DYNAMIC ENGINEERING
150 DuBois, Suite B/C
Santa Cruz, CA 95060
(831) 457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

PcieHLx5SMBBase
&
PcieHLx5SMBChan

WDF Driver Documentation
For the single port
PCIe4lHOTLinkx5-SMB

Developed with Windows Driver Foundation Ver1.9

Manual Revision 1p0 5/20/22
Corresponding Firmware: Revision 1p1
Corresponding Hardware: 10-2016-2801

PCIE4IHOTLinkx5-SMB

HOTLink Interface

Dynamic Engineering
150 DuBois, Suite B/C
Santa Cruz, CA 95060
(831) 457-8891

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.

Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Introduction

The PcieHLx5SMBBase and PcieHLx5SMBChan drivers are Windows device drivers for PCIe4lHOTLinkx5-SMB HOTLink design from Dynamic Engineering. These drivers were developed with the Windows Driver Foundation version 1.9 (WDF) from Microsoft, specifically the Kernel-Mode Driver Framework (KMDF).

The HOTLink board has a Xilinx Spartan-6-LX100 FPGA to implement a PCI interface, FIFOs and protocol control/status for one HOTLink channel. A programmable PLL is utilized to create a custom Byte I/O clock from 16 to 32 MHz for the HOTLink I/O. The PCI bus uses a 50 MHz clock and interfaces with an onboard PCI-to-PCIe bridge to provide a four-lane PCIe interface.

Each channel has a 256K byte receive data FIFO and a separate 256K byte bit transmit data FIFO implemented with FPGA internal RAM. These FIFOs can be accessed using either single-word reads or writes or DMA.

When PCIe4lHOTLinkx5-SMB is recognized by the PCI bus configuration utility it will load the PcieHLx5SMBBase driver which will create a device object for each device, initialize the hardware, create child devices for the I/O ports and request loading of the PcieHLx5SMBChan driver. The PcieHLx5SMBChan driver will create a device object for the I/O port and perform initialization. IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move blocks of DMA data in and out of the I/O.

The PCIe4lHOTLinkx5-SMB software package has two parts. The driver for Windows® 10 OS, and the User Application “UserAp” executable.

The driver is delivered electronically. The files supplied are installed into the client system to allow access to the hardware. 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.

The test software can be ported to your application to provide a running start. It is recommended to port the 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 test suite is designed to accommodate up to 5 boards. The number of boards can be expanded. See Main.c to increase the number of handles.

The hardware manual defines the pinout, the bitmaps and detailed configurations for each feature of the design. The driver handles all aspects of interacting with the hardware. For added explanations about what some of the driver functions do, please refer to the hardware manual for the version in use.

We strive to make a useable product. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us.

The reference software application has a loop to check for devices. The number of devices found, the locations, and device count are printed out at the top of the menu.

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 PCIe4lHOTLinkx5-SMB hardware manual.

Driver Installation

There are several files provided in each driver package. These files include PCIeHLx5SMBBasePublic.h, PCIeHLx5SMBChanPublic.h, PCIeHLx5SMBBase.inf, PCIeHLx5SMBBase.cat, PCIeHLx5SMBBase.sys, PCIeHLx5SMBChan.inf, PCIeHLx5SMBChan.cat, PCIeHLx5SMBChan.sys

Public.h is the C header file that defines the Application Program Interface (API) for the Base or Channel driver. This file is required at compile time by any application that wishes to interface with the drivers, but is not needed for driver installation. In addition, a second .h file is provided with handy definitions for HOTLink programming. UserAp includes this file and it is incorporated into the g_all.h global include within that package.

Windows 10 Installation

Copy the .sys, .inf, and .cat files to your preferred medium / location.

With the PCIe4LHOTLinkx5-SMB hardware installed, power-on the PCIe host computer.

- Open the ***Device Manager*** from the control panel.
- Under ***Other devices*** there should be an ***Other PCI Bridge Device****.
- Right-click on the ***Other PCI Bridge Device*** and select ***Update Driver Software***.
- Select ***Browse my computer for driver software***.
- Select ***adjust the path shown as needed***
- Select ***Next***.
- Select ***Close*** to close the update window.

The system should now display the PCIe4LHOTLinkx5-SMB adapter in the Device Manager. The Port will also show [now] as an unprogrammed device [missing driver]

- Repeat the above to install the channel portion of the driver

***** If the ***Other PCI Bridge Device*** is not displayed, click on the ***Scan for hardware changes*** icon on the tool-bar.

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 globally unique identifiers (GUID), which are defined in PcieHLx5SMBBasePublic.h and PcieHLx5SMBChanPublic.h. See main.c in the PcieHOTLinkUserApp project for an example of how to acquire handles for the base and channel devices.

Note: In order to build an application you must link with setupapi.lib.

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(), 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 PcieHLx5SMBBase driver are described below:

IOCTL_PcieHLx5SMB_BASE_GET_INFO

Function: Returns the device driver version, design version, design type, user switch value, device instance number and PLL device ID.

Input: None

Output: PcieHLx5SMB_BASE_DRIVER_DEVICE_INFO structure

Notes: The switch value is the configuration of the 8-bit onboard dipswitch that has been selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. See the definition of PcieHLx5SMB_BASE_DRIVER_DEVICE_INFO below.

```
// Driver/Device information
typedef struct _PcieHLx5SMB_BASE_DRIVER_DEVICE_INFO {
   UCHAR    DriverRev;  // driver revision
   UCHAR    DesignRev;     // Flash design revision major
   UCHAR    DesignRevMin; // Flash design revision minor
   UCHAR    DesignType;
   UCHAR    InstanceNum;  // number of device if >1
   UCHAR    SwitchValue;  // value of DIP switch
   UCHAR    PllDeviceId;   // ID of PLL installed
   UCHAR    BridgeCnfgd;  // True if bridge detected
} PcieHLx5SMB_BASE_DRIVER_DEVICE_INFO, *PPcieHLx5SMB_BASE_DRIVER_DEVICE_INFO;
```

IOCTL_PcieHLx5SMB_BASE_LOAD_PLL_DATA

Function: Writes to the internal registers of the PLL.

Input: PcieHLx5SMB_BASE_PLL_DATA structure

Output: None

Notes: The PcieHLx5SMB_BASE_PLL_DATA structure has only one field: Data – an array of 40 bytes containing the PLL register data to write. See below for the definition of PcieHLx5SMB_BASE_PLL_DATA.

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

typedef struct _PcieHLx5SMB_BASE_PLL_DATA {
   UCHAR    Data[PLL_MESSAGE_SIZE];
} PcieHLx5SMB_BASE_PLL_DATA, *PPcieHLx5SMB_BASE_PLL_DATA;
```

IOCTL_PcieHLx5SMB_BASE_READ_PLL_DATA

Function: Returns the contents of the internal registers of the PLL.

Input: None

Output: PcieHLx5SMB_BASE_PLL_DATA structure

Notes: The register data is written to the PcieHLx5SMB_BASE_PLL_DATA structure in an array of 40 bytes. See definition of PcieHLx5SMB_BASE_PLL_DATA above.

IOCTL_PcieHLx5SMB_BASE_SET_ENDIAN

Function: Controls Endianness of data .

Input: Endianness flag

Output: None

Notes: Set flag to enable byte swapping during DMA access for Big Endian operation. Target accesses are unaffected as data is converted by host in those cases. Clear flag to operate in standard Little Endian mode.

IOCTL_PcieHLx5SMB_BASE_BRIDGE_RECONFIG

Function: Attempt to reconfigure Bridge for optimize DMA settings

Input: none

Output: None

Notes: Call this IOCTL to reconfigure the bridge with optimized settings. Called during initialization. Occasionally the status returned by device info is negative. Re-run with this call to correct.

IOCTL_PcieHLx5SMB_BASE_GET_BASE

Function: Read the Base primary control register

Input: none

Output: ULONG

Notes: Read the base control register with this IOCTL. Not usually required as other IOCTLs with defined accesses are used. If you are curious what setting is really in there, read with this.

IOCTL_PcieHLx5SMB_BASE_GET_STATUS

Function: Read the base status register

Input: none

Output: ULONG

Notes: Read the base status register with this IOCTL. Interrupt and PLL status in the base status register.

IOCTL_PcieHLx5SMB_BASE_RESET

Function: Reset Control for DDR and ports – Currently unused

Input: none

Output: none

Notes: Currently not implemented.

IOCTL_PcieHLx5SMB_BASE_ENABLE_INTERRUPT

Function: Device level Master Interrupt Enable

Input: none

Output: none

Notes: Set to use user programmable interrupts. DMA calls automatically enable.

IOCTL_PcieHLx5SMB_BASE_DISABLE_INTERRUPT

Function: Device level Master Interrupt Disable

Input: none

Output: none

Notes: Set to block user programmable interrupts. Leave disabled to poll using interrupt status.

IOCTL_PcieHLx5SMB_BASE_SET_COUNT_CONTROL

Function: Sets the group-start trigger counter control configuration.

Input: PcieHLx5SMB_BASE_CHAN_START_CONFIG structure

Output: None

Notes: This call determines the group-start characteristics for enabled channel transmitters. If StartNow is true, then a trigger pulse is immediately sent to the channel transmitters. If CountEnable is true, the 20-bit counter starts counting at the rate of one megahertz. If ClearStart is true, the counter starts counting from zero, otherwise it loads the StartCount. When the counter reaches the TriggerCount, a trigger pulse is sent to the channel transmitters. If Continuous is true, the counter continues counting otherwise it stops when triggered. When the counter reaches the EndCount, the counter is re-initialized. If ClearEnable is true, the counter goes to zero, otherwise StartCount is loaded. See definition of PcieHLx5SMB_BASE_CHAN_START_CONFIG below.

```
typedef struct _PcieHLx5SMB_BASE_CHAN_START_CONFIG {
   BOOLEAN  StartNow;      // Start grouped transmit immediately
   BOOLEAN  CountEnable;   // Enable counter
   BOOLEAN  ClearStart;    // Clear count on start, else load StartCount
   BOOLEAN  ClearEnable;   // Clear count on rollover, else load StartCount
   BOOLEAN  Continuous;    // Counter runs continuously else single pass
   ULONG    StartCount;    // Preload count (0-0xfffff)
   ULONG    TriggerCount;  // Count to start transmissions (0-0xfffff)
   ULONG    EndCount;      // Rollover count (0-0xfffff)
} PcieHLx5SMB_BASE_CHAN_START_CONFIG, *PPcieHLx5SMB_BASE_CHAN_START_CONFIG;
```

IOCTL_PcieHLx5SMB_BASE_GET_COUNT_CONTROL

Function: Returns the group-start trigger counter control configuration.

Input: None

Output: PcieHLx5SMB_BASE_CHAN_START_CONFIG structure

Notes: Three 20-bit count registers and one control register are read and the information is returned in the PcieHLx5SMB_BASE_PLL_DATA structure. See definition of PcieHLx5SMB_BASE_CHAN_START_CONFIG above.

IOCTL_PcieHLx5SMB_BASE_GET_ISR_STATUS

Function: Returns the accumulated status that was read in the ISR.

Input: None

Output: Interrupt status value (unsigned long integer)

Notes: This call was added to test some of the group-start capabilities of the design. When a frame done interrupt occurs this call is immediately made and the channels that have the group-start feature enabled should all show an interrupt status in the BASE_INT_CHAN_MASK field. Since these status bits are cleared in the channel DPC, the status bits are accumulated until the GetIsrStatus call is made so they will not be lost as the channel DPCs are run. When the GetIsrStatus is made, the status bits are cleared. See the Base status-bit field definitions below.

```
#define BASE_INT_CHAN_MASK 0x0000003F
#define BASE_PLL_WR_MASK   0x00000700
#define BASE_PLL_RD_MASK   0x00007000
#define BASE_PLL_STAT_MASK 0x00070000
#define BASE_CHAN_MASK     0x07000000
```

The IOCTLs defined for the PcieHLx5SMBChan driver are described below:

IOCTL_PcieHLx5SMB_CHAN_GET_INFO

Function: Returns the driver version and instance number of the device.

Input: None

Output: PcieHLx5SMB_CHAN_DRIVER_DEVICE_INFO structure

Notes: See the definition of PcieHLx5SMB_CHAN_DRIVER_DEVICE_INFO below.

```
// Driver/Device information
typedef struct _PcieHLx5SMB_CHAN_DRIVER_DEVICE_INFO {
   UCHAR    DriverRev;  // channel driver revision
   UCHAR    ChannelNum; // 0 for single port design
   UCHAR    DesignRev; // Flash design Major Revision
   UCHAR….DesignRevMin; // Flash design Minor Revision
   UCHAR    DesignType;
   UCHAR    SwitchValue;   // Board user switch value
   UCHAR    InstanceNum;   // Board instance from base driver
} PcieHLx5SMB_CHAN_DRIVER_DEVICE_INFO,
*PPcieHLx5SMB_CHAN_DRIVER_DEVICE_INFO;
```

IOCTL_PcieHLx5SMB_CHAN_SET_CONFIG

Function: Sets the requested channel control configuration.

Input: PcieHLx5SMB_CHAN_CONFIG structure

Output: None

Notes: Specifies the enabled interrupt sources, DMA preemption behavior, transmit start mode, data storage mode and other control parameters. See the definitions of PcieHLx5SMB_CHAN_CONFIG and its subordinate structures below.

```
typedef struct _PcieHLx5SMB_CHAN_INTS {
   BOOLEAN  TxAmtInt;   // Transmit FIFO almost empty interrupt
   BOOLEAN  RxAfIInt;   // Receive FIFO almost full interrupt
   BOOLEAN  RxOvfIInt;  // Receive FIFO overflow interrupt
   BOOLEAN  TxFrmDnInt; // Transmit frame done interrupt
   BOOLEAN  RxFrmDnInt; // Receive frame done interrupt
} PcieHLx5SMB_CHAN_INTS, *PPcieHLx5SMB_CHAN_INTS;
```

```
// Channel DMA priority (use sparingly)
typedef enum _PcieHLx5SMB_DMA_PRMPT {
   PcieHLx5SMB_NONE,    // No priority
   PcieHLx5SMB_READ,    // Read DMA has priority
   PcieHLx5SMB_WRITE,   // Write DMA has priority
   PcieHLx5SMB_RDWR     // Read and Write DMA have priority
} PcieHLx5SMB_DMA_PRMPT, *PPcieHLx5SMB_DMA_PRMPT;
```

Note: this control to be depricated in next release in favor of HW control.

```
 // Channel Receiver storage mode
typedef enum _PcieHLx5SMB_RX_MODE {
   STORE_ALL,  // Store data and all control
   DATA_ONLY,  // Store data only
   SINGLE_CTRL,// Store data and non-repeated control
} PcieHLx5SMB_RX_MODE, *PPcieHLx5SMB_RX_MODE;

 // Channel Transmitter start mode
typedef enum _PcieHLx5SMB_TX_START {
   VIDEO_FRM,  // Video frame mode
   SYNC_NONE,  // Ignore group-sync signal
   SYNC_FIRST, // Synchronize first frame with sync signal
   SYNC_ALL,   // Synchronize all frames with sync signal
} PcieHLx5SMB_TX_START, *PPcieHLx5SMB_TX_START;
```

```
typedef struct _PcieHLx5SMB_CHAN_CONFIG {
   BOOLEAN          TxEnable;   // Enable HOTLink transmitter
   BOOLEAN          RxEnable;   // Enable HOTLink receiver
   BOOLEAN          FifoTestEn; // Enables auto tx->rx FIFO transfer
   BOOLEAN          TxOutEn;    // Enable transmitter output
   BOOLEAN          TxBitEn;    // Built-in-test enable (sends test pattern)
   BOOLEAN          TxLdEn;     // Enables loading of built-in-test data
   BOOLEAN          TxClearEn;  // Enables clearing Tx Frame request when frame done
   BOOLEAN          RxInASel;   // Selects Rx input '1'=External, '0'=Local Tx
   BOOLEAN          RxBitEn;    // Built-in-test enable (verifies test pattern)
   BOOLEAN          RxReframe;  // '1'=K28.5 re-syncs data, '0'=sync locked
   BOOLEAN          RxTestEn;   // Forces the receiver to start immediately
   BOOLEAN          MuxEnable;  // '1'=Enable Tx/Rx Mux, '0'=Mux disabled
   BOOLEAN          TxSelect;   // '1'=Transmit selected, '0'=Receive selected
   BOOLEAN          NoStopSeq;  // '1'=No stop sequence, '0'=Send stop sequence
   PcieHLx5SMB_TX_START    TxStart;    // Video or various stream synch options
   PcieHLx5SMB_RX_MODE     RxStoreMd;  // Receiver storage mode
   PcieHLx5SMB_CHAN_INTS   IntConfig;  // Interrupt condition enables
   PcieHLx5SMB_DMA_PRMPT   DmaPriority;// DMA preemption control
} PcieHLx5SMB_CHAN_CONFIG, *PPcieHLx5SMB_CHAN_CONFIG;
```

IOCTL_PcieHLx5SMB_CHAN_GET_CONFIG

Function: Returns the fields set in the previous call.

Input: None

Output: PcieHLx5SMB_CHAN_CONFIG structure

Notes: See the definitions of PcieHLx5SMB_INTS, PcieHLx5SMB_DMA_PRMPT, PcieHLx5SMB_RX_MODE, PcieHLx5SMB_TX_START and PcieHLx5SMB_CHAN_CONFIG above.

IOCTL_PcieHLx5SMB_CHAN_GET_STATUS

Function: Returns the channel's status register value and clears the latched status bits.

Input: None

Output: Value of the channel's status register (unsigned long integer)

Notes: See the status bit definitions below. Only the bits in CHAN_STAT_MASK will be returned. The bits in CHAN_STAT_LATCH_MASK will be cleared by this call only if they are set when the register was read. This prevents the possibility of missing an interrupt condition that occurs after the register has been read but before the latched register bits are cleared. See definitions in HW manual [for the bits listed]

```
// Status bit definitions
#define CHAN_STAT_TX_FF_MT              0x00000001
#define CHAN_STAT_TX_FF_AMT             0x00000002
#define CHAN_STAT_TX_FF_FL              0x00000004
#define CHAN_STAT_TX_FF_VLD             0x00000008
#define CHAN_STAT_RX_FF_MT              0x00000010
#define CHAN_STAT_RX_FF_AFL             0x00000020
#define CHAN_STAT_RX_FF_FL              0x00000040
#define CHAN_STAT_RX_FF_VLD             0x00000080
#define CHAN_STAT_TX_AMT_INT            0x00000100
#define CHAN_STAT_RX_AFL_INT            0x00000200
#define CHAN_STAT_RX_OVFL               0x00000400
#define CHAN_STAT_RX_SYM_ERR            0x00000800
#define CHAN_STAT_WR_DMA_INT            0x00001000
#define CHAN_STAT_RD_DMA_INT            0x00002000
#define CHAN_STAT_WR_DMA_ERR            0x00004000
#define CHAN_STAT_RD_DMA_ERR            0x00008000
#define CHAN_STAT_WR_DMA_RDY            0x00010000
#define CHAN_STAT_RD_DMA_RDY            0x00020000
#define CHAN_STAT_RX_DATA_RDY           0x00040000
#define CHAN_STAT_TX_DATA_READ          0x00080000
#define CHAN_STAT_TX_FRAME_DN           0x00100000
#define CHAN_STAT_RX_FRAME_DN           0x00200000
#define CHAN_STAT_RX_ACTIVE             0x00400000
#define CHAN_STAT_RXIO_FF_FL            0x00800000
#define CHAN_STAT_AUX_FF_MT             0x01000000
#define CHAN_STAT_AUX_FF_AFL            0x02000000
#define CHAN_STAT_AUX_FF_FL             0x04000000
#define CHAN_STAT_TX_DAT_VLD            0x08000000
#define CHAN_STAT_TX_VLD_MASK           0x30000000
#define CHAN_STAT_LOC_INT               0x40000000
#define CHAN_STAT_INT_ACTIVE            0x80000000
```

IOCTL_PcieHLx5SMB_CHAN_SET_FIFO_LEVELS

Function: Sets the transmitter almost empty and receiver almost full levels for the channel.

Input: PcieHLx5SMB_CHAN_FIFO_LEVELS structure

Output: None

Notes: These values are initialized to the default values ⅛ FIFO and ⅞ FIFO respectively when the driver initializes. The FIFO counts are compared to these levels to set the state of the CHAN_STAT_TX_FF_AMT and CHAN_STAT_RX_FF_AFL status bits and latch the CHAN_STAT_TX_AMT_LT and CHAN_STAT_RX_AFL_LT latched status bits. Also if the control bits CHAN_CNTRL_URGNT_OUT_EN and/or CHAN_CNTRL_URGNT_IN_EN are set, the FIFO level values are used to determine when to give priority to an output or input DMA channel that is running out of data or room to store data. See the definition of PcieHLx5SMB_CHAN_FIFO_LEVELS below.

```
typedef struct _PcieHLx5SMB_CHAN_FIFO_LEVELS {
   ULONG    AlmostFull;
   ULONG    AlmostEmpty;
} PcieHLx5SMB_CHAN_FIFO_LEVELS, *PPcieHLx5SMB_CHAN_FIFO_LEVELS;
```

IOCTL_PcieHLx5SMB_CHAN_GET_FIFO_LEVELS

Function: Returns the transmitter almost empty and receiver almost full levels for the channel.

Input: None

Output: PcieHLx5SMB_CHAN_FIFO_LEVELS structure

Notes: Returns the values set in the previous call.

IOCTL_PcieHLx5SMB_CHAN_GET_FIFO_COUNTS

Function: Returns the number of data words in the transmit and receive data FIFOs.

Input: None

Output: PcieHLx5SMB_CHAN_FIFO_COUNTS structure

Notes: The counts represent the total data stored in the pipelines. The main storage is from the two 256Kbyte FIFOs. Additional storage to support transfer functions increases the count slightly. See HW manual for more detail. See the definition of PcieHLx5SMB_CHAN_FIFO_COUNTS below.

```
typedef struct _PcieHLx5SMB_CHAN_FIFO_COUNTS {
   ULONG   TxCount;
   ULONG   RxCount;
} PcieHLx5SMB_CHAN_FIFO_COUNTS, *PPcieHLx5SMB_CHAN_FIFO_COUNTS;
```

IOCTL_PcieHLx5SMB_CHAN_RESET_FIFOS

Function: Resets one or both FIFOs for the referenced channel.

Input: PcieHLx5SMB_FIFO_SEL enumeration type

Output: None

Notes: Resets the transmit or receive FIFO or both depending on the input parameter selection. See the definition of PcieHLx5SMB_CHAN_FIFO_SEL below.

```
 // Used for FIFO reset call
typedef enum _PcieHLx5SMB_CHAN_FIFO_SEL {
   PcieHLx5SMB_TX,
   PcieHLx5SMB_RX,
   PcieHLx5SMB_BOTH
} PcieHLx5SMB_CHAN_FIFO_SEL, *PPcieHLx5SMB_CHAN_FIFO_SEL;
```

IOCTL_PcieHLx5SMB_CHAN_WRITE_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_PcieHLx5SMB_CHAN_READ_FIFO

Function: Returns a 32-bit data word from the receive FIFO.

Input: None

Output: FIFO word (unsigned long integer)

Notes: Used to make single-word accesses to the receive FIFO instead of using DMA.

IOCTL_PcieHLx5SMB_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 this event to be signaled.

IOCTL_PcieHLx5SMB_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 user interrupt occurs to re-enable it.

IOCTL_PcieHLx5SMB_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_PcieHLx5SMB_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.

IOCTL_PcieHLx5SMB_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 status that was read while servicing the last interrupt caused by one of the user-enabled channel interrupt conditions. The interrupts that deal with the DMA transfers do not affect this value. The new field is true if the stored ISR status has been updated since the last time this call was made. See below for the definition of PcieHLx5SMB_CHAN_ISR_STATUS.

```
typedef struct _PcieHLx5SMB_CHAN_ISR_STATUS {
  ULONG    Status;  // Value of status register read in ISR
  BOOLEAN  New;     // True if the status has changed since last GetIsrStatus call
} PcieHLx5SMB_CHAN_ISR_STATUS, *PPcieHLx5SMB_CHAN_ISR_STATUS;
```

IOCTL_PcieHLx5SMB_CHAN_SET_IO_PARAMS

Function: Sets the start and stop sequences and byte count for I/O transfers.

Input: PcieHLx5SMB_CHAN_IO_PARAMS structure

Output: None

Notes: Start and Stop sequences are inserted by the transmitter to mark the beginning and end of a data-frame. The receiver uses these sequences to determine when to start storing data, when to stop, and for detecting byte-counts for each data-frame. A new field, FrmSpcr was added to control subsequent frame timing in the sync initial frame transmit start mode. This count determines the number of idle bytes sent between frames. See the definitions of PcieHLx5SMB_CHAN_IO_CHAR and PcieHLx5SMB_CHAN_IO_PARAMS below.

```
typedef struct _PcieHLx5SMB_CHAN_IO_CHAR {
   BOOLEAN CntrlChar;
   UCHAR   Byte;
} PcieHLx5SMB_CHAN_IO_CHAR, *PPcieHLx5SMB_CHAN_IO_CHAR;

#define MAX_CHARS_PER_SEQ  3

 // Defaults loaded if fields are zero
 // (Start-0x105,0x104; Stop-0x104,0x105; Count-0x008000)
typedef struct _PcieHLx5SMB_CHAN_IO_PARAMS {
   UCHAR              StartCnt;            // Zero to three characters
   UCHAR              StopCnt;             // Zero to three characters
   PcieHLx5SMB_CHAN_IO_CHAR   StartSeq[MAX_CHARS_PER_SEQ];
   PcieHLx5SMB_CHAN_IO_CHAR   StopSeq[MAX_CHARS_PER_SEQ];
   ULONG              ByteCnt;             // 16 MByte max count (24 bits)
   ULONG              FrmSpcr;             // 16 MByte max count (24 bits)
} PcieHLx5SMB_CHAN_IO_PARAMS, *PPcieHLx5SMB_CHAN_IO_PARAMS;
```

IOCTL_PcieHLx5SMB_CHAN_GET_IO_PARAMS

Function: Returns the start and stop sequences, byte count and inter-frame spacer for I/O transfers.

Input: None

Output: PcieHLx5SMB_CHAN_IO_PARAMS structure

Notes: Returns the values set in the previous call. See structure definitions above.

IOCTL_PcieHLx5SMB_CHAN_GET_STATUS_II

Function: Read the expansion Status Register

Input: None

Output: ULONG

Notes: See HW manual for current bit meanings. See public.h for #define statements.

Write

HOTLink 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.

See examples in the UserAp file set.

Read

HOTLink 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.

See examples in the UserAp file set

Warranty and Repair

https://www.dyneng.com/warranty.html

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 or e-mail and arrange to work with an engineer. We will work with you to determine the cause of the issue.

Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois, Suite B/C
Santa Cruz, CA 95060
(831) 457-8891
support@dyneng.com