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-NG1 Windows 10 WDF Driver Documentation

Developed with Windows Driver Foundation Ver1.19

Revision 1p1 01/03/23
PMC: 10-1999-0305

PMC-Parallel-485-NG1
WDF Device Drivers for
PMC-Parallel-485-NG1

Dynamic Engineering

150 DuBois, Suite B & C

Santa Cruz, CA 95060

(831) 457-8891

Table of Contents

Introduction

The PMC-Parallel-485-NG1 driver was developed with the Windows Driver Foundation version 1.19 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

“PMC-Parallel-485-NG1” 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. The NG1 version has some IO converted to special definitions [See HW manual] and two cascaded counters for generating system time.

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 examples that may prove useful in your debugging / integration.

When PMC-Parallel-485-NG1 is recognized by the PCI bus configuration utility it will start the PMC-Parallel-485-NG1 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-NG1 user manual as appropriate (also referred to as the hardware manual).

There are several files provided in each driver package. These files include PmcPar485Ng1_Public.h, PmcPara485Ng1.inf, PmcPara485Ng1.cat, and PmcPara485Ng1.sys. These files are in a folder within the UserAp file set.

PmcPar485Ng1_Public.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 PmcPara485.inf, PmcPara485.cat, and PmcPara485Ng1.sys (Win10/11 version) to a CD, USB memory device, or local directory as preferred.

With the PMC-Parallel-485-NG1 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_P485_NG1_GET_INFO
Function: Returns the device driver version, Xilinx flash revision, user switch value, Type, and device instance number.
Input: None
Output: PMC_P485_NG1_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_P485_NG1_DRIVER_DEVICE_INFO below.

```
typedef struct _PMC_P485_NG1_DRIVER_DEVICE_INFO {
      UCHAR    DriverRev;
      UCHAR    SwitchValue;
      UCHAR    DesignId;         // Firmware design identifier
      UCHAR    MajorRev;         // Firmware design revision
      UCHAR    MinorRev;         // Firmware design minor revision
      ULONG    InstanceNumber;
} PMC_P485_NG1_DRIVER_DEVICE_INFO, * PPMC_P485_NG1_DRIVER_DEVICE_INFO;
```

IOCTL_PMC_P485_NG1_SET_CLOCK32
Function: Set the 32 bit Clock Configuration
Input: PMC_PARA_485_NG1_CONFIG1
Output: None
Notes: See definition of PMC_PARA_485_NG1_CONFIG1 See Hardware manual for control bit map.

IOCTL_PMC_P485_NG1_GET_CLOCK32
Function: Return the Clock 32 register configuration
Input: None
Output: Clock1 parameters PMC_PARA_485_NG1_CONFIG1
Notes: See definition of PMC_PARA_485_NG1_CONFIG1 below. See Hardware manual for control bit map.

```
typedef enum _COS_PRESELECT {
      NO_CLOCK,    // clock is parked
      OSCILLATOR, // select local reference oscillator
      EXT_CLOCK,  // select external received clock
      PCI_CLOCK,  // select system PCI reference
} COS_PRESELECT, * PCOS_PRESELECT;

typedef enum _COS_ENABLESELECT {
      InternalEn, // Select to use the SW controlled Enable
      ExternalEn,   // Select to use the external enable to capture
data
} COS_ENABLESELECT, * PCOS_ENABLESELECT;

typedef enum _COS_RESETSOURCE {
      InternalCounter, // select to use the internal counters as the
reset source
      ExternalSource,  // select to use the externally received reset
} COS_RESETSOURCE, * PCOS_RESETSOURCE;

typedef enum _COS_POSTSELECT {
      InputClock,       // select for selected reference clock
      DividedClock,     // select for divided selected ref clock
} COS_POSTSELECT, * PCOS_POSTSELECT;

typedef struct _PMC_PARA_485_NG1_CONFIG1 {
      ULONG                  Divisor;
      COS_PRESELECT          PreSelClock;
      COS_POSTSELECT         PostSelClock;
      COS_RESETSOURCE        ResetSource;    //
      COS_ENABLESELECT       EnableSelect;     //
      BOOLEAN                ClkEnState; // When internal True for
'1', False for '0'
} PMC_PARA_485_NG1_CONFIG1, * PPMC_PARA_485_NG1_CONFIG1;
```

IOCTL_PMC_P485_NG1_SET_CLOCK14

Function: Set the 14 bit Clock Configuration
Input: PMC_PARA_485_NG1_CONFIG2
Output: None
Notes: See definition of PMC_PARA_485_NG1_CONFIG2 See Hardware manual for control bit map.

IOCTL_PMC_P485_NG1_GET_CLOCK14

Function: Return the 14 bit Clock register configuration
Input: None
Output: Clock1 parameters PMC_PARA_485_NG1_CONFIG2
Notes: See definition of PMC_PARA_485_NG1_CONFIG2 below. See Hardware manual for control bit map.

```
typedef enum _EDGE_SELECT {
      BIT5,
      BIT6,
      BIT7,
      BIT8,
      BIT9,
      BIT10,
      BIT11,
      BIT12,
} EDGE_SELECT, * PEDGE_SELECT;

typedef struct _PMC_PARA_485_NG1_CONFIG2 {
      ULONG                   Divisor;
      COS_PRESELECT           PreSelClock;
      COS_POSTSELECT          PostSelClock;
      COS_RESETSOURCE         ResetSource;
      EDGE_SELECT             EdgeDetection;
} PMC_PARA_485_NG1_CONFIG2, * PPMC_PARA_485_NG1_CONFIG2;
```

IOCTL_PMC_P485_NG1_SET_INT_CONT

Function: Set the Interrupt Control Register
Input: PMC_PARA_485_NG1_INT_CONT
Output: None
Notes: See definition of PMC_PARA_485_NG1_INT_CONT See Hardware manual for control bit map.

IOCTL_PMC_P485_NG1_GET_INT_CONT

Function: Return the Interrupt Control register contents
Input: None
Output: Interrupt Enables: PMC_PARA_485_INT_CONT
Notes: See definition of PMC_PARA_485_INT_CONT below. See Hardware manual for control bit map.

```
typedef struct _PMC_PARA_485_NG1_INT_CONT {
      BOOLEAN  EdgeEn;     // True to enable Edge Interrupt
      BOOLEAN  ResetEn1;   // True to enable Reset Counter 1 interrupt
      BOOLEAN  ResetEn2;   // True to enable Reset Counter 2 interrupt
      BOOLEAN  IO7MuxCntl; // True = Data, False = Edge
} PMC_PARA_485_NG1_INT_CONT, * PPMC_PARA_485_NG1_INT_CONT;
```

IOCTL_PMC_P485_NG1_GET_INT_STATUS

Function: Return the interrupt status register value
Input: None
Output: PMC_PARA_485_NG1_INT_STAT
Notes: Both Masked and Unmasked status available

```
typedef struct _PMC_PARA_485_NG1_INTSTAT {
      BOOLEAN  EdgeIntUnmasked;
      BOOLEAN  EdgeIntMasked;
      BOOLEAN  Rst1IntUnmasked;
      BOOLEAN  Rst1IntMasked;
      BOOLEAN  Rst2IntUnmasked;
      BOOLEAN  Rst2IntMasked;
      BOOLEAN  IntRequest;
      BOOLEAN  ForceIntActive;
} PMC_PARA_485_NG1_INTSTAT, * PPMC_PARA_485_NG1_INTSTAT;
```

IOCTL_PMC_P485_NG1_CLEAR_STATUS

Function: clear the sticky bits
Input: PMC_PARA_485_NG1_INTCLEAR
Output:
Notes:

```
typedef struct _PMC_PARA_485_NG1_INTCLEAR {
      BOOLEAN  EdgeInt;
      BOOLEAN  Rst1Int;
      BOOLEAN  Rst2Int;
} PMC_PARA_485_NG1_INTCLEAR, * PPMC_PARA_485_NG1_INTCLEAR;
```

IOCTL_PMC_P485_NG1_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_P485_NG1_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_P485_NG1_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_P485_NG1_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_P485_NG1_SET_DIR_TERM

Function: Write a value to the direction termination registers
Input: PMC_PARA_485_NG1_DIRTERM
Output: None
Notes:

IOCTL_PMC_P485_NG1_GET_DIR_TERM

Function: Read the value from the direction Termination registers
Input: None
Output: PMC_PARA_485_NG1_DIRTERM
Notes: See hardware manual for PMC-Parallel-485-NG1 direction termination control bit map.

```
typedef struct _PMC_PARA_485_NG1_DIRTERM {
      ULONG    Direction;
      ULONG    Termination;
} PMC_PARA_485_NG1_DIRTERM, * PPMC_PARA_485_NG1_DIRTERM;
```

IOCTL_PMC_P485_NG1_PUT_DATA_WORD

Function: Writes a value to the output data registers
Input: ULONG
Output: None

Notes: Sets the 32 bits of the FPGA internal register. Depending on the Direction and Special function bits the corresponding output IO will be set/cleared.

IOCTL_PMC_P485_NG1_GET_DATA_WORD

Function: Reads the value from the output data register
Input: None
Output: ULONG
Notes: Returns the Register version. See READ DATA WORD for IO version

IOCTL_PMC_P485_NG1_READ_DATA_WORD

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 etc.

IOCTL_PMC_P485_NG1_READ_FILTERED_DATA

Function: Read the Filtered input data
Input: None
Output: ULONG
Notes: After Polarity, Interrupt Enable applied. Latched data. See Clearing function

IOCTL_PMC_P485_NG1_CLEAR_LAT_DATA

Function: Clear the corresponding bits in the latched data register
Input: ULONG
Output:
Notes: Typically used to clear the stored data after reading by writing back the same value. Clears the filtered data register.

IOCTL_PMC_P485_NG1_PRELOAD_COUNT32

Function: Writes a value to the preload register for the 32 bit counter
Input: ULONG[32 bits]
Output: None
Notes: Sets the Pre-Load register for the 32 bit counter.   This is the initial value used by the counter. Counts up and rolls over. Can be set close to max value to provide an offset to “0000” time. Can help with system initialization with multiple

devices.

IOCTL_PMC_P485_NG1_PRELOAD_COUNT14

Function: Writes a value to the preload register for 14 bit Counter
Input: ULONG [14 bits]
Output: None
Notes: Sets the Pre-Load register for the 14 bit counter. This is the initial value used by the counter. Counts up and rolls over. Can be set close to max value to provide an offset to "0000" time. Can help with system initialization with multiple devices.

IOCTL_PMC_P485_NG1_LOAD_COUNT32

Function: Move the value in the Preload register for 32 bit Counter into the counter
Input: None
Output: None
Notes: Count is updated when next Clock Reference edge occurs, can be some delay after load command supplied [if slow reference selected]

IOCTL_PMC_P485_NG1_READ_COUNT32

Function: Reads the current count on the 32 bit Counter
Input: None
Output: ULONG
Notes:

IOCTL_PMC_P485_NG1_READ_EDGE_COUNT

Function: Read value stored when edge detected
Input: None
Output: ULONG
Notes: 32 bit counter value from counter 1 when Edge Detect asserted. Updated for each edge detected. Most recent value returned.

IOCTL_PMC_P485_NG1_LOAD_COUNT14

Function: Move the value in the Preload register for 14 bit Counter into the Counter
Input: None
Output: None
Notes: Count is updated when next Clock Reference edge occurs, can be some delay after load command supplied [if slow reference selected]

IOCTL_PMC_P485_NG1_READ_COUNT14

Function: Reads the current count on the 14 bit Counter
Input: None
Output: ULONG
Notes:

IOCTL_PMC_P485_NG1_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_P485_NG1_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_P485_NG1_CLRFORCE_INTERRUPT

Function: Clear the Force Interrupt bit. Also cleared in ISR.
Input: None
Output: None
Notes: IOCTL provided for test software and debugging when Force Int may not be cleared properly. Not used for standard software applications.

IOCTL_PMC_P485_NG1_GET_ISR_STATUS

Function: Return value of status register when ISR invoked
Input: DEVICE_EXTENSION
Output: ULONG
Notes: See bit map in Public file.

IOCTL_PMC_P485_NG1_GET_RSW

Function: Return the 8-bit switch configuration
Input: None
Output: PMC_P485_NG1_REVSPECSW
Notes: See hardware manual for PMC_P485_NG1_USER_SWITCH register address details.

```
typedef struct _PMC_P485_NG1_REVSPECSW {
      UCHAR    SwitchValue;   // current Switch value
      UCHAR    MajorRev;            // Firmware design revision
      UCHAR    MinorRev;            // Firmware design minor revision
      UCHAR    Special;       // Special Bits (0)ClkExt, (1)ClkEnExt
} PMC_P485_NG1_REVSPECSW, * PPMC_P485_NG1_REVSPECSW;
```

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