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

User Manual

PCI LVDS 8T Driver Documentation

Revision A
Corresponding Hardware: Revision C
10-2001-0903

PCI LVDS 8T
PCI based 8 channel LVDS transmitter card

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

Manual Revision A. Revised April 11, 2002

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

List of Figures

no figures in this document

Introduction

The LV8T driver is a Windows NT driver for the PCI LVDS 8T board from Dynamic Engineering. This driver can control up to 10 boards in a system. If more boards are required please contact Dynamic Engineering. The PCI LVDS 8T board sends 8 channels of LVDS data. A separate "Device Object" controls each LVDS channel, and a separate handle references each Device Object. IO Control calls (IOCTLs) are used to configure the hardware and the driver. WriteFile() calls are used to load LVDS data into the device. IOCTLs refer to a single Device Object instance, and therefore refer to a single channel on a device (except the IOCTL_LV8T_RESET_ALL_OUTPUT_FIFOS call, which affects all 8 channels on a given board).

Handles can be opened to specific channels on the device 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 LV8T driver, Symbolic Link names are formed as LV8Tx_n where x indicates the zero based board number and n indicates the zero based channel number on the board. E.g. the second channel on the third board is LV8T2_1.

WriteFile() is used to write LVDS data to the SDRAM space associated with a specific channel. The channel is specified by passing the appropriate handle opened via the CreateFile() function call. The driver writes directly into one of two SDRAM banks on the device. WriteFile() calls are limited to channels 0 and 4 on the device, depending on which bank is to be accessed. Channel 0 accesses the first SDRAM bank, which is used to store data for output channels 0 – 3. Channel 4 accesses the second SDRAM bank, used for storing data for output channels 4 – 7.

Authors Note

This documentation is provided to supplement the PCI_LVDS_8T [LV8T] hardware manual. 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.

Driver Installation

There are several files provided in each driver drop. These files include lv8t.sys, lv8t.reg, ddlv8t.h, lv8ttest.exe, and driver source files.

The lv8t.sys file is the binary driver file. In order to install the driver, place this file in your Winnt\system32\drivers directory.

The lv8t.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 LV8T 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 ddlv8t.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 PCI LVDS 8T device. It is not needed by the driver installation.

The lv8ttest.exe file is a sample Windows NT console application that makes calls into the LV8T driver. It is not required during the driver installation.

Driver Startup

There are several tasks the LV8T driver must do when it is started. It must scan all possible PCI buses to detect every PCI LVDS 8T device in the system. It must create 8 "Device Objects" for every board it finds, one per channel. 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 PCI LVDS 8T board.

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 channel. 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_LV8T_GET_STATUS

Function: Returns the status of a specified channel.
Input: None
Output: LV8T_STATUS
Notes: Returns Status information for a given channel obtained from the DMA Status register, the Address Generator Address A and Address B registers, and the Tx Status register. The LV8T_STATUS structure returned contains two ULONG fields: CardID and ChannelState. The CardID field will contain the settings of the user-defined dip-switch on the board. ChannelState will contain various information about the activity of the channel and the state of the FIFOs. See the definition of LV8T_STATUS for more information.

IOCTL_LV8T_SET_MEMORY_CONFIG

Function: Sets the specified channel's memory configuration.
Input: LV8T_MEMORY
Output: None
Notes: Sets the Address Generator memory configuration for a given channel. The LV8T_MEMORY input structure contains four ULONG fields: Add_A, Add_B, Add_C, and Add_D; a UCHAR field: LoopCount; and two boolean fields: LoopEnable and Continuous. The four address fields specify the beginning and end of memory regions used in the different addressing modes for the specified channel. The LoopCount field specifies the number of loops performed in loop mode, which is enabled by the LoopEnable field. The Continuous field enables continuous operation where the channel starts over upon reaching the end of the specified memory operation. See the PCI LVDS 8T manual for more information.

IOCTL_LV8T_GET_MEMORY_CONFIG

Function: Retrieves the specified channel's memory configuration.
Input: None.
Output: LV8T_MEMORY
Notes: Retrieves the Address Generator memory configuration for the specified channel. The LV8T_MEMORY output structure is described above.

IOCTL_LV8T_SET_CHANNEL_CONFIG

Function: Sets up the Tx registers for the appropriate channel.
Input: LV8T_CHANNEL_CONFIG

Output: None
Notes: Sets up the specified channel's configuration using the information supplied in the LV8T_CHANNEL_CONFIG input structure. This structure contains four UCHAR fields: WCount, XCount, YCount, and ZCount. These fields specify the number of times samples of different types are repeated when the expand mode is enabled. Two USHORT fields: Idle0 and Idle1 specify the pattern that is sent before the transmission is started and after it ends, respectively. The remaining five fields are boolean quantities: ExtTrig1Enable and ExtTrig2Enable when TRUE require the assertion of the respective external trigger to start transmission, Expand enables the expand mode described above, ClockSelect selects the Tx clock when TRUE and the PCI clock otherwise, finally SerializeEnable enables the LVDS serializer and drivers.

IOCTL_LV8T_GET_CHANNEL_CONFIG

Function: Retreives information from the specified channel's Tx registers
Input: None
Output: LV8T_CHANNEL_CONFIG
Notes: Retreives information from the specified channel's Tx registers. The LV8T_CHANNEL_CONFIG output structure is described above.

IOCTL_LV8T_SET_TIMING_CONFIG

Function: Sets the timing parameters for the data transmission
Input: LV8T_TIMING_CONFIG
Output: None
Notes: This call can be made from any channel with the same effect, since the same register is accessed. Sets the timing configuration using the information supplied in the LV8T_TIMING_CONFIG input structure. The structure contains a USHORT field: Divisor that controls the frequency of the divided clock when this is enabled, an enumerated type field: ClockSource that specifies one of four possible clock sources, and three boolean fields. DivClock enables the use of the divided clock when TRUE or the clock source when FALSE. MasterMode configures the board as a master (supplies start pulse and clock to other boards) when TRUE or a slave (receives and retransmits start pulse and clock) when FALSE. ExtPulseEn determines the source of the start pulse that is output when MasterMode is true. If ExtPulseEn is TRUE the pulse input received from the front panel connector is re-transmitted out the rear and front panel outputs, when FALSE the locally generated pulse is transmitted.

IOCTL_LV8T_GET_ TIMING_CONFIG

Function: Retrieves the timing configuration.
Input: None
Output: LV8T_TRIGGER_CONFIG
Notes: Retrieves the timing configuration information from the board, any channel can be used to access this information. The LV8T_TRIGGER_CONFIG output structure is described above.

IOCTL_LV8T_SET_LOCAL_START

Function: Configures the channel specific start and restart parameters.
Input: LV8T_LOCAL_START_CONFIG
Output: None
Notes: Sets the local start parameters for the specified channel. The LV8T_LOCAL_START_CONFIG input structure contains two boolean fields: Start , which sets the Address generator and Tx start bits, and Restart which enables the Tx state machine to transition from the terminal idle state back to the initial idle state. This IOCTL is used in conjunction with the next (IOCTL_LV8T_SET_MASTER_START) to start the channel transmission.

IOCTL_LV8T_SET_MASTER_START

Function: Configures the overall start and external trigger output parameters.
Input: LV8T_MASTER_START_CONFIG
Output: None
Notes: Sets the master start signal and external enable pulse output. The LV8T_MASTER_START_CONFIG input structure contains two boolean fields: Pulse , which generates a pulse that is driven off the board when the board is configured as a master, and Enable, which is used in conjunction with the local start to begin the data transmission.

IOCTL_LV8T_START_READBACK_MODE

Function: Configures the specified channel to read back Tx data over the PCI bus.
Input: None
Output: LV8T_RDBK_CONFIG
Notes: This IOCTL is provided as a debugging tool. It allows data written to the output FIFO to be read back over the PCI bus. The LV8T_RDBK_CONFIG output structure stores parameter values to be

restored by IOCTL_LV8T_STOP_READBACK_MODE when the data read back is stopped.

IOCTL_LV8T_STOP_READBACK_MODE

Function: Configures the specified channel to read back Tx data over the PCI bus.
Input: LV8T_RDBK_CONFIG
Output: None
Notes: This IOCTL is called when Tx data read back is no longer desired. It restores the parameters stored in the LV8T_RDBK_CONFIG input structure to the values that were present before the read back mode was started.

IOCTL_LV8T_GET_TX_DATA

Function: Reads one 32-bit Tx data word.
Input: None
Output: ULONG
Notes: This IOCTL is used after read back mode is initiated. Each time it is called it retrieves one 32-bit Tx data word, which is comprised of two LVDS data samples.

IOCTL_LV8T_START_SLAVEWRITE_MODE

Function: Configures the specified channel to write data to the SDRAM without DMA.
Input: ULONG
Output: None
Notes: This IOCTL is provided as a debugging tool. It allows data to be written to the SDRAM over the PCI bus one word at a time. It is only valid for channels 0 and 4 depending on which SDRAM bank is to be written. The input parameter is the number of bytes of data to be written. The value set by IOCTL_LV8T_SET_WRITE_OFFSET is used as the starting point for the data writes.

IOCTL_LV8T_STOP_SLAVEWRITE_MODE

Function: Configures the specified channel to read back Tx data over the PCI bus.
Input: None
Output: ULONG

Notes: This IOCTL stops the slave write mode started in the previous IOCTL and restores the memory configuration that existed before this mode was started. The parameter returned is the number of bytes actually written to memory.

IOCTL_LV8T_PUT_RAM_DATA

Function: Writes one 32-bit data word to the SDRAM.
Input: ULONG
Output: None
Notes: This IOCTL is used after the slave write mode is initiated. Each time it is called it writes one 32-bit data word to the SDRAM.

IOCTL_LV8T_SET_WRITE_OFFSET

Function: Sets the starting offset in SDRAM for a WriteFile or SlaveWrite operation.
Input: ULONG
Output: None
Notes: Sets a software variable that is used to set the Starting Address in the SDRAM for WriteFile and SlaveWrite calls. This IOCTL will succeed only if called on channel 0 or channel 4.

IOCTL_LV8T_GET_WRITE_OFFSET

Function: Retrieves the starting offset in SDRAM for a WriteFile or SlaveWrite operation.
Input: None
Output: ULONG
Notes: Retrieves the software variable that is used to set the Starting Address in the SDRAM for WriteFile and SlaveWrite calls. This IOCTL will succeed only if called on channel 0 or channel 4.

IOCTL_LV8T_RESET_ALL_OUTPUT_FIFOS

Function: Resets all 8 output FIFOs
Input: None
Output: None
Notes: This function resets all 8 output FIFOs. In order to accomplish this, the function selects the PCI clock for all Tx channels. It then resets the output FIFOs by writing to the DMA Base Control register. The Tx channels are then restored to their original values. This function will affect all 8

channels. No other operations should be taking place when this function is called.

Write

DMA mode

LV8T DMA data is written to the device using write driver calls. A write call refers to a single Device Object in the driver, which controls a single channel. Writes are executed using the Win32 function WriteFile() and passing in the handle to the device opened with CreateFile(). WriteFile() takes as an input parameter a pointer to a pre allocated buffer and a DWORD that represents the size of the buffer. WriteFile () takes as an output parameter a pointer to a DWORD that represents the number of bytes written by WriteFile (). WriteFile can only be used with channel 0 or channel 4. Use channel 0 to access the 256MB SDRAM bank used for channels 0-3. Use channel 4 to access the 256MB SDRAM bank used for channels 4-7. To use WriteFile, first set up the Write Offset. The Write Offset is the offset in the SDRAM bank at which you wish to start your WriteFile. The default Offset when the driver first loads will be zero. Valid values for a Write Offset are between 0 and 0x1ffffff. The WriteFile will begin at the Write Offset and continue writing data until the specified length is transferred or the end of SDRAM is reached. A WriteFile call will not ultimately alter the channel's memory configuration. It will temporarily set the Address Generator registers to appropriate values to complete the WriteFile call, but when the WriteFile call completes these registers will be returned to their original settings.

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 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