The Embedded I/O Company

TIP550-SW-72

LynxOS Device Driver

8(4) Channel 12-Bit DAC

Version 1.0.0

User Manual

Issue 1.0

December 2003

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

TEWS TECHNOLOGIES LLC
1 E. Liberty Street, Sixth Floor Reno, Nevada 89504 / USA
Phone: +1 (775) 686 6077 Fax: +1 (775) 686 6024
e-mail: usasales@tews.com www.tews.com

TIP550-SW-72

8(4) Channel 12-Bit DAC

LynxOS Device Driver

Issue	Description	Date
1.0	First Issue	December 12, 2003

Table of Contents

1 Introduction

The TIP550-SW-72 LynxOS device driver allows the operation of a TIP550 IPAC module on LynxOS operating systems.

Because the TIP550 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, write and ioctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TIP550 device driver includes the following functions:

- writing new values to the specified DAC channel
- reading device information data
- DAC gain/offset correction with factory calibration data stored in the on-board IDPROM
- TEWS TECHNOLOGIES IPAC carrier driver support.

2 Installation

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

The directory A:\TIP550-SW-72 contains the following files:

TIP550-SW-72.pdf	This manual in PDF format
TIP550-SW-72.tar	Device Driver and Example sources

The TAR archive TIP550-SW-72.tar contains the following files and directories:

tip550/tip550.c	Driver source code
tip550/tip550.h	Definitions and data structures for driver and application
tip550/tip550def.h	Definitions and data structures for the driver
tip550/tip550_info.c	Device information definition
tip550/tip550_info.h	Device information definition header
tip550/tip550.cfg	Driver configuration file include
tip550/tip550.import	Linker import file for PowerPC platforms
tip550/Makefile	Device driver make file
tip550/Example/example.c	Example application source
tip550/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/tip550 or /sys/drivers.cpci_x86/tip550

2. Copy the following files to this directory:
   - tip550.c
   - tip550def.h
   - tip550.import
   - Makefile

3. Copy tip550.h to /usr/include/

4. Copy tip550_info.c to /sys/devices.xxx or */sys/devices* if /sys/devices.xxx does not exist (xxx represents the BSP).

5. Copy tip550_info.h to /sys/dheaders/

6. Copy tip550.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/tip550, 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 = ... tip550_info.x
   ```

   And at the end of the Makefile

   ```
   tip550_info.o:$(DHEADERS)/tip550_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:tip550.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/tip550_0, /dev/tip550_1, /dev/tip550_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/tip550*, 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/devices.xxx/* or */sys/devices* if */sys/devices.xxx* does not exist (xxx represents the BSP).
2. To create a device definition file for the major device (this work only on native system)

   ```
   make t550info
   ```

3. To install the driver enter:

   ```
   drinstall –c tip550.obj
   ```

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

   ```
   devinstall –c –d <driver-ID> t550info
   ```

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

   ```
   mknod /dev/tip550_0 c <major_no> 0
   mknod /dev/tip550_1 c <major_no> 1
   mknod /dev/tip550_2 c <major_no> 2
   ...
   ```

   The <major_no> is returned by the devinstll command.

If all steps are successful completed the TIP550 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TIP550 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 TIP550 driver and devices into the LynxOS system, the configuration include file tip550.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tip550.cfg on the distribution disk contains the driver entry (*C:tip550:\....*) and a major device entry (*D:TIP550:t550info::*) with nine minor device entries (*“N: tip550_0:0”, ..., “N: tip550_8:8”*).

If the driver should support more than nine TIP550, additional minor device entries must be added. To create the device node */dev/tip550_9* the line *N:tip550_9:9* must be added at the end of the file tip550.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:tip550:\
     :t550open:t550close::t550write:\
     ::t550ioctl:t550install:t550uninstall
D:TIP550:t550info::
N:tip550_0:0
N:tip550_1:1
N:tip550_2:2
N:tip550_3:3
N:tip550_4:4
N:tip550_5:5
N:tip550_6:6
N:tip550_7:7
N:tip550_8:8
```

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

/dev/tip550_0 ... /dev/tip550_8

3 TIP550 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 (TIP550 device) named in path for reading and writing. The value of oflags indicates the intended use of the file. In case of a TIP550 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 TIP550 device already exists this argument is ignored.

EXAMPLE

```
int  fd


fd = open ("/dev/tip550_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 write()

NAME

write() – write to a file

SYNOPSIS

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

DESCRIPTION

This function attempts to write to the specified DAC channel of the TIP550 associated with the file descriptor *fd* from a structure (*T550_WRITE_BUFFER)* pointed by *buff*. The argument *count* specifies the length of the buffer and must be set to the length of the structure *T550_WRITE_BUFFER*.

The *T550_WRITE_BUFFER* structure has the following layout:

```
typedef struct {
        int     chan;
        int     corr;
        int     data;
} T550_WRITE_BUFFER;
```

chan

Selects the DAC channel. Valid channel numbers are 1...8 for TIP550-10 and channel numbers 1...4 for TIP550-11.

corr

Set this parameter to TRUE (1) to perform an automatic gain and offset correction with calibration data stored in the IDPROM. FLASE (0) means do not perform any correction and write directly to the DAC output.

data

Contains the new DAC output value.

Output Mode	data range	voltage range
Unipolar	0...4095	0...10V
Bipolar	-2048...2047	-10V...10V

EXAMPLE

```
int fd;
int  result;
T550_WRITE_BUFFER  wrBuf;


wrBuf.chan    = 1;
wrBuf.corr    = 0;          /* no data correction */
wrBuf.data    = 4095;       /* full-scale if unipolar output */

result = write(fd, &wrBuf, sizeof(T550_WRITE_BUFFER));

if (result < 0)) {
    // process error;
}
```

RETURNS

When write succeeds, 0 (OK) 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 the following TIP550 specific error code:

ENXIO	Illegal device
EINVAL	This error code is returned if the specified channel number is out of range.
EACCES	The output voltage mode isn't configured until now. Please call the ioctl function T550_CONFIG before.
EIO	DA conversion hasn't finished within the maximum allowed time period.
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).
ETIMEDOUT	The fix device access timeout has elapsed because other write requests to this device are pending.

SEE ALSO

LynxOS System Call - write()

3.4 ioctl()

NAME

ioctl() - I/O device control

SYNOPSIS

```
#include <ioctl.h>
#include <tip550.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 TIP550.h :

Value	Meaning
T550_CONFIG	Configure output voltage mode
T550_DEVINFO	Read device 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 TIP550 ioctl function returns always standard error codes.

SEE ALSO

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

3.4.1 T550_CONFIG

NAME

T550_CONFIG - Configure output voltage mode

DESCRIPTION

This ioctl function must be called to configure the output voltage mode for channel group 1 and 2 to match the hardware configuration of the device (jumper J1-4). Be sure that this ioctl function is called before the first write; otherwise write will fail (EACCES).

A pointer to the *T550_CONFIG_BUFFER* structure is passed by the parameter *arg* to the driver.

The *T550_CONFIG_BUFFER* structure has the following layout:

```
typedef struct {
        int     output_mode_1;
        int     output_mode_2;
} T550_CONFIG_BUFFER;
```

output_mode_1, output_mode_2

These structure elements specifies the output voltage mode for channel group 1 (channel 1...4) and channel group 2 (channel 5...8). For TIP550-11 devices the channel group 2 isn't relevant (only 4 channels at all) and should be set to 0.

Each channel group can be configured separately either for unipolar (*T550_UNIPOLAR*) or bipolar (*T550_BIPOLAR*) output.

symbol	voltage range
T550_UNIPOLAR	0V ... 10V
T550_BIPOLAR	-10V ... 10V

EXAMPLE

```
int  fd;
int  result;
T550_CONFIG_BUFFER  confBuf;


confBuf.output_mode_1 = T550_UNIPOLAR;
confBuf.output_mode_2 = T550_BIPOLAR;

result = ioctl(fd, T550_CONFIG, &confBuf);

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

ERRORS

EINVAL	Invalid output voltage mode.

SEE ALSO

ioctl man pages

3.4.2 T550_DEVINFO

NAME

T550_DEVINFO - Read device information data

DESCRIPTION

This ioctl function reads the module variant, the current output voltage setting and the factory calibration data from the specified device and returns this information in the *T550_INFO_BUFFER* structure to the caller.

A pointer to the *T550_INFO_BUFFER* structure is passed by the parameter *arg* to the driver.

The *T550_INFO_BUFFER* structure has the following layout:

```
typedef struct {
        int     variant;
        int     output_mode_1;
        int     output_mode_2;
        int     offset_corr[8];
        int     gain_corr[8];
} T550_INFO_BUFFER;
```

variant

Returns the module variant.

value	module variant
10	TIP550-10
11	TIP550-11

output_mode_1

Returns the actual output voltage mode for channel group 1 (channel 1...4).

value	symbol	voltage range
1	T550_UNIPOLAR	0V ... 10V
2	T550_BIPOLAR	-10V ... 10V

output_mode_2

Returns the actual output voltage mode for channel group 2 (channel 5...8). For TIP550-11 modules -1 is returned.

value	symbol	voltage range
1	T550_UNIPOLAR	0V ... 10V
2	T550_BIPOLAR	-10V ... 10V

offset_corr

Returns the factory offset calibration data for channel 1...8 in the unit ¼ LSB. For TIP550-11 modules only the values for channel 1..4 are valid.

gain_corr

Returns the factory gain calibration data for channel 1...8 in the unit ¼ LSB. For TIP550-11 modules only the values for channel 1..4 are valid.

EXAMPLE

```
int  fd;
int  result;
T550_INFO_BUFFER  infoBuf;


result = ioctl(fd, T550_DEVINFO, &infoBuf);

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. tip550) 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, tip550.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#=19) recognized @ slot=0 carrier=<TEWS TECHNOLOGIES - VME
Carrier>
TIP550 - TIP550 - 8(4) Channel 12-Bit D/A version 1.0.0 (2003-12-12)
TIP550  :  Probe new TIP550 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.