DYNAMIC ENGINEERING
435 Park Dr., Ben Lomond, Calif. 95005
831-336-8891 **Fax** 831-336-3840
http://www.dyneng.com
sales@dyneng.com
Est. 1988

PmcPario

Driver Documentation

Win32 Driver Model

Revision A
Corresponding Hardware: Revision E
10-1999-0105

PmcPario
WDM Device Drivers for the
PMC-Parallel-IO Pmc Module

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

Manual Revision A. Revised November 9, 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 Engineeringís 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 IP Module carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Introduction

The PmcPario driver is a Win32 driver model (WDM) device driver for the PMC-Parallel-IO board from Dynamic Engineering. Each PMC-Parallel-IO board implements a parallel interface using 64 open-collector TTL I/O drivers. A separate Device Object controls each PMC-Parallel-IO board, and a separate handle references each Device Object. IO Control calls (IOCTLs) are used to configure the board and to transfer data to and from the deviceís parallel interface.

Note

This documentation will provide information about all calls made to the driver, and how the driver interacts with the hardware for each of these calls. For more detailed information on the hardware implementation, refer to the PMC-Parallel-IO device user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include PmcPario.sys, PmcPario.inf, DDPmcPario.h, PmcParioGUID.h, PmcParioDef.h, PParioTest.exe, and PParioTest source files.

Windows 2000 Installation

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

With the hardware 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 PmcPario.inf file.
- Select ***Next***.
- Select ***Finish*** to close the ***Found New Hardware Wizard***.

Windows XP Installation

Copy PmcPario.inf to the WINDOWS\INF folder and copy PmcPario.sys to a floppy disk, or CD if preferred. Right click on the PmcPario.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.

Note: The INF folder is hidden by default, you must select ***Show hidden files and folders*** in the ***Tools/Folder Options/View*** menu selection in Windows Explorer to access this folder.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear. The **PMC-Parallel-IO** should be named in the dialogue box. Follow the steps below:

- Insert the disk prepared above in the appropriate drive.
- Select ***Install from a list or specific location***
- Select ***Next***
- Select ***Donít 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 ***PMC-Parallel-IO Device*** 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***.

This process must be completed for each new device that is installed.

The DDPmcPario.h file is the C header file that defines the Application Program Interface (API) to the driver. The PmcParioGUID.h file is a C header file that defines the device interface identifier for the PmcPario. These files are required at compile time by any application that wishes to interface with the PmcPario driver. The PmcParioDef.h file contains the relevant bit defines for the PMC-Parallel-IO registers. These files are not needed for driver installation.

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

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

```
// The maximum length of the device name for
//  a given instance of an interface
#define MAX_DEVICE_NAME 256
// Handle to the device object
HANDLE                           hPmcPario = 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_PMCPARIO,
                                  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_PMCPARIO,
                                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 and Create the handle to the device
hPmcPario = CreateFile(deviceName,
                      GENERIC_READ    | GENERIC_WRITE,
                      FILE_SHARE_READ | FILE_SHARE_WRITE,
                      NULL,
                      OPEN_EXISTING,
                      NULL,
                      NULL);

if(hPmcPario == 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 and pass data in and out. IOCTLs refer to a single Device Object in the driver, 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.

IOCTL_PMCPARIO_GET_INFO

Function: Returns the current driver version and instance number.
Input: none
Output: PMCPARIO_DDINFO structure
Notes: This call does not access the hardware, only driver parameters. See DDPmcPario.h for the definition of PMCPARIO_DDINFO.

IOCTL_PMCPARIO_SET_OUT_DATA

Function: Sets the value of the TTL outputs on the board.
Input: PMCPARIO_DATA structure
Output: none
Notes: The input data structure has two unsigned long int fields, LoWord and HiWord. These correspond to the 64 TTL lines on the board.

IOCTL_PMCPARIO_GET_OUT_DATA

Function: Returns the state of the TTL outputs in the output data register.
Input: none
Output: PMCPARIO_DATA structure
Notes: This call returns the state of the output data registers on the board. The drivers are open collector with pull-up resistors, therefore if an IO line is being driven externally the actual value of the IO bus may not match this value.

IOCTL_PMCPARIO_READ_IN_DATA

Function: Reads the input/output data bus directly.
Input: none
Output: PMCPARIO_DATA structure
Notes: This call reads the input data from the TTL input lines and returns a PMCPARIO_DATA structure that reports the state of the 64 TTL IO bus lines.

IOCTL_PMCPARIO_SET_CLOCK_CONFIG

Function: Sets the clock configuration parameters.
Input: PMCPARIO_CLOCK_CONFIG structure
Output: none
Notes: Controls the frequency of the internally generated clock, the state of the internal clock enable, and selects the internal or external source for the clock and clock enable. This clock and enable are used to clock the bus data value into the data read-back registers accessed in the IOCTL_PMCPARIO_READ_IN_DATA call.

IOCTL_PMCPARIO_GET_CLOCK_CONFIG

Function: Returns the configuration of the clock control register.
Input: none
Output: PMCPARIO_CLOCK_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCPARIO_SET_INT_CONFIG

Function: Sets interrupt configuration parameters.
Input: PMCPARIO_INT_CONFIG structure
Output: none
Notes: Enables and controls the behavior of the two interrupts connected to bit 0 and 1 of the IO bus data. These interrupts can be individually enabled and configured to respond to a high or low data value or a rising or falling edge on the corresponding data line.

IOCTL_PMCPARIO_GET_INT_CONFIG

Function: Returns the configuration of the interrupt control register.
Input: none
Output: PMCPARIO_INT_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCPARIO_GET_INT_STATUS

Function: Returns the control/status bits in the Plx ICS register.
Input: none
Output: unsigned long int
Notes: The Plx-9052 interrupt control/status bits are read by this call. See PmcParioDef.h for the bit definitions.

IOCTL_PMCPARIO_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 an interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. When it is desired to un-register the event, set the event handle input parameter to NULL.

IOCTL_PMCPARIO_ENABLE_INTERRUPT

Function: Enables the interrupts in the Plx-9052.
Input: none
Output: none
Notes: Sets the Plx interrupt enables. This IOCTL is used in the user interrupt processing function to begin interrupt processing or to re-enable the interrupts after they were disabled in the driver interrupt service routine.

IOCTL_PMCPARIO_DISABLE_INTERRUPT

Function: Disables the Plx-9052 interrupts.
Input: none
Output: none
Notes: Clears the Plx interrupt enables. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_PMCPARIO_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: none
Output: none
Notes: Causes an interrupt to be asserted on the PCI bus provided the interrupts are enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_PMCPARIO_GET_ISR_STATUS

Function: Returns the Plx-9052 interrupt status read in the last ISR.
Input: none
Output: unsigned long int
Notes: The status contains the status bits of the Plx ICS register read in the last ISR execution.

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options. http://www.dyneng.com/warranty.html

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 ìcockpit errorî 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 customerís 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 customerís 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