

# **DYNAMIC ENGINEERING**

435 Park Dr., Ben Lomond, Calif. 95005

831-336-8891 Fax 831-336-3840

<http://www.dyneng.com>

[sales@dyneng.com](mailto:sales@dyneng.com)

Est. 1988

# **AlteraGeneric**

## **Driver Documentation**

### **Win32 Driver Model**

Revision C

Corresponding Hardware: Revision D

10-2002-0704

Corresponding Firmware: Revision A

**AltGeneric**  
WDM Device Driver for the  
AlteraGeneric Generic design for the  
PCI-Altera-485

Dynamic Engineering  
435 Park Drive  
Ben Lomond, CA 95005  
831- 336-8891  
831-336-3840 FAX

©2003 by Dynamic Engineering.

Trademarks and registered trademarks are owned by their respective  
manufacturers.  
Manual Revision C Revised December 15, 2004.

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.

This product has been designed to operate with compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.



---

---

# Table of Contents

---

---

<b>Introduction</b>	<b>4</b>
<b>Note</b>	<b>4</b>
<b>Driver Installation</b>	<b>4</b>
<b>Windows 2000 Installation</b>	<b>5</b>
<b>Windows XP Installation</b>	<b>5</b>
<b>Driver Startup</b>	<b>6</b>
<b>IO Controls</b>	<b>9</b>
IOCTL_ALTGENERIC_GET_INFO	9
IOCTL_ALTGENERIC_SET_LEDS	9
IOCTL_ALTGENERIC_READ_PLL_DATA	9
IOCTL_ALTGENERIC_LOAD_PLL_DATA	9
IOCTL_ALTGENERIC_PUT_DATA	10
IOCTL_ALTGENERIC_GET_DATA	10
IOCTL_ALTGENERIC_RESET_RX_FIFOS	10
IOCTL_ALTATP_SET_RX_LEVEL	10
IOCTL_ALTGENERIC_REGISTER_EVENT	10
IOCTL_ALTGENERIC_ENABLE_INTERRUPT	11
IOCTL_ALTGENERIC_DISABLE_INTERRUPT	11
IOCTL_ALTGENERIC_FORCE_INTERRUPT	11
 <b>WARRANTY AND REPAIR</b>	 <b>11</b>
<b>Service Policy</b>	<b>12</b>
Out of Warranty Repairs	12
<b>For Service Contact:</b>	<b>12</b>



## Introduction

The AltGeneric driver is a Win32 driver model (WDM) device driver for a generic Altera design from Dynamic Engineering. This design is for the Altera EP20K400EBC652 FPGA on the PCI-Altera-485 board. The Altera is programmed from the PCI interface with a configuration file resident on the host hard drive. The Altera design ID field is then read and the appropriate driver is loaded. The ID number for an unknown design defaults to 0xFF. If the ID is not registered, but the Altera loads successfully, this Altera driver will be loaded which allows the LEDs, PLLs, and interrupts to be specifically controlled, but requires all addresses above the base address to be accessed with a generic data structure that contains two unsigned long int fields: an address offset and a data field.

The Altera controls 40 RS-485 transceivers and 12 TTL I/O lines. There are also 8 programmable PLLs with three clock outputs each that are programmed by the Altera and connect to 24 clock 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.

The Altera design is treated as a hot-swappable child of the PciAlt parent. This means that different Altera configurations can be loaded at any time without powering down. A new Altera driver will be loaded automatically provided the design ID matches a known value. As stated above, if the ID is not known, but the Altera loads successfully, this generic Altera driver will be loaded.

A handle to the current Altera driver can be obtained using a CreateFile() call (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-485 user manual (also referred to as the hardware manual).

## Driver Installation

**NOTE: The PciAlt driver must be installed before any Altera design can be recognized and the appropriate driver loaded!**



There are several files provided in each driver package. These files include AltGeneric.sys, AlteraDesigns.inf, DDAltGeneric.h, AltGenericGUID.h, AltGenericDef.h, AGTest.exe, and AGTest source files.

## Windows 2000 Installation

Copy AlteraDesigns.inf and AltGeneric.sys to a floppy disk, or CD if preferred.

With the PCI-Altera-485 installed, power-on the PCI host computer and wait for the **Found New Hardware Wizard** dialogue window to appear.

- Select **Next**.
- Select **Search for a suitable driver for my device**.
- Select **Next**.
- Insert the disk prepared above in the desired drive.
- Select the appropriate drive e.g. **Floppy disk drives**.
- Select **Next**.
- The wizard should find the AlteraDesigns.inf file.
- Select **Next**.
- Select **Finish** to close the **Found New Hardware Wizard**.

## Windows XP Installation

Copy AlteraDesigns.inf to the WINDOWS\INF folder and copy AltGeneric.sys to a floppy disk, or CD if preferred. Right click on the AlteraDesigns.inf file icon in the WINDOWS\INF folder and select **Install** from the pop-up menu. This will create a precompiled information file (.pnf) in the same directory.

With the PCI-Altera-485 installed, power-on the PCI host computer and wait for the **Found New Hardware Wizard** dialogue window to appear, or select the Add Hardware control panel.

- Insert the disk prepared above in the appropriate drive.
- Select **Install from a list or specific location**
- Select **Next**.
- Select **Don't search. I will choose the driver to install**.
- Select **Next**.
- Select **Show All Devices** from the list
- Select **Next**.
- Select **Dynamic Engineering** from the Manufacturer list
- Select **Generic Altera Design** from the Model list
- Select **Next**.
- Select **Yes** on the Update Driver Warning dialogue box.



- Enter the drive e.g. **A:\** in the **Files Needed** dialogue box.
- Select **OK**.
- Select **Finish** to close the **Found New Hardware Wizard**.

The DDAltGeneric.h file is a C header file that defines the Application Program Interface (API) to the driver. The AltGenericGUID.h file is a C header file that defines the device interface identifier for the AltGeneric driver. These files are required at compile time by any application that wishes to interface with the AltGeneric driver. The AltGenericDef.h file contains the relevant bit defines for the AltGeneric registers. These files are not needed for driver installation.

The AGTest.exe file is a sample Win32 console application that makes calls into the AltGeneric driver to test the driver calls without actually writing any application code. It is not required during the driver installation. Open a command prompt console window and type **AGTest ñd0 -?** to display a list of commands (the AGTest.exe file must be in the directory that the window is referencing). The commands are all of the form **AGTest ñdn ñim** where **n** and **m** are the device number and the driver ioctl number respectively. This application is intended to test the proper functioning of the driver calls, not for normal operation.

## Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware.

A handle can be opened to a specific board by using the CreateFile() function call and passing in the device name obtained from the system.

The interface to the device is identified using a globally unique identifier (GUID), which is defined in AltGenericGUID.h.

Below is example code for opening a handle for device 0. The device number is underlined in the SetupDiEnumDeviceInterfaces call.

```
// The maximum length of the device name for a given interface
#define MAX_DEVICE_NAME 256
// Handle to the device object
HANDLE hAltGeneric = INVALID_HANDLE_VALUE;
// Return status from command
LONG status;
// Handle to device interface information structure
HDEVINFO hDeviceInfo;
// The actual symbolic link name to use in the createfile
CHAR deviceName[MAX_DEVICE_NAME];
// Size of buffer required to get the symbolic link name
DWORD requiredSize;
```



```

// Interface data structures for this device
SP_DEVICE_INTERFACE_DATA      interfaceData;
PSP_DEVICE_INTERFACE_DETAIL_DATA pDeviceDetail;

hDeviceInfo = SetupDiGetClassDevs((LPGUID)&GUID_DEVINTERFACE_ALTGENERIC,
                                  NULL,
                                  NULL,
                                  DIGCF_PRESENT | DIGCF_DEVICEINTERFACE);

if(hDeviceInfo == INVALID_HANDLE_VALUE)
{
    printf("***Error: couldn't get class info, (%d)\n",
           GetLastError());
    exit(-1);
}

interfaceData.cbSize = sizeof(interfaceData);

// Find the interface for device 0
if(!SetupDiEnumDeviceInterfaces(hDeviceInfo,
                                NULL,
                                (LPGUID)&GUID_DEVINTERFACE_ALTGENERIC,
                                0,
                                &interfaceData))
{
    status = GetLastError();
    if(status == ERROR_NO_MORE_ITEMS)
    {
        printf("***Error: couldn't find device(no more items), (%d)\n", 0);
        SetupDiDestroyDeviceInfoList(hDeviceInfo);
        exit(-1);
    }
    else
    {
        printf("***Error: couldn't enum device, (%d)\n",
               status);
        SetupDiDestroyDeviceInfoList(hDeviceInfo);
        exit(-1);
    }
}

// Get the details data to obtain the symbolic link name
if(!SetupDiGetDeviceInterfaceDetail(hDeviceInfo,
                                     &interfaceData,
                                     NULL,
                                     0,
                                     &requiredSize,
                                     NULL))
{
    if(GetLastError() != ERROR_INSUFFICIENT_BUFFER)
    {
        printf("***Error: couldn't get interface detail, (%d)\n",
               GetLastError());
        SetupDiDestroyDeviceInfoList(hDeviceInfo);
        exit(-1);
    }
}

```



```

// Allocate a buffer to get detail
pDeviceDetail = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(requiredSize);
if(pDeviceDetail == NULL)
{
    printf("***Error: couldn't allocate interface detail\n");
    SetupDiDestroyDeviceInfoList(hDeviceInfo);
    exit(-1);
}

pDeviceDetail->cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

// Get the detail info
if(!SetupDiGetDeviceInterfaceDetail(hDeviceInfo,
                                     &interfaceData,
                                     pDeviceDetail,
                                     requiredSize,
                                     NULL,
                                     NULL))
{
    printf("***Error: couldn't get interface detail(2), (%d)\n",
           GetLastError());
    SetupDiDestroyDeviceInfoList(hDeviceInfo);
    free(pDeviceDetail);
    exit(-1);
}

// Save the name
lstrcpyn(deviceName,
         pDeviceDetail->DevicePath,
         MAX_DEVICE_NAME);

// Cleanup search
free(pDeviceDetail);
SetupDiDestroyDeviceInfoList(hDeviceInfo);

// Open driver
// Create the handle to the device
hAltGeneric = CreateFile(deviceName,
                         GENERIC_READ | GENERIC_WRITE,
                         FILE_SHARE_READ | FILE_SHARE_WRITE,
                         NULL,
                         OPEN_EXISTING,
                         NULL,
                         NULL);

if(hAltGeneric == INVALID_HANDLE_VALUE)
{
    printf("***Error: couldn't open %s, (%d)\n", deviceName,
           GetLastError());
    exit(-1);
}

```





## IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object which controls a single board. IOCTLs are called using the Win32 function DeviceIoControl(), and passing in the handle to the device opened with CreateFile(). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used. The IOCTLs defined in this driver are as follows:

### IOCTL\_ALTGENERIC\_GET\_INFO

**Function:** Return the Driver Version and PLL device IDs.

**Input:** none

**Output:** DRIVER\_ALT\_DEVICE\_INFO structure

**Notes:** The PLL device IDs are dynamically detected when the driver starts up. They can be one of two values: 0x69 or 0x6A.

### IOCTL\_ALTGENERIC\_SET\_LEDS

**Function:** Controls the state of the four LEDs ñ A\_Led0-3 on the upper right-hand corner of the board.

**Input:** Unsigned char

**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\_ALTGENERIC\_READ\_PLL\_DATA

**Function:** Return the contents of a PLLís 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 struct in an array of 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\_ALTGENERIC\_LOAD\_PLL\_DATA

**Function:** Load the internal registers of a PLL.

**Input:** PLL\_LOAD structure

**Output:** none

**Notes:** The PLL\_LOAD structure has two fields: Channel ñ the number of the PLL to write to, and Data ñ 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\_ALTGENERIC\_PUT\_DATA**

**Function:** Writes one long word to the Altera memory space.

**Input:** ALT\_DATA\_LOAD structure

**Output:** none

**Notes:** The ALT\_DATA\_LOAD structure has two unsigned long int fields: Address: the address offset value from the Altera base address, and Data: the data value to write to the above address.

## **IOCTL\_ALTGENERIC\_GET\_DATA**

**Function:** Reads one long word from the Altera memory space.

**Input:** Unsigned long int

**Output:** Unsigned long int

**Notes:** As in the previous call the address offset value is passed into this call, but in this case the data value read from that address is returned.

## **IOCTL\_ALTGENERIC\_RESET\_RX\_FIFOS**

**Function:** Reset the Rx FIFOs.

**Input:** none

**Output:** none

**Notes:** Resets all eight receive FIFOs.

## **IOCTL\_ALTATP\_SET\_RX\_LEVEL**

**Function:** Set an Rx FIFO almost full level.

**Input:** RX\_LEVEL\_LOAD structure

**Output:** none

**Notes:** The RX\_DATA\_LOAD structure has two eight-bit fields: Channel  $\tilde{n}$  the number of the single receive FIFO to write to, and Data  $\tilde{n}$  the almost full level to write.

## **IOCTL\_ALTGENERIC\_REGISTER\_EVENT**

**Function:** Registers an event to be signaled when an interrupt occurs.

**Input:** Handle to Event object

**Output:** none

**Notes:** The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt.



## **IOCTL\_ALTGENERIC\_ENABLE\_INTERRUPT**

**Function:** Enable 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\_ALTGENERIC\_DISABLE\_INTERRUPT**

**Function:** Disable the master interrupt.

**Input:** none

**Output:** none

**Notes:** Used when interrupt processing is no longer desired.

## **IOCTL\_ALTGENERIC\_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 a 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  
435 Park Dr.  
Ben Lomond, CA 95005  
831-336-8891  
831-336-3840 fax  
support@dyneng.com

All information provided is Copyright Dynamic Engineering

