DYNAMIC ENGINEERING
150 DuBois St., Suite C Santa Cruz, CA 95060
831-457-8891 **Fax** 831-457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

IP-BISERIAL6-BA27

Driver Documentation

Developed with Windows Driver Foundation Ver1.9

Manual Revision A
Corresponding Hardware: Revision A
10-2016-3201
FLASH revision A1

IP-BIS6-BA27

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

Table of Contents

Introduction

The IP-BIS6-BA27 driver is a Windows device driver for the IP-Test Industry-pack (IP) module from Dynamic Engineering. This driver was developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

The IP-BIS6-BA27 driver package has two parts. The driver is installed into the Windows® OS, and the User Application "UserApp" executable.

The driver is 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. It is recommended to port the Register tests to your application to get started. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system. The test suite is designed to accommodate up to 5 boards. The number of boards can be expanded. See Main.c to increase the number of handles.

The hardware manual defines the pinout, the bitmaps and detailed configurations for each feature of the design. The driver handles all aspects of interacting with the hardware. For added explanations about what some of the driver functions do, please refer to the hardware manual.

We strive to make a useable product, and while we can guarantee operation we can't foresee all concepts for client implementation. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us, [engineering@dyneng.com] and we will consider and in many cases add them.

When the IP-BIS6-BA27 board is recognized by the IP Carrier Driver, the carrier driver will start the IP-BIS6-BA27 driver which will create a device object for the board. If more than one is found additional copies of the driver are loaded. The carrier driver will load the info storage register on the IP-BIS6-BA27 with the carrier switch setting and the slot number of the IP-BIS6-BA27 device. From within the IP-BIS6-BA27 driver the user can access the switch and slot information to determine the specific device being accessed when more than one are installed.

The reference software application has a loop to check for devices. The number of devices found, the locations, and device count are printed out at the top of the menu.

IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move 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 IP-BIS6-BA27 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include IpBis6Ba27.sys, IpBis6Ba27Public.h, IpPublic.h, WdfCoInstaller01009.dll, IpModDrivers.inf and ipmoddrivers.cat.

IpBis6Ba27Public.h and IpPublic.h are C header files that define the Application Program Interface (API) to the driver. These files are required at compile time by any application that wishes to interface with the driver, but are not needed for driver installation.

Note: Other IP module drivers are included in the package since they were all signed together and must be present to validate the digital signature. These other IP module driver files must be present when the IpBis6Ba27 driver is installed, to verify the digital signature in ipmoddrivers.cat, otherwise they can be ignored.

Warning: The appropriate IP carrier driver must be installed before any IP modules can be detected by the system.

Windows 7 Installation

Copy IpModDrivers.inf, ipmoddrivers.cat, WdfCoInstaller01009.dll, IpBis6Ba27.sys and the other IP module drivers to a removable memory device or other accessible location as preferred.

With the IP hardware installed, power-on the host computer.

- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an item for each IP module installed on the IP carrier. The label for a module installed in the first slot of the first PCIe3IP carrier would read ***PcieCar0 IP Slot A****.
- Right-click on the first device and select ***Update Driver Software***.
- Insert the removable memory device prepared above if necessary.
- Select ***Browse my computer for driver software***.
- Select ***Browse*** and navigate to the memory device or other location prepared above.
- Select ***Next***. The IpBis6Ba27 device driver should now be installed.
- Select ***Close*** to close the update window.
- Right-click on the remaining IP slot icons and repeat the above procedure as necessary.

***** If the [***Carrier] IP Slot [x]*** devices are not displayed, click on the ***Scan for hardware changes*** icon on the Device Manager 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 a globally unique identifier (GUID), which is defined in IpBis6Ba27Public.h.

The *main.c* file provided with the user test software can be used as an example to show how to obtain a handle to an IpBis6Ba27 device.

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single module. IOCTLs are called using the Win32 function DeviceIoControl() (see below), 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 IpBis6Ba27 driver are described below:

IOCTL_IP_BIS6_BA27_GET_INFO

Function: Returns the driver and firmware revisions, module instance number and location and other information.
Input: None
Output: DRIVER_IP_DEVICE_INFO structure
Notes: This call does not access the hardware, only stored driver parameters. NewIpCntl indicates that the module's carrier has expanded slot control capabilities. See the definition of DRIVER_IP_DEVICE_INFO below.

```
typedef struct _DRIVER_IP_DEVICE_INFO {
   UCHAR    DriverRev;         // Driver revision
   UCHAR    FirmwareRev;       // Firmware major revision
   UCHAR    FirmwareRevMin;    // Firmware minor revision
   UCHAR    InstanceNum;       // Zero-based device number
   UCHAR    CarrierSwitch;     // 0..0xFF
   UCHAR    CarrierSlotNum;    // 0..7 -> IP slots A, B, C, D, E, F, G or H
   UCHAR    CarDriverRev;      // Carrier driver revision
   UCHAR    CarFirmwareRev;    // Carrier firmware major revision
   UCHAR    CarFirmwareRevMin;// Carrier firmware minor revision
   UCHAR    CarCPLDRev;        //**Used for PCIe carriers only**0xFF for others
   UCHAR    CarCPLDRevMin;     //**Used for PCIe carriers only**0xFF for others
   BOOLEAN  Ip32MCapable;      // IP capable of both 8MHz and 32MHz operation
   BOOLEAN  NewIpCntl;         // New IP slot control interface
   WCHAR    LocationString[IP_LOC_STRING_SIZE];
} DRIVER_IP_DEVICE_INFO, *PDRIVER_IP_DEVICE_INFO;
```

IOCTL_IP_BIS6_BA27_SET_IP_CONTROL

Function: Sets various control parameters for the IP slot the module is installed in.
Input: IP_SLOT_CONTROL structure
Output: None
Notes: Controls the IP clock speed, interrupt enables and data manipulation options for the IP slot that the board occupies. See the definition of IP_SLOT_CONTROL below. For more information refer to the IP carrier hardware manual.

```
typedef struct _IP_SLOT_CONTROL {
   BOOLEAN  Clock32Sel;
   BOOLEAN  ClockDis;
   BOOLEAN  ByteSwap;
   BOOLEAN  WordSwap;
   BOOLEAN  WrIncDis;
   BOOLEAN  RdIncDis;
   UCHAR    WrWordSel;
   UCHAR    RdWordSel;
   BOOLEAN  BsErrTmOutSel;
   BOOLEAN  ActCountEn;
} IP_SLOT_CONTROL, *PIP_SLOT_CONTROL;
```

IOCTL_IP_BIS6_BA27_GET_IP_STATE

Function: Returns control/status information for the IP slot the module is installed in.
Input: None
Output: IP_SLOT_STATE structure
Notes: Returns the slot control parameters set in the previous call as well as status information for the IP slot that the board occupies. See the definition of IP_SLOT_STATE below.

```
typedef struct _IP_SLOT_STATE {
   BOOLEAN  Clock32Sel;
   BOOLEAN  ClockDis;
   BOOLEAN  ByteSwap;
   BOOLEAN  WordSwap;
   BOOLEAN  WrIncDis;
   BOOLEAN  RdIncDis;
   UCHAR    WrWordSel;
   UCHAR    RdWordSel;
   BOOLEAN  BsErrTmOutSel;
   BOOLEAN  ActCountEn;
 // Slot Status
   BOOLEAN  IpInt0En;
   BOOLEAN  IpInt1En;
   BOOLEAN  IpBusErrIntEn;
   BOOLEAN  IpInt0Actv;
   BOOLEAN  IpInt1Actv;
   BOOLEAN  IpBusError;
   BOOLEAN  IpForceInt;
   BOOLEAN  WrBusError;
   BOOLEAN  RdBusError;
} IP_SLOT_STATE, *PIP_SLOT_STATE;.
```

IOCTL_IP_BIS6_BA27_SET_BASE_CONFIG

Function: Sets base control register configuration.
Input: IP_BIS6_BA27_BASE_CONFIG structure
Output: none
Notes: See the definition of IP_BIS6_BA27_BASE_CONFIG below. Bit definitions can be found in the '_Base' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_BIS6_BA27_BASE_CONFIG {
   BOOLEAN     TxStart;
   BOOLEAN     TxIntEn;
   BOOLEAN     FaeIntEn;
   BOOLEAN     TxAutoClearEn;
   BOOLEAN     TxMode32;
   BOOLEAN     TxOdd;
   BOOLEAN     TxParOff;
   BOOLEAN     TxOrder;
   BOOLEAN     ClkIoTxSel;
} IP_BIS6_BA27_BASE_CONFIG, *PIP_BIS6_BA27_BASE_CONFIG;
```

IOCTL_IP_BIS6_BA27_GET_BASE_CONFIG

Function: Returns the base control configuration.
Input: none
Output: IP_BIS6_BA27_BASE_CONFIG structure
Notes: See the definition of IP_BIS6_BA27_BASE_CONFIG above. Bit definitions can be found in the '_Base' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_RX_CONFIG

Function: Sets rx control register configuration.
Input: IP_BIS6_BA27_RX_CONFIG structure
Output: none
Notes: See the definition of IP_BIS6_BA27_RX_CONFIG below. Bit definitions can be found in the '_Rx_cntl' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_BIS6_BA27_RX_CONFIG {
   BOOLEAN     RxStart;
   BOOLEAN     RxIntEn;
   BOOLEAN     FafIntEn;
   BOOLEAN     FifoBypassEn;
   BOOLEAN     RxAutoClearEn;
   BOOLEAN     RxMode32;
   BOOLEAN     RxOdd;
   BOOLEAN     RxParOff;
   BOOLEAN     RxOrder;
   BOOLEAN     IoReset;
} IP_BIS6_BA27_RX_CONFIG, *PIP_BIS6_BA27_RX_CONFIG;
```

IOCTL_IP_BIS6_BA27_GET_RX_CONFIG

Function: Returns the rx control configuration.
Input: none
Output: IP_BIS6_BA27_RX_CONFIG structure
Notes: See the definition of IP_BIS6_BA27_RX_CONFIG above. Bit definitions can be found in the '_Rx_cntl' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_IP_ID

Function: Returns IP module information.
Input: None
Output: IP-IDENTITY structure
Notes: See the definition of I IP_IDENTITY below.

```
typedef struct _IP_IDENTITY {
   UCHAR    IpManuf;
   UCHAR    IpModel;
   UCHAR    IpRevision;
   UCHAR    IpCustomer;
   USHORT   IpVersion;
} IP_IDENTITY, *PIP_IDENTITY;
```

IOCTL_IP_BIS6_BA27_GET_VERSION

Function: Returns the Module driver flash minor and major revisions.
Input: None
Output: IP_MOD_VERSION structure
Notes: See the definition of I IP_MOD_VERSION below. Bit definitions can be found under the '_REV' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_MOD_VERSION {
   UCHAR    minorFlashRev;
   UCHAR    majorFlashRev;
} IP_MOD_VERSION, *PIP_MOD_VERSION;
```

IOCTL_IP_BIS6_BA27_GET_STATUS

Function: Returns the status bits in the status register.
Input: none
Output: USHORT
Notes:. Bit definitions can be found in the '_BaseStatus' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_CLEAR_STATUS

Function: Clears the stinky status bits.
Input: USHORT
Output: none
Notes:. Bit definitions can be found in the '_BaseStatus' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_HALF_DIV

Function: Write a value to the half div registers.
Input: USHORT
Output: none
Notes: Definition can be found in the '_HalfDiv' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_HALF_DIV

Function: Reads from the half div registers.
Input: none
Output: USHORT
Notes: Definition can be found in the '_HalfDiv' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_DATA_DELAY

Function: Write a value to the data delay registers.
Input: USHORT
Output: none
Notes: Definition can be found in the '_DataDelay' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_DATA_DELAY

Function: Read from the data delay register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_DataDelay' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_FIFO_LEVEL

Function: Write a value to the almost empty and almost full registers.
Input: IP_BIS6_BA27_FIFO_LVL structure
Output: none
Notes: See definition of IP_BIS6_BA27_FIFO_LVL below. Definitions can be found in the '_AMT and _AFL' sections under Register Definitions in the Hardware manual.

```
typedef struct _IP_BIS6_BA27_FIFO_LVL {
   USHORT   AlmostEmpty;
   USHORT   AlmostFull;
} IP_BIS6_BA27_FIFO_LVL, *PIP_BIS6_BA27_FIFO_LVL;
```

IOCTL_IP_BIS6_BA27_GET_FIFO_LEVEL

Function: Read from the almost empty and almost full registers.
Input: none
Output: IP_BIS6_BA27_FIFO_LVL structure
Notes: See definition of IP_BIS6_BA27_FIFO_LVL above. Definitions can be found in the '_AMT and _AFL' sections under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_FIFO_DATA

Function: Write data to the FIFO registers.
Input: USHORT
Output: none
Notes: Definition can be found in the '_FIFO' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_FIFO_DATA

Function: Read data from the FIFO register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_FIFO' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_FIFO_STATUS

Function: Read from the FIFO status register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_FIFO_Status' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_DIRECTION

Function: Write a value to the direction register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_Direction' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_DIRECTION

Function: Read from the direction register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_Direction' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_TERMINATION

Function: Write a value to the termination register.
Input: USHORT
Output: none
Notes: Definition can be found in the '_Termination' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_TERMINATION

Function: Read from the termination register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_Termination' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_FIFO_COUNTS

Function: Read from the fifo count registers.
Input: none
Output: IP_BIS6_BA27_FIFO_CNT structure
Notes: : See definition of IP_BIS6_BA27_FIFO_CNT below. Definition can be found in the '_TxFifoCnt and _RxFifoCnt' sections under Register Definitions in the Hardware manual.

```
typedef struct _IP_BIS6_BA27_FIFO_CNT {
   USHORT   TxFifoCnt;
   USHORT   RxFifoCnt;
} IP_BIS6_BA27_FIFO_CNT, *PIP_BIS6_BA27_FIFO_CNT;
```

IOCTL_IP_BIS6_BA27_SET_DATA_TX

Function: Write data to the DataTx registers.
Input: USHORT
Output: none
Notes: Definition can be found in the '_DataTx' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_DATA_TX

Function: Read data from the DataTx register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_DataTx' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_DATA_IO

Function: Read data from the DataIo register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_DataIo' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_DELAY_COUNT

Function: Write to the delay count registers.
Input: ULONG
Output: none
Notes: Definition can be found in the '_DelayCnt' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_DELAY_COUNT

Function: Read from the delay count register.
Input: none
Output: ULONG
Notes: Definition can be found in the '_DelayCnt' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_PULSE_WIDTH

Function: Write to the pulse width registers.
Input: USHORT
Output: none
Notes: Definition can be found in the '_PulseWidth' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_GET_PULSE_WIDTH

Function: Read from the pulse width register.
Input: none
Output: USHORT
Notes: Definition can be found in the '_PulseWidth' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_SET_TX_DONE_CNTRL

Function: Write to the done pulse control registers.
Input: IP_BIS6_BA27_TX_DONE_CNTRL structure
Output: none
Notes: See definition of IP_BIS6_BA27_ TX_DONE_CNTRL below. The USHORT TxDoneControl, in the structure, gives the user control of the undefined bits in the register. Definition can be found in the '_DonePulseCntl' section under Register Definitions in the Hardware manual.

```
typedef struct _IP_BIS6_BA27_TX_DONE_CNTRL {
   BOOLEAN     TxDoneAssert;
   BOOLEAN     TxDoneEnable;
   USHORT      TxDoneControl;
} IP_BIS6_BA27_TX_DONE_CNTRL, *PIP_BIS6_BA27_TX_DONE_CNTRL;
```

IOCTL_IP_BIS6_BA27_GET_TX_DONE_CNTRL

Function: Read from the done pulse control register.
Input: none
Output: IP_BIS6_BA27_TX_DONE_CNTRL structure
Notes: Definition can be found in the '_DonePulseCntl' section under Register Definitions in the Hardware manual.

IOCTL_IP_BIS6_BA27_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. In order to un-register the event, set the event handle to NULL while making this call.

IOCTL_IP_BIS6_BA27_ENABLE_INTERRUPT

Function: Sets the master interrupt enable.
Input: None
Output: None
Notes: Sets the master interrupt enable, leaving all other bit values in the base register unchanged. This IOCTL is used in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver ISR. This allows the driver to set the master interrupt enable without knowing the state of the other base configuration bits.

IOCTL_IP_BIS6_BA27_DISABLE_INTERRUPT

Function: Clears the master interrupt enable.
Input: None
Output: None
Notes: Clears the master interrupt enable, leaving all other bit values in the base register unchanged. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IP_BIS6_BA27_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: none
Output: none
Notes: Causes an interrupt to be asserted on the IP bus. This IOCTL is used for development, to test interrupt processing.

IOCTL_IP_BIS6_BA27_SET_VECTOR

Function: Writes an 8 bit value to the interrupt vector register.
Input: UCHAR
Output: None
Notes: Required when used in non auto-vectored systems.

IOCTL_IP_BIS6_BA27_GET_VECTOR

Function: Returns the current interrupt vector value.
Input: none
Output: UCHAR
Notes:

IOCTL_IP_BIS6_BA27_GET_ISR_STATUS

Function: Returns the interrupt status, vector read in the last ISR, and the filtered data bits.
Input: none
Output: INT_STAT structure
Notes: The status contains the contents of the INT_STAT register and the FILTERED_DATA register read in the ISR.

```
// Interrupt status and vector
typedef struct _ISR_STATUS {
   USHORT   IntStatus;
   USHORT   IntVector;
} ISR_STATUS, *PISR_STATUS;
```

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