The Embedded I/O Company

TEWS TECHNOLOGIES

CARRIER-SW-42

VxWorks Device Driver

IPAC Carrier

Version 4.0.x

User Manual

Issue 4.0.0

March 2017

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-42

VxWorks Device Driver

IPAC Carrier

Supported Modules:

TPCI100
TPCI200
TCP201
TCP211
TCP212
TCP213
TCP220
TPCE200
TAMC100
TAMC200
TAMC220
TVME200
TVME201
TVME210
TVME211
TVME220
TVME230
TVME8240A
TVME8300
PCI40
CPCI100/200

Issue	Description	Date
1.0.0	First Issue	September 16, 2005
1.1.0	Structure definition (ipac_resource) and example applications modified	November 22, 2005
1.2.0	SBS TECHNOLOGIES Carrier Card Support	January 11, 2006
1.2.1	Support of custom carrier boards	April 10, 2006
1.2.2	New Address TEWS TECHNOLOGIES LLC ChangeLog.txt added to file list	December 5, 2006
1.3.0	Type of Parameters in ipFindDevice() changed	June26, 2007
2.0.0	VxBus and SMP support	January 27, 2010
2.0.1	Legacy vs. VxBus Driver modified	March 26, 2010
3.0.0	VxWorks 64-Bit Support added	December 5, 2011
4.0.0	VxWork 7 GEN1+GEN2 support added	March 16, 2017

Table of Contents

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 drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so called carrier driver that hides all of these carrier board differences under a well-defined interface.

During the initialization phase the carrier driver will collect information of supported carrier boards and plugged IPAC modules in an internal data base. The data base contains address information for all IP spaces (IO, ID and MEM), the corresponding interrupt vector and level and additional information to identify plugged IP modules and the underlying carrier board.

All resource information necessary for IPAC module driver initialization can be retrieved from this data base by calling the ipFindDevice() function with appropriate arguments to specify the IPAC module we are looking for. If necessary, this function will enable interrupts on the carrier board (e.g. PCI carrier) and related system buses (PCIbus and VMEbus).

Due to the fact that the TEWS TECHNOLOGIES carrier driver and IPAC module drivers are independent, the carrier driver can also be used by custom drivers without any modification.

The CARRIER-SW-42 supports the modules listed below:

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
TPCE200	PCIe Carrier for 4 IndustryPack modules
TAMC100	AMC Carrier for 1 IndustryPack module
TAMC200	AMC Carrier for 3 IndustryPack modules
TAMC220	AMC Carrier for 3 IndustryPack modules
TVME200	VMEbus carrier for 4 IndustryPack modules
TVME201	VMEbus carrier for 4 IndustryPack modules
TVME210	VMEbus carrier for 2 IndustryPack modules
TVME211	VMEbus carrier for 2 IndustryPack modules
TVME220	VMEbus carrier for 4 IndustryPack modules
TVME230	PCI Expansion Card (SPAN) for 4 IndustryPack modules
TVME8240A	SBC with IndustryPack Interface
TVME8300	SBC with IndustryPack Interface
PCI40	SBS PCI carrier for 4 IndustryPack modules
CPCI100/200	SBS CompactPCI carrier for 2/4 IndustryPack modules

To get more information about the features and use of supported devices it is recommended to read the manuals listed below.

Installation Guide – TEWS TECHNOLOGIES VxWorks Device Drivers
Carrier Board User Manual

2 Installation

The installation process for the legacy and VxBus enabled IPAC carrier driver is described in detail in the manual “Installation-Guide for TEWS TECHNOLOGIES VxWorks Device Drivers” (Installation_Guide-<release>.pdf) which is part of this distribution.

Basically the installation process described for the hypothetical driver TDRV999-SW-42 is similar to this IPAC carrier driver with the following exceptions.

2.1 Installing the Legacy Device Driver

For Intel x86 based legacy BSPs the function to add MMU mapping entries, which must be inserted in the BSP file sysLib.c is (see also 4.2):

```
ipCarrierPciInit();
```

2.2 VxBus Enabled Device Driver

Due to the fact that the IPAC carrier driver does not contain an application interface (API) and example application such as the hypothetical driver TDRV999-SW-42 the carrier driver must be added in the “Kernel Configuration” directly.

To add the carrier driver to a VxWorks project include the component *“TEWS IPAC CARRIER Driver“ (DRV_TEWS_IPAC_CARRIER)* in the folder *“hardware (default) - Device Drivers”.*

3 Configuration

3.1 VME Bus Carrier Board Setup

The IPAC carrier driver provides a programmable interface to support IPAC modules on VME carrier boards (e.g. TVME200). Due to the fact that Wind River has not designed a standard interface to access VME Bus resources the carrier driver provides an interface to handle most of all proprietary VME Bus interface implementations.

This interface provides functions to add or remove VME carrier slots dynamically after the carrier driver is started. This gives the programmer the chance to map VME addresses and calculate virtual addresses to access the IPAC slot address spaces (IO, ID and MEM spaces).

On the other hand the carrier driver doesn't know the proprietary mechanism how to connect an interrupt service routine (ISR) to a VME Bus interrupt or how to enable or disable VME Bus interrupt level. Consequently this must be implemented by a programmer who is familiar with this VME Bus interface implementation. For this the carrier driver provides a function pointer interface which must be setup by the customer. Every time the carrier driver tries to connect an ISR to an IPAC module plugged on a VME Bus carrier board it invokes the interrupt connect and enable callback functions.

A working example code is implemented in the file custom_carrier_exa.c which is part of this distribution. This example was designed for older VxWorks 5.5 and 6.x releases with a quasi-standard VME Bus interface with intConnect(), sysIntEnable() and sysIntDisable() library functions.

The callback function pointer interface contains the following function pointers which are defined in the header file ipac_slots.h. Every function pointer must be setup with a valid function pointer (default is NULL) otherwise the carrier driver will return ERROR if an IPAC module driver tries to connect an interrupt server routine to a VME Bus IPAC slot.

3.1.1 _customIntConnect

NAME

_customIntConnect – Connect an ISR to a VME Bus interrupt vector

SYNOPSIS

```
STATUS _customIntConnect
(
        VOIDFUNCPTR     *vector,
        VOIDFUNCPTR     routine,
        long            parameter,
        int             slotIndex
);
```

PARAMETER

vector

Interrupt vector to connect.

routine

Function pointer to the interrupt service routine (ISR).

parameter

Parameter passed to the ISR at invocation.

slotIndex

Mirrored slot index defined in the IPAC slot table.

DESCRIPTION

The function assigned to this function pointer is called if the carrier driver tries to connect an interrupt service routine to a customer supplied IPAC slot. The example below shows an example implementation of such function with the legacy intConnect() system call.

EXAMPLE

```
#include "ipac_slots.h"

STATUS __customIntConnect(   VOIDFUNCPTR *vector,
                             VOIDFUNCPTR routine,
                             long parameter,
                             int slotIndex)
{
     /* customer specific interrupt connect function */
     return intConnect(vector, routine, parameter);
}
```

RETURNS

OK on success or ERROR if the function fails.

3.1.2 _customIntDisconnect

NAME

_customIntDisconnect – Disconnect an ISR from a VME Bus interrupt vector

SYNOPSIS

```
void _customIntDisconnect
(
        VOIDFUNCPTR   *vector,
        VOIDFUNCPTR   routine,
        long          parameter,
        int           slotIndex
);
```

PARAMETER

vector

Interrupt vector to connect.

routine

Function pointer to the interrupt service routine (ISR).

parameter

Parameter passed to the ISR at invocation.

slotIndex

Mirrored slot index defined in the IPAC slot table.

DESCRIPTION

The function assigned to this function pointer is called if the carrier driver tries to disconnect an interrupt service routine from a customer supplied IPAC slot. The example below shows an example implementation of such function with the legacy intDisconnect() system call.

On older VxWorks releases an interrupt disconnect system call may not available. Nevertheless a valid function must be assigned to the function pointer.

EXAMPLE

```
#include "ipac_slots.h"

void __customIntDisconnect(  VOIDFUNCPTR *vector,
                             VOIDFUNCPTR routine,
                             long parameter,
                             int slotIndex)
{
     /* customer specific interrupt disconnect function */
     intDisconnect(vector, routine, parameter);
}
```

3.1.3 _customIntEnable

NAME

_customIntEnable – Enable a VME Bus interrupt level

SYNOPSIS

```
STATUS _customIntEnable
(
    int             level,
    int             slotIndex
);
```

PARAMETER

level

VME Bus interrupt level to enable

slotIndex

Mirrored slot index defined in the IPAC slot table.

DESCRIPTION

The function assigned to this function pointer is called if the carrier driver tries to enable a VME Bus interrupt level at a customer supplied IPAC slot. The example below shows an example implementation of such function with the legacy sysIntEnable() system call.

EXAMPLE

```
#include "ipac_slots.h"

STATUS __customIntEnable( int level, int slotIndex )
{
    /* customer specific interrupt enable function */
    sysIntEnable(level);
    return OK;
}
```

RETURNS

OK on success or ERROR if the function fails.

3.1.4 _customIntDisable

NAME

_customIntDisable – Disable a VME Bus interrupt level

SYNOPSIS

```
void _customIntDisable
(
      int               level,
      int               slotIndex
);
```

PARAMETER

level

VME Bus interrupt level to enable

slotIndex

Mirrored slot index defined in the IPAC slot table.

DESCRIPTION

The function assigned to this function pointer is called if the carrier driver tries to disable a VME Bus interrupt level at a customer supplied IPAC slot. The example below shows an example implementation of such function with the legacy sysIntDisable() system call.

EXAMPLE

```
#include "ipac_slots.h"

void __customIntDisable( int level, int slotIndex )
{
     /* customer specific interrupt disable function */
     sysIntDisable(level);
}
```

3.1.5 carrierAddNonePnPSlots

NAME

carrierAddNonePnPSlots – Add VME Bus IPAC slots

SYNOPSIS

```
STATUS carrierAddNonePnPSlots
(
        struct carrier_slot_desc   slot_desc[]
)
```

PARAMETER

slot_desc[]

This parameter is a pointer to a structure of type carrier_slot_desc.

```
struct carrier_slot_desc
{
        int             slotIndex;
        unsigned long   ioBase;
        unsigned long   idBase;
        unsigned long   memBase;
        int             intVec;
        int             int0Lvl;
        int             int1Lvl;
}
```

slotIndex

Specifies the slot on the carrier board (slot A = 0, slot B = 1 and so on or -1 for end of list). For identification purposes the slotIndex is passed to every callback function.

ioBase

Mapped address of the IPAC IO space as seen from the CPU (not the VME address).

idBase

Mapped address of the IPAC ID space as seen from the CPU (not the VME address).

memBase

Mapped address of the IPAC MEM space as seen from the CPU (not the VME address).

intVec

VME Bus interrupt vector used by the IPAC module driver to setup the device hardware and connect the interrupt service routine.

int0Lvl

VME Bus interrupt level (1..7) for IPAC INTREQ0#.

int1Lvl

VME bus interrupt level (1..7) for IPAC INTREQ1#.

DESCRIPTION

This function adds custom supplied IPAC slots to the database of the IPAC carrier driver. Resource information for each slot is specified in the array slots of type struct carrier_slot_desc.

See also the header file ipac_slots.h and the example custom_carrier_exa.c for details.

EXAMPLE

```
#include "ipac_slots.h"

/* basic setup of carrier driver - only necessary for legacy driver */
extern STATUS ipCarrierInit(void);

struct carrier_slot_desc  slots[] =
{
/*                                                    Interrupt          */
/*  slot   IO-Space     ID-Space   MEM-Space    Vector   Level         */
/*  index    base         base       base              INT0 INT1       */
/*   ------------------------------------------------------------------*/
     { 0, 0xF1FF6000,  0xF1FF6080, 0xF0D00000,   0xA0,   1,   2 },
     { 1, 0xF1FF6100,  0xF1FF6180, 0xF0D40000,   0xA4,   3,   4 },
     { 2, 0xF1FF6200,  0xF1FF6280, 0xF0D80000,   0xA8,   5,   6 },
     { 3, 0xF1FF6300,  0xF1FF6380, 0xF0DC0000,   0xAC,   7,   0 },

     /* end of list entry (slot_index must be -1) */
     { -1, 0, 0, 0, 0, 0, 0 },
};

/*
**   Customer supplied callback functions (see examples above)
**
**   __customIntConnect(…){…}
**   __customIntDisconnect(…){…}
**   __customIntEnable(…){…}
**   __customIntDisable(…){…}
*/

…
```

```
…

void setupCustomCarrier(void)
{
     /*
     ** setup custom supplied callback function interface
     */
     _customIntConnect    = __customIntConnect;
     _customIntDisconnect = __customIntDisconnect;
     _customIntEnable     = __customIntEnable;
     _customIntDisable    = __customIntDisable;

     /*
     ** Basic initialization of IPAC carrier driver
     ** (only required for legacy carrier driver)
     */
     if (ipCarrierInit() == ERROR)
     {
          printf("Initialization of IPAC carrier failed\n");
     }


     if (carrierAddNonePnPSlots(slots) == OK)
     {
          printf("Adding custom supplied carrier slots was successful\n");
     }
     else
     {
          printf("Adding custom supplied carrier slots failed\n");
     }
}
```

RETURNS

OK on success or ERROR if the function fails.

3.1.6 carrierRemoveNonePnPSlots

NAME

carrierRemoveNonePnPSlots – Remove all VME Bus IPAC slots

SYNOPSIS

```
STATUS carrierRemoveNonePnPSlots
(
        void
)
```

DESCRIPTION

This function removes all customer supplied IPAC slots from the carrier driver. If at least on slot is in use by an IPAC module driver the function fails.

RETURNS

OK on success or ERROR if the function fails.

4 Interface Functions

4.1 ipCarrierInit

NAME

ipCarrierInit - IPAC carrier driver initialization

SYNOPSIS

#include “ipac_carrier.h”

STATUS ipCarrierInit(void)

DESCRIPTION

For the legacy carrier driver this function must be called before the first call to ipFindDevice(). For the VxBus carrier driver the initialization function is called automatically from the VxBus subsystem during startup.

During carrier driver initialization, the peripheral buses will be scanned for supported carrier boards. All found slots on PnP (PCI) carrier boards and manually added slots (VME bus) will be added to an internal data base. In a second phase, the carrier driver will check every slot for mounted IPAC modules. All collected information will now be available for the ipFindDevice() function.

Calling the ipCarrierInit() function is mandatory for the legacy carrier driver and unnecessary for the VxBus carrier driver. For compatibility purposes this function is also available (dummy) for the VxBus carrier driver.

EXAMPLE

```
#include "ipac_carrier.h”

/*
**  First initialize the IP carrier driver if not already done.
**  Note: ipCarrierInit() can be called several times.
*/
if (ipCarrierInit() == ERROR) {
     printf("ERROR: IPAC carrier driver initialization failed\n");
}
```

RETURNS

OK if initialization was successful or ERROR if not.

4.2 ipCarrierPciInit

NAME

ipCarrierPciInit – Generic PCI device initialization

SYNOPSIS

void ipCarrierPciInit()

DESCRIPTION

This function is only required for Intel x86 VxWorks platforms (see also the manual Installation-Guide.pdf). The purpose is to setup the MMU mapping for all required carrier board PCI spaces (base address register) on supported (Compact)PCI carrier boards.

The right place to call the function *ipCarrierPciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

This function is only declared and necessary for legacy carrier driver.

EXAMPLE

```
extern void ipCarrierPciInit();

ipCarrierPciInit();
```

4.3 ipFindDevice

NAME

ipFindDevice – Find the specified IPAC module on supported carrier boards

SYNOPSIS

```
STATUS ipFindDevice
(
        unsigned long           manufacturerID,
        unsigned long           modelNumber,
        int                     index,
        unsigned long           slotConfig,
        struct ipac_resource    *ipac
)
```

DESCRIPTION

This function searches for the IPAC module specified by the manufacturer ID, the IPAC model number and the sequence index in the internal database. If the specified module was found (return value OK), the carrier slot will be configured as specified in the argument slotConfig and the structure ipac_resource will be filled with information required to setup the appropriate IPAC module driver.

The argument index specifies the sequence number if more than one module of the same type is installed on supported carrier boards. To select the first module, index must be set to 0, for the second module set index to 1 and so on.

The sequence of IPAC modules is always deterministic. 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. For VME bus carrier boards the sequence index is given by the setup of the slot description array.

The configuration for the carrier slot where the allocated IPAC module is installed is passed by the argument slotConfig to the carrier driver. More than one configuration items can be combined by using a bit-wise OR.

PARAMETER

manufacturerID

Specifies an 8-bit (IDPROM Data Format I) or 24-bit (IDPROM Data Format II) board manufacturer ID (e.g. 0xB3 for TEWS TECHNOLOGIES).

modelNumber

Specifies an 8-bit (IDPROM Data Format I) or 24-bit (IDPROM Data Format II) model number (manufacturer specific).

index

Sequence index to select a certain IPAC module if more than one module of the same type is installed.

slotConfig

Specifies configuration items for the carrier slot where the IPAC module is installed. The following configuration flags are defined. Use a bit-wise OR to combine more than one flag.

Value	Description
IPAC_INT0_EN	Enable INTREQ0# on the carrier board and the related bus interrupt level. If the auto enable interrupt level feature is enable the CPU board interrupt level will be enabled also with the appropriate system function (intEnable(), sysIntEnable() or vxbIntEnable()).
IPAC_INT1_EN	Same as INTREQ0# for IPAC INTREQ1#
IPAC_EDGE_SENS	Enable edge-sensitive interrupt requests. Only supported by the VxBus enabled carrier driver.
IPAC_LEVEL_SENS	Enable level-sensitive interrupt requests (default)
IPAC_CLK_8MHZ	IPAC clock rate is 8 MHz (default)
IPAC_CLK_32MHZ	IPAC clock rate is 32 MHz
IPAC_MEM_8BIT	The IPAC MEM space is 8-bit wide. Only supported by TEWS PCI carrier boards.
IPAC_MEM_16BIT	The IPAC MEM space is 16-bit wide (default)
IPAC_IACK_CYC	If the IPAC module requires an IACK cycle to acknowledge a pending interrupt, this flag must be set. This configuration is only relevant for PCI carrier boards. If set, the carrier driver will install an ISR which is called before the IPAC module driver ISR is called. This ISR performs a read access to the carrier slot INT space to obtain the interrupt vector from the module (IACK cycle).

ipac

On success this structure is filled with resource information of the slot where the IPAC module is installed. This information can be used to setup the appropriate IPAC module driver.
The memory for the IPAC resource information must be allocated statically (e.g. in the device control block) and must be unique for every IPAC module. Because the IPAC carrier access functions reference the space descriptors within this structure they must be available as long as the IPAC module is referenced by the device driver or application program.

```
struct ipac_resource {
        int                     carrier_type;
        int                     slotIndex;
        unsigned char           *ioBase;
        unsigned char           *idBase;
        unsigned char           *memBase;
        int                     intVec;
        int                     int0Lvl;
        int                     int1Lvl;
        int                     moduleId;
        struct addr_space_desc  idSpace;
        struct addr_space_desc  ioSpace;
        struct addr_space_desc  memSpace;
        unsigned long           slotConfig;
        void                    *pSlotInfo;
        void                    *pVxBusInst;
};
```

carrier_type

Type of carrier board where the IP module is plugged (IPAC_TEWS_PCI, IPAC_SBS_PCI, IPAC_VME …).

slotIndex

Specifies the slot on the carrier board (slot A = 0, slot B = 1…).

ioBase, idBase, memBase

Pointer to the IPAC module IO, ID and MEM space

intVec

Interrupt vector which can be used to connect the module ISR. Only valid for legacy driver PCI bus and VME bus carrier boards.

int0Lvl, int1Lvl

Interrupt level which corresponds to the IPAC module INTREQ0# and INTREQ1#.
Only valid for legacy driver PCI bus and VME bus carrier boards.

moduleId

1:1 copy of the ipCarrierFind() argument *index*.

idSpace, ioSpace, memSpace

Address space descriptor for IPAC module ID, IO and MEM space. Beside the real address an address space descriptor defines the type of access (e.g. endian mode).

EXAMPLE

```
#include ”ipac_carrier.h”

STATUS    result;
struct ipac_resource  ipac1, ipac2, ipac3;


/*
**   Find an IP module from TEWS TECHNOLOGIES (manufacturer = 0xB3)
**   with model number 0x33. This module does not use interrupts and
**   we need only the IO space base address for the related driver.
*/

result = ipFindDevice(0xB3, 0x33, 0, IPAC_CLK_8MHZ, &ipac1);

if (result == ERROR)
{
     printf("ERROR: No IP found\n");
}


/*
**   Find the same module type as above but attach the second module
**   found.
*/

result = ipFindDevice(0xB3, 0x33, 1, IPAC_CLK_8MHZ, &ipac2);

if (result == ERROR)
{
     printf("ERROR: No IP found\n");
}


/*
**   The following module generates level sensitive interrupts at INT0.
**   and has a 16-bit wide memory interface. Important for PCI carrier,
**   this module requires an IACK cycle to acknowledge a pending
**   interrupt.
*/

result = ipFindDevice(  0xB3,
                        0x1C,
                        0,
```

```
                        IPAC_INT0_EN | IPAC_LEVEL_SENS
                        | IPAC_CLK_8MHZ | IPAC_MEM_16BIT | IPAC_IACK_CYC,
                        &ipac3);

if (result == ERROR)
{
     printf("ERROR: No IP found\n");
}
```

RETURNS

OK if the specified IPAC module was found or ERROR if not.

4.4 ipFreeDevice

NAME

ipFreeDevice – Free IPAC module resources

SYNOPSIS

STATUS ipFreeDevice(struct ipac_resource *ipac)

DESCRIPTION

This function returns allocated resources for this IPAC module instance and setup the carrier board slot (only PCI carrier) to a well-known inactive state. Before calling this function the IPAC interrupt handling must be disabled by calling the ipac_free_irq() function.

On success the carrier board “in use” count will be decremented to make this VxBus instance removable for hot-plugging purposes (if supported).

This function is only implemented in the VxBus carrier driver.

PARAMETER

ipac

Pointer to IPAC module resource handle that was allocated by a prior call to ipFindDevice().

EXAMPLE

```
#include “ipac_carrier.h”

STATUS    result;
struct ipac_resource  ipac;

result = ipFreeDevice(&ipac)

if (result == ERROR)
{
     printf("ERROR: freeing IPAC device failed\n");
}
```

RETURNS

OK if the specified IPAC device was successful freed ERROR if not.

4.5 ipac_map_space

NAME

ipac_map_space – Obtain an IPAC address space descriptor

SYNOPSIS

struct addr_space_desc *ipac_map_space (struct ipac_resource *ipac, int space_id)

DESCRIPTION

This function returns a space descriptor handle for specified IPAC module space. This handle will be used to access the corresponding IPAC space with the ipac_read_*() and ipac_write_*() access functions.

As long as the returned address space descriptor is used to access the IPAC spaces the referenced *ipac* resource descriptor must be available.

PARAMETER

ipac

Pointer to IPAC module resource handle that was allocated by ipFindDevice().

space_id

Selects the address space. Valid space identifiers are:

Value	Description
IPAC_IO_SPACE	IPAC module IO space
IPAC_ID_SPACE	IPAC module ID space
IPAC_MEM_SPACE	IPAC module MEM space

EXAMPLE

```
#include “ipac_carrier.h”

struct ipac_resource  ipac;
struct addr_space_desc *id_space;


result = ipFindDevice(0xB3, 0x33, 0, IPAC_CLK_8MHZ, &ipac);

/*...*/

if ((id_space = ipac_map_space(ipac, IPAC_ID_SPACE)) == NULL)
{
     printf("mapping ID space failed");
}
```

RETURNS

Returns a pointer to the space descriptor inside the module resource handle or NULL if an error occurred.

4.6 ipac_request_irq

NAME

ipac_request_irq – Connect an interrupt service routine to the module interrupt

SYNOPSIS

```
STATUS ipac_request_irq
(
        struct ipac_resource    *ipac,
        VOIDFUNCPTR             *vector,
        VOIDFUNCPTR             routine,
        long                    parameter
);
```

DESCRIPTION

This function connects an interrupt service routine to the module interrupt and enables the appropriate interrupt level if the auto interrupt enable facility is enabled.

PARAMETER

ipac

Pointer to IPAC module resource handle that was allocated by ipFindDevice().

vector

Interrupt vector to connect. This parameter is not used for VxBus devices.

routine

Pointer to the interrupt service routine to connect.

parameter

Argument that will be passed to the interrupt service routine.

EXAMPLE

```
#include “ipac_carrier.h”

struct ipac_resource  ipac;
STATUS result;
int arg;


result = ipFindDevice(0xB3, 0x33, 0, IPAC_CLK_8MHZ, &ipac);

/*...*/

result = ipac_request_irq(   ipac,
                             INUM_TO_IVEC(ipac.intVec),
                             ISR_func,
                             arg );

if (result == ERROR)
{
     printf(“connectig ISR failed\n”);
}
```

RETURNS

OK on success or ERROR if the ISR cannot be connected.

4.7 ipac_free_irq

NAME

ipac_free_irq – Disconnect an interrupt service routine from the module interrupt

SYNOPSIS

```
STATUS ipac_free_irq
(
        struct ipac_resource    *ipac,
        VOIDFUNCPTR             *vector,
        VOIDFUNCPTR             routine,
        long                    parameter
);
```

DESCRIPTION

This function disconnects an interrupt service routine from the module interrupt and disables the appropriate interrupt level if the auto interrupt enable facility is enabled.

This function is only implemented in the VxBus carrier driver.

PARAMETER

ipac

Pointer to IPAC module resource handle that was allocated by ipFindDevice().

vector

Interrupt vector to disconnect

routine

Pointer to the interrupt service routine to disconnect

parameter

Argument passed to the interrupt service routine.

EXAMPLE

```
#include “ipac_carrier.h”

struct ipac_resource  ipac;
STATUS result;
int arg;


result = ipFindDevice(0xB3, 0x33, 0, IPAC_CLK_8MHZ, &ipac);

/*...*/

result = ipac_request_irq(   ipac,
                             INUM_TO_IVEC(ipac.intVec),
                             ISR_func,
                             arg );

/*...*/

result = ipac_free_irq( ipac,
                        INUM_TO_IVEC(ipac.intVec),
                        ISR_func,
                        arg );

if (result == ERROR)
{
     printf(“connectig ISR failed\n”);
}
```

RETURNS

OK on success or ERROR if the ISR cannot be connected.

4.8 ipac_interrupt_ack

NAME

ipac_interrupt_ack – Acknowledge a pending interrupt and return the interrupt status

SYNOPSIS

STATUS ipac_interrupt_ack(struct ipac_resource *ipac, struct ipac_intstatus *intstatus);

DESCRIPTION

This function performs an IACK cycle by reading the INT space of PCI based carrier boards and returns the vector and status of both IPAC interrupt request lines.

For IPAC module interrupts that are acknowledged by this function the slot configuration flag IPAC_IACK_CYC should not be set, otherwise a pending interrupt may be acknowledged by the auto IACK facility (see also 4.3) before this function is called.

Usually this function will be called within the interrupt service routine for an IPAC module which does not provide a dedicated interrupt status register.

This function is only implemented in the VxBus carrier driver.

PARAMETER

ipac

Pointer to IPAC module resource handle that was allocated by ipFindDevice().

intstatus

Pointer to a variable of type struct ipac_intstatus, which obtains the read vector and interrupt status

```
struct ipac_intstatus {
        int     INT0_active;
        int     INT0_vector;
        int     INT1_active;
        int     INT1_vector;
};
```

INT0_active

TRUE if the IPAC INT0 interrupt is active

INT0_vector

Read interrupt vector (INT space IACK cycle) if INT0 is active and this feature is available.

INT1_active

TRUE if the IPAC INT1 interrupt is active

INT1_vector

Read interrupt vector (INT space IACK cycle) if INT1 is active and this feature is available.

EXAMPLE

```
#include “ipac_carrier.h”

struct ipac_resource  ipac;
struct ipac_intstatus intstatus;
STATUS result;


result = ipac_interrupt_ack(&ipac, &instatus);

if (result == ERROR)
{
     printf(“Feature is not available\n”);
}
```

RETURNS

Returns OK if this feature is available (TEWS PCI carrier boards) or ERROR if not.

4.9 ipac_read_uchar

NAME

ipac_read_uchar – Read one byte (8-bit) from IPAC space

SYNOPSIS

unsigned char ipac_read_uchar(struct addr_space_desc *space, unsigned long offset);

DESCRIPTION

Read one byte (8-bit) from the IPAC space location specified by the address space descriptor and the relative offset.

This access function always expects big-endian IPAC spaces either by hardware design (TVME8xxx CPU boards) or re-programming the BAR layout of TEWS TECHNOLOGIES (Compact)PCI carrier boards. For little-endian SBS (Compact)PCI carrier boards the byte lanes will be swapped.

On PowerPC boards the access will be ordered (EIEIO).

PARAMETER

space

Address space descriptor pointer.

offset

Address offset (bytes) within this space.

EXAMPLE

```
#include ”ipac_carrier.h”

struct addr_space_desc *id_space;
unsigned char modelNumber;

/* ... */

modelNumber = ipac_read_uchar( id_space, 0x0B );
```

RETURNS

Returns the read byte.

4.10 ipac_read_ushort

NAME

ipac_read_ushort – Read one word (16-bit) from IPAC space

SYNOPSIS

unsigned short ipac_read_ushort(struct addr_space_desc *space, unsigned long offset);

DESCRIPTION

Read one word (16-bit) from the IPAC space location specified by the address space descriptor and the relative offset.

This access function always performs a big-endian access. Depending on the CPU architecture and carrier board hardware byte lanes will be swapped as necessary.

On PowerPC boards the access will be ordered (EIEIO).

PARAMETER

space

Address space descriptor pointer.

offset

Address offset (bytes) within this space.

EXAMPLE

```
#include ”ipac_carrier.h”

struct addr_space_desc *io_space;
unsigned short data;

/* ... */

data = ipac_read_ushort( io_space, 0x40 );
```

RETURNS

Returns the read word.

4.11 ipac_read_ulong

NAME

ipac_read_ulong – Read one long word (32-bit) from IPAC space

SYNOPSIS

unsigned long ipac_read_ulong(struct addr_space_desc *space, unsigned long offset);

DESCRIPTION

Read one long word (32-bit) from the IPAC space location specified by the address space descriptor and the relative offset.

This access function always performs a big-endian access. Depending on the CPU architecture and carrier board hardware word byte and word lanes will be swapped as necessary.

On PowerPC boards the access will be ordered (EIEIO).

PARAMETER

space

Address space descriptor pointer.

offset

Address offset (bytes) within this space.

EXAMPLE

```
#include ”ipac_carrier.h”

struct addr_space_desc *mem_space;
unsigned long data;

/* ... */

data = ipac_read_ulong( mem_space, 0x1000 );
```

RETURNS

Returns the read long word.

4.12 ipac_write_uchar

NAME

ipac_write_uchar – Write one byte (8-bit) to an IPAC space

SYNOPSIS

void ipac_write_uchar(struct addr_space_desc *space, unsigned long offset , unsigned char value);

DESCRIPTION

Write one byte (8-bit) to the IPAC space location specified by the address space descriptor and the relative offset.

This access function always expects big-endian IPAC spaces either by hardware design (TVME8xxx CPU boards) or re-programming the BAR layout of TEWS TECHNOLOGIES (Compact)PCI carrier boards. For little-endian SBS (Compact)PCI carrier boards the byte lanes will be swapped.

On PowerPC boards the access will be ordered (EIEIO).

PARAMETER

space

Address space descriptor pointer.

offset

Address offset (bytes) within this space.

value

Data byte (8-bit) to write to the specified location.

EXAMPLE

```
#include ”ipac_carrier.h”

struct addr_space_desc *io_space;

/* ... */

/* write 0x55 to offset 0x20 within the IO space */
ipac_write_uchar( io_space, 0x20, 0x55 );
```

4.13 ipac_write_ushort

NAME

ipac_write_ushort – Write one word (16-bit) to an IPAC space

SYNOPSIS

void ipac_write_ushort(struct addr_space_desc *space, unsigned long offset , unsigned short value);

DESCRIPTION

Write one word (16-bit) to the IPAC space location specified by the address space descriptor and the relative offset.

This access function always performs a big-endian access. Depending on the CPU architecture and carrier board hardware byte lanes will be swapped as necessary.

On PowerPC boards the access will be ordered (EIEIO).

PARAMETER

space

Address space descriptor pointer.

offset

Address offset (bytes) within this space.

value

Data word (16-bit) to write to the specified location.

EXAMPLE

```
#include ”ipac_carrier.h”

struct addr_space_desc *mem_space;

/* ... */

/* write 0xaa55 to offset 0x1000 within the MEM space */
ipac_write_ushort( io_space, 0x1000, 0xaa55 );
```

4.14 ipac_write_ulong

NAME

ipac_write_ulong – Write one long word (32-bit) to an IPAC space

SYNOPSIS

void ipac_write_ulong(struct addr_space_desc *space, unsigned long offset , unsigned long value);

DESCRIPTION

Write one long word (32-bit) to the IPAC space location specified by the address space descriptor and the relative offset.

This access function always performs a big-endian access. Depending on the CPU architecture and carrier board hardware word byte and word lanes will be swapped as necessary.

On PowerPC boards the access will be ordered (EIEIO).

PARAMETER

space

Address space descriptor pointer.

offset

Address offset (bytes) within this space.

value

Data long word (32-bit) to write to the specified location.

EXAMPLE

```
#include ”ipac_carrier.h”

struct addr_space_desc *mem_space;

/* ... */

/* write 0xdadada55 to offset 0x1000 within the MEM space */
ipac_write_ulong( mem_space, 0x1000, 0xdadada55 );
```

4.15 ipCarrierShow

NAME

ipCarrierShow – Show the contents of IPAC carrier data base

SYNOPSIS

void ipCarrierShow()

DESCRIPTION

This function can be used for debugging purposes to display the contents of the internal IPAC carrier data base. Usually this function is called at the VxWorks target shell to get information on found carrier boards, IPAC modules and slot resources (e.g. to access IPAC spaces manually).

EXAMPLE

```
-> ipCarrierInit
value = 0 = 0x0

-> ipCarrierShow

TVME8240 carrier with 4 slots found @ PCI bus=0, device=16
 - Slot[0]: IO=0xF5000000, ID=0xF5000080, INT=0xF50000C0, MEM8=0xF4000000,
MEM16=0xF2000000, IVEC=0x34, INT0_LVL=0
x34, INT1_LVL=0x34
            +++ Valid IP module mounted (Manufacturer=0xB3, Model=0x22)
 - Slot[1]: IO=0xF5000100, ID=0xF5000180, INT=0xF50001C0, MEM8=0xF4400000,
MEM16=0xF2800000, IVEC=0x34, INT0_LVL=0
x34, INT1_LVL=0x34
            --- NO or BAD IP module
 - Slot[2]: IO=0xF5000200, ID=0xF5000280, INT=0xF50002C0, MEM8=0xF4800000,
MEM16=0xF3000000, IVEC=0x34, INT0_LVL=0
x34, INT1_LVL=0x34
            --- NO or BAD IP module
 - Slot[3]: IO=0xF5000300, ID=0xF5000380, INT=0xF50003C0, MEM8=0xF4C00000,
MEM16=0xF3800000, IVEC=0x34, INT0_LVL=0
x34, INT1_LVL=0x34
            +++ Valid IP module mounted (Manufacturer=0xB3, Model=0x1C)
```

```
VMEbus carrier with 4 slots found
 - Slot[0]: IO=0xF1FF6000, ID=0xF1FF6080, INT=0x00000000, MEM8=0xF0D00000,
MEM16=0xF0D00000, IVEC=0xa0, INT0_LVL=0
x1, INT1_LVL=0x2
            --- NO or BAD IP module
 - Slot[1]: IO=0xF1FF6100, ID=0xF1FF6180, INT=0x00000000, MEM8=0xF0D40000,
MEM16=0xF0D40000, IVEC=0xa4, INT0_LVL=0
x3, INT1_LVL=0x4
            --- NO or BAD IP module
 - Slot[2]: IO=0xF1FF6200, ID=0xF1FF6280, INT=0x00000000, MEM8=0xF0D80000,
MEM16=0xF0D80000, IVEC=0xa8, INT0_LVL=0
x5, INT1_LVL=0x6
            --- NO or BAD IP module
 - Slot[3]: IO=0xF1FF6300, ID=0xF1FF6380, INT=0x00000000, MEM8=0xF0DC0000,
MEM16=0xF0DC0000, IVEC=0xac, INT0_LVL=0
x7, INT1_LVL=0x0
            --- NO or BAD IP module
```