilinx.
16	RMT_DATA	UI Xilinx program data
15	PCI_EN_INTR	Enable PCI interrupt.
14	RMT_EN_INTR	Enable UI Xilinx interrupt.
13–9		not used
8	RFIFO_ENB	After the UI Xilinx has been programmed to your satisfaction: 1. Toggle bit D3 of the UI Xilinx command register. 2. Set this bit to enable the burst data FIFO.
7		not used
6–0	RMT_ADDR	128-byte address of UI Xilinx register

To program the UI Xilinx:

1. Clear the PROG and INIT pins.
2. Wait for DONE (bit 20) to be clear.
3. Set the PROG and INIT pins.
4. Loop until INIT state (bit 21) is set.
5. Wait four μs.
6. Write programming data, one bit at a time, to bit 16 with bit 17 set.
7. After all data is written, continue writing ones to bit 16 until DONE (bit 20) is set.

The programming has failed if it has not completed after 32 clock cycles.

PCI Interrupt Status Register

Size	32-bit
I/O	read-only
Address	0xC8
Access	EDT_DMA_STATUS
Comment	The driver uses this register initially to determine the source of a PCI interrupt.

Bit	PCD_	Description
16–31		not used
15	PCI_INTR	PCI interrupt. When asserted, the PCI CD/CDa is asserting an interrupt on the PCI bus.
14		not used
13	RMT_INTR	UI Xilinx interrupt. When asserted, the UI Xilinx interrupt is set. If bits 14 and 15 of the the PCI interrupt and UI Xilinx configuration register are aserted, the UI Xilinx causes a PCI interrupt.
12	DMA_INTR	End of DMA interrupt. Asserted when at least one of the DMA interrupts is asserted in the scatter-gather DMA next count and control register. Causes a PCI interrupt if bit 15 of the PCI interrupt and UI Xilinx configuration register is enabled.
11–0		not used

UI Xilinx Data Register

Size	32-bit
I/O	read-write
Address	0xCC
Access	EDT_REMOTE_DATA

Bit	Description
31–8	not used
7–0	Read or write data for UI Xilinx, using the address specified in EDT_RMT_ADDR (bits 0–6) of the PCI Interrupt and UI Xilinx Configuration Register.

UI Xilinx Registers

The UI Xilinx is a field-programmable gate array used to implement the PCI CD/CDa interface or to test the board. The UI Xilinx is programmed serially, using the PCI Interrupt and UI Xilinx Configuration Register on page 26.

NOTE The following registers are defined to control the interface and reside in the UI Xilinx. In order to access those registers, the PCI CD/CDa requires that the UI Xilinx be loaded with a program that defines them. If the Xilinx is not loaded, or loaded with a different program, these registers are inaccessible.

The Xilinx IC is programmed when the PCI CD/CDa driver is loaded, or by the application program. If you have received a customized application program for your PCI CD/CDa from Engineering Design Team, some or all of these registers may not be defined. Consult the documentation that came with your customized program instead.

Command Register

Size 8-bit

I/O read-write

Address 0x00

Access PCD_CMD

Bit	PCD_	Description
4–7	STAT_INT_EN	A value of one enables the corresponding STAT bit to cause an interrupt when it is asserted.
3	ENABLE	Set to one to enable the PCI CD/CDa interface. This bit is set after the direction is chosen and typically after the first DMA buffer is ready. To reset direction or flags, toggle this bit. To flush the DMA FIFOs, clear then set this bit.
2	DATA_INV	If this bit is set, the PCI CD/CDa inverts the data.
1	FORCEBNR	A value of 1indicates that the board is not ready.
0	SELRX	A value of 1 indicates that data is coming in to the PCI CD/CDa. A value of 0 indicates that data is going out from the PCI CD/CDa.

Data Path Status Register

Size 8-bit

I/O read-only

Address 0x01

Access PCD_DATA_PATH_STAT

Bit	PCD_	Description
7	IDV	Reflects IDV state. ***(PCI CD only — spare on PCI CDa)***
6	INFFAFULL	If set, input FIFO is almost full.
4–5	INFFULL	If set, input FIFO is full.
3	OVERFLOW	This bit is asserted when the input FIFO is full and the IDV signal is high. Reset this bit with the ENABLE bit in the Command Register on page 28.
2	UNDERFLOW	If the DNR signal is low and the ODV signal goes low because the output FIFO runs out of data, then this bit is asserted and remains so throughout the data transfer. Reset this bit with the ENABLE bit in the command register.
1	IF_NOT_EMP	If this bit is set, the input FIFO is not empty.
0	OF_NOT_EMP	If this bit is set, the output FIFO is not empty.

Funct Register

Size 8-bit

I/O read-write

Address 0x02

Access PCD_FUNCT

Bit	PCD_	Description
7	PLLCLK	Set to enable the PLL circuit; see the PLL Divider Register for the use of this bit. Applies to the PCI CDa, and to the PCI CD running `pcd_src.bit` only.
6	SELAV	PCI CD only: used to program the PLL and select the clock. PCI CDa: not used
5	FREQ7	PCI CD only: used to program the PLL and select the clock. PCI CDa: not used
4	FREQ6	PCI CD only: used to program the PLL and select the clock. PCI CDa: not used
0–3	FUNCT	Sets the state of the user-definable FUNCT outputs.

Stat Register

Size 8-bit

I/O read-only

Address 0x03

Access PCD_STAT

Bit	PCD_	Description
7–4	STAT_INT	Interrupt bits for the status bits. If the following conditions are both true, then the corresponding bit of these four can be asserted to cause a PCI Bus interrupt: • The device interrupt is enabled using the RMT_EN_INTR bit in the PCI Interrupt and UI Xilinx Configuration Register. • The corresponding bit is asserted in the Command Register (one of bits 7–4, named STAT_INT_EN). The PCI Bus interrupt is then caused when the corresponding STAT signal is asserted according to the polarity specified in the Stat Polarity Register. To reset the interrupt, disable and re-enable the appropriate STAT_INT_EN bit in the Command Register.
3–0	STAT	The state of user-definable STAT input signals as last sampled by the RXT clock signal.

Stat Polarity Register

Size 8-bit

I/O read-write

Address 0x04

Access PCD_STAT_POLARITY

Bit	PCD_	Description
7–6		not used
5	ENA_OUT_CTRL	When set, enables the OUTPUT DISABLE signal on pin 22. (PCI CDa — not used)
4	STAT_INT_ENA	Provides global enable or disable for all interrupt bits in Stat Register on page 30, allowing the driver to disable and re-enable them in one operation, without altering the state of the Stat register. This bit is used mainly by the driver to disable the Stat interrupts to determine which other interrupts are pending. A value of 1 enables the interrupts.
3–0	POLARITY	A value of 0 indicates that a change from 0 to 1 from one clock cycle to the next causes an interrupt in bits 7–4 of the Stat Register on page 30, if the corresponding STAT_INT_EN bit is also enabled in the Command Register on page 28. A value of 1 causes the same event when the STAT_INT bit changes from 1 to 0 from one clock cycle to the next.

Direction Control Registers

Size	8-bit
I/O	read-write
Address	0x06, 0x07
Access	PCD_DIRA, PCD_DIRB
Comment	PCI CD only — not used for PCI CDa The direction control registers determine whether the physical drivers or receivers on the PCI CD interface are inputs or outputs. The PCI CDa's inputs are always enabled. To enable outputs, use bit 1 (SELRXT) of the Interface Configuration Register on page 33.

Each pin on the PCI CD can be programmed as either an input or an output. Pins are normally configured for inputs or outputs as documented in the connector pinout shown on page Table 3 on page 17. The PCI CD driver modifies the data bits appropriately for a *read()* or *write()* system call. Setting the same pins to serve as both input and output is useful for testing. Setting pins to serve as neither input nor output is not useful.

Direction Control Register A controls the data direction. Direction Control Register B controls the direction of the control bit.

For example, to implement the interface documented in Table 3 for a *write()* operation, set the A register (address 0x06) to 0x0F, and set the B register to 0xCC. For a *read()*, set A to 0xF0 and B to 0xCC.

Register	Bit	Description
B	15	Pins 22 and 62 are outputs when this bit is low.
	14	Pins 19, 59, 21, and 61 are outputs when this bit is low.
	13	Pins 32–39 are outputs when this bit is low.
	12	Pins 72–79 are outputs when this bit is low.
	11	Pins 24–31 are outputs when this bit is low.
	10	Pins 64–71 are outputs when this bit is low.
	9	Pins 24–31 are inputs when this bit is low.
	8	Pins 64–71 are inputs when this bit is low.
A	7	Pins 11–18 are outputs when this bit is low.
	6	Pins 51–58 are outputs when this bit is low.
	5	Pins 3–10 are outputs when this bit is low.
	4	Pins 43–50 are outputs when this bit is low.
	3	Pins 11–18 are inputs when this bit is low.
	2	Pins 51–58 are inputs when this bit is low.
	1	Pins 3–10 are inputs when this bit is low.
	0	Pins 43–50 are inputs when this bit is low.

Table 6. Direction Control Register A and B

Programmed I/O Low Register

Size	8-bit
I/O	read-write
Address	0x08
Access	PCD_PIO_OUTLOW

Bit	Description
7–0	Outputs data on the low eight bits of the 16-bit word. First write the low eight bits you wish to output to this register, then write the high eight bits to Programmed I/O High Register.

Programmed I/O High Register

Size	8-bit
I/O	read-write
Address	0x09
Access	PCD_PIO_OUTHI

Bit	Description
7–0	Outputs data on the high eight bits of the 16-bit word. First write the low eight bits you wish to output to the Programmed I/O Low Register, then write the high eight bits to this register. When the byte is output, an ODV signal (Output Data Valid) is also output for one TXT clock cycle.

Interface Configuration Register

Size 8-bit

I/O read-write

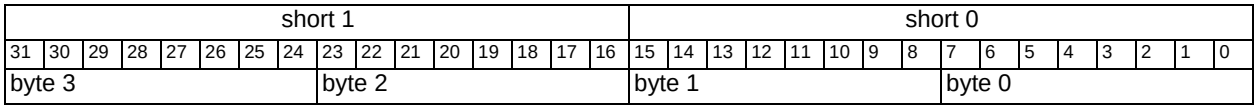
Address 0x0F

Access PCD_CONFIG

Bit	PCD_	Description
7	SETIDV	Set input data valid (used for debugging).
6	PIOEN	Enables programmed I/O. A value of 1 translates DMA channel buffers and enables the Programmed I/O Low Register and the Programmed I/O High Register. Write the desired 16-bit word, the low eight bits first to the Programmed I/O Low Register, and then the high eight bits to the Programmed I/O High Register. When the Programmed I/O High Register is written to, the firmware generates an ODV pulse in mid-clock, to enable the device to latch the data.
5	SETDNR	Set this bit to stop transfer to the device, as if the device had set DNR.
4	DED	Disable output delay. If set, may cause ODV transitioning on DMA start and underflows.
3	SHORTSWAP	Set to 1 if the host computer writes the first 16-bit word on bits 16–31 of the PCI data bus (bigendian format) instead of bits 0–15 as defined in the PCI Bus specification. See Figure 8 for the details of data word structure.
2	SETAV64	PCI CDa only: For debugging. Set to tell PCI Xilinx that data is always available. PCI CD: not used
1	SELRXT	PCI CDa only: Set to select RXT as the source for TXT and SENDT. (See Table 1 on page 14 for descriptions of these signals.) PCI CD: not used
0	BYTESWAP	A value of 1 swaps the order of bytes in a 16-bit word of data coming in from the data source. See Figure 8 for the details of data word structure.

Figure 8 shows the structure of a 32-bit data word, with no swapping in effect. With SHORTSWAP set, short 0 appears before short 1. With BYTESWAP set, byte 2 appears before byte 3, and byte 0 before byte 1. With both set, byte 0 appears first, followed by byte 1, byte 2, and finally byte 3.

Figure 8. Data Word Structure Without Swapping



PCI CDa Registers

The following four registers apply to the PCI CDa only.

PLL Programming Register

Size	8-bit
I/O	read-write
Address	0x20
Access	EDT_SS_PLL_CTL
Comment	The program `set_ss_vco` uses this register to program the serial interface of the PLL.

Bit	Name	Description
7	PLL_SCLK	Connected to the PLL serial clock input.
6	PLL_DATA	Connected to the PLL serial data input.
5–4		not used
3–0	PLL_STROBE	Connected to the strobe inputs of the PLL .

PLL Divider Register

Size	16-bit
I/O	read-only
Address	0x24, 0x25
Access	EDT_SS_PLL0_CLK,
Comment	The program `set_ss_vco` sets this register, which is a post-scalar divider used to achieve lower frequencies than those to which you can program the PLL directly. After this division, the clocks are divided by two to even the duty cycle. To enable the PLL circuit, set bit 7 in the Funct Register.

Bit	Description
15–0	Programmable post-scalar divider to set PLL frequency.

Output Data Valid Delay Register

Size	8-bit
I/O	read-write
Address	0x28
Access	ODV_DELAY

Bit	Description
7–0	Set the number of 16-bit words by which to delay output. The specified number of outgoing words accumulate in the FIFO, reducing or eliminating ODV transitioning on startup.

LED Control Register

Size 8-bit

I/O read-write

Address 0x30

Access LED_CTL

Bit	Name	Description
7–3		not used
2	LED2	Set to turn on LED when both LED0 and LED1 are clear.
1–0	LED1, LED0	Set to specify the signal to drive the LED:

LED1	***LED0***	***Source***
0	0	LED signal (bit 2 must also be set)
0	1	IDV
1	0	DNR
1	1	RXT