

DYNAMIC ENGINEERING

150 DuBois, Suite C

Santa Cruz, CA 95060

(831) 457-8891 **Fax** (831) 457-4793

<http://www.dyneng.com>

sales@dyneng.com

Est. 1988

PMC Biserial S311

Software Manual

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Revision B
10-2000-0101

PmcBisS311
WDF Device Drivers for the
PMC-BiSerial-S311

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

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 PMC carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.

©2018 by Dynamic Engineering.

Trademarks and registered trademarks are owned by their respective manufactures.
Manual Revision B. Revised October 2, 2018.



Table of Contents

INTRODUCTION	4
DRIVER INSTALLATION	6
Windows 7 Installation	6
IO Controls	7
IOCTL_PMC_BIS_S311_GET_INFO	8
IOCTL_PMC_BIS_S311_SET_CONFIG	8
IOCTL_PMC_BIS_S311_GET_CONFIG	8
IOCTL_PMC_BIS_S311_GET_STATUS	9
IOCTL_PMC_BIS_S311_SET_TX_CONFIG	9
IOCTL_PMC_BIS_S311_GET_TX_CONFIG	10
IOCTL_PMC_BIS_S311_SET_RX_CONFIG	10
IOCTL_PMC_BIS_S311_GET_RX_CONFIG	11
IOCTL_PMC_BIS_S311_SET_FIFO_LEVELS	11
IOCTL_PMC_BIS_S311_GET_FIFO_LEVELS	12
IOCTL_PMC_BIS_S311_SET_TERMINATIONS	12
IOCTL_PMC_BIS_S311_GET_TERMINATIONS	12
IOCTL_PMC_BIS_S311_START_TX	13
IOCTL_PMC_BIS_S311_START_RX	13
IOCTL_PMC_BIS_S311_STOP_TX	13
IOCTL_PMC_BIS_S311_STOP_RX	13
IOCTL_PMC_BIS_S311_GET_RX_COUNT	13
IOCTL_PMC_BIS_S311_PUT_TX_DATA	14
IOCTL_PMC_BIS_S311_GET_RX_DATA	14
IOCTL_PMC_BIS_S311_GET_TX_DATA	14
IOCTL_PMC_BIS_S311_PUT_RX_DATA	14
IOCTL_PMC_BIS_S311_RESET_FIFOS	14
IOCTL_PMC_BIS_S311_REGISTER_EVENT	15
IOCTL_PMC_BIS_S311_ENABLE_INTERRUPT	15
IOCTL_PMC_BIS_S311_DISABLE_INTERRUPT	15
IOCTL_PMC_BIS_S311_FORCE_INTERRUPT	15
IOCTL_PMC_BIS_S311_GET_ISR_STATUS	16
Write	17
Read	17
WARRANTY AND REPAIR	18
Service Policy	18
Support	18
For Service Contact:	18



Introduction

The PmcBisS311 driver was developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF). The PMC-BiSerial-S311 board has a Spartan-40XL Xilinx FPGA to implement the PCI interface, protocol control and status for one transmit and one receive channel. The board also has an onboard 10 MHz oscillator to provide a timing reference and twenty RS-485 line driver/receivers.

Nine of the twenty RS-485 transceivers are used in this design, four as drivers and five as receivers. There are also two sets of 4K by 32-bit FIFOs, one for the transmitter and one for the receiver. These are used to buffer bi-directional data between the PCI bus and the I/O state machines.

The PmcBisS311 driver package has two parts. The driver is installed into the Windows® OS and the User Application “Userapp” executable.

The driver and test are delivered as installed or executable items to be used directly or indirectly by the user. The Userapp code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

UserApp 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 regtest's 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.

When the PMC-BiSerial-S311 is recognized by the PCI bus configuration utility it will start the PBS311 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 IO user manual (also referred to as the hardware manual).



Driver Installation

There are several files provided in each driver package. These files include PmcBisS311Public.h, PmcBisS311.inf, pmcbiss311.cat, PmcBisS311.sys, and WdfColInstaller01009.dll.

PmcBisS311Public.h is the C header file that defines the Application Program Interface (API) for the PmcBisS311 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.

Windows 7 Installation

Copy PmcBisS311.inf, pmcbiss311.cat, PmcBisS311.sys, and WdfColInstaller01009.dll (Win7 version) to a floppy disk, CD or USB memory device as preferred.

With the PMC Parallel IO hardware installed, power-on the PCI 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**.
- Insert the disk or memory device prepared above in the desired drive.
- Select **Browse my computer for driver software**.
- Select **Let me pick from a list of device drivers on my computer**.
- Select **Next**.
- Select **Have Disk** and enter the path to the device prepared above.
- Select **Next**.
- Select **Close** to close the update window.

The system should now display the PmcBisS311 PCI 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 PmcBisS311Public.h. See main.c in the PmcBisS311UserApp 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 Win32 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 BISERIAL S311 driver are described below:

IOCTL_PMC_BIS_S311_GET_INFO

Function: Returns the current driver version and instance number.

Input: none

Output: PMC_BIS_S311_INFO structure

Notes: This call does not access the hardware, only driver parameters. See the definition of PMC_BIS_S311_INFO below.

```
typedef struct _PMC_BIS_S311_INFO {
    UCHAR    DriverVersion;
    UCHAR    SwitchValue;
    ULONG    InstanceNumber;
} PMC_BIS_S311_INFO, *PPMC_BIS_S311_INFO;
```

IOCTL_PMC_BIS_S311_SET_CONFIG

Function: Sets the value of the configuration register

Input: PMC_BIS_S311_CONFIG structure

Output: none

Notes: Controls the clock source, divisor and output clock selection for determining the transmit reference clock frequency. See the definition of PMC_BIS_S311_CONFIG below. Bit definitions can be found in the 'bis_base' section under [Register Definitions in the Hardware manual](#).

```
typedef struct _PMC_BIS_S311_CONFIG {
    PMC_BIS_S311_CLOCK    ClkSrc;
    USHORT                Divisor;
    BOOLEAN                PostSel;
} PMC_BIS_S311_CONFIG, *PPMC_BIS_S311_CONFIG;
```

IOCTL_PMC_BIS_S311_GET_CONFIG

Function: Returns the state of the configuration register.

Input: none

Output: PMC_BIS_S311_CONFIG structure

Notes: Returns the base configuration register clock control values See the definition of PMC_BIS_S311_CONFIG above. Bit definitions can be found in the 'bis_base' section under [Register Definitions in the Hardware manual](#).



IOCTL_PMC_BIS_S311_GET_STATUS

Function: Returns the board status.

Input: None

Output: Status0 register value (unsigned long integer)

Notes: Returns Status information for a given board obtained from the 'bis_stat0' register. This includes FIFO flags indicating the amount of data in the transmit and receive FIFOs and latched interrupt status bits indicating the cause of an interrupt. After the status is read, a value is written back to this register to clear only the specific latched interrupt conditions that were read. This will insure that no interrupt cause is missed due to being asserted between the read and write cycles. Bit definitions can be found in the 'bis_stat0' section under [Register Definitions in the Hardware manual](#).

IOCTL_PMC_BIS_S311_SET_TX_CONFIG

Function: Sets the transmitter configuration of the board.

Input: PMC_BIS_S311_TX_CONFIG structure

Output: None

Notes: Controls the enabling of the transmit and transmit FIFO almost empty interrupts and whether the external receiver ready signal is ignored or processed. See the definition of PMC_BIS_S311_TX_CONFIG below. Bit definitions can be found in the 'bis_tx' section under [Register Definitions in the Hardware manual](#).

```
typedef struct _PMC_BIS_S311_TX_CONFIG {
    BOOLEAN      TxIntEn;
    BOOLEAN      AlmostMTIntEn;
    BOOLEAN      IgnoreReady;
} PMC_BIS_S311_TX_CONFIG, *PPMC_BIS_S311_TX_CONFIG;
```



IOCTL_PMC_BIS_S311_GET_TX_CONFIG

Function: Returns the transmitter configuration of the board.

Input: None

Output: PMC_BIS_S311_TX_STATE structure

Notes: Returns the transmit configuration items set in the previous call, as well as the transmitter start bit state. See the definition of PMC_BIS_S311_TX_STATE below. Bit definitions can be found in the 'bis_tx' section under [Register Definitions in the Hardware manual](#).

```
typedef struct _PMC_BIS_S311_TX_STATE {
    BOOLEAN    TxStart;
    BOOLEAN    TxIntEn;
    BOOLEAN    AlmostMTIntEn;
    BOOLEAN    IgnoreReady;
} PMC_BIS_S311_TX_STATE, *PPMC_BIS_S311_TX_STATE;
```

IOCTL_PMC_BIS_S311_SET_RX_CONFIG

Function: Sets the receiver configuration of the board.

Input: PMC_BIS_S311_RX_CONFIG structure

Output: None

Notes: Controls the individual enables of the receive, overflow, and almost full interrupts and whether the receive interrupt is asserted on every word received or only on words where the mode bit matches the filtering criterion. Data filtering can be enabled using the same mode bit filter definition. The EnableTest field is needed for loading the receive FIFO from the PCI bus. See the definition of PMC_BIS_S311_RX_CONFIG below. Bit definitions can be found in the 'bis_rx' section under [Register Definitions in the Hardware manual](#).

```
typedef struct _PMC_BIS_S311_RX_CONFIG {
    BOOLEAN    RxIntEn;
    BOOLEAN    AlmostFullIntEn;
    BOOLEAN    OverflowIntEn;
    BOOLEAN    FilterIntEn;
    BOOLEAN    EnableFilter;
    BOOLEAN    KeepOnes;
    BOOLEAN    EnableTest;
} PMC_BIS_S311_RX_CONFIG, *PPMC_BIS_S311_RX_CONFIG;
```



IOCTL_PMC_BIS_S311_GET_RX_CONFIG

Function: Returns the receiver configuration of the board.

Input: None

Output: PMC_BIS_S311_RX_STATE structure

Notes: Returns the receiver configuration items set in the previous call, as well as the receiver start bit state. See the definition of PMC_BIS_S311_RX_STATE below. Bit definitions can be found in the 'bis_rx' section under [Register Definitions in the Hardware manual](#).

```
typedef struct _PMC_BIS_S311_RX_STATE
{
    BOOLEAN      RxStart;
    BOOLEAN      RxIntEn;
    BOOLEAN      AlmostFullIntEn;
    BOOLEAN      OverflowIntEn;
    BOOLEAN      FilterIntEn;
    BOOLEAN      EnableFilter;
    BOOLEAN      KeepOnes;
    BOOLEAN      EnableTest;
} PMC_BIS_S311_RX_STATE, *PPMC_BIS_S311_RX_STATE;
```

IOCTL_PMC_BIS_S311_SET_FIFO_LEVELS

Function: Sets receive almost full and transmit almost empty FIFO levels.

Input: FIFO_LEVELS structure

Output: None

Notes: Sets the almost full level for the receive FIFO; the number of words below full, above which the PAF flag is asserted. Sets the almost empty level for the transmit FIFO; the number of words above empty, below which the PAE flag is asserted. The transmit and receive state machines are stopped by this command, since normal FIFO data accesses are disabled when these level registers are accessed. See the definition of FIFO_LEVELS below.

```
typedef struct _FIFO_LEVELS {
    ULONG      TxAlmostEmpty;
    ULONG      RxAlmostFull;
} FIFO_LEVELS, *PFIFO_LEVELS;
```



IOCTL_PMC_BIS_S311_GET_FIFO_LEVELS

Function: Returns receive almost full and transmit almost empty FIFO levels.

Input: None

Output: FIFO_LEVELS structure

Notes: Returns the almost full level for the receive FIFO and the almost empty level for the transmit FIFO. The transmit and receive state machines are stopped by this command, since normal FIFO data accesses are disabled when these level registers are accessed. See the definition of `FIFO_LEVELS` above.

IOCTL_PMC_BIS_S311_SET_TERMINATIONS

Function: Sets the configuration of the driver terminations.

Input: `PMC_BIS_S311_TERM_CONFIG` structure

Output: None

Notes: Sets the configuration of the terminations for the I/O lines. See the definition of `PMC_BIS_S311_TERM_CONFIG` below. Bit definitions can be found in the 'bis_dir_term' section under [Register Definitions in the Hardware manual](#).

```
typedef struct _PMC_BIS_S311_TERM_CONFIG {
    BOOLEAN    Term0En;
    BOOLEAN    Term1En;
    BOOLEAN    Term2En;
    BOOLEAN    Term3En;
    BOOLEAN    Term4_7En;
    BOOLEAN    Term8_11En;
    BOOLEAN    Term12_15En;
    BOOLEAN    Term16_19En;
} PMC_BIS_S311_TERM_CONFIG, *PPMC_BIS_S311_TERM_CONFIG;
```

IOCTL_PMC_BIS_S311_GET_TERMINATIONS

Function: Returns the configuration of the driver terminations.

Input: None

Output: `PMC_BIS_S311_TERM_CONFIG` structure

Notes: Returns the configuration of the terminations for the I/O lines. See the definition of `PMC_BIS_S311_TERM_CONFIG` above. Bit definitions can be found in the 'bis_dir_term' section under [Register Definitions in the Hardware manual](#).



IOCTL_PMC_BIS_S311_START_TX

Function: Starts the transmitter state machine.

Input: None

Output: None

Notes: Sets the start bit for the transmitter, leaving all other configuration bits the same.

IOCTL_PMC_BIS_S311_START_RX

Function: Starts the receiver state machine.

Input: None

Output: None

Notes: Sets the start bit for the receiver, leaving all other configuration bits the same.

IOCTL_PMC_BIS_S311_STOP_TX

Function: Stops the transmitter state machine.

Input: None

Output: Status0 register value (unsigned long integer)

Notes: Clears the start bit for the transmitter and returns the board Status. This is used to abort a transmission.

IOCTL_PMC_BIS_S311_STOP_RX

Function: Stops the receiver state machine.

Input: None

Output: Status0 register value (unsigned long integer)

Notes: Clears the start bit for the receiver and returns the received word count. This will also clear the word count register, although the old count remains latched until the next received word.

IOCTL_PMC_BIS_S311_GET_RX_COUNT

Function: Returns the number of received words stored.

Input: None

Output: Received data-word count (unsigned long integer)

Notes: Returns the received word count accumulated since the counter was last reset. If data filtering is enabled, the counter only counts words that are actually stored. The counter is also cleared by this command although the old count will remain until the next word is stored in the FIFO.



IOCTL_PMC_BIS_S311_PUT_TX_DATA

Function: Loads one data word into the transmit FIFO.

Input: Transmit data value

Output: None

Notes: Loads a single transmit data word into the transmit FIFO. This IOCTL can be used to load small amounts of data instead of the WriteFile() call.

IOCTL_PMC_BIS_S311_GET_RX_DATA

Function: Reads one data word from the receive FIFO.

Input: None

Output: Received data value (unsigned long integer)

Notes: Reads a single receive data word from the receive FIFO. This IOCTL can be used to read small amounts of data instead of the ReadFile() call.

IOCTL_PMC_BIS_S311_GET_TX_DATA

Function: Reads one data word from the transmit FIFO.

Input: None

Output: Data value (unsigned long integer)

Notes: Used for transmit FIFO loop-back testing.

IOCTL_PMC_BIS_S311_PUT_RX_DATA

Function: Loads one data word into the receive FIFO.

Input: Data value (unsigned long integer)

Output: None

Notes: Used for receive FIFO loop-back testing. EnableTest must be set with the IOCTL_PMC_BIS_S311_SET_RX_CONFIG call.

IOCTL_PMC_BIS_S311_RESET_FIFOS

Function: Resets both the transmit and receive FIFOs.

Input: None

Output: Status0 register value (unsigned long integer)

Notes: Resets both the transmit and receive FIFOs. This will clear all data and reset the almost full and empty values to the default value of seven. This call returns the value of the status0 register.



IOCTL_PMC_BIS_S311_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_PMC_BIS_S311_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_PMC_BIS_S311_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_PMC_BIS_S311_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_PMC_BIS_S311_GET_ISR_STATUS

Function: Returns the Plx-9052 interrupt status read in the last ISR.

Input: none

Output: PMC_BIS_S311_ISR_STATUS

Notes: The status contains the status and control bits of the Plx ICS register read in the last ISR execution.

```
typedef struct _PMC_BIS_S311_ISR_STATUS
{
    ULONG      IsrStatus;
    BOOLEAN    PlxLocIntStat1;
    BOOLEAN    PlxLocIntStat2;
} PMC_BIS_S311_ISR_STATUS, *PPMC_BIS_S311_ISR_STATUS;
```



Write

Data can be written to the device using the write command. Writes are executed using the function WriteFile() and passing in the handle to the device opened with CreateFile(), a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually written, and a pointer to an optional Overlapped structure for performing asynchronous IO. It should be noted that asynchronous IO has not been tested. The size of buffer in bytes should fall on a long word boundary. The total number of writes should not exceed the number that fit in the FIFO. Writing more than will fit into the FIFO will result in data being dropped [overflow]. Fit means locations remaining in the FIFO at the time of the write command.

Read

Data can be read from the device using the read command. Reads are executed using the function ReadFile() and passing in the handle to the device opened with CreateFile(), a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually read, and a pointer to an optional Overlapped structure for performing asynchronous IO. It should be noted that asynchronous IO has not been tested. The size of buffer in bytes should fall on a long word boundary. The total number of reads should not exceed the number of data in the FIFO. Reading more than stored will result in duplicated data [underflow].

For PmcBisS311 write and read are implemented with multiple Kernel level single word write and read for higher performance.



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 “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 C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax
support@dyneng.com

All information provided is Copyright Dynamic Engineering

