DYNAMIC ENGINEERING

150 DuBois, Suite B & C

Santa Cruz, CA 95060
(831) 457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

PMC-Parallel-485
Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation
Ver1.19

Revision 01p1 9/16/22
PMC: 10-1999-0305

PMC-Parallel-485
WDF Device Drivers for
PMC-Parallel-485

Dynamic Engineering
150 DuBois, Suite B & C
Santa Cruz, CA 95060
(831) 457-8891

Table of Contents

Introduction

The PMC-Parallel-485 driver was developed with the Windows Driver Foundation version 1.19 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

“PMC-Parallel-485” features a Spartan II Xilinx FPGA to implement the PCI interface, and IO processing, control and status for 32 differential IO. 50 MHz reference can be used for the clock divider.

UserAp is a stand-alone code set with a simple and powerful menu plus a series of tests that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering. The test software can be ported to your application to provide a running start. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system.

UserAp is delivered with multiple example tests plus 2 utilities that may prove useful in your debugging / integration. At the end of the UserAp menu is an item “Print Registers”. When executed – select the appropriate “test” number in the menu – the current contents of the registers are displayed. The structure for the Base register is shown with the structure selection and current status. The remainder are shown as hex numbers.

The second utility is “Modify Registers”. This utility allows the user to select a register to change, shows the current contents and allows the user to change the contents. The utility will allow multiple changes to the same register, switching to a new register. For example, one can enable selected outputs, and set and change the IO definition. In addition, there are quick start options to set the HW up to output or input. Modify Registers can be used to create patterns on the IO to interact with your system or to read the current state of your system IO.

When PMC-Parallel-485 is recognized by the PCI bus configuration utility it will start the PMC-Parallel-485 driver to allow communication with the device. IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move blocks of data in and out of the device.

Note - This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. ***For more detailed information on the hardware implementation,*** refer to the PMC-Parallel-485 user manual as appropriate (also referred to as the hardware manual).

There are several files provided in each driver package. These files include PmcPar485Public.h, PmcPar485.inf, pmcpar485.cat, and PmcPar485.sys. These files are in a folder within the UserAp file set.

PmcPar485Public.h is the C header file that defines the Application Program Interface (API) for the PMC-Parallel-485 driver. This file is required at compile time by any application that wishes to interface with the drivers, but is not needed for driver installation. This file is included with the UserAp file set.

Windows 10 Installation

Copy PmcPar485.inf, pmcpar485.cat, and PmcPar485.sys (Win10 version) to a CD, USB memory device, or local directory as preferred.

With the PMC-Parallel-485 hardware installed, power-on the host computer.
- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an ***Other PCI Bridge Device****.
- Right-click on the ***Other PCI Bridge Device*** and select ***Update Driver Software***.
- Select ***Browse my computer for driver software***.
- Select ***Navigate to the folder or device. If at the root select the sub folders button.***
- Select ***Next***.
- Select ***Close*** to close the update window.

The system should now display the PMC-Parallel-485 adapter in the Device Manager.

***** If the ***Other PCI Bridge Device*** is not displayed, click on the ***Scan for hardware changes*** icon on the tool-bar.

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 globally unique identifiers (GUID), which are defined in PmcPar485Public.h. See main.c in the PmcPar485UserAp project for an example of how to acquire a handle to the device.

The main file provided is designed to work with our test menu and includes user interaction steps to allow the user to select which board is being tested in a multiple board environment. The integrator can hardcode for single board systems or use an automatic loop to operate in multiple board systems without using user interaction. For multiple user systems it is suggested that the board number is associated with a switch setting so the calls can be associated with a particular board from a physical point of view.

IO Controls

The drivers use IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single board or I/O channel. IOCTLs are called using the Win function DeviceIoControl(), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with CreateFile()
  DWORD         dwIoControlCode, // Control code defined in API header
file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned, // Pointer to return length parameter
  LPOVERLAPPED lpOverlapped,     // Optional pointer to overlapped
structure
);                                //   used for asynchronous I/O
```

The IOCTLs defined for the PMC-Parallel-485 driver are described below:
23 currently defined.

IOCTL_PMC_PAR_485_GET_INFO
Function: Returns the device driver version, Xilinx flash revision, user switch value, Type, and device instance number.
Input: None
Output: PMC_PAR_485_DRIVER_DEVICE_INFO structure
Notes: The switch value is the configuration of the 8-bit onboard dipswitch selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. Revision Major and Revision Minor represent the current Flash revision Major. Minor. See the definition of PMC_PAR_485_DRIVER_DEVICE_INFO below.

```
typedef struct _PMC_PAR_485_DRIVER_DEVICE_INFO {
   ULONG    InstanceNumber;
   UCHAR    DriverVersion;
   UCHAR    RevisionMajor;
   UCHAR    RevisionMinor;
   UCHAR    SwitchValue;
} PMC_PAR_485_DRIVER_DEVICE_INFO, *PPMC_PAR_485_DRIVER_DEVICE_INFO;
```

IOCTL_PMC_PAR_485_SET_BASE_CONFIG

Function: Set the base register configuration
Input: Base register parameters PMC_PAR_485_BASE_CONFIG
Output: None
Notes: See definition of PMC_PAR_485_BASE_CONFIG & CLK_PRE_SEL below. See Hardware manual for control bit map.

IOCTL_PMC_PAR_485_GET_BASE_CONFIG

Function: Return the base register configuration
Input: None
Output: Base register parameters PMC_PAR_485_BASE_CONFIG
Notes: See definition of PMC_PAR_485_BASE_CONFIG & CLK_PRE_SEL below. See Hardware manual for control bit map.

```
typedef struct _PMC_PAR_485_BASE_CONFIG {
       USHORT   ClkDiv;        // bits 11-0  Clock Rate Divisor
       BOOLEAN  ClkPostSel;    // bit 12 '1' for Divided clock else input selected
       CLK_PRE_SEL   ClkPreSel;// unsigned int ClkPreSel : 2;
                               // bits 14-13 00 = 0, 01 = osc, 10 = ext, 11 = PCI
       BOOLEAN  ClkExtDir;     // bit 15 '1' = out
       BOOLEAN  ClkEnInt;      // bit 16 1 = enable
       BOOLEAN  ClkEnSel;      // bit 17 1 = use External Clock enable, 0 =Internal

} PMC_PAR_485_BASE_CONFIG, * PPMC_PAR_485_BASE_CONFIG;


typedef enum _CLK_PRE_SEL {
       CLK_STATIC,
       CLK_OSC,
       CLK_EXT,
       CLK_PCI
} CLK_PRE_SEL, * PCLK_PRE_SEL;
```

IOCTL_PMC_PAR_485_REGISTER_EVENT

Function: Register an Event object 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.

IOCTL_PMC_PAR_485_ENABLE_INTERRUPT

Function: Enable the user interrupts - Set Master Int Enable
Input: None
Output: None
Notes: 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_PMC_PAR_485_DISABLE_INTERRUPT

Function: Disable the Master Interrupt enable
Input: None
Output: None
Notes: This IOCTL is used when interrupt processing is no longer desired.

IOCTL_PMC_PAR_485_FORCE_INTERRUPT

Function: Cause an interrupt
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_PMC_PAR_485_SET_EDGE_LEVEL

Function: Writes a value to the edge/level registers
Input: ULONG
Output: None
Notes: When an edge enable bit is a '1', the corresponding data bit will be latched when an edge occurs, provided its mask bit is enabled. If the bit is a '0', the level of the signal is used. The polarity of the edge or level depends on the state of the corresponding bit in the polarity register.

IOCTL_PMC_PAR_485_GET_EDGE_LEVEL

Function: Reads the value from the edge/level registers
Input: None
Output: ULONG
Notes: When an edge enable bit is a '1', the corresponding data bit will be latched when an edge occurs, provided its mask bit is enabled. If the bit is a '0', the level of the signal is used. The polarity of the edge or level depends on the state of the corresponding bit in the polarity register.

IOCTL_PMC_PAR_485_SET_INT_MASK

Function: Writes a value to the interrupt enable registers
Input: ULONG
Output: None
Notes: Input bits can be monitored and kept from being active by setting the corresponding mask bit.

IOCTL_PMC_PAR_485_GET_INT_MASK

Function: Reads the value from the interrupt enable registers
Input: None
Output: ULONG
Notes: Input bits can be monitored and kept from being active by setting the corresponding mask bit.

IOCTL_PMC_PAR_485_SET_POLARITY

Function: Write a value to the polarity registers
Input: ULONG
Output: None
Notes: If an input bit is active low then the corresponding polarity bit should be set to '1'. The data input latch will capture the level of a signal. Active low signals should be inverted to capture the active state.

IOCTL_PMC_PAR_485_GET_POLARITY

Function: Read the value from the polarity registers
Input: None
Output: ULONG
Notes: If the corresponding edge enable bit is set, then a '1' indicates a falling edge and a '0' indicates a rising edge. 1=invert, 0=normal.

IOCTL_PMC_PAR_485_SET_DIR_TERM

Function: Write a value to the direction termination registers
Input: ULONG
Output: None
Notes: Value of the termination registers sets the direction for each of the 32 differential pairs

IOCTL_PMC_PAR_485_GET_DIR_TERM

Function: Read the value from the direction Termination registers
Input: None
Output: ULONG
Notes: See hardware manual for PMC-Parallel-485 direction termination control bit map.

IOCTL_PMC_PAR_485_SET_DATA_OUT

Function: Writes a value to the output data registers
Input: ULONG
Output: None
Notes: Sets the 32 bits of the Xilinx internal register, and appears on the output diver within two clock periods of being written.

IOCTL_PMC_PAR_485_GET_DATA_OUT

Function: Reads the value from the output data registers
Input: None
Output: ULONG
Notes: pmc_par485_dataout is a Xilinx internal register, and which outputs data written to it after two clock periods of being written.

IOCTL_PMC_PAR_485_READ_DIRECT

Function: Read the direct input data
Input: None
Output: ULONG
Notes: Returns the “natural” data located at the input port, unaltered by Polarity, Interrupt Enable, or Level bits.

IOCTL_PMC_PAR_485_READ_FILTERED

Function: Read the Filtered input data
Input: None
Output: ULONG
Notes: XIO after Polarity, Interrupt Enable, Level bits selected - active level interrupter.

IOCTL_PMC_PAR_485_GET_AND_CLEAR_ISR_STATUS

Function: Return the interrupt status read in the last ISR
Input: None
Output: PMC_PAR_485_ISR_STAT
Notes: PMC_PAR_485_ISR_STAT structure with Filtered data plus Status register.

IOCTL_PMC_PAR_485_GET_ISR_STATUS

Function: Return the interrupt status read in the last ISR
Input: None
Output: PMC_PAR_485_ISR_STAT
Notes: PMC_PAR_485_ISR_STAT structure with Filtered data plus Status register.

```
typedef struct _PMC_PAR_485_ISR_STAT {
     ULONG    InterruptStatus;
     ULONG    FilteredData;
} PMC_PAR_485_ISR_STAT, * PPMC_PAR_485_ISR_STAT;
```

IOCTL_PMC_PAR_485_GET_SWITCH

Function: Return the 8-bit switch configuration
Input: None
Output: PMC_PAR_485_USER_SWITCH
Notes: See hardware manual for PMC_PAR_485_USER_SWITCH register address.

IOCTL_PMC_PAR_485_SET_MASTER_INT_EN

Function: Enable the Master Interrupt
Input: DEVICE_EXTENSION
Output: None
Notes:

IOCTL_PMC_PAR_485_CLEAR_MASTER_INT_EN

Function: Disable the Master Interrupt
Input: DEVICE_EXTENSION
Output: None
Notes:

Utility Functions

Print Registers

Function: Displays 32-bit register contents in Hex notation and expands any register structs onto terminal screen.
Notes: See Hardware manual for breakdown of any register structure and bit mapping.

Modify Registers

Function: Lists each “Writable” register, and it’s 32-bit hex content for user to select from and modify. Displays each 32-bit “Read Only” register content in Hex notation.
Options:

- Set individual bits mapped to attributes of register struct
- Update entire register with new hex value from user input.
- Toggle register back and forth between two hex values.
- Apply default register values that enable specific tasks.
  Ex. Default Read, Default Write, Reset

Notes: See Hardware manual for breakdown of any register structure and bit mapping.

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.
https://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 "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 or e-mail and arrange to work with an engineer. We will work with you to determine the cause of the issue.

Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite B & C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com