DYNAMIC ENGINEERING
150 DuBois, Suite C
Santa Cruz, CA 95060
(831) 457-8891 **Fax** (831) 457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

pci3ip, pci5ip, cpci2ip, cpci4ip, pc104pip, pc104p4ip

IP Carrier Device Drivers

Linux Driver Documentation

Revision A
Corresponding Hardware:

Pci3ip	10-1999-0404	Revision D	Firmware: Revision G
Pci5ip	10-2002-0304	Revision D	Firmware: Revision E
cPci2ip	10-2002-0805	Revision E	Firmware: Revision D
cPci4ip	10-2004-0903	Revision C	Firmware: Revision B
Pc104pip	10-2005-0401	Revision A	Firmware: Revision A
Pc104p4ip	10-2003-0503	Revision C	Firmware: Revision B

pci3ip, pci5ip, cpci2ip, cpci4ip, pc104pip, pc104p4ip
Linux Device Drivers for
PCI based IP Carriers

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

Manual Revision A. Revised June 3, 2008.

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.

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.

Table of Contents

Introduction

The pci3ip, pci5ip, cpci2ip, cpci4ip, pc104pip and pc104p4ip drivers are modular Linux drivers for their respective Industry Pack (IP) carriers from Dynamic Engineering. Each carrier can hold a number of IP modules (corresponding to the number preceding the ip designation). When the driver is loaded, it enumerates the carrier's IP bus by reading the ID proms of each installed IP. The driver makes this information available to both the user and the IP load script through the sys file system (/sys/bus/ip/devices/ip_***x_y***/) where ***x*** is the zero-based carrier number and ***y*** is the zero-based ip module number. Each device directory contains a file named devtype which contains the name of the particular IP module installed in that slot.

A device node in the /dev directory must be created for each board in order to access the hardware. A separate handle to each carrier can be obtained using open() calls (see code sample below). IO Control calls, ioctl()'s are used to configure the IP carrier and read status, although this is not necessary to operate the individual IP modules. The carrier driver is responsible for reading the value of its 8-bit user switch, operating the onboard LEDs, and a few other operations. The carrier's main function is to act as an IP bus interface providing resources for the IP modules and module drivers, which independently operate through their own file handles. See the appropriate IP driver documentation for information on the capabilities of a particular IP module. If no driver exists for a particular IP module, a generic driver ip_gen will be loaded. This allows read and write access to the IP mem and io space by passing in address offset and data values.

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 particular carrier's user manual. The 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 for both the carrier driver and carrier test application are provided in the driver delivery. Makefiles in each directory build and install the driver and test application with 'make' and 'make install' commands. A 'make clean' command will remove all executables and object files. Copy the source file directory tree to a directory where the driver and application code will be built and run ***make*** in the base directory to build the driver and test application. Run ***make install*** in each directory to copy the .ko file to the /lib/module/*version*/kernel/driver/misc/ directory and the test executables and load scripts to the /usr/local/bin/ directory (root privilege is required to install).

Load and unload scripts are provided to facilitate starting and stopping the carrier drivers and their installed IPs. The load_ip script will load the carrier driver, explore the

/sys/bus/ip/devices/ device tree to discover how many carriers and which IP modules are installed. Then it parses the /proc/devices file for the driver major numbers, creates the required number or carrier device nodes in the /dev directory, loads the appropriate IP drivers, and creates device nodes for each of the installed IP modules. Similarly the unload_ip script first unloads the installed IP module drivers, and then unloads the carrier drivers. These scripts and the ip_car_list and ip_driver_table files are copied to 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 pci3ip device ***dev_num***.

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

do
{
   printf("\nEnter target board number (starting with zero): \n");
   scanf("%d", &dev_num);
   if(dev_num < 0 || dev_num > NUM_DEVICES - 1)
      printf("\nTarget board number %d out of range!\n", dev_num);
}
while(dev_num < 0 || dev_num > NUM_DEVICES - 1);

sprintf(Name, "/dev/pci3ip_%d", dev_num);
hpci3ip = open(Name , O_RDWR);
if(hpci3ip < 2)
{
   printf("\n%sFAILED to open!\n", Name);
   return 1;
}
```

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 *car_name*_api.h file and an optional parameter used to pass data in and/or out of the device. The ioctl commands defined for the IP carriers are listed below.

IOCTL_*CAR_NAME*_GET_INFO

Function: Returns the current driver version and carrier instance number.
Input: None
Output: DRIVER_CARRIER_DEVICE_INFO structure
Notes: This call does not access the hardware, only driver parameters. See *car_name*_api.h for the definition of DRIVER_CARRIER_DEVICE_INFO.

IOCTL_*CAR_NAME*_GET_SW_ID

Function: Reads the eight-position onboard dipswitch.
Input: None
Output: unsigned long int
Notes: The switch can be used for any purpose that the user wishes. It can uniquely identify the boards installed in a chassis, or be used to distinguish configuration classes to the user's application software.

IOCTL_*CAR_NAME*_SET_BASE_CONFIG

Function: Writes to the carrier's base configuration register.
Input: unsigned long int
Output: None
Notes: This call is used to control the on-board LEDs, and control the bus error interrupt enable and the bus error interrupt clear. The bus error interrupt enable defaults to TRUE when the driver initializes, however it can be disabled using this call. If the bus error interrupt clear is written as a one in this call, to clear the latched bus error status, it will be automatically cleared and does not need to be re-written as a zero.

IOCTL_*CAR_NAME*_GET_BASE_CONFIG

Function: Returns the configuration of the base control register.
Input: None
Output: unsigned long int
Notes: This call is used mainly for testing or for saving the configuration for later restoration.

IOCTL_*CAR_NAME*_GET_STATUS

Function: Returns the carrier and IP interrupt status.
Input: None
Output: unsigned long int
Notes: Returns the masked and unmasked interrupt status for all the IP slots as well as the bus error interrupt and combined status. See *car_name*_api.h for details on all the status bits.

IOCTL_*CAR_NAME*_WAIT_ON_INTERRUPT

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

IOCTL_*CAR_NAME*_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus. This IOCTL is used for development, to test interrupt processing.

IOCTL_*CAR_NAME*_READ_ID_PROM

Function: Returns the contents of the IP ID prom for a particular slot.
Input: Slot id (char)
Output: ID_DATA structure
Notes: Returns the contents of the requested IP ID prom. The slot id (A-C for the pci3ip, A-E for the pci5ip, A-B for the cpci2ip, A-D for the cpci4ip, B-E for the pc104p4ip and no id for the pc104pip) is passed into this call and the ID_DATA structure is returned. ID_DATA contains two Boolean fields that indicate if the IP prom is valid and if it is capable of 32 MHz operation. It also contains a 12-element array of unsigned chars that shows the ID prom contents, provided the prom was found to be valid. See *car_name*_api.h for the definition of ID_DATA.

IOCTL_*CAR_NAME*_GET_INT_STATUS

Function: Returns the interrupt status read in the last ISR call.
Input: None
Output: INT_STAT structure
Notes: INT_STAT contains two fields. An unsigned long int that contains the contents of the interrupt status register read when the last interrupt was serviced and a BOOLEAN field that is true if the wait queue was inactive during that interrupt, which usually means that the interrupt timed-out. See *car_name*_api.h for the definition of INT_STAT.

IOCTL_*CAR_NAME*_RESET_ALL_IPS

Function: Resets all the IP slots.
Input: None
Output: None
Notes: Resets all three IP slots by setting and then clearing the BASE_RESET_ALL_IPS bit in the base configuration register. This bit cannot be controlled by the IOCTL_*CAR_NAME*_SET_BASE_CONFIG call.

IOCTL_*CAR_NAME*_IDENTIFY

Function: Flashes the user LEDs three times.
Input: None
Output: None
Notes: This call can be used when more than one device is installed in a chassis and it is desired to identify the physical location of a particular device.

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 a “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, Suite C
Santa Cruz, CA 95060
(831) 457-8891
(831) 457-4793 fax
support@dyneng.com