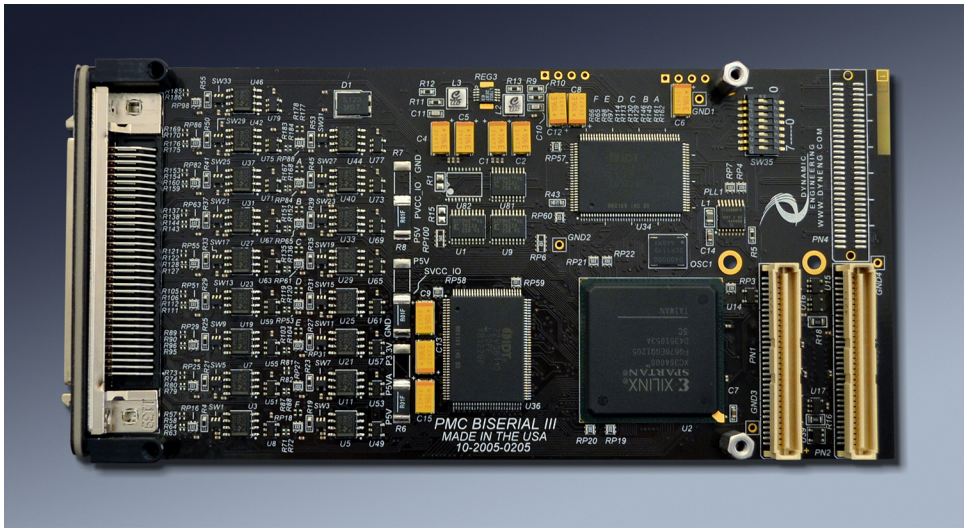
DYNAMIC ENGINEERING
150 DuBois St., Suite C Santa Cruz, CA 95060
(831) 457-8891 **Fax** (831) 457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

User Manual

PMC-BiSerial UART Hardware Manual

8-Channel UART Interface

Manual Revision A



Corresponding Hardware: 10-2005-0205

PMC-Biserial-UART

8-Channel UART Interface

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

Manual Revised 3/9/2015

Table of Contents

List of Figures

Product Description

PMC-BISERIAL-UART is part of the Dynamic Engineering family of modular I/O. PMC-BISERIAL-UART is a PMC with options for bezel and rear IO, up to 2 MHz signaling, multiple modes of operation, 1K byte of storage per Tx or Rx node. Currently 8 channels per PMC are provided.

PMC-BISERIAL-UART uses a 10 mm inter-board spacing for the front panel, standoffs, and PMC connectors. The 10 mm height is the "standard" height and will work in most systems with most carriers. If your carrier has non-standard connectors (height) to mate with PMC-BISERIAL-UART, please let us know. We may be able to do a special build with a different height connector to compensate.

Feature Table:
1. 255x32 FIFO's for Rx and Tx data storage per channel
2. 255x16 FIFO's for Packet definitions Rx and Tx
3. 3 operating modes, 32 bit packed, 8 bit unpacked, and packetized
4. 8 position Switch
5. VxWorks driver and reference software. Linux and Windows by request.
6. Industrial temperature components [-40 ⇔ +85C]
7. Standard baud rates and non-standard baud rates programmable based on a 32 MHz reference. 2M Tx and Rx max rate.

The following diagram shows the PMC-BISERIAL-UART configuration:

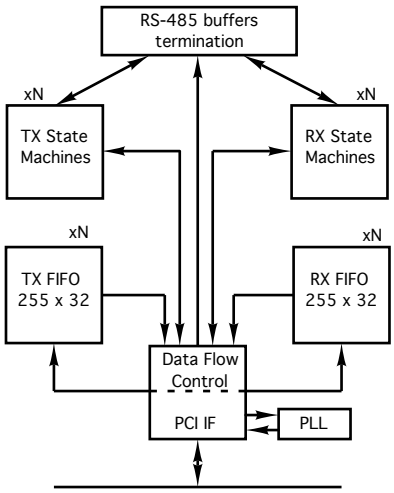


FIGURE 1 PMC-BISERIAL-UART BLOCK DIAGRAM

Please note: The Packet FIFO's provide an additional 256 x 16 per channel per direction [2xN] to store packet sizes for transmission or definitions from reception.

If you can use the BiSerial hardware but need an alternate protocol please contact Dynamic Engineering. We will redesign the state machines and create a custom interface protocol. That protocol will then be offered as a "standard" special order product. Please see our web page for current protocols offered. Please contact Dynamic Engineering with your custom application.

The UART protocol implemented provides RS422 data inputs and outputs. The transceivers have supporting programmable terminations to allow for in cable and on-board termination situations. The receivers are open cable safe – marking state is detected when undriven.

Baud rates are programmed for each transmitter and receiver separately. The design uses a distributed enable concept to allow all channels to be referenced to the master 32 MHz clock and be programmed to unique counts.

The transmitter has a pulse generator that puts out 1 clock period per programmed count. The state-machine is referenced to the master clock and sequences when the pulsed enable is present. This allows all transmit UART's to use the same reference clock and results in much better timing within the FPGA with limited clock resources.

Rx data is asynchronous and potentially noisy. Rx data is synchronized and filtered

with the master reference clock before being presented to the UART decoder. Within the UART, data is sampled and checked for being in the marking state before looking for the first start bit.

Transitions are detected and used to update the reference count. When transitions are not detected; the reference count and programmed baud rate [expected count] are used to determine when to capture bits. The receiver uses the programmed count to determine when to sample the data received. The transition detections are filtered to only be applicable within 1/8th of the expected transition. The receiver can handle quite a lot of jitter in this manner. Depending on the data [number of transitions] up to +/- 1/8th bit period per bit cell (with a transition).

Each PMC-BISERIAL-UART channel is supported by two 255 by 32-bit FIFO's. The TX FIFO supports long-word writes, and the RX FIFO supports long-word reads. A FIFO test bit in each channel control register enables the data to be routed from the TX to the RX FIFO for loop-back testing of the FIFO's. The FIFO's are used for packed, unpacked, and packetized modes of operation.

In packed mode 32 bit data is assumed, 4 bytes per LW to transmit or receive. Bytes are sent/received 0,1,2,3 with byte 0 being the data bits 7-0 on the PCI bus. 1/4 of the reads and/or writes are needed in this mode compared to unpacked.

Unpacked mode operates more like a traditional byte wide UART. Only Byte 0 is used for each LW read / written to the FIFO's. Effectively a 255 byte FIFO for TX and RX in this mode compared to 1020 bytes possible in packed mode.

With both packed and unpacked modes, if the UART is enabled the data is sent and received on demand. As soon as there is data in the output FIFO it is transmitted. If the FIFO becomes empty the transmitter waits in the marking state until more data is ready to send. Similarly the receiver writes data as it comes in without any concept of a frame or packet.

In packetized mode the transmitter waits for the packet descriptor FIFO [255x16] to have at least one descriptor loaded. As data for the packet becomes available it is transmitted. Any number of bytes can be sent in this mode. Data is packed with the possible exception of the last LW in a packet. 1,2,3, or 4 bytes can be sent from the last LW read for a particular packet. The next packet will start on the next LW boundary. Packets can be stacked in memory and unloaded as described [just multiple times]. In addition the inter-packet timer can be utilized to add delay between consecutive packets.

The receiver uses a programmable timeout to determine the end of the packet. It is suggested to use the equivalent time to 2 characters modified as needed for the inter-character gap you expect in your system. Data being received is stored locally and built into a LW to write to the Rx FIFO. When an inter-character gap exceeds the programmed delay the accumulation stops and the data captured is written to the FIFO. In addition, data is written to the FIFO when a complete LW is available. When the end of packet is detected the packet length and packet status are written to the Rx Packet FIFO. The accumulated status is written along with the length to allow multiple packets to be stored and accurate status per packet to be available.

Interrupts can be programmed from a variety of sources. The FIFO's have counts and comparators to allow almost full and almost empty situations to cause interrupts. In addition an interrupt is available for packet transmitted, packet received, and various error conditions. All interrupts are individually maskable, and a channel master interrupt enable is provided to disable all interrupts on a channel simultaneously. The current real-time status is also available from the FIFO's making it possible to operate in a polled mode.

When using internal loop-back the Almost Full and Almost Empty counts should be set to x10 or more from the end of the FIFO.

More on byte alignment: Transmit bytes are read from byte positions 0->3 byte lane wise [7-0] first, [15-8] second, [23-16] third and [31-24] last and the bytes are transmitted in this order. For message byte-counts not divisible by four, the last long-word is read as described. Any unused bytes are considered padding with the next message starting with the next FIFO long-word. For example, with 7 bytes to send, a word of 4 bytes will be read, then the lower 3 bytes will be read and sent and the 8th byte will be dropped.

In the receive direction the action is similar. Bytes are written as long-words to the RX FIFO. The first byte received is loaded into long-word byte 0 [7-0], then byte 1 [15-8], byte 2 [23-16] and byte 3 [31-24]. Whenever a message does not have a complete long-word to load and the end-of-packet character is received, zero-padding of the unused upper-bytes will occur before the long-word is written to the FIFO.

Dynamic Engineering offers drivers and reference software for Windows®, Linux, and VxWorks. Drivers and reference SW are available AS-IS to clients of the PMC-BISERIAL-UART. Support contracts are encouraged to help with integration and enhancements. www.dyneng.com/TechnicalSupportFromDE.pdf

Theory of Operation

PMC-BISERIAL-UART provides UART's for transferring data from one point to another using the standard UART transfer protocol.

While UART's are mature devices, enhancements will necessitate updates over time. PMC-BISERIAL-UART features the ability to reprogram the FPGA storage FLASH to allow updates via software. A programming adapter is required to use this feature on this HW set.

A logic block within the Xilinx controls the PCI interface to the host CPU. PMC-BISERIAL-UART design requires one wait state for read or writes cycles to any address. The wait states refer to the number of clocks after the PCI-core decode before the "terminate with data" state is reached. Two additional clock periods account for the 1 clock delay to decode the signals from the PCI bus and to convert the terminate-with-data state into the TRDY signal.

There are multiple UART's each with separate Receiver and Transmitter. Each pair is organized into a Channel within the FPGA. Frequency of operation [Baud rate], mode of operation, Parity, Stop bits, interrupt conditions are all programmable on a channel basis.

Each channel has separate state-machines to control the Transmit and Receive operation. The Tx state-machine uses the programmed values to regulate the transfer of data from the transmit storage FIFO and transmit packet FIFO to the Tx line. The Rx state-machine uses the programmed values to regulate the transfer of data from the line to the receive storage FIFO and to store descriptors into the Rx packet FIFO.

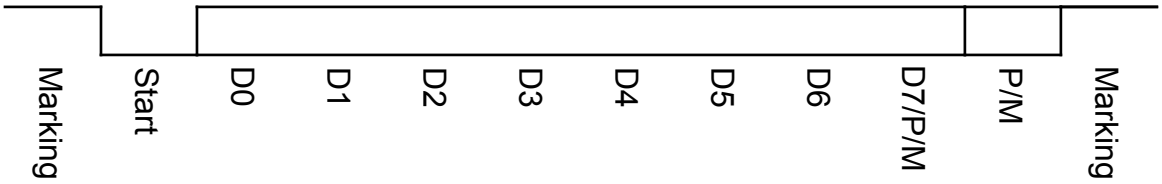


FIGURE 2 UART TRANSFER ENCODING

The Transmit state-machine will transmit a high level followed by the first falling edge of the transmission. The falling edge is the leading edge of the start bit. The start bit is 1 period wide and followed by the first data bit [LSB] of the byte being transmitted. D1-D6 follow. If the UART is programmed for 8 bit data the next period is D7. If programmed for 7 bit data the next position can be Parity if that is enabled or the marking state. The shortest transfer of a byte is 7 bit data, no parity and 1 stop bit for a total of 1[start]+7[data]+1[stop] = 9 bits. If 8 bit data is selected and parity is enabled the length becomes 1+8+1+1 = 11 bits. If 2 stop bits are selected an extra clock period is inserted between byte transfers.

The receiver does not have a clock to work with and uses over-sampling to detect the transitions and the programmed expected transfer rate to count into the bit periods to determine the bit value. The receiver also checks the expected termination values are present – for example a framing error is detected if the received signal is low when marking is expected.

Parity can be programmed to be odd, even or level. When odd the parity bit is set/cleared to make the number of 1's odd. For example if the data is "AA" an even number of bits are set in the data so the parity would be set "0 01010101 1 1" would be the string with start, data, parity and stop shown. Please note the lsb first nature of the data. The spaces are added for clarity. For even parity the reverse is true, with parity set/cleared to make the total of the data and parity fields an even count.

In addition to the framing and parity errors, FIFO over-run is flagged. When the Rx FIFO is full and a write is attempted the error is captured. A full FIFO will not accept the new write so that data is lost.

Break characters are detected by the RX state-machine and prioritized in terms of status. Status is determined Break, Frame, Parity with only one type of error or condition reported per incident. Interrupts can be generated from the occurrence.

When Break or Frame is detected the receiver resynchronizes before looking for new characters. With parity errors the error is flagged and processing continues without resynchronization.

Over reading the Rx FIFO is not an error condition. The FIFO will continue to provide the last read data multiple times. The FIFO count should be read prior to doing read multiple commands to prevent under-run.

On the Tx side an empty FIFO causes the transmitter to go to the marking state once the last word read has been transmitted. When more data is available that data will be transmitted. No under-run error is generated for this situation.

Programming

Programming PMC-BISERIAL-UART board requires only the ability to read and write data from the host. The base address is determined during system configuration of the PCI bus. The base address refers to the first user address for the slot in which the board is installed. The VendorId = 0xDCBA. The CardId = 0x0057.

In order to transfer data to another UART, several steps must be performed. First a physical connection must be established with the appropriate interface cable. Then the Channel of interest must be programmed with the appropriate UART parameters for transmit and or receive operations. Each channel has a separate register set for control bits, baud rate and other parameters. Once programmed you can load data into the Tx buffer for transmission or read from the Rx when data becomes available.

Be sure to select the correct mode of operation, and note the Rx and Tx do not need to be the same.

The hardware supports several modes of operation. Choose the right mode based on your environment. For example if you are operating with a console program and need to remain compatible with other standard UART's the 8bit [unpacked] mode will be the right choice. 3/4 of the FIFO is lost and still provides 512 bytes for both Rx and Tx.

If you need more performance and can do some adaptation the Packet or Packed modes are very useful. The Packed mode is easy to use but requires byte counts that are multiples of 4. Packet mode is the best of both with almost complete FIFO utilization and the ability to send non-LW boundary message lengths. Packet mode Packets can be stored ahead and transmitted based on the packet descriptor being written or pre-loaded and sent out as the HW becomes ready. Packet received and packet transmitted interrupts are available to help optimize operation.

The reference software has examples of using all three modes of operation.

The baud rate is programmable and should be set to a value close to the value expected. The jitter tolerance will allow slightly off frequencies to work, but will effectively have no jitter tolerance when operating in this manner. The baud rate is programmable directly based on the reference frequency allowing 1 part in 32 $x10^6$. With the RS485 IO the maximum rate tested is 2M.

Firmware Updates

Revision A1: First release. See feature table for new features.

Base Address Map

Register Name		Offset	Description
#define	BASE_REG	0x0000	//0 485 IO control, JTAG programming control
#define	BASE_GP	0x0004	//1 Switch, programming errors, Aux boards present, revision
#define	BASE_INT	0x0008	//2 Interrupt Status

FIGURE 3 PMC-BISERIAL-UART BASE ADDRESS MAP

UART Channel Address Map

Register Name		Offset	Description
#define	CHAN_CNTL	0x0000	//0 UART Channel Control Bits R/W
#define	CHAN_STAT	0x0004	//1 UART Channel Status Bits Read /write to clear
#define	CHAN_TX_UART_FIFO	0x0008	//2 UART Channel Write to TX UART FIFO
#define	CHAN_RX_UART_FIFO	0x0008	//2 UART Channel Read from RX UART FIFO
#define	CHAN_FRAME_TIME	0x000C	//3 UART Channel Programmable End of Frame Time Out R/W 24 bits
#define	CHAN_BAUD_RATE	0x0010	//4 UART Channel Programmable frequency 15-0 = Tx, 31-16 = Rx
#define	CHAN_CNTLB	0x0014	//5 Expanded UART control bits & Tx Packet delay
#define	CHAN_FIFO_LVL	0x0018	//6 UART Channel Programmable FIFO levels 31-16 = Rx AFL, 15-0 = Tx Amt
#define	CHAN_TX_PKT_FIFO	0x001C	//7 UART Channel Write to TX Packet FIFO
#define	CHAN_RX_PKT_FIFO	0x001C	//7 UART Channel Read from Packet FIFO
#define	CHAN_RX_FIFO_CNT	0x0020	//8 UART Channel Packet and Data FIFO's
#define	CHAN_TX_FIFO_CNT	0x0024	//9 UART Channel Packet and Data FIFO's

FIGURE 4 PMC-BISERIAL-UART UART CHANNEL ADDRESS MAP

There are N UART channels. Each channel has a separate set of control registers as shown in Figure 4. The offset for each of the channels up to 16 is shown in Figure 5.

Channel Offsets

```
#define   CH_0    0x0050 //20 address pointer for channel 0
#define   CH_1    0x0078 //30 address pointer for channel 1
#define   CH_2    0x00A0 //40 address pointer for channel 2
#define   CH_3    0x00C8 //50 address pointer for channel 3
#define   CH_4    0x00F0 //60 address pointer for channel 4
#define   CH_5    0x0118 //70 address pointer for channel 5
#define   CH_6    0x0140 //80 address pointer for channel 6
#define   CH_7    0x0168 //90 address pointer for channel 7
#define   CH_8    0x0190 //100 address pointer for channel 8
#define   CH_9    0x01B8 //110 address pointer for channel 9
#define   CH_10   0x01E0 //120 address pointer for channel 10
#define   CH_11   0x0208 //130 address pointer for channel 11
#define   CH_12   0x0230 //140 address pointer for channel 12
#define   CH_13   0x0258 //150 address pointer for channel 13
#define   CH_14   0x0280 //160 address pointer for channel 14
#define   CH_15   0x02A8 //170 address pointer for channel 15
```

FIGURE 5 PMC-BISERIAL-UART CHANNEL OFFSETS

The base address for PMC-BISERIAL-UART is set by the system. For Base features the base address is added to the base feature offset. For Channel features the base address is added to the Channel Offset and to the Channel Feature. Address = Base + Channel Offset+Channel Feature. All addresses are on LW boundaries and all accesses affect the entire LW. Writing a byte still affects the other three bytes.

Register Definitions

BASE_REG

Base Control Register (read/write)

Base Control Register
Unused this application

FIGURE 6 PMC-BISERIAL-UART BASE CONTROL REGISTER

All bits are active high and are reset on system power-up or reset.

BASE_GP

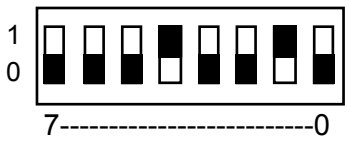
Base General Purpose Register (read/write)

Base General Purpose Register

```
#define    BASE_STAT_SW_MASK    0x000000FF // 7-0 are switch bit when installed
#define    BASE_STAT_REV_MAJ    0x0000FF00 // Design major revision
#define    BASE_STAT_XIL_TYP    0x000F0000 // Design number -- static per
                                           implementation
#define    BASE_STAT_REV_MIN    0x0FF00000 // Design minor revision
```

FIGURE 7 PMC-BISERIAL-UART BASE GP REGISTER

Switch 7-0: The user switch is read through this port. The bits are read as the lowest byte. Access the read-only port as a long word and mask off the undefined bits. The dip-switch positions are defined in the silkscreen. For example the switch figure below indicates a 0x12. The switch is an optional item. Bits have no meaning if not installed.



The Major Revision is used to track FLASH releases to the client. The revision will be updated when official releases to clients occur to allow the client to tell if a board has been updated. Currently 1.

The Minor Revision is used to track FLASH updates during development and for unofficial releases to clients. This revision may roll over depending on the number of iterations needed. Currently 1.

The Xilinx Type is the design number for a particular version of the board. A new type will be assigned for each new design implemented. In addition the CardID will also be updated. UART is type 20.

BASE_INT

Base Interrupt Status Register (read/write)

Base Interrupt Status

```
#define    BASE_INT_CH_0     0x00000001 // Set if interrupt active channel 0
#define    BASE_INT_CH_1     0x00000002 // Set if interrupt active channel 1
#define    BASE_INT_CH_2     0x00000004 // Set if interrupt active channel 2
#define    BASE_INT_CH_3     0x00000008 // Set if interrupt active channel 3
#define    BASE_INT_CH_4     0x00000010 // Set if interrupt active channel 4
#define    BASE_INT_CH_5     0x00000020 // Set if interrupt active channel 5
#define    BASE_INT_CH_6     0x00000040 // Set if interrupt active channel 6
#define    BASE_INT_CH_7     0x00000080 // Set if interrupt active channel 7
#define    BASE_INT_CH_8     0x00000100 // Set if interrupt active channel 8
#define    BASE_INT_CH_9     0x00000200 // Set if interrupt active channel 9
#define    BASE_INT_CH_10    0x00000400 // Set if interrupt active channel 10
#define    BASE_INT_CH_11    0x00000800 // Set if interrupt active channel 11
#define    BASE_INT_CH_12    0x00001000 // Set if interrupt active channel 12
#define    BASE_INT_CH_13    0x00002000 // Set if interrupt active channel 13
#define    BASE_INT_CH_14    0x00004000 // Set if interrupt active channel 14
#define    BASE_INT_CH_15    0x00008000 // Set if interrupt active channel 15
#define    BASE_INT_CH_16    0x00010000 // Set if interrupt active channel 16
#define    BASE_INT_CH_17    0x00020000 // Set if interrupt active channel 17
#define    BASE_INT_CH_18    0x00040000 // Set if interrupt active channel 18
#define    BASE_INT_CH_19    0x00080000 // Set if interrupt active channel 19
#define    BASE_INT_CH_20    0x00100000 // Set if interrupt active channel 20
#define    BASE_INT_CH_21    0x00200000 // Set if interrupt active channel 21
#define    BASE_INT_CH_22    0x00400000 // Set if interrupt active channel 22
#define    BASE_INT_CH_23    0x00800000 // Set if interrupt active channel 23
#define    BASE_INT_CH_24    0x01000000 // Set if interrupt active channel 24
#define    BASE_INT_CH_25    0x02000000 // Set if interrupt active channel 25
#define    BASE_INT_CH_26    0x04000000 // Set if interrupt active channel 26
#define    BASE_INT_CH_27    0x08000000 // Set if interrupt active channel 27
```

FIGURE 8 PMC-BISERIAL-UART BASE INTERRUPT STATUS

Each UART channel has a multitude of interrupt options. Those possible interrupts are combined into one for the channel and used to generate a board level interrupt and to provide the status in the register as shown. Clear the interrupt by servicing the source channel. Multiple interrupts can be detected in one read. Channels up to the N count are valid. For example if N = 8 , BASE_INT_CH0 ⇔ BASE_INT_CH7 are valid.

UART_CHAN_CONT

UART CHANNEL CONTROL

```
#define     ChRstA          0x0001 // set to reset channel Tx side
#define     LoopBackA       0x0002// set to loop-back FIFO data
#define     TxEnable        0x0004// set to enable Tx operation
#define     RxEnable        0x0008// set to enable Rx operation

#define     RxErrIen        0x0010// set to enable interrupt on Error
#define     TxFfAmtIen      0x0020// set to enable Transmit almost empty interrupt
#define     RxFfAfIIen      0x0040// set to enable Receiver almost full interrupt
#define     DCDFallIen      0x0080// set to enable DCD transition interrupt falling edge

#define     CTSien          0x0100// Set to enable CTS transition interrupt
#define     ForceInt        0x0200// set to force an interrupt from this channel
#define     RxOverFlowIen   0x0400// set to enable Rx Data FIFO overflow interrupt
#define     RxPckLvlIen     0x0800// set to enable Packet FIFO not empty interrupt

#define     ChRstB          0x1000 // set to reset channel Rx side
#define     TxBreak         0x2000 // set to cause Tx break – Space on TXD
#define     DCDRiseIen      0x4000 // set to enable rising edge DCD interrupt
#define     MastIntEn       0x8000// set to allow any interrupts from this channel

#define     TxParityOn      0x10000// set to use parity on Tx
#define     TxParityOdd     0x20000// set to generate odd parity when Parity is On
#define     TxStopBits      0x40000// set to transmit 2 or more stop bits
#define     TxLength        0x80000// set to transmit 8 bits cleared = 7 bit data

#define     RxParityOn      0x100000// set to use parity on Rx
#define     RxParityOdd     0x200000// set to expect odd parity when Parity is On
#define     RxStopBits      0x400000// set to expect 2 or more stop bits for framing
#define     RxLength        0x800000// set to expect 8 bits, cleared = 7 bit data

#define     TxPckEn         0x1000000// set to transmit based on a packets - if cleared
                                     data sent based on Packed or UnPacked Data
#define     TxOneByte       0x2000000// set to transmit based on 1 byte per
#define     RxPckEn         0x8000000// set to create Rx packet descriptors.  Clear to
                                     disable this function - with Packed or UnPacked Data
#define     TxParityLvl     0x10000000// Set to use level parity
#define     RxParityLvl     0x20000000// Set to use level parity
#define     RxOneByte       0x40000000// set to Receive based on 1 byte per LW
```

FIGURE 9 PMC-BISERIAL-UART UART CHAN CONTROL

ChRstA, ChRstB : When bit(s) is/are set to one, most functions within the channel are reset. Holding registers are not reset. Memories, state-machines etc. are reset. Clear for normal operation. The “A/B” indicates this signal is Or’d with the RST signal to make the channel reset based on local or global resets. A for Tx Functions, B for Rx. Software timed – leave asserted for at least one UART reference clock period – 271 ns

Loop-BackA: When this bit is set to a one, any data written to the transmit FIFO will be immediately transferred to the receive FIFO. This allows for fully testing the data FIFOs without connecting externally. When this bit is zero, normal operation is enabled. The “A” indicates HW protection to require both Tx and Rx enables to be disabled to do loop-back testing.

TxEnable when set allows the Transmit state-machine to operate. Depending on the mode other conditions will also need to be met before transmission will begin. Please note: if the channel is above 6 [7 and above] the AUX status will need to be present for the RS232 transceivers to be enabled. This bit should be set after the other pertinent parameters are programmed.

RxEnable when set allows the Receive state-machine to operate. This bit should be set after the other pertinent parameters are programmed.

pertinent parameters: Baud Rate, FIFO levels, character level controls [parity, number of bits etc.] When switching modes the enable should be disabled and then re-enabled to allow the state-machine to return to idle before resuming processing. Allow several clock periods at 271 nS per period.

RxErrIen is set to allow the error conditions of Parity, Framing, Packet FIFO overrun to cause an interrupt to the host. When cleared the status is available but the interrupt is not.

TxFfAmtIen is set to allow the Transmit FIFO Almost Empty condition to cause an interrupt. When cleared the status is available but the interrupt is not. An interrupt will be generated when the transmit FIFO level becomes equal or less than the value specified in the TX_AMT register, provided the channel master interrupt enable is asserted.

RxFfAftIen is set to allow the Receive FIFO Almost Full condition to cause an interrupt. When cleared the status is available but the interrupt is not. An interrupt will be generated when the receive FIFO level becomes equal or greater to the value specified in the RX_AFL register, provided the channel master interrupt enable is asserted.

DCDFallIen, DCDRiseIen are set to allow the Data Carrier Detect transitions to cause

an interrupt. The falling and or rising edge of DCD is used to trigger the interrupt condition. When cleared the status is available but the interrupt is not. The signal is considered to be rising when the voltage switches from a negative level to a positive value. For example -12V to +12V. Falling is the reverse. The RS232 receivers invert, a second inversion within the FPGA on this signal causes the above behavior and allows the status to properly track this signal.

CTSien is set to allow the Clear To Send transition to cause an interrupt. Rising edge of CTS is used to trigger the interrupt condition. When cleared the status is available but the interrupt is not. The signal is considered to be rising when the voltage switches from a negative level to a positive value. For example -12V to +12V. Falling is the reverse. The RS232 receivers invert, a second inversion within the FPGA on this signal causes the above behavior and allows the status to properly track this signal.

ForceInt is set to cause an interrupt to occur. Used for SW development and test purposes.

RxOverFlowIen is set to allow the Rx FIFO overflow condition to cause an interrupt. When cleared the status is available but the interrupt is not.

RxPckLvlien is set to allow the Rx Packet Received interrupt. If enabled and a Packet Descriptor is in the Packet FIFO the interrupt is set. This is a level based interrupt. Clear by reading the descriptors in the packet FIFO.

TxBreak when set forces the TXD line low which creates a “Break” condition on the transmit line – forced into the spacing state. Software timed.

MasterIntEn when set allows any of the programmable interrupt conditions to be passed to the host. When cleared no interrupts are generated by this channel.

TxParityOn when set causes the transmitted data to have parity inserted. When cleared parity is not added.

TxParityOdd when set causes odd parity when Parity is enabled and Level is not enabled. When cleared even parity is inserted if enabled.

TxStopBits when set causes the HW to add a wait state – an extra marking state between characters sent. The minimum is 1 stop bit [sent when TxStopBits is not set]. If another character is not ready when the current one is completed additional marking bits will also be inserted.

TxLength when set causes 8 bit characters [considered standard] and when cleared 7

bits per byte are transmitted. The Msb is trimmed when in the 7 bit mode.

RxParityOn when set causes the receiver to expect data with parity inserted. Parity is checked in this mode and parity errors reported. When cleared, parity is not expected and potential framing errors captured if parity is received.

RxParityOdd when set causes odd parity to be checked when Parity is enabled and not in level mode. When cleared even parity is checked if enabled.

RxStopBits when set causes the HW to expect a wait state – an extra marking state between characters sent. The minimum is 1 stop bit [sent when RxStopBits is not set]. If a start bit is received when a second stop bit is expected a framing error will result.

RxLength when set causes 8 bit characters to be expected in the data stream[considered standard] and when cleared 7 bits per byte are received. The data is LSB aligned when received in 7 bit mode. Framing errors can result if 8 bit data is received when 7 is expected and vice-versa.

TxPckEn when set, enables operation in Packet Mode [Packetized]. When cleared uses the Empty status alone to determine if transmission should occur and how much to send. See also TxOneByte.

TxOneByte when set and not in packet mode causes data to be transmitted based on using only the LS byte from the FIFO [unpacked mode – standard low speed UART operation and use with console operation]. When cleared all 4 bytes are transmitted per LW [packed mode – higher bandwidth but requires LW based data transfers – divisible by 4 data frames]

Programming note: Packetized mode is a hybrid of the packed and unpacked modes allowing for higher bandwidth operation via lower overhead for medium to larger messages. Please see the packet FIFO description for more details of using this mode.

RxPckEn when set causes the Rx state-machine to group received data into packets and to load packet descriptors into the Rx Packet FIFO. Packet lengths are automatically determined based on the programmed FrameTime. Be sure to program this time-out if in Packet Mode for Rx.

RxOneByte when set and not in Packet Mode causes the received data to be loaded one byte per LW in the Rx Data FIFO. When cleared the data is loaded 4 bytes per LW into the Rx Data FIFO.

TxParityLvl when set and parity enabled causes the inserted parity to be a level with the

ODD/EVEN control determining the level. ODD forces a ‘1’ and Even forces a ‘0’.

RxParityLvl when set and parity enabled checks the inserted parity to be a level with the ODD/EVEN control determining the level. ODD expects a ‘1’ and Even expects a ‘0’.

UART_CHAN_STAT

UART CHANNEL STATUS

```
#define     TxFfMt          0x00000001 // Transmit FIFO Empty
#define     TxFfAmt         0x00000002 // Almost Empty
#define     TxFfFl          0x00000004 // Full

#define     RxFfMt          0x00000010 // Receive FIFO Empty
#define     RxFfAfl         0x00000020 // Almost Full
#define     RxFfFl          0x00000040 // Full

#define     RxParErrLat     0x00000100 // -- status bits in each packet descriptor and
                                       latched here for non packet mode operation
#define     RxFrameErrLat   0x00000200 // -- status bits in each packet descriptor and
                                       latched here for non packet mode operation
#define     RxDataOvFlLt    0x00000400 //
#define     RxPckOvFlLt     0x00000800 //

#define     RxPckFifoMt     0x00010000 // Receive Packet FIFO Empty
#define     RxPckFifoFull   0x00020000 // Receive Packet FIFO Full
#define     TxPckFifoMt     0x00040000 // Transmit Packet FIFO Empty
#define     TxPckFifoFull   0x00080000 // Transmit Packet FIFO Full

#define     TxPckDoneLat    0x00800000 // Tx Packet Done Latched

#define     BreakStatLat    0x10000000 // -- Latched COS edge of Break Condition
#define     BreakStat       0x20000000 // -- Current Rx Break Status
#define     TxAmtLt         0x40000000 // -- Tx Almost Empty latched status
#define     RxAflLt         0x80000000 // -- Rx Almost Full latched status
```

FIGURE 10 PMC-BISERIAL-UART UART CHAN CONTROL

Transmit FIFO Empty: When a one is read, the transmit data FIFO for the corresponding channel contains no data; when a zero is read, there is at least one data-word in the FIFO.

Transmit FIFO Almost Empty: When a one is read, the number of data-words in the transmit data FIFO for the corresponding channel is less than or equal to the value written to the CHAN_FIFO_LVL register for that channel; when a zero is read, the level is more than that value.

Transmit FIFO Full: When a one is read, the transmit data FIFO for the corresponding channel is full; when a zero is read, there is room for at least one more data-word in the

FIFO.

Receive FIFO Empty: When a one is read, the receive data FIFO for the corresponding channel contains no data; when a zero is read, there is at least one data-word in the FIFO.

Receive FIFO Almost Full: When a one is read, the number of data-words in the receive data FIFO for the corresponding channel is greater or equal to the value written to the CHAN_FIFO_LVL register for that channel; when a zero is read, the level is less than that value.

Receive FIFO Full: When a one is read, the receive data FIFO for the corresponding channel is full; when a zero is read, there is room for at least one more data-word in the FIFO.

Parity Error Detected: When a one is read, it indicates that a parity error has occurred since the status was last cleared. This bit is latched and must be cleared by writing the same bit back to the channel status port. A zero indicates that no parity error has occurred. Parity can be programmed to be odd, even, level or not implemented. An error indicates the received encoding does not match the programmed encoding.

Frame Error Detected: When a one is read, it indicates that a frame error has occurred since the status was last cleared. This bit is latched and must be cleared by writing the same bit back to the channel status port. A zero indicates that no frame error has occurred. A frame error occurs when the size of the received character including packaging does not match the programmed size.

Start bit is always 1 period wide
Data is 7 or 8 periods wide
Parity is 0 or 1 period wide
Stop Bits are either 1 or 2 minimum periods wide

Leading to the minimum character of 1+7+1 = 9 bits and the max of 1+8+1+2 = 12 bits. The Hardware automatically determines the expected size based on the parameters.

RxDataOvFlLt when set the Rx Data FIFO has had an overflow condition – FIFO is full when time to write the next data word. When cleared no error has occurred. This is a latched bit and is cleared by writing back with this bit position set.

RxDataOvFlLt: when set the Rx Packet FIFO has had an overflow condition – FIFO is full when time to write the next packet descriptor. When cleared no error has occurred. This is a latched bit and is cleared by writing back with this bit position set.

RxPckFifoMt : When a one is read, the receive Packet FIFO for the corresponding channel contains no data; when a zero is read, there is at least one descriptor in the FIFO.

RxPckFifoFl: When a one is read, the receive Packet FIFO for the corresponding channel is full; when a zero is read, there is room for at least one more descriptor in the FIFO.

TxPckFifoMt : When a one is read, the transmit Packet FIFO for the corresponding channel contains no data; when a zero is read, there is at least one descriptor in the FIFO.

TxPckFifoFl: When a one is read, the transmit Packet FIFO for the corresponding channel is full; when a zero is read, there is room for at least one more descriptor in the FIFO.

BreakStat is the synchronized line level of the Rx Break Status. Reading this value returns the current state of Break Status for this channel. When set a Break is currently in effect. When '0' break is not being received. Only has meaning when receiver is enabled and has made it through synchronization.

BreakStatLat is set when a programmed edge is captured based on the Break Status. For example if the rising edge is enabled, when a Break is detected the latch is set. Similarly if the falling edge is enabled the status is set when the status transitions low meaning the break is turned off. This is a sticky bit and is cleared by writing back with the same bit position set.

TxAmtLt is set when the Transmit Data FIFO level <= the programmed Almost Empty number of words [set with CHAN_FIFO_LVL]. TxAmtLt is a sticky bit and is cleared by writing back with the bit position set.

RxAflLt is set when the Receive Data FIFO level >= the programmed Almost Full number of words [set with CHAN_FIFO_LVL]. RxAflLt is a sticky bit and is cleared by writing back with the bit position set.

TxPckDoneLat is a sticky bit set when a packet has been transmitted. Cleared by writing back to the status register with this bit set. This signal can be enabled to generate an interrupt.

CHAN_UART_FIFO

UART FIFO
#define CHAN_UART_FIFO_MASK_PACKED 0xFFFFFFFF // #define CHAN_UART_FIFO_MASK_UNPACKED 0x000000FF //

FIGURE 11 PMC-BISERIAL-UART UART FIFO

Writing to the Chan Data FIFO or UART FIFO will load data for the transmitter to utilize. Data can be written in Packed, Unpacked, or Packetized formats.

Packed data has 4 bytes per LW loaded as shown with the corresponding Mask.

UnPacked data has 1 byte per LW loaded as shown with the corresponding Mask.

Packetized is a hybrid where Packed data is used for the data format with the exception of the last word which has 1, 2, 3, or 4 bytes loaded. The Packet FIFO is used to control the number of bytes sent per packet loaded.

Packed is the most efficient data structure in terms of bytes loaded per LW used.
Packetized comes in second and as the total number of bytes in a packet increases becomes close to the efficiency of the Packed mode but with the flexibility of odd byte counts.
UnPacked is the least efficient and the most flexible.

When reading from the CHAN_UART_FIFO address the data from the Rx Data FIFO is presented. The data is packed in the same manner as described above. Packed mode provides 32 bits per LW read, UnPacked returns data in the lower byte only, and Packetized a combination of Packed and an odd length word depending on the size of the packet.

For non Packed modes the non-loaded bytes are set to zero.

CHAN_FRAME_TIME

Programmable Time Out
#define CHAN_FRAME_TIME_MASK 0x00FFFFFF //

FIGURE 12 PMC-BISERIAL-UART FRAME TIME

CHAN_FRAME_TIME is a programmable count to determine how long to wait without a new character arriving for the receiver to declare “end of packet”. The count is based on the master clock [32 MHz]. The objective is to have a time long enough to be sure all characters belonging to a packet are captured into the same packet and short enough to complete the packet in a timely fashion. If the transmitter is capable of back-to-back character transmission a 2 character period would be sufficient. If the data is not so densely packed larger delays may be desired.

CHAN_BAUD_RATE

TX & RX Frequency		
#define	CHAN_TX_BAUD_MASK	0x0000FFFF //
#define	CHAN_RX_BAUD_MASK	0xFFFF0000 //

FIGURE 13 PMC-BISERIAL-UART BAUD RATE

CHAN_BAUD_RATE is a programmable count to determine the frequency of operation. The master clock is the reference [32 MHz.]. The count programmed [N-1] determines the frequency of transmission or reception plus adjusts some of the filtering aspects of the receiver.

Rate	Recommended Setting [N-1 shown]
2M	15
1M	31
500K	63
250K	127
125K	255
62.5K	511
31.25K	1023

UART_CHAN_CONTB

UART CHANNEL CONTROL

```
#define   BreakRiseIen        0x0001 // set to enable capture of Break Detection
#define   BreakFallIen        0x0002// set to enable capture of Break removal
#define   BreakIen            0x0004// set to enable Break Interrupt
#define   TxPckDoneIen        0x0008// set to enable Tx Packet Done Interrupt

#define   DirTx               0x0010 // set to enable Tx Buffers
#define   TermRx              0x0020// set to enable Rx Termination
#define   TermTx              0x0040// set to enable Tx Termination


#define   TxPckDelayMask      0xFF00// 8 bits to define delay between TX packets
```

FIGURE 14 PMC-BISERIAL-UART UART CHANB CONTROL

Note: This is a 16 bit register [15-0]. All bits R/W. Undefined bits will return programmed value.

BreakRiseIen and BreakFallIen are used to select which edges of the Break detection status are used to generate latched status. Rising is associated with Break being asserted. Falling is associated with Break being removed.

BreakIen when set allows the captured [latched] status to generate an interrupt from the a change in state of Break. Clear the interrupt by clearing the latched status.

TxPckDoneIen when set '1' gates the Tx Packet Done latched status through to generate an interrupt. Clear the interrupt by clearing the latched status or disabling this bit.

DirTx when set enables the external and internal buffers to transmit. Normally set to '1'. When set to '0' the line level will tristate.

Note: the equivalent Rx control bit is set to receive in HW.

TermRx and TermTx when set cause the RS485 connection to have a 100 ohm resistor switched in. Analog switches are controlled to allow the parallel termination to be applied or not. Normal is Rx enabled '1' and Tx not enabled '0'. If terminations are in the cable both maybe off. Under some system conditions both may need to be enabled.

TxPckDelayMask defines the field used to determine the number of bit periods to delay between packets when transmitting in packet mode. When set to x00 no additional delay is added. When set to x01, 1 bit time is added. Please note the HW requires several bit times of marking state to start a new packet when one completes. The programmed times are in addition to this HW defined delay.

CHAN_FIFO_LVL

TX & RX FIFO Level		
#define	CHAN_TXAMT_FIFO_MASK	0x0000FFFF //
#define	CHAN_RXAFL_FIFO_MASK	0xFFFF0000 //

FIGURE 15 PMC-BISERIAL-UART FIFO LEVELS

The FIFO's are 255 deep. Unused bits should be set to zero when programming.

The TX mask is used to set the threshold for the Almost Empty condition. When the Count for the number of words in the FIFO is less than the programmed level the Almost Empty status becomes true.

The Rx mask is used to set the threshold for the Almost Full condition. When the count for the number of words in the Rx FIFO is equal or greater than the programmed level the Status is set.

For internal loop-back the Tx threshold should be set to at least 0x10 and Rx threshold set to xEF or less. The transfer engine for internal loop-back uses the almost full and almost empty status to determine if burst mode can be used. If the threshold is too small the transfer engine will not operate properly and attempt to do burst transfers when the FIFO's don't have enough room [RX or enough data TX].

CHAN_PACKET_FIFO

PACKET FIFO		
#define	CHAN_PKT_FIFO_MASK_TX	FFFF //
#define	CHAN_PKT_FIFO_MASK_RX	0FFF //

FIGURE 16 PMC-BISERIAL-UART PACKET FIFO

Writing to the Chan Packet FIFO will load a descriptor into the TX Packet FIFO. The descriptor is the number of bytes to send from the TX Data FIFO. The transmitter will wait for additional data if the Data FIFO is empty when time to read more data to complete a packet allowing packet sizes larger than the FIFO. Since the FIFO can be loaded during transmission the Almost Empty Status can be used to trigger adding more data to extend a packet. If a zero value is read the packet descriptor is ignored. 1⇔FFFF bytes.

When reading from the Channel Packet FIFO the descriptors for the data in the Rx FIFO are read plus the status for the packet. The lower bits 11-0 are the size of the data in bytes and the upper bits 15-12 are the status captured for that packet.

15 RxParErrLat
14 RxFrameErrLat
13 RxDataOvFlLt
12 RxPckOvFlLt

The definitions are found in the Channel Status register description.

Packets on the receive side are limited to the size of 1⇔FFF bytes.

Programming notes: When in Packet Mode the Channel Packet FIFO interrupt can be used to detect when new descriptors have been written to the FIFO. If larger Packets are anticipated, the AFL Data FIFO interrupt can be used to read the data in as it is received and then parsed based on the descriptor when it is ready. The MT status or count can be used if polling is preferred; to determine when the descriptor is ready.

The Frame Timer should be programmed to determine the conditions for the end of frame. If left at the default setting packets will not be properly detected resulting in non-optimal behavior.

CHAN_RX_FIFO_CNT

RX FIFO Counts		
#define	CHAN_PKT_CNT_MASK_RX	00FF0000 //
#define	CHAN_DATA_CNT_MASK_RX	000000FF //

FIGURE 17 PMC-BISERIAL-UART RX FIFO COUNTS

Reading from this port returns the Packet and Data FIFO counts. The FIFO's are 512 deep.

CHAN_TX_FIFO_CNT

RX FIFO Counts		
#define	CHAN_PKT_CNT_MASK_TX	00FF0000 //
#define	CHAN_DATA_CNT_MASK_TX	000000FF //

FIGURE 18 PMC-BISERIAL-UART TX FIFO COUNTS

Reading from this port returns the Packet and Data FIFO counts. The FIFO's are 255 deep.

LOOP-BACK & IO Connection Definitions

PMC-BISERIAL-UART can be used with direct end point cabling or with an interface. Dynamic Engineering uses HDEterm68 along with loop-back connections to accomplish loop-back.

The following table shows the connections the HDEterm68 used in the loop-back test. PMC BiSerial III has 34 Differential IO. Each UART uses 2 IO to create the TX and RX connections. The reference SW uses loop-back within the same channel as a test mechanism. IO0, IO1 form UART 1. The even IO are the TX and the Odd are the RX.

Twisted Pair: Pins shown for P1 SCSI connector and match on HDEterm68

UART1_TXP 1	UART1_RXP 2
UART1_TXN 35	UART1_RXN 36
UART2_TXP 3	UART2_RXP 4
UART2_TXN 37	UART2_RXN 38
UART3_TXP 5	UART3_RXP 6
UART3_TXN 39	UART3_RXN 40
UART4_TXP 7	UART4_RXP 8
UART4_TXN 41	UART4_RXN 42
UART5_TXP 9	UART5_RXP 10
UART5_TXN 43	UART5_RXN 44
UART6_TXP 11	UART6_RXP 12
UART6_TXN 45	UART6_RXN 46
UART7_TXP 13	UART7_RXP 14
UART7_TXN 47	UART7_RXN 48
UART8_TXP 15	UART8_RXP 16
UART8_TXN 49	UART8_RXN 50

Dynamic Engineering Drivers and Reference SW include loop-back tests using the above connections.

PMC PCI Pn1 Interface Pin Assignment

The figure below gives the pin assignments for the PMC Module PCI Pn1 Interface. See the User Manual for your carrier board for more information. Unused pins may be assigned by the specification and not needed by this design.

TCK	-12V	1	2
GND	INTA#	3	4
		5	6
BUSMODE1#	+5V	7	8
		9	10
GND		11	12
CLK	GND	13	14
GND		15	16
	+5V	17	18
	AD31	19	20
AD28	AD27	21	22
AD25	GND	23	24
GND	C/BE3#	25	26
AD22	AD21	27	28
AD19	+5V	29	30
	AD17	31	32
FRAME#	GND	33	34
GND	IRDY#	35	36
DEVSEL#	+5V	37	38
GND	LOCK#	39	40
		41	42
PAR	GND	43	44
	AD15	45	46
AD12	AD11	47	48
AD9	+5V	49	50
GND	C/BE0#	51	52
AD6	AD5	53	54
AD4	GND	55	56
	AD3	57	58
AD2	AD1	59	60
	+5V	61	62
GND		63	64

FIGURE 19 PMC-BISERIAL-UART PN1 INTERFACE

PMC PCI Pn2 Interface Pin Assignment

The figure below gives the pin assignments for the PMC Module PCI Pn2 Interface. See the User Manual for your carrier board for more information. Unused pins may be assigned by the specification and not needed by this design.

+12V		1	2
TMS	TDO	3	4
TDI	GND	5	6
GND		7	8
		9	10
	+3.3V	11	12
RST#	BUSMODE3#	13	14
+3.3V	BUSMODE4#	15	16
	GND	17	18
AD30	AD29	19	20
GND	AD26	21	22
AD24	+3.3V	23	24
IDSEL	AD23	25	26
+3.3V	AD20	27	28
AD18		29	30
AD16	C/BE2#	31	32
GND		33	34
TRDY#	+3.3V	35	36
GND	STOP#	37	38
PERR#	GND	39	40
+3.3V	SERR#	41	42
C/BE1#	GND	43	44
AD14	AD13	45	46
GND	AD10	47	48
AD8	+3.3V	49	50
AD7		51	52
+3.3V		53	54
	GND	55	56
		57	58
GND		59	60
	+3.3V	61	62
GND		63	64

FIGURE 20 PMC-BISERIAL-UART PN2 INTERFACE

Applications Guide

Interfacing

Some general interfacing guidelines are presented below. Do not hesitate to contact the factory if you need more assistance.

ESD
Proper ESD handling procedures must be followed when handling the PMC-BISERIAL-UART. The card is shipped in an anti-static, shielded bag. The card should remain in the bag until ready for use. When installing the card the installer must be properly grounded and the hardware should be on an anti-static workstation.

Start-up
Make sure that the "system" can see your hardware before trying to access it. Many BIOS will display the PCI devices found at boot up on a "splash screen" with the VendorID and CardId and an interrupt level. Look quickly, if the information is not available from the BIOS then a third party PCI device cataloging tool will be helpful.

Watch the system grounds
All electrically connected equipment should have a fail-safe common ground that is large enough to handle all current loads without affecting noise immunity. Power supplies and power consuming loads should all have their own ground wires back to a common point.

We provide the components. You provide the system. Only careful planning and practice can achieve safety and reliability. Inputs can be damaged by static discharge, or by applying voltage outside of the device rated voltages.

Construction and Reliability

Dynamic Engineering Modules are conceived and engineered for rugged industrial environments. PMC-BISERIAL-UART is constructed out of 0.062-inch thick High-Temp ROHS compliant FR4 material.

ROHS and standard processing are available options.

Through-hole and surface-mount components are used. PMC connectors are rated at 1 Amp per pin, 100 insertion cycles minimum. These connectors make consistent, correct insertion easy and reliable.

PMC's are secured against the carrier with four screws attached to the 2 stand-offs and 2 locations on the front panel. The four screws provide significant protection against shock, vibration, and incomplete insertion.

The PCB provides a (typical based on PMC) low temperature coefficient of 2.17 W/°C for uniform heat. This is based upon the temperature coefficient of the base FR4 material of 0.31 W/m-°C, and taking into account the thickness and area of the board. The coefficient means that if 2.17 Watts are applied uniformly on the component side, then the temperature difference between the component side and solder side is one degree Celsius.

PMC-BISERIAL-UART has internal thermal planes made up of heavy copper power and ground planes. The planes will spread the thermal load over the entire board to minimize hotspots and increase the "coolability". The components are Industrial temperature rated or better. Thermal vias are added under components to tie in with the thermal plane directly. Where possible devices with thermal ties were chosen to allow direct connection to the ground plane.

Thermal Considerations

The PMC-BISERIAL-UART design consists of CMOS circuits. The power dissipation due to internal circuitry is very low. It is possible to create higher power dissipation with the externally connected logic. If more than one Watt is required to be dissipated due to external loading, then forced-air cooling is recommended. With the one degree differential temperature to the solder side of the board, external cooling is easily accomplished.

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 suspected unit is at fault. Then call the Customer Service Department for a RETURN MATERIAL AUTHORIZATION (RMA) number. Carefully package the unit, in the original shipping carton if this is available, and ship prepaid and insured with the RMA number clearly written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. Dynamic Engineering will not be responsible for damages due to improper packaging of returned items. For service on Dynamic Engineering Products not purchased directly from Dynamic Engineering contact your reseller. Products returned to Dynamic Engineering for repair by other than the original customer will be treated as out-of-warranty.

Out of Warranty Repairs

Out of warranty repairs will be billed on a material and labor basis. The current minimum repair charge is $150. Customer approval will be obtained before repairing any item if the repair charges will exceed one half of the quantity one list price for that unit. Return transportation and insurance will be billed as part of the repair and is in addition to the minimum charge.

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

Specifications

Host Interface (PCI):	PCI Interface 33 MHz. 32-bit
Serial Interfaces:	8 UART channels each with Rx, Tx signals
TX Bit-rates generated:	user programmable for each UART channel with the standard baud rates up to 2M and custom programmed rates.
Software Interface:	Control Registers, FIFO's, and Status Ports
Initialization:	Hardware reset forces all registers to 0 except as noted
Access Modes:	Long-word boundary space (see memory map)
Wait States:	One for all addresses
Interrupt:	Multiple programmable interrupts per channel for flow control and error recognition.
DMA:	No DMA support.
Onboard Options:	All Options are Software Programmable
Interface Options :	Front or Rear IO. Front IO via P1 SCSI connector. Rear IO through Pn4.
Dimensions:	Standard Single PMC.
Construction:	High Temp ROHS compliant FR4 Multi-Layer Printed Circuit, Through-Hole and Surface-Mount Components
Temperature Coefficient:	2.17 W/^{o}C for uniform heat across PMC [similar for other formats]
Power	TBD

Order Information

Please refer to our PMC-BISERIAL-UART webpage for the most up to date information:
http://www.dyneng.com/pmc_biserial_III.html

PMC-BISERIAL-UART	Standard version with 8 UARTs, each with Rx, Tx RS-422 signals supported. Programmable for any baud rate 2M⇔ 150, programmable character length[7,8], stop bits[1,2], parity[odd, even, level, none]. 255x32 FIFO per Tx and Rx. Packet, Packed, and UnPacked operation supported. ROHS and non-ROHS assembly. Industrial temp components standard.
-Switch	Add the 8 position user switch to allow for multiple boards in one system or other user purposes.
-CC	Add conformal coating option. Recommended for condensing or near condensing environments
-ROHS	Leaded solder is standard on this product. Add -ROHS for ROHS processing.
HDEterm68	http://www.dyneng.com/HDEterm68.html is available as a breakout or for loop-back purposes. Available with several options including connector orientation, DIN rails, Terminal Block, header strip.
HDEcabl68	SCSI cable suitable to interconnect PMC BiSerial III and HDEterm68. Available in various lengths. Twisted shielded construction.

Glossary

Acronyms and other specialized names and their meaning:

PMC	PCI Mezzanine Card - establishes common connectors, connections, size and other mechanical features.
PCI	Peripheral Component Interconnect – parallel bus from host to this device.
VendorID	Manufacturers number for PCI/PCIe boards. DCBA is Dynamic Engineering's ID.
CardID	Unique number assigned to design to distinguish between all designs of a particular vendor.
UART	Universal Asynchronous Receiver Transmitter. Common serialized data transfer with start bit, stop bit, optional parity, optional 7/8 bit data. Can be over any electrical interface. RS232 and RS422 are most common.
Baud	Used as the bit period for this document. Not strictly correct but is the common usage when talking about UART's.
FIFO	First In First Out Memory
JTAG	Joint Test Action Group – a standard used to control serial data transfer for test and programming operations.
TAP	Test Access Port – basically a multi-state port that can be controlled with JTAG [TMS, TDI, TDO, TCK]. The TAP States are the states in the State machine controlled by the commands received over the JTAG link.

TMS	Test Mode State – this serial line provides the state switching controls. '1' indicates to move to the next state, '0' means stay put in cases where delays can happen, otherwise 0,1 are used to choose which branch to take. Due to complexity of state manipulation the instructions are usually precompiled. Rising edge of TCK valid.
TDI	Test Data In - this serial line provides the data input to the device controlled by the TMS commands. For example the data to program the FLASH comes on the TDI line while the commands to the state-machine to move through the necessary states comes over TMS. Rising edge of TCK valid.
TCK	Test Clock provides the synchronization for the TDI, TDO and TMS signals
TDO	Test Data Out is the shifted data out. Valid on the falling edge of TCK. Not all states output data.
Packet	Group of characters transferred. When the characteristics of a group of characters is known the data can be stored in packets, transferred as such and the system optimized as a result. Any number of characters can
Packed	When UART characters are always sent/received in groups of 4 allowing full use of host bus / FIFO bandwidth.
UnPacked	When UART characters are sent on an unknown basis requiring single character storage and transfer over the host bus.
MUX	Multiplexor – multiple signals multiplexed to one with a selection mechanism to control which path is active.
Flash	Non-volatile memory used on Dynamic Engineering boards to store FPGA configurations or BIOS.