

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

spwr_base & spwr_chan

Linux Driver Documentation

Manual Revision C

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 F

spwr_base & spwr_chan
Linux Device Drivers 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
831-457-4793 FAX

©2008-2010 by Dynamic Engineering.
Other trademarks and registered trademarks are
owned by their respective manufactures.
Manual Revision C. Revised July 29, 2010

This document contains information of proprietary interest to Dynamic Engineering. It has been supplied in confidence and the recipient, by accepting this material, agrees that the subject matter will not be copied or reproduced, in whole or in part, nor its contents revealed in any manner or to any person except to meet the purpose for which it was delivered.

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.....	4
Note.....	4
Driver Installation.....	5
Driver Startup.....	6
IO Controls.....	7
IOCTL_SPWR_BASE_GET_INFO.....	7
IOCTL_SPWR_BASE_LOAD_PLL_DATA.....	7
IOCTL_SPWR_BASE_READ_PLL_DATA.....	7
IOCTL_SPWR_BASE_SET_TIME_CONFIG.....	8
IOCTL_SPWR_BASE_GET_TIME_CONFIG.....	8
IOCTL_SPWR_CHAN_GET_INFO.....	9
IOCTL_SPWR_CHAN_SET_CONFIG.....	9
IOCTL_SPWR_CHAN_GET_CONFIG.....	9
IOCTL_SPWR_CHAN_GET_STATUS.....	9
IOCTL_SPWR_CHAN_WRITE_PACKET_LENGTH.....	9
IOCTL_SPWR_CHAN_READ_PACKET_LENGTH.....	10
IOCTL_SPWR_CHAN_SET_FIFO_LEVELS.....	10
IOCTL_SPWR_CHAN_GET_FIFO_LEVELS.....	10
IOCTL_SPWR_CHAN_GET_FIFO_COUNTS.....	10
IOCTL_SPWR_CHAN_RESET_FIFOS.....	11
IOCTL_SPWR_CHAN_WRITE_FIFO.....	11
IOCTL_SPWR_CHAN_READ_FIFO.....	11
IOCTL_SPWR_CHAN_WAIT_ON_INTERRUPT.....	11
IOCTL_SPWR_CHAN_ENABLE_INTERRUPT.....	11
IOCTL_SPWR_CHAN_DISABLE_INTERRUPT.....	11
IOCTL_SPWR_CHAN_FORCE_INTERRUPT.....	12
IOCTL_SPWR_CHAN_GET_ISR_STATUS.....	12
IOCTL_SPWR_CHAN_READ_TIME_CODE.....	12
Write.....	13
Read.....	13
Warranty and Repair.....	14
Service Policy.....	14
Out of Warranty Repairs.....	14
For Service Contact:.....	14

Introduction

The `spwr_base` and `spwr_chan` drivers are Linux device drivers for the PMC-SpaceWire, ccPMC-SpaceWire, PCI-SpaceWire and PC104p-SpaceWire from Dynamic Engineering. These SpaceWire boards have either a Spartan3-1000 Xilinx FPGA or a Spartan3-1500 Xilinx FPGA to implement the PCI interface, 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. Channels with external data FIFOs have two 1023 x 32-bit packet-length FIFOs, whereas channels with internal data FIFOs may have either two 1023 x 18-bit or two 1023 x 32-bit packet-length FIFOs depending on the amount of block RAM available in the Xilinx part used.

When the `spwr_base` module is installed, it interfaces with the PCI system configuration utility to acquire the memory and interrupt resources for each device installed. A `spwr` bus is created for each device and four channel devices are allocated. The interrupt is assigned and the address space partitioned for the base and four channel devices. When the `spwr_chan` driver is installed, it probes the `spwr` bus and finds and initializes the four channel devices created for each board. It allocates read and write list memory to hold the DMA page descriptors that are used by the hardware to perform PCI bus-master scatter-gather DMA.

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 user manual (also referred to as the hardware manual). The SpaceWire base and channel drivers were developed on Linux kernel version 2.6.32. If you are using a different version, some modification of the source code may be required.

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 user to a failed attempt to purge transmit packet data after a connection error; revised flow-control to be more efficient.



Driver Installation

The source files and makefiles for the drivers and test application are supplied in the driver archive file `space_wire2_0.tar.bz2`. Copy the directory structure to the computer where the driver is to be built.

From the top-level directory type “`KBUILD_NOPEDANTIC=1 make`” (necessary to suppress errors related to changing `CFLAGS` in Makefile) to build the object files then type “`make install`” to copy the files to the target location (must be root) (`/lib/modules/$(VERSION)/kernel/drivers/char/spwr/` for the driver and `/usr/local/bin/` for the test app). If so desired, type “`make clean`” to remove object and interim files after installation.

A `load_spwr` script is provided that will load the base driver, parse the `/proc/devices` file for the device’s major number, count the number of entries in the `/sys/bus/spwr/devices/` directory to determine the number of boards installed, create the required number of `/dev/spwr_base_x` (where `x` is the zero based board number) device nodes, load the channel driver, find that major number and create the required number of `/dev/spwr_chan_y` device nodes as well. You must have root privileges to execute this script.

`spwr_base_api.h` and `spwr_chan_api.h` are C header files that define the Application Program Interface (API) to the drivers and contain the relevant bit defines for the control/status registers of the SpaceWire devices. The `user_app` source code will provide examples of how to use the driver calls to control the hardware.

Driver Startup

Install the hardware and boot the computer. After the drivers have been installed run the load_spwr script to start the drivers and create the device interface nodes.

Handles can be opened to a specific board by using the open() function call and passing in the appropriate device names.

Below is example code for opening handles for device dev_num.

```
#typedef long          HANDLE
#define INPUT_SIZE    80

HANDLE    hspwr_base;
HANDLE    hspwr_chan[SPWR_BASE_NUM_CHANNELS];

char    Name[INPUT_SIZE];
int     i;
int     dev_num;
int     chan_num;

do
{
    printf("\nEnter target board number (starting with zero): \n");
    scanf("%d", &dev_num);
    if(dev_num < 0 || dev_num > NUM_SPWR_DEVICES)
        printf("\nTarget board number %d out of range!\n", dev_num);
}
while(dev_num < 0 || dev_num > NUM_SPWR_DEVICES);

sprintf(Name, "/dev/spwr_base_%d", dev_num);
hspwr_base = open(Name, O_RDWR);
if(hspwr_base < 2)
{
    printf("\n%s FAILED to open!\n", Name);
    return 1;
}

chan_num = dev_num * SPWR_BASE_NUM_CHANNELS

for(i = 0; i < SPWR_BASE_NUM_CHANNELS; i++)
{
    sprintf(Name, "/dev/spwr_chan_%d", chan_num + i);
    hspwr_chan[i] = open(Name, O_RDWR);
    if(hspwr_chan[i] < 2)
    {
        printf("\n%s FAILED to open!\n", Name);
        return 1;
    }
}
}
```



IO Controls

The driver uses `ioctl()` calls to configure the device and obtain status. The parameters passed to the `ioctl()` function include the handle obtained from the `open()` call, an integer command defined in the API header files and an optional parameter used to pass data in and/or out of the device. The `ioctl` commands defined for the SpaceWire base and channel devices are listed below.

The `ioctl()` calls defined for the `spwr_base` driver are described below:

IOCTL_SPWR_BASE_GET_INFO

Function: Returns the Driver version, Xilinx revision, Switch value, Instance number, and PLL ID.

Input: None

Output: `SPWR_BASE_DRIVER_DEVICE_INFO` structure

Notes: The current base driver version is 2.0. 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 `spwr_base_api.h` for the definition of `SPWR_BASE_DRIVER_DEVICE_INFO`.

IOCTL_SPWR_BASE_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.

Input: `SPWR_BASE_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.

IOCTL_SPWR_BASE_READ_PLL_DATA

Function: Returns the contents of the PLL's internal registers

Input: None

Output: `SPWR_BASE_PLL_DATA` structure

Notes: The register data is output in the `SPWR_BASE_PLL_DATA` structure in an array of 40 bytes.

IOCTL_SPWR_BASE_SET_TIME_CONFIG

Function: Sets the time-code timing and routing on the SpaceWire board.

Input: SPWR_BASE_TIME_CONFIG structure

Output: None

Notes: The master counter that controls the TICK_IN rate is clocked by the 80 MHz link clock. The Count, in the input data structure is the count at which the master counter will roll-over, increment the six-bit time-code count and issue a TICK_IN pulse. Flags specifies the two control flag bits sent in bit 6 and 7 of the time-code data byte. TimeSource is a four-value array of SPWR_TM_SRC values that determine the source of time-codes sent by each of the four 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).

IOCTL_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.

IOCTL_SPWR_CHAN_GET_INFO

Function: Returns the driver version, instance number, transmit and receive FIFO sizes, maximum packet-length of the referenced channel and a flag indicating whether the channel has enhanced control/status registers.

Input: None

Output: SPWR_CHAN_DRIVER_DEVICE_INFO structure

Notes: The current channel driver version is 2.0. The enhancements to the control/status registers are: A control bit to enable repeated transmit packet-length reuse and latched almost empty/full FIFO status bits. See the definition of SPWR_CHAN_DRIVER_DEVICE_INFO in the spwr_chan_api.h header file.

IOCTL_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 spwr_chan_api.h for the relevant channel control bit definitions. Only the bits in CHAN_CNTRL_MASK can be controlled by this call.

IOCTL_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.

IOCTL_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 IOCTL_SPWR_CHAN_READ_TIME_CODE call. In firmware version F a transmit packet-length FIFO count was added and the CHAN_STAT_TX_PKTFL status bit was replaced by the CHAN_STAT_TX_PURGERR latched status bit.

IOCTL_SPWR_CHAN_WRITE_PACKET_LENGTH

Function: Writes a transmitter packet length value to the packet length FIFO.

Input: Packet length value (unsigned short 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 17 high (bit 32 for 2 G Byte packet-length channels) causes the transmitted packet to be terminated with an EEP.

IOCTL_SPWR_CHAN_READ_PACKET_LENGTH

Function: Reads a received packet length value from the packet length FIFO.

Input: None

Output: Packet length value (unsigned short integer)

Notes: Only bits 16-0 (30-0 for 2 G Byte packet-length channels) are used for the packet-length (maximum of 128 K Bytes (2 G Bytes)). Bit 17 (bit 32 for 2 G Byte packet-length channels) 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. Reading the channel status will indicate whether a connection error was detected.

IOCTL_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 $\frac{1}{8}$ FIFO and $\frac{7}{8}$ FIFO 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, the FIFO level values are used to determine when to give priority to an output or input DMA channel that is running low on data or running out of room to store data.

IOCTL_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.

IOCTL_SPWR_CHAN_GET_FIFO_COUNTS

Function: Returns the number of data words in the transmit, receive and transmit 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. In firmware version F a transmit packet-length FIFO count was added and this field was added to the output structure. If the referenced hardware does not support the packet-length FIFO count, the TxPktCount field will be set to 0xFFFF otherwise it will reflect the number of values in the transmit packet-length FIFO (0 to 0x3FF).

IOCTL_SPWR_CHAN_RESET_FIFOS

Function: Resets one or both FIFOs for the channel

Input: SPWR_FIFO_SEL enumeration type

Output: None

Notes: Resets the transmit or receive FIFO or both 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.

IOCTL_SPWR_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_SPWR_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_SPWR_CHAN_WAIT_ON_INTERRUPT

Function: Inserts the calling process into a wait queue until an interrupt occurs.

Input: Time-out value in jiffies (unsigned long integer)

Output: None

Notes: This call is made in the user interrupt service routine to allow user-specified interrupt handlers for enabled interrupt conditions. The input parameter is a time-out value that causes the call to abort if the interrupt doesn't occur within the specified time. If the timeout is zero, the call will wait indefinitely for the interrupt to occur. The DMA interrupts do not use this mechanism; they are controlled automatically by the driver.

IOCTL_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.

IOCTL_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.



IOCTL_SPWR_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_SPWR_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.

Input: None

Output: SPWR_CHAN_ISR_STAT structure

Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled channel user interrupts and a flag to indicate whether the interrupt timed-out before it was seen. The interrupts that deal with the DMA transfers do not affect this value.

IOCTL_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 six-bit time-code count (Time) and the two-bit flag value (Flags) last received. The New field will be set to TRUE if the time-code has not been previously read by either a previous instance of this call or by an ISR responding to an enabled TICK_OUT interrupt.

Write

SpaceWire transmit data is written to the device using the write command. A handle to the device, a pointer to a pre-allocated buffer that contains the data to write and an unsigned long integer that represents the amount of data to write in bytes are passed to the write call. The driver will obtain physical addresses to the pages containing the data and will set-up a list of page descriptors in its list memory. The physical address of the first list entry is written to the board, which performs a bus-master scatter-gather DMA to transfer the data.

Read

SpaceWire received data is read from the device using the read command. A handle to the device, a pointer to a pre-allocated buffer that will contain the data read and an unsigned long integer that represents the amount of data to read in bytes are passed to the read call. The driver will obtain physical addresses to the buffer memory pages and will set-up a list of page descriptors in its list memory. The physical address of the first list entry is written to the board, which performs a bus-master scatter-gather DMA to transfer the data.

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 St. Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax
support@dyneng.com

All information provided is Copyright Dynamic Engineering.

