DYNAMIC ENGINEERING
150 DuBois, Suite C Santa Cruz, CA 95060
(831) 457-8891 **Fax** (831) 457-4793
www.dyneng.com
sales@dyneng.com
Est. 1988

alt_atp

Linux Device Driver Documentation

Revision A
Corresponding Hardware: Revision H
10-2002-0708
Corresponding Firmware: Altera ATP VHDL Revision F

alt_atp
Linux Device Driver for the
ATP Altera design on the
PCI-Altera-485/LVDS

Dynamic Engineering
150 DuBois, Suite C
Santa Cruz, CA 95060
(831) 457-8891
FAX: (831) 457-4793

Manual Revision A. Revised January 3, 2011.

Table of Contents

Introduction

The alt_atp driver is a Linux device driver for the ATP Altera design from Dynamic Engineering. This design is for the Altera EP20K400EBC652 FPGA on the PCI-Altera board. The Altera is programmed over the PCI interface with a configuration file from the host file system. The eight-bit Altera design ID field is then read from offset 0, bits 31-24 and the appropriate driver is loaded. The ID number for this design is 0x00. This Altera design is used to test the PCI-Altera hardware. The design allows all the various FIFO, IO, and PLL circuitry to be exercised, but is not suitable for practical application of the board.

The Altera controls 40 RS-485 or LVDS transceivers and 12 TTL I/O lines. There are also eight programmable PLLs with three clock outputs each that are programmed by the Altera and drive input pins on the Altera. Eight transmit FIFOs and eight receive FIFOs are connected between the Xilinx and the Altera to buffer data transfers for eight independent I/O channels.

An Altera design is treated as a hot-swappable child of the pci_alt parent. This means that different Altera configurations can be loaded at any time without powering down and a new Altera driver will be loaded automatically provided the design ID matches a known value. If the ID is not known, but the Altera loads successfully, a generic Altera driver will be loaded which allows the Rx FIFO resets and almost-full levels, LEDs, PLLs, and interrupts to be specifically controlled, but requires all other write accesses to use a structure that contains two unsigned long integer fields: an address offset and a data value field.

A handle to the current Altera driver can be obtained using the open() function call and passing in the appropriate device name (see below). IO Control calls (IOCTLs) are used to configure the Altera and read status.

Note

This documentation will provide information about all calls made to the driver, and how the driver interacts with the device for each of these calls. For more detailed information on the hardware implementation, refer to the PCI-Altera-user manual (also referred to as the hardware manual). The alt_atp driver was developed on Linux kernel version 2.6.35. If you are using a different version, some modification of the source code may be required.

Driver Installation

The source files and makefiles for the drivers and test applications are supplied in the driver archive file pci_altera.tar.bz2. Extract the directory structure to the computer where the driver is to be built.

From the top-level directory type “KBUILD_NOPEDANTIC=1 make” (necessary to suppress errors related to changing CFLAGS in Makefile). This will build the driver and test application object files. Next type “(sudo) make install” to copy the files to the target locations (/lib/modules/$(VERSION)/kernel/drivers/char/pci_alt/ for the drivers, /usr/local/bin/ for the test applications, /usr/lib/hotplug/firmware/ for the Altera configuration files, /sbin/hotplug/ for the firmware loader, and /etc/udev/rules.d/ for the configuration rule files) (you must be root for this to succeed). If so desired, type “make clean” to remove executable, object and interim files after installation.

Driver Startup

Install the hardware and boot the computer. Handles can be opened to a specific board by using the open() function call and passing in the appropriate device name.

Below is example code for opening a handle for device `dev_num`.

```
#typedef long HANDLE
#define       INPUT_SIZE 80

HANDLE   halt_atp;

char  Name[INPUT_SIZE];
int   dev_num;

do
{
   printf("\nEnter target board number (starting with zero): \n");
   scanf("%d", &dev_num);
   if(dev_num < 0 || dev_num > NUM_PCI_ALT_DEVICES)
      printf("\nTarget board number %d out of range!\n", dev_num);
}
while(dev_num < 0 || dev_num > NUM_PCI_ALT_DEVICES);

sprintf(Name, "/dev/alt_atp_%d", dev_num);
halt_atp = open(Name , O_RDWR);
if(halt_atp < 2)
{
   printf("\n%s FAILED to open!\n", Name);
   return 1;
}
```

IO Controls

The driver uses ioctl() calls to configure the device and obtain status. The parameters passed to the ioctl() function include the handle obtained from the open() call, an integer command that is defined in the API header file and an optional parameter used to pass data in and/or out of the device. If both input and output parameters are needed, a union of the two is used for the I/O parameter. The ioctl commands defined for the alt_atp driver are described below.

IOCTL_ALT_ATP_GET_INFO

Function: Returns the Driver Version and PLL device IDs.
Input: None
Output: ALT_ATP_DRIVER_DEVICE_INFO structure
Notes: The PLL device IDs are dynamically detected when the driver starts up. They are most likely 0x69, but could be the alternate ID value 0x6A.

IOCTL_ALT_ATP_SET_LEDS

Function: Controls the state of the four LEDs – A_Led0-3 on the upper right-hand corner of the board.
Input: Unsigned character
Output: None
Notes: A value of zero turns off all four LEDs. A one in a bit position 0..3 turns on the corresponding LED.

IOCTL_ALT_ATP_SET_CONFIG

Function: Sets the start bits for the receiver and transmitter state machines and the AX link signal direction and output value.
Input: RXTX_CONFIG structure
Output: None
Notes: See the structure and bit definitions in alt_atp_api.h for information on determining the structure field values.

IOCTL_ALT_ATP_GET_CONFIG

Function: Returns the configuration of the control register set in the previous call.
Input: None
Output: RXTX_CONFIG structure
Notes: See the structure and bit definitions in alt_atp_api.h for information on interpreting the structure field values.

IOCTL_ALT_ATP_SET_DIR

Function: Sets the direction of the 40 RS-485/LVDS IO lines.
Input: DTIO_BITS structure
Output: None
Notes: The input structure contains two long words; each of which controls 20 IO lines. See the structure definition in alt_atp_api.h for information on determining these values.

IOCTL_ALT_ATP_GET_DIR

Function: Returns the direction of the 40 RS-485/LVDS IO lines.
Input: None
Output: DTIO_BITS structure
Notes: The output structure contains two long words; each reports on 20 IO lines. See the structure definition in alt_atp_api.h for information on interpreting these values.

IOCTL_ALT_ATP_SET_TERM

Function: Sets the terminations on the 40 RS-485/LVDS IO lines.
Input: DTIO_BITS structure
Output: None
Notes: The input structure contains two long words; each controls 20 IO lines. See the structure definition in alt_atp_api.h for information on determining these values.

IOCTL_ALT_ATP_GET_TERM

Function: Returns the terminations on the 40 RS-485/LVDS IO lines.
Input: None
Output: DTIO_BITS structure
Notes: The output structure contains two long words; each reports on 20 IO lines. See the structure definition in alt_atp_api.h for information on interpreting these values.

IOCTL_ALT_ATP_SET_IO

Function: Sets the values driven onto the 40 RS-485/LVDS IO lines when they are configured as outputs.
Input: DTIO_BITS structure
Output: None
Notes: The input structure contains two long words; each controls 20 IO lines. See the structure definition in alt_atp_api.h for information on determining these values.

IOCTL_ALT_ATP_GET_IO

Function: Returns the values read from the 40 RS-485/LVDS IO lines.
Input: None
Output: DTIO_BITS structure
Notes: This call reads the actual I/O lines; depending on the value of the direction bit for the line in question, this may or may not match the value written in the previous call. The output structure contains two long words; each reports on 20 IO lines. See the structure definition in alt_atp_api.h for information on interpreting these values.

IOCTL_ALT_ATP_SET_TTL

Function: Sets the values of the 12 TTL lines.
Input: Unsigned long integer
Output: None
Notes: These are open drain lines that are pulled-up to +5 volts, therefore they must be set high in order to be used as inputs.

IOCTL_ALT_ATP_GET_TTL

Function: Returns the values read from the 12 TTL lines.
Input: None
Output: Unsigned long integer
Notes: These are open drain lines that are pulled-up to +5 volts, therefore they must be set high in order to be used as inputs, otherwise a low will be read regardless of any external values driven onto the bus line.

IOCTL_ALT_ATP_PUT_RX_DATA

Function: Loads an Rx data byte.
Input: RX_DATA_LOAD structure
Output: None
Notes: The RX_DATA_LOAD structure has two eight-bit fields: Channel – the number of the single receive FIFO to write to, and Data – the data byte to write.

IOCTL_ALT_ATP_GET_TX_DATA

Function: Reads a Tx data byte.
Input: Unsigned character
Output: Unsigned character
Notes: The number of the transmit FIFO to read from is passed to this command and a byte of data read from the specified channel's FIFO is returned

IOCTL_ALT_ATP_RESET_RX_FIFOS

Function: Resets the Rx FIFOs.
Input: None
Output: None
Notes: Resets all eight receive FIFOs.

IOCTL_ALT_ATP_SET_RX_LEVEL

Function: Sets an Rx FIFO almost full level.
Input: RX_LEVEL_LOAD structure
Output: None
Notes: The RX_DATA_LOAD structure has two fields: Channel – the 8-bit number of the single receive FIFO to write to, and Data – the 16-bit almost full level to write.

IOCTL_ALT_ATP_GET_FIFO_STATUS

Function: Returns the Rx and Tx FIFO level flags.
Input: None
Output: FIFO_STATUS structure
Notes: The FIFO_STATUS structure has six 8-bit fields. See the bit and structure definition in alt_atp_api.h for information on interpreting these values.

IOCTL_ALT_ATP_READ_PLL_DATA

Function: Returns the contents of a PLL's internal registers.
Input: Unsigned character
Output: PLL_READ structure
Notes: The channel number of the PLL to write to is passed in to this call and the register data is output in the PLL_READ structure in an array or 40 bytes. If channel is greater than seven, the first byte of the data array will return the value of the S2 bits from the eight PLLs.

IOCTL_ALT_ATP_LOAD_PLL_DATA

Function: Loads the internal registers of a PLL.
Input: PLL_LOAD structure
Output: None
Notes: The PLL_LOAD structure has two fields: Channel – the number of the PLL to write to, and Data – an array of 40 bytes containing the data to write. If channel is greater than seven, the first byte of data is written to the S2 bits for the eight PLLs.

IOCTL_ALT_ATP_SET_OSC_CONTROL

Function: Configures the oscillator counters and outputs.
Input: OSC_CONTROL structure
Output: None
Notes: The OSC_CONTROL structure has four fields:
OutEnables individually enables the 24 PLL inputs onto IO 0..23 when these lines are configured as outputs.
CountClear clears the counts of the 25 counters that count at the rates of the 24 PLL inputs and the reference oscillator.
CountEnable enables all the counters to count until the master counter (clocked by the reference oscillator) reaches a count of 0x1000000. At this point all counters stop counting and their counts can be read to verify the frequencies of the various PLL outputs.
MuxSelect selects which counter a IOCTL_ALT_ATP_READ_OSC_DATA call will read.

IOCTL_ALT_ATP_GET_OSC_CONTROL

Function: Returns the current oscillator counters and outputs configuration.
Input: None
Output: OSC_CONTROL structure
Notes: See the description of the previous call for information on this structure.

IOCTL_ALT_ATP_READ_OSC_DATA

Function: Reads and returns the value of the counter specified by MuxSelect in the IOCTL_ALT_ATP_SET_OSC_CONTROL call.
Input: None
Output: Unsigned long integer
Notes: The frequency of the PLL selected can be calculated by the following formula: Freq = count * 66.6667 MHz / 0x1000000.

IOCTL_ALT_ATP_WAIT_ON_INTERRUPT

Function: Inserts the calling process into a wait queue until an interrupt occurs.
Input: Time-out value in jiffies (unsigned long integer)
Output: None
Notes: This call is made in the user interrupt service routine to allow user-specified interrupt handlers for enabled interrupt conditions. The input parameter is a time-out value that causes the call to abort if the interrupt doesn't occur within the specified time. If the timeout is zero, the call will wait indefinitely.

IOCTL_ALT_ATP_ENABLE_INTERRUPT

Function: Enables the master interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to interrupts. The master interrupt enable is disabled in the driver interrupt service routine. This command must then be run again to re-enable it.

IOCTL_ALT_ATP_DISABLE_INTERRUPT

Function: Disables the master interrupt.
Input: None
Output: None
Notes: Used when interrupt processing is no longer desired.

IOCTL_ALT_ATP_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

Warranty and Repair

Dynamic Engineering warrants this product to be free from defects under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, Dynamic Engineering's sole responsibility shall be to repair, or at Dynamic Engineering's sole option to replace, the defective product.

Dynamic Engineering's warranty of and liability for defective products is limited to that set forth herein. Dynamic Engineering disclaims and excludes all other product warranties and product liability, expressed or implied, including but not limited to any implied warranties of merchantability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

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.

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing and in most cases it will be "cockpit error" rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer's making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer's invoicing policy.

Out of Warranty Repairs

Out of warranty support will be billed. The current minimum repair charge is $125. An open PO will be required.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois, Suite C
Santa Cruz, CA 95060
(831) 457-8891 - Fax (831) 457-4793
support@dyneng.com