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-LM9
Linux Driver

lm9_base
&
lm9_chan

Linux Driver Documentation

Linux kernel version 2.6.18

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

lm9_base & lm9_chan
Linux Device Drivers for the
PCIe-BiSerial3-DB37-LM9
Single Channel PCIe Board
LM9 Interface

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

Manual Revision A. Revised December 4, 2009

Table of Contents

Introduction 4
Note 4
Driver Installation 5
Driver Startup 6
IO Controls 7
IOCTL_LM9_BASE_GET_INFO 7
IOCTL_LM9_BASE_LOAD_PLL_DATA 7
IOCTL_LM9_BASE_READ_PLL_DATA 7
IOCTL_LM9_BASE_SET_GPIO_TERM 7
IOCTL_LM9_BASE_GET_GPIO_TERM 8
IOCTL_LM9_BASE_SET_GPIO_DIR 8
IOCTL_LM9_BASE_GET_GPIO_DIR 8
IOCTL_LM9_BASE_SET_GPIO_DATA 8
IOCTL_LM9_BASE_GET_GPIO_DATA 8
IOCTL_LM9_BASE_READ_GPIO 8
IOCTL_LM9_CHAN_GET_INFO 8
IOCTL_LM9_CHAN_SET_CONFIG 9
IOCTL_LM9_CHAN_GET_CONFIG 9
IOCTL_LM9_CHAN_GET_STATUS 9
IOCTL_LM9_CHAN_SET_FIFO_LEVELS 9
IOCTL_LM9_CHAN_GET_FIFO_LEVELS 9
IOCTL_LM9_CHAN_GET_FIFO_COUNTS 10
IOCTL_LM9_CHAN_RESET_FIFOS 10
IOCTL_LM9_CHAN_WRITE_FIFO 10
IOCTL_LM9_CHAN_READ_FIFO 10
IOCTL_LM9_CHAN_SET_TX_CONFIG 10
IOCTL_LM9_CHAN_GET_TX_CONFIG 10
IOCTL_LM9_CHAN_SET_RX_CONFIG 11
IOCTL_LM9_CHAN_GET_RX_CONFIG 11
IOCTL_LM9_CHAN_SET_TX_COUNT 11
IOCTL_LM9_CHAN_GET_TX_COUNT 11
IOCTL_LM9_CHAN_SET_RX_COUNT 11
IOCTL_LM9_CHAN_GET_RX_COUNT 11
IOCTL_LM9_CHAN_WRITE_TX_PACKET_FIFO 11
IOCTL_LM9_CHAN_READ_TX_PACKET_FIFO 12
IOCTL_LM9_CHAN_READ_RX_PACKET_FIFO 12
IOCTL_LM9_CHAN_SET_RX_TIMEOUT 12
IOCTL_LM9_CHAN_GET_RX_TIMEOUT 12
IOCTL_LM9_CHAN_WAIT_ON_INTERRUPT 12
IOCTL_LM9_CHAN_ENABLE_INTERRUPT 13
IOCTL_LM9_CHAN_DISABLE_INTERRUPT 13
IOCTL_LM9_CHAN_FORCE_INTERRUPT 13
IOCTL_LM9_CHAN_GET_ISR_STATUS 13
Write 14
Read 14
Warranty and Repair 14
Service Policy 15
Out of Warranty Repairs 15
For Service Contact: 15

Introduction

The lm9_base and lm9_chan drivers are Linux device drivers for the PCIe-BiSerial3-DB37-LM9 from Dynamic Engineering. The BiSerial3-DB37-LM9 board has a Spartan3-2000 Xilinx FPGA to implement the PCI interface, FIFOs and protocol control and status for a single I/O channel. There is also a programmable PLL clock output for the transmit reference frequency when internal timing is selected. The I/O channel has a 4k x 32-bit receive and transmit data FIFO and a 2k x 32-bit receive and transmit packet-length FIFO.

When the lm9_base module is loaded, it interfaces with the PCI bus sub-system to acquire the memory and interrupt resources for each device installed. An lm9 bus is created for each device and a channel device is allocated. The interrupt is assigned and the address space partitioned for the channel device. When the lm9_chan driver is installed, it probes the lm9 bus and finds and initializes the channel device for each board. It allocates read and write list memory to hold the DMA page descriptors that are used by the hardware to perform scatter-gather 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-LM9 user manual (also referred to as the hardware manual). The LM9 base and channel drivers were developed on Linux kernel version 2.6.18. If you are using a different version, some modification of the source code might be required.

Driver Installation

The source files and Make files for the drivers and test application are supplied in the driver archive file lm9.tar.bz2. Copy the directory structure to the computer where the driver is to be built. From the top-level directory type “make” to build the object files then type “make install” to copy the files to the target location (must be root for this to succeed) (/lib/modules/$(VERSION)/kernel/drivers/misc/ for the driver and /usr/local/bin/ for the test app). After installation, you can type “make clean” to remove object files and executables.

A load_lm9 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/lm9/devices/ directory to determine the number of boards installed, create the required number of /dev/lm9_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/lm9_chan_***y*** device nodes as well.

The Application Program Interface (API) for the drivers and relevant structures and bit defines for control and status on the PCIe-BiSerial3-DB37-LM9 are defined in the C header files lm9_base_api.h and lm9_chan_api.h. The user_app source code will provide examples of how to use the driver calls to control the hardware. A PLL configuration file (test.jed) is provided that sets the PLL clock A output to 1 MHz for use with the test application software.

Driver Startup

Install the hardware and boot the computer. After the drivers have been installed run the load_lm9 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   hlm9_base;
HANDLE   hlm9_chan[LM9_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_LM9_DEVICES)
      printf("\nTarget board number %d out of range!\n", dev_num);
}
while(dev_num < 0 || dev_num > NUM_LM9_DEVICES);

sprintf(Name, "/dev/lm9_base_%d", dev_num);
hlm9_base = open(Name , O_RDWR);
if(hlm9_base < 2)
{
   printf("\n%s FAILED to open!\n", Name);
   return 1;
}

chan_num = dev_num * LM9_BASE_NUM_CHANNELS

for(i = 0; i < LM9_BASE_NUM_CHANNELS; i++)
{
   sprintf(Name, "/dev/lm9_chan_%d", chan_num + i);
   hlm9_chan[i] = open(Name, O_RDWR);
   if(hlm9_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 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-LM9 are listed below.

IOCTL_LM9_BASE_GET_INFO

Function: Returns the Driver version, Xilinx revision, Switch value, Instance number, and PLL device ID.
Input: None
Output: LM9_BASE_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 lm9_base_api.h for the definition of LM9_BASE_DRIVER_DEVICE_INFO.

IOCTL_LM9_BASE_LOAD_PLL_DATA

Function: Loads the internal registers of the PLL.
Input: LM9_BASE_PLL_DATA structure
Output: None
Notes: The PLL internal register data is loaded into the LM9_BASE_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_LM9_BASE_READ_PLL_DATA

Function: Returns the contents of the PLL's internal registers
Input: None
Output: LM9_BASE_PLL_DATA structure
Notes: The register data is output in the LM9_BASE_PLL_DATA structure in an array of 40 bytes.

IOCTL_LM9_BASE_SET_GPIO_TERM

Function: Enables the termination resistors on selected GPIO lines.
Input: Value of termination control register (unsigned long integer)
Output: None
Notes: Writing a one to a specific bit enables the termination on the corresponding GPIO line, writing a zero disables the termination. Bits 0 to 11 control GPIO lines 6 to 17. The lower six lines (0 – 5) are already configured and can't be changed.

IOCTL_LM9_BASE_GET_GPIO_TERM
Function: Returns the value of the GPIO termination control register.
Input: None
Output: Value of termination control register (unsigned long integer)
Notes:

IOCTL_LM9_BASE_SET_GPIO_DIR
Function: Controls the direction of selected GPIO lines.
Input: Value of direction control register (unsigned long integer)
Output: None
Notes: Writing a one to a specific bit configures the corresponding GPIO line as an output, writing a zero configures the line as an input. Bits 0 to 11 control GPIO lines 6 to 17. The lower six lines (0 – 5) are already configured and can't be changed.

IOCTL_LM9_BASE_GET_GPIO_DIR
Function: Returns the value of the GPIO direction control register.
Input: None
Output: Value of direction control register (unsigned long integer)
Notes:

IOCTL_LM9_BASE_SET_GPIO_DATA
Function: Writes a value to the GPIO data register.
Input: Value of data register (unsigned long integer)
Output: None
Notes: The data in this register will be driven onto GPIO lines 6 to 17 if the corresponding line is configured as an output. If a line is configured as an input, data can still be written and read, but it will not affect the level of the external GPIO line.

IOCTL_LM9_BASE_GET_GPIO_DATA
Function: Returns the contents of the GPIO data register.
Input: None
Output: Value of data register (unsigned long integer)
Notes:

IOCTL_LM9_BASE_READ_GPIO
Function: Reads the current state of the external GPIO lines.
Input: None
Output: Value of GPIO lines (unsigned long integer)
Notes: The external GPIO lines 6 to 17 are read by this call.

IOCTL_LM9_CHAN_GET_INFO
Function: Returns the driver version and instance number of the referenced channel.
Input: None
Output: LM9_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of LM9_CHAN_DRIVER_DEVICE_INFO in the lm9_chan_api.h header file.

IOCTL_LM9_CHAN_SET_CONFIG
Function: Sets channel configuration values.
Input: Channel control configuration (LM9_CHAN_CONFIG)
Output: None
Notes: See lm9_chan_api.h for the definition of LM9_CHAN_CONFIG.

IOCTL_LM9_CHAN_GET_CONFIG
Function: Returns the channel's control configuration.
Input: None
Output: Channel control configuration (LM9_CHAN_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 lm9_chan_api.h for the definition of LM9_CHAN_STATE.

IOCTL_LM9_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 set, after the status is read.

IOCTL_LM9_CHAN_SET_FIFO_LEVELS
Function: Sets the transmitter almost empty and receiver almost full levels for the channel.
Input: LM9_CHAN_FIFO_LEVELS structure
Output: None
Notes: These values are initialized to the default values 16 transmit FIFO words and 4084 receive FIFO words respectively when the driver initializes. The FIFO counts are compared to these levels to determine the value of the CHAN_STAT_TX_FF_AMT and CHAN_STAT_RX_FF_AFL status bits. Also, if TxUrgent and/or RxUrgent are TRUE, these levels are used to determine when to give priority to an input or output DMA channel.

IOCTL_LM9_CHAN_GET_FIFO_LEVELS
Function: Returns the transmitter almost empty and receiver almost full levels for the channel.
Input: None
Output: LM9_CHAN_FIFO_LEVELS structure
Notes:

IOCTL_LM9_CHAN_GET_FIFO_COUNTS
Function: Returns the number of data words in the transmit and receive FIFOs.
Input: None
Output: LM9_CHAN_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 will be a maximum of 4095 32-bit words, but the receive count can be a maximum of 4099 32-bit words.

IOCTL_LM9_CHAN_RESET_FIFOS
Function: Resets the transmit data and packet-length FIFOs, the receive data and packet-length FIFOs or both transmit and receive data and packet-length FIFOs for the channel
Input: LM9_CHAN_FIFO_SEL enumeration type
Output: None
Notes: There are three values for LM9_CHAN_FIFO_SEL, LM9_TX, LM9_RX, or LM9_BOTH. For each selection the corresponding data and packet-length FIFOs will both be reset.

IOCTL_LM9_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_LM9_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_LM9_CHAN_SET_TX_CONFIG
Function: Sets the channel transmitter configuration parameters.
Input: Transmitter control configuration (LM9_CHAN_TX_CONFIG)
Output: None
Notes: See lm9_chan_api.h for the definition of LM9_CHAN_TX_CONFIG.

IOCTL_LM9_CHAN_GET_TX_CONFIG
Function: Returns the channel transmitter configuration parameters.
Input: None
Output: Transmitter control configuration (LM9_CHAN_TX_CONFIG)
Notes: See lm9_chan_api.h for the definition of LM9_CHAN_TX_CONFIG.

IOCTL_LM9_CHAN_SET_RX_CONFIG
Function: Sets the channel receiver configuration parameters.
Input: Receiver control configuration (LM9_CHAN_RX_CONFIG)
Output: None
Notes: See lm9_chan_api.h for the definition of LM9_CHAN_RX_CONFIG.

IOCTL_LM9_CHAN_GET_RX_CONFIG
Function: Returns channel receiver configuration parameters.
Input: None
Output: Receiver control configuration (LM9_CHAN_RX_CONFIG)
Notes: See lm9_chan_api.h for the definition of LM9_CHAN_RX_CONFIG.

IOCTL_LM9_CHAN_SET_TX_COUNT
Function: Sets the transmit packet byte-count register value.
Input: Value of the packet-length register (unsigned long integer)
Output: None
Notes: When the RegPacket field in the TX configuration is true, this value is used to determine the transmit packet-length instead of a value read from the packet-length FIFO.

IOCTL_LM9_CHAN_GET_TX_COUNT
Function: Returns the transmit packet byte-count register value.
Input: None
Output: Value of the packet-length register (unsigned long integer)
Notes: Returns the value set in the previous call.

IOCTL_LM9_CHAN_SET_RX_COUNT
Function: Sets the expected receive packet byte-count register value.
Input: Value of the packet-length register (unsigned long integer)
Output: None
Notes: Specifies the number of bytes in a received packet.

IOCTL_LM9_CHAN_GET_RX_COUNT
Function: Returns the expected receive packet byte-count register value.
Input: None
Output: Value of the channel register (unsigned long integer)
Notes: Returns the value set in the previous call.

IOCTL_LM9_CHAN_WRITE_TX_PACKET_FIFO
Function: Writes a value to the transmit packet-length FIFO.
Input: Transmit packet-length (unsigned long integer)
Output: None
Notes: When the packet-length FIFO path is selected this FIFO is read to determine the length of each transmit data packet.

IOCTL_LM9_CHAN_READ_TX_PACKET_FIFO
Function: Reads a value from the transmit packet-length FIFO.
Input: None
Output: Transmit packet-length (unsigned long integer)
Notes: This call is used for testing the packet-length FIFO.

IOCTL_LM9_CHAN_READ_RX_PACKET_FIFO
Function: Reads a value from the receive packet-length FIFO.
Input: None
Output: Receive packet-length (unsigned long integer)
Notes: The receive packet-length FIFO is read to determine the amount of data received in a packet.

IOCTL_LM9_CHAN_SET_RX_TIMEOUT
Function: Writes a value to the receiver timeout count register.
Input: Timeout count (unsigned long integer)
Output: None
Notes: When the receiver timeout option is enabled, the value in this register is used to determine when a gap in the data has exceeded the expected value and the current packet is terminated. The timeout counter uses the 33 MHz PCI clock.

IOCTL_LM9_CHAN_GET_RX_TIMEOUT
Function: Returns the value of the receiver timeout count register.
Input: None
Output: Timeout count (unsigned long integer)
Notes: Returns the value set in the previous call.

IOCTL_LM9_CHAN_WAIT_ON_INTERRUPT
Function: Inserts the calling process into the interrupt wait queue until an interrupt occurs.
Input: Time-out value in jiffies (unsigned long 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_LM9_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 to re-enable interrupts after an interrupt occurs.

IOCTL_LM9_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_LM9_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_LM9_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: LM9_CHAN_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_LM9_CHAN_WAIT_ON_INTERRUPT was exceeded. The interrupts that deal with the DMA transfers do not affect this value.

Write

Lm9 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 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

Lm9 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 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