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 Biserial6 S311

Software Manual

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Revision B1
10-2015-0601/2

PmcBis6S311
WDF Device Drivers for the
PMC-BiSerial6 S311

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

Manual Revision B. Revised October 5, 2018.

Table of Contents

Introduction

The PmcBis6S311 driver was developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF)..

There are 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 PmcBis6S311 driver package has two parts. The driver is installed into the Windows® OS and the User Application “Userapp” exectutable.

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-VI-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.

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 BiSerial VI S311 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include PmcBis6S311Public.h, PmcBis6S311.inf, pmcbis6s311.cat, PmcBis6S311.sys,and WdfCoInstaller01009.dll.

PmcBis6S311Public.h is the C header file that defines the Application Program Interface (API) for the PmcBis6S311 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 PmcBis6S311.inf, pmcbis6s311.cat, PmcBis6S311.sys, and WdfCoInstaller01009.dll (Win7 version) to a CD or USB memory device as preferred.

With the PMC Parllel 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 PmcBis6S311 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 PmcBis6S311Public.h. See main.c in the PmcBis6S311UserApp 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 BISERIAL6 S311 driver are described below:

IOCTL_PMC_BIS6_S311_GET_INFO

Function: Returns the current driver version and instance number.
Input: none
Output: PMC_BIS6_S311_INFO structure
Notes: This call does not access the hardware, only driver parameters. See the definition of PMC_BIS6_S311_INFO below.

```
typedef struct _PMC_BIS6_S311_INFO
{
   UCHAR    DriverVersion;
   UCHAR    SwitchValue;
   ULONG    InstanceNumber;
   UCHAR    MajorRev;
   UCHAR    MinorRev;
} PMC_BIS6_S311_INFO, *PPMC_BIS6_S311_INFO;
```

IOCTL_PMC_BIS6_S311_SET_BASE_CONFIG

Function: Sets the value of the Clk outputs for each channel
Input: PMC_BIS6_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_BIS6_S311_CLOCK below. Bit definitions can be found in the 'PortnClk' section under Register Definitions in the Hardware manual.

```
typedef enum _PMC_PMC_BIS6_S311_CLOCK {
   S311_OSC,
   S311_EXT,
   S311_PCI
} PMC_BIS6_S311_CLOCK, *PPMC_BIS6_S311_CLOCK;

typedef struct _PMC_BIS6_S311_CONFIG {
   UCHAR                channel;
   PMC_BIS6_S311_CLOCK  ClkSrc;
   USHORT               Divisor;
   BOOLEAN              PostSel;
} PMC_BIS6_S311_CONFIG, *PPMC_BIS6_S311_CONFIG;
```

IOCTL_PMC_BIS6_S311_GET_BASE_CONFIG

Function: Returns the state of the TTL outputs in the output data register.
Input: none
Output: PMC_BIS6_S311_CONFIG structure
Notes: Returns the clock configuration register. See the definition of PMC_BIS6_S311_CONFIG above. Bit definitions can be found in the 'PortnClk' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_S311_GET_BASE_STATUS

Function: Returns the base status.
Input: None
Output: Status register value (unsigned long integer)
Notes: Returns the base status information for a given board obtained from the 'Status' register. Bit definitions can be found in the 'Status' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_S311_SET_ACTIVE_CHAN

Function: Sets the active channel for ReadFile and WriteFile.
Input: ACTIVE_CHAN structure
Output: none
Notes:

```
typedef struct _ACTIVE_CHAN {
   UCHAR    RxChan;
   UCHAR    TxChan;
} ACTIVE_CHAN, *PACTIVE_CHAN;
```

IOCTL_PMC_BIS6_S311_SET_CHAN_CONFIG

Function: Sets the channel configuration of the board.
Input: PMC_BIS6_S311_TX_CONFIG structure
Output: None
Notes: Controls the enabling of channel interrupts, filter controls, and whether the external receiver ready signal is ignored or processed. See the definition of PMC_BIS6_S311_CHAN_CONFIG below. Bit definitions can be found in the 'ChanCntl' section under Register Definitions in the Hardware manual.

```
typedef struct _PMC_BIS6_S311_CHAN_CONFIG{
   UCHAR          channel;
   BOOLEAN        TxIntEn;
   BOOLEAN        AlmostMTIntEn;
   BOOLEAN        TxRdyCntrl;
   BOOLEAN        RxIntEn;
   BOOLEAN        AlmostFullIntEn;
   BOOLEAN        OverflowIntEn;
   BOOLEAN        EnableFilter;
   BOOLEAN        FilterIntEn;
   BOOLEAN        FilterDef;
   BOOLEAN        EnableTest;
} PMC_BIS6_S311_CHAN_CONFIG, *PPMC_BIS6_S311_CHAN_CONFIG;
```

IOCTL_PMC_BIS6_S311_GET_CHAN_CONFIG

Function: Returns the channel configuration of the board.
Input: None
Output: PMC_BIS6_S311_CHAN_STATE structure
Notes: Returns the transmit configuration items set in the previous call, as well as the tx and rx start bit state. See the definition of PMC_BIS6_S311_CHAN_STATE below. Bit definitions can be found in the 'ChanCntl' section under Register Definitions in the Hardware manual.

```
typedef struct _PMC_BIS6_S311_CHAN_STATE{
   UCHAR          channel;
   BOOLEAN        Reset;
   BOOLEAN        ChanIntEn;
   BOOLEAN        TxStart;
   BOOLEAN        TxIntEn;
   BOOLEAN        AlmostMTIntEn;
   BOOLEAN        TxRdyCntrl;
   BOOLEAN        RxStart;
   BOOLEAN        RxIntEn;
   BOOLEAN        AlmostFullIntEn;
   BOOLEAN        OverflowIntEn;
   BOOLEAN        EnableFilter;
   BOOLEAN        FilterIntEn;
   BOOLEAN        FilterDef;
   BOOLEAN        EnableTest;
} PMC_BIS6_S311_CHAN_STATE, *PPMC_BIS6_S311_CHAN_STATE;
```

IOCTL_PMC_BIS6_S311_GET_CHAN_STATUS

Function: Returns the channel status.
Input: None
Output: Chan register value (unsigned long integer)
Notes: Returns Channel Interrupt Status information for a given board obtained from the 'ChanStatus' register. The 'Sticky' bits are cleared after the Status is returned. Therefore a subsequent call to this IOCTL will return with the sticky bits low. Bit definitions can be found in the 'ChanStatus' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_S311_CLEAR_WORD_COUNT

Function: Clears the word count register.
Input: Channel (unsigned character)
Output: None
Notes:

IOCTL_PMC_BIS6_S311_GET_WORD_COUNT

Function: Returns the word count.
Input: Channel (unsigned character)
Output: PMC_BIS6_S311_WORD_COUNT
Notes: This count is updated as each word is stored into the receive FIFO. Bit definition can be found in the 'ChanWordCnt' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_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. Bit definitions can be found in the 'ChanFifoLvl' section under Register Definitions in the Hardware manual.

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

IOCTL_PMC_BIS6_S311_GET_FIFO_LEVELS

Function: Returns receive almost full and transmit almost empty FIFO levels.
Input: Channel (unsigned character)
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. Bit definitions can be found in the 'ChanFifoLvl' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_S311_SET_TERMINATIONS

Function: Sets the configuration of the driver terminations.
Input: PMC_BIS6_S311_TERM_CONFIG structure
Output: None
Notes: Sets the configuration of the terminations for the I/O lines. See the definition of PMC_BIS6_S311_TERM_CONFIG below. Bit definitions can be found in the 'ChanTerm' section under Register Definitions in the Hardware manual.

```
typedef struct _PMC_BIS6_S311_TERM_CONFIG
{
   UCHAR          channel;
   BOOLEAN        TermRefClk;
   BOOLEAN        TermTxData;
   BOOLEAN        TermTxClk;
   BOOLEAN        TermTxRqst;
   BOOLEAN        TermTxRdy;
   BOOLEAN        TermRxData;
   BOOLEAN        TermRxClk;
   BOOLEAN        TermRxRqst;
   BOOLEAN        TermRxRdy;
} PMC_BIS6_S311_TERM_CONFIG, *PPMC_BIS6_S311_TERM_CONFIG;
```

IOCTL_PMC_BIS6_S311_GET_TERMINATIONS

Function: Returns the configuration of the driver terminations.
Input: Channel (unsigned character)
Output: PMC_BIS6_S311_TERM_CONFIG structure
Notes: Returns the configuration of the terminations for the I/O lines. See the definition of PMC_BIS6_S311_TERM_CONFIG above. Bit definitions can be found in the 'ChanTerm' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_S311_START_TX

Function: Starts the transmitter state machine.
Input: Channel (unsigned character)
Output: None
Notes: Sets the start bit for the transmitter, leaving all other configuration bits the same.

IOCTL_PMC_BIS6_S311_START_RX

Function: Starts the receiver state machine.
Input: Channel (unsigned character)
Output: None
Notes: Sets the start bit for the receiver, leaving all other configuration bits the same.

IOCTL_PMC_BIS6_S311_STOP_TX

Function: Stops the transmitter state machine.
Input: Channel (unsigned character)
Output: Status 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_BIS6_S311_STOP_RX

Function: Stops the receiver state machine.
Input: Channel (unsigned character)
Output: Status 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_BIS6_S311_SET_PREREAD_TX

Function: Write to Preread register
Input: Channel (unsigned character)
Output: None
Notes: This IOCTL is necessary in conjunction with the TX FIFO data ready bit in Chan Status (bit 5), to read back valid data when using the GET_TX_DATA IOCTL described below. Register definition can be found in the 'PreReadTxFifo' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS6_S311_GET_FIFO_COUNT

Function: Returns the number of words stored in the TX and RX FIFOs.
Input: Channel (unsigned character)
Output: PMC_BIS6_S311_FIFO_COUNT
Notes: Returns the FIFOS counts accumulated since the counter was last reset. If data filtering is enabled, the counter only counts words that are actually stored. See the definition of PMC_BIS6_S311_FIFO_COUNT below. Register definition can be found in the 'ChanWordCnt' section under Register Definitions in the Hardware manual.

```
typedef struct _PMC_BIS6_S311_FIFO_COUNT
{
   UCHAR    channel;
   ULONG    TxFifoCount;
   ULONG    RxFifoCount;
} PMC_BIS6_S311_FIFO_COUNT, *PPMC_BIS6_S311_FIFO_COUNT;
```

IOCTL_PMC_BIS6_S311_PUT_TX_DATA

Function: Loads one data word into the transmit FIFO.
Input: PMC_BIS6_S311_DATA
Output: None
Notes: Loads a single transmit data word into the transmit FIFO. See the definition of PMC_BIS6_S311_FIFO_DATA below.

```
typedef struct _PMC_BIS6_S311_DATA
{
   UCHAR    channel;
   ULONG    FifoData;
} PMC_BIS6_S311_DATA, *PPMC_BIS6_S311_DATA;
```

IOCTL_PMC_BIS6_S311_GET_RX_DATA

Function: Reads one data word from the receive FIFO.
Input: Channel (unsigned character)
Output: Received data value (unsigned long integer)
Notes: Reads a single receive data word from the receive FIFO.

IOCTL_PMC_BIS6_S311_GET_TX_DATA

Function: Reads one data word from the transmit FIFO.
Input: Channel (unsigned character)
Output: Data value (unsigned long integer)
Notes: Used for transmit FIFO loop-back testing.

IOCTL_PMC_BIS6_S311_PUT_RX_DATA

Function: Loads one data word into the receive FIFO.
Input: PMC_BIS6_S311_DATA
Output: None
Notes: Used for receive FIFO loop-back testing. EnableTest must be set with the IOCTL_PMC_BIS6_S311_SET_CHAN_CONFIG call. See the definition of PMC_BIS6_S311_FIFO_DATA above.

IOCTL_PMC_BIS6_S311_RESET_FIFOS

Function: Resets both the transmit and receive FIFOs for given channel
Input: Channel (unsigned character)
Output: Status register value (unsigned long integer)
Notes: Resets both the transmit and receive FIFOs for given channel. 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 status register.

IOCTL_PMC_BIS6_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_BIS6_S311_ENABLE_BASE_INTERRUPT

Function: Enables the base interrupts.
Input: none
Output: none
Notes: Sets the interrupt enable. 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. The Base Interrupt must be enabled for channel interrupts to occur.

IOCTL_PMC_BIS6_S311_DISABLE_BASE_INTERRUPT

Function: Disables the base interrupt.
Input: none
Output: none
Notes: Clears the interrupt enable. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_PMC_BIS6_S311_FORCE_BASE_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_BIS6_S311_ENABLE_CHAN_INTERRUPT

Function: Enables the interrupts for given channel.
Input: Channel (unsigned character)
Output: none
Notes: Sets the channel interrupt enable for given channel. 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. The Base Interrupt must also be enabled for channel interrupts to occur.

IOCTL_PMC_BIS6_S311_DISABLE_CHAN_INTERRUPT

Function: Disables the interrupt for given channel.
Input: Channel (unsigned character)
Output: none
Notes: Clears the channel interrupt enable for given channel. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_PMC_BIS6_S311_FORCE_CHAN_INTERRUPT

Function: Causes a system interrupt to occur for given channel.
Input: Channel (unsigned character)
Output: none
Notes: Causes a channel 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_BIS6_S311_GET_ISR_STATUS

Function: Returns the interrupt status read in the last ISR.
Input: none
Output: PMC_BIS6_S311_INT_STATUS
Notes: The status contains the status and control bits of the Chan Status register read in the last ISR execution.

```
typedef struct _PMC_BIS6_INT_STAT
{
   ULONG    IntBaseStat;
   USHORT   IntChanStat[NUM_CHANS];
} PMC_BIS6_INT_STAT, *PPMC_BIS6_INT_STAT;
```

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. IOCTL_PMC_BIS6_S311_SET_ACTIVE_CHAN is used to set the desired channel to be written.

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]. IOCTL_PMC_BIS6_S311_SET_ACTIVE_CHAN is used to set the desired channel to be read.

For PmcBis6S311 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