DYNAMIC ENGINEERING
150 DuBois St., Suite C Santa Cruz, CA 95060
831-457-8891
https://www.dyneng.com
sales@dyneng.com
Est. 1988

IP-BiSerial-VI-GPIO

"lpBis6Gpio"

Linux Driver Documentation

Manual Revision 1p1
Corresponding Hardware: Revision 01
10-2016-3201
FLASH revision 1p1

IpBis6Gpio

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

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

INTRODUCTION3

Driver Installation6

Linux Installation6
Pre-Requisites:**Error! Bookmark not defined.**
Install:6

Quick Start7

LIBIP_GPIO API DESCRIPTIONS9

libip_gpio_init9

libip_gpio_exit9

libip_Program_Pll9

libip_Get_Config_IO and libip_Set_Config_IO9

libip_Set_Tx_Data9

libip_Get_IO9

libip_Force_Int and libip_clr_Force_Int9

libip_Get_Info9

libip_Clear_IO9

libip_Int_Enable and libip_Int_Disable10

ip_gpio_await_int10

Various Getter/Setters10

WARRANTY AND REPAIR11

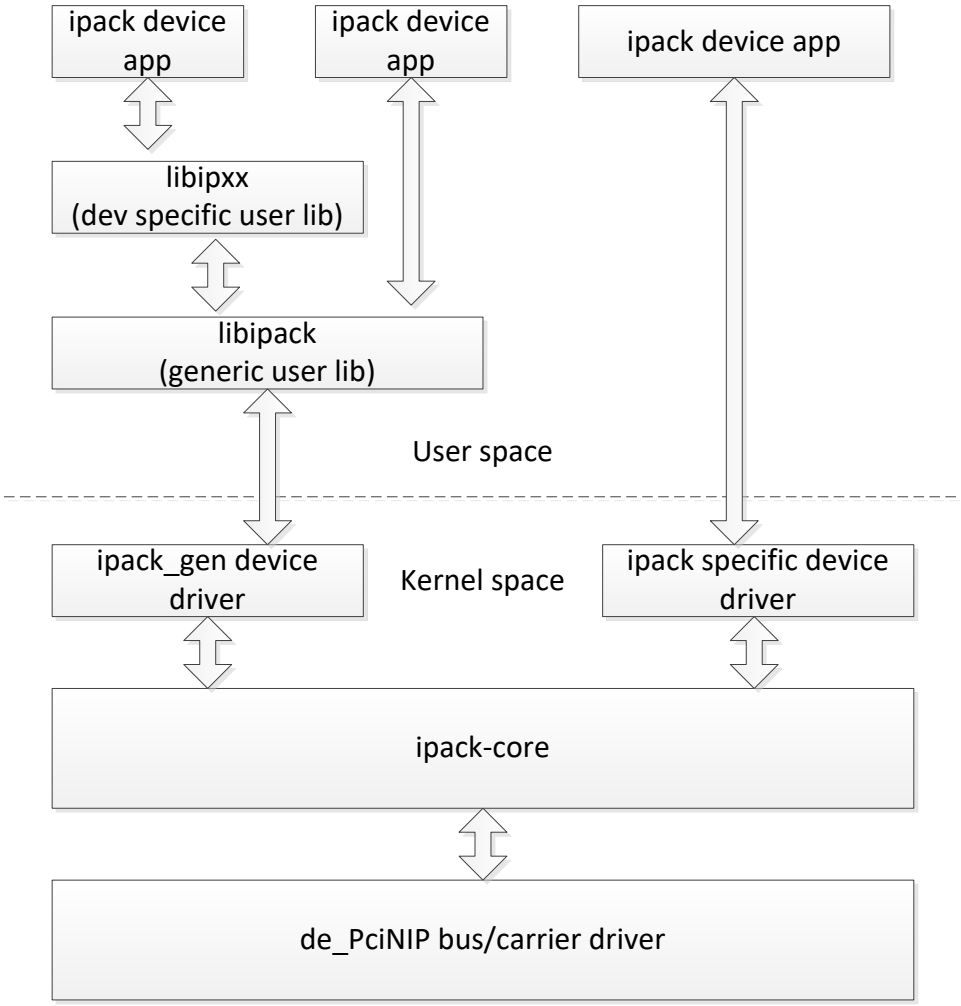
Service Policy11
Support11

For Service Contact:11

Introduction

Dynamic Engineering has developed and supplies user-level IPACK (Industry Pack) libraries which support both generic IPACK operations, and device specific functions. These libraries interface with the ipack-core (Open Source ported from 3.5 kernel) via the ipack_gen(eric) driver. Thus, this kernel module serves as a gasket between the user-libraries and the ipack-core. The Dynamic Engineering PciNIP driver is a bus/carrier driver supporting all our released carrier/bridge cards interfacing with the ipack-core.

The libraries and ipack-gen driver have been validated on an i7 Ubuntu server running 3.8.0-44 kernel (64 bit) SMP (little Endian platform and a P2020 (PPC) target running 3.0.48-rt70 SMP kernel (big Endian platform).



The User app is an example application with a simple, and powerful menu plus a series of “tests” that can be run on the installed hardware. Each of the tests execute calls to the library. With the sequence of calls demonstrated, the functions of the hardware are

utilized for loop-back testing. The software is used for manufacturing tests at Dynamic Engineering.

The hardware manual defines the pin-out, the bitmaps and detailed configurations for each feature of the design. The library handles all aspects of interacting with the hardware. For added explanations about what some of the library functions do, please refer to the hardware manual.

We strive to make a usable product. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us.

Note

This documentation will provide information about all calls made to the library, and how the library interacts with the device for each of these calls. For more detailed information on the hardware implementation, refer to the IpBis6Gpio user manual (also referred to as the hardware manual).

Driver Installation

The library software has several components. There are three kernel modules that must be installed to use the library for this specific module. The first two modules, ipack and de_PciNIP conrol the IP Carrier. The third module, ipack_gen is a generic IP driver with a custom Interrupt Handler for modules that require a custom IRQ. Once those modules are installed, the user level library for this module can be used with your application. Technically there are two library layers, one for this module and one that acts as a layer between the customized library and the generic ipack_gen driver. Applications made to use this module should only use the calls from the library specifically designed for the associated module.

Linux Installation

Install:

1. Extract the provided zip files (de_PciNIP.zip and libipxx.zip) and place them together in a directory of your choice.
2. Using a terminal navigate to the directory containing the extracted files.
3. Navigate to <your_directory>/de_PciNip/build and run the “build_all” script (or “build_all_ppc” if running on a PowerPC processor).
4. Next, run the “bnm” script as Sudo.
5. Navigate back to the directory you created, then navigate to <your_directory>/libipxx/ipgpio/build again, run “build_all” and then “bnm” (again bnm must be run as Sudo).
6. Multiple Modules:
    a. To install multiple modules, all that needs to be done is to have the libraries compiled (the drivers are already installed at this point). Navigate to the desire module folder and only build the library. See the “build_all” script to for an example of how to compile just the library and/or other software with the provided libraries.

Quick Start

Once the drivers are installed and the library is compiled, here is a quick minimal example of how to use the library:

```
#include <stdio.h>
#include <stdint.h>
#include "libip_bis6_gpio.h"

#define MAX_NUM_MODULES 3 //this depends on the carrier model

int main (int argc, char* argv[]){
	ipack_handle_t module[MAX_NUM_MODULES] = {0};
	ip_gpio_conf_t config[MAX_NUM_MODULES] = {};
	ip_gpio_cstm_wait_t int_data = {};
	uint_32 data;

	int num_modules, i;

	/*********************************************************
	* libip_gpio_init returns the number of modules, while
	* also populating the handles in modules[].
	* The 1 as the first parameter means the routine will
	* find all IP-BIS6-GPIO modules on the carrier.
	*
	* NOTE: The PLL is programmed with a default file during
	* initialization, so it can be used immediately.
	*********************************************************/
	num_modules = libip_gpio_init(1,module);
	if(num_modules < 0){
		printf("Could not Initialize Library\n");
		exit(-1);
	}


	/*********************************************************
	* Each module can be configured
	*********************************************************/

	for (i = 0; i < num_modules; i++){
		//if interrupts are set to level, then a call to disable interrupts needs to be added here first

		config[i].base_control.out_en = 1; //enable tx to send data out (see HW Manual for more
info)
		config[i].base_control.clk_cos_sel = 1; //use PLL instead of reference clock
		config[i].base_control.local_reset = 0;

		config[i].direction_bits = 0xFFF; //enable 12 transmit pins
		config[i].termination_bits = 0xFFF000; //enable 8 termination pins
		config[i].int_enable = 0xFFF000; //enable 4 interrupt pins
		config[i].rising_enable = 0xFFF000; //enable rising edge interrupts
		config[i].falling_enable = 0xFFF000; //enable falling edge interrupts
		config[i].pol_bits = 0x00; //set polarity to low
		config[i].edge_bits = 0xFFF000; //set to allow edge interrupts versus level
```

```
		//library call to program module settings, returns 0 on success.
		if(libip_Set_Config_IO(module[i], &config[i])){
			printf("failed to configure module: %d\n",i);
		}

		//clear out any data that may be sitting (just in case)
		libip_Clear_IO(module[i]);
	}

	/*******************************************************
	* Transmit Data
	*******************************************************/
	data = 0xA5A // only the first 12 bits are set to transmit
	if(libip_Set_Tx_Data(module[0], &data)){
		printf("Could not transmit data\n");
	}


	/*******************************************************
	* Wait for Interrupts
	*
	* Wait for interrupt on module 0 with a 3 second timeout
	* if timeout = -1, then it would wait forever
	*******************************************************/

	if((ret = libip_gpio_await_int(module[0], 3, &int_data))){
		if(ret == -ETIMEOUT){
			printf("Interrupt wait timed out\n");
		}
		else{
			printf("Wait for int, failed\n");
		}
	}

	/*******************************************************
	* Transmit Data
	*
	* Must remember to exit to post library semaphore
	*******************************************************/
	libip_exit();
}
```

Libip_gpio API descriptions

The following library APIs provide user-level access to specific Dynamic Engineering IPACK modules. More detailed descriptions can be found in libip_bis6_gpio.h.

libip_gpio_init

Initialize library. This function must be invoked prior to utilizing any of the following access routines. This function returns a list of IP-BIS6-GPIO modules either containing the first module found, or all modules.

libip_gpio_exit

Exits the library, must be called upon exit to release any locks on the library.

libip_Program_Pll

This programs the PLL with a provided file path or default PLL file NULL is passed as the pll_file_path parameter.

libip_Get_Config_IO and libip_Set_Config_IO

These are the getter/setter functions for configuring the module. The configuration structure allows the user to customize each pin setting.

libip_Set_Tx_Data

This is functions sets the values in the transmit register, if the latch_en is set to 1 in the control register (set in libip_Set_Config_IO above), the data will transmit immediately.

libip_Get_IO

This function reads the filtered and unfiltered data registers and returns the values.

libip_Force_Int and libip_clr_Force_Int

These functions force the module to trigger a system interrupt and clear the forced system interrupt via the forced interrupt.

libip_Get_Info

This function fills in the bd_info struct passed to it with the modules revision numbers (major and min), the slot number, and the dip switch value.

libip_Clear_IO

this function clears any data in TX register and clears any status in the COS falling and rising registers

libip_Int_Enable and libip_Int_Disable

These functions enable and disable interrupts via the master interrupt control register. If the master is not enabled, no system level interrupts can be triggered by this module.

ip_gpio_await_int

This function will wait for a system level interrupt to occur and will return the interrupt data associated with the interrupt. All interrupt types are handled by this routine.

Various Getter/Setters

The following functions are merely various getter/setter functions for individual registers, mainly helpful for debugging/testing.

int libip_Get_Tx_Data(ipack_handle_t handle, uint32_t *data)
int libip_Get_Half_Div(ipack_handle_t handle, uint16_t *half_div_val)
int libip_Set_Half_Div(ipack_handle_t handle, uint16_t *half_div_val)
int libip_Set_Cos_Rising_Stat(ipack_handle_t handle, uint32_t *reg_data)
int libip_Get_Cos_Rising_Stat(ipack_handle_t handle, uint32_t *reg_data)
int libip_Set_Cos_Falling_Stat(ipack_handle_t handle, uint32_t *reg_data)
int libip_Get_Cos_Falling_Stat(ipack_handle_t handle, uint32_t *reg_data)
int libip_Get_Direct_Data(ipack_handle_t handle, uint32_t *data)
int libip_Get_Filtered_Data(ipack_handle_t handle, uint32_t *data)
int libip_Get_Vector(ipack_handle_t handle, uint32_t *data)
int libip_Set_Vector(ipack_handle_t handle, uint32_t *data)
int libip_Get_Int_Status(ipack_handle_t handle, uint16_t *data)
int libip_Get_Master_Int_Config(ipack_handle_t handle, uint16_t *data)
int libip_Set_Master_Int_Config(ipack_handle_t handle, uint16_t *data)
int libip_Get_Control_Reg_Config(ipack_handle_t handle, uint16_t *data)
int libip_Set_Control_Reg_Config(ipack_handle_t handle, uint16_t *data)

Warranty and Repair

Please refer to the warranty page on our website for the current warranty offered and options.
http://www.dyneng.com/warranty.html

Service Policy

The driver has gone through extensive testing, and while not infallible, problems experienced will likely be “cockpit error” rather than an error with the driver. We will work with you to determine the cause of the issue. If the effort is more than a quick conversation, we will offer a support contract. We can write updates to the driver to add features, create middleware etc.

Support

The software described in this manual is provided at no cost to clients who have purchased the corresponding hardware. Minimal support is included along with the documentation. For help with integration into your project please contact sales@dyneng.com for a support contract. Several options are available. With a contract in place Dynamic Engineers can help with system debugging, special software development, or whatever you need to get going.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite C
Santa Cruz, CA 95060
831-457-8891
support@dyneng.com