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

PCI NECL II STE3 Base & Channels

Driver Documentation

Win32 Driver Model

Manual Revision A
Corresponding Hardware: Revision A/B
10-2011-0101/2
Corresponding Firmware:
STE3: Design 1, Revision 1

STE3Base & STE3Chan
WDM Device Drivers for the
PCI-NECL2-STE3

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

Manual Revision A Revised May 25, 2011

Table of Contents

Introduction 5
Note: 6
Driver Installation 7
Windows 2000 Installation 8
Windows XP Installation 8
Driver Startup 9
IO Controls 10
IOCTL_STE3_BASE_GET_INFO 10
IOCTL_STE3_BASE_LOAD_PLL_DATA 10
IOCTL_STE3_BASE_READ_PLL_DATA 11
IOCTL_STE3_BASE_SET_BASEREG 11
IOCTL_STE3_BASE_GET_BASEREG 11
IOCTL_STE3_BASE_GET_STATUS 11
IOCTL_STE3_BASE_SET_GPIO_DIR 11
IOCTL_STE3_BASE_GET_GPIO_DIR 11
IOCTL_STE3_BASE_SET_GPIO_DATA 12
IOCTL_STE3_BASE_GET_GPIO 12
IOCTL_STE3_BASE_GET_GPIO_DATA 12
IOCTL_STE3_CHAN_GET_INFO 13
IOCTL_STE3_CHAN_GET_STATUS 13
IOCTL_STE3_CHAN_CLR_STATUS 13
IOCTL_STE3_CHAN_SET_FIFO_LEVELS 13
IOCTL_STE3_CHAN_GET_FIFO_LEVELS 14
IOCTL_STE3_CHAN_GET_FIFO_COUNTS 14
IOCTL_STE3_CHAN_RESET_FIFOS 14
IOCTL_STE3_CHAN_REGISTER_EVENT 14
IOCTL_STE3_CHAN_ENABLE_INTERRUPT 14
IOCTL_STE3_CHAN_DISABLE_INTERRUPT 15
IOCTL_STE3_CHAN_FORCE_INTERRUPT 15
IOCTL_STE3_CHAN_GET_ISR_STATUS 15
IOCTL_STE3_CHAN_SWW_TX_FIFO 15
IOCTL_STE3_CHAN_SWR_RX_FIFO 15
IOCTL_STE3_CHAN_SET_CONTROL 16
IOCTL_STE3_CHAN_GET_CONTROL 16
IOCTL_STE3_CHAN_SET_TX 16
IOCTL_STE3_CHAN_GET_TX 16
IOCTL_STE3_CHAN_SET_TX_READY 17

IOCTL_STE3_CHAN_GET_TX_READY 17
IOCTL_STE3_CHAN_SET_TX_AMT 17
IOCTL_STE3_CHAN_GET_TX_AMT 17
IOCTL_STE3_CHAN_SET_RX 18
IOCTL_STE3_CHAN_GET_RX 18
IOCTL_STE3_CHAN_SET_RX_AFL 18
IOCTL_STE3_CHAN_GET_RX_AFL 18
IOCTL_STE3_CHAN_SET_RX_MEM_A 19
IOCTL_STE3_CHAN_GET_RX_MEM_A 19
IOCTL_STE3_CHAN_SET_TX_MEM_A 19
IOCTL_STE3_CHAN_GET_TX_MEM_A 19
IOCTL_STE3_CHAN_SET_RX_MEM_B 20
IOCTL_STE3_CHAN_GET_RX_MEM_B 20
IOCTL_STE3_CHAN_SET_TX_MEM_B 20
IOCTL_STE3_CHAN_GET_TX_MEM_B 20
IOCTL_STE3_CHAN_SET_RX_MEM_C 21
IOCTL_STE3_CHAN_GET_RX_MEM_C 21
IOCTL_STE3_CHAN_SET_TX_MEM_C 21
IOCTL_STE3_CHAN_GET_TX_MEM_C 21
IOCTL_STE3_CHAN_SET_RX_MEM_D 22
IOCTL_STE3_CHAN_GET_RX_MEM_D 22
IOCTL_STE3_CHAN_SET_TX_MEM_D 22
IOCTL_STE3_CHAN_GET_TX_MEM_D 22
IOCTL_STE3_CHAN_SET_RX_LOOP_CNT 23
IOCTL_STE3_CHAN_GET_RX_LOOP_CNT 23
IOCTL_STE3_CHAN_SET_TX_LOOP_CNT 23
IOCTL_STE3_CHAN_GET_TX_LOOP_CNT 23
IOCTL_STE3_CHAN_SET_RX_SDRAM_CMD 24
IOCTL_STE3_CHAN_GET_RX_SDRAM_CMD 24
IOCTL_STE3_CHAN_SET_TX_SDRAM_CMD 24
IOCTL_STE3_CHAN_GET_TX_SDRAM_CMD 24
Write 25
Read 25
Service Policy 27
Out of Warranty Repairs 27
For Service Contact: 27
Appendix 28
Reference copy of structures for evaluation 28
Base: 28
Channel: 29

Introduction

STE3Base and STE3Chan drivers are Win32 driver model (WDM) device drivers for PCI-NECL2-STE3 from Dynamic Engineering.

The STE3 driver package has two parts. The driver is installed into the Windows® OS, and the User Application "Userap" executable.

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

UserAp is a stand-alone code set with a simple, and powerful menu plus a series of "tests" that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering. For example most Dynamic Engineering PCI based designs support DMA. DMA is demonstrated with the memory based loop-back tests. The tests can be ported and modified to fit your requirements.

The test software can be ported to your application to provide a running start. It is recommended to port the switch and status tests to your application to get started. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system.

The hardware has features common to the board level and features that are set apart in "channels". The channels have the same offsets within the channel, and the same status and control bit locations allowing for symmetrical software in the calling routines. The driver supports the channels with a variable passed in to identify which channel is being accessed. The hardware manual defines the pinout for each channel and the bitmaps and detailed configurations for each channel. The driver handles all aspects of interacting with the channels and base features. In some cases separate structures are used for different channels with specific channel based control bits passed. For example the UART channel has specific parameters for Parity control.

We strive to make a useable product, and while we can guarantee operation we can't

foresee all concepts for client implementation. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us, [engineering@dyneng.com] and we will consider and in many cases add them.

PCI-NECL2-STE3 has a Spartan6 Xilinx FPGA to implement the PCI interface, FIFO's and protocol control and status for the IO. A subset of the IO are grouped into a channel with TX and RX capability. Transmission and Reception can be accomplished with each channel under software control. Please refer to the HW manual for a much more complete description of the HW features. Additional TTL IO are available via the GPIO port.

A basic summary of base and channel board level features:

Base	PLL, GPIO, Switch, FLASH Revision, Design Revision
Channel 0	STE3 interface w/ SDRAM
Channel 1	SDRAM only Interface

When the PciNecl2Ste3 board is recognized by the PCI bus configuration utility it will start the STE3Base driver which will create a device object for each board, initialize the hardware, create a child devices for the channel(s) and request loading of the STE3Chan driver. The STE3Chan driver will create a device object for the I/O channel(s) and perform initialization on the channel(s). IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move blocks of data in and out of the device.

Note:

This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. For more detailed information on the hardware implementation, refer to the PciNecl2Ste3 user manual (also referred to as the hardware manual).

This manual discusses the basic "in and out" of each IOCTL used with STE3. There may be "extra" IOCTL calls defined in the include file that do not pertain to STE3 operation. These IOCTL's represent currently unimplemented features in STE3.

Driver Installation

There are several files provided in each driver package. These files include driver: STE3Base.sys, PciNecl2Ste3.inf, DDSTE3Base.h, STE3BaseGUID.h, STE3Chan.sys, DDSTE3Chan.h, STE3ChanGUID.h. Userap: User Application source files.

STE3BaseGUID.h and STE3ChanGUID.h are C header files that define the device interface identifiers for the drivers. DDSTE3Base.h and DDSTE3Chan.h files are C header files that define the Application Program Interface (API) to the drivers. These files are required at compile time by any application that wishes to interface with the drivers, but they are not needed for driver installation. The files are included with the Userap fileset.

Windows 2000 Installation

Copy PciNecl2Ste3.inf, STE3Base.sys and STE3Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation. The files are stored at the root of the PciNecl2Ste3Userap file set.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Select ***Next***.
_ Select ***Search for a suitable driver for my device.***
_ Select ***Next***.
_ Insert the disk prepared above in the desired drive.
_ Select the appropriate drive e.g. ***Floppy disk drives***.
_ Select ***Next***.
_ The wizard should find the PciNecl2Ste3.inf file.
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.***
Repeat this for each channel as necessary.

Windows XP Installation

Copy PciNecl2Ste3.inf, STE3Base.sys and STE3Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

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

Driver Startup

Once the drivers have been installed they will start automatically when the system recognizes the hardware. In many systems the drivers are associated with the slot the hardware is installed into. If you move slot positions you may need to reinstall the driver,

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

The interfaces to the devices are identified using globally unique identifiers (GUIDs), which are defined in STE3BaseGUID.h and STE3ChanGUID.h.

The User Application software contains a file called “main.c”. Main has the initialization needed to get the handles to the base and channel assets of the installed PciNecl2Ste3 device.

The main file provided is designed to work with our test menu and includes user interaction steps to allow the user to select which board is being tested in a multiple board environment. The integrator can hardcode for single board systems or use an automatic loop to operate in multiple board systems without using user interaction. For multiple user systems it is suggested that the board number is associated with a switch setting so the calls can be associated with a particular board from a physical point of view. The DIPswitch is provided for this purpose.

IO Controls

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

```
BOOL DeviceIoControl(
  HANDLE        hDevice,              // Handle opened with
                                      CreateFile()
  DWORD         dwIoControlCode,      // Control code defined in
                                      API header file
  LPVOID        lpInBuffer,           // Pointer to input parameter
  DWORD         nInBufferSize,        // Size of input parameter
  LPVOID        lpOutBuffer,          // Pointer to output
                                      parameter
  DWORD         nOutBufferSize,       // Size of output parameter
  LPDWORD       lpBytesReturned,      // Pointer to return length
                                      parameter
  LPOVERLAPPED  lpOverlapped,         // Optional pointer to
                                      overlapped structure
);                                    // used for asynchronous I/O
```

The IOCTLs defined for the STE3Base driver are described below:
Please note that the address map is included in the DD file for reference when writing your own driver for a different OS.

IOCTL_STE3_BASE_GET_INFO
Function: Return the Instance Number, Switch value, PLL device ID, Xilinx rev and Current Driver Version
Input: None
Output: STE3_BASE_DRIVER_DEVICE_INFO : Structure
Notes: Switch value is the configuration of the on-board dip-switch that has been set by the User (see the board silk screen for bit position and polarity). The PLL ID is the device address of the PLL device. This value, which is set at the factory, is usually 0x69 but may also be 0x6A. See DDSTE3Base.h for the definition of STE3_BASE_DRIVER_DEVICE_INFO.

IOCTL_STE3_BASE_LOAD_PLL_DATA
Function: Loads the internal registers of the PLL.
Input: STE3_BASE_PLL_DATA structure
Output: None
Notes:

IOCTL_STE3_BASE_READ_PLL_DATA
Function: Returns the contents of the PLL's internal registers
Input: None
Output: STE3_BASE_PLL_DATA structure
Notes: The register data is output in the STE3_BASE_PLL_DATA structure in an array of 40 bytes.

IOCTL_STE3_BASE_SET_BASEREG
Function: Write to Base Control Register - general access to base control register of card, use with bit definitions
Input: ULONG
Output: none
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_STE3_BASE_GET_BASEREG
Function: Read from Base Control Register - general access from base control register of card, use with bit definitions
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_STE3_BASE_GET_STATUS
Function: Read from Status Register
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access from the register. See DDSTE3Base.h for bit map information. See the HW manual for exact definitions of bits.

IOCTL_STE3_BASE_SET_GPIO_DIR
Function: Write to GPIO Port Direction Register.
Input: ULONG
Output: none
Notes: D11-0 bits control GPIO Port bits 11-0. Set to transmit, clear to receive.

IOCTL_STE3_BASE_GET_GPIO_DIR
Function: Read-back from Parallel Port Direction Register.
Input: none
Output: ULONG
Notes:

IOCTL_STE3_BASE_SET_GPIO_DATA
Function: Write to GPIO Port Data Register.
Input: ULONG
Output: none
Notes: D13-0 bits control Parallel Port bits 11-0. If set to transmit in [DIR register] the Data register bits are output as defined in this [Data] register.

IOCTL_STE3_BASE_GET_GPIO
Function: Read-back from IO lines directly
Input: none
Output: ULONG
Notes: The IO levels may or may not match the output data since some of the lines may not be enabled to transmit or there may be external devices affecting the levels. 11-0 defined.

IOCTL_STE3_BASE_GET_GPIO_DATA
Function: Read-back from Data Definition register
Input: none
Output: ULONG
Notes: Direct read-back from the data definition register, will always match the data definition independent of direction register programming.

The IOCTLs defined for the STE3Chan driver are described below:
In the STE3 implementation both the Transmitter and the Receiver interface are implemented within the same channel (0,1,2,3). The Receiver accepts data from the external equipment. The Transmitter provides data to the external equipment. Loop-back can be accomplished with each channel looped to itself.

Address and bit map information is included in the DDSTE3Chan.h file to support those who are writing drivers for other OS.

IOCTL_STE3_CHAN_GET_INFO
Function: Return the Instance Number and Current Driver Version
Input: None
Output: STE3_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of STE3_CHAN_DRIVER_DEVICE_INFO in the DDSTE3Chan.h header file.

IOCTL_STE3_CHAN_GET_STATUS
Function: Return the value of the status register and clear latched bits
Input: None
Output: Status register value(ULONG)
Notes: Latched interrupt and error status are cleared by write-back. Defines available in DDSTE3Chan.h Detailed definitions are available in the HW manual. Several of the defines have different meanings depending on the channel being accessed. The .h file has the base definitions followed by specific definitions for the UART and Manchester interfaces [for the bits with alternate definitions].

IOCTL_STE3_CHAN_CLR_STATUS
Function: Clear Error Bits latched and not cleared by status read
Input: ULONG
Output: none
Notes: Clear latched error bits. Allows polling on FIFO status without losing potential Error conditions. Write back with same bit position set to clear. Defines available in DDSTE3Chan.h Detailed definitions are available in the HW manual.

IOCTL_STE3_CHAN_SET_FIFO_LEVELS
Function: Sets the transmitter almost empty and receiver almost full levels for the channel.
Input: STE3_CHAN_FIFO_LEVELS structure
Output: None
Notes: The FIFO counts are compared to these levels to determine the value of the STAT_TX_FF_AE and STAT_RX_FF_AF status bits.

IOCTL_STE3_CHAN_GET_FIFO_LEVELS
Function: Returns the transmitter almost empty and receiver almost full levels for the channel.
Input: None
Output: STE3_CHAN_FIFO_LEVELS structure
Notes:

IOCTL_STE3_CHAN_GET_FIFO_COUNTS
Function: Returns the number of data words in FIFO's.
Input: None
Output: STE3_CHAN_FIFO_COUNTS structure
Notes: Returns the actual TX FIFO data counts and count including DMA pipeline RX FIFO, All non-IO FIFO memory is included – SDRAM plus attached FIFO's.

IOCTL_STE3_CHAN_RESET_FIFOS
Function: Resets one or all FIFO's for the referenced channel.
Input: STE3_FIFO_SEL enumeration type See structure definition in DDSTE3Chan.h
Output: None
Notes: Resets Transmit, Receive, External or All FIFO's. Please note the Tx / Rx state machines are also reset by this command.

IOCTL_STE3_CHAN_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts **do not** cause the event to be signaled.

IOCTL_STE3_CHAN_ENABLE_INTERRUPT

Function: Enables the channel Master Interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to user interrupts. The master interrupt enable is disabled in the driver interrupt service routine when a user interrupt is serviced. Therefore this command must be run after each interrupt occurs to re-enable it.

IOCTL_STE3_CHAN_DISABLE_INTERRUPT

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

IOCTL_STE3_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the channel master interrupt is enabled. This IOCTL is used for development, to test interrupt processing. Board level master interrupt also needs to be set.

IOCTL_STE3_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (unsigned long integer)
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled channel interrupts. The interrupts that deal with the DMA transfers do not affect this value. Masked version of channel status.

IOCTL_STE3_CHAN_SWW_TX_FIFO
Function: Writes a 32-bit data word to the transmit FIFO.
Input: FIFO word (unsigned long integer)
Output: none
Notes: Used to make single-word accesses to the transmit FIFO instead of using DMA.

IOCTL_STE3_CHAN_SWR_RX_FIFO
Function: Returns a 32-bit data word from the receive FIFO.
Input: None
Output: FIFO word (unsigned long integer)
Notes: Used to make single-word accesses to the receive FIFO instead of using DMA. Please note, Data read from this port is no longer available in the FIFO for DMA or other use.

IOCTL_STE3_CHAN_SET_CONTROL
Function: write to Channel Control register using structure
Input: STE3_CHAN_CONT
Output: None
Notes: See DDSTE3Chan.h for structure. Some Channel Control functions are specific to a particular channel. The HW manual has detailed information on the function of each option and which channels are able to use that option.

IOCTL_STE3_CHAN_GET_CONTROL
Function: Read from Channel Control register using structure
Input: None
Output: STE3_CHAN_CONT
Notes: See DDSTE3Chan.h for structure.

IOCTL_STE3_CHAN_SET_TX
Function: write to Channel Tx Control register using structure
Input: STE3_CHAN_TX_CONTROL
Output: None
Notes: See DDSTE3Chan.h for structure.

IOCTL_STE3_CHAN_GET_TX
Function: Read from Channel TX Control register using structure
Input: None
Output: STE3_CHAN_TX_CONTROL
Notes: See DDSTE3Chan.h for structure.

IOCTL_STE3_CHAN_SET_TX_READY
Function: write to Channel TX Ready Count Register
Input: ULONG [max of 2044 bytes]
Output: None
Notes: Set the count for starting the Transmitter. When the transmitter is enabled the State-machine waits for the READY COUNT level matched against the total FIFO path for TX .

IOCTL_STE3_CHAN_GET_TX_READY
Function: Read from Channel TX Ready Count Register
Input: None
Output: ULONG
Notes: Read back port.

IOCTL_STE3_CHAN_SET_TX_AMT
Function: write to Channel TX Almost Empty Count Register
Input: ULONG
Output: None
Notes: Set the count for comparing against the TX FIFO path total count for the Almost Empty condition. The Almost Empty level interrupt control count is set with this register.

IOCTL_STE3_CHAN_GET_TX_AMT
Function: Read from Channel TX Almost Empty Count Register
Input: None
Output: ULONG
Notes: Read back port.

IOCTL_STE3_CHAN_SET_RX
Function: write to Channel Receiver Control register using structure
Input: STE3_CHAN_RX_CONTROL
Output: None
Notes: See DDSTE3Chan.h for structure. See HW manual for more detail on individual bit control. Valid for LLST and NMS. See separate UART and Downlink commands.

IOCTL_STE3_CHAN_GET_RX
Function: Read from Channel Receiver Control register using structure
Input: None
Output: STE3_CHAN_RX_CONTROL
Notes: See DDSTE3Chan.h for structure.

IOCTL_STE3_CHAN_SET_RX_AFL
Function: write to Channel Receiver Almost Full Register
Input: ULONG
Output: None
Notes: Set the level to compare against for the Almost Full Level interrupt. The level is compared against the total RX FIFO path data and when the FIFO path has more data than the AFL register value the AFL interrupt and status are set.

IOCTL_STE3_CHAN_GET_RX_AFL
Function: Read from Channel Receiver Almost Full Register
Input: None
Output: ULONG
Notes:

For a complete discussion of the SDRAM set-up to go along with the register IOCTL's please refer to the PCI-NECL2-STE3 hardware manual.

IOCTL_STE3_CHAN_SET_RX_MEM_A
Function: write to Channel Receiver Memory A register
Input: ULONG
Output: None
Notes: This register is used to set the initial address within SDRAM for the receiver.

IOCTL_STE3_CHAN_GET_RX_MEM_A
Function: Read from Channel Receiver Memory A register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_TX_MEM_A
Function: write to Channel Transmitter Memory A register
Input: ULONG
Output: None
Notes: This register is used to set the initial address within SDRAM for the transmitter.

IOCTL_STE3_CHAN_GET_TX_MEM_A
Function: Read from Channel Transmitter Memory A register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_RX_MEM_B
Function: write to Channel Receiver Memory B register
Input: ULONG
Output: None
Notes: This register is used to set the loop-back address within SDRAM for the receiver to use if retransmitting data using loop-control

IOCTL_STE3_CHAN_GET_RX_MEM_B
Function: Read from Channel Receiver Memory B register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_TX_MEM_B
Function: write to Channel Transmitter Memory B register
Input: ULONG
Output: None
Notes: This register is used to set the loop-back address within SDRAM for the transmitter to use if retransmitting data using loop-control.

IOCTL_STE3_CHAN_GET_TX_MEM_B
Function: Read from Channel Transmitter Memory B register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_RX_MEM_C
Function: write to Channel Receiver Memory C register
Input: ULONG
Output: None
Notes: This register is used to set the “end of the body” address within SDRAM for the receiver to use if retransmitting data using loop-control

IOCTL_STE3_CHAN_GET_RX_MEM_C
Function: Read from Channel Receiver Memory C register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_TX_MEM_C
Function: write to Channel Transmitter Memory C register
Input: ULONG
Output: None
Notes: This register is used to set the “end of body” address within SDRAM for the transmitter to use if retransmitting data using loop-control.

IOCTL_STE3_CHAN_GET_TX_MEM_C
Function: Read from Channel Transmitter Memory C register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_RX_MEM_D
Function: write to Channel Receiver Memory D register
Input: ULONG
Output: None
Notes: This register is used to set the end address within SDRAM.

IOCTL_STE3_CHAN_GET_RX_MEM_D
Function: Read from Channel Receiver Memory D register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_TX_MEM_D
Function: write to Channel Transmitter Memory D register
Input: ULONG
Output: None
Notes: This register is used to set the end address within SDRAM.

IOCTL_STE3_CHAN_GET_TX_MEM_D
Function: Read from Channel Transmitter Memory D register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_RX_LOOP_CNT
Function: write to Channel Receiver Loop Count register
Input: ULONG bits 7-0 define count
Output: None
Notes: This register is used to set the number of loops through the body to perform.

IOCTL_STE3_CHAN_GET_RX_LOOP_CNT
Function: Read from Channel Receiver Loop Count register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_TX_LOOP_CNT
Function: write to Channel Transmitter Loop Count register
Input: ULONG bits 7-0 define count
Output: None
Notes: This register is used to set the number of loops through the body to perform.

IOCTL_STE3_CHAN_GET_TX_LOOP_CNT
Function: Read from Channel Transmitter Loop Count register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_RX_SDRAM_CMD
Function: write to Channel Receiver SDRAM Command register
Input: ULONG bits 7-0 define operation
Output: None
Notes: This register is used to set the operational mode of the SDRAM for STE3 set to 0x08.

IOCTL_STE3_CHAN_GET_RX_SDRAM_CMD
Function: Read from Channel Receiver SDRAM Command register
Input: None
Output: ULONG
Notes:

IOCTL_STE3_CHAN_SET_TX_SDRAM_CMD
Function: write to Channel Transmitter SDRAM Command register
Input: ULONG bits 7-0 define operation
Output: None
Notes: This register is used to set the operational mode of the SDRAM for STE3 set to 0x08.

IOCTL_STE3_CHAN_GET_TX_SDRAM_CMD
Function: Read from Channel Transmitter SDRAM Command register
Input: None
Output: ULONG
Notes:

Write

DMA data is written to the referenced I/O channel device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually written, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Read

DMA data is read from the referenced I/O channel device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually read, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Examples of using DMA are provided in the reference software FIFO and IO loop-tests.

Warranty and Repair

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

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

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

Service Policy

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

Out of Warranty Repairs

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

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax

support@dyneng.com

Appendix

Reference copy of structures for evaluation

The following structures shown are available in the DDSTE3Chan.h and DDSTE3Base.h files included with the driver. The structures are included here for your evaluation when considering the driver package. The electronic versions included with the driver should be used with your project. The names track the register bit definitions. For details about particular signals please refer to the HW manual.

Base:

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

 // Driver/Device information
typedef struct _STE3_BASE_DRIVER_DEVICE_INFO
{
   UCHAR    DriverVersion;
   UCHAR    XilinxVersion;
   UCHAR    XilinxDesign;
   UCHAR    PllDeviceId;
   UCHAR    SwitchValue;
   ULONG    InstanceNumber;
} STE3_BASE_DRIVER_DEVICE_INFO, *PSTE3_BASE_DRIVER_DEVICE_INFO;

typedef struct _STE3_BASE_PLL_DATA
{
   UCHAR    Data[PLL_MESSAGE_SIZE];
} STE3_BASE_PLL_DATA, *PSTE3_BASE_PLL_DATA;
```

Channel:

```
typedef struct _STE3_CHAN_DRIVER_DEVICE_INFO
{
  UCHAR    DriverVersion;
  ULONG    InstanceNumber;
} STE3_CHAN_DRIVER_DEVICE_INFO, *PSTE3_CHAN_DRIVER_DEVICE_INFO;

typedef enum _STE3_CHAN_FIFO_SEL {STE3_RX, STE3_TX, STE3_EXT, STE3_ALL}
STE3_CHAN_FIFO_SEL, *PSTE3_CHAN_FIFO_SEL;

typedef struct _STE3_CHAN_FIFO_LEVELS
{
  ULONG   AlmostFull;              // Set to control Master HW with Almost full definition
  USHORT   AlmostEmpty;            // set to control Target HW with Almost Empty definition,
Also controls Interrupt request
} STE3_CHAN_FIFO_LEVELS, *PSTE3_CHAN_FIFO_LEVELS;

typedef struct _STE3_CHAN_FIFO_COUNTS
{
  ULONG              RxCountwPipe;      // RX SDRAM plus DMA FIFO and IO FIFO plus
                                        pipeline
  ULONG              TxCount;           // TX SDRAM plus DMA FIFO and IO FIFO
} STE3_CHAN_FIFO_COUNTS, *PSTE3_CHAN_FIFO_COUNTS;

typedef struct _STE3_CHAN_CONT
{
  BOOLEAN            FifoTestEn;        // BiPass Mode Control
  BOOLEAN            MIntEn;            // Master Interrupt Enable
  BOOLEAN            WrDmaEn;           // Write DMA Interrupt Enable
  BOOLEAN            RdDmaEn;           // Read DMA Interrupt Enable
  BOOLEAN            TxUrgent;          // Set for higher priority TX DMA processing
  BOOLEAN            RxUrgent;          // Set for higher priority RX DMA processing
  BOOLEAN            SdramInit;         // Set to enable Initialization of SDRAM
  BOOLEAN            ByEnChTx;          // Set to enable bypass of SDRAM in TX path
  BOOLEAN            ByEnChRx;          // Set to enable bypass of SDRAM in RX path
} STE3_CHAN_CONT, *PSTE3_CHAN_CONT;
```

```
typedef struct _STE3_CHAN_RX_CONTROL
{
  BOOLEAN          RxStart;          //0 set to begin RX Data Acquisition
  BOOLEAN          RxIntEn;          //2 set to enable RX interrupt for each image
                                       captured
  BOOLEAN          RxAFIntEn;        //3 set to enable RX DMA FIFO based interrupt
                                       [almost full]
  BOOLEAN          RxOvFlEn;         //4 set to enable RX OverFlow interrupt
  BOOLEAN          RxByteOrder;      //5 N/A STE3
  BOOLEAN          RxEnLoad;         //8 set to enable Loading SDRAM from SDRAM RX
                                       IO FIFO
  BOOLEAN          RxEnUnLoad;       //9 set to enable UnLoading SDRAM to SDRAM RX
                                       DMA FIFO
} STE3_CHAN_RX_CONTROL, *PSTE3_CHAN_RX_CONTROL;


typedef struct _STE3_CHAN_TX_CONTROL
{
  BOOLEAN          TxStart;          //0 start TX state machine
  BOOLEAN          TxIntEn;          //2 set to enable TX interrupt - occurs when each
                                       image is transmitted
  BOOLEAN          TxAEIntEn;        //3 set to enable TX FIFO based interrupt [almost
                                       empty] based on latched status (requires not
                                       almost empty to become almost mt
  BOOLEAN          TxUnFlEn;         //4 N/A STE3
  BOOLEAN          TxByteOrder;      //5 N/A STE3
  BOOLEAN          TxEnLoad;         //8 set to enable Loading SDRAM from SDRAM TX
                                       DMA FIFO
  BOOLEAN          TxEnUnLoad;       //9 set to enable UnLoading SDRAM to SDRAM TX
                                       IO FIFO
} STE3_CHAN_TX_CONTROL, *PSTE3_CHAN_TX_CONTROL;
```