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 BiSerial3 RL1

Software Manual

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Rev A1
Flash Rev C
10-2005-0205

PmcBis3Rl1
WDF Device Drivers for the
PMC Biserial 3 Rl1

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

Manual Revision A. Revised October 10, 2018.

Table of Contents

INTRODUCTION 4

DRIVER INSTALLATION 5

Windows 7 Installation 5

IO Controls 6
IOCTL_PMC_BIS3_RL1_BASE_GET_INFO 7
IOCTL_PMC_BIS3_RL1_BASE_LOAD_PLL_DATA 7
IOCTL_PMC_BIS3_RL1_BASE_READ_PLL_DATA 8
IOCTL_PMC_BIS3_RL1_CHAN_GET_INFO 8
IOCTL_PMC_BIS3_RL1_SET_CHAN_CONFIG 8
IOCTL_PMC_BIS3_RL1_GET_CHAN_STATE 9
IOCTL_PMC_BIS3_RL1_CHAN_GET_STATUS 10
IOCTL_PMC_BIS3_RL1_CHAN_SET_FIFO_LEVELS 10
IOCTL_PMC_BIS3_RL1_CHAN_GET_FIFO_LEVELS 10
IOCTL_PMC_BIS3_RL1_CHAN_GET_FIFO_COUNT 11
IOCTL_PMC_BIS3_RL1_CHAN_RESET_FIFOS 11
IOCTL_PMC_BIS3_RL1_CHAN_WRITE_FIFO 12
IOCTL_PMC_BIS3_RL1_CHAN_READ_FIFO 12
IOCTL_PMC_BIS3_RL1_CHAN_SET_TX_CONFIG 12
IOCTL_PMC_BIS3_RL1_CHAN_GET_TX_STATE 13
IOCTL_PMC_BIS3_RL1_CHAN_SET_RX_CONFIG 13
IOCTL_PMC_BIS3_RL1_CHAN_GET_RX_STATE 14
IOCTL_PMC_BIS3_RL1_CHAN_START_TX 14
IOCTL_PMC_BIS3_RL1_CHAN_STOP_TX 14
IOCTL_PMC_BIS3_RL1_CHAN_START_RX 15
IOCTL_PMC_BIS3_RL1_CHAN_STOP_RX 15
IOCTL_PMC_BIS3_RL1_CHAN_GET_RX_BYTE_COUNT 15
IOCTL_PMC_BIS3_RL1_REGISTER_EVENT 16
IOCTL_PMC_BIS3_RL1_ENABLE_INTERRUPT 16
IOCTL_PMC_BIS3_RL1_DISABLE_INTERRUPT 16
IOCTL_PMC_BIS3_RL1_FORCE_INTERRUPT 16
IOCTL_PMC_BIS3_RL1_GET_ISR_STATUS 17

Write 18

Read 18

WARRANTY AND REPAIR 19

Service Policy 19
Support 19

For Service Contact: 19

Introduction

The Pmc-BiSerial-III-RL1 base and chan drivers were developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF). It was developed using 64 bit Windows operating system with an Intel Core i7 Processor.

The PMC-BiSerial-III board has a Spartan3-1500 Xilinx FPGA to implement the PCI interface, FIFOs and protocol control and status for eight serial channels. Each channel has two 1k x 32-bit data FIFOs for data transmission and reception.

The 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 register 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.

When the PMC-BiSerial-III-RL1 is recognized by the PCI bus configuration utility it will start the PmcBis3RI1 drivers 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-BiSerial-III-RI1 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include PmcBis3Rl1BasePublic.h, PmcBis3Rl1Base.inf, pmcbis3rl1base.cat, PmcBis3Rl1Base.sys, PmcBis3Rl1ChanPublic.h, PmcBis3Rl1Chan.inf, pmcbis3rl1chan.cat, PmcBis3Rl1Chan.sys, and WdfCoInstaller01009.dll.

PmcBis3Rl1BasePublic.h are PmcBis3Rl1ChanPublic.h the C header file that define the Application Program Interface (API) for the PmcBis3Rl1 drivers. These files are 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 PmcBis3Rl1Base.inf, pmcbis3rl1base.cat, PmcBis3Rl1Base.sys, PmcBis3Rl1Chan.inf, pmcbis3rl1chan.cat, PmcBis3Rl1Chan.sys, and WdfCoInstaller01009.dll (Win7 version) to a CD or USB memory device as preferred.

With the PMC BIS3 RL1 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.
- Follow the same steps to install the channel drivers.

The system should now display the PmcBis3Rl1 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 PmcBis3Rl1Public.h. See main.c in the PmcBis3Rl1UserApp 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 PmcBis3Rl1Base driver are described below:

IOCTL_PMC_BIS3_RL1_BASE_GET_INFO

Function: Returns the current driver version and instance number.
Input: none
Output: PMC_BIS3_RL1_BASE_DRIVER_DEVICE_INFO structure
Notes: Switch value is the configuration of the onboard dipswitch that has been selected by the User (see the board silk screen for bit position and polarity). The PLL ID is the device address of the PLL. This value, which is set at the factory, is usually 0x69 but may also be 0x6A. See the definition of PMC_BIS3_RL1_DRIVER_DEVICE_INFO below. Refer to the PrintInfo function found in the PrintInfo.c file in the UserApp for an example of use.

```
typedef struct PMC_BIS3_RL1_BASE_DRIVER_DEVICE_INFO
{
   UCHAR    DriverVersion;
   UCHAR    XilinxVersion;
   UCHAR    PllDeviceId;
   UCHAR    SwitchValue;
   ULONG    InstanceNumber;
   BOOLEAN  PllALocked;
   BOOLEAN  PllBLocked;
} PMC_BIS3_RL1_BASE_DRIVER_DEVICE_INFO, *PPMC_BIS3_RL1_BASE_DRIVER_DEVICE_INFO;
```

IOCTL_PMC_BIS3_RL1_BASE_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.
Input: PMC_BIS3_RL1_PLL_DATA structure
Output: None
Notes: The PMC_BIS3_RL1_PLL_DATA structure has only one field: Data – an array of 40 bytes containing the data to write. See the definition of PMC_BIS3_RL1_PLL_DATA below. Refer to the PLL_if_test function found in the UserApp for an example of use.

```
#define PLL_MESSAGE1_SIZE     16
#define PLL_MESSAGE2_SIZE     24
#define PLL_MESSAGE_SIZE     (PLL_MESSAGE1_SIZE + PLL_MESSAGE2_SIZE)

typedef struct PMC_BIS3_RL1_BASE_PLL_DATA
{
   UCHAR    Data[PLL_MESSAGE_SIZE];
} PMC_BIS3_RL1_BASE_PLL_DATA, *PPMC_BIS3_RL1_BASE_PLL_DATA;
```

IOCTL_PMC_BIS3_RL1_BASE_READ_PLL_DATA

Function: Returns the contents of the PLL's internal registers
Input: None
Output: PMC_BIS3_RL1_PLL_DATA structure
Notes: The register data is output in the PMC_BIS3_RL1_PLL_DATA structure in an array of 40 bytes. Refer to the PLL_if_test function found in the UserApp for an example of use.

The IOCTLs defined for the PmcBis3Rl1Chan driver are described below:

IOCTL_PMC_BIS3_RL1_CHAN_GET_INFO

Function: Returns the current driver version and instance number.
Input: none
Output: RL1_CHAN_DRIVER_DEVICE_INFO structure
Notes:. See the definition of RL1_CHAN_DRIVER_DEVICE_INFO below. Refer to the PrintInfo function found in the PrintInfo.c file in the UserApp for an example of use.

```
typedef struct _RL1_CHAN_DRIVER_DEVICE_INFO
{
   UCHAR    DriverVersion;
   ULONG    InstanceNumber;
} RL1_CHAN_DRIVER_DEVICE_INFO, *PRL1_CHAN_DRIVER_DEVICE_INFO;
```

IOCTL_PMC_BIS3_RL1_CHAN_SET_CONFIG

Function: Specifies fields in the channel configuration register
Input: RL1_CHAN_CONFIG structure
Output: none
Notes: This call controls channel configuration items that are not transmit or receive specific. The AutoDirSwitch field enables the automatic switching from transmit to receive and vice versa when the current active direction signals that it is done. When IoClockASel is true, PLL clock A is selected as the clock source, when it is false PLL clock B is selected. When ClockDiv is equal to one, the undivided clock source will be used for the 16x reference clock. Otherwise the clock source can be divided by any even number from two to thirty-two. See the definition of RL1_CHAN_CONFIG below. Register definition can be found in the 'RL1_CHAN_0-7_CONTROL' section under Register Definitions in the Hardware manual. Refer to the ioloop_tst function found in the UserApp for an example of use.

```
typedef struct _RL1_CHAN_CONFIG
{
   BOOLEAN  FifoTestEn;
   BOOLEAN  DmaInPreemptEn;
   BOOLEAN  DmaOutPreemptEn;C
   BOOLEAN  FullDuplexEn;
   BOOLEAN  AutoDirSwitch;
   BOOLEAN  IoClockASel;
   BOOLEAN  ClockDivSel;
   UCHAR    ClockDiv;
} RL1_CHAN_CONFIG, *PRL1_CHAN_CONFIG;
```

IOCTL_PMC_BIS3_RL1_CHAN_GET_STATE

Function: Returns the channel configuration
Input: none
Output: RL1_CHAN_STATE structure
Notes: The states of the interrupt enables are returned for informational purposes only. The values of these fields are controlled by other driver calls. The MIntEn field is the master interrupt enable for all user interrupts controlled by the EnableInterrupt and DisableInterrupt calls, whereas the WrDmaEn and RdDmaEn fields are automatically controlled by the driver in response to WriteFile and ReadFile calls. Register definition can be found in the 'RL1_CHAN_0-7_CONTROL' section under Register Definitions in the Hardware manual. See the definition of RL1_CHAN_STATE below.

```
typedef struct _RL1_CHAN_STATE
{
   BOOLEAN  FifoTestEn;
   BOOLEAN  DmaInPreemptEn;
   BOOLEAN  DmaOutPreemptEn;
   BOOLEAN  FullDuplexEn;
   BOOLEAN  AutoDirSwitch;
   BOOLEAN  IoClockASel;
   BOOLEAN  ClockDivSel;
   UCHAR    ClockDiv;       // divide by 1,2,4,6...32
   BOOLEAN  MIntEn;
   BOOLEAN  WrDmaEn;
   BOOLEAN  RdDmaEn;
   BOOLEAN  FifoResetTX;
   BOOLEAN  FifoResetRX;
} RL1_CHAN_STATE, *PRL1_CHAN_STATE;
```

IOCTL_PMC_BIS3_RL1_CHAN_GET_STATUS

Function: Returns the interrupt status bit mask and clears the latched bits.
Input: None
Output: Interrupt status channel mask (unsigned long integer)
Notes: Only the bits in STATUS_MASK will be returned. The bits in STATUS_LATCH_MASK will be cleared by this call only if they are set when the register was read. This prevents the possibility of missing an interrupt condition that occurs after the register has been read but before the latched register bits are cleared.Bit definitions can be found in the 'RL1_CHAN_0-7_STATUS' section under Register Definitions in the Hardware manual. Refer to the ioloop_tst function found in the UserApp for an example of use

IOCTL_PMC_BIS3_RL1_CHAN_SET_FIFO_LEVELS

Function: Sets receive almost full and transmit almost empty FIFO levels.
Input: RL1_CHAN_FIFO_LEVELS structure
Output: None
Notes: These FIFO levels are used to determine TX almost empty and RX almost full status when the FIFO data counts reach the specified levels. They are also used to signal priority for the DMA request/grant arbiter, if this has been enabled for the referenced channel. Register definition can be found in the 'RL1_CHAN_0-7_TX_AMT_LVL' and 'RL1_CHAN_0-7_AFL_LVL' sections under Register Definitions in the Hardware manual. Refer to the ioloop_tst function found in the UserApp for an example of use

```
typedef struct _RL1_CHAN_FIFO_LEVELS
{
   USHORT   AlmostFull;
   USHORT   AlmostEmpty;
} RL1_CHAN_FIFO_LEVELS, *PRL1_CHAN_FIFO_LEVELS;
```

IOCTL_PMC_BIS3_RL1_CHAN_GET_FIFO_LEVELS

Function: Returns receive almost full and transmit almost empty FIFO levels.
Input: Channel (unsigned character)
Output: RL1_CHAN_FIFO_LEVELS structure
Notes: Register definition can be found in the 'RL1_CHAN_0-7_TX_AMT_LVL' and 'RL1_CHAN_0-7_AFL_LVL' sections under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS3_RL1_CHAN_GET_FIFO_COUNT

Function: Returns the number of words stored in the TX and RX FIFOs.
Input: None
Output: RL1_CHAN_FIFO_COUNTS
Notes: Returns the number of words in the referenced channels I/O data circuitry. For the transmitter this is a maximum of one more than the FIFO size and for the receiver the data-count can be as much as four words more than the FIFO size. The excess is due to data pipe-line latches in the I/O data-path. . See the definition of RL1_CHAN_FIFO_COUNTS below. Register definition can be found in the 'RL1_CHAN_0-7_TX_FIFO_COUNT' and 'RL1_CHAN_0-7_TX_FIFO_COUNT' sections under Register Definitions in the Hardware manual. Refer to the ioloop_tst function found in the UserApp for an example of use

```
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_BIS3_RL1_CHAN_RESET_FIFOS

Function: Resets TX and/or RX FIFOs
Input: RL1_CHAN_FIFO_SEL
Output: None
Notes: Resets either one or both transmit and receive FIFOs depending on input. This will clear all data. Refer to the ioloop_tst function found in the UserApp for an example of use

```
typedef enum _RL1_CHAN_FIFO_SEL
{
   RL1_TX,
   RL1_RX,
   RL1_BOTH
} RL1_CHAN_FIFO_SEL, *PRL1_CHAN_FIFO_SEL;
```

IOCTL_PMC_BIS3_RL1_CHAN_WRITE_FIFO

Function: Writes a single 32-bit word to the channel's transmit FIFO.
Input: FIFO data word (unsigned long integer)
Output: None
Notes: Normally the write command is used to load data into the device. This call can be used for small amounts of data, but is inefficient for larger sized transfers.

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

IOCTL_PMC_BIS3_RL1_CHAN_READ_FIFO

Function: Reads a single 32-bit word from the channel's receive FIFO.
Input: None
Output: FIFO data word (unsigned long integer)
Notes: Normally the read command is used to retrieve data from the device. This call can be used for small amounts of data, but is inefficient for larger sized transfers.

IOCTL_PMC_BIS3_RL1_CHAN_SET_TX_CONFIG

Function: Specifies various parameters that control the behavior of the transmitter.
Input: RL1_CHAN_TX_CONFIG structure
Output: None
Notes: See below for the definition of RL1_CHAN_TX_CONFIG. Register definition can be found in the 'RL1_CHAN_0-7_TX_CONTROL' section under Register Definitions in the Hardware manual. Refer to the ioloop_tst function found in the UserApp for an example of use

```
typedef struct _RL1_CHAN_TX_CONFIG
{
   BOOLEAN           TxIntEnable;
   BOOLEAN           TxFifoIntEn;
   BOOLEAN           ClearEnable;
   BOOLEAN           StopTwoSel;
   RL1_CHAN_PAR_SEL  Parity;
} RL1_CHAN_TX_CONFIG, *PRL1_CHAN_TX_CONFIG;
```

IOCTL_PMC_BIS3_RL1_CHAN_GET_TX_STATE

Function: Returns the parameters set in the previous call as well as the state of the transmitter enable bit.
Input: None
Output: RL1_CHAN_TX_STATE structure
Notes: See below for the definition of RL1_CHAN_TX_ STATE. If the ClearEnable field has been set to true, the Enabled field can be monitored to indicate when the current message has completed. Register definition can be found in the 'RL1_CHAN_0-7_TX_CONTROL' section under Register Definitions in the Hardware manual.

```
typedef struct _RL1_CHAN_TX_STATE
{
   BOOLEAN           Enabled;
   BOOLEAN           TxIntEnable;
   BOOLEAN           TxFifoIntEn;
   BOOLEAN           ClearEnable;
   BOOLEAN           StopTwoSel;
   RL1_CHAN_PAR_SEL  Parity;
} RL1_CHAN_TX_STATE, *PRL1_CHAN_TX_STATE;
```

IOCTL_PMC_BIS3_RL1_CHAN_SET_RX_CONFIG

Input: RL1_CHAN_RX_CONFIG structure
Output: None
Notes: TermEnable activates the 100Ω shunt termination on the receive data lines. When the interface is operating in half-duplex mode, the termination will only be active when the transmitter is not active. See below for the definition of RL1_CHAN_RX_CONFIG. Register definition can be found in the 'RL1_CHAN_0-7_RX_CONTROL' section under Register Definitions in the Hardware manual. Refer to the ioloop_tst function found in the UserApp for an example of use

```
typedef struct _RL1_CHAN_RX_CONFIG
{
   BOOLEAN           RxIntEnable;
   BOOLEAN           RxFifoIntEn;
   BOOLEAN           RxOvflIntEn;
   BOOLEAN           ClearEnable;
   BOOLEAN           TermEnable;
   BOOLEAN           StopTwoSel;
   RL1_CHAN_PAR_SEL  Parity;
} RL1_CHAN_RX_CONFIG, *PRL1_CHAN_RX_CONFIG;
```

IOCTL_PMC_BIS3_RL1_CHAN_GET_RX_STATE

Function: Returns the parameters set in the previous call as well as the state of the receiver enable bit.
Input: None
Output: RL1_CHAN_RX_STATE structure
Notes: If the ClearEnable field has been set to true, the Enabled field can be monitored to indicate when the current message has completed. See below for the definition of RL1_CHAN_RX_STATE. Register definition can be found in the 'RL1_CHAN_0-7_RX_CONTROL' section under Register Definitions in the Hardware manual.

```
typedef struct _RL1_CHAN_RX_STATE
{
   BOOLEAN           Enabled;
   BOOLEAN           RxIntEnable;
   BOOLEAN           RxFifoIntEn;
   BOOLEAN           RxOvflIntEn;
   BOOLEAN           ClearEnable;
   BOOLEAN           TermEnable;
   BOOLEAN           StopTwoSel;
   RL1_CHAN_PAR_SEL  Parity;
} RL1_CHAN_RX_STATE, *PRL1_CHAN_RX_STATE;
```

IOCTL_PMC_BIS3_RL1_CHAN_START_TX

Function: Starts a data transmission provided valid data is available to send.
Input: Number of bytes to send (unsigned short integer)
Output: None
Notes: If the input field is NULL or zero, the transmission will continue until all FIFO data has been sent. If this field is non-zero, only the specified number of bytes will be sent. Register definition can be found in the 'RL1_CHAN_0-7_TX_START_LATCH section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS3_RL1_CHAN_STOP_TX

Function: Abort or cancel a data transmission.
Input: None
Output: None
Notes: This call will cancel a transmit request that has not started or stop a transmission in progress. Register definition can be found in the 'RL1_CHAN_0-7_TX_START_LATCH section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS3_RL1_CHAN_START_RX

Function: Enable the receiver to look for data and store it in the receive FIFO.
Input: None
Output: None
Notes: . Register definition can be found in the 'RL1_CHAN_0-7_RX_START_LATCH section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS3_RL1_CHAN_STOP_RX

Function: Abort or cancel a data reception.
Input: None
Output: None
Notes: This call will cancel a receive request that has not started or stop a reception in progress. Register definition can be found in the 'RL1_CHAN_0-7_RX_START_LATCH section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS3_RL1_CHAN_GET_RX_BYTE_COUNT

Function: Returns the number of bytes received in the last message.
Input: None
Output: Received byte count (unsigned short integer)
Notes: Each channel contains a 16-bit counter that increments each time a data byte is received. When the received data input is high for at least 8 bit-periods after the end of a data-byte, the receiver sets the STAT_RX_INT status bit, transfers this count to the byte-count register and clears the counter for the next message. The byte-count register value is returned by this call. The value will remain valid until the end of a subsequent message. Register definition can be found in the 'RL1_CHAN_0-7_BYTE_COUNT' section under Register Definitions in the Hardware manual.

IOCTL_PMC_BIS3_RL1_CHAN_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The caller creates an event with CreateEvent(). The returned handle is the input to this IOCTL. The driver then signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. Refer to the interrupt function found in the UserApp for an example of use

IOCTL_PMC_BIS3_RL1_CHAN_ENABLE_INTERRUPT

Function: Enables the master interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to local interrupts. The master interrupt enable is disabled in the driver interrupt service routine. Therefore this command must be run after an interrupt occurs to be ready for the next interrupt. Refer to the interrupt function found in the UserApp for an example of use

IOCTL_PMC_BIS3_RL1_CHAN_DISABLE_INTERRUPT

Function: Disables the master interrupt.
Input: None
Output: None
Notes: This call is used when local interrupt processing is no longer desired. Refer to the interrupt function found in the UserApp for an example of use

IOCTL_PMC_BIS3_RL1_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the master interrupt is enabled. This IOCTL is used for development, to test interrupt processing. Refer to the interrupt function found in the UserApp for an example of use

IOCTL_PMC_BIS3_RL1_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (unsigned long integer)
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled channel interrupts. The latched status bits are cleared in the driver interrupt service routine. Refer to the interrupt function found in the UserApp for an example of use.

Write

PMC-BiSerial-III RL1 DMA data is written to the device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the target device, a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the number of bytes to be transferred, 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 I/O.

Read

PMC-BiSerial-III RL1 DMA data is read from the device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the target device, a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the number of bytes to be transferred, 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 I/O.

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