User Manual

PCI-Altera-485/LVDS

Re-configurable Logic
with RS-485/LVDS and TTL IO

Supplementary Manual for Engineering Kit VHDL

Revision A
Corresponding Hardware: Revision E/F/G
10-2002-0705/0706/0707
PCI-Altera-485/LVDS
PCI based Re-configurable logic with RS-485/LVDS and TTL IO

This document contains information of proprietary interest to Dynamic Engineering. It has been supplied in confidence and the recipient, by accepting this material, agrees that the subject matter will not be copied or reproduced, in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered.

Dynamic Engineering has made every effort to ensure that this manual is accurate and complete. Still, the company reserves the right to make improvements or changes in the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Dynamic Engineering’s products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

Connection of incompatible hardware is likely to cause serious damage.

©2002-2004 by Dynamic Engineering. Other trademarks and registered trademarks are owned by their respective manufacturers. Manual Revision A. Revised 12/9/04
# Table of Contents

## PRODUCT DESCRIPTION  5

## PROGRAMMING  9

**Address Map Altera**  11

**VHDL Files**  12
- Example Instantiation  13
- Design Modification example:  14
- PCIALT_BASEA  16
- PCIALT_RX_TX  18
- PCIALT_DIR1,2  18
- PCIALT_TERM1,2  19
- PCIALT_IO1,2  19
- Application Note: Spare IO  19
- PCIALT_FIFO_NA  20
- PCIALT_TTL  20
- PCIALT_FIFOSTAT1  21
- PCIALT_OSCCNT  22
- PCIALT_OSCDAT  23
- PCIALT_FIFOSTAT2  23
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1</td>
<td>PCI-ALTERA-485/LVDS BLOCK DIAGRAM</td>
<td>6</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>VHDL BLOCK DIAGRAM</td>
<td>7</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>PCI-ALTERA-485/LVDS ALTERA-ATP ADDRESS MAP</td>
<td>11</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>PCI-ALTERA-485/LVDS ALTERA BASE CONTROL REGISTER</td>
<td>16</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>PCI-ALTERA-485/LVDS ALTERA BASE CONTROL REGISTER</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 6</td>
<td>PCI-ALTERA-485/LVDS DIRECTION CONTROL REGISTERS</td>
<td>18</td>
</tr>
<tr>
<td>FIGURE 7</td>
<td>PCI-ALTERA-485/LVDS TERMINATION CONTROL REGISTERS</td>
<td>19</td>
</tr>
<tr>
<td>FIGURE 8</td>
<td>PCI-ALTERA-485/LVDS IO CONTROL REGISTERS</td>
<td>19</td>
</tr>
<tr>
<td>FIGURE 9</td>
<td>PCI-ALTERA-485/LVDS ALTERA FIFO PORTS</td>
<td>20</td>
</tr>
<tr>
<td>FIGURE 10</td>
<td>PCI-ALTERA-485/LVDS TTL CONTROL REGISTER</td>
<td>20</td>
</tr>
<tr>
<td>FIGURE 11</td>
<td>PCI-ALTERA-485/LVDS FIFO STATUS PORT1</td>
<td>21</td>
</tr>
<tr>
<td>FIGURE 12</td>
<td>PCI-ALTERA-485/LVDS OSCILLATOR CONTROL REGISTER</td>
<td>22</td>
</tr>
<tr>
<td>FIGURE 13</td>
<td>PCI-ALTERA-485/LVDS OSCILLATOR DATA PORT</td>
<td>23</td>
</tr>
<tr>
<td>FIGURE 14</td>
<td>PCI-ALTERA-485/LVDS FIFO STATUS PORT2</td>
<td>23</td>
</tr>
</tbody>
</table>
Product Description

PCI-Altera-485/LVDS is part of the PCI Compatible family of modular I/O components. The PCI-Altera-485/LVDS provides a user configurable 20K400E FPGA, along with 40 RS-485 or 40 LVDS transceivers, PLL and FIFO support, full DMA capabilities in a half-length single slot card. The 20K400E can be replaced with larger FPGAs for really large projects.

The RS-485 and LVDS parts can be mixed. The standard RS-485 devices are 5V parts with 40 MHz bandwidth. The standard LVDS parts are 3.3V parts. When mixed a different 485 device is utilized with lower bandwidth [Max3485].

The PCI bus implementation is 32 bits at 33 MHz, universal voltage. The hardware supports direct access software controlled read/write access to all locations plus DMA support to the high bandwidth ports. The hardware is optimized for back-to-back DMA accesses to support the multiple ports available on the PCI-Altera-485/LVDS.

The PCI-Altera-485/LVDS utilizes a PLX 9054 device for the PCI interface, and a dedicated Xilinx FPGA to manage the 9054 and provide for loading the Altera.

The User functionality of the PCI-Altera comes from installing custom VHDL into the Altera FPGA. The board level details can be found in the PCI-Altera reference manual. This manual covers the VHDL implementation done for the Engineering Kit. The Engineering Kit provides a known good starting point for new designs. The kit comes with VHDL and project information to allow you to open the Altera project with Quartus.

A recommended approach to working with Dynamic Engineering drivers and hardware for the PCI-Altera is to build on the engineering kit. The base design has a working interface for the PCI bus, FIFOS, PLLs and IO. The design can be modified to add your features without breaking the basic features that allow the driver to control the hardware. This manual and the additional notes within the VHDL will delineate the design aspects that can be changed and those which should not be changed.
The board level diagram shows the relationship of the major components of the PCI-Altera design. The following diagram shows the relationship of the major components of the VHDL implemented for the Engineering Kit.
The VHDL included in the Engineering Kit is designed to allow the "userap" C software to make driver calls and test the PCI Altera hardware. The PCI bus can communicate directly with the Altera via PLX and the Bus I/F vhdl. The direct path is usually used for set-up of operational parameters, and for status.

Each of the REG blocks are read-write registers accessible via the PCI bus. The RD and WR blocks are unidirectional. Each of the major functions controlled by the VHDL block is shown.
The lower right corner of the block diagram shows the RS-485 or LVDS IO control and test blocks. The PLLs are connected to 32 bit counters and to the IO. There are 24 PLL clock references which are muxed with the lower 24 IO bits. REG 18 controls the source of the IO bit. The clocks can be output onto the IO lines if the direction bit is set to output for the line of interest and the clock mux is set to pick the clock. The clock mux was added to allow the user easy access to the PLL outputs for debugging purposes. You can program a channel and check that the programmed frequency is what you expected. The clocks are also routed to 32 bit counters to allow the ATP software to test that the PLLs are working. The Userap software loads each channel with a unique stepped frequency description. The software then runs the counters until the reference frequency has output bit 24 set. The counts are then read back and compared against an expected result to see if the PLLs are operating properly. The logic is not needed for most applications and can be removed or left in for BIT capability. The count mux is also controlled with REG18. The counts are read back from RD 19.

In the ATP VHDL the data for the IO comes from WR6 and WR7. If the clock mux is set to data and the IO direction is set to out then the value of the WR6,7 are set on the IO lines. The data is read back from RD6,7. As you can see the read back path is from the IO side rather than the register side to allow input from the external IO. Most users will want to modify this section to use a state-machine with data paths utilizing the FIFOs.

The ATP VHDL uses a loop-back between the RX and TX side of each channel to allow direct loop-back from the Xilinx side of the FIFOs. The data is written by the Xilinx to the TX side, The Altera has state-machines which copy the data from the TX to the RX FIFO. The Xilinx reads the data back from the RX FIFO. The data can also be read from or written to the FIFOs from the PCI/PLX interface. Leaving this functionality in place will enhance your BIT capabilities.

The FIFO status can be read from RD17. The FIFO status has the flags associated with the Altera side of the FIFOs. The Xilinx has additional status available. In the ATP VHDL the status is used for the loop-back of the FIFO data. In your design the status can be used to help control your IO functions.

There are 12 TTL IO available. The read port is buffered and connected to the output side of the ‘125s [external to the Altera open drain buffers]. For the transmitters, each channel when set low drives the associated line low. When set high the buffer is disabled and the pull-up resistor brings the IO line high. For the ATP this port is used as a simple parallel port. Loop-back can be done without a loop-back cable due to the architecture. The ATP test fixture HDEterm100 with added loop-back wiring is used to make sure that the signal path is operational including the connector.
Programming

The PCI-ALTERA-485/LVDS is tested in a Windows environment. We use the Dynamic Engineering Driver to do the low level accesses to the hardware. We use MS Visual C++ in conjunction with the driver to write our test software. Please consider purchasing the Software Support Engineering Kit to work with the Hardware Support Engineering Kit for the PCI-ALTERA-485/LVDS; the software kit includes our test suite.

The Driver is layered with separate parts for the base design and the Altera application specific features. The base driver, a generic driver, and a driver for the ATP version are included in the package. The ATP [acceptance test procedure] uses "C" to make calls “UserAp” to the ATP driver to access the hardware and perform tests. The C code is also included. The ATP driver includes the Generic driver capabilities to allow the user to incrementally update the ATP VHDL code, using the UserAp software to access those portions not changed and user code with the generic capabilities to communicate with your changed hardware.

The base driver automatically loads the Altera design driver based on the ID of the design. If the design is not recognized then the generic driver is installed instead. The ATP has a specific ID. Please refer to the VHDL, and driver manuals for more information.

Please note that the driver takes care of setting up the PLX. The following is provided for hardware only engineering kits.

Before communication with the Xilinx or Altera devices can happen the PLX device requires some initialization. The local bus address space must be enabled and if interrupts are to be used the PLX must be enabled for those too.

Writing to the PLX local configuration address offset Ox4 (LASOBA) with Ox01 will enable the local bus for memory space access, and re-map the local address to offset O.

Writing to the Bus Region Descriptors [LBRDO] Ox18 with Ox40430343 will put the local bus into a well behaved state to inter-operate with the Xilinx and Altera. Specifically we are disabling the pre-fetch capability for the memory and ROM spaces. With the FIFO interfaces pre-fetching can lead to loss of data. More detail is available in the PLX 9054 HW design manual.

Writing Ox01200000 to [MARBR] Ox08 will set the Mode/DMA Arbitration to the correct state for operation.

To use interrupts from the Xilinx or Altera the [INTCSR] Ox68 will need to be programmed. OxF000900 will enable the local bus interrupt and PCI interrupt capability. To disable the local side interrupt clear bit 11.
Operation with DMA requires additional register programming within the PLX and Xilinx devices. The Dynamic Engineering Driver takes care of all of the initialization if it is used. Windows 2000 and XP are currently supported.

Access to the user FPGA is done with the same base address and all local addresses with address 13 set. 0x00000000 -> 0x0001FFF is allocated to the Xilinx and 0x0002000 -> 0x0003FFF to the user FPGA. The Xilinx decodes the lower 32 longwords. The internal registers for the Xilinx are defined in the following pages.

The Xilinx specific registers are covered in the Board level hardware manual and the driver manuals.
### Address Map Altera

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCIALT_BASEA</td>
<td>0x00002000 // 0 base control register</td>
</tr>
<tr>
<td>PCIALT_RXTX</td>
<td>0x00002004 // 1 Enables for the 8 Tx and 8 Rx state machines</td>
</tr>
<tr>
<td>PCIALT_DIR1</td>
<td>0x00002008 // 2 Direction control register lower 20 bits</td>
</tr>
<tr>
<td>PCIALT_DIR2</td>
<td>0x0000200C // 3 Direction control register upper 20 bits</td>
</tr>
<tr>
<td>PCIALT_TERM1</td>
<td>0x00002010 // 4 Termination control register lower 20 bits</td>
</tr>
<tr>
<td>PCIALT_TERM2</td>
<td>0x00002014 // 5 Termination control register upper 20 bits</td>
</tr>
<tr>
<td>PCIALT_I01</td>
<td>0x00002018 // 6 485/LVDS IO lower 20 bits</td>
</tr>
<tr>
<td>PCIALT_I02</td>
<td>0x0000201C // 7 485/LVDS IO upper 20 bits</td>
</tr>
<tr>
<td>PCIALT_FIFO_0A</td>
<td>0x00002020 // 8 TX_FIFO0 - read, RX_FIFO0 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_1A</td>
<td>0x00002024 // 9 TX_FIFO1 - read, RX_FIFO1 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_2A</td>
<td>0x00002028 // 10 TX_FIFO2 - read, RX_FIFO2 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_3A</td>
<td>0x0000202C // 11 TX_FIFO3 - read, RX_FIFO3 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_4A</td>
<td>0x00002030 // 12 TX_FIFO4 - read, RX_FIFO4 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_5A</td>
<td>0x00002034 // 13 TX_FIFO5 - read, RX_FIFO5 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_6A</td>
<td>0x00002038 // 14 TX_FIFO6 - read, RX_FIFO6 - write</td>
</tr>
<tr>
<td>PCIALT_FIFO_7A</td>
<td>0x0000203C // 15 TX_FIFO7 - read, RX_FIFO7 - write</td>
</tr>
<tr>
<td>PCIALT_TTL</td>
<td>0x00002040 // 16 Write - TTL data out, Read - TTL data bus in</td>
</tr>
<tr>
<td>PCIALT_FIFOSTAT1</td>
<td>0x00002044 // 17 Altera FIFO empty, full status port</td>
</tr>
<tr>
<td>PCIALT_OSC_CNT</td>
<td>0x00002048 // 18 Oscillator/Counter Mux Control Port</td>
</tr>
<tr>
<td>PCIALT_OSC_DATA</td>
<td>0x0000204C // 19 Counter Data Read Port</td>
</tr>
<tr>
<td>PCIALT_FIFOSTAT2</td>
<td>0x00002050 // 20 Altera FIFO programmable almost MT/full status port</td>
</tr>
</tbody>
</table>

*FIGURE 3  PCI-ALTERA-485/LVDS ALTERA-ATP ADDRESS MAP*

The Altera FPGA is completely programmable – the address map above and definitions below only have meaning if the Engineering Kit is used as a starting point for your design. The reference software and reference Altera hardware implementation are used to perform the ATP on each board prior to shipment. The Engineering kit also includes the HDEterm100 and a cable to interconnect the PCI-Altera-485/LVDS with the HDEterm100. For more information please refer to the web page.

Whether you use the reference design or not please note that the Altera decoding starts at address 0x2000.

To maximize the driver utility it is recommended that the BaseA and FIFO0-7A addresses are left alone in the new VHDL. The base driver uses these ports to determine the size of the FIFOs and to check the design ID. There are notes in the VHDL. Additional decoding can be added to the decode module should your design require more than the spare decodes provide.

Please note that any VHDL which is changed in definition will need to be controlled with the Generic driver or the ATP driver using the Generic Driver capabilities. Please see the driver manuals for more information.

If possible to add your logic above address 0x50 and use muxes to tie your logic in with the base design then it is possible to have the full ATP capability in addition to your logic. In many projects it is advantageous to have a BIT capability. The ATP code can provide that for you.
The project hierarchy has top.vhd at the top. Top is used as a conduit to bring the standardized pin names and special Altera implementation into the design. The signals are remapped to userapp.vhd. Within userapp.vhd the signals are distributed to the submodules which use and control them. Userapp.vhd acts in many ways as the top layer of the design. We use one level of indirection because it helps with portability between designs and manufacturers.

The following files are currently in the design directory supplied with the VHDL project. A short description follows each name. Each of the files has comments and descriptive information where the code itself is not obvious as to operation or intent. Some basic detail follows.

**VHDL Files**

- userapp.vhd: bus interface and register definitions, logic tie off
- user_decode.vhd: decode the bus interface signals into read and write decodes
- cntr32.vhd: 32 bit up counter with preload and clear
- frx_ld.vhd: FIFO RX write state machine
- ftx_rd.vhd: FIFO TX read state machine
- lat1.vhd: 1 bit latch with clock enable
- lat4.vhd: 4 bit latch with clock enable
- lat8.vhd: 8 bit latch with clock enable
- load_rx.vhd: state machine to load the RX FIFO directly from the bus interface
- mux24_32.vhd: 25 to 1 mux where 23-0 select 23-0 and 31-24 select 24.
- read_tx.vhd: state machine to read the TX FIFO directly
- rlat12.vhd: read write 12 bit latch with 0 fill for the upper bits on read-back
- rlat20.vhd: read write 20 bit latch with 0 fill for the upper bits on read-back
- rw_lat2.vhd: read write 2 bit latch
- rw_lat4.vhd: read write 4 bit latch
- rw_lat8.vhd: read write 8 bit latch
- rw_lat16.vhd: read write 16 bit latch
- rw_lat20.vhd: read write 20 bit latch
- rw_lat32.vhd: read write 32 bit latch
- wlat12.vhd: 12 bit latch
- wlat20.vhd: 20 bit latch

Files not normally altered by the user:

- top.vhd: Top of hierarchy
- data_io.vhd: data mux for interface with PLX not normally modified by user
- pll_en.vhd: state machine to control the PLL enable signal.

The registers are read-writeable wherever possible. The VHDL module for the registers is mainly rw_lat32.vhd. The register has clock, data (bidirectional), clock enable, data enable, and reset inputs. The clock is the PCI reference clock in most cases. The data is tied to the bidirectional internal data bus. The clock
enable is from the write decode. The read enable is from the read decode. The timing is controlled within the user decode VHDL block. The output side of the latch is routed to the function it controls. There are vhdl modules for 2, 4, 8, 16, 20, and 32 bit wide registers in the design directory.

Example component definition

```vhdl
component RW_LAT32
    port ( 
        INOUTX : inout std_logic_vector(31 downto 0);
        clk : in std_logic;
        rst : in std_logic;
        clk_en : in std_logic;
        bus_en : in std_logic;

        OUTX : out std_logic_vector(31 downto 0)
    );
end component;
```

Example Instantiation

```vhdl
osc_cntrl : RW_LAT32
    port map ( 
        INOUTX => data,
        CLK => clk,
        RST => rst,
        clk_en => load_18,
        bus_en => read_18,

        OUTX => osc_cntrl
    );
```

The data signal is a 32 bit wide bi-directional bus. Clk is the reference clock used for the bus interface and all synchronous devices controlled by that interface. Load_18 and read_18 are the write and read decodes. Osc_cntrl is the output bus.

If you need a function that is similar to one of ours it is usually quicker to modify than to start over. You can modify our VHDL by opening, “saving as” and then adding to your project. Any changes made to the signals will need to be carried through to the next level above the modified VHDL.
Design Modification example:

The design has LEDs attached to the Xilinx and to the Altera. To make a change in the Altera controlled LEDs first look at the signal definition block at the top of userapp.vhd.

```vhdl
led : out std_logic_vector(3 downto 0); -- user LEDs
```

The LED bus is a 4 bit [one per LED] bus defined as output. Next search for “led” to find where the signal is defined in the current design. After several “find next” you will find the following block of vhdl.

```vhdl
-- LED control register - address = 0x2000
LED_reg : LAT4

port map (  
    DIN => data(27 downto 24),  
    CLK => clk,  
    RST => rst,  
    clk_en => load_0,  
    out_en => vcc,  
    DOUT => led_cntl  
);

led(0) <= not led_cntl(0);  
led(1) <= not led_cntl(1);  
led(2) <= not led_cntl(2);  
led(3) <= not led_cntl(3);
```

The LEDs are controlled by the register bits 27-24 of the base control register. You could have found this by looking at the register map definitions or the block diagram as well as searching the VHDL.

The LEDs are tied via a current limiting resistor to VCC and the Altera provides a switched “gnd”. The register bits are inverted so a ‘1’ written to the corresponding LED bit causes a low on the output and the LED to illuminate.

Lets change the controls to have all 4 turn on together. All we need to do in this case is to change the inversion definitions.

```vhdl
led(0) <= not led_cntl(0);  
led(1) <= not led_cntl(0);  
led(2) <= not led_cntl(0);  
led(3) <= not led_cntl(0);
```

Bit 0 of the LED control bus will now turn all 4 LEDs on and off.
To implement:
Save the file userapp.
Compile the design.
Move the new .rbf file to the AlteraDesigns directory
Reload the Altera using the driver tools or reboot to automatically reload
Use the ATP software and notice that the LEDs flash together instead of separately.

As practice, it would be good to control the new LED bit with the Generic capability in the ATP driver. The address is offset 0 from 0x2000. The bit is 24. We did not remove the control bits nor the read-back capability so you will still be able to read-write to the base control register. The compiler will remove the unused led_cntl(1..3) signals. Please note that the generic access will be done in parallel with the driver calls. The driver will have a copy of the register in memory that is not consistent with the register programming when the same register is modified outside of the ”standard call”. For modified designs it is best to use new registers to avoid a synchronization issue.
The following register definitions are a repeat of what is in the base hardware manual. The definitions are repeated here for convenience.

**PCIALT_BASEA**

[0x2000 Main Control Register Port read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>Design ID [read only]</td>
</tr>
<tr>
<td>27-24</td>
<td>LED 3-0 [write only]</td>
</tr>
<tr>
<td>23</td>
<td>Lintn status - Asserted = ‘0’ [read only]</td>
</tr>
<tr>
<td>22</td>
<td>Local_int status - Asserted = ‘1’ [read only]</td>
</tr>
<tr>
<td>21</td>
<td>Force_int - cause an interrupt by enabling this bit</td>
</tr>
<tr>
<td>20</td>
<td>M_int_en - interrupt driver enable, master interrupt enable</td>
</tr>
<tr>
<td>19</td>
<td>FRX_LD</td>
</tr>
<tr>
<td>18</td>
<td>FRX Reset</td>
</tr>
<tr>
<td>17</td>
<td>PLL enable</td>
</tr>
<tr>
<td>16-9</td>
<td>s2 data</td>
</tr>
<tr>
<td>8</td>
<td>i2c Data</td>
</tr>
<tr>
<td>7-0</td>
<td>i2c clock PLL 7 - 0</td>
</tr>
</tbody>
</table>

**FIGURE 4**  PCI-ALTERA-485/LVDS ALTERA BASE CONTROL REGISTER

The PCI-Altera-485/LVDS has 8 PLL devices which are programmed to produce the desired frequency with an i2c bus. Each PLL has a common data pin and independent clocks and upper "data" bit.

The common data line has a pull-up on the board. When the PLL is enabled and the i2c data bit is set to ‘0’ in the register the external line is driven low. When not enabled or i2c data is set to ‘1’ in the register, the external line is tri-stated and pulled-up by the resistor. For a read operation the data should be set to ‘1’ to allow the pll to drive the line.

The upper selection bit can be set in the register and directly driven to the PLLs. With separate i2c clock lines each of the PLLs can be programmed independently.

The clock line for the PLL to be programmed is toggled along with the data to create a bit stream with a “software clock”. Set the bit to the next state and toggle the clock line and repeat.

To read over the i2c bus a command is first written and then the bus read for the response. The i2c data register bit contains the state of the bus when read. The software will toggle the clock line and when the low-to-high transition is made, read the data bit then repeat until the message is captured.

The engineering kit contains the logic and software required to program the PLL and to read the programmed frequency back. The software to determine the frequency command words is available from Cypress Semiconductor. The part
number is CY22393FC. Cypress has a utility available for calculating the frequency control words for the PLLs. [http://www.dyneng.com/CyberClocks.zip](http://www.dyneng.com/CyberClocks.zip) is the URL for the Cypress software used to calculate the PLL programming words.

The PLLs respond to one of two addresses [only one works]. As part of our ATP our software determines the address of each PLL and prints it out. A label is attached to the shipping bag with the PLL addresses for the user's convenience. The software is part of the engineering kit and can be ported to your application.

When FRX Reset = ‘1’ all of the receive FIFOs are reset. The default = ‘0’ and is used for normal operation.

When FRX_LD is set to ‘1’ the programmable flag for the receive FIFOs can be accessed. Make FRX_LD = ‘0’ for normal operation.

The TX FIFOs are controlled from the Xilinx side of the interface. The RX FIFOs are controlled from the Altera side of the interface.

To guarantee proper operation, the FIFOs should be reset after power up. Set the reset control bit and then reset the control bit. The reset should be applied after the clocks are stable. The reset must be “seen” by the FIFO. Apply the reset condition for a period longer than the clock period used on the FIFOs.

To program the FIFO [PAF] flag first set the FRX_LD bit, then set and clear the FOUT_Reset bit. The next sequence of data written to the FIFOs will program the flags. When the sequence is completed take FRX_LD low. Please note that you do not reset the part again. If reset occurs after programming the flags, the flags will revert to the default values.

The 4k part used on the PCI-Altera-485/LVDS has a 12 bit range for the PAE and PAF flags. The PAE flag is written first. The default is 7. The lower 8 bits are written followed by the upper 4. Bit positions 7-0 and 3-0 respectively. The PAE flag is written to in order to get to the programming position for the PAF. The PAF value can be re-written by setting the FRX_LD bit and repeating the load sequence. Each of the RX FIFOs has an independent write port address allowing different values to be programmed for each channel.

M_INT_EN is the master interrupt enable for Altera. Please note that the PLX interrupt enable must also be enabled for a PCI interrupt to be generated. Default is disabled. When ‘1’ the master enable is “enabled”.

FORCE_INT when ‘1’ and the master enabled causes an interrupt to be generated. This bit is useful for software debugging.

The LEDs are enabled when the corresponding bit is set. The LEDs are for user purposes. One suggestion is to use them for debugging your Altera – the LEDs could be used to indicate what state the hardware is in or when a channel is active etc. The reference design uses them as register bits which could be used
for software interaction.

**PCIALT_RX_TX**

[0x2004 Rx/Tx Control Register Port read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Ax_link_stat [read only]</td>
</tr>
<tr>
<td>17</td>
<td>Ax_link_dir '1' = output, '0' = input</td>
</tr>
<tr>
<td>16</td>
<td>Ax_link_dat</td>
</tr>
<tr>
<td>15 - 8</td>
<td>strt_tx7-0</td>
</tr>
<tr>
<td>7 - 0</td>
<td>strt_rx7-0</td>
</tr>
</tbody>
</table>

**FIGURE 5**

PCI-ALTERA-485/LVDS ALTERA BASE CONTROL REGISTER

The start bits are used to enable and disable the Rx and Tx state machines from running in the reference design. The state machines move data from a Tx FIFO to the corresponding Rx FIFO provided the Tx is not empty and the Rx is not full.

The Ax_link line is a bi-directional line between the Xilinx and Altera. The Ax_link_dir signal controls the direction of the line on the Altera side [1 = output, 0 = input].

The Ax_link_dat defines the logic state when the line is configured as an output.

The Ax_link_stat signal is used to read the state of the line when it is configured as an input.

**PCIALT_DIR1,2**

[0x2008, 0x200C Direction Control Ports read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-0</td>
<td>dir39 - 20, dir19 - 0</td>
</tr>
</tbody>
</table>

**FIGURE 6**

PCI-ALTERA-485/LVDS DIRECTION CONTROL REGISTERS

The Direction Control registers are used to select transmit or receive on each of the RS-485 lines. The Altera internal circuitry and the differential transceiver are affected by the selections made. ‘1’ = transmit and ‘0’ = receive in a particular bit position. All IO are independent. All IO default to receive on reset / power on.
PCIALT_TERM1,2
[0x2010, 0x2014 Termination Control Ports read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-0</td>
<td>term39 - 20, term19 - 0</td>
</tr>
</tbody>
</table>

The Termination Control registers are used to select the state of the programmable termination on each of the RS-485 differential pairs. ‘1’ = activate termination and ‘0’ = isolate termination in a particular bit position. All IO are independent. All IO default to not terminated. Please note that the termination is independent of the direction.

PCIALT_IO1,2
[0x2018, 0x201C IO Control Ports read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-0</td>
<td>io39 - 20, io19 - 0</td>
</tr>
</tbody>
</table>

For each bit in the IO control register that has the corresponding Direction bit set to transmit the output is enabled. When enabled the data in the IO register is driven onto the IO line to the transceiver and out onto the differential pair. If the direction bit is not set the corresponding bit in the IO register is not driven onto the IO line. A read from the IO Register will return the value on the IO lines. If the direction is set to transmit then the value returned should match the value written to the register. If the direction is set to receive then the value returned will be set by the “system”. The software should map out the bits set to transmit when reading from this port.

Application Note: Spare IO

Frequently a system will need some dedicated IO but not all of the IO. If the registers are left in the design and the intermediate buses used to combine with your implementation the spare bits can be used as an auxiliary port with very little effort. Comment out the “io” definition for the lines that are under your state-machine control as well as the direction and termination bits [if you need real time control] then replace those definitions with your state-machine implementation.
PCIALT_FIFO_NA

[0x2020, 0x2024, 0x2028, 0x202C, 0x2030, 0x2034, 0x2038, 0x203C  Altera FIFO Ports read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-0</td>
<td>FIFO 7 - 0</td>
</tr>
</tbody>
</table>

FIGURE 9  
PCI-ALTERA-485/LVDS ALTERA FIFO PORTS

The Altera side of the RX can be written and the TX FIFOs can be read from these ports. Useful for loop-back testing. In normal operation the State-machine(s) within the Altera would be used to load data or read data from these ports.

PCIALT_TTL

[0x201C TTL Control Ports read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-0</td>
<td>TTL 11 - 0</td>
</tr>
</tbody>
</table>

FIGURE 10  
PCI-ALTERA-485/LVDS TTL CONTROL REGISTER

The TTL IO are designed with a ‘125 style gate and read-back buffer. The ‘125 provides an “open drain” tri-state gate. The PCI-Altera-485/LVDS has a pull-up on each line. When the gate is enabled the corresponding line is pulled low. When the gate is disabled the line is pulled-high with the pull-up.

The TTL port resets to "FFF" to cause the IO to be in the tri-stated condition. The software can write a ‘1’ or ‘O’ to any bit and cause the ‘1’ or ‘O’ on the IO line. If an external line is driving the IO bit then the line may remain at ‘O’ when set to the undriven state.

The read port returns the state of the IO lines – not necessarily the same as the write port.
PCIALT_FIFOSTAT1

[0x2044 FIFO Status Port read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 - 24</td>
<td>FTX FF 7 - 0</td>
</tr>
<tr>
<td>23 - 16</td>
<td>FTX MT 7 - 0</td>
</tr>
<tr>
<td>15 - 8</td>
<td>FRX FF 7 - 0</td>
</tr>
<tr>
<td>7 - 0</td>
<td>FRX MT 7 - 0</td>
</tr>
</tbody>
</table>

**FIGURE 11**

The status from the Rx and Tx FIFOs is available in this port. The status bits are active high. When ‘1’ the status is active and when ‘0’ the status is inactive.

FTX FF = FIFO Transmit Full Flag. When ‘1’ the FIFO associated with the bit is full. When ‘0’ the FIFO is not full.

FTX MT = FIFO Transmit Empty Flag. When ‘1’ the FIFO associated with the bit is empty. When ‘0’ the FIFO is not empty.

FRX FF = FIFO Receive Full Flag. When ‘1’ the FIFO associated with the bit is full. When ‘0’ the FIFO is not full.

FRX MT = FIFO Receive Empty Flag. When ‘1’ the FIFO associated with the bit is empty. When ‘0’ the FIFO is not empty.
The PLLn_abc bits when set ['1'] cause the corresponding IO bits to be controlled by the PLL clock as indicated. A clock waveform corresponding to the programmed PLL selected is driven to the IO. Please note that the direction register must also be programmed for the selected bits. The bits not selected are controlled by the IO registers. The upper bits: 39 – 24, are not affected by the pcialt485_osccnt register.

The CNT CLR bit when set clears the counters that count the PLL clock transitions. The CNT CLR function is used to reset the counters to a known state for functional test.

CNT EN when ‘1’ enables the counters to track the transitions of the PLL clocks. Each PLL output has a 32-bit counter. In addition a master counter based on the oscillator input frequency [66.6667 MHz] is available for reference. When bit 24 of the master counter transitions high [approximately 1/4 sec.], all counters stop counting. The counts can be read to verify the frequency of the output clock.

The MUX select bits are used to select the counter to read. There is only one port to read the 24 counters from. The MUX select bits select which one drives the OSC Data port. The decoding is a straight binary encoding of the counter
number corresponding to the control bit number in the Oscillator Control Register definition table. "00000" selects PLL0_a and "10000" selects PLL5_b etc. The master count is available on select 24.

**PCIALT_OSCDAT**

[0x204C Oscillator Data Port]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 - 0</td>
<td>Counter Value</td>
</tr>
</tbody>
</table>

**FIGURE 13**

PCI-ALTERA-485/LVDS OSCILLATOR DATA PORT

When read the OSCDAT port provides the data from the counter selected with the mux in the OSCDAT register. The ports are used for checking that the PLLs are functioning correctly. The Master clock value is captured in an additional counter. The relative counts of the different PLLs can be compared with the reference count and checked for proper operation.

**PCIALT_FIFOSTAT2**

[0x2044 FIFO Status Port read/write]

<table>
<thead>
<tr>
<th>DATA BIT</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 - 8</td>
<td>FTX PAE 7 - 0</td>
</tr>
<tr>
<td>7 - 0</td>
<td>FRX PAF 7 - 0</td>
</tr>
</tbody>
</table>

**FIGURE 14**

PCI-ALTERA-485/LVDS FIFO STATUS PORT2

The status from the Rx and Tx FIFOs is available in this port. The status bits are active high. When ‘1’ the status is active and when ‘0’ the status is inactive.

FTX PAE = FIFO Transmit Programmable Almost Empty flag. When ‘1’ the FIFO associated with the bit is almost empty. When ‘0’ the FIFO has more data than the Almost Empty level.

FRX PAF = FIFO Receive Programmable Almost Full. When ‘1’ the FIFO associated with the bit is almost full. When ‘0’ the FIFO has more room than the Almost Full level is set to.

All information provided is Copyright Dynamic Engineering