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

SpaceWireDriver.rta

Driver Documentation

INtime Real-Time Driver

Manual Revision A
Corresponding Hardware: 10-2004-0803
PMC-SpaceWire Revision C
Corresponding Hardware: 10-2006-0102
PCI-SpaceWire Revision B
Corresponding Hardware: 10-2008-1101
ccPMC-SpaceWire Revision A
Corresponding Hardware: 10-2009-0902
PC104p-SpaceWire Revision B
Corresponding Firmware: Revision H

SpaceWireDriver.rta
INtime Real-Time Driver for the
(cc)PMC/PCI/PC104p-SpaceWire
4-Channel SpaceWire Interface

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

Manual Revision A Revised October 26, 2011

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

SpaceWireDriver.rta is a real-time application/device driver developed and used with the INtime® operating system from TenAsys Corporation. The driver can control a PMC-SpaceWire, ccPMC-SpaceWire, PCI-SpaceWire or PC104p-SpaceWire PCI-based SpaceWire interface board from Dynamic Engineering.

Some changes to the SpaceWire FPGA program were required in order to make it possible to be used by INtime. INtime works in conjunction with Windows to allow the creation of real-time processes that coexist and communicate with Windows using dedicated core(s) on a multi-core processor or sharing the core of a single-core processor. INtime cannot share interrupts with a Windows device (due to real-time determinism issues). Almost all PCI devices must share interrupt lines, so legacy (INTA-D) interrupts are difficult or impossible to use. This version of the Xilinx FPGA program requests MSI (message-signaled-interrupts), which are not shared and thus avoids this issue. Although most Windows systems do not support MSI, the INtime software does, so devices that are passed to INtime control can use this feature.

The required changes to the FPGA program added enough complexity so that the design will no longer fit in the Spartan3-1000 FPGA. For some time we have been building SpaceWire boards with the Spartan3-1500 FPGA with the same footprint, while maintaining version updates for the Spartan3-1000. With this design update, we can no longer support the legacy part.

The Spartan3-1500 Xilinx FPGA implements the PCI interface, internal FIFOs and protocol control and status for four SpaceWire channels. There is also a programmable PLL with four clock outputs to create separate programmable I/O clocks for each SpaceWire channel. Each channel has either two 1k x 32-bit internal data FIFOs; or a 1k x 32-bit internal transmit data FIFO and a 128k x 32-bit external receive data FIFO; or two 128k x 32-bit external data FIFOs for transmit and receive data. All channels have two 1023 x 32-bit packet-length FIFOs that support 31-bit packet lengths plus one error bit per packet.

When the SpaceWire PCI signature is recognized by the INtime software, the interrupt is initialized and device register memory allocated. Hardware initialization routines write default values in the stored variables and control registers, determine the device ID of the PLL and program the PLL with a default program that sets all four I/O channel clocks to 100 MHz. Each channel is also initialized; its FIFO sizes detected and default values written to the programmable transmit almost empty and receive almost full FIFO registers. These levels correspond to 1/8 and 7/8 of their respective FIFO sizes.

Numerous threads are started to communicate with any application that wishes to use the SpaceWire board. Each thread creates a mailbox to receive requests or commands and another to return status or results. The handles to these mailboxes are cataloged so that they are known and available to other applications.
There is one thread that handles communication with the base level of the SpaceWire

board and there are three threads for each of the I/O channels. One channel thread handles I/O control/status messages, one handles write requests and the last handles read requests.

Notes

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 SpaceWire hardware manual.

Recent Firmware Updates

Revision D: Improved high-speed performance for packet-lengths not divisible by four; extended the maximum packet-lengths to 128 K Bytes and 2 G Bytes depending on Xilinx part and FIFO configuration; added a control bit to reuse a single packet-length and added latched transmit FIFO almost empty and receive FIFO almost full status bits.

Revision E: Corrected low-speed connection problem caused by missing back-to-back FCTs; revised handling of EEPs so that they do not cause a connection error when received and allow sending of an EEP by setting the error bit in the transmit packet-length FIFO (needed by router nodes); revised Xilinx configuration/revision numbering.

Revision F: Added transmit packet-length FIFO count field above transmit data FIFO count field at the same address; replaced transmit packet-length FIFO full status bit with transmitter purge error latched status bit to alert the user of a failure to purge all the data from an aborted transmit packet after a connection error; revised flow-control to be more responsive and efficient.

Revision G: Added receive packet-length FIFO count field above receive data FIFO count field at the same address; revised input and output DMA state-machines to avoid a potential lock-up condition when preemption (DMA arbitration priority) is enabled. Six-bit retry counters were added to each of the eight DMA controllers. The counts are compared to a value written to bits 31-26 of the base control register.

Revision H: Special version developed for use with the INtime® real-time operating system. This version requests MSI (message-signaled-interrupts) so that the SpaceWire card does not have to share interrupts with another PCI device. Although most Windows systems do not support MSI, the INtime software does, so devices that are passed to INtime control can use this feature. If MSI is not enabled by the operating system, the legacy interrupts are used.

Software Installation

See the INtime® User's Guide from TenAsys Corproration for instructions on installing the INtime software on your system.

In order to obtain large memory buffers for read and write DMA, it was necessary to modify the default configuration in the INtime Node Management control panel as follows:

- Under the System Wide tab change the Default Vseg Size (in MBytes) to 512.
- Under the Kernel tab change the Kernel memory (in MB) to 512.

Driver Startup

If not done already, open the INtime Device Manager and right-click on the SpaceWire device. Select Pass to INtime (non interrupt or MSI) and save the configuration (you will probably need to restart your system). Open the Node Management control panel and start the INtime software node that will control the SpaceWire board. After the real-time node is started, open the INtime Explorer and select Load RT app either from the File menu or the Toolbar just below the menu selections. A navigation panel will appear with the title "Load Real Time Application File". Navigate to the location of SpaceWireDriver.rta, select it and click on the Open button.

The driver will start, configure the SpaceWire board, and make messaging mailboxes available to communicate with your application.

Control/Status Messages

All control/status messages consist of a one-byte command identification code followed by zero or more bytes containing the data required to complete the request. A return message of one or more bytes is always sent. If no return data is required, a single zero-byte is returned. If return data is requested, for instance reading a status register, that data value or data structure will be returned in a byte sequence of the required length. All messages in both directions are terminated with a null character '\0' = 0.

The control/status messages defined for the SpaceWire INtime base driver message thread are defined in SpWrBasePublic.h and are described below:

SPWR_BASE_GET_INFO

Function: Returns the Driver version, Xilinx revision, Switch value, Instance number, and PLL ID.
Input: None
Output: SPWR_BASE_INFO structure
Notes: The design type and design rev are defined in the Xilinx FPGA program and read from the switch/user info port. The switch value is the configuration of the on-board dip-switch that is 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 SpWrBasePublic.h for the definition of SPWR_BASE_INFO.

SPWR_BASE_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL device.
Input: SPWR_PLL_DATA structure
Output: None
Notes: After the PLL has been configured, the register array data is analyzed to determine the programmed frequencies and the IO clock A-D initial divisor fields in the base control register are automatically updated.

SPWR_BASE_READ_PLL_DATA

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

SPWR_BASE_SET_TIME_CONFIG
Function: Sets the time-code master timing and channel routing on the SpaceWire board.
Input: SPWR_BASE_TIME_CONFIG structure
Output: None
Notes: The master time counter controls the TICK rate for SpaceWire time-code tokens. This counter is clocked by the 80 MHz link clock and has a maximum count of 1,048,575, which is equivalent to ~13.1 milliseconds between time-codes. The *Count* field in the input data structure is the count at which the master counter will roll-over to zero, increment the six-bit time-code data-count and issue a TICK_IN pulse to any channels configured as a time master. The *Flags* field specifies the two control flag bits sent in bit 6 and 7 of the time-code data byte. *TimeSource* is a four-deep array of SPWR_TM_SRC enumerated type values that determine the source of time-codes sent by each of the four I/O channels. These values specify one of the following six time-code sources: master timer, any of the four channel's time-code outputs, or none (disabled). Time-codes are the highest priority token of a SpaceWire network.

SPWR_BASE_GET_TIME_CONFIG
Function: Returns the time-code timing and routing on the SpaceWire board.
Input: None
Output: SPWR_BASE_TIME_CONFIG structure
Notes: Returns the values set in the previous call.

SPWR_BASE_SET_DMA_RETRY_COUNT
Function: Sets the six-bit retry count field value.
Input: Retry count (unsigned character)
Output: None
Notes: This count is used by each of the DMA controllers to compare to their individual retry counters. When a DMA controller requests the PCI bus, but is unable to transfer any data it increments its retry counter. If it succeeds in transferring data the retry counter is cleared. If the counter reaches the specified retry count value, the counter is cleared and the controller is forced to drop its bus request to the DMA arbiter. If other DMA controllers are requesting the bus, the DMA arbiter selects one of these and grants it access to the PCI bus. A stored retry count of zero results in unlimited retries.

SPWR_BASE_GET_DMA_RETRY_COUNT
Function: Returns the six-bit retry count field value.
Input: None
Output: Retry count (unsigned character)
Notes: Returns the value set in the previous call.

The control/status messages defined for the SpaceWire INtime channel driver message threads are defined in SpWrChanPublic.h and are described below:

SPWR_CHAN_GET_INFO
Function: Returns the driver version, instance number and transmit and receive FIFO sizes.
Input: None
Output: SPWR_CHAN_INFO structure
Notes: See the definition of SPWR_CHAN_INFO in the SpWrChanPublic.h header file.

SPWR_CHAN_SET_CONFIG
Function: Writes a configuration value to the channel control register.
Input: Value of channel control register (unsigned long integer)
Output: None
Notes: See SpWrChanPublic.h for the relevant channel control bit definitions. Only the bits in CHAN_CNTRL_MASK can be controlled by this call.

SPWR_CHAN_GET_CONFIG
Function: Returns the channel's control configuration.
Input: None
Output: Value of the channel control register (unsigned long integer)
Notes: Returns the values of the bits in CHAN_CNTRL_READ_MASK.

SPWR_CHAN_GET_STATUS
Function: Returns the channel's status value and clears the latched bits.
Input: None
Output: Value of channel status register (unsigned long integer)
Notes: The latched bits in CHAN_STAT_LATCH_MASK will be cleared if they are set when the status is read. Even though CHAN_STAT_TICK_RCVD is latched, it is not cleared by this call. It will be cleared in the ISR if the CHAN_CNTRL_TICK_INTEN bit is set or by the SPWR_CHAN_READ_TIME_CODE call.

SPWR_CHAN_WRITE_PACKET_LENGTH
Function: Writes a transmit packet-length value to the transmitter packet-length FIFO.
Input: Packet length value (unsigned long integer)
Output: None
Notes: When operating in packet mode, no data will be sent until at least one value is written to the transmit packet-length FIFO. Setting bit 32 high causes the transmitted packet to be terminated with an EEP rather than an EOP. This is used when relaying a packet that was either terminated prematurely by an error condition, or was received with an EEP.

SPWR_CHAN_READ_PACKET_LENGTH
Function: Reads a received packet-length value from the receiver packet-length FIFO.
Input: None
Output: Packet length value (unsigned long integer)
Notes: Only bits 30-0 are used for the packet-length (maximum of 2 G Bytes). Bit 32 is an error flag that indicates that an error condition occurred during the reception of the referenced packet or that it was terminated by an EEP when it was received. Reading the channel status will indicate whether a connection error was detected.

SPWR_CHAN_SET_FIFO_LEVELS
Function: Sets the transmitter almost empty and receiver almost full levels for the channel.
Input: SPWR_CHAN_FIFO_LEVELS structure
Output: None
Notes: These values are initialized to the default values ⅛ and ⅞ FIFO size respectively when the driver initializes. The FIFO counts are compared to these levels to set the value 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, these 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.

SPWR_CHAN_GET_FIFO_LEVELS
Function: Returns the transmitter almost empty and receiver almost full levels for the channel.
Input: None
Output: SPWR_CHAN_FIFO_LEVELS structure
Notes: Returns the values set in the previous call.

SPWR_CHAN_GET_FIFO_COUNTS
Function: Returns the number of data words in the transmit and receive data and packet-length FIFOs.
Input: None
Output: SPWR_CHAN_FIFO_COUNTS structure
Notes: There is one pipe-line latch for the transmit FIFO data and four for the receive FIFO data. These are counted in the FIFO counts. That means, for the internal FIFO channels, the transmit count can be a maximum of 1025 32-bit words and the receive count can be a maximum of 1028 32-bit words. For external FIFO channels, the transmit count can be a maximum of 131,073 32-bit words and the receive count can be a maximum of 131,076 32-bit words. The receiver and transmitter packet-length counts can be at most 1024.

SPWR_CHAN_RESET_FIFOS
Function: Resets one or both FIFOs for the referenced channel.
Input: SPWR_FIFO_SEL enumeration type
Output: None
Notes: Resets the transmitter and/or receiver data FIFO depending on the input parameter selection. Also resets the corresponding packet-length FIFO(s) and sets the programmable almost full/empty levels back to the default values for the FIFO(s) that were reset.

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

SPWR_CHAN_READ_FIFO
Function: Reads and returns one 32-bit data word from the receiver data FIFO.
Input: None
Output: FIFO word (unsigned long integer)
Notes: Used to make single-word accesses to the receive FIFO instead of using DMA.

SPWR_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. Also purges and re-enables the user interrupt semaphore used to signal user-enabled interrupt conditions.

SPWR_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.

SPWR_CHAN_FORCE_INTERRUPT
Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted when the channel master interrupt is enabled. This is used for development and to test interrupt processing.

SPWR_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 user-enabled channel interrupts. The interrupts that deal with the DMA transfers do not affect this value.

SPWR_CHAN_READ_TIME_CODE

Function: Returns the last time-code received and clears the tick received latched bit.
Input: None
Output: SPWR_CHAN_TIME_CODE structure
Notes: Returns the value of the time-code data byte last received in the Time field. The New field will be set to TRUE if the time-code has not been previously read, either by a previous instance of this call or by an ISR responding to an enabled TICK_OUT interrupt.

SPWR_CHAN_SETUP_WR_DMA_BUFFER

Function: Allocates a page-aligned memory buffer for the write DMA that is greater or equal to the requested length, creates a handle for the buffer, and catalogs the handle.
Input: The length of the memory in bytes (unsigned long integer)
Output: None
Notes: The length is rounded up to the nearest page boundary. The memory buffer handle is cataloged so that an application can map it into its memory space. The physical address of the memory is obtained and stored for the write DMA.

SPWR_CHAN_SETUP_RD_DMA_BUFFER

Function: Allocates a page-aligned memory buffer for the read DMA that is greater or equal to the requested length, creates a handle for the buffer, and catalogs the handle.
Input: The length of the memory in bytes (unsigned long integer)
Output: None
Notes: The length is rounded up to the nearest page boundary. The memory buffer handle is cataloged so that an application can map it into its memory space. The physical address of the memory is obtained and stored for the read DMA.

SPWR_CHAN_FREE_WR_DMA_BUFFER

Function: Uncatalogs then destroys the memory handle and frees the write DMA buffer memory.
Input: None
Output: The number of bytes freed (unsigned long integer)
Notes: Undoes the SPWR_CHAN_SETUP_WR_DMA_BUFFER call.

SPWR_CHAN_FREE_RD_DMA_BUFFER
Function: Uncatalogs then destroys the memory handle and frees the read DMA buffer memory.
Input: None
Output: The number of bytes freed (unsigned long integer)
Notes: Undoes the SPWR_CHAN_SETUP_RD_DMA_BUFFER call.

Write

SpaceWire transmit data is written to the device using the write command. A 32-bit word representing the number of bytes to transfer is received by the channel write thread. If the length is greater than the size of the write DMA buffer previously allocated, only the buffer length will be transferred. If the length is greater than 4,190,208 bytes (4092 pages) the buffer is partitioned into multiple list entries, each of which is no greater than 4,190,208 bytes (a hardware limit for a single list entry). Twelve byte DMA descriptors are created for each list entry comprised of the physical address of the memory, the number of bytes in that list entry, and the physical address of the next descriptor. When a next pointer is encountered with the end-of-chain bit set (bit 0), it indicates that there are no more list entries and the DMA completes when that entry has been processed. A write DMA interrupt is generated when the last list entry is completed.

Read

SpaceWire received data is read from the device using the read command. A 32-bit word representing the number of bytes to transfer is received by the channel read thread. If the length is greater than the size of the read DMA buffer previously allocated, only the buffer length will be transferred. If the length is greater than 4,190,208 bytes (4092 pages) the buffer is partitioned into multiple list entries, each of which is no greater than 4,190,208 bytes (a hardware limit for a single list entry). Twelve byte DMA descriptors are created for each list entry comprised of the physical address of the memory, the number of bytes in that list entry, and the physical address of the next descriptor with the direction bit set (bit 1) indicating a transfer from the device to memory. When a next pointer is encountered with the end-of-chain bit set (bit 0), it indicates that there are no more list entries and the DMA completes when that entry has been processed. A read DMA interrupt is generated when the last list entry is completed.

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 Street, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax

support@dyneng.com