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

PCIeBiSerialDb37-RTN8
Linux Driver

rtn8

Linux Driver Documentation

Linux kernel version 2.6.37

Revision A
Corresponding Hardware: Revision A
10-2009-0401
Corresponding Firmware: Revision B

rtn8
Linux Device Driver for the
PCIe-BiSerial3-DB37-RTN8
Bidirectional LVDS PCIe Board

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

Manual Revision A. Revised June 23, 2011

Table of Contents

Introduction

The rtn8 is a Linux device driver for the PCIe-BiSerial3-DB37-RTN8 from Dynamic Engineering. The BiSerial3-DB37-RTN8 board has a Spartan3-2000 Xilinx FPGA to implement the PCI interface, internal FIFOs and protocol control and status for a single I/O channel. There is also a programmable PLL clock output to generate the transmitter I/O reference frequency. The Transmitter is supported with a combination of 4Kx32 and 8Kx32 internal FIFOs for a total of 12Kx32. Using DMA transfers the transmit side can provide a continuous flow of data at the output or data bursts depending on the programmed byte wide data speed and the system DMA capabilities. The Receiver has a 4Kx8 first stage internal FIFO followed by two 128Kx32 external FIFOs and a 1Kx32 internal FIFO. The 1Kx32 FIFO serves as the output DMA source FIFO and the two 128K x32 FIFOs as the bulk storage for the receiver.

When the rtn8_base module is loaded, it interfaces with the PCI bus sub-system to acquire the memory and interrupt resources for each device installed. The interrupt is assigned and the address space mapped. Read and write list memory is allocated to hold the DMA page descriptors that are used by the hardware to perform scatter-gather bus-master DMA.

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 PCIe-BiSerial3-DB37-RTN8 user manual (also referred to as the hardware manual). The rtn8 driver was developed on Linux kernel version 2.6.37. If you are using a different version, some modification of the source code may be required.

Driver Installation

The source files and Make files for the driver and test application are supplied in the driver archive file rtn8.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” to build the object files. Then type “sudo make install” to copy the files to the target locations (must have root privilege for this to succeed), (/lib/modules/$(VERSION)/kernel/drivers/char/rtn8/ for the driver, /usr/local/bin/ for the test app and rtn8_mknd, and /etc/udev/rules.d/ for the rtn8-devnodes.rules file). After installation, you can type “make clean” to remove object files and executables if desired.

After the driver is installed type (sudo) depmod to register the driver modules with the kernel (must have root privileges).

The Application Program Interface (API) for the drivers and relevant structures and bit defines for control and status on the PCIe-BiSerial3-DB37-RTN8 are defined in the C header file rtn8_api.h. The user_app source code will provide examples of how to use the driver calls to control the hardware. PLL configuration files (rtn8.jed and rtn8_50.jed) are provided to set the PLL Clock A output to 80 and 50 MHz respectively for use with the test application software.

The .jed files will be copied into the user’s home directory. If the test software is run from this directory with “/usr/local/bin/rtn8_test”, then the .jed files can be accessed from the test prompt using just the file name without extra path information.

Driver Startup

Install the hardware and boot the computer. After the driver has been installed, the driver should load automatically when the system recognizes the PCI signature.

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

Below is example code for opening a handle for device `dev_num`.

```
#typedef long  HANDLE
#define  INPUT_SIZE  80

HANDLE   hrtn8;

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

do
{
   printf("\nEnter target board number (starting with zero): \n");
   scanf("%d", &dev_num);
   if(dev_num < 0 || dev_num > NUM_RTN8_DEVICES)
      printf("\nTarget board number %d out of range!\n", dev_num);
}
while(dev_num < 0 || dev_num > NUM_RTN8_DEVICES);

sprintf(Name, "/dev/rtn8_%d", dev_num);
hrtn8 = open(Name , O_RDWR);
if(hrtn8_base < 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 number 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 PCIe-BiSerial3-DB37-RTN8 are listed below.

IOCTL_RTN8_GET_INFO

Function: Returns the Driver version, Xilinx design and revision, Switch value, Instance number, and PLL device ID.
Input: None
Output: RTN8_DRIVER_DEVICE_INFO structure
Notes: Switch value is the configuration of the on-board dip-switch that has been set by the user (see the board silk screen for bit position and polarity). The PLL device ID is the device address of the PLL device. This value, which is set at the factory, is usually 0x69 but may alternatively be 0x6A. See rtn8_api.h for the definition of RTN8_DRIVER_DEVICE_INFO.

IOCTL_RTN8_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.
Input: RTN8_PLL_DATA structure
Output: None
Notes: The PLL internal register data is loaded into the RTN8_PLL_DATA structure in an array of 40 bytes. Appropriate values for this array can be derived from .jed files created by the CyberClock utility from Cypress Semiconductor.

IOCTL_RTN8_READ_PLL_DATA

Function: Returns the contents of the PLL's internal registers
Input: None
Output: RTN8_PLL_DATA structure
Notes: The register data is output in the RTN8_PLL_DATA structure in an array of 40 bytes.

IOCTL_RTN8_SET_CONFIG

Function: Sets the configuration values.
Input: Control configuration (RTN8_CONFIG)
Output: None
Notes: See rtn8_api.h for the definition of RTN8_CONFIG.

IOCTL_RTN8_GET_CONFIG

Function: Returns the control configuration.
Input: None
Output: Control configuration (RTN8_STATE)
Notes: The state of the master interrupt enable and read and write DMA enables is read by this call in addition to the values set by the SET_CONFIG call above. See rtn8_api.h for the definition of RTN8_STATE.

IOCTL_RTN8_GET_STATUS
Function: Returns the status value and clears the latched bits.
Input: None
Output: Value of the status register (unsigned integer)
Notes: All the valid status bits will be returned and the latched bits in CHAN_STAT_LATCH_MASK will be cleared, if set, after the status is read.

IOCTL_RTN8_SET_FIFO_LEVELS
Function: Sets the transmitter almost empty and receiver almost full levels.
Input: RTN8_FIFO_LEVELS structure
Output: None
Notes: These values are initialized to the default values 16 transmit FIFO words and 265,184 receive FIFO words respectively when the driver initializes. The FIFO counts are compared to these levels to determine the value of the STAT_TX_FFALL_AMT and STAT_RX_FFALL_AFL status bits. Also, if TxUrgent and/or RxUrgent are TRUE, these levels are used to determine what level to activate priority for an input or output DMA.

IOCTL_RTN8_GET_FIFO_LEVELS
Function: Returns the transmitter almost empty and receiver almost full levels.
Input: None
Output: RTN8_FIFO_LEVELS structure
Notes:

IOCTL_RTN8_GET_FIFO_COUNTS
Function: Returns the number of data words in the transmit and receive FIFOs.
Input: None
Output: RTN8_FIFO_COUNTS structure
Notes: There are four pipe-line latches in the receive FIFO data-path that are counted in the receive FIFO count. The transmit count can be a maximum of 12,286 32-bit words, and the receive count can be a maximum of 265,215 32-bit words.

IOCTL_RTN8_RESET_FIFOS
Function: Resets the transmit FIFOs, the receive FIFOs or both transmit and receive FIFOs.
Input: RTN8_FIFO_SEL enumeration type
Output: None
Notes: There are three values for RTN8_FIFO_SEL, RTN8_TX, RTN8_RX, or RTN8_BOTH. For each selection the corresponding data FIFOs will be reset.

IOCTL_RTN8_WRITE_FIFO
Function: Writes a 32-bit data-word to the transmit FIFO.
Input: FIFO word (unsigned integer)
Output: None
Notes: Used to make single-word accesses to the transmit FIFO instead of using DMA.

IOCTL_RTN8_READ_FIFO
Function: Returns a 32-bit data word from the receive FIFO.
Input: None
Output: FIFO word (unsigned integer)
Notes: Used to make single-word accesses to the receive FIFO instead of using DMA.

IOCTL_RTN8_SET_TX_CONFIG
Function: Sets the transmitter configuration parameters.
Input: Transmitter control configuration (RTN8_TX_CONFIG)
Output: None
Notes: See rtn8_api.h for the definition of RTN8_TX_CONFIG.

IOCTL_RTN8_GET_TX_CONFIG
Function: Returns the transmitter configuration parameters.
Input: None
Output: Transmitter control configuration (RTN8_TX_CONFIG)
Notes: See rtn8_api.h for the definition of RTN8_TX_CONFIG.

IOCTL_RTN8_SET_RX_CONFIG
Function: Sets the receiver configuration parameters.
Input: Receiver control configuration (RTN8_RX_CONFIG)
Output: None
Notes: See rtn8_api.h for the definition of RTN8_RX_CONFIG.

IOCTL_RTN8_GET_RX_CONFIG
Function: Returns receiver configuration parameters.
Input: None
Output: Receiver control configuration (RTN8_RX_CONFIG)
Notes: See rtn8_api.h for the definition of RTN8_RX_CONFIG.

IOCTL_RTN8_WAIT_ON_INTERRUPT
Function: Inserts the calling process into the interrupt wait queue until an interrupt occurs.
Input: Time-out value in jiffies (unsigned integer)
Output: None
Notes: This call is used to implement a user defined interrupt service routine. It will return when an interrupt occurs or when the delay time specified expires. If the delay is set to zero, the call will wait indefinitely. The delay time is dependent on the platform setting for jiffy, which could be anything from 10 milliseconds to less than 1 millisecond. The DMA interrupts do not use this mechanism; they are controlled automatically by the driver.

IOCTL_RTN8_ENABLE_INTERRUPT

Function: Enables the 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 to re-enable interrupts after each interrupt occurs.

IOCTL_RTN8_DISABLE_INTERRUPT

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

IOCTL_RTN8_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 master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_RTN8_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: RTN8_ISR_STAT structure
Notes: The Status field is the status that was read in the last interrupt service routine that serviced an enabled user interrupt. The TimedOut field of the structure will be true if the time-out value set in IOCTL_RTN8_WAIT_ON_INTERRUPT was exceeded. The interrupts that deal with the DMA transfers do not affect this value.

Write

Rtn8 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 integer that represents the number of bytes of data to write 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

Rtn8 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 integer that represents the number of bytes of data to read 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 merchandise ability 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 $130. 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