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

PMC BiSerial III BA19 Base & Channels

Driver Documentation

Win32 Driver Model

Manual Revision A
Corresponding Hardware: Revision D
10-2005-0204
Corresponding Firmware:
BA19: Design A, Revision 2

BA19Base & BA19Chan
WDM Device Drivers for the
PMC-BiSerial-III-BA19

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

Manual Revision A Revised July 9, 2009

Table of Contents

Introduction

The BA19Base and BA19Chan drivers are Win32 driver model (WDM) device drivers for the PMC-BiSerial-III-BA19 from Dynamic Engineering.

The BA19 driver package has three parts. The driver is installed into the Windows® OS, the test executable and the User Application "Userap" executable.

The driver and test are delivered as installed or executable items to be used directly or indirectly by the user. The Userap code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

The "test" executable allows the user to use the driver in script form from a DOS window. Each driver call can be accessed, parameters set and returned. Normally not needed or used by the integrator, but a very handy tool in certain circumstances. The test executable has a "help" menu to explain the calls, parameters and returned information.

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. For example most Dynamic Engineering PCI based designs support DMA. DMA is demonstrated with the memory based loop-back tests. The tests can be ported and modified to fit your requirements.

The test software can be ported to your application to provide a running start. It is recommended to port the switch and status 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 hardware has features common to the board level and features that are set apart in "channels". The channels have the same offsets within the channel, and the same status and control bit locations allowing for symmetrical software in the calling routines. The driver supports the channels with a variable passed in to identify which channel is being accessed. The hardware manual defines the pinout for each channel and the

bitmaps and detailed configurations for each channel. The driver handles all aspects of interacting with the channels and base features.

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.

The PMC BiSerial III board has a Spartan3 Xilinx FPGA to implement the PCI interface, FIFOs and protocol control and status for the IO. The IO are grouped into two ports; both part of channel 0. A Master port which requests data from the Target and a Target port are provided. Please refer to the HW manual for a much more complete description of the HW features.

When the PMC-BiSerial-III-BA19 board is recognized by the PCI bus configuration utility it will start the PmcBis3BA19Base driver which will create a device object for each board, initialize the hardware, create a child devices for the channel and request loading of the PmcBis3BA19Chan driver. The PmcBis3BA19Chan driver will create a device object for the I/O channel and perform initialization on the channel. 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-BA19 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include driver: BA19Base.sys, PmcBA19.inf, DDBA19Base.h, BA19BaseGUID.h, BA19Chan.sys, DDBA19Chan.h, BA19ChanGUID.h. Driver Test: BA19Test.exe, Userap: User Application source files.

BA19BaseGUID.h and BA19ChanGUID.h are C header files that define the device interface identifiers for the drivers. DDBA19Base.h and DDBA19Chan.h files are C header files that define the Application Program Interface (API) to the drivers. These files are required at compile time by any application that wishes to interface with the drivers, but they are not needed for driver installation. The files are included with the Userap fileset.

BA19Test.exe is a sample Win32 console applications that makes calls into the BA19Base/BA19Chan drivers to test each driver call without actually writing any application code. They are not required during driver installation either. Please note that the test driver software is incomplete at this time. Please refer to the User Application software package as a reference for using the driver.

To run BA19Test, open a command prompt console window and type ***BA19Test -d0 -?*** to display a list of commands (the PmcBis3BA19Test.exe file must be in the directory that the window is referencing). The commands are all of the form ***BA19Test -dn -im*** where ***n*** and ***m*** are the device number and PmcBis3BA19Base driver ioctl number respectively or ***BA19Test -cn -im*** where ***n*** and ***m*** are the channel number (0-1) and PmcParTtlBA19Chan driver ioctl number respectively.

This test application is intended to test the proper functioning of each driver call, **not** for normal operation. Many integration efforts will never need the debugger capability that the test menu represents. The test capability will allow the designer to access the card without any other software in the way to make sure that the system can “see” the card and to do basic card manipulations.

Windows 2000 Installation

Copy PmcBA19.inf, BA19Base.sys and BA19Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Select ***Next***.
_ Select ***Search for a suitable driver for my device.***
_ Select ***Next***.
_ Insert the disk prepared above in the desired drive.
_ Select the appropriate drive e.g. ***Floppy disk drives***.
_ Select ***Next***.
_ The wizard should find the PmcBA19.inf file.
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.*** Repeat this for each channel as necessary.

Windows XP Installation

Copy PmcBA19.inf, BA19Base.sys and BA19Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Insert the disk prepared above in the desired drive.
_ Select ***No when asked to connect to Windows Update***.
_ Select ***Next***.
_ Select ***Install the software automatically.***
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.*** Proceed as above for each channel as necessary.

Driver Startup

Once the drivers have been installed they will start automatically when the system recognizes the hardware.

Handles can be opened to a specific board by using the CreateFile() function call and passing in the device names obtained from the system.

The interfaces to the devices are identified using globally unique identifiers (GUIDs), which are defined in BA19BaseGUID.h and BA19ChanGUID.h.

The User Application software contains a file called “main.c”. Main has the initialization needed to get the handles to the base and channel assets of the installed PMC-BiSerial-III-BA19 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() (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 BA19Base driver are described below:
Please note that the address map is included in the DD file for reference when writing your own driver for a different OS.

IOCTL_BA19_BASE_GET_INFO
Function: Return the Instance Number, Switch value, PLL device ID, Xilinx rev and Current Driver Version
Input: None
Output: BA19_BASE_DRIVER_DEVICE_INFO : Structure
Notes: Switch value is the configuration of the on-board dip-switch that has been set by the User (see the board silk screen for bit position and polarity). The PLL ID is the device address of the PLL device. This value, which is set at the factory, is usually 0x69 but may also be 0x6A. See DDBA19Base.h for the definition of SPWR_BASE_DRIVER_DEVICE_INFO.

IOCTL_BA19_BASE_LOAD_PLL_DATA
Function: Loads the internal registers of the PLL.
Input: BA19_BASE_PLL_DATA structure
Output: None
Notes:

IOCTL_BA19_BASE_READ_PLL_DATA
Function: Returns the contents of the PLL's internal registers
Input: None
Output: BA19_BASE_PLL_DATA structure
Notes: The register data is output in the BA19_BASE_PLL_DATA structure in an array of 40 bytes.

IOCTL_BA19_BASE_SET_BASEREG
Function: Write to Base Control Register - general access to base control register of card, use with bit definitions
Input: ULONG
Output: none
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_BA19_BASE_GET_BASEREG
Function: Read from Base Control Register - general access from base control register of card, use with bit definitions
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_BA19_BASE_GET_STATUS
Function: Read from Status Register
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access from the register. See DDBA19Base.h for bit map information. See the HW manual for exact definitions of bits.

The IOCTLs defined for the PmcBis3BA19Chan driver are described below:
In the BA19 implementation both the Master and the Target interface are implemented within the same channel (0). The Master requests data and is a Receive function. The Target provides data and is a transmit function. Some documentation may show Rx or Tx instead of Master or Target.

Address and bit map information is included in the DDBA19Chan.h file to support those who are writing drivers for other OS.

IOCTL_BA19_CHAN_GET_INFO
Function: Return the Instance Number and Current Driver Version
Input: None
Output: BA19_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of BA19_CHAN_DRIVER_DEVICE_INFO in the DDBA19Chan.h header file.

IOCTL_BA19_CHAN_GET_STATUS
Function: Return the value of the status register and clear latched bits
Input: None
Output: Status register value(ULONG)
Notes: Latched interrupt status and DMA error bits are cleared by read – [call writes back and clears bits]. Other Latched Error bits not cleared by read. See quick reference status bits below. Defines available in DDBA19Chan.h Detailed definitions are available in the HW manual.

STAT_TARGET_FIFO_MT	0x00000001 //0 set when TARGET FIFO is empty
STAT_TARGET_FIFO_AE	0x00000002 //1 set when TARGET FIFO is Almost Empty
STAT_TARGET_FIFO_FULL	0x00000004 //2 set when TARGET FIFO is Full
STAT_MASTER_FIFO_MT	0x00000010 //4 set when MASTER FIFO is Empty
STAT_MASTER_FIFO_AF	0x00000020 //5 set when MASTER FIFO is Almost Full
STAT_MASTER_FIFO_FULL	0x00000040 //6 set when MASTER FIFO is Full
STAT_MASTER_FIFO_AF_INT	0x00000080 //7 set when MASTER FIFO is Almost full based on Interrupt level program level
STAT_TARGET_FIFO_ERR	0x00000200 //9 Receive Done Interrupt Occurred
STAT_MASTER_SYNC_ERR	0x00000400 //10 Transmit FIFO Interrupt Occurred
STAT_MASTER_FIFO_ERR	0x00000800 //11 Receive FIFO Interrupt Occurred
STAT_WR_DMA_ERR	0x00001000 //12 write DMA error
STAT_RD_DMA_ERR	0x00002000 //13 read DMA error
STAT_WR_DMA_INT	0x00004000 //14 write DMA Interrupt
STAT_RD_DMA_INT	0x00008000 //15 read DMA Interrupt

STAT_MASTER_IDLE	0x00010000 //16 set when Master is in Idle state
STAT_TARGET_IDLE	0x00020000 //17 set when Target is in Idle state
STAT_DMA_RD_IDLE	0x00040000 //18 set when Burst Out [read] DMA state-machine is in the idle state
STAT_DMA_WR_IDLE	0x00080000 //19 set when Burst In [write] DMA state-machine is in the idle state
MASTER_LVL_INT_MSKED	0x20000000 //29 enabled Almost Full Interrupt request, before channel mask
TARGET_LVL_INT_MSKED	0x40000000 //30 enabled Almost empty Interrupt request, before channel mask
STAT_ACTIVE_INT	0x80000000 //31 channel interrupt is active [after mask and includes DMA]

IOCTL_BA19_CHAN_CLR_STATUS
Function: Clear Error Bits latched and not cleared by status read
Input: ULONG
Output: none
Notes: Clear latched error bits. Allows polling on FIFO status without losing potential Error conditions. Write back with same bit position set to clear. Defines available in DDBA19Chan.h Detailed definitions are available in the HW manual.

IOCTL_BA19_CHAN_SET_FIFO_LEVELS
Function: Sets the transmitter almost empty and receiver almost full levels for the channel.
Input: BA19_CHAN_FIFO_LEVELS structure
Output: None
Notes: The FIFO counts are compared to these levels to determine the value of the STAT_TX_FF_AMT and STAT_RX_FF_AFL status bits.

IOCTL_BA19_CHAN_GET_FIFO_LEVELS
Function: Returns the transmitter almost empty and receiver almost full levels for the channel.
Input: None
Output: BA19_CHAN_FIFO_LEVELS structure
Notes:

IOCTL_BA19_CHAN_GET_FIFO_COUNTS
Function: Returns the number of data words in FIFO's.
Input: None
Output: BA19_CHAN_FIFO_COUNTS structure
Notes: Returns the actual Target [TX] FIFO data counts and count including DMA pipeline Master [RX] FIFO.

IOCTL_BA19_CHAN_RESET_FIFOS
Function: Resets one or both internal FIFOs for the referenced channel.
Input: BA19_FIFO_SEL enumeration type See structure definition in DDBA19Chan.h
Output: None
Notes: Resets Target, Master, Both (Master and Target) , All FIFO's .

IOCTL_BA19_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_BA19_CHAN_ENABLE_INTERRUPT

Function: Enables the channel 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 when a user interrupt is serviced. Therefore this command must be run after each interrupt occurs to re-enable it.

IOCTL_BA19_CHAN_DISABLE_INTERRUPT

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

IOCTL_BA19_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 channel master interrupt is enabled. This IOCTL is used for development, to test interrupt processing. Board level master interrupt also needs to be set.

IOCTL_BA19_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 interrupts that deal with the DMA transfers do not affect this value. Masked version of channel status.

IOCTL_BA19_CHAN_SWW_TX_FIFO
Function: Writes a 32-bit data word to the transmit FIFO.
Input: FIFO word (unsigned long integer)
Output: none
Notes: Used to make single-word accesses to the transmit FIFO instead of using DMA.

IOCTL_BA19_CHAN_SWR_RX_FIFO
Function: Returns a 32-bit data word from the receive FIFO.
Input: None
Output: FIFO word (unsigned long integer)
Notes: Used to make single-word accesses to the receive FIFO instead of using DMA.

IOCTL_BA19_CHAN_SET_CONT
Function: write to Channel Control register using structure
Input: BA19_CHAN_CONT
Output: None
Notes: See DDBA19Chan.h for structure. See below for quick reference.

IOCTL_BA19_CHAN_GET_CONT
Function: Read from Channel Control register using structure
Input: None
Output: BA19_CHAN_CONT
Notes: See DDBA19Chan.h for structure. See below for quick reference.

```
FifoTestEn;   // BiPass Mode Control
MIntEn;       // Master Interrupt Enable
WrDmaEn;      // Write DMA Interrupt Enable
RdDmaEn;      // Read DMA Interrupt Enable
```

IOCTL_BA19_CHAN_SET_MASTER_REG
Function: write to Channel Master Control register using structure
Input: BA19_CHAN_MASTER_CONTROL
Output: None
Notes: See DDBA19Chan.h for structure.

IOCTL_BA19_CHAN_GET_MASTER_REG
Function: Read from Channel Master Control register using structure
Input: None
Output: BA19_CHAN_MASTER_CONTROL
Notes: See DDBA19Chan.h for structure.

Quick Reference:
MStart; // start Master state machine
MEndian; // Set to reverse byte order
MLvlIntEn; // Set: enable interrupt Clr: disable

IOCTL_BA19_CHAN_SET_MASTER_COUNT
Function: write to Channel Master Count register
Input: ULONG
Output: None
Notes: Set the count for the expect size of a data block to be received

IOCTL_BA19_CHAN_GET_MASTER_COUNT
Function: Read from Channel Master Count Register
Input: None
Output: ULONG
Notes:

IOCTL_BA19_CHAN_SET_TARGET_REG
Function: write to Channel Target Control register using structure
Input: BA19_CHAN_TARGET_CONTROL
Output: None
Notes: See DDBA19Chan.h for structure.

IOCTL_BA19_CHAN_GET_TARGET_REG
Function: Read from Channel Target Control register using structure
Input: None
Output: BA19_CHAN_MASTER_CONTROL
Notes: See DDBA19Chan.h for structure.

Quick Reference:
TStart; // start Master state machine
TEndian; // Set to reverse byte order
TLvlIntEn; // Set: enable interrupt Clr: disable

IOCTL_BA19_CHAN_SET_MASTER_COUNT
Function: write to Channel Target Count register
Input: ULONG
Output: None
Notes: Set the count for the size of a data block to be transmitted

IOCTL_BA19_CHAN_GET_MASTER_COUNT
Function: Read from Channel Target Count register
Input: None
Output: ULONG
Notes:

Write

DMA data is written to the referenced I/O channel device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the I/O channel 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

DMA data is read from the referenced I/O channel device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the I/O channel 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

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 Street, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax

support@dyneng.com

Appendix

Reference copy of structures for evaluation

The following structures shown are available in the DDORBChan.h and DDBA19Base.h files included with the driver. The structures are included here for your evaluation when considering the driver package. The electronic versions included with the driver should be used with your project. The names track the register bit definitions. For details about particular signals please refer to the HW manual.

Base:

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

 // Driver/Device information
typedef struct _BA19_BASE_DRIVER_DEVICE_INFO
{
   UCHAR    DriverVersion;
   UCHAR    XilinxVersion;
   UCHAR    XilinxDesign;
   UCHAR    PllDeviceId;
   UCHAR    SwitchValue;
   ULONG    InstanceNumber;
} BA19_BASE_DRIVER_DEVICE_INFO, *PBA19_BASE_DRIVER_DEVICE_INFO;

typedef struct _BA19_BASE_PLL_DATA
{
   UCHAR    Data[PLL_MESSAGE_SIZE];
} BA19_BASE_PLL_DATA, *PBA19_BASE_PLL_DATA;
```

Channel:

```
typedef struct _BA19_CHAN_DRIVER_DEVICE_INFO
{
  UCHAR    DriverVersion;
  ULONG    InstanceNumber;
} BA19_CHAN_DRIVER_DEVICE_INFO, *PBA19_CHAN_DRIVER_DEVICE_INFO;

typedef enum _BA19_CHAN_FIFO_SEL {BA19_MAS, BA19_TAR, BA19_BOTH}
BA19_CHAN_FIFO_SEL, *PBA19_CHAN_FIFO_SEL;

typedef struct _BA19_CHAN_FIFO_LEVELS
{
  USHORT   AlmostFull;              // Set to control Master HW with Almost full definition
  USHORT   AlmostFullIntLevel;      // set for almost full definition to use with Level Interrupt
  USHORT   AlmostEmpty;             // set to control Target HW with Almost Empty definition,
Also controls Interrupt request
} BA19_CHAN_FIFO_LEVELS, *PBA19_CHAN_FIFO_LEVELS;

typedef struct _BA19_CHAN_FIFO_COUNTS
{
  USHORT   RxCountwPipe;
  USHORT   TxCount;
} BA19_CHAN_FIFO_COUNTS, *PBA19_CHAN_FIFO_COUNTS;

typedef struct _BA19_CHAN_CONT
{
  BOOLEAN                           FifoTestEn;// BiPass Mode Control
  BOOLEAN                           MIntEn;    // Master Interrupt Enable
  BOOLEAN                           WrDmaEn;   // Write DMA Interrupt Enable
  BOOLEAN                           RdDmaEn;   // Read DMA Interrupt Enable
} BA19_CHAN_CONT, *PBA19_CHAN_CONT;
```

```
typedef struct _BA19_CHAN_MASTER_CONTROL
{
  BOOLEAN                         MStart;// start Master state machine
  BOOLEAN                         MEndian;      // Set to reverse byte order
  BOOLEAN                         MLvlIntEn;    // Set: enable interrupt Clr: disable  Interrupt
when Master FIFO AlmostFull flag is true, requires MIntEn for system to see
} BA19_CHAN_MASTER_CONTROL, *PBA19_CHAN_MASTER_CONTROL;

typedef struct _BA19_CHAN_TARGET_CONTROL
{
  BOOLEAN                         TStart; // start Target state machine
  BOOLEAN                         TEndian;      // Set to reverse byte order
  BOOLEAN                         TLvlIntEn;    // Set: enable interrupt Clr: disable  Interrupt
when Target FIFO AlmostMt flag is true, requires MIntEn for system to see
} BA19_CHAN_TARGET_CONTROL, *PBA19_CHAN_TARGET_CONTROL;
```