SSD16IO

16-channel Synchronous Serial I/O

for use with PCI SS/GS/CDa Main Boards

May 22, 2007
008-02663-01

The information in this document is subject to change without notice and does not represent a commitment on the part of Engineering Design Team, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement.

Engineering Design Team, Inc. ("EDT"), makes no warranties, express or implied, including without limitation the implied warranties of merchantibility and fitness for a particular purpose, regarding the software described in this document ("the software"). EDT does not warrant, guarantee, or make any representations regarding the use or the results of the use of the software in terms of its correctness, accuracy, reliability, currentness, or otherwise. The entire risk as to the results and performance of the software is assumed by you. The exclusion of implied warranties is not permitted by some jurisdictions. The above exclusion may not apply to you.

In no event will EDT, its directors, officers, employees, or agents be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising out of the use or inability to use the software even if EDT has been advised of the possibility of such damages. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. EDT's liability to you for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, tort [including negligence], product liability or otherwise), will be limited to $50 (fifty U.S. dollars).

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, without the express written agreement of Engineering Design Team, Inc.

Copyright © Engineering Design Team, Inc. 1997–2007. All rights reserved.

EDT and Engineering Design Team are trademarks of Engineering Design Team, Inc.

Xilinx is a registered trademark of Xilinx, Inc.

Contents

About the SSD16IO 16-channel Synchronous Serial I/O Board

The SSD16IO 16-channel Synchronous Serial I/O mezzanine board configuration enables the PCI SS/GS or PCICDa main boards to transfer 16 channels of synchronous serial input/output between an external device and the PCI Bus host computer at speeds of up to 50 megabits per second on each channel (or up to 70 megabits with the fast firmware, if you forego certain functionality). To do so, the PCI SS/GS main board requires an RS-422, LVDS, or ECL mezzanine board; the PCI CDa comes in either LVDS or RS-422 versions and requires no mezzanine board. Both main boards require you to load the PCI Xilinx with the 16-channel bitfile approrpiate for the board, and the user interface Xilinx with the appropriate SSD16IO bitfile.

The external device produces synchronous serial data, which then passes from the external device through the connector to the user interface Xilinx on the main board. The SSD16IO firmware running on that Xilinx constructs 32-bit data words, which it then passes to the PCI Xilinx and from there out the PCI bus to the host computer.

Each channel consists of one clock signal and one data signal, without handshaking. Channels can be enabled and configured individually for input or output, and can be configured to construct data words in various ways:

- Data can be latched on the rising or falling edge of the clock signal.
- Data words can be constructed either least or most significant bit first, or you can swap byte or word order.
- You can select a channel whose clock signal can be used as the output clock for data output from 14 other channels.
- You can select one or more channels that will output a clock signal only when valid data is also output. (Or, if you do not need this functionality, you can use the fast 70-megabit-per-second firmware.)

This document describes the modifications made to the user interface Xilinx on the main board to send and receive 16 synchronous serial channels.

The SSD16IO firmware was developed using the Xilinx Project Navigator. If you wish to develop your own firmware, the VHDL source is available and the project set up for you to start. Contact EDT for details.

Related Manuals

Detailed documentation on EDT's C software library routines, helpful for writing your applications, is available on EDT's website in either HTML or PDF form. The *PCI SS/GS Main Board User's Guide* is available in PDF form.

Manual	URL
EDT DMA Software Library (HTML)	www.edt.com/api
EDT DMA Software Library (PDF)	www.edt.com/manuals/misc/api.pdf
PCI SS/GS Main Board User's Guide	www.edt.com/manuals/PCD/pciss_gs.pdf
PCI CD/CDa User's Guide	www.edt.com/manuals/PCD/pcicd.pdf

About the DMA Interface

The SSD16IO 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.

When data comes in from the user device, the UI FPGA sends it to input and output FIFO buffers, which smooth data transfer between the user device and the PCI Bus, as well as accommodating data during the transition from one DMA to the next. Host DMA transfers are queued in hardware, minimizing the amount of FIFO required.

To ensure maximum throughput, EDT's DMA library, the DMA driver, and the FPGA configuration files all support pipelining.

- The library routines as well as the driver preallocate kernel resources for DMA (for example, memory), rather than waiting for an application to request a DMA transfer (typically with an EDT library routine call such as `edt_read`, `edt_write`, or `edt_start_buffers`). When one DMA transfer ends, the resources remain allocated and available for use by the next DMA transfer.
- A portion of host memory can be configured as *ring buffers:* a set of buffers preallocated for DMA and reused in round-robin fashion.
- The FPGA fabric provides two sets of DMA registers, so that when one DMA transfer starts, the registers required for the next are already prepared, thus enabling zero-latency transitions between DMA transfers.

You can set the number of ring buffers and their size with the EDT DMA library call `edt_configure_ring_buffers`. Configure the ring buffers according to your application's DMA requirements — a useful configuration is often four one-megabyte ring buffers. Four ring buffers allow one to be used for the current DMA transfer, one for pending DMA, and one for the application, with one extra to ensure zero-latency transitions.

You can fine-tune your application to the latency requirements of a particular system by increasing or decreasing the size of the ring buffers; slow systems may need larger ring buffers, while fast systems may achieve better performance with more smaller ones.

Some host systems may restrict your ability to allocate particularly large ring buffers, or particularly large numbers of them. For example, some Windows systems limit DMA resources to a maximum of 64 MB in all. If you suspect this might be a problem in your system, be sure that your code checks for error returns after calling `edt_configure_ring_buffers` and before calling `edt_start_buffers`.

Installation

In the instructions below, substitute appropriate values for the placeholders in *italics*.

To install the SSD16IO:

1. Install the Pcd driver software as specified on the software disk sleeve.
2. Install the mezzanine board on the main board, if necessary, and install the board assembly in the host computer as specified by the computer manufacturer.
3. To configure the board, at the command prompt, enter:

   `initpcd -u` *`unit number`* `-f` *`configuration file`*

   For example, to configure board 0 with the sample configuration file provided, enter:

   `initpcd -u 0 -f pcd_config/ssd16io.cfg`

About the Software and Firmware

The SSD16IO ships with the following SSD16IO-specific software:

`ssd16io.bit` VHDL configuration file for the user interface Xilinx on PCI SS/GS boards with LVDS or RS-422 mezzanine boards, or on PCI CDa main boards (LVDS or RS-422). Enables transfer of up to 50 megabits per second on each channel. Using the Output Clock Gate Register, you can select one or more channels that will output a clock signal only when valid data is also output.

`eclssd16.bit` VHDL configuration file for the user interface Xilinx on PCI SS/GS main boards with ECL mezzanine boards.

`ssd16io_fast.bit` VHDL configuration file for the user interface Xilinx on PCI SS/GS boards with LVDS or RS-422 mezzanine boards, or on PCI CDa main boards (LVDS or RS-422). Enables transfer of up to 70 megabits per second on each channel. Does not allow you to select one or more channels that will output a clock signal only when valid data is also output.

Compatible 16-channel bitfiles must be loaded in the PCI Xilinx on the main boards. These are:

`pciss16.bit` The VHDL bitfile for the PCI Xilinx on PCI SS boards.

`pcigs16.bit` The VHDL bitfile for the PCI Xilinx on PCI GS boards.

`cda16.bit` The VHDL bitfile for the PCI Xilinx on PCI CDa boards.

Sample software initiallization files are editable text files that you can customize for your own applications. Sample software initiallization files for all board configurations are in the `pcd_config` subdirectory of the distribution directory, including:

`ssd16io.cfg` Sample configuration file to configure the SSD16IO for operation with `ssd16io.bit`.

`ssd16io_fast.cfg`
: Sample configuration file to configure the SSD16IO for operation with `ssd16io_fast.bit`.

The file names you see in the EDT distribution do not match the file names given above because PCI Bus slots come in two varieties: those supplying 3 V power, and those supplying 5 V power. Different firmware is required for the two kinds of slots, but the correct firmware file is chosen automatically when you run `pciload` or any other EDT-supplied firmware loading utility.

For example, you may see files named `cda16_3v.bit` and `cda16_5v.bit`, but the correct argument to supply to load the firmware is `cda16.bit`.

In some cases, you may also see additional firmware files incorporating changes required for various board revisions, or files with the same name in different subdirectories. You need not be concerned with any of these variations of name or path, however. In all cases, the names given above are the correct arguments to supply to the firmware-loading utilities.

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 SSD16IO. 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.

FPGA Configuration Files

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.

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 SSD16IO are listed at the beginning of this section. Instructions for loading them are provided in Configuring the SSD16IO.

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 SSD16IO are listed at the beginning of this section. Instructions for their use are provided in Configuring the SSD16IO.

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:`). For example:

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

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 SSD16IO to specific frequencies used by the UI FPGA for input and output.

Utility Files

`initpcd`	A utility for initializing and configuring the SSD16IO.
`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 SSD16IO

For the SSD16IO 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 SSD16IO

The utility `initpcd` loads the UI FPGA configuration files, programs the registers, sets the clocks (if necessary), and gets the SSD16IO 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 SSD16IO, your application can concern itself solely with performing DMA and other application-specific operations; it will therefore omit SSD16IO-specific operations and be portable to other EDT boards that peform DMA.

To configure the SSD16IO, enter:

```
initpcd -u unit_number -f 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 ssd16io.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

The loopback test determines the board configuration, loads the appropriate bitfile, generates test data and tests the board and its components with no external device connected. Test files are included — see About the Software and Firmware on page 3 for the complete list.

NOTE The loopback test overwrites the bitfile in the user interface Xilinx. Before you can use the board again, you'll need to reconfigure it after the test has completed.

To perform this test:

1. Leave the board in the host computer with the mezzanine board (if any) attached, but disconnect any external device and its cable.
2. In a command window, enter:

   `sslooptest -u` *`unit number`*

   The test outcome varies depending on the main board and mezzanine board installed. Errors are redirected to the file `sslooptest.err` in the current directory; if no such file exists, the test completed without errors.

   Loopback test output for a functional board contains lines such as:

```
Total errs=0 bufs=4000; Channel errs(NNNNxxxxxxxxxxxx) bufs(YYYYxxxxxxxxxxxx)
```

   `Total errs` shows the error count so far. `bufs` shows the number of buffers in use. The sixteen characters after `Channel errs` show the absence (`N`) or presence (`Y`) of a data error in a specific channel (0–15); an `x` indicates a channel is not in use.

   Similarly, a `Y` after `Channel... bufs` shows a buffer in use; an `x`, that the corresponding channel is not in use. An `N` indicates that DMA is not occurring in a specific channel.
3. After the test has completed, reconfigure the board using `initpcd` (or your own application) to disable loopback.
4. Reconnect the board to the external device.

Registers

The following registers are implemented but not used:

- Data Path (0x01)
- Function (0x02)
- Status (0x03)
- Status Polarity (0x04)
- Direction Control (0x06 and 0x07)
- Differential Direction (0x22)
- PLL 0 Divider (0x24 and 0x25) is not used for PCI SS/GS boards;
  PLL 1 Divider (0x26 and 0x27) is not used for PCI CDa boards.
- PLL 2 Divider (0x28 and 0x29)
- PLL 3 Divider (0x2A and 0x2B)
- Bit Error Control (0x58)

Command Register

Size	8-bit
I/O	read-write
Address	0x00
Access	PCD_CMD

Bit	Name	Description
7	WORDFLUSH	When set, enables transfer of one word at a time. When clear, enables burst mode.
6–4		not used
3	CMD_EN	Set this bit, and enable the required channels in the Channel Enable Register, for DMA to occur. When clear, resets all channels, flushes the FIFOs, and clears all under- and overflow bits.
2–0		not used

Configuration Register

Size 8-bit

I/O read-write

Address 0x0F

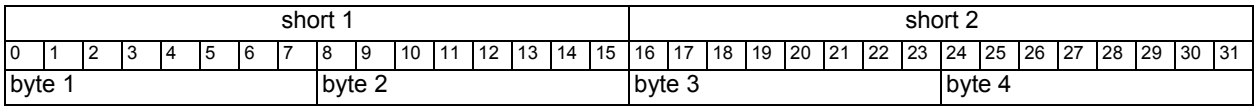
Access PCD_CONFIG

Bit	Name	Description
7–4		not used
3	SSWAP	Swaps the order of the two 16-bit short words in one 32-bit data word, so that *short 2* is transferred before *short 1*. Does not change the order of the bits within each short. See Figure 1 for the details of data word structure.
2–1		not used
0	BSWAP	Swaps the order of bytes 1 and 2, and also bytes 3 and 4, in a 32-bit data word, so that the bytes are transferred in the order 2, 1, 4, 3. Does not change the order of the bits within each byte. See Figure 1 for the details of data word structure.

NOTE The Least Significant Bit First Register can also affect the order in which data is transferred.

Figure 1 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 1. Data Word Structure



Channel Enable Register

Size 16-bit

I/O read-write

Address 0x10 and 0x11

Access SSD16_CHEN

Bit	Name	Description
15–0	CH_ENABLE	A value of one in a bit enables the corresponding channel for DMA. Channels correspond to register bits as shown in Table 1.

Table 1. How Channels Correspond to Bits in Registers

Register	register with low address								register with high address							
Bit number	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
Channel number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Channel Direction Register

Size	16-bit
I/O	read-write
Address	0x12 and 0x13
Access	SSD16_CHDIR

Bit	Name	Description
15–0	CH_DIR	A value of zero in a bit enables input for the corresponding DMA channel; a value of one enables output. Channels correspond to register bits as shown in Table 1.

Channel Edge Register

Size	16-bit
I/O	read-write
Address	0x14 and 0x15
Access	SSD16_CHEDGE

Bit	Name	Description
15–0	EDGE	For input channels, a value of one in a bit indicates that the corresponding channel latches data on the rising edge of its clock. a value of 0 indicates that it latches data on the falling edge. For output channels, a value of one indicates that a new bit is output on the rising edge, and a value of zero that it's output on the falling edge. Channels correspond to register bits as shown in Table 1.

Least Significant Bit First Register

Size	16-bit
I/O	read-write
Address	0x16 and 0x17
Access	SSD16_LSB

Bit	Name	Description
15–0	LSB_FIRST	When set for a channel, the least significant bit of each 8-bit data byte is the first bit, and the most significant bit is the last. When clear for a channel, the most significant bit of the byte is the first bit.

NOTE Byte Swap and Short Swap in the Configuration Register can also affect the order of bits in a 32-bit word. A combination of these bits allow the data to be formatted correctly for your host computer and application.

Underflow Register

Size 16-bit

I/O read only

Address 0x18 and 0x19

Access SSD16_UNDER

Bit	Name	Description
15–0	UNDERFLOW	A value of 1 in a bit indicates that the corresponding channel's internal FIFO has underflowed since the previous CMD_EN or CHANNEL_ENABLE. Reset by first disabling, then re-enabling, the channel (see the Channel Enable Register).

Overflow Register

Size 16-bit

I/O read only

Address 0x1A and 0x1B

Access SSD16_OVER

Bit	Name	Description
15–0	OVERFLOW	A value of 1 in a bit indicates that the corresponding channel's internal FIFO has overflowed since the previous CMD_EN or CHANNEL_ENABLE. Reset by first disabling, then re-enabling, the channel (see the Channel Enable Register).

Data Invert Register

Size 16-bit

I/O read only

Address 0x1C and 0x1D

Access SSD16_CHINVERT

Bit	Name	Description
15–0	CH_INVERT	A value of 1 in a bit indicates that the corresponding channel's data is inverted — a one becomes a zero, and vice-versa.

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 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.

Clock Select Register

Size	8-bit
I/O	read-write
Address	0x21
Access	EDT_SS_CLK_SEL
Comment	Selects output clock timing source. The internal clock is the default. External clocks let you choose an input channel's clock to serve as the output transmit clock for all output channels.

Bit	Values (0x)	Description
7–0	00	Internal from PLL1
	01	External, channel 0 input clock
	02	External, channel 1 input clock
	03	External, channel 2 input clock
	04	External, channel 3 input clock
	05	External, channel 4 input clock
	06	External, channel 5 input clock
	07	External, channel 6 input clock
	08	External, channel 7 input clock
	09	External, channel 8 input clock
	0A	External, channel 9 input clock
	0B	External, channel 10 input clock
	0C	External, channel 11 input clock
	0D	External, channel 12 input clock
	0E	External, channel 13 input clock
	0F	External, channel 14 input clock
	10	External, channel 15 input clock
	20	External, EXTCLKIN input clock *Not available for ECL mezzanine boards.*
	40	Enable PLL0 out on EXTCLKIN for board under test (testing only) *Not available for ECL mezzanine boards.*

PRBS15 Generator Register

Size	8-bit
I/O	read-write
Address	0x22
Access	SSD16_PRBS15_EN

Bit	Description
7–0	Set any bit to generate PRBS15 test code out on all channels. Clear all bits to stop PRBS15 code generation.

PLL 0 Divider Register *(PCI CDa only)*

Size 16-bit

I/O read-write

Address 0x24 and 0x25

Access EDT_SS_PLL0_CLK

Comment This register is set by `set_ss_vco`.

Bit	Name	Description
15–0	PLL0_DIV	A post-scalar divider used to achieve lower frequencies than those at which the PLLs can be programmed. After this division (if any), the clocks are divided by two for an even duty cycle — half the time high, and half low. `set_ss_vco` takes this into account. `sslooptest` uses PLL0 to set the output clock for the diagnostic PRBS15 test data generator.

PLL 1 Divider Register *(PCI SS/GS only)*

Size 16-bit

I/O read-write

Address 0x26 and 0x27

Access EDT_SS_PLL1_CLK

Comment This register is set by `set_ss_vco`.

Bit	Name	Description
15–0	PLL1_DIV	A post-scalar divider used to achieve lower frequencies than those at which the PLLs can be programmed. After this division (if any), the clocks are divided by two for an even duty cycle — half the time high, and half low. `set_ss_vco` takes this into account. `sslooptest` uses PLL1 to set the output clock for the diagnostic PRBS15 test data generator.

Idle Pattern Register

Size 32-bit

I/O read-write

Address 0x2C, 0x2D, 0x 2E, 0x2F

Access SSD16_IDLE_PAT_{L0, L1, L2, L3}

Bit	Name	Description
31–0	IDLE_PAT	After the bitfile has been loaded into the user interface Xilinx, when no DMA is occurring, the bitfile sends the 32-bit idle pattern written to this register — by default, all zeroes — at the configured output clock rate. When DMA starts, the idle pattern stops at the next 32-bit boundary.

Output Delay Count Register

Size	8-bit
I/O	read-write
Address	0x30
Access	OUTPUT_DELAY_COUNT

Bit	Name	Description
7–0	DELAY_COUNT	Number of clock cycles to wait before ouputting DMA data. Set this register to the appropriate number of clock cycles to prevent FIFOs at each channel from starting empty. The correct value must be determined by experiemntation. Clear (the default) if the DMA engine can keep up with output.

Output Clock Gate Register

Size	16-bit
I/O	read-write
Address	0x32, 0x33
Access	OUTPUT_CLOCK_GATE
Comment	Not implemented in `ssd16io_fast.bit`

Bit	Name	Description
15–0	CH_GATE	When set, the clock signal is output only with valid DMA data for the corresponding channel. When clear, the clock signal is output continuously, whether DMA is occurring or not.

Bit Error Control Register

Size	8-bit
I/O	read-write
Address	0x58
Access	SSD16_PRBS_ERR_CTRL
Comment	Used for testing.

Bit	Name	Description
7	RESET	Set to reset the PRBS15 test code generator.
6	ERROR	Set to insert an error in the PRBS15 pattern generated, for testing purposes.
5–0		not used

Board ID Register

Size	8-bit
I/O	read-write
Address	0x7F
Access	EDT_BOARDID
Comment	Returns a unique four-bit code corresponding to the mezzanine board installed. A value of 2 indicates an extended board ID. To read an extended board ID code, use the application `extbdid.exe` or the EDT DMA library routine `edt_get_boardID`.

Bit	Name	Description
7–5		used by `extbdid.exe`
4		not used; always set
3–0	BOARD_ID	The ID code of the installed mezzanine board:
		12 3x3G
		11 OC192
		10 16TE3
		F Combo I/O, ECL
		E Combo II I/O, RS-422
		D Combo III I/O, ECL
		C Combo III I/O, LVDS
		B Combo III I/O, RS-422
		A SRXL (with Graychips)
		9 TLK1501 I/O
		8 ECL I/O
		7 Combo II I/O, LVDS
		6 OCM
		5 HRC for E4, STM-1, OC3
		4–2 reserved
		1 LVDS I/O
		0 RS-422 I/O

Pinouts

The 16-channel Synchronous Serial I/O connects your device to the PCI CDa main board using the 80-pin connector as shown in Table 2, and to the PCI SS/GS main board using the 68-pin connector as shown in Table 3.

Signals labeled as free are connected to wires; your firmware can access these signals.

Table 2. SSD16IO to PCI CDa Connector Pinout

Pin	Signal	Pin	Signal
1	ground	41	ground
2	free	42	free
3	CH2D+	43	CH0D+
4	CH2D–	44	CH0D–
5	CH2CLK+	45	CH0CLK+
6	CH2CLK–	46	CH0CLK–
7	CH3D+	47	CH1D+
8	CH3D–	48	CH1D–
9	CH3CLK+	49	CH1CLK+
10	CH3CLK–	50	CH1CLK–
11	CH6D+	51	CH4D+
12	CH6D–	52	CH4D–
13	CH6CLK+	53	CH4CLK+
14	CH6CLK–	54	CH4CLK–
15	CH7D+	55	CH5D+
16	CH7D–	56	CH5D–
17	CH7CLK+	57	CH5CLK+
18	CH7CLK–	58	CH5CLK–
19	EXTCLKIN+	59	EXTCLKIN–
20	+5 V	60	+5 V
21	free	61	free
22	free	62	free
23	free	63	free
24	CH8D+	64	CH10D+
25	CH8D–	65	CH10D–
26	CH8CLK+	66	CH10CLK+
27	CH8CLK–	67	CH10CLK–
28	CH9D+	68	CH11D+
29	CH9D–	69	CH11D–
30	CH9CLK+	70	CH11CLK+
31	CH9CLK–	71	CH11CLK–
32	CH12D+	72	CH14D+
33	CH12D–	73	CH14D–
34	CH12CLK+	74	CH14CLK+
35	CH12CLK–	75	CH14CLK–
36	CH13D+	76	CH15D+
37	CH13D–	77	CH15D–
38	CH13CLK+	78	CH15CLK+
39	CH13CLK–	79	CH15CLK–
40	ground	80	ground

The board uses a high-density 68-pin SCSI-type I/O connector (Tyco part number 787169-7), with a straight-shielded backshell (Tyco part number 750752-1). You can use a typical SCSI cable (Tyco part number 749621-7) if your equipment has a SCSI connector.

Table 3. SSD16IO to PCI SS/GS Connector Pinout

P3	Signal	P3	Signal
1	CH15CLK+	35	CH15CLK–
2	CH0D+	36	CH0D–
3	CH0CLK+	37	CH0CLK–
4	CH1D+	38	CH1D–
5	CH1CLK+	39	CH1CLK–
6	CH2D+	40	CH2D–
7	CH2CLK+	41	CH2CLK–
8	CH3D+	42	CH3D–
9	CH3CLK+	43	CH3CLK–
10	CH4D+	44	CH4D–
11	CH4CLK+	45	CH4CLK–
12	CH9CLK+	46	CH9CLK–
13	CH5D+	47	CH5D–
14	CH5CLK+	48	CH5CLK–
15	CH6D+	49	CH6D–
16	CH6CLK+	50	CH6CLK–
17	CH12CLK+	51	CH12CLK–
18	CH13D+	52	CH13D–
19	CH7D+	53	CH7D–
20	CH7CLK+	54	CH7CLK–
21	EXTCLKIN+	55	EXTCLKIN–
22	CH8D+	56	CH8D–
23	CH12D+	57	CH12D–
24	CH11CLK+	58	CH11CLK–
25	CH11D+	59	CH11D–
26	CH10CLK+	60	CH10CLK–
27	CH8CLK+	61	CH8CLK–
28	CH9D+	62	CH9D–
29	CH10D+	63	CH10D–
30	CH13CLK+	64	CH13CLK–
31	CH14D+	65	CH14D–
32	CH14CLK+	66	CH14CLK–
33	CH15D+	67	CH15D–
34	ground	68	ground