The Embedded I/O Company

TEWS TECHNOLOGIES

CARRIER-SW-72

LynxOS Device Driver

IPAC Carrier

Version 1.3.x

User Manual

Issue 1.3.0

September 2009

TEWS TECHNOLOGIES GmbH

Am Bahnhof 7 25469 Halstenbek, Germany

Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19

e-mail: info@tews.com www.tews.com

CARRIER-SW-72

LynxOS Device Driver

IPAC Carrier

Issue	Description	Date
1.0	First Issue	December 15, 2003
1.1	Generic IPAC Device Driver	January 14, 2004
1.2.0	TVME8xxx IPAC Carrier Support	April 5, 2005
1.2.1	New Address TEWS LLC	July 17, 2008
1.3.0	Address TEWS LLC removed, Corrections	September 11, 2009

Table of Contents

1 INTRODUCTION 4
2 INSTALLATION 5
2.1 Carrier Class Driver Installation 6
2.1.1 Build the Carrier Class Driver Object 6
2.1.2 Create Device Information Declaration 6
2.1.3 Modify the Device and Driver Configuration File 7
2.1.4 Rebuild the Kernel 7
2.2 VME Carrier Driver Installation 8
2.2.1 Build the VME Carrier Driver Object 8
2.2.2 Create Device Information Declaration 8
2.2.3 Modify the Device and Driver Configuration File 9
2.2.4 Rebuild the Kernel 9
2.3 TEWS PCI Carrier Driver Installation 10
2.3.1 Build the TEWS PCI Carrier Driver Object 10
2.3.2 Create Device Information Declaration 10
2.3.3 Modify the Device and Driver Configuration File 11
2.3.4 Rebuild the Kernel 11
2.4 SBS PCI Carrier Driver Installation 12
2.4.1 Build the SBS PCI Carrier Driver Object 12
2.4.2 Create Device Information Declaration 12
2.4.3 Modify the Device and Driver Configuration File 13
2.4.4 Rebuild the Kernel 13
3 CUSTOMER IPAC CARRIER SUPPORT 14
4 GENERIC IPAC DEVICE DRIVER 16
4.1 Installation 16
4.2 Device Driver Installation 17
4.2.1 Static Installation 17
4.2.1.1 Build the driver object 17
4.2.1.2 Create Device Information Declaration 17
4.2.1.3 Modify the Device and Driver Configuration File 17
4.2.1.4 Rebuild the Kernel 18
4.2.2 Dynamic Installation 19
4.2.2.1 Build the driver object 19
4.2.2.2 Create Device Information Declaration 19
4.2.2.3 Uninstall dynamic loaded driver 19
4.2.3 Bind IPAC modules 20
4.2.4 Configuration File: CONFIG.TBL 22
4.3 Generic IPAC Driver Programming 23
4.3.1 open() 23
4.3.2 close() 24
4.3.3 read() 25
4.3.4 write() 28
4.3.5 ioctl() 31
4.3.5.1 GEN_IPAC_RESET_SLOT 32
4.3.5.2 GEN_IPAC_MOD_INFO 33
5 APPENDIX 34
5.1 Supported IPAC Carrier Boards 34
5.2 Enumeration of IPAC slots 34
5.3 Diagnostic 35

1 Introduction

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device driver which work with any supported carrier board, TEWS TECHNOLOGIES has designed a software architecture that hides all of these carrier board differences under a well defined interface.

Basically the IPAC and carrier device drivers are implemented with a three level driver stacking. The carrier port driver is the lowest level. It handles the implementation details of the IPAC carrier board. The carrier class driver at the second level includes the management of IPAC slots and modules and provides a common interface between the IPAC driver and the carrier board driver. At the highest level resides the IPAC device driver.

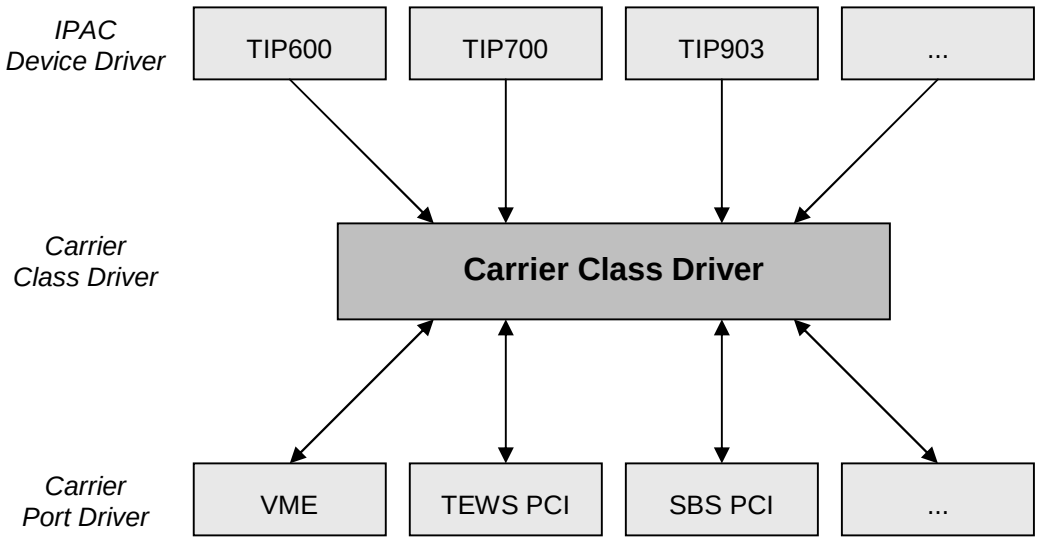

Figure 1: Stacked Driver Architecture

2 Installation

Usually the software is delivered together with the IPAC port driver.

The directory CARRIER-SW-72 on the distribution media contains the following files and directories:

CARRIER-SW-72-1.3.0.pdf	This manual in PDF format
CARRIER-SW-72-SRC.tar.gz	TAR archive with driver source code
ChangeLog.txt	Release history
Release.txt	Release information

The archive CARRIER-SW-72-SRC.tar contains the following files and directories:

ipac_carrier/ipac_carrier.h	Common include file
ipac_carrier/carrier_class	Sub-directory with carrier class driver sources
ipac_carrier/carrier_vme	Sub-directory with VMEbus carrier port driver sources
ipac_carrier/carrier_tews_pci	Sub-directory with TEWS PCI carrier port driver sources
ipac_carrier/carrier_sbs_pci	Sub-directory with SBS PCI carrier port driver sources
ipac_carrier/generic_ipac	Sub-directory with Generic IPAC port driver sources

In order to perform an installation, extract all files of the TAR archive CARRIER-SW-72-SRC.tar to a temporary directory.

Depending on the system configuration the following carrier driver components must be installed:

carrier_class	Always required
carrier_vme	Required for all VMEbus IPAC Carriers from TEWS TECHNOLOGIES (e.g. TVME200), SBS TECHNOLOGIES and other Vendors.
carrier_tews_pci	Required for all PCI and Compact PCI IPAC Carrier from TEWS TECHNOLOGIES (e.g. TPCI200, TPCI100, TCP201, ...) and VMEbus PowerPC Boards with local IPAC Carrier (TVME8240, TVME8300, TVME230).
carrier_sbs_pci	Required for all PCI and Compact PCI IPAC Carrier from SBS TECHNOLOGIES (e.g. cPCI100, PCI40, ...).
generic_ipac	Optional component. Can be installed to access non TEWS TECHNOLOGIES IPAC modules or as template for writing own device drivers, which uses the IPAC Carrier Device Driver.

The following chapters demonstrate the installation process of each carrier driver component in detail.

The IPAC carrier driver supports only the static installation method.

2.1 Carrier Class Driver Installation

The *carrier_class* driver includes the management of IPAC slots and modules and provides a common interface between the IPAC port driver and the carrier board driver. This carrier driver component is always required and must be the first entry from all carrier driver components in the CONFIG.TBL. It's absolute necessary that the carrier class driver is started first by system.

In order to install the *carrier_class* driver the following steps must be executed.

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/carrier_class or /sys/drivers.cpci_x86/carrier_class or...

2. Copy the following files from the carrier_class directory to this directory:
   - carrier_class.c
   - Makefile

3. Copy ipac_carrier.h to /usr/include/

4. Copy carrier_class_info.c to /sys/devices.xxx or */sys/devices* if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy carrier_class_info.h to /sys/dheaders/

6. Copy carrier_class.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ...

2.1.1 Build the Carrier Class Driver Object

1. Change to the directory /sys/drivers.xxx/carrier_class, where xxx represents the BSP that supports the target hardware.

2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... carrier_class_info.x
   ```

   And at the end of the Makefile

   ```
   carrier_class_info.o:$(DHEADERS)/carrier_class_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file but before other carrier driver components and IPAC port drivers (tipxxx.cfg).

   ```
   I:carrier_class.cfg
   ```

2.1.4 Rebuild the Kernel

If other carrier driver components will be also installed, rebuilding of the kernel image can be done after installation of all other components.

1. Change to the directory /sys/lynx.os *(/sys/bsp.xxx*)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

2.2 VME Carrier Driver Installation

The *carrier_vme* driver is required for all VMEbus IPAC carriers such as TEWS TECHNOLOGIES TVME200, TVME201 and so on. Because the IPAC slots must be configured manually it can also be used for other non Plug and Play buses (e.g. ISA IPAC carrier).

How to add carrier slots to the *carrier_vme* driver is described in detail in **chapter 3 - *Customer IPAC Carrier Support*.**

The *carrier_vme* driver must be included after the *carrier_class* driver entry **and** after the VMEbus driver (e.g. uvme) in CONFIG.TBL.

In order to install the *carrier_vme* driver the following steps must be executed.

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/carrier_vme or /sys/drivers.cpci_x86/carrier_vme or...

2. Copy the following files from the *carrier_vme* directory to this directory:
   - carrier_vme.c
   - Makefile

3. Copy carrier_vme_info.c to /sys/devices.xxx or */sys/devices* if /sys/devices.xxx does not exist (xxx represents the BSP).

4. Copy carrier_vme_info.h to /sys/dheaders/

5. Copy carrier_vme.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ...

2.2.1 Build the VME Carrier Driver Object

1. Change to the directory /sys/drivers.xxx/carrier_vme, where xxx represents the BSP that supports the target hardware.

2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.2.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... carrier_vme_info.x
   ```

   And at the end of the Makefile

   ```
   carrier_vme_info.o:$(DHEADERS)/carrier_vme_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.2.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry after the *carrier_class* driver entry.

   ```
   I:carrier_vme.cfg
   ```

2.2.4 Rebuild the Kernel

If other carrier driver components will be also installed, rebuilding of the kernel image can be done after installation of all other components.

1. Change to the directory /sys/lynx.os *(/sys/bsp.xxx*)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

2.3 TEWS PCI Carrier Driver Installation

The *carrier_tews_pci* driver is required for all TEWS TECHNOLOGIES PCI and Compact PCI IPAC Carrier such as TPCI100, TCP201 and so on. Because the carrier_tews_pci driver is a Plug and Play driver, no additional configuration is necessary.

The *carrier_tews_pci* driver must be included after the *carrier_class* driver entry in CONFIG.TBL.

In order to install the *carrier_tews_pci* driver the following steps must be executed.

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/carrier_tews_pci or /sys/drivers.cpci_x86/carrier_tews_pci or...

2. Copy the following files from the *carrier_tews_pci* directory to this directory:
   - carrier_tews_pci.c
   - tews_pci.h
   - Makefile

3. Copy carrier_tews_pci_info.c to /sys/devices.xxx or */sys/devices* if /sys/devices.xxx does not exist (xxx represents the BSP).

4. Copy carrier_tews_pci_info.h to /sys/dheaders/

5. Copy carrier_tews_pci.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ...

2.3.1 Build the TEWS PCI Carrier Driver Object

1. Change to the directory /sys/drivers.xxx/carrier_tews_pci, where xxx represents the BSP that supports the target hardware.

2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.3.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... carrier_tews_pci_info.x
   ```

   And at the end of the Makefile

   ```
   carrier_tews_pci_info.o:$(DHEADERS)/carrier_tews_pci_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.3.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry after the carrier_class driver entry.

   ```
   I:carrier_tews_pci.cfg
   ```

2.3.4 Rebuild the Kernel

If other carrier driver components will be also installed, rebuilding of the kernel image can be done after installation of all other components.

1. Change to the directory /sys/lynx.os *(/sys/bsp.xxx*)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

2.4 SBS PCI Carrier Driver Installation

The *carrier_sbs_pci* driver is required for all SBS TECHNOLOGIES PCI and Compact PCI IPAC Carrier such as cPCI100, PCI40 and so on. Because the carrier_sbs_pci driver is a Plug and Play driver, no additional configuration is necessary.

The *carrier_sbs_pci* driver must be included after the *carrier_class* driver entry in CONFIG.TBL.

In order to install the *carrier_sbs_pci* driver the following steps must be executed.

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/carrier_sbs_pci or /sys/drivers.cpci_x86/carrier_sbs_pci or...

2. Copy the following files from the *carrier_sbs_pci* directory to this directory:
   - carrier_sbs_pci.c
   - sbs_pci.h
   - Makefile

3. Copy carrier_sbs_pci_info.c to /sys/devices.xxx or */sys/devices* if /sys/devices.xxx does not exist (xxx represents the BSP).

4. Copy carrier_sbs_pci_info.h to /sys/dheaders/

5. Copy carrier_sbs_pci.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ...

2.4.1 Build the SBS PCI Carrier Driver Object

1. Change to the directory /sys/drivers.xxx/carrier_sbs_pci, where xxx represents the BSP that supports the target hardware.

2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.4.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... carrier_sbs_pci_info.x
   ```

   And at the end of the Makefile

   ```
   carrier_sbs_pci_info.o:$(DHEADERS)/carrier_sbs_pci_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

2.4.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry after the *carrier_class* driver entry.

   ```
   I:carrier_sbs_pci.cfg
   ```

2.4.4 Rebuild the Kernel

If other carrier driver components will be also installed, rebuilding of the kernel image can be done after installation of all other components.

1. Change to the directory /sys/lynx.os *(/sys/bsp.xxx*)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

3 Customer IPAC Carrier Support

If the used IPAC carrier isn't supported by the carrier port drivers on the distribution diskette and the carrier board is a PCI bus carrier please contact TEWS TECHNOLOGIES.

Usually we will implement the carrier port driver without any charge within a few days.

If the carrier board is VMEbus based you can create IPAC slot entries in the VME carrier port driver.

To add IPAC slots, change to the installation directory of the device information file carrier_vme_info.c (usally /sys/devices.xxx). Open the file carrier_vme_info.c with an appropriate editor and add new entries in the array carrier_vme_info [] after the comment /* Please add slot entries here! */.

The creation of new slot entries is very easy. Please copy and paste an entry from the example and change address and interrupt parameter as necessary. Be sure using always virtual addresses! All fields are described in detail in the structure definition below.

A slot entry must be created for each used slot. Be sure that slot entries were created only for used slots, otherwise a bus error occurs and the VME carrier driver is removed from the system.

After modification you have to rebuild the device information library and the kernel image.

Using IPAC modules on VMEbus IPAC carrier requires installation of an appropriate LynxOS VMEbus driver. For example Tundra Universe based VMEbus bridges requires the uvmedrvr. For installation uncomment appropriate entries (I:uvme.cfg) in the CONFIG.TBL in the BSP directory. Be sure that used PCI slave windows (VMEbus master windows) are enabled (check file /dheaders/uvmeinfo.h). For further information please read the related man pages (uvmedrvr.4 and vmedrvr.4).

The structure **CARRIER_VME_INFO** has the following layout:

```
typedef struct {
        int             slot_index;
        unsigned long   ID_virt_address;
        unsigned long   ID_space_size;
        unsigned long   IO_virt_address;
        unsigned long   IO_space_size;
        unsigned long   MEM_virt_address;
        unsigned long   MEM_space_size;
        int             system_interrupt_vector;
        int             module_interrupt_vector;
        int             interrupt_level;
} CARRIER_VME_INFO;
```

slot_index

Zero-based slot position on a carrier e.g. 0 for slot A, 3 for slot D.

ID_virt_address

Specifies the absolute virtual address of the IPAC ID space of this slot.

ID_space_size

Specifies the size of the ID space in bytes.

IO_virt_address

Specifies the absolute virtual address of the IPAC IO space of this slot.

IO_space_size

Specifies the size of the IO space in bytes.

MEM_virt_address

Specifies the absolute virtual address of the IPAC MEM space of this slot.

MEM_space_size

Specifies the size of the MEM space in bytes.

system_interrupt_vector

Interrupt vector used to register the ISR in the kernel.

module_interrupt_vector

Interrupt vector used to setup the vector register of the IPAC module. Usually this vector is equal to the *system_interrupt_vector* above.

interrupt_level

This parameter is reserved for future use and can be set to 0.

4 Generic IPAC Device Driver

The generic IPAC device driver is an ordinary LynxOS device driver, which is stacked on the TEWS TECHNOLOGIES IPAC carrier driver.

The standard file (I/O) functions (open, close, read, write and ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

This driver can be used to access (read/write) all address spaces (IO, ID and MEM) of non TEWS TECHNOLOGIES IPAC modules via a well defined interface. If necessary this driver can be used as template for writing own device drivers with more complex I/O operations and interrupt handling.

This driver can handle any number of different IPAC modules. Supported IPAC modules can be specified by its manufacturer ID and model number in a variable length array in the file gen_ipac.c. See below (4.2.3) for further information how to add and configure new IPAC modules.

4.1 Installation

The software is delivered together with the IPAC Carrier Driver on a PC formatted 3½" HD diskette. Please refer to chapter 2 how to extract IPAC Carrier driver components.

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files from the *generic_ipac* directory to their target directories:

1. Create a new directory in the system drivers directory path /sys/drivers.xxx, where xxx represents the BSP that supports the target hardware.

   For example: /sys/drivers.pp_drm/gen_ipac or /sys/drivers.cpci_x86/gen_ipac

2. Copy the following files from the generic_ipac directory to this directory:
   - gen_ipac.c
   - gen_ipac_def.h
   - gen_ipac.import
   - Makefile

3. Copy gen_ipac.h to /usr/include/

4. Copy gen_ipac_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy gen_ipac_info.h to /sys/dheaders/

6. Copy gen_ipac.cfg to */sys/cfg.xxx/*, where xxx represents the BSP for the target platform

For example: /sys/cfg.ppc or /sys/cfg.x86 ....

Before building a new device driver, the TEWS TECHNOLOGIES IPAC carrier driver must be installed properly, because this driver includes the header file *ipac_carrier.h*, which is part of the IPAC carrier driver distribution. Please refer to chapter 2 in this manual

4.2 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

Both installation methods require the TEWS TECHNOLOGIES IPAC Carrier Driver. Please refer to chapter 2.

4.2.1 Static Installation

With this method, the driver object code is linked with the kernel routines and is installed during system start-up.

4.2.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/gen_ipac, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

4.2.1.2 Create Device Information Declaration

1. Change to the directory /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).
2. Add the following dependencies to the Makefile

   ```
   DEVICE_FILES_all = ... gen_ipac_info.x
   ```

   And at the end of the Makefile

   ```
   gen_ipac_info.o:$(DHEADERS)/gen_ipac_info.h
   ```

3. To update the library /sys/lib/libdevices.a enter:

   ```
   make install
   ```

4.2.1.3 Modify the Device and Driver Configuration File

In order to insert the driver object code into the kernel image, an appropriate entry in file CONFIG.TBL must be created.

1. Change to the directory /sys/lynx.os/ respective /sys/bsp.xxx, where xxx represents the BSP that supports the target hardware.
2. Create an entry at the end of the file CONFIG.TBL

   Insert the following entry at the end of this file. Be sure that the necessary TEWS TECHNOLOGIES IPAC carrier driver is included **before** this entry.

   ```
   I:gen_ipac.cfg
   ```

4.2.1.4 Rebuild the Kernel

1. Change to the directory /sys/lynx.os/ (/sys/bsp.xxx)
2. Enter the following command to rebuild the kernel:

   ```
   make install
   ```

3. Reboot the newly created operating system by the following command (not necessary for KDIs):

   ```
   reboot –aN
   ```

   The N flag instructs init to run mknod and create all the nodes mentioned in the new nodetab.

4. After reboot you should find the following new devices (depends on the device configuration): /dev/gen_ipac_0, /dev/gen_ipac_1, /dev/gen_ipac_2 …

4.2.2 Dynamic Installation

This method allows you to install the driver after the operating system is booted. The driver object code is attached to the end of the kernel image and the operating system dynamically adds this driver to its internal structures. The driver can also be removed dynamically.

4.2.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/gen_ipac, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make
   ```

4.2.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/gen_ipac, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this works only on native system)

   ```
   make ipac_info
   ```

3. To install the driver enter:

   ```
   drinstall –c gen_ipac.obj
   ```

   If successful, drinstall returns a unique <driver-ID>
4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> ipac_info
   ```

   The <driver-ID> is returned by the drinstall command
5. To create nodes for the devices enter:

   ```
   mknod /dev/gen_ipac_0 c <major_no> 0
   mknod /dev/gen_ipac_1 c <major_no> 1
   mknod /dev/gen_ipac_2 c <major_no> 2
   ...
   ```

   The <major_no> is returned by the devinstall command.

If all steps are successful completed the generic IPAC driver is ready to use.

4.2.2.3 Uninstall dynamic loaded driver

To uninstall the generic IPAC device driver enter the following commands:

```
devinstall –u –c <device-ID>
drinstall –u <driver-ID>
```

4.2.3 Bind IPAC modules

The generic IPAC carrier driver can be bind to any IPAC module, which is plugged on an IPAC carrier board supported by the TEWS TECHNOLOGIES IPAC carrier driver.

Every IPAC module is clear defined by a unique manufacturer ID and a manufacturer specific model number, which are located in the IDPROM area. To bind a certain IPAC module type to the generic IPAC driver a suitable entry in the *ipac_module_id array ipac_id* in the file *gen_ipac.c* must be created. This entry must contain the manufacturer ID and model number of the desired IPAC module(s) and configuration information for the underlying carrier slot(s).

To bind more than one IPAC module of the same type to the generic IPAC driver only one entry in the *ipac_module_id* array must be created. Each of the attached IPAC modules can be accessed by a different minor device.

To bind different IPAC modules to the generic IPAC driver separate entries in the *ipac_module_id* array must be created. Different minor devices will be created in the scanning order of this array.

In both cases the *ipac_module_id* array must be terminated by a zero entry (manufacturer_ID = 0).

The structure *ipac_module_id* has the following layout:

```
struct ipac_module_id {
        unsigned long       manufacturer;
        unsigned long       model_number;
        unsigned long       slot_config;
        unsigned long       mem_size;
        unsigned long       private_data;
};
```

manufacturer

Manufacturer ID of the desired IPAC module (offset 0x09 in the IDPROM).

model_number

Manufacturer specific model number (offset 0x0B in the IDPROM).

slot_config

Set of bit flags which specify properties of the IPAC module. This information will be used by the carrier class driver to setup the carrier slot.

Flag	Meaning
IPAC_INT0_EN	Enables IPAC interrupt line 0
IPAC_INT1_EN	Enables IPAC interrupt line 1
IPAC_EDGE_SENS	Interrupt detection is edge sensitive
IPAC_LEVEL_SENS	Interrupt detection is level sensitive (default)
IPAC_CLK_8MHZ	IPAC clock is 8 MHz (default)
IPAC_CLK_32MHZ	IPAC clock is 32 MHz
IPAC_MEM_8BIT	IPAC memory space is 8-bit wide (D0...D7 only)
IPAC_MEM_16BIT	IPAC memory space is 16-bit wide (default)

mem_size, private_data

Unused.

EXAMPLE

In the example configuration below the generic IPAC carrier driver will support two different IPAC module types from TEWS TECHNOLOGIES (manufacturer ID = 0xB3).

Both modules are running with 8 MHz IP-clock. The first type (model number = 0x1C) will support level sensitive interrupts at IP INT0. The second does not support interrupts.

```
/*****************************************************************************
**
**  Supported IPAC modules by this driver and their initialization values
**
*****************************************************************************/
struct ipac_module_id ipac_id[] = {
     {
          manufacturer:      0xB3,
          model_number:      0x1C,
          slot_config:       IPAC_INT0_EN | IPAC_LEVEL_SENS | IPAC_CLK_8MHZ,
          private_data:      0,
     },
     {
          manufacturer:      0xB3,
          model_number:      0x77,
          slot_config:       IPAC_CLK_8MHZ,
          private_data:      0,
     },

     /* Add addtional supported modules here */
     {
          manufacturer:      0,   /* end of list */
          model_number:      0,
     }
};
```

The ipac_module_id array must be modified in the driver source file gen_ipac.c. Please search for ipac_id to find the correct location within this file.

4.2.4 Configuration File: CONFIG.TBL

The device and driver configuration file CONFIG.TBL contains entries for device drivers and its major and minor device declarations. Each time the system is rebuild, the config utility read this file and produces a new set of driver and device configuration tables and a corresponding nodetab.

To install the generic IPAC driver and devices into the LynxOS system, the configuration include file gen_ipac.cfg must be included in the CONFIG.TBL (see also 4.2.1.3).

The file gen_ipac.cfg on the distribution disk contains the driver entry (*C:gen_ipac:*\....) and a major device entry (*D:GEN_IPAC:ipac_info::*) with 9 minor device entries (*"N: gen_ipac_0:0", ..., "N: gen_ipac_8:8"*).

If the driver should support more than nine IPAC modules, additional minor device entries must be added. To create the device node */dev/gen_ipac_9* the line *N:gen_ipac_9:9* must be added at the end of the file gen_ipac.cfg. For the next node a minor device entry with 10 must be added and so on.

This example shows the predefined driver entry:

```
#    Format:
#    C:driver-name:open:close:read:write:select:control:install:uninstall
#    D:device-name:info-block-name:raw-partner-name
#    N:node-name:minor-dev

C:gen_ipac:\
     :gen_ipac_open:gen_ipac_close:gen_ipac_read:gen_ipac_write:\
     ::gen_ipac_ioctl:gen_ipac_install:gen_ipac_uninstall
D:GEN_IPAC:ipac_info::
N:gen_ipac_0:0
N:gen_ipac_1:1
N:gen_ipac_2:2
N:gen_ipac_3:3
N:gen_ipac_4:4
N:gen_ipac_5:5
N:gen_ipac_6:6
N:gen_ipac_7:7
N:gen_ipac_8:8
```

The configuration above creates the following node in the /dev directory.

/dev/gen_ipac_0 ... /dev/gen_ipac_8

4.3 Generic IPAC Driver Programming

LynxOS system calls are all available directly to any C program. They are implemented as ordinary function calls to "glue" routines in the system library, which trap to the OS code.

Note that many system calls use data structures, which should be obtained in a program from appropriate header files. Necessary header files are listed with the system call synopsis.

4.3.1 open()

NAME

open() - open a file

SYNOPSIS

#include <sys/file.h>
#include <sys/types.h>
#include <fcntl.h>

int open (char *path, int oflags[, mode_t mode])

DESCRIPTION

Opens a file (generic IPAC driver device) named in path for reading and writing. The value of oflags indicates the intended use of the file. In case of a generic IPAC devices oflags must be set to O_RDWR to open the file for both reading and writing.

The mode argument is required only when a file is created. Because a generic IPAC device already exists this argument is ignored.

EXAMPLE

```
int  fd


fd = open ("/dev/gen_ipac_0", O_RDWR);
```

RETURNS

Open returns a file descriptor number if successful or 1 on error. The global variable *errno* contains the detailed error code.

4.3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device associated with the valid file descriptor handle fd.

EXAMPLE

```
int  result;

result = close(fd);
```

RETURNS

Close returns 0 (OK) if successful, or –1 on error. The global variable errno contains the detailed error code.

SEE ALSO

LynxOS System Call - close()

4.3.3 read()

NAME

read() - read from a file

SYNOPSIS

#include <gen_ipac.h>

int read (int fd, char *buff, int count)

DESCRIPTION

This read function reads the specified number of data items from an IPAC address space. The location must be specified by the IPAC address space and an offset within this address space.

A pointer to the callers read buffer (GEN_IPAC_RW_BUF) and the size of this structure is passed by the parameters buff and count to the device.

The GEN_IPAC_RW_BUF structure has the following layout:

```
typedef struct {
        int     space;
        int     size;
        int     offset;
        int     count;
        union {
                unsigned char       b[GEN_IPAC_MAX_BUF_UCHAR];
                unsigned short      w[GEN_IPAC_MAX_BUF_USHORT];
                unsigned long       l[GEN_IPAC_MAX_BUF_ULONG];
        } data;
} GEN_IPAC_RW_BUF, *PGEN_IPAC_RW_BUF;
```

space

This parameter specifies the IPAC address space to be accessed by read command.

Value	Description
GEN_IPAC_IO_SPACE	Access IPAC IO space
GEN_IPAC_ID_SPACE	Access IPAC ID space
GEN_IPAC_MEM_SPACE	Access IPAC MEM space

size

This parameter specifies the access width in bytes.

Value	Access Width
GEN_IPAC_8BIT	8-Bit access (unsigned char)
GEN_IPAC_16BIT	16-Bit access (unsigned short)
GEN_IPAC_32BIT	32-Bit access (unsigned long)

offset

This parameter specifies the byte address offset within the selected IPAC space.

count

This parameter specifies the number of items to read. In other word the number of 8-bit, 16-bit or 32-bit values to read. For example: if count is set to 3 and the access size is 4, 12 byte will be returned if the read operation was successful.

data

This union contains three different arrays for 8-bit, 16-bit and 32-bit access, which returns the read data items. The array b[] should be used for 8-bit (byte) access, the array w[] for 16-bit (word) access and the array l[] for 32-bit (long word) access. If the length of a certain array isn't suitable for the application, it's possible to adapt the size of the arrays as desired by modifying the header file gen_ipac.h and rebuilding the device driver and application program.

EXAMPLE

```
int  fd;
int  i;
int  NumBytes;
GEN_IPAC_RW_BUF  rwBuf;

rwBuf.space   = GEN_IPAC_ID_SPACE;
rwBuf.size    = GEN_IPAC_8BIT;
rwBuf.offset  = 0;
rwBuf.count   = 64;

NumBytes = read(fd, &rwBuf, sizeof(rwBuf));

if(NumBytes > 0) {
     for (i=0; i<64; i++) {
          printf("%02X ", rwBuf.data.b[i]);
     }
}
else {
     /* handle read error */
}
```

RETURNS

When read succeeds, the number of bytes read is returned. If read fails, -1 (SYSERR) is returned.

On error, errno will contain a standard read error code (see also LynxOS System Call – read) or one of the following driver specific error codes:

ENXIO	Illegal device
EINVAL	Invalid argument. This error is reported if the size of the read buffer is wrong or if the space or size parameters are out of range.
ERANGE	The requested data buffer is too large.
EACCES	An access error has occurred.

SEE ALSO

LynxOS System Call - read()

4.3.4 write()

NAME

write() – write to a file

SYNOPSIS

#include <gen_ipac.h>

int write (int fd, char *buff, int count)

DESCRIPTION

This write function writes the specified number of data items to an IPAC address space. The location must be specified by the IPAC address space and an offset within this address space.

A pointer to the callers write buffer (GEN_IPAC_RW_BUF) and the size of this structure are passed by the parameters buff and count to the device.

The GEN_IPAC_RW_BUF structure has the following layout:

```
typedef struct {
        int     space;
        int     size;
        int     offset;
        int     count;
        union {
                unsigned char       b[GEN_IPAC_MAX_BUF_UCHAR];
                unsigned short      w[GEN_IPAC_MAX_BUF_USHORT];
                unsigned long       l[GEN_IPAC_MAX_BUF_ULONG];
        } data;
} GEN_IPAC_RW_BUF, *PGEN_IPAC_RW_BUF;
```

space

This parameter specifies the IPAC address space to be accessed by write command.

Value	Description
GEN_IPAC_IO_SPACE	Access IPAC IO space
GEN_IPAC_ID_SPACE	Access IPAC ID space
GEN_IPAC_MEM_SPACE	Access IPAC MEM space

size

This parameter specifies the access width in bytes.

Value	Access Width
GEN_IPAC_8BIT	8-Bit access (unsigned char)
GEN_IPAC_16BIT	16-Bit access (unsigned short)
GEN_IPAC_32BIT	32-Bit access (unsigned long)

offset

This parameter specifies the byte address offset within the selected IPAC space.

count

This parameter specifies the number of items to write. In other word the number of 8-bit, 16-bit or 32-bit values to write. For example: if count is set to 8 and the access size is 2, 16 bytes will be written into the specified IPAC space.

data

This union contains three different arrays for 8-bit, 16-bit and 32-bit access, which contains the data to write. The array b[] should be used for 8-bit (byte) access, the array w[] for 16-bit (word) access and the array l[] for 32-bit (long word) access. If the length of a certain array isn't suitable for the application, it's possible to adapt the size of the arrays as desired by modifying the header file gen_ipac.h and rebuilding the device driver and application program.

EXAMPLE

```
int  fd;
int  i;
int  NumBytes;
GEN_IPAC_RW_BUF  rwBuf;

rwBuf.space   = GEN_IPAC_MEM_SPACE;
rwBuf.size    = GEN_IPAC_32BIT;
rwBuf.offset  = 0;
rwBuf.count   = 1;
rwBuf.data.l[0] = 0x12345678;

NumBytes = write(fd, &rwBuf, sizeof(rwBuf));

if(NumBytes < 0) {
     /* handle write error */
}
```

RETURNS

When write succeeds, the number of bytes actually written is returned. If write fails, -1 (SYSERR) is returned.

On error, errno will contain a standard write error code (see also LynxOS System Call – write) or one of the following driver specific error codes:

ENXIO	Illegal device
EINVAL	Invalid argument. This error is reported if the size of the write buffer is wrong or if the space or size parameters are out of range.
ERANGE	The requested data buffer is too large.
EACCES	An access error has occurred.

SEE ALSO

LynxOS System Call - read()

4.3.5 ioctl()

NAME

ioctl() - I/O device control

SYNOPSIS

#include <ioctl.h>
#include <gen_ipac.h>

int ioctl (int fd, int request, char *arg)

DESCRIPTION

ioctl provides a way of sending special commands to a device driver. The call sends the value of request and the pointer arg to the device associated with the descriptor fd.

The following ioctl codes are defined in gen_ipac.h :

Value	Meaning
GEN_IPAC_RESET_SLOT	Reset IPAC module
GEN_IPAC_MOD_INFO	Read module information data

See behind for more detailed information on each control code.

RETURNS

On success, zero is returned. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code. The generic IPAC driver ioctl function returns always standard error codes.

SEE ALSO

LynxOS System Call – ioctl() for detailed description of possible error codes.

4.3.5.1 GEN_IPAC_RESET_SLOT

NAME

GEN_IPAC_RESET_SLOT - Reset IPAC module

DESCRIPTION

This control function asserts the IP RESET# signal to reset the IPAC module which is attached to this device respective file descriptor. After 200 ms the onboard logic negates the IP RESET# signal and completes the IP RESET cycle.

This function is only available for TEWS TECHNOLOGIES IPAC carrier boards. For other IPAC carriers SYSERR is returned.

EXAMPLE

```
int  fd;
int  result;


result = ioctl(fd, GEN_IPAC_RESET_SLOT, NULL);

if (result < 0) {
     /* handle ioctl error */
}
```

ERRORS

EPERM	Resetting a single slot isn't supported by this IPAC carrier.

4.3.5.2 GEN_IPAC_MOD_INFO

NAME

GEN_IPAC_MOD_INFO - Read module information data

DESCRIPTION

This control function returns information data about the attached IPAC module, which can be used for instance to identify a certain IPAC module on a carrier board.

A pointer to the caller's buffer (GEN_IPAC_INFO) is passed by the argument pointer arg to the driver.

The GEN_IPAC_INFO structure has the following layout:

```
typedef struct {
    int     ManufacturerID;
    int     ModelNumber;
    int     SlotIndex;
} GEN_IPAC_INFO, *PGEN_IPAC_INFO;
```

ManufacturerID

Obtains the manufacturer ID of the attached IPAC module (read from offset 0x09 in the IDPROM).

ModelNumber

Obtains the model number of the attached IPAC module (read from offset 0x0B in the IDPROM).

SlotIndex

Obtains the location of the carrier board where the IPAC module is plugged (0 = slot A, 1 = slot B ...).

EXAMPLE

```
int  fd;
int  result;
GEN_IPAC_INFO  info;

result = ioctl(fd, GEN_IPAC_MOD_INFO, (char*)&info);

if (result < 0) {
     /* handle ioctl error */
}
```

5 Appendix

5.1 Supported IPAC Carrier Boards

The following TEWS TECHNOLOGIES and SBS IPAC carrier boards are supported:

Driver	Carrier Board	Descrition
carrier_tews_pci	TPCI100	PCI carrier for 2 IndustryPack modules
	TPCI200	PCI carrier for 4 IndustryPack modules
	TCP201	Compact PCI carrier for 4 IndustryPack modules
	TCP211	Compact PCI carrier for 2 IndustryPack modules
	TCP212	Compact PCI carrier for 2 IndustryPack modules
	TCP213	Compact PCI carrier for 2 IndustryPack modules
	TCP220	Compact PCI carrier for 4 IndustryPack modules
	TVME8240	On-board Carrier for 4 IndustryPack modules
	TVME8300	On-board Carrier for 4 IndustryPack modules
	TVME230	PCI expansion carrier for 4 IndustryPack modules
carrier_vme	TVMExxx	All TEWS TECHNOLOGIES VMEbus IPAC Carrier
	custom	Almost all standard VMEbus IPAC Carrier
carrier_sbs_pci	PCI40	PCI carrier for 4 IndustryPack modules
	cPCI100	Compact PCI carrier for 2 IndustryPack modules
	cPCI200	Compact PCI carrier for 4 IndustryPack modules

5.2 Enumeration of IPAC slots

If more than one IPAC module is installed, maybe on different carrier boards. It's sometimes necessary to know which device node belongs to a certain slot on a carrier board.

The search and allocation order of the carrier class driver is always deterministic and never accidental. Usually the PCI bus will be searched from lower buses to higher buses and from lower devices to higher devices.

On carrier boards the slots will be enumerated from lower slots to higher slots.

If different carrier boards are installed in the system the order depends on the start order of the carrier port drivers defined in the CONFIG.TBL.

5.3 Diagnostic

If your installed IPAC port driver (e.g. tip600) doesn't find any devices although the IPAC is properly plugged on a carrier port, it's interesting to know what's going on in the system.

Usually all TEWS TECHNOLOGIES device driver announced significant event or errors via the device driver routine kkprintf(). To enable the debug output you must define the macro DEBUG in the device driver source files (e.g. carrier_class.c, carrier_tews_pci.c, tip600.c,...).

The following output appears at the LynxOS debug console if the carrier and IPAC driver starts:

```
TEWS TECHNOLOGIES - IPAC Carrier Class Driver version 1.3.0 (2009-09-10)
TEWS TECHNOLOGIES - (Compact)PCI IPAC Carrier version 1.3.0 (2009-09-10)
IPAC_CC  :  carrier driver <TEWS TECHNOLOGIES - (Compact)PCI IPAC Carrier>
registered
IPAC_CC  :  UCHAR access  [byte lanes must be swapped]
IPAC_CC  :  USHORT access [byte lanes are correct]
IPAC_CC  :  ULONG access  [word lanes must be swapped]
IPAC_CC  :  IPAC (Manuf-ID=B3, Model#=04) recognized @ slot=0 carrier=<TEWS
TECHNOLOGIES - (Compact)PCI IPAC Carrier>
IPAC_CC  :  IPAC access failed (slot=1, carrier=<TEWS TECHNOLOGIES -
(Compact)PCI IPAC Carrier>)
IPAC_CC  :  carrier driver <TEWS TECHNOLOGIES - SBS (Compact)PCI IPAC
Carrier> registered
TIP600 Digital Input driver version 2.0.0 (2009-09-10)
IPAC_CC  :  IPAC driver <TIP600 Digital Input driver> registered
IPAC_CC  :  Call probe function of <TIP600 Digital Input driver> for module
[179/4]
TIP600  :  Probe new TIP600 mounted on <TEWS TECHNOLOGIES - (Compact)PCI
IPAC Carrier> at slot A
```

If you can't solve the problem by yourself, please contact TEWS TECHNOLOGIES with a detailed description of the error condition, your system configuration and the debug outputs.