The Embedded I/O Company

TEWS TECHNOLOGIES

TIP700-SW-72

LynxOS Device Driver

16 Channel Digital Output

Version 1.0.x

User Manual

Issue 1.0.0

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

TIP700-SW-72

LynxOS Device Driver

16 Channel Digital Output

Supported Modules:
TIP700

Issue	Description	Date
1.0.0	First Issue	September 14, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP700-SW-72 LynxOS device driver allows the operation of the TIP700 digital output module conforming to the LynxOS I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close()*, and *ioctl()* functions.

The TIP700-SW-72 device driver supports the following features:

- writing output value
- reading current output value
- enable, disable and trigger output watchdog

The TIP700-SW-72 device driver supports the modules listed below:

TIP700	16 Channel digital Output	(IndustryPack®)

To get more information about the features and use of TIP700 devices it is recommended to read the manuals listed below.

TIP700 User manual

TIP700 Engineering Manual

CARRIER-SW-72 User Manual

1.2 IPAC Carrier Driver

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 differences of different carrier boards under a well defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-72 is part of this TIP700-SW-72 distribution. It is located in directory CARRIER-SW-72 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-72 User Manual for a detailed description how to install and setup the CARRIER-SW-72 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

Following files are located on the distribution media:

Directory path 'TIP700-SW-72':

TIP700-SW-72-SRC.tar.gz	GZIP compressed archive with driver source code
TIP700-SW-72-1.0.0.pdf	PDF copy of this manual
ChangeLog.txt	Release history
Release.txt	Release information

For installation the files have to be copied to the desired target directory.

The GZIP compressed archive TIP700-SW-72-SRC.tar.gz contains the following files and directories:

Directory path 'tip700':

tip700.c	TIP700 device driver source
tip700def.h	TIP700 driver include file
tip700.h	TIP700 include file for driver and application
tip700_info.c	TIP700 Device information definition
tip700_info.h	TIP700 Device information definition header
tip700.cfg	TIP700 Driver configuration file include
tip700.import	Linker import file
Makefile	Device driver make file
example/tip700exa.c	Example application
example/Makefile	Example application makefile

In order to perform an installation, extract all files of the archive TIP700-SW-72-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP700-SW-72-SRC.tar.gz' will extract the files into the local directory.

2.1 Device Driver Installation

The two methods of driver installation are as follows:

(1) Static Installation
(2) Dynamic Installation (only native LynxOS 4 systems)

Please always install the Carrier Driver (CARRIER-SW-72) first.

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/tip700, 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 = … tip700_info.x
   ```

   And at the end of the Makefile

   ```
   tip700_info.o:$(DHEADERS)/tip700_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.

   ```
   I:tip700.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/tip700a, /dev/tip700b, …

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.

This method is only available for systems before LynxOS 5.0

2.1.2.1 Build the driver object

1. Change to the directory /sys/drivers.xxx/tip700, where xxx represents the BSP that supports the target hardware.
2. To make the dynamic link-able driver enter:

   ```
   make dldd
   ```

2.1.2.2 Create Device Information Declaration

1. Change to the directory /sys/drivers.xxx/tip700, 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 systems)

   ```
   make t700info
   ```

3. To install the driver enter:

   ```
   drinstall –c tip700.obj
   ```

   If successful, drinstall returns a unique <driver-ID>

4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t700info
   ```

   The <driver-ID> is returned by the drinstall command

5. To create the node for the devices enter:

   ```
   mknod /dev/tip700a c <major_no> 0
   ```

The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TIP700 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TIP700 device enter the following commands:

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

2.1.3 Device Information Definition File

The device information definition contains information necessary to install the TIP700 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tip700_info.h*.

There are no configurable parameters.

2.1.4 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 TIP700 driver and devices into the LynxOS system, the configuration include file tip700.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tip700.cfg on the distribution disk contains the driver entry (*C:tip700:\....*) and a major device entry (*D:TIP700:t700info::*) with 4 minor device entries (*“N: tip700a:0”, ..., “N: tip700d:3”*).

If the driver should support more than four TIP700, additional minor device entries must be added. To create the device node */dev/tip700e* the line *N:tip700e:4* must be added at the end of the file tip700.cfg. For the next node a minor device entry with 5 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:tip700:t700open:t700close::::t700ioctl:t700install:t700uninstall
D:TIP700:t700_info::
N:tip700a:0
N:tip700b:1
N:tip700c:2
N:tip700d:3
```

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

/dev/tip700a ... /dev/tip700d

3 TIP700 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 (TIP700 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TIP700 device ***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 TIP700 device already exists this argument is ignored.

EXAMPLE

```
int fd

fd = open ("/dev/tip700a", O_RDWR);
if (fd == -1)
{
     /* Handle error */
}
```

RETURNS

open returns a file descriptor number if successful, or –1 on error.

SEE ALSO

LynxOS System Call - open()

3.2 close()

NAME

close() – close a file

SYNOPSIS

int close(int fd)

DESCRIPTION

This function closes an opened device.

EXAMPLE

```
int  result;

result = close(fd);
if (result == -1)
{
     /* Handle error */
}
```

RETURNS

close returns 0 (OK) if successful, or –1 on error

SEE ALSO

LynxOS System Call - close()

3.3 ioctl()

NAME

ioctl() – I/O device control

SYNOPSIS

#include <ioctl.h>
#include <tip700.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 supported by the driver and are defined in *tip700.h*:

Symbol	Meaning
TIP700_WRITE	Set output value
TIP700_READ	Read current output value
TIP700_WDENABLE	Enable output watchdog
TIP700_WDDISABLE	Disable output watchdog
TIP700_WDTRIGGER	Trigger output watchdog (rewrite current output value)

See behind for more detailed information on each control code.

RETURNS

ioctl returns 0 if successful, or –1 on error.

On error, *errno* will contain a standard error code (see also LynxOS System Call – ioctl).

SEE ALSO

LynxOS System Call - ioctl().

3.3.1 TIP700_WRITE

NAME

TIP700_WRITE – Set output value

DESCRIPTION

This function sets the output value of the device. A pointer to the new output value (unsigned short) must be passed by the parameter *arg* to the device. Bit-0 specified the new value of Output 1, bit-1 the value of Output 2 and so on.

EXAMPLE

```
#include <tip700.h>

int                     fd;
int                     result;
unsigned short          outVal;

/* --- set all lines off, exept Output 4 shall be on --- */
outVal = 0x0008;

result = ioctl(fd, TIP700_WRITE, (char*)&outVal);
if (result >= 0)
{
     /* New output set */
}
else
{
     /* Setting output failed */
}
```

3.3.2 TIP700_READ

NAME

TIP700_READ – Get current output value

DESCRIPTION

This function reads the current output value of the device. There may be a difference between the output value and the state of the output lines, if the watchdog is enabled. Keep in mind this function returns the last written output value. A pointer on a buffer for the returned output value (unsigned short) must be passed by the parameter *arg* to the device. Bit-0 will return the output value of Output 1, bit-1 the output value of Output 2 and so on.

EXAMPLE

```
#include <tip700.h>

int                     fd;
int                     result;
unsigned short          outVal;

/* --- Get current output value --- */
result = ioctl(fd, TIP700_READ, (char*)&outVal);
if (result >= 0)
{
     printf("Current output value: %04Xh\n", outVal);
}
else
{
     /* Setting output failed */
}
```

3.3.3 TIP700_WDENABLE

NAME

TIP700_WDENABLE – Enable output watchdog

DESCRIPTION

This function enables the output watchdog of the device. There is no function dependent parameter and the parameter *arg* can be set to *NULL*.

EXAMPLE

```
#include <tip700.h>

int                     fd;
int                     result;

/* --- Enable output watchdog --- */
result = ioctl(fd, TIP700_WDENABLE, NULL);
if (result >= 0)
{
     /* Watchdog enabled */
}
else
{
     /* Setting output failed */
}
```

3.3.4 TIP700_WDENABLE

NAME

TIP700_WDDISABLE – Disable output watchdog

DESCRIPTION

This function disables the output watchdog of the device. There is no function dependent parameter and the parameter *arg* can be set to *NULL*.

EXAMPLE

```
#include <tip700.h>

int                     fd;
int                     result;

/* --- Disable output watchdog --- */
result = ioctl(fd, TIP700_WDDISABLE, NULL);
if (result >= 0)
{
     /* Watchdog disabled */
}
else
{
     /* Setting output failed */
}
```

3.3.5 TIP700_WDTRIGGER

NAME

TIP700_WDTRIGGER – Enable output watchdog

DESCRIPTION

This function triggers the output watchdog of the device. The watchdog trigger is implemented as a rewrite of the last written output value. There is no function dependent parameter and the parameter *arg* can be set to *NULL*.

EXAMPLE

```
#include <tip700.h>

int                     fd;
int                     result;

/* --- Trigger output watchdog --- */
result = ioctl(fd, TIP700_WDTRIGGER, NULL);
if (result >= 0)
{
     /* Watchdog triggered */
}
else
{
     /* Setting output failed */
}
```

4 Debugging and Diagnostic

If your installed IPAC port driver (e.g. tip700) 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, tip700.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 1.3.0 (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#=05) 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
TIP700 Digital Output driver version 1.0.0 (2009-09-14)
IPAC_CC  :  IPAC driver <TIP700 Digital Output driver> registered
IPAC_CC  :  Call probe function of <TIP700 Digital Output driver> for
module [179/5]
TIP700  :  Probe new TIP700 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.