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

PB3Hw2

Driver Documentation

Win32 Driver Model

Revision B
Corresponding Hardware: Revision E
10-2005-0505
Corresponding Firmware: Revision G

PB3Hw2 WDM driver for the
PMC-BiSerial-III-HW2
Bi-Directional Serial Data Interface
PMC Module

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

Manual Revision B. Revised May 30, 2014

Table of Contents

Introduction

The PB3Hw2 driver is a Win32 driver model (WDM) device driver for the PMC-BiSerial-III-HW2 from Dynamic Engineering. The PMC-BiSerial-III-HW2 board has a Spartan3-1500 Xilinx FPGA to implement the PCI interface, Dual-Port RAM and protocol control and status for up to 32 channels. There is also a programmable PLL with two clock outputs that are used as the clock reference for the SDLC and asynchronous interfaces. Each channel has one or more 2k-byte dual-port RAM for data transmission and reception.

When the PMC-BiSerial-III-HW2 is recognized by the PCI bus configuration utility it will start the PB3Hw2 driver to allow communication with the device. 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 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-BiSerial-III-HW2 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include PB3Hw2.sys, PB3Hw2.inf, DDPB3Hw2.h, PB3Hw2GUID.h, Hw2Test.exe, and Hw2Test source files.

DDPB3Hw2.h is a C header file that defines the Application Program Interface (API) to the driver. PB3Hw2GUID.h is a C header files that defines the device interface identifier for the PB3Hw2 driver. These files are required at compile time by any application that wishes to interface with the driver, but they are not needed for driver installation.

The PB3Hw2Test.exe file is a sample Win32 console application that makes calls to the PB3Hw2 driver to test each driver call without actually writing any application code. It is not required during the driver installation.

To run PB3Hw2Test.exe, open a command prompt console window and type a command. Type ***PB3Hw2Test -d0 -?*** to display a list of commands (the PB3Hw2Test.exe file must be in the directory that the window is referencing). The commands are all of the form ***PB3Hw2Test -dn -im*** where ***n*** and ***m*** are the device number and PB3Hw2 driver ioctl number respectively. This application is intended to test the proper functioning of the driver calls, not for normal operation.

Windows 2000 Installation

Copy PB3Hw2.inf and PB3Hw2.sys to a floppy disk, or CD if preferred.

With the PMC-BiSerial-III-HW2 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 PB3Hw2.inf file.
- Select ***Next***.
- Select ***Finish*** to close the ***Found New Hardware Wizard***.

Windows XP Installation

Copy PB3Hw2.inf and PB3Hw2.sys to a floppy disk, or CD if preferred.

With the PMC-BiSerial-III-HW2 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 appropriate 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***.

Driver Startup

Once the driver has been installed it will start automatically when the system recognizes the hardware.

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

The interface to the device is identified using a globally unique identifier (GUID), which is defined in PB3Hw2GUID.h.

The *main.c* file provided with the user test software 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 the user switch setting so the calls can be associated with a particular board from a physical point of view.

IO Controls

The driver uses IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object 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.

```
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 in this driver are described below:

IOCTL_PB3_HW2_GET_INFO

Function: Returns the Driver version, PLL ID, Switch value, Xilinx version, and Instance number.
Input: None
Output: PB3_HW2_DDINFO structure
Notes: Switch value is the configuration of the onboard dipswitch that has been selected by the User (see the board silk screen for bit position and polarity). The Xilinx version is a 32-bit value; the upper 16 bits are the design number and the lower 16 bits are the revision of that design. The PLL ID is the device address of the PLL. This value, which is set at the factory, is usually 0x69 but may also be 0x6A. See the definition of PB3_HW2_DDINFO below.

```
typedef struct _PB3_HW2_DDINFO {
   UCHAR    DriverVersion;
   UCHAR    SwitchValue;
   UCHAR    PllDeviceId;
   ULONG    XilinxVersion;
   ULONG    InstanceNumber;
} PB3_HW2_DDINFO, *PPB3_HW2_DDINFO;
```

IOCTL_PB3_HW2_SET_CHANNEL_MODE

Function: Specifies the mode of operation for the eight groups of four channels.
Input: Eight element array of mode types (PB3_HW2_MODE structure)
Output: None
Notes: There are three modes of operation for the channel groups: SDLC, ASYNC or HW mode. Currently the first two channel blocks can only be set to HW mode and the remaining six blocks can be set to either SDLC or ASYNC mode. See the definition of PB3_HW2_MODE_TYPE and PB3_HW2_MODE below.

```
#define PB3_HW2_NUM_CHANNELS    32
#define PB3_HW2_CHAN_PER_BLOCK  4
#define PB3_HW2_NUM_CHANBLOCKS (PB3_HW2_NUM_CHANNELS/PB3_HW2_CHAN_PER_BLOCK)

typedef enum _PB3_HW2_MODE_TYPE {
   HW2_SDLC,
   HW2_ASYNC,
   HW2_HW
} PB3_HW2_MODE_TYPE, *PPB3_HW2_MODE_TYPE;

typedef struct _PB3_HW2_MODE {
   PB3_HW2_MODE_TYPE   ChanBlockModes[PB3_HW2_NUM_CHANBLOCKS];
} PB3_HW2_MODE, *PPB3_HW2_MODE;
```

IOCTL_PB3_HW2_GET_CHANNEL_MODE

Function: Returns the mode of operation for the eight groups of four channels.
Input: None
Output: Eight element array of mode types (PB3_HW2_MODE structure)
Notes: This call returns the values written in the previous call.

IOCTL_PB3_HW2_SET_ACTIVE_CHANNEL

Function: Specifies the channel and offset for ReadFile or WriteFile call.
Input: Channel number and offset (PB3_HW2_MEM_ACCESS structure)
Output: None
Notes: The active channel and offset setting will remain in effect until it is overwritten. See the definition of PB3_HW2_MEM_ACCESS below.

```
typedef struct _PB3_HW2_MEM_ACCESS {
   UCHAR       Channel;
   USHORT      Offset;
} PB3_HW2_MEM_ACCESS, *PPB3_HW2_MEM_ACCESS;
```

IOCTL_PB3_HW2_PUT_DATA_WORD
Function: Writes a long word to the dual-port RAM for one channel.
Input: Channel number, memory offset, and data value to write (PB3_HW2_WRITE_WORD structure)
Output: None
Notes: This call is used to write a single long word to the data memory of one channel. All the parameters are specified in this call and the stored active channel and offset values remain unchanged. See the definition of PB3_HW2_WRITE_WORD below.

```
typedef struct _PB3_HW2_WRITE_WORD {
   UCHAR       Channel;
   USHORT      Offset;
   ULONG       Data;
} PB3_HW2_WRITE_WORD, *PPB3_HW2_WRITE_WORD;
```

IOCTL_PB3_HW2_GET_DATA_WORD
Function: Returns a long word value from the dual-port RAM for one channel.
Input: Channel number and offset (PB3_HW2_MEM_ACCESS structure)
Output: Data value at memory location (unsigned long integer)
Notes: This call is used to read a single long word from the data memory of one channel. All the memory parameters are specified in this call and the stored active channel and offset values remain unchanged. See the definition of PB3_HW2_MEM_ACCESS with the SET_ACTIVE_CHANNEL call above.

IOCTL_PB3_HW2_SET_HW_CHANNEL_CONTROL
Function: Writes the configuration of an HW1 channel to its control register.
Input: Channel number and configuration parameters (PB3_HW2_HW_CNTL structure)
Output: None
Notes: See the definition of PB3_HW2_HW_CNTL below.

```
typedef struct _PB3_HW2_HW_CNTL {
   UCHAR       Channel;
   USHORT      EndOfMessage;
   BOOLEAN     Transmit;
   BOOLEAN     HighSpeed;
   BOOLEAN     BiDirectional;
   BOOLEAN     InsertIdles;
   BOOLEAN     ClearEnable;
   BOOLEAN     IntEnable;
} PB3_HW2_HW_CNTL, *PPB3_HW2_HW_CNTL;
```

IOCTL_PB3_HW2_GET_HW_CHANNEL_STATE
Function: Returns a channel's control configuration.
Input: Channel number (unsigned character)
Output: A channel's status values (PB3_HW2_HW_STATE structure)
Notes: See the definition of PB3_HW2_HW_STATE below.

```
typedef struct _PB3_HW2_HW_STATE {
   USHORT      EndAddress;
   USHORT      EndOfMessage;
   BOOLEAN     Transmit;
   BOOLEAN     HighSpeed;
   BOOLEAN     BiDirectional;
   BOOLEAN     InsertIdles;
   BOOLEAN     ClearEnable;
   BOOLEAN     IntEnable;
   BOOLEAN     ManError;
   BOOLEAN     PostAmbleError;
   BOOLEAN     CrcError;
   BOOLEAN     Ready;
} PB3_HW2_HW_STATE, *PPB3_HW2_HW_STATE;
```

IOCTL_PB3_HW2_START_CHANNELS
Function: Starts one or more channels.
Input: Channel mask (unsigned long integer)
Output: None
Notes: Each bit in the input word represents one channel to be started according to its position. Bit 0 represents channel 0, bit 1 represents channel 1, etc.

IOCTL_PB3_HW2_STOP_CHANNELS
Function: Stops one or more channels.
Input: Channel mask (unsigned long integer)
Output: None
Notes: Each bit in the input word represents one channel to be stopped according to its position. Bit 0 represents channel 0, bit 1 represents channel 1, etc.

IOCTL_PB3_HW2_CHECK_CHANNELS
Function: Returns a bit-mask of the running channels.
Input: None
Output: Channel mask (unsigned long integer)
Notes: Each bit in the output word represents one channel that is running according to its position. Bit 0 represents channel 0, bit 1 represents channel 1, etc.

IOCTL_PB3_HW2_SET_SDLC_CHANNEL_CONTROL

Function: Writes the configuration of a channel to its control register.
Input: Channel number and configuration parameters (PB3_HW2_SDLC_CNTL structure)
Output: None
Notes: See the definition of PB3_HW2_SDLC_CNTL below. The LoadTx/RxAddress fields are used when multiple frames are sent or received. When FALSE, new starting addresses aren't loaded and read/writes will follow the previous message-frame.

```
typedef struct _PB3_HW2_SDLC_CNTL {
   UCHAR       Channel;
   BOOLEAN     TxEnable;
   BOOLEAN     RxEnable;
   USHORT      TxStartAddress;
   BOOLEAN     LoadTxAddress;
   USHORT      TxEndAddress;
   USHORT      RxStartAddress;
   BOOLEAN     LoadRxAddress;
   BOOLEAN     TxClearEnable;
   BOOLEAN     TxIntEnable;
   BOOLEAN     TxDnIntEnable;
   BOOLEAN     RxIntEnable;
   BOOLEAN     AbortIntEnable;
   BOOLEAN     TxIdleFrmEnd;
   BOOLEAN     TxFlgsShrZero;
   BOOLEAN     SendAbort;
} PB3_HW2_SDLC_CNTL, *PPB3_HW2_SDLC_CNTL;
```

IOCTL_PB3_HW2_GET_SDLC_CHANNEL_STATE

Function: Returns a channel's status and configuration.
Input: Channel number (unsigned character)
Output: A channel's status values (PB3_HW2_SDLC_STATE structure)
Notes: See the definition of PB3_HW2_SDLC_STATE below.

```
typedef struct _PB3_HW2_SDLC_STATE {
   BOOLEAN     TxEnable;
   BOOLEAN     RxEnable;
   USHORT      RxEndAddress;
   BOOLEAN     TxSndngFrm;
   BOOLEAN     TxFrmDone;
   BOOLEAN     TxClearEnable;
   BOOLEAN     TxIntEnable;
   BOOLEAN     TxDnIntEnable;
   BOOLEAN     RxIntEnable;
   BOOLEAN     AbortIntEnable;
   BOOLEAN     TxFlgsShrZero;
   BOOLEAN     TxIdleFrmEnd;
   BOOLEAN     AbortReceived;
   BOOLEAN     IdleDetected;
} PB3_HW2_SDLC_STATE, *PPB3_HW2_SDLC_STATE;
```

IOCTL_PB3_HW2_SET_ASYNC_CHANNEL_CONTROL
Function: Writes the configuration of a channel to its control register.
Input: Channel number and configuration parameters (PB3_HW2_ASYNC_CNTL structure)
Output: None
Notes: See the definition of PB3_HW2_ASYNC_CNTL below.

```
typedef struct _PB3_HW2_ASYNC_CNTL {
   UCHAR       Channel;
   BOOLEAN     TxEnable;
   BOOLEAN     RxEnable;
   USHORT      TxStartAddress;
   USHORT      TxEndAddress;
   USHORT      RxStartAddress;
   BOOLEAN     PllBClkSel;
   BOOLEAN     TxClearEnable;
   BOOLEAN     TxIntEnable;
   BOOLEAN     RxIntEnable;
} PB3_HW2_ASYNC_CNTL, *PPB3_HW2_ASYNC_CNTL;
```

IOCTL_PB3_HW2_GET_ASYNC_CHANNEL_STATE
Function: Returns a channel's status and control configuration.
Input: Channel number (unsigned character)
Output: A channel's status values (PB3_HW2_ASYNC_STATE structure)
Notes: See the definition of PB3_HW2_ASYNC_STATE below.

```
typedef struct {
   BOOLEAN     TxEnable;
   BOOLEAN     RxEnable;
   BOOLEAN     TxClearEnable;
   BOOLEAN     PllBClkSel;
   BOOLEAN     TxIntEnable;
   BOOLEAN     RxIntEnable;
   USHORT      RxEndAddress;
   BOOLEAN     FrameError;
} PB3_HW2_ASYNC_STATE, *PPB3_HW2_ASYNC_STATE;
```

IOCTL_PB3_HW2_SET_DATA
Function: Sets the data values for the 34 output bits when the data register bits are selected.
Input: Data value mask (PB3_HW2_IO structure)
Output: None
Notes: The mux and direction bits must be in the proper state for these values to be driven onto the IO lines instead of the channel outputs. See the definition of PB3_HW2_IO below.

```
typedef struct _PB3_HW2_IO {
   ULONG       IoData;
   BOOLEAN     Bit32;
   BOOLEAN     Bit33;
} PB3_HW2_IO, *PPB3_HW2_IO;
```

IOCTL_PB3_HW2_GET_DATA
Function: Returns the data values for the 34 output register bits.
Input: None
Output: Data value mask (PB3_HW2_IO structure)
Notes: This call returns the values written in the previous call.

IOCTL_PB3_HW2_SET_DIR
Function: Sets the direction of the 34 output bits when the data register bits are selected.
Input: Direction value mask (PB3_HW2_IO structure)
Output: None
Notes: These direction controls are only valid when the corresponding mux bit value is zero. When the mux value is one, the corresponding channel state-machine controls the direction and value of the IO line.

IOCTL_PB3_HW2_GET_DIR
Function: Returns the direction values for the 34 output register bits.
Input: None
Output: Direction value mask (PB3_HW2_IO structure)
Notes: This call returns the values written in the previous call.

IOCTL_PB3_HW2_SET_TERM
Function: Sets the termination of the 34 IO lines.
Input: Termination value mask (PB3_HW2_IO structure)
Output: None
Notes: The terminations are switched in or out based on the values written in this call. They are independent of the mux and direction bits.

IOCTL_PB3_HW2_GET_TERM
Function: Returns the termination values for the 34 IO lines.
Input: None
Output: Termination value mask (PB3_HW2_IO structure)
Notes: This call returns the values written in the previous call.

IOCTL_PB3_HW2_SET_MUX
Function: Sets the state of the IO mux for the 34 IO lines.
Input: Mux value mask (PB3_HW2_IO structure)
Output: None
Notes: When a bit is set to one the corresponding channel state-machine controls that IO line. When a bit is set to zero the state of the IO line depends on the respective direction and data bit.

IOCTL_PB3_HW2_GET_MUX
Function: Returns the state of the IO mux for the 34 IO lines.
Input: None
Output: Mux value mask (PB3_HW2_IO structure)
Notes: This call returns the values written in the previous call.

IOCTL_PB3_HW2_READ_DATA
Function: Returns the current values of the 34 IO lines.
Input: None
Output: Data value mask (PB3_HW2_IO structure)
Notes: This call returns the real-time value of the IO lines regardless of who is driving them.

IOCTL_PB3_HW2_GET_INT_STATUS
Function: Returns the interrupt status bit mask and clears the latched bits.
Input: None
Output: Interrupt status channel mask (unsigned long integer)
Notes: This command returns the mask of the channels that have an interrupt pending. Channel bits that are read as true are then cleared by writing only those bits back to the interrupt status register thus preventing missed interrupts that occur between the read and the write of the register.

IOCTL_PB3_HW2_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(). The returned handle is the input to this IOCTL. The driver then 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.

IOCTL_PB3_HW2_ENABLE_INTERRUPT

Function: Enables the master interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to local interrupts. The master interrupt enable is disabled in the driver interrupt service routine. Therefore this command must be run after an interrupt occurs to be ready for the next interrupt.

IOCTL_PB3_HW2_DISABLE_INTERRUPT

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

IOCTL_PB3_HW2_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 master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_PB3_HW2_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 latched status bits are cleared in the driver interrupt service routine.

IOCTL_PB3_HW2_WRITE_I2O_ADDRESS

Function: Specifies the physical address that will be used to perform the I2O accesses.
Input: unsigned long integer
Output: None
Notes: When the driver initializes it allocates some non-paged pool memory and stores the physical address of that memory in the I2O address register. That memory is then used to test the functioning of the I2O interface. This call overwrites the value in the I2O address register. It is the user's responsibility to ensure that the value written is a valid physical address for the desired location.

IOCTL_PB3_HW2_SET_I2O_CONTROL

Function: Enables the I2O interface and or clears the stored interrupt status.
Input: None
Output: PB3_HW2_I2O_CNTL structure
Notes: See the definition of PB3_HW2_I2O_CNTL below.

```
typedef struct _PB3_HW2_I2O_CNTL {
   BOOLEAN     Enable;
   BOOLEAN     Clear;
} PB3_HW2_I2O_CNTL, *PPB3_HW2_I2O_CNTL;
```

IOCTL_PB3_HW2_I2O_TEST_READ

Function: Returns the value that was written to the stored I2O address.
Input: None
Output: I2O status value (unsigned long integer)
Notes: This call reads the external memory location that the I2O status word was written to and returns that value. This call is used to test the proper functioning of the I2O interface.

IOCTL_PB3_HW2_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.
Input: PB3_HW2_PLL_DATA structure
Output: None
Notes: The PB3_HW2_PLL_DATA structure has only one field: Data – an array of 40 bytes containing the data to write. See the definition of PB3_HW2_PLL_DATA below.

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

typedef struct _PB3_HW2_PLL_DATA {
   UCHAR   Data[PLL_MESSAGE_SIZE];
} PB3_HW2_PLL_DATA, *PPB3_HW2_PLL_DATA;
```

IOCTL_PB3_HW2_READ_PLL_DATA

Function: Returns the contents of the PLL's internal registers
Input: None
Output: PB3_HW2_PLL_DATA structure
Notes: The register data is output in the PB3_HW2_PLL_DATA structure in an array of 40 bytes.

Write

PMC-BiSerial-III-HW2 RAM data is written to the device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the 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

PMC-BiSerial-III-HW2 RAM data is read from the device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the 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 merchantability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

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

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing and in most cases it will be “cockpit error” rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer’s making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer’s invoicing policy.

Out of Warranty Repairs

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

For Service Contact:

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