The Embedded I/O Company

TEWS TECHNOLOGIES

TIP845-SW-72

LynxOS Device Driver

48 Channel 14 bit A/D Conversion

Version 1.0.x

User Manual

Issue 1.0

September 2004

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
Phone: +49-(0)4101-4058-0
e-mail: info@tews.com

25469 Halstenbek / Germany
Fax: +49-(0)4101-4058-19
www.tews.com

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor
Phone: +1 (775) 686 6077
e-mail: usasales@tews.com

Reno, Nevada 89504 / USA
Fax: +1 (775) 686 6024
www.tews.com

TIP845-SW-72

48 Channel 14 bit A/D Conversion

LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	September 14, 2004

Table of Contents

1 Introduction

The TIP845-SW-72 LynxOS device driver allows the operation of a TIP845 IPAC module on LynxOS operating systems.

Because the TIP845 device driver is stacked on the TEWS TECHNOLOGIES IPAC carrier driver, it's necessary to install also the IPAC carrier driver. Please refer to the IPAC carrier driver user manual for further information.

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

The TIP845 device driver includes the following functions:

- Reading a single ADC input channel with or without data correction
- Reading 1 to 48(24) single ended (differential) channels simultaneously through the TIP845 built in sequencer with programmable sample rate
- Reading module correction values stored in the ID PROM
- TEWS TECHNOLOGIES IPAC carrier driver support.

2 Installation

The software is delivered on a PC formatted 3½" HD diskette.

The directory A:\TIP845-SW-72 contains the following files:

TIP845-SW-72.pdf	This manual in PDF format
TIP845-SW-72.tar	Device Driver and Example sources

The TAR archive TIP845-SW-72.tar contains the following files and directories:

tip845/tip845.c	Driver source code
tip845/tip845.h	Definitions and data structures for driver and application
tip845/tip845def.h	Definitions and data structures for the driver
tip845/tip845_info.c	Device information definition
tip845/tip845_info.h	Device information definition header
tip845/tip845.cfg	Driver configuration file include
tip845/tip845.import	Linker imports file for PowerPC platforms
tip845/Makefile	Device driver make file
tip845/Example/example.c	Example application source
tip845/Example/Makefile	Example make file

In order to perform a driver installation first extract the TAR file to a temporary directory then copy the following files 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/tip845 or /sys/drivers.cpci_x86/tip845

2. Copy the following files to this directory:
   - tip845.c
   - tip845def.h
   - tip845.import
   - Makefile

3. Copy tip845.h to /usr/include/

4. Copy tip845_info.c to /sys/devices.xxx/ or /sys/devices if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tip845_info.h to /sys/dheaders/

6. Copy tip845.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 the IPAC carrier driver user manual in the directory path *A:\CARRIER-SW-72* on the separate distribution diskette.

2.1 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 the IPAC Carrier Driver User Manual for detailed information.

2.1.1 Static Installation

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

2.1.1.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip845, where xxx represents the BSP that supports the target hardware.
2. To update the library /sys/lib/libdrivers.a enter:

   ```
   make install
   ```

2.1.1.2 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 = ... tip845_info.x
   ```

   And at the end of the Makefile

   ```
   tip845_info.o:$(DHEADERS)/tip845_info.h
   ```

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

   ```
   make install
   ```

2.1.1.3 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:tip845.cfg
   ```

2.1.1.4 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/tip845_0, /dev/tip845_1, /dev/tip845_2, …

2.1.2 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.

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip845, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter :

   ```
   make
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tip845, where xxx represents the BSP that supports the target hardware.
2. To create a device definition file for the major device (this work only on native system)

   ```
   make t845info
   ```

3. To install the driver enter:

   ```
   drinstall –c tip845.obj
   ```

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

   ```
   devinstall –c –d <driver-ID> t845info
   ```

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

   ```
   mknod /dev/tip845_0 c <major_no> 0
   mknod /dev/tip845_1 c <major_no> 1
   mknod /dev/tip845_2 c <major_no> 2
   ...
   ```

   The <major_no> is returned by the devinstll command.

If all steps are successful completed the TIP845 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TIP845 device enter the following commands:

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

2.1.3 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 TIP845 driver and devices into the LynxOS system, the configuration include file tip845.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tip845.cfg on the distribution disk contains the driver entry (*C:tip845:*\....) and a major device entry (*D:TIP845:t845info::*) with 9 minor device entries (*"N: tip845_0:0", ..., "N: tip845_8:8"*).

If the driver should support more than nine TIP845, additional minor device entries must be added. To create the device node */dev/tip845_9* the line *N:tip845_9:9* must be added at the end of the file tip845.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:tip845:\
     :t845open:t845close:t845read::\
     ::t845ioctl:t845install:t845uninstall
D:TIP845:t845info::
N:tip845_0:0
N:tip845_1:1
N:tip845_2:2
N:tip845_3:3
N:tip845_4:4
N:tip845_5:5
N:tip845_6:6
N:tip845_7:7
N:tip845_8:8
```

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

/dev/tip845_0 ... /dev/tip845_8

3 TIP845 Device 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.

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 (TIP845 device) named in path for reading and writing. The value of oflags indicates the intended use of the file. In case of a TIP845 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 TIP845 device already exists this argument is ignored.

EXAMPLE

```
int  fd


fd = open ("/dev/tip845_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.

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()

3.3 read()

NAME

read() - read from a file

SYNOPSIS

```
#include <tip845.h>

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

DESCRIPTION

The read function attempts to start an AD conversion on a single channel (manual mode) or to wait for sequencer data on all configured channels (sequencer mode). For more details about the sequencer see IOCTL (T845_INIT_SEQUENCER) and the “TIP845 Hardware Manual”. After first driver installation the module is in manual mode and the T845_IO_BUFFER is used. To switch into the sequencer mode do an IOCTL (T845_INIT_SEQUENCER). In sequencer mode the T845_SEQ_DATA_BUFFER is used for a read request.

A pointer to the callers read buffer (*T845_IO_BUFFER* or *T845_SEQ_DATA_BUFFER*) and the size of this structure is passed by the parameters *buff* and *count* to the device.

Manual Mode

```
typedef struct
{
        unsigned int        channel;
        unsigned int        gain;
        unsigned int        mode;
        unsigned int        correction;
        long                data;
        int                 timeout;
} T845_IO_BUFFER, *PT845_IO_BUFFER;
```

channel

Specifies the channel number at which to read the AD value. Valid channel numbers are 1...T845_MAX_SINGLE_CHAN if Single-Ended is selected for, if differential is selected the valid channel numbers are in the range of 1...T845_MAX_DIFF_CHAN. (see file “tip845.h”)

gain

Specifies the gain, which shall be used to read the AD value. Valid gains are:

T845_GAIN_1	Select Gain 1x	-10 V…+10 V
T845_GAIN_2	Select Gain 2x	-5 V… +5 V

T845_GAIN_4	Select Gain 4x	-2.5 V… +2.5 V
T845_GAIN_8	Select Gain 8x	-1.25 V… +1.25 V

mode

Specifies the channel input interface. If it should be used with a differential interface, this member must have the value *T845_DIFF*, otherwise the value should be *T845_SINGLE,* if it should be used with a single-ended input.

correction

If this parameter is *T845_CORR* the driver performs an automatic offset and gain correction with factory calibration data stored in the TIP845 ID-PROM, otherwise the value should be *T845_NOCORR*.

data

Analog input value read from the specified 14-bit ADC channel. The analog data is returned as sign extended two's complement long value with 16-bit resolution. The lower two bits are manipulated through the offset and gain calibration values with ¼ LSB resolution. (see also TIP845 Hardware User Manual). The range of data is -32768 to +32767, but not all values are possible (14-bit ADC resolution)

timeout

This parameter defines the maximum time the user is willing to wait to access the device, because of other pending read requests. Possible values are -1 for waiting a indefinitely amount of time or “blocking read”, 0 for no waiting or “non-blocking read” and 1..MAX_INTEGER for waiting timeout ticks.

EXAMPLE

```
int  fd;
int  result;
T845_IO_BUFFER  ADCBuf;


ADCBuf.gain        = T845_GAIN_1;
ADCBuf.mode        = T845_SINGLE;
ADCBuf.channel     = 1;
ADCBuf.correction  = T845_CORR;

result = read(fd, (char*)&ioBuf, sizeof(ioBuf));

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

Sequencer Mode

```
typedef struct
{
        unsigned short      correction;
        long                data[T845_MAX_SINGLE_CHAN];
        unsigned short      config[T845_MAX_SINGLE_CHAN];
        int                 timeout;
        unsigned short      error;
} T845_SEQ_DATA_BUFFER, *PT845_SEQ_DATA_BUFFER;
```

correction

If this parameter is *T845_CORR* the driver performs an automatic offset and gain correction with factory calibration data stored in the TIP845 ID-PROM, otherwise the value should be *T845_NOCORR*.

data

After a successful read requests this array contains the analog input values from all initialized 14-bit ADC sequencer channels. The analog data is returned as sign extended two's complement long value with 16-bit resolution. The lower two bits are manipulated through the offset and gain calibration values with ¼ LSB resolution. (see also TIP845 Hardware User Manual). The range of data is -32768 to +32767, but not all values are possible (14-bit ADC resolution). To get the data of an single ended channel *n = 1..T845_MAX_SINGLE_CHAN* read *data[T845_S2I(n)]* for a differential channel with *n = 1..T845_MAX_DIFF_CHAN* read *data[T845_D2I(n)]*.

config

This array parameter returns the user configuration set by ioctl(…, TIP845_INIT_SEQUENCER, …). See TIP845_INIT_SEQUENCER for more details.

timeout

This parameter defines the maximum time the user is willing to wait for new sequencer data. Possible values are -1 for waiting a indefinitely amount of time or “blocking read”, 0 for no waiting or “non-blocking read” and 1..MAX_INTEGER for waiting timeout ticks.

error

This parameter returns a more detailed error description when an EIO error occurred. On instruction RAM error the flag TIP845_IRAM_ERROR is set. On sequencer timer error the TIP845_TIMER_ERROR is set. On data overflow error the TIP845_DATAOF_ERROR is set. For more details see TIP845 hardware manual.

EXAMPLE

```
...
T845_SEQ_DATA_BUFFER  seqDataBuf;
...

seqDataBuf.correction   = T845_CORR;
seqDataBuf.timeout      = 700;

result = read(fd, (char*)&seqDataBuf, sizeof(seqDataBuf));

if(result < 0)
{
     /* handle read error */
}
else
{
     for (i = 1; i <= T845_MAX_DIFF_CHAN; i++)
     {
          chIdxA = T845_D2I(i);
          config = seqDataBuf.config[chIdxA];
          if (config & T845_ENA)
          {
               if (config & T845_SEDIFF)
               {
                    sprintf(select, "DIFF %d", i);
               }
               else
               {
                    sprintf(select, "SE %d", (i<<1)-1);
               }
               printf( "ADC(%s) \t\tValue = %ld / 0x%04lx\n", select,
seqDataBuf.data[chIdxA], seqDataBuf.data[chIdxA]);
          }

          chIdxB = chIdxA + 1;
          config = seqDataBuf.config[chIdxB];
          if (config & T845_ENA)
          {
               sprintf(select, "SE %d", (i<<1));
               printf( "ADC(%s) \t\tValue = %ld / 0x%04lx\n", select,
seqDataBuf.data[chIdxB], seqDataBuf.data[chIdxB]);
          }
     }
}
```

RETURNS

When read succeeds, the size of the read buffer 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 TIP845 specific error codes:

ENXIO	Illegal device
EINVAL	Invalid argument. This error code is returned if the size of the read buffer is too small or if the gain or channel parameter out of range (Manual Mode).
EIO	AD conversion hasn't finished within the maximum allowed time period (Manual Mode). See read buffer parameter *error* for a detailed description (Sequencer Mode).
ETIMEDOUT	The user defined timeout has elapsed because other write requests to this device are pending (Manual Mode) The user defined timeout has elapsed because the sequencer returned no data within the specified time (Sequencer Mode).
EAGAIN	You've set a timeout value, but there are no timeouts available. Do it again without a timeout.
EINTR	Interrupted system call (probably by a signal).

SEE ALSO

LynxOS System Call - read()

3.4 ioctl()

NAME

ioctl() - I/O device control

SYNOPSIS

```
#include <ioctl.h>
#include <tip845.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 TIP845.h :

Value	Meaning
T845_READ_PARAM	Read module parameter
T845_INIT_SEQUENCER	Setup and start the TIP845 built in sequencer
T845_STOP_SEQUENCER	Stop the sequencer and switch back to manual mode.

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 TIP845 ioctl function returns always standard error codes.

SEE ALSO

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

3.4.1 T845_READ_PARAM

NAME

T845_READ_PARAM - Read module parameter

DESCRIPTION

This ioctl function attempts to read the module type and calibration data of the TIP845 associated with the open file descriptor, *fd*, into the parameter buffer pointed to by *arg.*

The parameter buffer (*T845_PARAM_BUFFER*) has the following layout:

```
typedef struct
{
        int             calGain[4];
        int             calOffs[4];
} T845_PARAM_BUFFER, *PT845_PARAM_BUFFER;
```

calGain

Receives the gain error of the input amplifier for four possible gain selections in the unit ¼ LSB (see also Hardware User Manual).

calOffs

Receives the offset (zero) error of the input amplifier for four possible gain selections in the unit ¼ LSB (see also Hardware User Manual).

EXAMPLE

```
int  fd;
int  result;
T845_PARAM_BUFFER  ParamBuf;

result = ioctl(fd, T845_READ_PARAM, &ParamBuf);

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

ERRORS

No function specific errors will be returned.

SEE ALSO

ioctl man pages

3.4.2 T845_INIT_SEQUENCER

NAME

T845_INIT_SEQUENCER – Setup and start the TIP845 built in sequencer

DESCRIPTION

This ioctl function setups all channels for a periodically ADC conversion sequence and starts the sequencer modul. The TIP845 associated with the open file descriptor *fd*, receives the data buffer pointed to by *arg.*

The data buffer (*T845_SEQ_CONF_BUFFER*) has the following layout:

```
typedef struct
{
        unsigned short      config[T845_MAX_SINGLE_CHAN];
        unsigned short      seq_timer;

} T845_SEQ_CONF_BUFFER, *PT845_SEQ_CONF_BUFFER;
```

config

This array parameter holds the configuration for each sequencer ADC channel. Following flags can be set (ORed) to configure a certain channel.

T845_GAIN_1	Select Gain 1x	-10 V…+10 V
T845_GAIN_2	Select Gain 2x	-5 V… +5 V
T845_GAIN_4	Select Gain 4x	-2.5 V… +2.5 V
T845_GAIN_8	Select Gain 8x	-1.25 V… +1.25 V
T845_SEDIFF	Configure this channel as differential	
T845_ENA	Tell the sequencer to add this channel to its conversion sequence	

To write configuration for a single ended channel *n = 1…T845_MAX_SINGLE_CHAN* use *config[T845_S2I(n)]* for a differential channel *n = 1…T845_MAX_DIFF_CHAN* use *config[T845_D2I(n)]*.

seq_timer

This parameter sets the sequencer period in steps of 100E-06 seconds. So a seq_timer value equals to 1 means 100E-06 sec. the max. sequencer period is 6,5535 sec.. Be careful with low sequencer timer periods, it may cause a lot of ISR work. A special case is a seq_timer value of 0. This starts the sequencer in a continuously mode. For more details see “TIP845 Hardware Manual”.

EXAMPLE

```
int  fd;
int  result;
T845_SEQ_CONF_BUFFER  buf;

/*
** Init diff. channel 1 and single ended channel 3 (2 is used by diff. 1)
*/
buf.config[T845_D2I(1)] = T845_GAIN_2 | T845_SEDIFF | T845_ENA;
buf.config[T845_S2I(3)] = T845_GAIN_1 | T845_ENA;
buf.seq_timer = 100;    /* sequencer period of 10 ms = 100 * 100E-06    */

result = ioctl(fd, T845_INIT_SEQUENCER, &buf);


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

ERRORS

No function specific errors will be returned.

SEE ALSO

ioctl man pages

3.4.3 T845_STOP_SEQUENCER

NAME

T845_STOP_SEQUENCER – Stop the sequencer and switch to manual mode

DESCRIPTION

This *ioctl* function stops the sequencer of the TIP845 associated with the open file descriptor *fd.* The parameter *arg* is unused. After this ioctl call, only manual ADC conversion is available. For more details see *read* request.

EXAMPLE

```
int  fd;
int  result;

result = ioctl(fd, T845_STOP_SEQUENCER, NULL);

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

ERRORS

No function specific errors will be returned.

SEE ALSO

ioctl man pages

4 Debugging and Diagnostic

If your installed IPAC port driver (e.g. tip845) 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, tip845.c,...).

The debug output should appear on the console. If not please check the symbol KKPF_PORT in uparam.h. This symbol should be configured to a valid COM port (e.g. SKDB_COM1).

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

```
TEWS TECHNOLOGIES - IPAC Carrier Class Driver version 1.0.0 (2003-11-28)
TEWS TECHNOLOGIES - VME Carrier version 1.0.0 (2003-12-05)
IPAC_CC  :  IPAC (Manuf-ID=B3, Model#=18) recognized @ slot=0 carrier=<TEWS TECHNOLOGIES - VME
Carrier>
TIP845 - 48 Channel 14 bit A/D Conversion version 1.0.0 (2004-09-14)
TIP845  :  Probe new TIP845 mounted on <TEWS TECHNOLOGIES - VME 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.