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
(VxWorks)

Industry Pack

PCiNIP carrier/bus driver
vxbIpackLib generic library
libIphv IP-Parallel-HV application library
libIpCtrb IP-BiSerial-VI-CTRB application library
libIpSIB IP-BiSerial-VI-SIB application library
libIpOpto IP-OptoISO-16 application library

IPACK

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

Product Description

Dynamic Engineering has developed a carrier/bus driver (dePciNIP) for the Dynamic Engineering PCIe and PCI carrier cards supporting Industry Pack Modules. Further, this package contains vxbIpackLib, libIphv, libIpCtrb, libIpOpto and sample applications demonstrating proper operation of the Dynamic Engineering IP-Parallel-HV, IP-BiSerial-VI-CTRB and IP-OptoISO modules.

The dePciNIP bus controller driver interfaces with vxbIpackLib. This library implements a generic IPACK bus interface and will support any vendor's carrier card driver which conforms to the defined interface. The library registers the IPACK bus with the VxBus system as well as announcing IPACK devices to VxBus upon discovery by the underlying carrier/bus controller driver.

An application may either interface with an IPACK module via the generic interface (reads and writes to a module), or register a driver for a specific IPACK device with VxBus vis this generic library.

If a driver has claimed a particular device, vxbIpackLib will not support generic read/write access to that device. Only "orphaned" devices can be managed via the generic interface.

The application library libIphv provides an example of utilization of the vxbIpackLib generic interface to configure and control the IP-HV module.

The PciNIP carrier card supports up to 5 IPACK modules depending upon the carrier. The IP-Parallel-HV module is a digital interface card supporting up to 24 high voltage input/outputs. The IP-BiSerial-VI-CTRB module supports 8 channels of counter/timer or 1 shot functionality depending upon SW configuration. The IP-OptoISO module supports 16 FET channels.

Please see the Pcie/PCI carrier, IP-Parallel-HV, IP-BiSerial-VI-CTRB, and IP-OptoISO hardware manuals for further HW specific information.

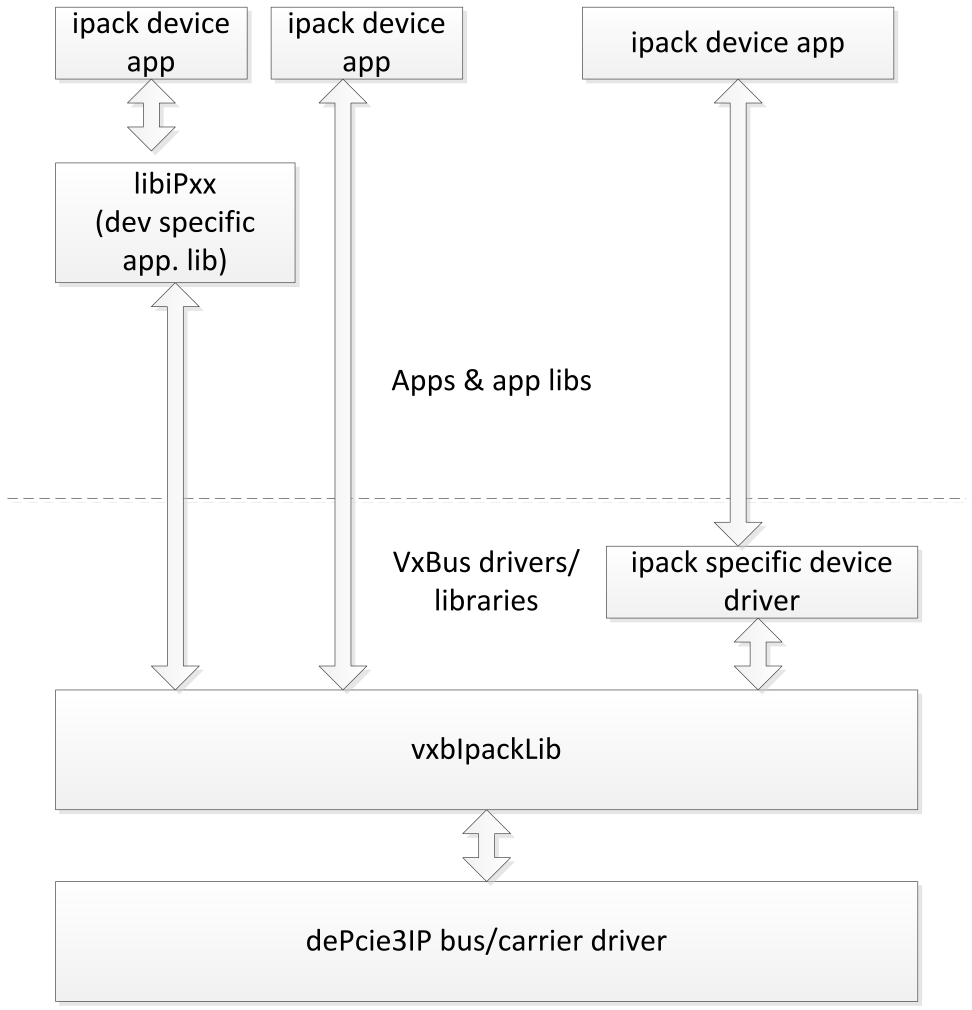
Software Description

The dePciNIP driver is responsible for discovering IPACK modules and providing various call-back functions for accessing these modules to vxbIpackLib.

Many of the APIs are publically available (via command line or direct application reference) assuming a driver has not claimed a particular device. In this case, the driver claiming module is responsible for obtaining the call-back functions and directly invoking them.

The generic IPACK interface should be sufficient for controlling most IPACK modules especially if used in conjunction with other Dynamic Engineering app libraries supporting those specific devices. More user libraries will be added to this package as they are developed.

This diagram below depicts the VxWorks IPACK layering provided by this package. In this diagram the Pcie3IP carrier card is shown. The dePciNIP supports all Dynamic Engineering IPACK carrier cards both current and legacy.



The driver, libraries, and application have been validated on a P2020 (multi-core PPC) platform running VxWorks SMP revision 6.9. This platform is big endian.

vxbIpackLib API descriptions

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

```
/*********************************************************************
*
* ipackBusAnnounceDevs - Announce IPACK devices
*
* This routine announces all detected IPACK modules on the specified
* bus.  Each carrier card supports one IPACK bus.
*
* INPUT PARMS:
*   busCtrlID   - Device ID of the carrier card.
*   numModules  - Number of modules discovered
*   modParms    - Pointer to list of module parameters.
*   busNum      - Bus number to assign, or -1 = auto assign
*                 bus number
*
* RETURNS: OK, or ERROR
*
* ERRNO:  N/A
*/
STATUS ipackBusAnnounceDevs (VXB_DEVICE_ID busCtrlID, UINT numModules,
                           VXB_IPACK_MOD_PARMS *modParms, int busNum);
```

```
/*******************************************************************
*
* vxbIpackFindModules - Find IPACK modules
*
* This routine returns a list of modules matching the specified
* manufacturer ID and model number.
*
* INPUT PARMS:
*   manId       - Manufacturer ID or VXB_IP_ANY
*   modelNum    - Model number or VXB_IP_ANY
*   designId    - Design Id or VXB_IPACK_ANY
*   numMods     - Max. number of modules to find.
*   modList     - List of modules returned in this parameter
*                  (size of numMods).
*
* RETURNS: Number of modules found
*
* ERRNO:  N/A
*/
UINT vxbIpackFindModules (UINT16 manId, UINT16 modelNum, UINT numMods,
                             UINT16 designId, VXB_DEVICE_ID *modList);

/*******************************************************************
*
* vxbIpackRead - Read from an IPACK module
*
* This function reads data as specified from an IPACK module.
*
* INPUT PARMS:
*   pDev        - Module device ID.
*   space       - Module address region to read from.
*   size        - Access size (byte, short, or long).
*   offset      - Offset within address region.
*   count       - Number of elements to read.
*   opts        - ipackOpts_t (IPACK_LO_WD|IPACK_HI_WD|IPACK_AUTO_INC)
*   vals        - Data read (size of count elements)
*
* RETURNS: OK, or ERROR
*
* ERRNO:  N/A
*/
STATUS vxbIpackRead (VXB_DEVICE_ID pDev, VXB_IPACK_SPACE space,
      VXB_IPACK_SIZE size, UINT32 offset, UINT32 count, ipackOpts_t
      opts, void *vals);
```

```
/*******************************************************************
*
* vxbIpackWrite - Write to an IPACK module
*
* This writes data as specified to an IPACK module.
*
* INPUT PARMS:
*   pDev        - Module device ID.
*   space       - Module address region to write.
*   size        - Access size (byte, short, or long).
*   offset      - Offset within address region.
*   count       - Number of elements to write.
*   opts        - ipackOpts_t (IPACK_LO_WD|IPACK_HI_WD|IPACK_AUTO_INC|
*                                                    IPACK_WR_FLUSH)
*   vals        - Data to be written (size of count elements).
*
* RETURNS: OK, or ERROR
*
* ERRNO:  N/A
*/
STATUS vxbIpackWrite (VXB_DEVICE_ID pDev, VXB_IPACK_SPACE space,
        VXB_IPACK_SIZE size, UINT32 offset, UINT32 count, ipackOpts_t
        opts, void *vals);

/*******************************************************************
*
* vxbIpackRqstIrq - Install an IPACK module ISR
*
* Install an ISR to handle IPACK module interrupt and enable module
* interrupt.  This function (ISR) must not block as it is invoked from
* interrupt context.
*
* INPUT PARMS:
*   pDev        - Module device ID.
*   handler     - ISR to be installed (function pointer).
*   arg         - Argument to be passed to ISR.
*
* RETURNS: OK, or ERROR
*
* ERRNO:  N/A
*/
STATUS vxbIpackRqstIrq (VXB_DEVICE_ID pDev, STATUS (*handler)(void *),
                                                          void *arg);
```

```
/*******************************************************************
*
* vxbIpackFreeIrq - Uninstall an IPACK module ISR
*
* Remove installed IPACK module ISR and disable module interrupt.
*
* INPUT PARMS:
*   pDev        - Module device ID.
*
* RETURNS: OK, or ERROR
*
* ERRNO:  N/A
*/
STATUS vxbIpackFreeIrq (VXB_DEVICE_ID pDev);

/*******************************************************************
*
* vxbIpackDeviceShow - Show IPACK module parameters
*
* Display IPACK module parameters
*
* INPUT PARMS:
*   pDev        - Module device ID.
*
* RETURNS: N/A (void
*
* ERRNO:  N/A
*/
void vxbIpackDeviceShow (VXB_DEVICE_ID pDev);

/*******************************************************************
*
* vxbIpackBusShow - Show bus controller/carrier parameters
*
* Displays bus controller/carrier parameters.  This function currently
* only supports Dynamic Engineering carrier cards.
*
* INPUT PARMS:
*   pDev        - Bus controller device ID.
*
* RETURNS: N/A (void
*
* ERRNO:  N/A
*/
void vxbIpackBusShow (VXB_DEVICE_ID busCtrlDev);
```

libIpxx API descriptions

The following library APIs provide application access to specific Dynamic Engineering

IPACK modules. Currently, the IP-Parallel_HV, IP-BiSerial-IV-CTRB, and IP-BiSerial-IV-SIB are the only IPACK modules currently supported by a device specific app library. This section shall expand as libraries are added for other Dynamic Engineering Industry Pack modules.

libIphv API descriptions

```
/*********************************************************************
*  iphvInit
*
*  Initialize IP-HV library. This function returns a list of IP-HV
*  modules.
*
*  Parameters:
*  numModules   - Number of modules
*  modules      - Pointer to array of size numModules.
*                 Will contain VXB_DEVICE_IDs of modules found.
*  Returns:
*  Number of IP-HV modules discovered
*/
int iphvInit (UINT numModules, VXB_DEVICE_ID *modules);

/*********************************************************************
*  iphvExit
*
*  Exit/shutdown library. This function should be invoked upon
*  application termination, it disables all active module interrupts,
*  and resets HW to default settings.
*
*  Parameters: (NA, void)
*
*  Returns:
*  NA void
*/
void iphvExit (void);
```

```
/*******************************************************************
*  iphvConfigMod
*
*  Configure IP-HV module. This routine is invoked to setup various
*  control parameters prior to issuing I/O.
*
*  Parameters:
*  pDev     -   Device ID of IP-HV module to configure.
*  config   -   pointer to IP-HV module configuration parameters.
*
*  Returns:
*  RETURNS OK upon success, ERROR upon failure
*/
int iphvConfigMod (VXB_DEVICE_ID pDev, iphvConfigMod_t* config);


/*******************************************************************
*  iphvAwaitInt
*
*  Wait for an interrupt from the specified IP-HV module.
*
*  Parameters:
*  pDev     -   Module device ID.
*  intStat  -   Interrupt status read (upon success).
*  timeout  -   Timeout in ticks.
*
*  Returns:
*  OK upon success, ERROR upon failure
*/
STATUS iphvAwaitInt (VXB_DEVICE_ID pDev, UINT32 *intStat,
                                               _Vx_ticks_t timeout);
```

libIpCtrb API descriptions

```
/*****************************************************************************
*  libIpCtrbInit
*
*  Initialize library. This function must be invoked prior to utilizing any
*  of the following access routines.  This function returns a list of IP-CTRB
*  modules either containing the first module found, or all modules.
*
*  Parameters:
*  maxMods  -   (1 to MAX_IPACK_MODS)
*  modules  -   pointer to an array of size (maxMods)
*
*  Returns:
*  Number of modules upon success, ERROR upon failure
*/
int libIpCtrbInit (int maxMods, VXB_DEVICE_ID *modules);

/*****************************************************************************
*  libIpCtrbExit
*
*  Exit/shutdown library. This function should be invoked upon application
*  termination.
*
*  Parameters:
*  modules:    - pointer to an array returned from libIpCtrbInit
*
*  Returns:
*  OK upon success, ERROR upon failure
*/
STATUS libIpCtrbExit (VXB_DEVICE_ID *pDev);
```

```
/*****************************************************************************
*  ipCtrbInitiateTimer
*
*  Start a IP-CTRB module timer. This routine is invoked to initiate a timer.
*  It configures HW to generate an external pulse, and interrupt (if enabled)
*  when the timer expires.  This routine is non-blocking, interrupt occurence
*  can be determined by invoking ipCtrbAwaitInt.
*
*  Parameters:
*  pDev     -   Module device id returned in module list (libIpCtrbInit)
*               specifying which IP-CTRB module.
*  channel  -   Channel on the module to initiate a timer (0-7)
*  extClk   -   TRUE = use external clock, FALSE = internal.
*  reload   -   FALSE = normal counter mode, TRUE = counter mode/auto reload.
*  intEnbl  -   Enable interrupt generation (FALSE=no int, TRUE=interrupt)
*  duration -   5-4292967295 usec.
*
*  Special Considerations:
*  The function will fail if a timer is currently active, to cancel an
*  outstanding timer, invoke the function ipCtrbResetChan.  Further,
*  if interrupts are not enabled, timer expiration can be determined by
*  invoking this routine, it will fail until timer is no longer active.
*
*  Returns:
*  OK upon success, ERROR upon failure
*/
STATUS ipCtrbInitiateTimer (VXB_DEVICE_ID pDev, UINT channel,
                  BOOL extClk, BOOL reload, BOOL intEnbl, UINT32 duration);
```

```
/*****************************************************************************
*  ipCtrbInitiate1shot
*
*  Start a IP-CTRB module one shot timer. This routine is invoked to
*  initiate a one shot timer.  It will configure HW to generate an external
*  pulse of the duration specified. The one shot is started based upon an
*  internal or external trigger. It can be configured to start on the rising
*  or falling edge of the selected trigger. This routine is non-blocking,
*  if interrupts are not enabled, completion mayb be determined by invoking
*  ipctrb_await_int.
*
*  Parameters:
*  pDev -       Module device ID returned in module list (libIpCtrbInit)
*               specifying which IP-CTRB module.
*  channel  -   Channel on the module to initiate the one shot (0-7).
*  extClk   -   TRUE = use external clock, FALSE = internal.
*  trigger  -   FALSE = internal trigger (clock), TRUE = external trigger
*  edgeSel  -   FALSE = falling edge, TRUE = rising edge of trigger
*  intEnbl  -   Enable interrupt generation (FALSE=no int, TRUE=interrupt)
*  duration -   5-4292967295 pulse width (usec).
*
*  Special Considerations:
*  The function will fail if a one shot is currently active, to cancel an
*  outstanding timer, invoke the function ipCtrbResetChan.  Further,
*  if interrupts are not enabled, timer expiration can be determined by
*  invoking this routine, it will fail until one shot is no longer active.
*
*  Returns:
*  OK upon success, ERROR upon failure
*/
STATUS ipCtrbInitiate1shot (VXB_DEVICE_ID pDev, UINT8 channel, BOOL
trigger, BOOL extClk, BOOL edgeSel, BOOL intEnbl, UINT32 duration);
```

```
/******************************************************************************
*  ipCtrbResetChan
*
*  Reset IP-CTRB channel.
*  This routine will cancel any outstanding request for this channel.
*
*  Parameters:
*  pDev     -   Device ID returned in module list (libIpCtrbInit) specifying
*               which IP-CTRB module.
*  channel  -   Channel on the module to reset (0-7).
*
*  Returns:
*  OK upon success, ERROR upon failure.
*/
STATUS ipCtrbResetChan (VXB_DEVICE_ID pDev, UINT8 channel);

/******************************************************************************
*  ipCtrbAwaitInt
*
*  Await timer/counter interrupt on a channel of the IP-CTRB module.
*
*  Parameters:
*  pDev     -   Device id returned in module list (libIpCtrbInit) specifying
*                  which IP-CTRB module.
*  channel -   Channel of interest.
*  timeout  -   Timeout awaiting interrupt in usec.
*
*  Returns:
*  OK upon success, ERROR upon failure.
*/
STATUS ipCtrbAwaitInt (VXB_DEVICE_ID pDev, UINT8 channel, UINT32
timeout);
```

libIpSib API descriptions

```
/*******************************************************************************
*  libIpSibInit
*
*  Initialize library. This function must be invoked prior to utilizing any
*  of the following access routines.  This function returns a list of IP-SIB
*  modules.  Further all channels will be defaulted to SDC/SDT mode
*  configuration.
*
*  Parameters:
*  maxMods -   Maximum number of SIB modules to find.
*  modules  -   pointer to an array of size maxMods.
*
*  Returns:
*  Number of modules upon success, ERROR upon failure
*/
int libIpSibInit (int maxMods, VXB_DEVICE_ID *modules);

/*******************************************************************************
*  libIpSibExit
*
*  Exit/shutdown library. This function should be invoked upon application
*  termination.
*
*  Parameters:
*  modules:    - pointer to an array returned from libIpSibInit
*
*  Returns:
*  N/A void
*/
void libIpSibExit (VXB_DEVICE_ID *modules);
```

```
/*****************************************************************************
*  ipSibConfigCh
*
*  Configure IP-SIB channel
*
*  Parameters:
*  pDev     -   Module device id returned in module list (libIpSibInit)
*               specifying which IP-SIB module.
*  channel -   Channel on the module to configure (0-1)
*  mode     -   SIB mode of operation (0=SDC/SDT, 1=USIP/USOP)
*  ctsPol   -   Polarity of cts signal (0=active high, 1=active low)
*               (Don't care for SDC/SDT mode, CTS is disabled).
*
*  Returns:
*  OK upon success, ERROR upon failure
*/
STATUS ipSibConfigCh (VXB_DEVICE_ID pDev, UINT8 channel, UINT32 mode,
                                                          UINT32 ctsPol);

/*****************************************************************************
*  ipSibRead
*
*  Read from a SIB channel.
*
*  Parameters:
*  pDev     -   Module device id returned in module list (libIpSibInit)
*               specifying which IP-SIB module.
*  channel -  Read channel.
*  *buf      -  Buffer of size count.
*  count     -  Number of words to read.
*  timeout  -  Non-zero value implies blocking read of duration timeout
*               msec. If 0 is specified, non-blocking read
*
*  Special Considerations:
*  Channel must be configured prior to initiating a read or write.
*  Maximum read/write size is 511 bytes
*
*  Returns:
*  Count of words read, or ERROR upon failure.
*/
int ipSibRead (VXB_DEVICE_ID pDev, UINT8 channel, UINT16 *buf, int count,
                                                          UINT32 timeout);
```

```
/*****************************************************************************
*  ipSibWrite
*
*  Write to a SIB channel.
*
*  Parameters:
*  pDev     -   Module device id returned in module list (libIpSibInit)
*               specifying which IP-SIB module.
*  channel -   Write channel.
*  *buf      -   Buffer of size count.
*  count     -   Number of words to write.
*  timeout  -   Non-zero value implies blocking write of duration timeout
*               msec. If 0 is specified, non-blocking write.
*
*  Special Considerations:
*  Channel must be configured prior to initiating a read or write.
*  Maximum read/write size is 511 bytes
*
*  Returns:
*  Count of words written, ERROR upon failure.
*/
int ipSibWrite (VXB_DEVICE_ID pDev, UINT8 channel, UINT16 *buf, int count,
                                                        UINT32 timeout);

/*****************************************************************************
*  ipSibResetChan
*
*  Reset IP-SIB channel.
*
*  Parameters:
*  channel  -   Channel on the module to reset (0-1).
*
*  Returns:
*  0 upon success, < 0 upon error (standard Linux errno).
*/
int ipSibResetChan (VXB_DEVICE_ID pDev, UINT8 channel);
```

libIpOpto API descriptions

```
/*******************************************************************************
*  libIpOptoInit
*
*  Initialize library. This function must be invoked prior to utilizing any
*  of the following access routines.  This function returns a list of IP-OPTO
*  modules.  The counter (CTB) will be started during initialization.
*
*  Parameters:
*  maxMods  -   Maximum number of OPTO modules to find
*  modules  -   pointer to an array of size maxMods

*  Returns:
*  Number of modules upon success, ERROR upon failure
*/
int libIpOptoInit (int maxMods, VXB_DEVICE_ID *modules);

/*******************************************************************************
*  libIpOptoExit
*
*  Exit/shutdown library. This function should be invoked upon application
*  termination.
*
*  Parameters:
*  modules:    - pointer to an array returned from libIpOptoInit
*
*  Returns:
*  N/A void
*/
void libIpOptoExit (VXB_DEVICE_ID *modules);
```

```
/*****************************************************************************
*  ipOptoCfgWavefm
*
*  Configure and initiate or terminate waveform generation (CTA).  Once
*  waveform is configured and enabled, it can be utilized for FET control
*  and/or interrupt generation.  Interrupt generation and FET switching
*  will occur at the rate of period/2.
*
*  Parameters:
*  pDev      -   Module device id returned in module list (libIpOptoInit)
*                specifying which IP-OPTO module.
*  wavEnbl  -   Enable/disable waveform generation
*                (IPOPTO_ENBL or IPOPTO_DISABLE)
*  period   -   Period of waveform in usec
*                (Don't care if wav_enbl == IPOPTO_DISABLE);
*
*  Special Considerations:
*  The function will fail if waveform generation is currently active and
*  waveform generation is currently active.  To modify waveform, it first
*  must be idle.
*  ipopto_await_int maybe utilzed to determine interrupt assertion.
*
*  Returns:
*  OK upon success, ERROR upon failure
*/
STATUS ipOptoCfgWavefm (VXB_DEVICE_ID pDev, ipOptoEnbl_t wavEnbl,
                                                            UINT32 period);
```

```
/****************************************************************************
*  ipOptoFetCtrl
*
*  This function configures 1 or more FET channels.  The channel(s) may
*  operate in manual or waveform driven mode.  Manual mode can
*  enable/disable FET (on/off).
*  In auto mode, FET switching is driven by waveform configured via
*  ipOptoCfgWavefm.
*
*  Parameters:
*  pDev     -  Module device id returned in module list (libIpOptoInit)
*              which IP-OPTO module.
*  chanMsk  -  Bit mask specifying which channels to configure
*             e.g. 0x8001 means configure channels 15 and 0.
*  modeMsk  -  Bit mask specifying mode for specified channels.
*             e.g. 0x0001 = Channel 15 : Manual
                          Chanel   0 : Auto (waveform driven)
*  enblMsk  -  Bit mask specifying on/off (Don't care for auto mode).
*             e.g. 0x1001 == 0x1000, Channel 0 on.
*
*  Special Considerations:
*  If auto mode specified (waveform driven), ipOptoCfgWavefm must be invoked
*  prior to this function for successful execution.  Otherwise call will fail.
*
*  Returns:
*  Bit mask status, 0 returned on success, !0 upon failure.
*  e.g. for example above.
*  0x0001 = config failed for channel 0
*/
UINT16 ipOptoFetCtrl (VXB_DEVICE_ID pDev, UINT16 chanMsk,
                                         UINT16 modeMsk, UINT16 enblMsk);
```

```
/*****************************************************************************
*  ipOptoAwaitInt
*
*  Await timer/counter interrupt from CTA (if waveform generation is enabled).
*  This function will enable interrupt generation when invoked, and disable
*  interrupt generation upon exit.
*
*  Parameters:
*  pDev     -  Module device id returned in module list (libIpOptoInit)
*              which IP-OPTO module.
*  timeout  -  Timeout awaiting interrupt in msec.
*
*  Special Considerations:
*  If waveform generation has not been enabled, this function will immediately
*  fail and return an error.
*
*  Returns:
*  OK upon success, ERROR upon failure.
*/
STATUS ipOptoAwaitInt (VXB_DEVICE_ID, UINT32 timeout);

/*****************************************************************************
*  ipOptoGetCounter
*
*  This function reads the current 32 bit counter value (CTB).  The counter
*  is automatically initiated during initialization.  This value is
*  converted to usec based upon IP BUS speed setting.  This counter can
*  be reset upon read completion via the reset parameter.
*
*  Parameters:
*  pDev     -  Module device id returned in module list (libIpOptoInit)
*              which IP-OPTO module.
*  reset    -  IPOPTO_ENABLE (reset) or IPOPTO_DISABLE (don't reset).
*              which IP-OPTO module.
*
*  Returns:
*  Counter value in usecs.
*/
int ipOptoGetCounter (VXB_DEVICE_ID, ipOptoEnbl_t reset);
```

Installation

Copy the tar ball containing this driver to the ${WIND_BASE}/target/3rdparty/dyneng directory of your project. If a dyneng directory does not exist, create one. After extraction you should find the following files in the dyneng tree:

Makefile
ipack
    40dePcie3IP.cdf 40vxbIpackLib.cdf dePcie3IP.c dePcie3IP.dc dePcie3IP.dr dePcie3IP.h Makefile vxbIpackControlGet.c vxbIpackControlGet.mk vxbIpackLib.c vxbIpackLib.dc vxbIpackLib.dr vxbIpackLib.h README release_notes.txt
ipack/apps
    ipIoApp.c libIphv.c libIphv.h libIpCtrb.c libIpCtrb.h ip1ShotApp.c ipTimerApp.c libIpSib.c libIpSib.h ipSibApp.c libIpOpto.h libIpOpto.c ipOptoApp.c Makefile

The README file contained in ipack directory details the configuration and build steps required to include this package in your VxWorks image. This information is specifically not included in this document to avoid conflicts that may occur due to updates in the source tree.

Sample applications

libIphv

The application ipIoApp.c demonstrates proper usage of library functions/operations for both vxbIpackLib and libIphv. As previously mentioned, the Dynamic Engineering IP-Parallel-HV module is employed for demonstration purposes. The application verifies proper operation of this module in a VxWorks environment.

Load libIphv.o and ipIoApp.o as directed in the README file

Invocation parameters (iploApp)

The application can be run either as a single instance (one instance performs reads and writes), or two instances, one reader, one writer demonstrating simultaneous port operation. The application can be invoked from either the terminal or telnet session.

Sample application invocation is as follows:

Single instance invocation:
ipIo "b"
Two instances (two shells)
ipIo "r" (start first)
ipIo w (start within 10 seconds)

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.

libIpCtrb

The applications ipTimer.c and ip1ShotApp.c demonstrates proper usage of library functions/operations for both vxbIpackLib and libCtrb in a VxWorks environment.

Load libIpCtrb.o, ipTimerApp.o and ip1ShotApp.o as directed in the README file

Invocation parameters (ipTimerApp, ip1ShotApp)

The applications can be run either standalone or in conjunction with a Dynamic Engineering Test fixture. In standalone mode, only internal clock and triggering can be demonstrated/validated.

The test fixture generates an external clock, and propagates an external trigger pulse. If used to validate external trigger functionality, only 1 channel can be tested at a time. **Further, the value EXT_FIXTURE must be defined (top of source file libIpCtrb.c) to utilize the external test fixture for external triggers. In normal operation, EXT_FIXTURE must be undefined or #undef EXT_FIXTURE.**

Note: ipack device Ids may be determined by invoking the command vxBusShow

The following is partial output from that command:

```
 IPACK_Bus @ 0x00362898 with bridge @ 0x00370918
   Device Instances:
```

Orphan Devices:

ipackBus unit 0 on IPACK_Bus @ 0x00370a18 with busInfo 0x00000000

ipackBus unit 0 on IPACK_Bus @ 0x00370b18 with busInfo 0x00000000

In this example, 2 modules are installed, device IDs are 0x00370a18 and 0x00370b18

Application invocation is as follows:

ipTimerApp invocation:

ipTimer deviceId clock(0=internal|1=external, mode(0=norm|1=reload), duration(usec), num_chnls(1|8), channel

For example,
ipTimer 0x00370a18,0,0,1000,1,0

The application will exercise the timer functionality on module 0x00370a18, channel 0 using the internal clock in normal mode with a timer duration of 1 msec. The application will execute 500,000 iterations by first initiating a timer, then awaits the corresponding completion interrupt. The app will continue until until a failure is detected or interrupted with a <CR> from the shell

ipTimer 0x00370a18,0,0,1000,8
Same test as above, except all 8 channels are executed.

Ip1ShotApp invocation:

Ip1Shot deviceId clock(0=internal,1=external, trigger(0=internal|1=external), edgeSel(0=falling|1=rising), num_chnls(1|8), channel

For example,
Ip1Shot 0x00370a18,0,0,1,1000,1,7

The application will exercise the 1-shot functionality on module 0x00370a18, channel 7 using the internal clock, internal trigger, rising edge with a pulse width of 1 msec. The application will execute 500,000 iterations by first initiating a 1-shot, then awaits the corresponding completion interrupt. The app will continue until until a failure is detected or interrupted with a <CR> from the shell

Note: 1-shot can be run for all 8 channels as above with the app, however, only 1 port can be run when specifying external trigger, otherwise the app will fail.

libIpSib

The application ipSib.c demonstrates proper usage of library functions/operations for both vxbIpackLib and libSib in a VxWorks environment as well as validating HW functionality.
Load libIpSib.o, and ipSibApp.o as directed in the README file.

Invocation parameters (ipSibApp)

The appropriate Dynamic Engineering test fixture must be attached to successfully execute the application. There are 2 different fixtures depending upon mode under test. One fixture is utilized for SDC/SDT mode and another is required for USIP/USOP mode testing. Both channels can be tested simultaneously in USIP/USOP mode, however only 1 channel may be tested at time in SDC/SDT mode. **Further, the value EXT_FIXTURE must be defined (top of source file libIpSib.c) when external fixtures are employed**

Note: ipack device Ids may be determined by invoking the command vxBusShow

The following is partial output from that command:

```
 IPACK_Bus @ 0x00362898 with bridge @ 0x00370918

   Device Instances:

   Orphan Devices:

      ipackBus unit 0 on IPACK_Bus @ 0x00370a18 with busInfo 0x00000000

      ipackBus unit 0 on IPACK_Bus @ 0x00370b18 with busInfo 0x00000000
```

In this example, 2 modules are installed, device IDs are 0x00370a18 and 0x00370b18.

Application invocation is as follows:

ipSibApp invocation:

Two instances of the application must be invoked to validate either mode of operation. The first instance started is the reader and is invoked as follows for USIP/USOP mode:

```
ipSib deviceId, channel(0|1) reader(1=reader|0=writer),
mode(0=SDC/SDT|1=USOP/USIP),   cts polarity(0=active high|1=low), pkt
len(1-511)
```

For example, initiate reader executing in USOP/USIP mode with active high polarity and packet length of 256 on channel 0:

```
ipSib 0x00370b18,0,1,1,0,256
```

Initiate the writer in another terminal with same channel, mode, cts polarity, and packet length parameters within 5 seconds of starting reader:

```
ipSib 0x00370b18,0,0,1,0,256
```

The applications will exercise the SIB functionality on module 0x00370b18. The writer alternates packet data patterns. The reader validates data integrity upon reception of each packet. The apps will continue execution until an error is encountered, the iteration count is achieved, or interrupted via <CR> from the keyboard.

Invocation parameters (ipSibApp)

The appropriate Dynamic Engineering test fixture must be attached to successfully execute the application. There are 2 different fixtures depending upon mode under test. One fixture is utilized for SDC/SDT mode and another is required for USIP/USOP mode testing. Both channels can be tested simultaneously in USIP/USOP mode, however only 1 channel may be tested at time in SDC/SDT mode. **Further, the value EXT_FIXTURE must be defined (top of source file libIpSib.c) when external fixtures are employed**

Note: ipack device Ids may be determined by invoking the command vxBusShow

The following is partial output from that command:

```
 IPACK_Bus @ 0x00362898 with bridge @ 0x00370918
   Device Instances:
```

Orphan Devices:

ipackBus unit 0 on IPACK_Bus @ 0x00370a18 with busInfo 0x00000000

ipackBus unit 0 on IPACK_Bus @ 0x00370b18 with busInfo 0x00000000

In this example, 2 modules are installed, device IDs are 0x00370a18 and 0x00370b18.

Application invocation is as follows:

ipSibApp invocation:

Two instances of the application must be invoked to validate either mode of operation. The first instance started is the reader and is invoked as follows for USIP/USOP mode:

ipSib deviceId, channel(0|1) reader(1=reader|0=writer), mode(0=SDC/SDT|1=USOP/USIP), cts polarity(0=active high|1=low), pkt len(1-511)

For example, initiate reader executing in USOP/USIP mode with active high polarity and packet length of 256 on channel 0:

```
ipSib 0x00370b18,0,1,1,0,256
```

Initiate the writer in another terminal with same channel, mode, cts polarity, and packet length parameters within 5 seconds of starting reader:

```
ipSib 0x00370b18,0,0,1,0,256
```

The applications will exercise the SIB functionality on module 0x00370b18. The writer alternates packet data patterns. The reader validates data integrity upon reception of each packet. The apps will continue execution until an error is encountered, the iteration count is achieved, or interrupted via <CR> from the keyboard.

libIpOpto

The application ipOptoApp.c demonstrates proper usage of library functions/operations for both vxbIpackLib and libIpOpto in a VxWorks environment.

Load libIpOpto.o, ipOptoApp.o as directed in the README file

Invocation parameters (ipOptoApp)

The application must be run in conjunction with a Dynamic Engineering Test fixture. The test fixture enables visual inspection of proper FET switching via LEDs populating the fixture.

Application invocation is as follows:

> **ipOptoApp invocation**:
>
> ipOpto modNum, mode(0=waveform|1=manual), period (usec, don't care for manual mode)
>
> Manual mode validation:
> ipOpto 0,1,0 (assuming carrier is populated with one IP-OPTO module)
>
> You should observe each LED cycle on/off beginning with LED 0 (channel 0) every ½ second. This will continue for 60 iterations or until aborted via <CR>.
>
> Waveform mode validation:
> ip_opto 0, 0, 250000 (period in usec)
>
> > Maximum period is approximately 134 seconds, minimum period is 10 usec.
>
> Two alternate patterns are executed. Every other LED will be lit for both patterns. One pattern begins with LEDs 0,2,4,… being waveform driven Other pattern, LEDs 1,3,5,.. are waveform controlled. LEDs not waveform driven are disabled. Waveform driven LEDs are cycled on/off twice based upon the specified period, then the next pattern is executed. This cycle is repeated 60 iterations or until aborted via <CR>.

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

Software support contracts are available to update, add features, change for different revisions of OS etc. Please contact Dynamic Engineering for these options.

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