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

PC104p Bis3 Ba14

Software Manual

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Revision A1
10-2006-0803

PC104pBis3Ba14
WDF Device Drivers for the
PC104p BiSerial III Ba14

Dynamic Engineering

150 DuBois, Suite C

Santa Cruz, CA 95060

(831) 457-8891

FAX: (831) 457-4793

Manual Revision A. Revised August 7, 2018.

Table of Contents

Introduction

The PC104p-BiSerial-III driver was 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, using a Dynamic Engineering PCIBPC104pET carrier. It was also tested on a 32 bit Windows machine with an Intel Atom Processor E3825.

The PC104p-BiSerial-III board has a Spartan3-1500 Xilinx FPGA to implement the PCI interface, FIFOs and protocol control and status for four serial channels. Each channel has two 2k x 32-bit data FIFOs for data transmission and reception.

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 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 PC104p-BiSerial-III-Ba14 is recognized by the PCI bus configuration utility it will start the ba14 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 PC104p-BiSerial-III-Ba14 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include Ba14BasePublic.h, Ba14ChanPublic.h, Ba14Base.inf, Ba14Chan.inf, ba14base.cat, ba14chan.cat, Ba14Base.sys, Ba14Chan.sys, and WdfCoInstaller01009.dll.

Ba14BasePublic.h and Ba14ChanPublic.h are the C header files that define the Application Program Interface (API) for the Pc104pBis3Ba4 driver. 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 Ba14Base.inf, Ba14Chan.inf, ba14base.cat, ba14chan.cat, Ba14Base.sys, Ba14Chan.sys, and WdfCoInstaller01009.dll (Win7 version) to a CD or USB memory device as preferred.

With the PC104P BIS3 BA14 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 Pc104pBis3Ba14 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 Ba14BasePublic.h and Ba14ChanPublic.c. See main.c in the Pc104pBis3Ba14UserApp 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 PC104pBis3Ba14 driver are described below:

IOCTL_BA14_BASE_GET_INFO

Function: Returns the current driver version and instance number.
Input: none
Output: BA14_BASE_DRIVER_DEVICE_INFO structure
Notes: This call does not access the hardware, only driver parameters. See the definition of BA14_BASE_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 _BA14_BASE_DRIVER_DEVICE_INFO
{
   UCHAR    DriverVersion;
   UCHAR    RevMajor;
   UCHAR    SwitchValue;
   ULONG    InstanceNumber;
} BA14_BASE_DRIVER_DEVICE_INFO, *PBA14_BASE_DRIVER_DEVICE_INFO;
```

IOCTL_BA14_BASE_SET_CONFIG

Function: Sets the value for the Base Control Register
Input: BA14_BASE_CONFIG structure
Output: none
Notes: In current version of device only the Sel_Disable bit is used. Bit definitions can be found in the '_BASE' section under Register Definitions in the Hardware manual.

IOCTL_BA14_BASE_GET_CONFIG

Function: Returns the state of the Base Control register.
Input: none
Output: BA14_BASE_CONFIG structure
Notes: Bit definitions can be found in the '_BASE' section under Register Definitions in the Hardware manual.

IOCTL_BA14_BASE_GET_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_BA14_BASE_SET_DIR_TERM

Function: Sets the direction (input or output) and termination (off or on) of the 16 RS-485 I/O lines.
Input: BA14_BASE_DIR_TERM structure
Output: None
Notes: The bits in each of the structure fields operate on the respective I/O line i.e. if direction bit 0 is a one, I/O line 0 is an output; if termination bit 6 is a one, I/O line 6 is terminated etc. See the definition of BA14_BASE_DIR_TERM below. Bit definitions can be found in the '_DIR_TERM' section under Register Definitions in the Hardware manual. Refer to the data_cntl_test function found in the reg_test.c file in the UserApp for an example of use.

```
typedef struct _BA14_BASE_DIR_TERM
{
   USHORT   Direction;
   USHORT   Termination;
} BA14_BASE_DIR_TERM, *PBA14_BASE_DIR_TERM;
```

IOCTL_BA14_BASE_GET_DIR_TERM

Function: Returns the direction and termination of the 16 RS-485 I/O lines.
Input: None
Output: BA14_BASE_DIR_TERM structure
Notes: Bit definitions can be found in the '_DIR_TERM' section under Register Definitions in the Hardware manual. Refer to the data_cntl_test function found in the reg_test.c file in the UserApp for an example of use.

IOCTL_BA14_BASE_SET_IO_CONFIG

Function: Sets the source and data value of the 16 RS-485 output lines.
Input: BA14_BASE_485_DATA_CNTL structure
Output: None
Notes: The bits in each of the structure fields operate on the respective I/O line to specify the data when that line is configured as an output. When a bit in the Select field is a one, the data source for the I/O line is the register loaded from the Data field. Otherwise the data source is the transmit I/O state machine. Bit definitions can be found in the '_PARDAT_485' and '_PARCNTL' sections under Register Definitions in the Hardware manual. Refer to the data_cntl_test function found in the reg_test.c file in the UserApp for an example of use.

```
typedef struct _BA14_BASE_485_DATA_CNTL
{
   USHORT   Data;
   USHORT   Select;
} BA14_BASE_485_DATA_CNTL, *PBA14_BASE_485_DATA_CNTL;
```

IOCTL_BA14_BASE_GET_IO_CONFIG

Function: Returns the source and data value of the 16 RS-485 output lines.
Input: None
Output: BA14_BASE_485_DATA_CNTL structure
Notes: Bit definitions can be found in the '_PARDAT_485' and '_PARCNTL' sections under Register Definitions in the Hardware manual. Refer to the data_cntl_test function found in the reg_test.c file in the UserApp for an example of use.

IOCTL_BA14_BASE_READ_IO_DATA

Function: Returns the data values on the 16 RS-485 input lines..
Input: None
Output: Unsigned long integer
Notes:

IOCTL_BA14_BASE_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. Refer to the interrupt function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_BASE_ENABLE_INTERRUPT

Function: Enables the interrupts for.
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. Refer to the interrupt function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_BASE_DISABLE_INTERRUPT

Function: Disables the interrupt for given channel.
Input: None
Output: None
Notes: Clears the interrupt enable for. This IOCTL is used when interrupt processing is no longer desired. Refer to the interrupt function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_BASE_FORCE_INTERRUPT

Function: Causes a system interrupt to occur for given channel.
Input: None
Output: None
Notes: Causes a interrupt to be asserted on the PCI bus provided the interrupts are enabled. This IOCTL is used for development, to test interrupt processing. Refer to the interrupt function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_BASE_GET_ISR_STATUS

Function: Returns the interrupt status read in the last ISR.
Input: none
Output: Unsigned long integer
Notes: The status contains the status and control bits of the Status register read in the last ISR execution. Refer to the interrupt function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_GET_INFO

Function: Returns the Driver version and Instance number.
Input: None
Output: BA14_CHAN_CONT structure
Notes: Bit definitions can be found in the '_BASE 0' section under Register Definitions in the Hardware manual. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_SET_CONFIG

Function: Sets the channel configuration of the board.
Input: BA14_CHAN_CONT structure
Output: None
Notes: See the definition of BA14_CHAN_CONT below. Bit definitions can be found in the '_BASE 0' section under Register Definitions in the Hardware manual. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

```
typedef struct _BA14_CHAN_CONT
{
   BOOLEAN     Fifo_Bypass;
   BOOLEAN     Tx_Enable;
   BOOLEAN     Rx_Enable;
   BOOLEAN     Tx_Clr_Dis;
   CHAN_SELECT ChanSelect;
} BA14_CHAN_CONT, *PBA14_CHAN_CONT;
```

IOCTL_BA14_CHAN_GET_CONFIG

Function: Returns the channel configuration of the board.
Input: None
Output: BA14_CHAN_CONT structure
Notes: Bit definitions can be found in the '_BASE 0' section under Register Definitions in the Hardware manual. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_RESET_FIFOS

Function: Resets both the transmit and receive FIFOs.
Input: None
Output: None
Notes:

IOCTL_BA14_CHAN_GET_STATUS

Function: Returns the channel status.
Input: None
Output: Chan status value (unsigned long integer)
Notes: Returns Channel Interrupt Status information for a given board obtained from the 'ChanStatus' register. Bit definitions can be found in the '_INT 0' section under Register Definitions in the Hardware manual.

IOCTL_BA14_CHAN_SET_FIFO_LEVELS

Function: Sets receive almost full and transmit almost empty FIFO levels.
Input: BA14_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. Bit definitions can be found in the '_TXAMTC' and the '_RXAFC' sections under Register Definitions in the Hardware manual. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

```
typedef struct _BA14_CHAN_FIFO_LEVELS
{
   USHORT   AlmostFull;
   USHORT   AlmostEmpty;
} BA14_CHAN_FIFO_LEVELS, *PBA14_CHAN_FIFO_LEVELS;
```

IOCTL_BA14_CHAN_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. Bit definitions can be found in the '_TXAMTC' and the '_RXAFC' sections under Register Definitions in the Hardware manual. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_WRITE_FIFO

Function: Write one data word into the transmit FIFO.
Input: Unsigned long integer
Output: None
Notes: Loads a single transmit data word into the transmit FIFO. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_READ_FIFO

Function: Reads one data word from the receive FIFO.
Input: None
Output: Unsigned long integer
Notes: Reads a single receive data word from the receive FIFO. Refer to the ext_LB function found in the fifo_test.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_GET_FIFO_COUNTS

Function: Returns the number of words stored in the TX and RX FIFOs.
Input: None
Output: BA14_CHAN_FIFO_COUNTS
Notes: Returns the FIFO counts. See the definition of BA14_CHAN_FIFO_COUNTS below. Register definition can be found in the 'ChanFifoCnt' section under Register Definitions in the Hardware manual.

```
typedef struct _BA14_CHAN_FIFO_COUNTS
{
   USHORT   TxCount;
   USHORT   RxCount;
} BA14_CHAN_FIFO_COUNTS, *PBA14_CHAN_FIFO_COUNTS;
```

IOCTL_BA14_CHAN_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. Refer to the interrupt_chan function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_ENABLE_INTERRUPT

Function: Enables the interrupts for given channel.
Input: None
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. Refer to the interrupt_chan function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_DISABLE_INTERRUPT

Function: Disables the interrupt for given channel.
Input: None
Output: None
Notes: Clears the channel interrupt enable for given channel. This IOCTL is used when interrupt processing is no longer desired. Refer to the interrupt_chan function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur for given channel.
Input: None
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. Refer to the interrupt_chan function found in the interrupt.c file in the UserApp for an example of use.

IOCTL_BA14_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the last ISR.
Input: none
Output: Unsigned long integer
Notes: The status contains the status and control bits of the Chan Status register read in the last ISR execution. Refer to the interrupt_chan function found in the interrupt.c file in the UserApp for an example of use.

Write

PC104p-BiSerial-III 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 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.

Read

PC104p-BiSerial-III 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 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.

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