DYNAMIC ENGINEERING
435 Park Dr., Ben Lomond, Calif. 95005
831-336-8891 Fax 831-336-3840
http://www.dyneng.com
sales@dyneng.com
Est. 1988

PMC-Serial-RTN5
Driver Documentation

Revision C
Corresponding Hardware: Revision C
10-2003-0303

PmcSer
NT driver for the PMC-Serial-RTN5
Serial Data Interface
PMC Module

Dynamic Engineering
435 Park Drive
Ben Lomond, CA 95005
831- 336-8891
831-336-3840 FAX

Manual Revision C. Revised July 20, 2005.

Dynamic Engineering has made every effort to ensure that this manual is accurate and complete. Still, the company reserves the right to make improvements or changes in the product described in this document at any time and without notice. Furthermore, Dynamic Engineering assumes no liability arising out of the application or use of the device described herein.

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

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.

This product has been designed to operate with PMC Module carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Introduction

The PmcSer driver is a Windows NT driver for the PMC-Serial-RTN5 board from Dynamic Engineering. This driver can control up to 10 boards in a system. Each PMC-Serial-RTN5 board has an XR16C854 Quad UART and a Z85230 Enhanced Serial Communication Controller. A separate Device Object controls each UART and SCC channel on the PMC-Serial-RTN5 board, and a separate handle references each Device Object. IO Control calls (IOCTLs) are used to configure the hardware and ReadFile() and WriteFile() calls are used to transfer data to and from the device over the PCI bus.

A handle can be opened to a specific board in Win32 by using the CreateFile() function call and passing in a Symbolic Link name. A Symbolic Link is the name of the device recognized by Windows. For the PmcSer driver, symbolic link names are formed as PmcSer*n_m* where *n* indicates the zero-based board number and *m* represents the zero-based channel number (UART channels are 0..3, SCC channel A is 4, and SCC channel B is 5) e.g. the SCC channel A on the third board is PmcSer2_4.

ReadFile() and WriteFile() are used to transfer data to/from a specific board specified by passing the appropriate handle opened via the CreateFile() function call. The amount of data transferred by either of these calls is not limited to the device FIFO size, but can be arbitrarily large.

Note

This documentation will provide information about all calls made to the driver, and how the driver interacts with the device for each of these calls. For more detailed information on the hardware implementation, refer to the PMC-Serial-RTN5 device user manual. This version of the driver always operates the SCC<->IO in half duplex mode.

Driver Installation

There are several files provided in each driver delivery. These files include PmcSer.sys, PmcSer.reg, DDPmcSer.h, PmcSerDef.h, PmcSerTest.exe, and PmcSerTest source files.

The PmcSer.sys file is the binary driver file. In order to install the driver, place this file in your Winnt\system32\drivers directory.
The PmcSer.reg file is the Windows NT registry entry file. This file contains the modifications to the Windows registry required to allow Windows to recognize the driver. In order to install the driver, double click on this file (or right click and select the Merge option in the context menu). This will merge the PmcSer entries required by the driver into the Windows NT registry. Windows must be restarted after merging this file into the registry for the driver to work.

The DDPmcSer.h file is the C header file that defines the Application Interface (API) to the driver. This file is required at compile time by any

application that wishes to interface with the PmcSer device driver. It is not needed by the driver installation.

The PmcSerTest.exe file is a sample Windows NT console application that makes calls into the PmcSer driver for board 0. It is not required during the driver installation. Open a command prompt console window and type *PmcSerTest -d0 -?* to display a list of commands (the PmcSerTest.exe file must be in the directory that the window is referencing). The commands are all of the form *PmcSerTest -dn -im* where *n* and *m* are the channel number and driver ioctl number respectively. This application is intended to test the proper functioning of the driver calls, not for normal operation.

Driver Startup

There are several tasks the PmcSer driver must do when it is started. It must scan all possible PCI buses to detect every PMC-Serial-RTN5 device in the system. It must create six Device Objects for every board it finds. It must initialize each of these Device Objects. It must register callbacks (Interrupt Service Routines and Deferred Procedure Calls) with Windows. Finally it must initialize the PMC-Serial-RTN5.

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object in the driver, which controls a single board. IOCTLs are called using the Win32 function DeviceIoControl(), and passing in the handle to the device opened with CreateFile(). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

IOCTL_PMCSER_GET_INFO

Function: Returns the current driver version, user switch value, UART device ID and revision, and Xilinx revision.
Input: None
Output: DRIVER_PMCSER_DEVICE_INFO structure
Notes: This call only accesses the hardware to read the user-switch setting. All other values are constants or are read and stored during driver start-up. See DDPmcSer.h for the definition of DRIVER_PMCSER_DEVICE_INFO.

IOCTL_PMCSER_SET_BASE_CONFIG

Function: Sets configuration parameters in the PMC-Serial-RTN5 base control register.
Input: PMCSER_BASE_CONFIG structure
Output: None
Notes: Selects the reference clock source(s) for the UART channels and the SCC, the bus timeout and SCC interrupt enable states, and other

miscellaneous controls. This call controls the routing of the SCC bi-directional signals Sync and TRxClk to either input or output drivers as follows:
SYNCOUT1: SyncA drives auxout0, SyncB drives auxout1
SYNCOUT2: SyncA drives auxout0 and IO 6+, SyncB drives auxout1 and IO 14+
SYNCIN: SyncA is driven by IO 4-, SyncB is driven by IO 12-
TRCOUT1: TRxClkA drives IO 7-, TRxClkB drives IO 15-
TRCOUT2: TRxClkA drives IO 7- and IO 7+, TRxClkB drives IO 15- and IO 15+
TRCIN232: TRxClkA is driven by IO 5-, TRxClkB is driven by IO 13-
TRCINAUX: TRxClkA is driven by auxin0, TRxClkB is driven by auxin1.
The direction of these signals to/from the SCC must be set independently using the SET_SCC_CLOCK_CONFIG and SET_SCC_SYNC_CONFIG calls. See DDPmcSer.h for the definition of PMCSER_BASE_CONFIG. See the PMC-Serial-RTN5 user manual for more information on SCC IO pinouts.

IOCTL_PMCSER_GET_BASE_CONFIG

Function: Returns the configuration of the base control register.
Input: None
Output: PMCSER_BASE_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_CHAN_CONFIG

Function: Sets configuration parameters in a PMC-Serial-RTN5 UART channel control register.
Input: PMCSER_UCHN_CONFIG structure
Output: None
Notes: Controls the the I/O driver receive termination (valid for channels A and B only) and the PCI bus to UART data interface configuration. See DDPmcSer.h for the definition of PMCSER_UCHN_CONFIG.

IOCTL_PMCSER_GET_CHAN_CONFIG

Function: Returns the configuration of a PMC-Serial-RTN5 UART channel control register.
Input: None
Output: PMCSER_UCHN_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_UART_DATA_CONFIG

Function: Sets the configuration of a UART channel data word and baud rate.
Input: UART_DATA_CONFIG structure
Output: None

Notes: Controls the baud rate, number of data bits, number of stop bits and the parity configuration for a UART channel. Accesses the UART LCR, DLL, and DLM registers. See DDPmcSer.h for the definition of UART_DATA_CONFIG. See the XR16C854 data sheet for the UART internal register descriptions.

IOCTL_PMCSER_GET_UART_DATA_CONFIG

Function: Returns the data configuration of a UART channel.
Input: None
Output: UART_DATA_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_UART_INTEN

Function: Sets the possible interrupt sources for a UART channel.
Input: UART_INT_CONFIG structure
Output: None
Notes: Selects any of seven interrupt sources for a UART channel. Accesses the UART IER register. See DDPmcSer.h for the definition of UART_INT_CONFIG. See the XR16C854 data sheet for the UART interrupt enable register description.

IOCTL_PMCSER_GET_UART_INTEN

Function: Returns the interrupt enable configuration of a UART channel.
Input: None
Output: UART_INT_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_UART_MODEM_CONTROL

Function: Sets the modem control signals and internal loop-back enable for a UART channel.
Input: UART_MODEM_CONTROL structure
Output: None
Notes: Controls the state of the modem control signals (RTS, DTR) and internal loop-back signals (OP1, OP2) for a UART channel. Accesses the UART MCR register. See DDPmcSer.h for the definition of UART_MODEM_CONTROL. See the XR16C854 data sheet for the UART modem control register description.

IOCTL_PMCSER_GET_UART_MODEM_CONTROL

Function: Returns the modem control signals for a UART channel.
Input: None
Output: UART_MODEM_CONTROL structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_UART_FLOW_CONTROL_PARAMS

Function: Sets the flow control parameters for a UART channel.
Input: UART_FLOW_PARAMS structure
Output: None
Notes: Sets the Rx and Tx FIFO trigger levels, Rx hysteresis value, and the Xon and Xoff character values. Accesses the UART FCTR, EMSR, TRG, XON1, XON2, XOFF1, and XOFF2 registers. See DDPmcSer.h for the definition of UART_FLOW_PARAMS. See the XR16C854 data sheet for the UART internal register descriptions. The EMSR and TRG registers are write only, so there is no corresponding GET_UART_FLOW_CONTROL_PARAMS for this call.

IOCTL_PMCSER_SET_UART_FLOW_CONTROL_MODE

Function: Sets the flow control mode for a UART channel.
Input: UART_FLOW_CONFIG structure
Output: None
Notes: Controls whether hardware, software, or no flow control is used for a UART channel, and further details of the selected mode. Accesses the UART EFR register. See DDPmcSer.h for the definition of UART_FLOW_CONFIG. See the XR16C854 data sheet for the UART enhanced function register description.

IOCTL_PMCSER_GET_UART_FLOW_CONTROL_MODE

Function: Returns the flow control mode for a UART channel.
Input: None
Output: UART_FLOW_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_TIMEOUT_CONFIG

Function: Sets the bus timeout count and data value.
Input: PMCSER_TIMEOUT_CONFIG structure
Output: None
Notes: Sets the timeout count, the number of PCI clocks that the bus interface will wait before signaling a timeout interrupt and returning the data specified in the timeout data field. This will only occur when pre-read data is accessed and there is insufficient data stored to satisfy the request. See DDPmcSer.h for the definition of PMCSER_TIMEOUT_CONFIG.

IOCTL_PMCSER_GET_TIMEOUT_CONFIG

Function: Returns the bus timeout count and data values.
Input: None
Output: PMCSER_TIMEOUT_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_RESET_UART

Function: Resets the UART device.
Input: None
Output: None
Notes: Resets the UART and restores the data word configuration, interrupt enables, and FIFO enables that were in effect before the reset. Other values are returned to the default conditions.

IOCTL_PMCSER_CONFIGURE_UART_FIFOS

Function: Enables and/or resets rx, tx or both of the UART FIFOs.
Input: UART_FIFO_CONTROL enumeration type
Output: None
Notes: Controls whether the UART FIFOs are enabled or disabled. If the FIFOs are enabled either the transmit, or receive FIFOs can be reset. See DDPmcSer.h for the definition of UART_FIFO_ CONTROL. An Rx FIFO reset will not delete any data pre-read from the Rx FIFO.

IOCTL_PMCSER_GET_UART_STATUS

Function: Reads various status values for a UART channel.
Input: None
Output: UART_STATUS structure
Notes: Reads and returns the value of the UART interrupt status register, line status register, and modem status register as well as the Rx and Tx FIFO data counts. See DDPmcSer.h for the definition of UART_STATUS.

IOCTL_PMCSER_GET_STATUS

Function: Returns the status bits in the INT_STAT register.
Input: None
Output: Unsigned long integer
Notes: Reads and returns the value of the INT_STAT register which indicates the state of the various interrupt sources. This call also clears the latched UART and SCC interrupt bits as well as the latched timer interrupt bit. See the bit definitions in the PmcSerDef.h header file for more information.

IOCTL_PMCSER_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to Event object
Output: None
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when an interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. In order to un-register the event, set the event handle to NULL while making this call.

IOCTL_PMCSER_ENABLE_INTERRUPT

Function: Sets the interrupt enable to true for the channel referenced.
Input: None
Output: None
Notes: For a UART channel, this call sets the UART channel control interrupt enable, leaving all other bit values in the channel control register the same. For an SCC channel, this call writes to the IEN register for the channel referenced. This IOCTL is used when interrupt processing is initiated and in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver interrupt service routine.

IOCTL_PMCSER_DISABLE_INTERRUPT

Function: Clears the appropriate interrupt enable.
Input: None
Output: None
Notes: Clears the interrupt enable for the channel referenced. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_PMCSER_FORCE_INTERRUPT

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

IOCTL_PMCSER_GET_ISR_STATUS

Function: Returns the interrupt status and the relevant UART or SCC interrupt status register value read in the last ISR.
Input: None
Output: PMCSER_INT_STAT structure
Notes: The status contains the contents of the INT_STAT register read in the last driver interrupt service routine execution and the interrupt status register value for a UART channel or the interrupt pending register of the SCC if its interrupt was pending in the last ISR. See DDPmcSer.h for the definition of PMCSER_INT_STAT.

IOCTL_PMCSER_SET_ALT232_DATA_CONFIG

Function: Writes enables and data values for all RS-232 and TTL outputs.
Input: PMCSER_ALT232DAT_CONFIG structure
Output: None
Notes: If an enable for a particular bit is set to a one, the corresponding data value for that bit supersedes the previously assigned output signal. There are 20 RS-232 signals controlled by the low 20 bits and the next two bits control the two TTL AUX outputs.

IOCTL_PMCSER_GET_ALT232_DATA_CONFIG

Function: Returns the alternate data values and enables for all RS-232 and TTL outputs.
Input: None
Output: PMCSER_ALT232DAT_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_RS232_DATA_RDBK

Function: Reads the RS-232 and TTL input data values.
Input: None
Output: unsigned long integer
Notes: Returns the value of the TTL/RS-232 data input bus.

IOCTL_PMCSER_SET_SCC_CLOCK_CONFIG

Function: Sets the clock source and time constant for the baud-rate generator, the Tx, Rx clock sources, and the clock multiple value. Also determines the direction and source of the TRxClk signal.
Input: SCC_CLOCK_CONFIG structure
Output: None
Notes: The baud rate is determined by the following formula:
Osc freq/(2 * clock multiple * (time constant + 2)). See DDPmcSer.h for the definition of SCC_CLOCK_CONFIG.

IOCTL_PMCSER_GET_SCC_CLOCK_CONFIG

Function: Returns the SCC channel's Tx and Rx clock source, rate, etc.
Input: None
Output: SCC_CLOCK_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_SCC_DATA_CONFIG

Function: Sets the SCC channel's Tx and Rx data word size, parity, and encoding.
Input: SCC_DATA_CONFIG structure
Output: None
Notes: The Rx data size can be 5, 6, 7, or 8 bits. Transmit data sizes less than five bits are possible, but require that the data be pre-formatted before being written to the transmit data buffer. See the SCC user manual for the details of this process. The number of stop bits can be 1, 1.5, 2, or 0. If zero stop bits are selected, this enables synchronous mode. If the 1x clock multiplier is selected, the 1.5 stop bit selection is not allowed. See DDPmcSer.h for the definition of SCC_DATA_CONFIG.

IOCTL_PMCSER_GET_SCC_DATA_CONFIG

Function: Returns the SCC channel's Tx and Rx data word size, parity, and encoding.
Input: None
Output: SCC_DATA_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SET_SCC_SYNC_CONFIG

Function: Sets the SCC channel's CRC parameters sync patterns, and sync type.
Input: SCC_SYNC_CONFIG structure
Output: None
Notes: In mono-sync mode, Sync0 contains the transmit sync and Sync1 contains the receive sync. In bi-sync mode, the sync character is contained in both fields with the lower bits in Sync0. In all cases the values are right justified. In SDLC mode, the SDLC flag is loaded automatically and Sync0 contains the secondary address field to compare against the address field of the SDLC frame. This process is modified if SyncNoLd is true in the SCC_RX_CONFIG, in this case only the upper four address bits are compared, so the receiver will respond to a range of 16 addresses. If external sync mode is selected, the direction of the sync signal is automatically changed to an input. The base control register must be configured accordingly. See DDPmcSer.h for the definition of SCC_SYNC_CONFIG.

IOCTL_PMCSER_GET_SCC_SYNC_CONFIG

Function: Returns the SCC channel's CRC parameters and sync type.
Input: None
Output: SCC_SYNC_CONFIG structure
Notes: Returns the values set in the previous call except the sync pattern values.

IOCTL_PMCSER_SET_SCC_INT_CONFIG

Function: Sets the SCC channel's interrupt configuration.
Input: SCC_INT_CONFIG structure
Output: None
Notes: The interrupts of the SCC are divided into three groups. The receive interrupt group consists of the receive character available and special condition interrupts. The special conditions include overrun, framing error, end-of-frame (SDLC), and (if enabled) parity error. The transmit buffer empty is the only transmitter interrupt. Finally the external interrupts are Tx underrun, break/abort, sync/hunt, CTS, DCD, and baud-rate-generator count-down to zero. See the SCC user manual for more information on the interrupt behavior and see DDPmcSer.h for the definition of SCC_INT_CONFIG.

IOCTL_PMCSER_GET_SCC_INT_CONFIG

Function: Returns the SCC channel's interrupt configuration.
Input: None
Output: SCC_INT_CONFIG structure
Notes: Returns the values set in the previous call.

IOCTL_PMCSER_SCC_RESETS

Function: Resets the entire SCC or only the referenced channel.
Input: SCC_RST_SEL enumeration type
Output: None
Notes: After the reset, the configuration values for the affected channel(s) are restored except the transmitter and receiver are disabled. See DDPmcSer.h for the definition of SCC_RST_SEL.

IOCTL_PMCSER_SCC_MISC_CMD

Function: Issues the SCC channels DPLL, CRC, and latch-reset commands. Also controls internal loop-back, auto-echo, and the SCLC status FIFO enable.
Input: SCC_MISC_CMD structure
Output: None
Notes: See the SCC user manual for information on the various commands issued. See DDPmcSer.h for the definition of SCC_MISC_CMD.

IOCTL_PMCSER_INIT_SCC_RX

Function: Initializes the SCC channel's receiver in a particular mode.
Input: SCC_RX_CONFIG structure
Output: None
Notes: See the SCC user manual for information on the various receive modes. See DDPmcSer.h for the definition of SCC_RX_CONFIG.

IOCTL_PMCSER_INIT_SCC_TX

Function: Initializes the SCC channel's transmitter in a particular mode.
Input: SCC_TX_CONFIG structure
Output: None
Notes: See the SCC user manual for information on the various transmit modes and features. See DDPmcSer.h for the definition of SCC_TX_CONFIG.

IOCTL_PMCSER_SCC_RX_EN

Function: Start or stop the SCC channel's receiver.
Input: enable (BOOLEAN type)
Output: None
Notes: When enable is set to true, the referenced receive channel is started, when enable is false the receiver is stopped.

IOCTL_PMCSER_SCC_TX_EN

Function: Start or stop an SCC channel's transmitter.
Input: enable (BOOLEAN type)
Output: None
Notes: When enable is set to true, the referenced transmit channel is started, when enable is false the transmitter is stopped.

IOCTL_PMCSER_GET_SCC_TREXT_STATUS

Function: Returns the SCC channel's Tx/Rx buffer and external status
Input: None
Output: SCC_TREXT_STAT structure
Notes: See DDPmcSer.h for the definition of SCC_TREXT_STAT.

IOCTL_PMCSER_GET_SCC_SPEC_STATUS

Function: Returns the SCC channel's special conditions status
Input: None
Output: SCC_SPEC_STAT structure
Notes: The ResCode field contains the SDLC residue code. See the SCC user manual for more information on interpreting this value. See DDPmcSer.h for the definition of SCC_SPEC_STAT.

IOCTL_PMCSER_GET_SCC_SDLC_STATUS

Function: Returns the SCC channel's SDLC status FIFO data and other SDLC staus.
Input: None
Output: SCC_SDLC_STAT structure
Notes: See DDPmcSer.h for the definition of SCC_SDLC_STAT.

IOCTL_PMCSER_SET_SCC_REG

Function: Writes the specified value to the SCC channel's register specified
Input: SCC_REG_CONFIG structure
Output: None
Notes: A generic register write call for the referenced SCC channel. See DDPmcSer.h for the definition of SCC_REG_CONFIG.

IOCTL_PMCSER_GET_SCC_REG

Function: Returns a register's value for the SCC channel referenced.
Input: Register offset (unsigned character)
Output: Register value (unsigned character)
Notes: A generic register read call for the referenced SCC channel.

IOCTL_PMCSER_SET_TIME_OUT

Function: Sets the I/O Timeout value.
Input: Timeout in milliseconds (unsigned long integer)
Output: None
Notes: Sets the time the driver will wait for an IO request to complete (read or write). If the value is set to zero (reset value), the wait will be infinite.

IOCTL_PMCSER_SET_EXPECTED_BAUDRATE

Function: Sets the expected baud rate for received data in order to detect when the transfer has finished.
Input: Baud rate in bits per second (unsigned long integer)
Output: None
Notes: Used to calculate the time the driver will wait for a receive interrupt while a ReadFile call is in progress. Once the timeout has expired

(provided the bufferlength requested has not been satisfied), any data left in the receive FIFO will be read and the call will return with STATUS_SUCCESS. If the value is set to zero (default), the wait will be infinite. This timeout is only used for read/receptions and will supersede the timeout value entered in the previous call.

Write

Data to be sent from the transmitter is written to the transmit FIFO using a WriteFile() call. The user supplies the device handle, a pointer to the buffer containing the data, the number of bytes to write, a pointer to a variable to store the amount of data actually transferred, and a pointer to an optional Overlapped structure for performing asynchronous IO. If the number of bytes requested exceeds the size of the buffer available, the driver will use interrupts to detect when more data can be written to the device. For a UART channel, if 16-bit or 32-bit writes are enabled, they will be used to implement this command. See Win32 help files for details the of the WriteFile() call.

Read

Received data can be read from the receive FIFO using a ReadFile() call. The user supplies the device handle, a pointer to the buffer to store the data in, the number of bytes to read, a pointer to a variable to store the amount of data actually transferred, and a pointer to an optional Overlapped structure for performing asynchronous IO. If the number of bytes requested exceeds the receive FIFO size, the driver will use interrupts to detect when more data has arrived. Timeouts can be set to terminate the call when insufficient data is received. For a UART channel, if 16-bit or 32-bit reads are enabled, they will be used to implement this command. See Win32 help files for the details of the ReadFile() 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
435 Park Dr.
Ben Lomond, CA 95005
831-336-8891
831-336-3840 fax
support@dyneng.com