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

IP-BiSerial-Q1

Driver Documentation

Linux Version

Revision A
Corresponding Hardware: Revision A
10-1997-0202

IP-BiSerial-Q1
Bi-directional Serial Data Interface
IP Module

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

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.

This product has been designed to operate with IP Module carriers and compatible user-provided equipment. Connection of incompatible hardware is likely to cause serious damage.

Manual Revision A. Revised July 21, 2003.

Table of Contents

Introduction

The lpbisq1 driver is a modular Linux driver for the IP BiSerial-Q1 board from Dynamic Engineering. Each IP BiSerial-Q1 board transmits and receives one channel of serial data with EIA-RS-485 differential drivers and receivers. When the driver is loaded, it reads the /proc/bus/p5ip/ip_devices file created by the Pci5ip driver to obtain its memory and interrupt resources. The driver remaps the slot control, IO, Mem, and Int memory spaces and reports these and other parameters in the /proc/bus/ipbisq1/devs file. A device node must be created for each board in the /dev directory to access the hardware. A separate handle references each board. IO Control calls, ioctl()'s are used to configure the hardware and read() and write() calls are used to transfer data to and from the device over the IP bus.

Note

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. For more detailed information on the hardware implementation, refer to the IP BiSerial-Q1 device user manual.

Driver Installation

A load_ip script is provided with the carrier driver that loads the carrier driver, creates the device node(s) and loads the appropriate IP drivers. The script parses the /proc/devices file for the driver major number and creates the required number of /dev/lpbisq1_*x* (where *x* is the zero based board number) device nodes,

The files provided in the driver drop include the driver object file (ipbisq1.o), two header files (ipbisq1def.h and ipbisq1API.h), a Makefile to install the driver object file and the test object files. Several test source code files as well as five compiled test executables for the five modules on a PCI5IP carrier are provided to run verification tests and to use as an example for user application code.

Copy the *.c and *.h files to a directory where the application code will be built, copy the other files to a temporary directory and run *make install* to install the driver object file in the

/lib/module/*version*/kernel/driver/add_on/i_pack/ directory and the test object files in the /usr/local/bin/ directory.

Driver Startup

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

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 0.

```
long int       hIpbisq1; // Device handle
char           Name[INPUT_SIZE];

sprintf(Name , "/dev/Ipbisq1_0");
hIpbisq1 = open(Name, O_RDWR);
if(hIpbisq1 < 2)
{
   printf("\n%s FAILED to open!\n", Name);
   return ERROR;
}
```

Please note: The slots on the PCI5IP are enumerated in the following order: A –> C –> E –> B –> D and that installed devices of the same IP type will be numbered from 0 on up according to the order they occur in this list.

IO Controls

The driver uses ioctl() calls to configure the device. The parameters passed to the ioctl() function include the handle obtained from the open() call, an integer command defined in the ipbisq1API.h file and an optional parameter used to pass data in and/or out of the device. The ioctl commands defined for the IP BiSerial-Q1 are listed below.

IOCTL_IPBISQ1_GET_TX_DATA

Function: Reads one word from the Tx FIFO.
Input: none
Output: unsigned short
Notes: This call is used for transmit FIFO write/read-back testing and is not needed for normal operation.

IOCTL_IPBISQ1_PUT_RX_DATA

Function: Writes one word to the Rx FIFO.
Input: unsigned short
Output: none
Notes: This call is used for receive FIFO write/read-back testing and is not needed for normal operation.

IOCTL_IPBISQ1_GET_STATUS0

Function: Returns the FIFO status.
Input: none
Output: unsigned short
Notes: Returns Status information for a given board obtained from the IPBISQ1_STATUS_0 register. This consists of the FIFO flags indicating the amount of data in the Tx and Rx FIFOs. See the bit definitions in the IPBISQ1.h header file for more information.

IOCTL_IPBISQ1_GET_STATUS1

Function: Returns the interrupt/error status.
Input: none
Output: unsigned short
Notes: Returns Status information for a given board obtained from the IPBISQ1_STATUS_1 register. This consists of latched error and interrupt status bits indicating the cause of an error or interrupt. After the status is read, a value is written back to this register to clear only the specific latched bits that were read. This will insure that no interrupt cause is missed due to being asserted between the read and write cycles. See the bit definitions in the ipbisq1def.h header file for more information.

IOCTL_IPBISQ1_SET_IP_CONTROL

Function: Sets the configuration of the board slot.
Input: unsigned long
Output: none
Notes: Controls the IP clock speed and interrupt enables for the IP slot that the board occupies. See the bit definitions in the ipbisq1def.h header file for more information.

IOCTL_IPBISQ1_GET_ IP_CONTROL

Function: Returns the configuration of the board slot.
Input: none
Output: unsigned long
Notes: Returns the slot configuration register value. See the bit definitions in the ipbisq1def.h header file for more information.

IOCTL_IPBISQ1_SET_TX_CONFIG

Function: Sets the transmitter configuration bits.
Input: unsigned short
Output: none
Notes: Controls the Tx and Tx FIFO almost empty interrupt enables, the Tx 16/32 bit mode, and the Tx parity sense.

IOCTL_IPBISQ1_GET_ TX_CONFIG

Function: Returns the transmitter configuration bits.
Input: none
Output: unsigned short
Notes: Returns the state of the bits listed above as well as the Tx start bit.

IOCTL_IPBISQ1_SET_ RX_CONFIG

Function: Sets the receiver configuration bits.
Input: unsigned short
Output: none
Notes: Controls the Rx and Rx FIFO almost full interrupt enables, the Rx 16/32 bit mode, the Rx auto-clear enable, the Rx parity sense and the Rx TTL/RS422 input enable.

IOCTL_IPBISQ1_GET_ RX_CONFIG

Function: Returns the receiver configuration bits.
Input: none
Output: unsigned short
Notes: Returns the state of the bits listed above as well as the Rx start bit.

IOCTL_IPBISQ1_SET_ CLOCK_CONFIG

Function: Sets the transmit clock configuration.
Input: unsigned short
Output: none
Notes: Controls the clock divisor and whether the input clock or the divided clock is put out.

IOCTL_IPBISQ1_GET_ CLOCK_CONFIG

Function: Returns the transmit clock configuration.
Input: none
Output: unsigned short

Notes: Returns the clock divisor and the output clock select control bits.

IOCTL_IPBISQ1_SET_FIFO_LEVELS

Function: Sets the Tx FIFO almost empty and Rx FIFO almost full levels.
Input: FIFO_LEVELS struct
Output: none
Notes: Sets the almost full level for the Rx FIFO; this value is the number of words below full that the PAF flag becomes asserted. Sets the almost empty level for the Tx FIFO; this value is the number of words above empty for which the PAE flag is asserted. The Tx and Rx state machines are stopped by this command, since normal FIFO data accesses are disabled when the FIFO level registers are accessed. Values are checked to not exceed the FIFO sizes.

IOCTL_IPBISQ1_GET_FIFO_LEVELS

Function: Returns Rx almost full and Tx almost empty FIFO levels.
Input: none
Output: FIFO_LEVELS struct
Notes: Returns the Rx almost full and the Tx almost empty FIFO levels. The Tx and Rx state machines are stopped by this command, since normal FIFO data accesses are disabled when the FIFO level registers are accessed.

IOCTL_IPBISQ1_RESET_FIFOS

Function: Resets the transmit and receive FIFOs.
Input: none
Output: none
Notes: Resets both FIFOs. This will clear all data and reset the almost full and empty levels to their default values.

IOCTL_IPBISQ1_START_TX

Function: Starts the Tx state machine.
Input: none
Output: none

Notes: Provided data has been loaded into the FIFO, this command will send transmit data.

IOCTL_IPBISQ1_STOP_TX

Function: Stops the Tx state machine.
Input: none
Output: none
Notes: This command will prevent Tx data from being sent. Used mainly to stop the transmitter when it has been started with no data loaded.

IOCTL_IPBISQ1_START_RX

Function: Starts the Rx state machine.
Input: none
Output: none
Notes: This command will enable the receiver to start looking for data.

IOCTL_IPBISQ1_STOP_RX

Function: Stops the Rx state machine.
Input: none
Output: none
Notes: This command will abort a reception. Used mainly to stop the Rx state machine when it is waiting for data, but no data is being received.

IOCTL_IPBISQ1_WAIT_ON_INTERRUPT

Function: Causes an entry to be placed in the interrupt wait queue.
Input: delay value to wait
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.

IOCTL_IPBISQ1_ENABLE_INTERRUPT

Function: Sets the master interrupt enable to true.
Input: none
Output: none
Notes: Sets the master interrupt enable, leaving all other bit values in the control2 register the same. Also checks the state of the IP slot control register interrupt 0 enable bit in the saved configuration, and sets it if needed. This IOCTL is used in the user interrupt processing function to re-enable the interrupts after they were disabled in the driver interrupt service routine. This allows that function to enable the interrupts without knowing the particulars of the other configuration bits.

IOCTL_IPBISQ1_DISABLE_INTERRUPT

Function: Sets the master interrupt enable to true.
Input: none
Output: none
Notes: Clears the master interrupt enable, leaving all other bit values in the interrupt enable configuration register the same. This IOCTL is used when interrupt processing is no longer desired.

IOCTL_IPBISQ1_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: none
Output: none
Notes: Causes an interrupt to be asserted on the IP bus if the master interrupt enable is set. This IOCTL is used for development, to test interrupt processing.

IOCTL_IPBISQ1_SET_VECTOR

Function: Sets the value of the interrupt vector.
Input: unsigned char
Output: none

Notes: This value is driven onto the low byte of the data bus in response to an INT_SEL strobe, which is used in vectored interrupt cycles. This value will be read in the interrupt service routine and stored for future reference.

IOCTL_IPBISQ1_GET_VECTOR

Function: Returns the current interrupt vector value.
Input: none
Output: unsigned char
Notes:

IOCTL_IPBISQ1_SET_DATA_DELAY

Function: Sets the data delay register value.
Input: unsigned short
Output: none
Notes: This value is used to distinguish between the inter-bit gap and the inter-word gap.

IOCTL_IPBISQ1_GET_DATA_DELAY

Function: Returns the value in the data delay register.
Input: none
Output: unsigned short
Notes:

IOCTL_IPBISQ1_GET_INT_STATUS

Function: Returns the interrupt status and interrupt vector.
Input: none
Output: INT_STAT struct
Notes: Returns the interrupt vector and the contents of the interrupt status register that were read in the last ISR call. These values are returned in the INT_STAT structure as well as a BOOLEAN field that is true if the wait queue was inactive during that interrupt, which usually means that the interrupt timed-out. See the bit definitions in the ipbisq1def.h header file and the struct definition in the ipbisq1API.h header file for more information.

write

Data to be sent from the transmitter is written to the transmit FIFO using a write() call. The user supplies the device handle, a pointer to the buffer containing the data, and the number of bytes to write. The number of bytes is checked to see if it exceeds the size of the FIFO and if not the command is executed with successive writes to the Tx FIFO port. The driver takes advantage of the carrier 32-bit double-write capability to load two FIFO words with a single PCI write until less than four bytes remain in the buffer.

read

Received data can be read from the receive FIFO using a read() call. The user supplies the device handle, a pointer to the buffer to store the data in, and the number of bytes to read. The number of bytes is checked to see if it exceeds the size of the FIFO and if not the command is executed with successive reads from the Rx FIFO port. The driver takes advantage of the carrier 32-bit double-read capability to read two FIFO words with a single PCI read until less than four bytes remain to be read.

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
435 Park Dr.
Ben Lomond, CA 95005
831-336-8891
831-336-3840 fax
support@dyneng.com