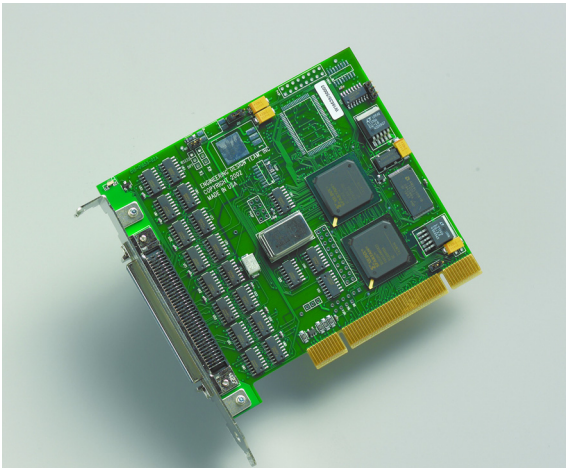
SSDIO

Synchronous Serial I/O

for PCI Local Bus Computers



May 15, 2007

008-01182-04

edt

a HEICO company

Copyright, Trademarks, Disclaimer

Contents

Synchronous Serial I/O Interface Overview

The SSDIO Synchronous Serial I/O Interface implements dual high-speed DMA channels, one for input and one for output, between an external device and the PCI Bus host computer. This document describes modifications made to the PCI CD to send and receive synchronous serial data. The change is made by loading a different FPGA configuration file into the UI Xilinx field-programmable gate array.

Each SSDIO input or output is composed of one data bit accompanied by a clock signal. The input data is sampled on either the rising or the falling edge of the clock signal, depending on the firmware, and stored in the host computer memory by the SSDIO DMA. The output data is read from the host computer memory by the PCI CD and shifted out one bit at a time on the positive or negative edge of the transmit clock, depending on the firmware.

Use the SSDIO with the SSD4 differential cable, EDT part number 016-00566, ending in four DB-25 male connectors. For SSDIO operation, channel 1 is the input and channel 4 is the output.

A 80-pin-to-80-pin connector for loopback testing with inputs tied to outputs is available with this cable: part number 016-02353-00.

Use this document with the *PCI CD/CDa User's Guide,* EDT part number 008-00965, available on our website at:

http://www.edt.com/manuals/PCD/pcicd.pdf

About the DMA Interface

The SSDIO 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`.

About the Software and Firmware

Install the SSDIO hardware and software according to the instructions in the *PCI CD/CDa User's Guide*, instead using one of the firmware files listed below:

`ssdio.bit` UI Xilinx configuration file for dual-channel synchronous serial input/output.

`ssdio_neg.bit` UI Xilinx configuration file for dual-channel synchronous serial input/output using the negative clock edge.

Compatible single-channel bitfiles must be loaded in the PCI Xilinx on the PCI CD. These are:

`pcd20.bit` The PCI Xilinx configuration file for PCI CD-20 boards.

`pcd60.bit` The PCI Xilinx configuration file for PCI CD-60 boards.

The following software initialization files are available for use with the `initpcd` automatic configuration utility:

`ssdio.cfg` Software initialization file for dual-channel synchronous serial input/output.

`ssdio_neg.cfg` Software initialization file for dual-channel synchronous serial input/output using the negative clock edge.

Software initialization files are also useful for examples of setting the output clock and otherwise configuring the board.

The following utilities are also available:

`rdssdio` Executable for testing the board; reads the serial input data and compares it to the known pattern.

`rdssdio.c` C source for the test program, useful as an example of data acquisition.

`wrssdio` Executable for testing the board; loads the firmware and generates a known stream of serial data.

`wrssdio.c` C source for the test program, useful as an example of loading the firmware, configuring the board, and generating data.

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

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

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

Utility Files

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

For the SSDIO 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 SSDIO

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

To configure the SSDIO, 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 ssdio.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

To test the SSDIO board, you'll need an 80-pin loopback connector, available from EDT (order part number 016-02353-00).

1. With the board installed in a host and powered on, plug the loopback connector into board. This connects the output channel to the input channel.
2. Start data output. At the command line, enter:

   `wrssdio -u` *`unit_number`*

   The default unit number is 0, indicating the first board in a system. If you have only one board in your system, you can omit the `-u` option. For a complete list of optional arguments, enter:

   `wrssdio -h`
3. Read and check the input data. At the command line, enter:

   `rdssdio -u` *`unit_number`*

   The default unit number is 0, indicating the first board in a system. You must use the same unit number for both `rdssdio` and `wrssdio`.

   If you have only one board in your system, you can omit the `-u` option. For a complete list of optional arguments, enter:

   `rdssdio -h`

   The program `rdssdio` reads the serial input data and compares it to the pattern of incrementing bytes generated by `wrssdio`. Unless configured with the `-c` option, `rdssdio` acquires eight 1 MB buffers and then checks for errors. The `-c` option specifies that it runs continuously until stopped with an operating system halt.

   If an error occurs, the output includes the string `ERROR`. If you do not see that string, then the board is operating correctly.

Connector Pinouts

Table 1 shows the signals out the 80-pin connector of the PCI CD when loaded with either `ssdio.bit` or `ssdio_neg.bit`.

Table 1. PCI CD (ssdio.bit, ssdio_neg.bit) Connector Pinout

Pin	Signal	Pin	Signal
1	ground	41	ground
2	ground	42	ground
3	reserved	43	INDATA+
4	reserved	44	INDATA–
5	reserved	45	INEN+
6	reserved	46	INEN–
7	reserved	47	reserved
8	reserved	48	reserved
9	reserved	49	reserved
10	reserved	50	reserved
11	OUTDATA+	51	reserved
12	OUTDATA–	52	reserved
13	SYNC+	53	reserved
14	SYNC–	54	reserved
15	reserved	55	reserved
16	reserved	56	reserved
17	reserved	57	reserved
18	reserved	58	reserved
19	not used	59	not used
20	+5V DC (fused)	60	+5V DC (fused)
21	not used	61	not used
22	not used	62	not used
23	ground	63	ground
24	reserved	64	INCLK+
25	reserved	65	INCLK–
26	reserved	66	reserved
27	reserved	67	reserved
28	reserved	68	reserved
29	reserved	69	reserved
30	reserved	70	reserved
31	reserved	71	reserved
32	reserved	72	reserved
33	reserved	73	reserved
34	reserved	74	reserved
35	reserved	75	reserved
36	reserved	76	reserved
37	reserved	77	reserved
38	OUTCLK+	78	reserved
39	OUTCLK–	79	reserved
40	ground	80	ground

Table 2 shows the signals from the 80-pin connector through the SSD4 cable and out the four DB-25 connectors.

Table 2. SSDIO Connector to 4-channel DB-25 Cable

Connector Pin	Channel-pin	Signal	Connector Pin	Channel-pin	Signal
1	CH1–25	Ground	41	CH2–25	Ground
2	CH1–25	Ground	42	CH2–25	Ground
3	CH2–3	reserved	43	CH1–3	INDATA +
4	CH2–4	reserved	44	CH1–4	INDATA –
5	CH2–5	reserved	45	CH1–5	INEN +
6	CH2–6	reserved	46	CH1–6	INEN –
7	CH2–7	reserved	47	CH1–7	reserved
8	CH2–8	reserved	48	CH1–8	reserved
9	CH2–9	reserved	49	CH1–9	reserved
10	CH2–10	reserved	50	CH1–10	reserved
11	CH4–3	OUTDATA +	51	CH3–3	reserved
12	CH4–4	OUTDATA –	52	CH3–4	reserved
13	CH4–5	SYNC +	53	CH3–5	reserved
14	CH4–6	SYNC –	54	CH3–6	reserved
15	CH4–7	reserved	55	CH3–7	reserved
16	CH4–8	reserved	56	CH3–8	reserved
17	CH4–9	reserved	57	CH3–9	reserved
18	CH4–10	reserved	58	CH3–10	reserved
19		not used	59	CH3–3	reserved
20		5V DC(fused)	60		5V DC(fused)
21		not used	61		not used
22		not used	62		not used
23	CH3–25	Ground	63	CH4–25	Ground
24		reserved	64	CH1–1	INCLK +
25		reserved	65	CH1–2	INCLK –
26		reserved	66		reserved
27		reserved	67		reserved
28		reserved	68		reserved
29		reserved	69		reserved
30		reserved	70		reserved
31		reserved	71		reserved
32		reserved	72		reserved
33		reserved	73		reserved
34		reserved	74		reserved
35		reserved	75		reserved
36		reserved	76		reserved
37		reserved	77		reserved
38	CH4–1	OUTCLK +	78		reserved
39	CH4–2	OUTCLK –	79		reserved
40	CH3–25	Ground	80	CH4–25	Ground

Register Modifications

The following SSDIO registers are modified from those described in the *PCI CD/CDa User's Guide*.

Command Register

Size 8-bit

I/O read-write

Address 0x00

Access PCD_CMD

Bit	PCD_	Description
4–7		not used
3	ENABLE	Set to one to enable the SSDIO 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		not used
1	FORCEBNR	A value of 1indicates that the board is not ready.
0	DIR	not used. Data received from the INDATA input is transferred with DMA channel 0 and data transmitted with OUTDATA output is transferred with DMA channel 1.

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	Set if the input FIFO (INDATA to memory) is full and input data must be discarded. Reset this bit with the ENABLE bit in the Command Register.
2	UNDERFLOW	Set if the output FIFO(DATA from memory to OUTDATA) runs out of data after data output has started and the ENABLE bit is set. Reset this bit with the ENABLE bit in the Command Register.
1	IF_NOT_EMP	If this bit is set, the input FIFO (INDATA to memory) is empty.
0	OF_NOT_EMP	If this bit is set, the output FIFO (DATA from memory to OUTDATA) is empty.

Funct Register

Size	8-bit
I/O	read-write
Address	0x02
Access	PCD_FUNCT
Comment	Used to control data acquisition and transmission.

Bit	PCD_	Description
7	PLLCLK	When clear, the phase-locked loop circuit is the transmit clock. When set, the signal coming in on INCLK is used instead.
6–4	reserved	Used to program the phase-locked loop circuit. Must always be low during normal operation
3	ENABLE_RDQ	Set to enable the read data qualifier ENIN signal. When this signal is enabled, it must be high at the rising edge of the clock for data to be stored.
2	STOP_CLK	Set to cause the transmitter to stop outputting clocks when no more data is available for transmission. If reset, the transmitter outputs the idle pattern bytes when no more data is available.
1	STROBE	Set to cause one dummy clock cycle of input data. It must be reset and set again to cause another. Used in conjunction with the three bits of the Stat Register to shift out the last bits of valid data that may be left in the input circuit, if the user clock should stop.
0	SHFT_RIGHT	Set to shift the data out from least significant bit of the byte to the most significant. Input data is also shifted in least significant bit first. If reset, the most significant bit is output first and presumed to arrive first.

Stat Register

Size	8-bit
I/O	read-only
Address	0x03
Access	PCD_STAT
Comment	The signals STAT(0–3) are disconnected. Three of these bits are used to determine if all serial input data has been flushed out of the receive circuits.

Bit	PCD_	Description
7–3		not used; set to zero
2–1	BYTECNT	indicate which byte of a 32-bit word is being received. To flush all valid input data, the software must write dummy clock cycles until both of these bits are set before the clock and the LAST_BIT signal is asserted after the clock. Write dummy clock cycles using the STROBE bit (bit 1) of the Funct register (address 0x02).
0	LAST_BIT	Set when the last bit written completes a received byte. To flush bit data, the software must write dummy clock cycles using the Funct Register STROBE bit (bit 1) until this bit is set.

Stat Polarity Register

Size	8-bit
I/O	read-write
Address	0x04
Access	PCD_STAT_POLARITY
Comment	This register is implemented and can be read or written, but is not used.

Idle Pattern Register

Size	16-bit
I/O	read-write
Address	0x18
Access	PCD_SSDIO_IDLE
Comment	The bit pattern stored in this register is output as data when no DMA data is available. If an 8-bit pattern is required, write the same pattern in both bytes of this register. If a constant 1 or 0 is required, write 0xFFFF or 0x0000, respectively.