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

Software User's Guide
(Linux)

Libipack/lib_gen/libipxx

IPACK user libraries
IPACK generic driver

Libipack

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

Product Description

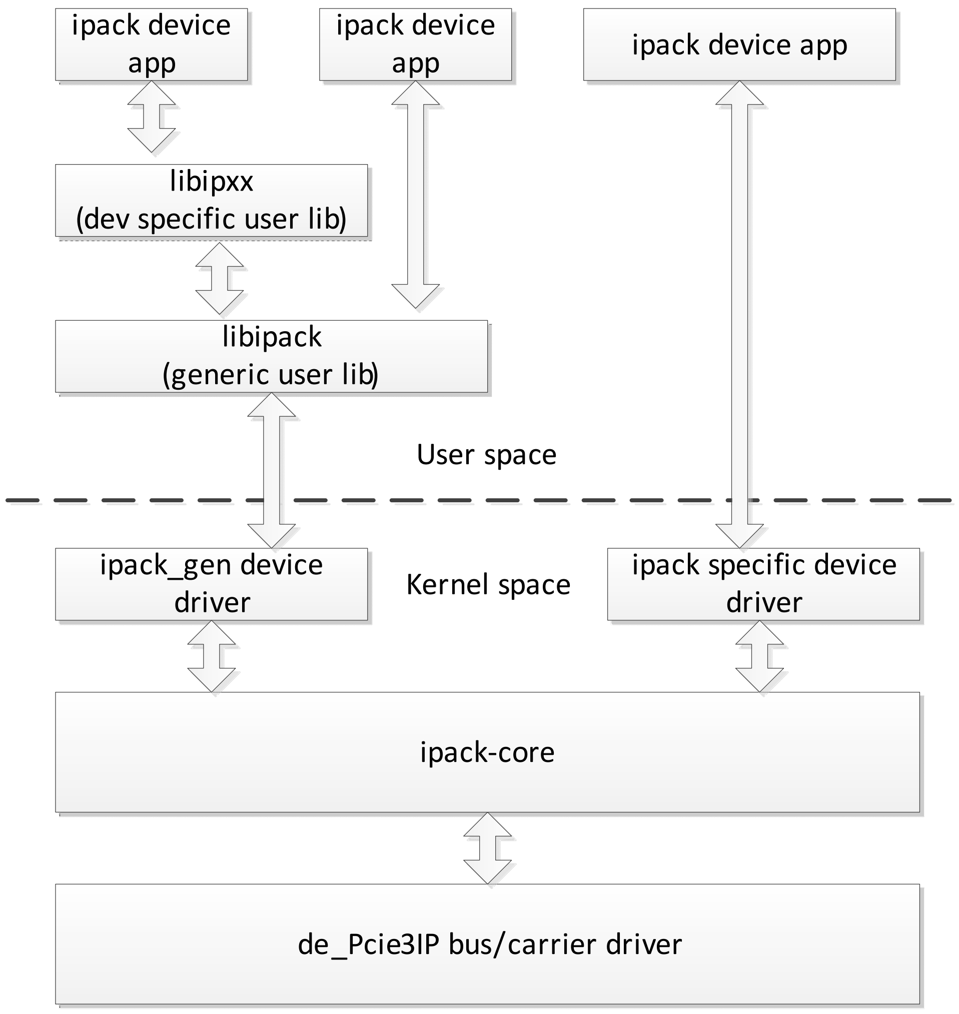
Dynamic Engineering has developed and supplies user-level IPACK (Industry Pack) libraries which support both generic IPACK operations, and device specific functions.
These libraries interface with the ipack-core (Open Source ported from 3.5 kernel) via the ipack_gen(eric) driver.   Thus, this kernel module serves as a gasket between the user-libraries and the ipack-core. The Dynamic Engineering PCIe3IP driver is a bus/carrier driver supporting the PCIe3IP carrier/bridge card interfacing with the ipack-core.

Software Description

As described in the PCIe3IP SW manual, the ipack-core and de_PCIe3IP kernel modules must be built and installed prior to utilization of any other IPACK components including those described within this document. Please see that manual for details WRT building and installing these modules.

Based upon specific IPACK device complexity, application developers have multiple methods available for interfacing with the device. A kernel driver may be developed or supplied as depicted on the right side of the diagram below.

This document will address the components and methods depicted on the left side of the diagram. Namely, libipack, lib_gen(eric), and a libipxx library.


ipack device app
ipack device app
ipack device app
libipxx (dev specific user lib)
libipack (generic user lib)
User space
ipack_gen device driver
Kernel space
ipack specific device driver
ipack-core
de_Pcie3IP bus/carrier driver

The libraries and ipack-gen driver have been validated on an i7 Ubuntu server running 3.8.0-44 kernel (64 bit) SMP.

Libipack API descriptions

The following APIs support generic Industry Pack operations and functions. Please review the following descriptions for caveats and general usage details.

```
/*********************************************************************
*  libipack_init
*
*  Initialize library. This function must be invoked prior to utilizing
*  any of the following access routines.  If utilized in conjunction
*  with any Dynamic Engineering module specific libraries, it will be
*  invoked implicitly.
*
*  Parameters:
*  N/A (void)
*
*  Returns:
*  Number of modules upon success, < 0 upon failure
*/
int libipack_init (void);

/*********************************************************************
*  libipack_exit
*
*  Exit/shutdown library. This function should be invoked upon
*  application termination.  If utilized in conjunction with any
*  Dynamic Engineering module specific user libraries, that library
*  will invoke this function upon exit.
*
*  Parameters:
*  N/A (void)
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
int     libipack_exit (void);
```

```
/**********************************************************************
*  ipack_get_modules
*
*  Find/get a list of specified IPACK modules. A specific manufacturer
*  ID and model number can be specified, or any combination of
*  manufacturer and model number via the parameter IPACK_ANY_ID.  A
*  list of IPACK modules meeting the criteria is returned. If utilized
*  in conjunction with any Dynamic Engineering module specific user
*  libraries, the library invoke this function during discovery
*  processing.
*
*  Parameters:
*  man_id       - IPACK manufacturer ID or IPACK_ANY_ID
*  model_num    - IPACK model number of IPACK_ANY_ID
*  modules      - pointer to an array of size (libipack:MAX_IP_MODULES)
*                 if find_all is true.  Otherwise, an array of a single
*                 element is sufficient.
*  Returns:
*  Number of modules upon success, < 0 upon failure
*/
int ipack_get_modules (unsigned int man_id, unsigned short model_num,
                                      int find_all, unsigned *modules);

/**********************************************************************
*  ipack_get_modinfo
*
*  Get IPACK module info for specified device.
*
*  Parameters:
*  handle       - IPACK handle returned from ipack_get_modules
*  modinfo      - Module info returned in this structure
*
*  Returns:
*  Number of modules upon success, < 0 upon failure
*/
int ipack_get_modinfo (ipack_handle_t handle,
                                          ipack_modinfo_t* modinfo);
```

```
/*********************************************************************
*  ipack_set_irq_parms
*
*  This function sets required IRQ processing parameters.  It must be
*  invoked prior to enabling a module interrupt.  If utilized in
*  conjunction with any Dynamic Engineering module specific user
*  libraries, that library will invoke this function during
*  configuration processing.
*
*  Parameters:
*  handle       - IPACK handle returned from ipack_get_modules
*  clr_regs     - Pointer to an array of 2 register offsets in I/O
*                 space.
*                 Many IPACK modules require interrupt to be cleared
*                 via read accesses.  Currently 2 register offsets are
*                 expected, though the same offset can be specified
*                 twice or NULL if this logic is not required.
*
*  Returns:
*  0 upon success, < 0 upon failure
*/
int ipack_set_irq_parms (ipack_handle_t handle, unsigned short
*clr_regs);

/*********************************************************************
*  ipack_wait_irq
*
*  This function awaits interrupt processing completion.
*  ipack_set_irq_parms must be invoked (directly, or indirectly via
*  module specific library prior to invocation.
*
*  Parameters:
*  handle       - IPACK handle returned from ipack_get_modules.
*  timeout      - timeout specified in Linux jiffies, 0 or WAIT_FOREVER
*                 are valid values.
*  int_stat     - Pointer to an unsigned int.  Status read from
*                 clr_regs is returned assigned to this variable.  NULL
*                 is a valid input parameter
*
*  Returns:
*  0 upon success, < 0 upon failure (likely timeout).
*/
int ipack_wait_irq (ipack_handle_t handle, unsigned long timeout,
                                          unsigned int *int_stat);
```

```
/*********************************************************************
*  ipack_readX
*
*  The following read functions reads from the specified module memory
*  region at the specified offset.
*
*  Parameters:
*  handle       - IPACK handle returned from ipack_get_modules.
*  region       - ipack_region_t (IO, ID, or MEM space).
*  offset       - byte offset from base of specified region
*  val          - Value read during access.
*                 input parameter
*
*  Returns:
*  Number of bytes read upon success, < 0 upon failure.
*/
ssize_t ipack_read8 (ipack_handle_t handle, ipack_region_t region,
                                 unsigned offset, unsigned char *val);
ssize_t ipack_read16 (ipack_handle_t handle, ipack_region_t region,
                                 unsigned offset,unsigned short *val);
ssize_t ipack_read32 (ipack_handle_t handle, ipack_region_t region,
                                  unsigned offset, unsigned int *val);

/*********************************************************************
*  ipack_writeX
*
*  The following write functions writes to the specified module memory
*  region at the specified offset.
*
*  Parameters:
*  handle       - IPACK handle returned from ipack_get_modules.
*  region       - ipack_region_t (IO, ID, or MEM space).
*  offset       - byte offset from base of specified region
*  val          - Value to be written.
*
*  Returns:
*  Number of bytes written upon success, < 0 upon failure.
*/
ssize_t ipack_write8 (ipack_handle_t handle, ipack_region_t region,
                                  unsigned offset, unsigned char val);
ssize_t ipack_write16 (ipack_handle_t handle, ipack_region_t region,
                                 unsigned offset, unsigned short val);
ssize_t ipack_write32 (ipack_handle_t handle, ipack_region_t region,
                                   unsigned offset, unsigned int val);
```

Libipxx API descriptions

The following library APIs provide user-level access to specific Dynamic Engineering IPACK modules. Currently, the IP-Parallel_HV is the only IPACK module supported by a device specific user library. This section shall expand as user libraries are added for other Dynamic Engineering Industry Pack modules.

Libiphv API descriptions

```
/*********************************************************************
*  libiphv_init
*
*  Initialize library. This function must be invoked prior to utilizing
*  any of the following access routines.  This function returns a list
*  of IP-HV modules either containing the first module found, or all
*  modules
*
*  Parameters:
*  find_all -    (0=find first, 1=find all)
*  modules  -   pointer to an array of size (libipack:MAX_IP_MODULES)
*               if find_all is true.  Otherwise, an array of a single
*               element is sufficient.
*  Returns:
*  Number of modules upon success, < 0 upon failure
*/
int libiphv_init (int find_all, unsigned *modules);

/*********************************************************************
*  libiphv_exit
*
*  Exit/shutdown library. This function should be invoked upon
*  application termination.
*
*  Parameters:
*  num_modules - Value returned from libiphv_init
*  modules:    - pointer to an array returned from libiphv_init
*                if find_all is true.  Otherwise, an array of a single
*                element is sufficient.
*  Returns:
*  0 upon success, < 0 upon failure
*/
int libiphv_exit (unsigned num_modules, unsigned *modules);
```

```
/*********************************************************************
*  libiphv_config_mod
*
*  Configure IP-HV module. This routine is invoked to setup various
*  control parameters prior to issuing I/O.
*
*  Parameters:
*  handle   -   Handle returned in module list (libiphv_init)
*               specifying which IP-HV module to configure.
*  config   -   pointer to IP-HV module configuration parameters (see
*               libiphv.h for details
*  Returns:
*  0 upon success, < 0 upon failure
*/
int iphv_config_mod (ipack_handle_t handle, iphv_config_mod_t* config);
```

Installation

1) Install ipack and de_PCIe3IP kernel modules, see SW manual for the de_PCIe3IP.
2) Copy ipack_gen.c, ipack_gen.h (ipack_gen) to your module build directory. Invoke the system "make." Alternatively a makefile for ipack_gen has been included for out of tree kernel module build. If this build method is utilized, cd to the build directory and invoke the script ./build_all. This script will invoke the Makefile to build ipack_gen.ko, compile/archive both libipack and libiphv, and finally building a test application (ip_IoApp).
3) Copy the resulting ipack.ko module to the target platform/directory.
4) Copy the startup script bnm to the target.
5) Invoke the script (./bnm), it will perform an insmod of ipack_gen and create the required device. The script may be invoked from the systems rc.local file as well.

Sample application

The application ip_ioApp.c demonstrates proper usage of library functions/operations for both libipack and libiphv. As previously mentioned, the Dynamic Engineering IP-Parallel-HV module is employed for demonstration purposes.

1) The build_all script contained in the build sub directory will compile, and archive the libraries, compile the sample app, as well as invoking Make for the kernel module ipack_gen.ko. Compilation steps for libraries and apps are as follows:

a) gcc -c -g -Wall ../de_Pcie3IP/libipack.c
b) ar rvs libipack.a libipack.o
c) gcc -c -g -Wall ../de_Pcie3IP/libiphv.c
d) ar rvs libiphv.a libiphv.o
e) gcc -g -Wall -o ip_io ../de_Pcie3IP/ip_IoApp.c libipack.a libiphv.a

Invocation parameters

Sample application invocation is as follows:

```
./ip_io
```

The application expects that a loopback fixture is attached to the IP-PARALLEL-HV module(s). It validates proper I/O and interrupt generation for all such modules installed. If the fixture is not attached, the test will fail for that module.

Support Contract

Dynamic Drivers are provided AS-IS and sometimes our clients need a little help. Please refer to the support contract page on our website for options about getting help with your driver use and SW development.

http://www.dyneng.com/TechnicalSupportFromDE.pdf

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

Before returning a product for repair, verify as well as possible that the suspected unit is at fault. Then call the Customer Service Department for a RETURN MATERIAL AUTHORIZATION (RMA) number. Carefully package the unit, in the original shipping carton if this is available, and ship prepaid and insured with the RMA number clearly written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. Dynamic Engineering will not be responsible for damages due to improper packaging of returned items. For service on Dynamic Engineering Products not purchased directly from Dynamic Engineering contact your reseller. Products returned to Dynamic Engineering for repair by other than the original customer will be treated as out-of-warranty.

Out of Warranty Repairs

Out of warranty repairs will be billed on a material and labor basis. The current minimum repair charge is $150. Customer approval will be obtained before repairing any item if the repair charges will exceed one half of the quantity one list price for that unit. Return transportation and insurance will be billed as part of the repair and is in addition to the minimum charge.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois St. Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 fax
InterNet Address support@dyneng.com