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

Bae9Base
&
Bae9Chan

Driver Documentation

Developed with Windows Driver Foundation

Revision B
Corresponding Hardware: Revision D, E
10-2005-0204/0205
Corresponding Firmware: Revision E

Bae9Base, Bae9Chan
WDF Device Drivers for the
PMC-BiSerial-III-BAE9
8-Channel PMC-Based UART Interface

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

Manual Revision B Revised February 25, 2013

Table of Contents

Introduction 5
Note 5
Driver Installation 5
Windows XP Installation 6
Windows 7 Installation 6
Driver Startup 7
IO Controls 7
IOCTL_BAE9_BASE_GET_INFO 7
IOCTL_BAE9_BASE_LOAD_PLL_DATA 8
IOCTL_BAE9_BASE_READ_PLL_DATA 8
IOCTL_BAE9_BASE_SET_CONFIG 8
IOCTL_BAE9_BASE_GET_CONFIG 9
IOCTL_BAE9_CHAN_GET_INFO 10
IOCTL_BAE9_CHAN_SET_CONFIG 10
IOCTL_BAE9_CHAN_GET_STATE 11
IOCTL_BAE9_CHAN_GET_STATUS 11
IOCTL_BAE9_CHAN_WRITE_TX_RAM 12
IOCTL_BAE9_CHAN_READ_TX_RAM 12
IOCTL_BAE9_CHAN_WRITE_RX_RAM 12
IOCTL_BAE9_CHAN_READ_RX_RAM 13
IOCTL_BAE9_CHAN_SET_TX_DMA_OFFSET 13
IOCTL_BAE9_CHAN_SET_RX_DMA_OFFSET 13
IOCTL_BAE9_CHAN_SET_TX_IO_OFFSET 13
IOCTL_BAE9_CHAN_SET_RX_IO_OFFSET 13
IOCTL_BAE9_CHAN_GET_TX_ADDR_OFFSETS 14
IOCTL_BAE9_CHAN_GET_RX_ADDR_OFFSETS 14
IOCTL_BAE9_CHAN_SET_TRIG_CONFIG 14
IOCTL_BAE9_CHAN_READ_TRIG_PARAMS 15
IOCTL_BAE9_CHAN_CLEAR_TRIG_COUNTS 15
IOCTL_BAE9_CHAN_READ_TRIG_LEVEL 15
IOCTL_BAE9_CHAN_SET_DISC_OUT_CONFIG 16
IOCTL_BAE9_CHAN_SET_TX_CONFIG 17
IOCTL_BAE9_CHAN_GET_TX_STATE 17
IOCTL_BAE9_CHAN_SET_RX_CONFIG 18
IOCTL_BAE9_CHAN_GET_RX_STATE 18
IOCTL_BAE9_CHAN_START_TX 18
IOCTL_BAE9_CHAN_STOP_TX 19
IOCTL_BAE9_CHAN_START_RX 19
IOCTL_BAE9_CHAN_STOP_RX 19
IOCTL_BAE9_CHAN_GET_RX_BYTE_COUNT 19
IOCTL_BAE9_CHAN_REGISTER_EVENT 20
IOCTL_BAE9_CHAN_ENABLE_INTERRUPT 20
IOCTL_BAE9_CHAN_DISABLE_INTERRUPT 20
IOCTL_BAE9_CHAN_FORCE_INTERRUPT 20
IOCTL_BAE9_CHAN_GET_ISR_STATUS 21

Write 22
Read 22
Warranty and Repair 22
Service Policy 23
Out of Warranty Repairs 23
For Service Contact: 23

Introduction

The Bae9Base and Bae9Chan drivers are Windows device drivers for the PMC-BiSerial-III BAE9 from Dynamic Engineering. These drivers were developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

The PMC-BiSerial-III board has a Spartan3-4000 Xilinx FPGA to implement the PCI interface, dual-port RAMs and protocol control and status for eight serial channels. Each channel has a 4k x 32-bit RAM for transmit data and a 2k x 32-bit RAM for received data.

When the PMC-BiSerial-III BAE9 is recognized by the PCI bus configuration utility it will start the Bae9Base and Bae9Chan drivers. The Bae9Base driver enumerates the channels and creates eight separate Bae9Chan device objects. This allows the I/O channels to be totally independent while the base driver controls the device items that are common. 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 I/O channel devices.

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 BAE9 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include Bae9Base.inf, Bae9Base.cat, Bae9Base.sys, Bae9BasePublic.h, Bae9Chan.inf, Bae9Chan.cat, Bae9Chan.sys, Bae9ChanPublic.h, WdfCoInstaller01009.dll, Bae9Test.exe and Bae9Test source files Bae9Test.cpp and Bae9Test.hpp.

Bae9BasePublic.h and Bae9ChanPublic.h are C header files that define the Application Program Interface (API) for the Bae9Base and Bae9Chan drivers. These files are required at compile time by any application that wishes to interface with the drivers, but are not needed for driver installation.

Bae9Test.exe is a menu-based console application that makes calls into the Bae9Base / Bae9Chan drivers to test each driver call without actually writing any application code. It is not required during the driver installation.

To run Bae9Test.exe, simply double-click on the test icon. A console window will open and the menu will be printed. Select a menu item and follow the prompts to execute the call. In Windows 7 Bae9Test must be run as administrator (right-click on icon).

Windows XP Installation

Copy Bae9Base.inf, Bae9Base.cat, Bae9Base.sys, Bae9Chan.inf, Bae9Chan.cat, Bae9Chan.sys and WdfCoInstaller01009.dll (XP version) to a floppy disk, CD or USB memory device as preferred.

With the PMC-BiSerial-III BAE9 hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.

- Insert the disk or memory device prepared above in the desired drive.
- Select ***No*** when asked to connect to Windows Updat***e***.
- Select ***Next***.
- Select ***Install the software automatically***. (If not found go to the next line)
- Select ***Install the software from a specific location***. (Specify your file's location)
- Select ***Next***.
- Select ***Finish*** to close the ***Found New Hardware Wizard***.

The system should now see the Bae9 I/O channels and reopen the ***New Hardware Wizard***. Proceed as above for each channel as necessary.

Windows 7 Installation

Copy Bae9Base.inf, Bae9Base.cat, Bae9Base.sys, Bae9Chan.inf, Bae9Chan.cat, Bae9Chan.sys and WdfCoInstaller01009.dll (Win7 version) to a floppy disk, CD or USB memory device as preferred.

With the PMC-BiSerial-III BAE9 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***.
- Browse to the location of the device prepared above.
- Select ***Next***.
- Select ***Close*** to close the update window.
  The system should now display the Bae9 I/O channels in the Device Manager.
- Right-click on each channel icon, select ***Update Driver Software*** and proceed as before.

***** 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 a globally unique identifier (GUID), which is defined in Bae9BasePublic.h and Bae9ChanPublic.h. See main.c in the PB3Bae9UserApp project for an example of how to acquire handles for the base and eight channel devices.

Note: In order to build an application you must link with setupapi.lib.

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() (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 Bae9Base driver are described below:

IOCTL_BAE9_BASE_GET_INFO

Function: Returns the device driver version, Xilinx flash revision, PLL device ID, user switch value, and device instance number.
Input: None
Output: BAE9_BASE_DRIVER_DEVICE_INFO structure
Notes: The switch value is the configuration of the 8-bit onboard dipswitch that has been selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. See the definition of BAE9_BASE_DRIVER_DEVICE_INFO below.

```
 // Driver/Device information
typedef struct _BAE9_BASE_DRIVER_DEVICE_INFO {
  UCHAR  DriverVersion;
  UCHAR  XilinxVersion;
  UCHAR  SwitchValue;
  ULONG  InstanceNumber;
  UCHAR  PllDeviceId;
} BAE9_BASE_DRIVER_DEVICE_INFO;
```

IOCTL_BAE9_BASE_LOAD_PLL_DATA

Function: Writes to the internal registers of the PLL.
Input:
BAE9_BASE_PLL_DATA structure
Output: None
Notes: The BAE9_BASE_PLL_DATA structure has only one field: Data – an array of 40 bytes containing the PLL register data to write. See below for the definition of BAE9_BASE_PLL_DATA.

```
 // Structures for IOCTLs
#define PLL_MESSAGE1_SIZE  16
#define PLL_MESSAGE2_SIZE  24
#define PLL_MESSAGE_SIZE  (PLL_MESSAGE1_SIZE + PLL_MESSAGE2_SIZE)

typedef struct _BAE9_BASE_PLL_DATA {
  UCHAR  Data[PLL_MESSAGE_SIZE];
} BAE9_BASE_PLL_DATA;
```

IOCTL_BAE9_BASE_READ_PLL_DATA

Function: Returns the contents of the internal registers of the PLL.
Input: None
Output: BAE9_BASE_PLL_DATA structure
Notes: The register data is written to the BAE9_BASE_PLL_DATA structure in an array of 40 bytes. See definition of BAE9_BASE_PLL_DATA above.

IOCTL_BAE9_BASE_SET_CONFIG

Function: Specifies the state of the ten trigger input signal terminations.
Input: BAE9_BASE_CONFIG structure
Output: None
Notes: This call controls the terminations for the ten trigger input signals that can be selected by any channel to trigger its TX UART and/or discrete output signal. See the definition of BAE9_BASE_CONFIG below.

```
typedef struct _BAE9_BASE_CONFIG {
  BOOLEAN  Trig0TermEn;   // Enable termination on trigger 0 input
  BOOLEAN  Trig1TermEn;   // Enable termination on trigger 1 input
  BOOLEAN  Trig2TermEn;   // Enable termination on trigger 2 input
  BOOLEAN  Trig3TermEn;   // Enable termination on trigger 3 input
  BOOLEAN  Trig4TermEn;   // Enable termination on trigger 4 input
  BOOLEAN  Trig5TermEn;   // Enable termination on trigger 5 input
  BOOLEAN  Trig6TermEn;   // Enable termination on trigger 6 input
  BOOLEAN  Trig7TermEn;   // Enable termination on trigger 7 input
  BOOLEAN  Trig8TermEn;   // Enable termination on trigger 8 input
  BOOLEAN  Trig9TermEn;   // Enable termination on trigger 9 input
} BAE9_BASE_CONFIG, *PBAE9_BASE_CONFIG;
```

IOCTL_BAE9_BASE_GET_CONFIG

Function: Returns the state of the trigger input terminations set in the previous call.
Input: None
Output: BAE9_BASE_CONFIG structure
Notes: See the definition of BAE9_BASE_CONFIG above.

The IOCTLs defined for the Bae9Chan driver are described below:

IOCTL_BAE9_CHAN_GET_INFO

Function: Returns the channel driver version and the channel instance number.
Input: None
Output: BAE9_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of BAE9_CHAN_DRIVER_DEVICE_INFO below.

```
// Driver/Device information
typedef struct _BAE9_CHAN_DRIVER_DEVICE_INFO {
  UCHAR  DriverVersion;
  ULONG  InstanceNumber;
} BAE9_CHAN_DRIVER_DEVICE_INFO;
```

IOCTL_BAE9_CHAN_SET_CONFIG

Function: Specifies fields in the channel control register.
Input: BAE9_CHAN_CONFIG structure
Output: None
Notes: This call controls channel configuration items that are not transmit or receive specific. TrigSelect can be any value between 0 and 9 and selects the discrete input line (24-33) to be used to trigger the TX UART and/or the discrete output signal. TxClkDiv field can be any value between 0x0F and 0x3F yielding I/O bit times of 16 to 64 I/O clock periods. TrigCountEn enables the trigger monitor function which counts the duration of the high and low levels of the trigger input signal and latches status bits if they are outside of the specified range. The four latch clear bits allow the corresponding latches to be cleared after they are read. FullDuplexEn selects full-duplex (‘1’) or half-duplex (‘0’) operation of the I/O subsystem. AutoDirSwitch enables the automatic switching from transmit to receive and vice versa when the current active direction is done. See the definition of BAE9_CHAN_CONFIG below.

```
typedef struct _BAE9_CHAN_CONFIG {
  UCHAR    TrigSelect;          // Trigger input mux select
  UCHAR    TxClkDiv;            // Transmit clock div count
  BOOLEAN  TrigCountEn;         // Count trigger hi and lo level time
 // Enable clearing of trigger limit latches
  BOOLEAN  TrigOnOvrLatClr;     //  Trigger '1' too long
  BOOLEAN  TrigOnUndrLatClr;    //  Trigger '1' too short
  BOOLEAN  TrigOffOvrLatClr;    //  Trigger '0' too long
  BOOLEAN  TrigOffUndrLatClr;   //  Trigger '0' too short
  BOOLEAN  FullDuplexEn;        // Enable full-duplex mode
  BOOLEAN  AutoDirSwitch;       // Auto-switch direction in half-duplex mode
} BAE9_CHAN_CONFIG;
```

IOCTL_BAE9_CHAN_GET_STATE

Function: Returns the fields set in the previous call as well as the states of the master and read and write interrupt enables.
Input: None
Output: BAE9_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. See the definition of BAE9_CHAN_STATE below.

```
typedef struct _BAE9_CHAN_STATE {
  UCHAR    TrigSelect;
  UCHAR    TxClkDiv;
  BOOLEAN  TrigCountEn;
  BOOLEAN  TrigOnOvrLatClr;
  BOOLEAN  TrigOnUndrLatClr;
  BOOLEAN  TrigOffOvrLatClr;
  BOOLEAN  TrigOffUndrLatClr;
  BOOLEAN  FullDuplexEn;
  BOOLEAN  AutoDirSwitch;
  BOOLEAN  MIntEn;         // Master interrupt enable (read only)
  BOOLEAN  WrDmaEn;        // Write DMA enable (read only)
  BOOLEAN  RdDmaEn;        // Read DMA enable (read only)
} BAE9_CHAN_STATE;
```

IOCTL_BAE9_CHAN_GET_STATUS

Function: Returns the channel's status register value and clears the latched status bits.
Input: None
Output: Value of the channel's status register (unsigned long integer)
Notes: See the status bit definitions below. Only the bits in CHAN_STAT_MASK will be returned. The bits in CHAN_STAT_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. The bits in CHAN_STAT_LIM_LAT_MASK are cleared by IOCTL_BAE9_CHAN_READ_TRIG_PARAMS, not by this call

```
 // Status bit definitions
#define CHAN_STAT_TX_INT          0x00000001
#define CHAN_STAT_RX_INT          0x00000002
#define CHAN_STAT_RX_PERR_LAT     0x00000004
#define CHAN_STAT_RX_FERR_LAT     0x00000008
#define CHAN_STAT_WR_DMA_ERR      0x00000010
#define CHAN_STAT_RD_DMA_ERR      0x00000020
#define CHAN_STAT_WR_DMA_INT      0x00000040
#define CHAN_STAT_RD_DMA_INT      0x00000080
#define CHAN_STAT_WR_DMA_RDY      0x00000100
```

```
#define CHAN_STAT_RD_DMA_RDY      0x00000200
#define CHAN_STAT_TRIG_ON_OVER    0x00001000
#define CHAN_STAT_TRIG_ON_UNDER   0x00002000
#define CHAN_STAT_TRIG_OFF_OVER   0x00004000
#define CHAN_STAT_TRIG_OFF_UNDER  0x00008000
#define CHAN_STAT_LOC_INT         0x08000000
#define CHAN_STAT_INT_ACTIVE      0x80000000

#define CHAN_STAT_LATCH_MASK      0x0000003F
#define CHAN_STAT_LIM_LAT_MASK    0x0000F000
#define CHAN_STAT_MASK            0x8800F3FF
```

IOCTL_BAE9_CHAN_WRITE_TX_RAM

Function: Writes a single 32-bit word to the specified address in the transmit RAM.
Input: Address offset and data value to write (BAE9_MEM_WORD_WRITE structure)
Output: None
Notes: See the definition of BAE9_MEM_WORD_WRITE below. Address indexes 32-bit words and has a maximum value of 0xFFF (16 Kbytes).

```
typedef struct _BAE9_MEM_WORD_WRITE {
  ULONG  Address;  // RAM address offset
  ULONG  Data;     // RAM data to write
} BAE9_MEM_WORD_WRITE;
```

IOCTL_BAE9_CHAN_READ_TX_RAM

Function: Reads a single 32-bit word from the specified address in the transmit RAM.
Input: Address (unsigned long integer)
Output: Data (unsigned long integer)
Notes: Address indexes 32-bit words and has a maximum value of 0xFFF (16 Kbytes).

IOCTL_BAE9_CHAN_WRITE_RX_RAM

Function: Writes a single 32-bit word to the specified address in the receive RAM.
Input: Address offset and data value to write (BAE9_MEM_WORD_WRITE structure)
Output: None
Notes: See the definition of BAE9_MEM_WORD_WRITE below. Address indexes 32-bit words and has a maximum value of 0x7FF (8 Kbytes).

```
typedef struct _BAE9_MEM_WORD_WRITE {
  ULONG    Address;  // RAM address offset
  ULONG    Data;     // RAM data to write
} BAE9_MEM_WORD_WRITE;
```

IOCTL_BAE9_CHAN_READ_RX_RAM

Function: Reads a single 32-bit word from the specified address in the receive RAM.
Input: Address (unsigned long integer)
Output: Data (unsigned long integer)
Notes: Address indexes 32-bit words and has a maximum value of 0x7FF (8 Kbytes).

IOCTL_BAE9_CHAN_SET_TX_DMA_OFFSET

Function: Specifies the transmit RAM starting address offset for the next write DMA.
Input: Byte Address (unsigned long integer)
Output: None
Notes: The byte address is used by the driver to calculate the Local Address field of the DMA chaining descriptors. This is a byte address and the two least significant bits will be stripped by the hardware to create the actual RAM address when the DMA is executed. Only long words can be addressed, individual byte operations are not supported. The address can be any value divisible by four from 0 to 0x3FFC.

IOCTL_BAE9_CHAN_SET_RX_DMA_OFFSET

Function: Specifies the receive RAM starting address offset for the next read DMA.
Input: Byte Address (unsigned long integer)
Output: None
Notes: The byte address is used by the driver to calculate the Local Address field of the DMA chaining descriptors. This is a byte address and the two least significant bits will be stripped by the hardware to create the actual RAM address when the DMA is executed. Only long words can be addressed, individual byte operations are not supported. The address can be any value divisible by four from 0 to 0x1FFC.

IOCTL_BAE9_CHAN_SET_TX_IO_OFFSET

Function: Specifies the transmit RAM starting address offset for the next message to send.
Input: Word Address (unsigned long integer)
Output: None
Notes: The address indexes 32-bit words and can be any value between 0 and 0xFFF.

IOCTL_BAE9_CHAN_SET_RX_IO_OFFSET

Function: Specifies the receive RAM starting address offset for storing the next received message.
Input: Word Address (unsigned long integer)
Output: None
Notes: The address indexes 32-bit words and can be any value between 0 and 0x7FF.

IOCTL_BAE9_CHAN_GET_TX_ADDR_OFFSETS

Function: Returns the next transmit RAM address offsets for write DMA and transmit I/O.
Input: None
Output: BAE9_MEM_ADDR_OFFSETS
Notes: See the definition of BAE9_MEM_ADDR_OFFSETS below. The DMA Address field is latched at the end of the last DMA, a new DMA offset will not affect this value until a new DMA is performed.

```
typedef struct _BAE9_MEM_ADDR_OFFSETS {
  USHORT  DmaAddr;  // Starting address for next DMA
  USHORT  IoAddr;   // Next address for I/O read/write
} BAE9_MEM_ADDR_OFFSETS;
```

IOCTL_BAE9_CHAN_GET_RX_ADDR_OFFSETS

Function: Returns the next receive RAM address offsets for read DMA and receive I/O.
Input: None
Output: BAE9_MEM_ADDR_OFFSETS
Notes: See the definition of BAE9_MEM_ADDR_OFFSETS above. The DMA Address field is latched at the end of the last DMA, a new DMA offset will not affect this value until a new DMA is performed.

IOCTL_BAE9_CHAN_SET_TRIG_CONFIG

Function: Specifies the count value limits for the trigger monitor limit latches.
Input: BAE9_CHAN_TRIG_CONFIG
Output: None
Notes: The field counts for the on and off, min and max limits were increased from 16 to 22 bits in design rev. D. See the definition of BAE9_CHAN_TRIG_CONFIG below.

```
typedef struct _BAE9_CHAN_TRIG_CONFIG {
// Change for firmware rev.D: The following fields were increased from
 //  16 to 22 bits, which required changing the field type to ULONG
  ULONG    OnMaxLimit;    // Max count for trigger = '1'
  ULONG    OnMinLimit;    // Min count for trigger = '1'
  ULONG    OffMaxLimit;   // Max count for trigger = '0'
  ULONG    OffMinLimit;   // Min count for trigger = '0'
} BAE9_CHAN_TRIG_CONFIG;
```

IOCTL_BAE9_CHAN_READ_TRIG_PARAMS

Function: Returns the state of the four limit latches and the time counts of the last high and low logic levels for the discrete input trigger signal.
Input: None
Output: BAE9_CHAN_TRIG_STATE
Notes: The field counts for the OnTime and OffTime were increased from 16 to 22 bits in design rev. D. The ability to output a constant level was also added. See the definition of BAE9_CHAN_TRIG_STATE below.

```
typedef struct _BAE9_CHAN_TRIG_STATE {
  BOOLEAN  OnOverLat;     // Max count exceeded (trigger = '1')
  BOOLEAN  OnUnderLat;    // Min count not reached (trigger = '1')
  BOOLEAN  OffOverLat;    // Max count exceeded (trigger = '0')
  BOOLEAN  OffUnderLat;   // Min count not reached (trigger = '0')
 // Change for firmware rev.D: The following fields were increased from
 //  16 to 22 bits, which required changing the field type to ULONG
  ULONG    OnTime;        // Time count for last '1' level
  ULONG    OffTime;       // Time count for last '0' level
} BAE9_CHAN_TRIG_STATE;
```

IOCTL_BAE9_CHAN_CLEAR_TRIG_COUNTS

Function: Stops the trigger monitor high and low level counters and clears the counts.
Input: None
Output: None
Notes: The ability to detect a constant level trigger input was added in design rev. E. This call is used in preparation for detecting such a signal. The trigger monitor counters are halted and cleared and they will not start again until a rising or falling edge occurs on the selected trigger input signal.

IOCTL_BAE9_CHAN_READ_TRIG_LEVEL

Function: Reports a steady-state on the trigger input signal for the selected channel.
Input: None
Output: BAE9_CHAN_TRIG_LEVEL
Notes: The ability to detect a constant level trigger input was added in design rev. E. If the trigger input signal for the selected channel stays at a constant level after the counters were cleared, one of the two fields in the output structure will be true and the other will be false depending on the polarity of the signal. See the definition of BAE9_CHAN_TRIG_LEVEL below.

```
 // Change for firmware rev.E: The following structure was added
typedef struct _BAE9_CHAN_TRIG_LEVEL {
  BOOLEAN  Level_0;  // True if trigger signal is steady-state '0'
  BOOLEAN  Level_1;  // True if trigger signal is steady-state '1'
} BAE9_CHAN_TRIG_LEVEL;
```

IOCTL_BAE9_CHAN_SET_DISC_OUT_CONFIG

Function: Specifies various parameters that control the behavior of the discrete output signal.
Input:
Output: None
Notes: The field counts for the Delay, Period and Duty (cycle) were increased from 16 to 22 bits in design rev. D. The ability to output a constant level was also added. See the definition BAE9_CHAN_DISC_OUT_CONFIG below.

```
typedef enum _BAE9_CHAN_OUT_MODE {
  BAE9_TRIGGERED,
  BAE9_PERIODIC,
  BAE9_ONE_SHOT,
  BAE9_TRIG_PERIODIC
} BAE9_CHAN_OUT_MODE;

typedef struct _BAE9_CHAN_DISC_OUT_CONFIG {
  BOOLEAN              Enable;        // Enable discrete output signal
 // Change for firmware rev.D: The following fields were increased from
 //  16 to 22 bits, which required changing the field type to ULONG
  ULONG                Delay;         // Delay count after trigger seen
  ULONG                Period;        // Count for full cycle
  ULONG                Duty;          // Count for active part of cycle
  BAE9_CHAN_OUT_MODE   Mode;          // Mode of operation (see above)
  BOOLEAN              LevelOut;      // Outputs a constant level
  BOOLEAN              InvTrigger;    // Invert the trigger input
  BOOLEAN              InvOutput;     // Invert the signal output
} BAE9_CHAN_DISC_OUT_CONFIG;
```

IOCTL_BAE9_CHAN_SET_TX_CONFIG

Function: Specifies various parameters that control the behavior of the transmitter.
Input: BAE9_CHAN_TX_CONFIG structure
Output: None
Notes: See the definition of BAE9_CHAN_TX_CONFIG and BAE9_CHAN_PAR_SEL below and the definition of BAE9_CHAN_OUT_MODE above.

```
typedef enum _BAE9_CHAN_PAR_SEL {
  BAE9_NONE,
  BAE9_ODD,
  BAE9_EVEN,
  BAE9_MARK,
  BAE9_SPACE
} BAE9_CHAN_PAR_SEL;

typedef struct _BAE9_CHAN_TX_CONFIG {
  BOOLEAN             TxIntEnable;  // Transmit done interrupt enable
  BOOLEAN             ClearEnable;  // Enable clearing start bit when done
  BOOLEAN             StopTwoSel;   // Use two stop-bits in serial output
  BAE9_CHAN_PAR_SEL   Parity;       // Parity definition (see above)
  USHORT              Delay;        // Delay count after trigger seen
  USHORT              Period;       // Count for full cycle
  BAE9_CHAN_OUT_MODE  Mode;         // Mode of operation (see above)
  BOOLEAN             InvTrigger;   // Invert the trigger input
} BAE9_CHAN_TX_CONFIG;
```

IOCTL_BAE9_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: BAE9_CHAN_TX_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 the definition of BAE9_CHAN_TX_STATE below.

```
typedef struct _BAE9_CHAN_TX_STATE {
  BOOLEAN             Enabled;      // Transmitter is enabled (read only)
  BOOLEAN             TxIntEnable;
  BOOLEAN             ClearEnable;
  BOOLEAN             StopTwoSel;
  BAE9_CHAN_PAR_SEL   Parity;
  BAE9_CHAN_OUT_MODE  Mode;
  BOOLEAN             InvTrigger;
} BAE9_CHAN_TX_STATE;
```

IOCTL_BAE9_CHAN_SET_RX_CONFIG

Function: Specifies various parameters that control the behavior of the receiver.
Input: BAE9_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 the definition of BAE9_CHAN_RX_CONFIG below.

```
typedef struct _BAE9_CHAN_RX_CONFIG {
  BOOLEAN            RxIntEnable;  // Receive done interrupt enable
  BOOLEAN            ClearEnable;  // Enable clearing start bit when done
  BOOLEAN            TermEnable;   // Enable the termination for the I/O line
  BOOLEAN            StopTwoSel;   // Check two stop-bits in serial input
  BOOLEAN            PerrIntEn;    // Parity error interrupt enable
  BOOLEAN            FerrIntEn;    // Framing error interrupt enable
  BAE9_CHAN_PAR_SEL  Parity;       // Parity definition (see above)
} BAE9_CHAN_RX_CONFIG;
```

IOCTL_BAE9_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: BAE9_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 the definition of BAE9_CHAN_RX_STATE below.

```
typedef struct _BAE9_CHAN_RX_STATE {
  BOOLEAN            Enabled;      // Receiver is enabled (read only)
  BOOLEAN            RxIntEnable;
  BOOLEAN            ClearEnable;
  BOOLEAN            TermEnable;
  BOOLEAN            StopTwoSel;
  BOOLEAN            PerrIntEn;
  BOOLEAN            FerrIntEn;
  BAE9_CHAN_PAR_SEL  Parity;
} BAE9_CHAN_RX_STATE;
```

IOCTL_BAE9_CHAN_START_TX

Function: Starts a data transmission.
Input: Number of bytes to send (unsigned short integer)
Output: None
Notes: The specified number of bytes will be sent.

IOCTL_BAE9_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.

IOCTL_BAE9_CHAN_START_RX

Function: Enables the receiver to look for data and store it in the receive RAM.
Input: None
Output: None
Notes: The first word of each stored message will be a status word. The upper 16 bits contain the number of bytes received in the message. The lower 16 bits contain the word address of the start of the next message i.e. the status word for that message. The message data starts with the next word after the status word.

IOCTL_BAE9_CHAN_STOP_RX

Function: Abort or cancel data reception.
Input: None
Output: None
Notes: This call will cancel a receive request that has not started or stop a reception in progress. It also disables the receiver when data reception is no longer desired.

IOCTL_BAE9_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.

IOCTL_BAE9_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() 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 a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts do not cause the event to be signaled.

IOCTL_BAE9_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 user interrupts. The master interrupt enable is disabled in the driver interrupt service routine. Therefore this command must be run after each user interrupt occurs to re-enable the interrupts.

IOCTL_BAE9_CHAN_DISABLE_INTERRUPT

Function: Disables the master interrupt.
Input: None
Output: None
Notes: This call is used when user interrupt processing is no longer desired.

IOCTL_BAE9_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 if the master interrupt is enabled. This IOCTL is used for test and development, to test interrupt processing.

IOCTL_BAE9_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (BAE9_CHAN_INT_STAT)
Notes: Returns the status that was read in the interrupt service routine for the last user interrupt serviced. Latched status bits (bits in STATUS_LATCH_MASK) that were set when the status was read in the ISR are returned along with the other status bits, but will have been automatically cleared in the interrupt DPC. See the definition of BAE9_CHAN_INT_STAT below.

```
typedef struct _BAE9_CHAN_INT_STAT {
  ULONG    Status;  // Value of status register read in ISR
  BOOLEAN  New;     // True if the status has changed since the last call
} BAE9_CHAN_INT_STAT;
```

Write

PMC-BiSerial-III BAE9 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. The data will be written to the transmit RAM starting at the byte address specified by the IOCTL_BAE9_CHAN_SET_TX_DMA_OFFSET call.

Read

PMC-BiSerial-III BAE9 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. The data will be read from the receive RAM starting at the byte address specified by the IOCTL_BAE9_CHAN_SET_RX_DMA_OFFSET call.

Warranty and Repair

Dynamic Engineering warrants this product to be free from defects under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, Dynamic Engineering's sole responsibility shall be to repair, or at Dynamic Engineering's sole option to replace, the defective product.

Dynamic Engineering's warranty of and liability for defective products is limited to that set forth herein. Dynamic Engineering disclaims and excludes all other product warranties and product liability, expressed or implied, including but not limited to any implied warranties of merchandisability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Dynamic Engineering’s products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

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 the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer’s making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer’s invoicing policy.

Out of Warranty Repairs

Out of warranty support will be billed. The current minimum repair charge is $125. An open PO will be required.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois, Suite C Santa Cruz, CA 95060
(831) 457-8891 Fax (831) 457-4793
support@dyneng.com