

# SSE

## Synchronous Serial ECL

008-01134-01



INCORPORATED

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 merchantability 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.

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. 1997–2001. All rights reserved.

Sun, SunOS, SBus, SPARC, and SPARCstation are trademarks of Sun Microsystems, Incorporated.

Windows NT is a registered trademark of Microsoft Corporation.

Intel and Pentium are registered trademarks of Intel Corporation.

UNIX is a registered trademark of X/Open Company, Ltd.

OPEN LOOK is a registered trademark of UNIX System Laboratories, Inc.

Red Hat is a trademark of Red Hat Software, Inc.

IRex is a trademark of Silicon Graphics, Inc.

AIX is a registered trademark of International Business Machines Corporation.

Xilinx is a registered trademark of Xilinx, Inc.

Kodak is a trademark of Eastman Kodak Company.

The software described in this manual is based in part on the work of the independent JPEG Group.

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

---

# Contents

Overview .....	1
Current Status .....	1
Hardware Installation .....	2
Included Files .....	2
Testing .....	2
Further Description of Included Files .....	4
Interface Connector Pinout .....	6
PCD_CMD: Command Register .....	7
Data Path Status Register .....	7
PCD_FUNCT: Funct Register .....	7
PCD_STAT: Stat Register .....	8
PCD_STAT_POLARITY: Stat Polarity Register .....	8
SSE_BYTESWAP: Stat Bit Swap Register .....	9
SSE_ECLCTR0: ECL Control Byte 0 .....	10
SSE_ECLCTR1: ECL Control Byte 1 .....	10
SSE_ECLCTR2: ECL Control Byte 2 .....	10
SSE_ECLSTAT: ECL Stat Bits .....	11



---

## Overview

The PCIGP-SSE occupies one slot of your PCI bus. It has two independent serial channels of one bit each and works at speeds in excess of 400 MHz, as limited by available PCI-bus bandwidth. Drivers for Windows NT/2000, Solaris 2.x, and Linux are provided, along with sample application programs.

The board has two 9-pin D connectors, one for each channel. Each channel accepts a differential data signal (two wires) and a differential clock (two more wires) at standard ECL signal levels. All incoming signals are terminated through 50 ohms to -2 volts.

The channels are independent; data arriving at each channel is stored into different regions of host computer memory over the PCI bus by our DMA engine.

Incoming serial data can be aligned to the appropriate byte or word boundary. It is up to the user to determine when the data is properly aligned.

Channel B can be turned around for use as an output. The output clock can come from either an on-board clock generator (50 Hz to 800 MHz) or from whatever clock is present on channel A. The output data can be either from counters internal to the board or from an outgoing DMA data stream from host memory.

### Current Status

Boards are tested at 400 MHz into channel A. With the FPGA (Field Programmable Gate Array) configuration files we are now sending out, channel B is limited to emitting an 8-bit binary count for testing channel A.

---

## Hardware Installation

Insert the PCIGP-SSE board into one of your PCI slots. Install a loopback cable (EDT part number LBKA 016-00067-00) between the two DB9 connectors on the PCIGP-SSE board.

## Software Installation

Install as directed in the PCI GP manual.

## Included Files

The following SSE files are included in the EDT PCI-CD driver directory (Solaris: /opt/EDTpcd; Linux: /usr/local/edt/pcd; WINNT: \edt\pcd):

README.sse	File of instructions
sseload.pdb	A batch script read by ssepdb to load the daughterboard Xilinx.
ssepdb.c	A debugger and utility for handling the SSE.
sserd.c	A sample application program for reading data from the SSE
sserot.c	A sample program to inspect for count data.
sserot8.bat	Runs sserot.c for each of eight possible bit alignments.
bitfiles/4036/sseb.bit	Config bitfile for the interface Xilinx on the bottom board.
bitfiles/XC2S100/sset.bit	Config bitfile for the Xilinx on the top board.

Program sserd.c is a good example of how to write an application for the PCIGP-SSE board.

## Testing

Conduct this test with the provided loopback cable installed between the two DB9 connectors.

Initialize the SSE with the following commands, substituting your unit number for the 0 given as an example below:

```
bitload -u 0 sseb.bit #Load bottom Xilinx (4036/4085 auto-selected)
ssepdb 0 sseload.pdb #Load top board Xilinx, set test gen to 50 MHz
```

Loading the top board Xilinx currently takes about 30 seconds, this will be sped up to a couple of seconds in a future release.

Now perform a dma read of the SSE and check the resultant data:

```
sserd -u 0 -o out 10000 #Read 10,000 bytes to file out
saserot8 out #Test it
```

If all goes well, should give output something like this:

```
salmon 150% sserd -u 0 -o out 10000
reading 10000 bytes
Writing 10000 bytes to out
```

```
salmon 151% sserot8 out
Rot:0 Errors:9961 Bytecount:9999
Rot:1 Errors:9961 Bytecount:9999
Rot:2 Errors:9961 Bytecount:9999
Rot:3 Errors:9961 Bytecount:9999
Rot:4 Errors:9961 Bytecount:9999
Rot:5 Errors:0 Bytecount:9999
Rot:6 Errors:9961 Bytecount:9999
Rot:7 Errors:9961 Bytecount:9999
```

At this point you can call `sserd` with `-s 5`, and it will use `sse_shift()` to shift the state machine in the SSE board five places to the left. (The `-s` option does not work when the SSE's on-board clock generator is set to less than 50 MHz as noted below.)

The data read from this call of `sserd` (and subsequent calls that do not use the `-s` option) will be properly aligned, assuming the incoming clock is not disturbed. Calling `sserot8` again will result in:

```
salmon 150% sserd -u 0 -s 5 -o out2 10000
reading 10000 bytes
Writing 10000 bytes to out2
```

```
salmon 151% sserot8 out2
Rot:0 Errors:0 Bytecount:9999
Rot:1 Errors:9961 Bytecount:9999
Rot:2 Errors:9961 Bytecount:9999
Rot:3 Errors:9961 Bytecount:9999
Rot:4 Errors:9961 Bytecount:9999
Rot:5 Errors:9961 Bytecount:9999
Rot:6 Errors:9961 Bytecount:9999
Rot:7 Errors:9961 Bytecount:9999
```

The file `out2` will have correctly aligned data.

## Further Description of Included Files

The utility `sserot` assumes 8-bit binary count data such as provided in hardware on the SSE by the test output mode on channel B. It reports the number of errors found, given an argument indicating how far to shift the bits. It reports a byte count one less than the size of the input file.

You can increase the bytecount (to `sserd`) up to however much RAM the driver is able to lock down, typically many tens of megabytes. In a continuously streaming application, you would use four ring buffers (as provided by EDT library calls) rather than the single buffer found in `sserd.c`.

### ***Bit, Byte, and Word Swapping***

The program `sserd.c` includes the following call:

```
edt_intf_write(edt_p, SSE_BYTESWAP, sse_bswap) ;
```

This writes the 8-bit value `sse_bswap` out to a hardware register named `SSE_BYTESWAP`. Control bits in `SSE_BYTESWAP` are defined by `SSE_SWAPBITS`, `SSE_SWAPBYTES`, and `SSE_SWAPSHORTS` in `edtdreg.h`. (The program `sserd.c` can set these when given the `'-b 1'`, `'-B 1'`, or `'-S 1'` command line options.)

Normally, data is read LS bit first. `SSE_SWAPBITS` causes each incoming byte to be flipped end for end, so it is MS bit first.

Data is also normally returned to the host in little-endian byte order. On some machines, this can be mishandled by the DMA hardware available on the host. `SSE_SWAPBYTES` exchanges the bytes in each 16-bit word, and `SSE_SWAPSHORTS` exchanges the 16-bit words in each 32-bit word.

### ***Setting the Frequency***

You can call `sse_set_out_clk(EdtDev *edt_p, double outFreq)` from your program to change the frequency of the onboard clock generator used to test the board. `sserd.c` does this when passed the `"-f freq"` argument, where `"freq"` is a floating point number in MHz.

You can also use the debugger (or edit the `sseload.pdb` script used above):

```
ssepdb 0
h (see a listing of debugger commands)
fosc 400 (set it to 400 MHz)
q (quit)
```

The precision to which the on-board clock generator can be set varies with the frequency chosen:

400-800 MHz	multiples of 2 MHz
200-400 MHz	multiples of 1 MHz
100-200 MHz	multiples of 0.5 MHz
50-100 MHz	multiples of 0.25 MHz
0.00050-50 MHz	very close, but somewhat arbitrary

### ***Bit Aligning in Hardware***

You can use the function `sse_shift(EdtDev *edt_p, int bitsToShift)` to rotate the bit alignment in hardware. `sserd.c` does this when passed the `"-s numbits"` argument. The bit alignment jumps whenever the incoming clock is disturbed.

The SSE's programmable clock (as set by the `sse_set_out_clk()` library call, or the debugger's `fosc` command) must be at least 50 MHz for the above hardware bit alignment scheme to work.

Note that this is only a handicap when testing with the loopback cable. In normal use the user supplies the clock and data for input, so the fact that the SSE's unused on-board clock generator must be 50 MHz or greater is not a burden.

An alternative is to rotate the data after it has been collected, using the `sserot.c` program (which is called from the `sserot8` script).

---

## Interface Connector Pinout

Both DB9 connectors are pinned out as follows:

- 1: Reserved
- 2: GND
- 3: Reserved
- 4: GND
- 5: Reserved
- 6: DAT-
- 7: DAT+
- 8: CLK-
- 9: CLK+

## Remote Xilinx Registers

### PCD\_CMD: Command Register

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

Bit	PCD_	Description
D0–2		Not used.
D3	ENABLE	Set to one to enable the PCIGP-SSE interface. This bit is set after the direction is chosen and typically after the first DMA buffer is ready. To reset direction or flags this bit must be reset.  To flush the DMA FIFOs, clear then set this bit.
D4–7	STAT_INT_EN	0

### Data Path Status Register

Not used.

### PCD\_FUNCT: Funct Register

Size	8-bit
I/O	read-write
Address	0x02

Bit	PCD_	Description
D0	XCPROG	Used to program daughterboard interface Xilinx.
D1	XCCLK	
D2	SCDAT	
D3		Not used.
D4	FSEL6	Used by the EDT library routine <code>edt_set_out_clock()</code> to program the phase-locked loop.
D5	FSEL7	
D6	AVCE	
D7	SELAVCLK	1 to select GP PLL clock, 0 for ECL PLL from 50-800 MHz.

**PCD\_STAT: Stat Register**

Size	8-bit
I/O	read-only
Address	0x03

Bit	PCD_	Description
D0	STAT0	Channel A overflow flag, cleared when RSTA asserted or ENABLE deasserted.
D1	STAT1	Channel B overflow flag, cleared when RSTA asserted or ENABLE deasserted.
D2	STAT2	XCINITL
D3	STAT3	XCDONE
D4–7	SINTO-3	<p>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:</p> <ul style="list-style-type: none"> <li>The device interrupt is enabled using the RMT_EN_INTR bit in the PCI Interrupt and Remote Xilinx Configuration register.</li> <li>The corresponding bit is asserted in the command register (one of bits 4–7, named STAT_INT_EN).</li> </ul> <p>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.</p>

**PCD\_STAT\_POLARITY: Stat Polarity Register**

Size	8-bit
I/O	read-write
Address	0x04

Bit	PCD_	Description
D0–3	POLARITY	<p>A value of 0 indicates that a change from 0 to 1 from one clock cycle to the next causes an interrupt in the corresponding bit of the STAT_INT register, if the corresponding bit is also enabled in STAT_INT_EN.</p> <p>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.</p>
D4	STAT_INT_ENA	Provides global enable or disable for all interrupt bits in Stat register, 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.
D5–7		Not used.

**SSE\_BYTESWAP: Stat Bit Swap Register**

Size           8-bit  
 I/O            read-write  
 Address        0x0F

Bit	PCD_	Description
D0	SWAPBYTES	Swap bytes within each 16-bit word.
D1	CHANB_OUT	0: channel B DMA is used for input 1: channel B DMA is used for output
D2	SWAPBITS	Swap bits end-for-end within each byte.
D3	SWAPSHORTS	Swap 16-bit words within each 32-bit word.
D4	RSTA	Reset channel A.
D5	RSTB	Reset channel B.
D6–7		Not used.

**SSE\_ECLCTR0: ECL Control Byte 0**

Size 8-bit  
 I/O read-write  
 Address 0x10

Bit	PCD_	Description
7-0	ECLCTR[7-0]	ECL control bits.

ECLCRT[0-6]: delay line input for channel A inputs DATA and CLOCK

ECLCRT7: strobe 0-6 into delay line for Channel A DATA, 20 picosecond increments

**SSE\_ECLCTR1: ECL Control Byte 1**

Size 8-bit  
 I/O read-write  
 Address 0x11

Bit	PCD_	Description
15-8	ECLCTR[15-8]	ECL control bits.

ECLCTR8: strobe 0-6 into delay line for Channel A CLOCK, 20 picosecond increments

ECLCTR9: rising edge causes Channel A input to swallow one clock

ECLCTR10: 1 selects divide-by-8 mode for Channel A and B input shifters

ECLCTR11: 1 selects internal loopback from output shifter for Channel A input

ECLCTR12: 1 selects output from Channel A shifter for Channel B input

ECLCTR13: 0 selects Channel B input clock from Channel B input, 1 from Channel A

ECLCTR14: 0 selects Channel B output clock from PLL, 1 from Channel A connector

ECLCTR15: 1 selects divide-by-8 mode for Channel B output shifter

**SSE\_ECLCTR2: ECL Control Byte 2**

Size 8-bit  
 I/O read-write  
 Address 0x12

Access

Bit	PCD_	Description
7-0	ECLCTR[23-16]	ECL control bits.

ECLCTR16: 0 sets Channel B to output mode, 1 for input mode

ECLCTR17: 1 selects reference clock as output from ECL clock generator, 0 selects PLL

ECLCTR18: Serial clock to configure ECL clock generator

ECLCTR19: Serial data to configure ECL clock generator

ECLCTR20: 1 loads ECL clock generator from shifted in configuration data

ECLCTR21: rising edge causes Channel B input to swallow one clock

ECLCTR22-23: not used

### **SSE\_ECLSTAT: ECL Stat Bits**

Size 8-bit

I/O read-write

Address 0x14

Bit	PCD_	Description
0-6	ECLSTAT[0-6]	7 bits of data back from ECL hardware.
7	BUSY	ECLCTR registers busy.

True if write to any of the ECL\_CONTROL registers has not yet had time to be transmitted to the daughterboard.

ECLSTAT0-6: reserved

BUSY: True after loading any ECLCTR register; turns false when this data has been shifted out to the hardware. Maximum busy time is 10 microseconds.

#### ***Additional Notes about ECL control bits***

All bits default to 0

Bits 0-8 are used to configure the programmable delay lines on the channel A clock and data inputs.

Bits 9 and 21 are used to bit-align the input shifters

Setting bits 11,12,13,15 might give useful loopback test (the loopback cable checks more hardware)

Bit 14 selects clock source for channel B output

Bit 16 determines if Channel B is output or input

Bit 17 when high causes ECL PLL to pass through clock from PCIGP PLL, low for normal use of ECL PLL.

Bits 18,19,20 used to determine frequency of ECL PLL clock generator

Setting ECLCTR bits 10,12,13,16, and aligning via ECLSTAT bits 0,1 and ECLCTR bits 9,21 allows input shifters to be daisy-chained for higher speeds (not yet implemented).



## Index

### C

- command register 7
- configuration space
  - status field 11

### D

- data path status register 7
- DIR bit 7

### E

- ENA\_OUT\_CTRL bit 8, 9
- ENABLE bit 7

### F

- FUNCT bit 7
- funct register 7

### I

- interface configuration register 10, 11

### P

- POLARITY bits 8

### R

- registers

- command 7
  - data path status 7
  - funct 7
  - interface configuration 10, 11
  - stat 8
  - stat polarity 8, 9

### S

- STAT bits 8
- stat polarity register 8, 9
- stat register 8
- STAT\_INT bits 8
- STAT\_INT\_EN bit 7
- STAT\_INT\_ENA bit 8, 9