The Embedded I/O Company

TEWS TECHNOLOGIES

TIP600-SW-72

LynxOS Device Driver

16 Channel Digital Input

Version 2.0.x

User Manual

Issue 2.0.0

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

TIP600-SW-72

LynxOS Device Driver

16 Channel Digital Input

Supported Modules:
TIP600

Issue	Description	Date
1.0.0	First Issue	March 27, 2003
2.0.0	General Revision, Carrier Support added, reviewed driver interface	September 10, 2009

Table of Contents

1 Introduction

1.1 Device Driver

The TIP600-SW-72 LynxOS device driver allows the operation of the TIP600 digital input 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 TIP600-SW-72 device driver supports the following features:

- reading current state of input lines (immediately)
- waiting for events on the input lines and than reading the state of the input lines
- configuring input debouncer

The TIP600-SW-72 device driver supports the modules listed below:

TIP600	16 Channel digital Input	(IndustryPack®)

To get more information about the features and use of TIP600 devices it is recommended to read the manuals listed below.

TIP600 User manual

TIP600 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 TIP600-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 'TIP600-SW-72':

TIP600-SW-72-SRC.tar.gz	GZIP compressed archive with driver source code
TIP600-SW-72-2.0.0.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 TIP600-SW-72-SRC.tar.gz contains the following files and directories:

Directory path 'tip600':

tip600.c	TIP600 device driver source
tip600def.h	TIP600 driver include file
tip600.h	TIP600 include file for driver and application
tip600_info.c	TIP600 Device information definition
tip600_info.h	TIP600 Device information definition header
tip600.cfg	TIP600 Driver configuration file include
tip600.import	Linker import file
Makefile	Device driver make file
example/tip600exa.c	Example application
example/Makefile	Example application makefile

In order to perform an installation, extract all files of the archive TIP600-SW-72-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TIP600-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/tip600, 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 = … tip600_info.x
   ```

   And at the end of the Makefile

   ```
   tip600_info.o:$(DHEADERS)/tip600_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:tip600.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/tip600a, /dev/tip600b, …

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/tip600, 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/tip600, 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 t600info
   ```

3. To install the driver enter:

   ```
   drinstall –c tip600.obj
   ```

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

4. To install the major device enter:

   ```
   devinstall –c –d <driver-ID> t600info
   ```

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

5. To create the node for the devices enter:

   ```
   mknod /dev/tip600a c <major_no> 0
   ```

The <major_no> is returned by the devinstall command.

If all steps are successfully completed, the TIP600 is ready to use.

2.1.2.3 Uninstall dynamic loaded driver

To uninstall the TIP600 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 TIP600 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tip600_info.h*.

This structure contains the following parameter:

debounceTime Defines the default debouncer time. The value is specified in us. A “0” disables input debouncing.

The following device declaration information defines a default debouncer time of ~ 1ms.

```
TIP600_INFO tip600 = {

     1000
};
```

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 TIP600 driver and devices into the LynxOS system, the configuration include file tip600.cfg must be included in the CONFIG.TBL (see also 2.1.1.3).

The file tip600.cfg on the distribution disk contains the driver entry (*C:tip600:\....*) and a major device entry (*D:TIP600:t600info::*) with 4 minor device entries (*"N: tip600a:0", ..., "N: tip600d:3"*).

If the driver should support more than four TIP600, additional minor device entries must be added. To create the device node */dev/tip600e* the line *N:tip600e:4* must be added at the end of the file tip600.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:tip600:t600open:t600close::::t600ioctl:t600install:t600uninstall
D:TIP600:t600_info::
N:tip600a:0
N:tip600b:1
N:tip600c:2
N:tip600d:3
```

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

/dev/tip600a ... /dev/tip600d

2.2 Maximum Number of Active Jobs Configuration

The maximum number of active event read jobs per major device can be configured. This can be simply made by changing the value of the symbol in tip600def.h.

MAX_REQUESTS	Defines the maximum number of active event wait jobs (default = 5). Valid numbers are in range between 1 and MAXINT.

3 TIP600 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 (TIP600 device) named in ***path*** for reading and writing. The value of ***oflags*** indicates the intended use of the file. In case of a TIP600 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 TIP600 device already exists this argument is ignored.

EXAMPLE

```
int fd

fd = open ("/dev/tip600a", 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 <tip600.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 *tip600.h*:

Symbol	Meaning
TIP600_READ	Read state of the digital input lines
TIP600_DEBCONFIG	Configure input debouncer

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 TIP600_READ

NAME

TIP600_READ – Read state of input lines

DESCRIPTION

This function reads the state of the input lines. The read operation is performed immediately when calling the function, or after a specified event has occurred. A pointer to the caller's read buffer (*TIP600_READ_BUFFER*) must be passed by the parameter *arg* to the device.

```
typedef struct
{
        unsigned short    value;      /* value on lines of Port */
        unsigned short    mask;       /* mask for lines of Port */
        unsigned short    match;      /* pattern creating event for Port */
        unsigned char     mode;       /* read mode */
        long              timeout;    /* timeout in ticks */
} TIP600_READ_BUFFER;
```

Members

mode

Specifies when the input state will be read. The following read modes are supported, and appropriate defines can be found in tip600.h:

Mode	Description
TIP600_NOWAIT	The state of the input lines will be read immediately.
TIP600_MATCH	The function will wait for a match event. Note: This mode is not recommended for use, especially with fast signal changes. Caused by execution delay and runtimes valid events may be overseen.
TIP600_HIGH_TR	The function will wait for a low to high transition on any of the input lines specified in mask
TIP600_LOW_TR	The function will wait for a high to low transition on any of the input lines specified in mask
TIP600_ANY_TR	The function will wait for any transition on any of the input lines specified in mask

mask

This function specifies the input lines the driver will use to wait for the specified transition. A set bit specifies the appropriate input line. Bit-0 specifies Input 1, bit-1 specifies Input 2, and so on. If the *mode* is set to *TIP600_NOWAIT* this parameter is unused.

match

This parameter is used only with mode set to *TIP600_MATCH*. It specifies the value that must occur on the input lines for the masked lines (*mask*).

value

This value returns the state of the input lines at the moment specified by *mode*. Bit-0 returns the state of Input 1, bit-1 the state of Input 2, and so on.

There is a delay between the occurrence of the specified event and reading the input value, which is based on the system and OS dependent interrupt latency.

timeout

This parameter specifies the maximum wait time (ticks), before the function terminates and returns with an error.
If the *mode* is set to *TIP600_NOWAIT* this parameter is unused.

EXAMPLE

```
#include <tip600.h>

int                     fd;
int                     result;
TIP600_READ_BUFFER      rdBuf;

/* --- read current input state (immediately) --- */
rdBuf.mode = TIP600_NOWAIT;

result = ioctl(fd, TIP600_READ, (char*)&rdBuf);
if (result >= 0)
{
     printf("INPUT: %04Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}
```

…

```
…

/* --- read input state after Input 4 switched from low to high --- */
rdBuf.mode    = TIP600_HIGH_TR;
rdBuf.mask    = (1 << 3);            /* set bit for Input 4 */
rdBuf.timeout = 10000;               /* timeout after 10000 ticks */

result = ioctl(fd, TIP600_READ, (char*)&rdBuf);
if (result >= 0)
{
     printf("INPUT: %04Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}

…

/* --- read if value on Input 1-4 (xxx5h) and 9-12 (xAxxh) matches --- */
rdBuf.mode    = TIP600_MATCH;
rdBuf.mask    = 0x0F0F;          /* mask bits 5-8 and 13-16 out */
rdBuf.match   = 0x0A05;          /* match value */
rdBuf.timeout = 10000;           /* timeout after 10000 ticks */

result = ioctl(fd, TIP600_READ, (char*)&rdBuf);
if (result >= 0)
{
     printf("INPUT: %04Xh\n", rdBuf.value);
}
else
{
     /* Read failed */
}
```

ERRORS

EINTR	The function was cancelled.
ETIMEDOUT	The maximum allowed time to finish the read request is exhausted.
EINVAL	An unsupported input parameter has been specified: - unknown read mode
EBUSY	The maximum number of active event jobs at the same time has been reached.

Other returned error codes are system error conditions.

3.3.2 TIP600_DEBCONFIG

NAME

TIP600_DEBCONFIG – Configure input debouncer

DESCRIPTION

This function configures the input debouncer of the device. A pointer to the new debouncer value (unsigned int) must be passed by the parameter *arg* to the device. The value specifies the debouncer time, i.e. the time a signal must be stable to be recognized. The debouncer time is specified in steps of µs, a value < 8 will disable the debouncing unit. The value is used to calculate the next supported debouncer time matching to the granularity of the TIP600. The maximum debouncer time is about 520 ms.

Keep in mind that the recognition of the input changes is delayed by debouncer time.

EXAMPLE

```
#include <tip600.h>

int                     fd;
int                     result;
int                     debTime;

/* --- configure a debouncer time of 40ms --- */
debTime = 40000;

result = ioctl(fd, TIP600_DEBCONFIG, (char*)&debTime);
if (result >= 0)
{
     /* OK */
}
else
{
     /* Read failed */
}
```

ERRORS

EINVAL	The specified debouncer time exceeds the maximum value.

Other returned error codes are system error conditions.

4 Debugging and Diagnostic

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

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

The 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#=04) recognized @ slot=0 carrier=<TEWS
TECHNOLOGIES - (Compact)PCI IPAC Carrier>
IPAC_CC  :  IPAC access failed (slot=1, carrier=<TEWS TECHNOLOGIES -
(Compact)PCI IPAC Carrier>)
IPAC_CC  :  carrier driver <TEWS TECHNOLOGIES - SBS (Compact)PCI IPAC
Carrier> registered
TIP600 Digital Input driver version 2.0.0 (2009-09-10)
IPAC_CC  :  IPAC driver <TIP600 Digital Input driver> registered
IPAC_CC  :  Call probe function of <TIP600 Digital Input driver> for module
[179/4]
TIP600  :  Probe new TIP600 mounted on <TEWS TECHNOLOGIES - (Compact)PCI
IPAC Carrier> at slot A
```

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