

TIP570-SW-72

LynxOS Device Driver

16/8 Channel 12 Bit ADC

and

8 Channel 12 Bit DAC

User Manual

Issue 1.0 Version 1.0.0

January 2003

TIP570-SW-72

16/8 Channel 12 Bit ADC and 8 Channel 12 Bit DAC

LynxOS Device Driver

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2003 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	January 24, 2003

Table of Content

1	INTRODUCTION.....	4
2	INSTALLATION.....	5
	2.1 Device Driver Installation	5
	2.1.1 Static Installation	5
	2.1.1.1 Build the driver object	5
	2.1.1.2 Create Device Information Declaration	6
	2.1.1.3 Modify the Device and Driver Configuration File	6
	2.1.1.4 Rebuild the Kernel.....	6
	2.1.2 Dynamic Installation	7
	2.1.3 Device Information Definition File	8
	2.1.4 Configuration File: CONFIG.TBL	9
3	TIP570 DEVICE DRIVER PROGRAMMING.....	10
	3.1 open()	10
	3.2 close().....	12
	3.3 read()	13
	3.4 write()	16
4	DEBUGGING AND DIAGNOSTIC.....	18

1 Introduction

The TIP570-SW-72 LynxOS device driver allows the operation of a TIP570 16/8 Channel 12 Bit ADC and 8 Channel 12 Bit DAC IP on PowerPC platforms.

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

The TIP570 device driver includes the following functions:

- Reading analog input value from a specified ADC channel
- Select input gains
- Using pipeline mode for input
- Support of single-ended and differential input
- Auto correction of input values with factory stored correction data
- Writing analog output value to a specified DAC channel
- Using latched mode for output
- Auto correction of output values with factory stored correction data

To understand all features of this device driver, it is recommended to read the TIP570 User Manual.

2 Installation

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

Following files are located on the diskette:

<code>tip570.c</code>	Driver source code
<code>tip570.h</code>	Definitions and data structures for driver and application
<code>tip570def.h</code>	Definitions and data structures for the driver
<code>tip570_info.c</code>	Device information definition
<code>tip570_info.h</code>	Device information definition header
<code>tip570.cfg</code>	Driver configuration file include
<code>tip570.import</code>	Linker import file
<code>Makefile</code>	Device driver make file
<code>Makefile.dldd</code>	Make file for dynamic driver installation
<code>example/example.c</code>	Example application source
<code>tip570-sw-72.pdf</code>	This Manual in PDF format

2.1 Device Driver Installation

The two methods of driver installation are as follows:

- Static Installation
- Dynamic Installation (only native LynxOS systems)

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.

In order to perform a static installation, copy the following files to their target directories:

1. Create a new directory in the system driver directory path `/sys/drivers.xxx`, where `xxx` represents the BSP that supports the target hardware. For example: `/sys/drivers.pp_drm/tip570`
2. Copy the following files to this directory: `tip570.c`, `tip570def.h`, `Makefile`
3. Copy `tip570.h` to `/usr/include/`
4. Copy `tip570_info.c` to `/sys/devices.xxx/` or `/sys/devices` if `/sys/devices.xxx` does not exist (`xxx` represents the BSP).
5. Copy `tip570_info.h` to `/sys/dheaders/`
6. Copy `tip570.cfg` to `/sys/cfg.ppc/`

2.1.1.1 Build the driver object

1. Change to the directory `/sys/drivers.xxx/tip570`, 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 = ... tip570_info.x`
3. And at the end of the Makefile
`Tip570_info.o:$(DHEADERS)/tip570_info.h`
4. 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`
`I:tip570.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`
4. The **N** flag instructs *init* to run *mknod* and create all the nodes mentioned in the new *nodetab*.
5. After reboot you should find the following new devices (depends on the device configuration):
`/dev/t570a1, ...]`

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.

The following steps describe how to do a dynamic installation:

1. Create a new directory in the system driver directory path `/sys/drivers.xxx`, where `xxx` represents the BSP that supports the target hardware. For example:
`/sys/drivers.pp_drm/tip570`

1. Copy the following files to this directory:

```
tip570.c
tip570def.h
tip570_info.c
tip570_info.h
tip570.import
Makefile.dldd
```

2. Copy `tip570.h` to `/usr/include`
3. Change to the directory `/sys/drivers.xxx/tip570`
4. To make the dynamic link-able driver enter:

```
make -f Makefile.dldd
```

5. Create a device definition file for one major device

```
gcc -DDLDD -o tip570_info tip570_info.c
./tip570_info > t570a_info
```

6. To install the driver enter:

```
drinstall -c tip570.obj

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

7. To install the major device enter:

```
devinstall -c -d <driver-ID> t570a_info
```

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

8. To create nodes for the devices enter:

```
mknod /dev/t570a1 c <major_no> 0...
```

If all steps are successful completed the TIP570 is ready to use.

To uninstall the TIP570 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 TIP570 major device.

The implementation of the device information definition is done through a C structure, which is defined in the header file *tip570_info.h*.

This structure contains the following parameter:

IpIoVirtualAddress

This parameter contains the kernel virtual address of the IP I/O space. This address depends on the configuration of the IP carrier board. In case of a VMEbus carrier this space usually appears in the VMEbus short I/O space A16/D16.

IpIdVirtualAddress

This parameter contains the kernel virtual address of the IP ID space (ID-PROM). This address depends on the configuration of the IP carrier board. In case of a VMEbus carrier this space usually appears in the VMEbus short I/O space A16/D16.

If the TIP570 is plugged on a VMEbus carrier be sure that the appropriate VMEbus driver *uvme* or *vme* is started (CONFIG.TBL). See also *Chapter 5 – Accessing Hardware* in the "Writing Device Drivers for LynxOS" manual and the man pages *uvmedrvr* and *vmedrvr* for information about the VMEbus configuration and mapping. Be sure that the used VME addressing modes (A16/D32) are enabled.

A device information definition is unique for every TIP570 major device. The file *tip570_info.c* on the distribution disk contains a device information declaration.

If the driver should support more major devices it is necessary to copy and paste an existing declaration and rename it with unique name for example **t570b_info**, **t570c_info** and so on.

It is also necessary to modify the device and driver configuration file, respectively the configuration include file *tip570.cfg*.

The following device declaration information expected that the IP spaces appear at virtual address 0xCFFF8000 for IP I/O and at virtual address 0xCFFF8080 for IP ID space

```
T570_INFO t570a_info =
{
    0xCfff8000      /* IP I/O Space      */
    0xCfff8080      /* IP ID Space       */
};
```


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 reads the file and produces a new set of driver and device configuration tables and a corresponding *nodetab*.

To install the TIP570 driver and devices into the LynxOS system, the configuration include file *tip570.cfg* must be included in the *CONFIG.TBL* (see also).

The file *tip570.cfg* on the distribution disk contains the driver entry (C:tip570:\....) and one enabled major device entry (D:TIP570 1:t570a_info::) with one minor device entry (N: t570a1:0).

If the driver should support more than one major device (TIP570) the following entries for major and minor devices must be enabled by removing the comment character (#). By copy and paste an existing major and minor entry and renaming the new entries, it is possible to add any number of additional TIP570 devices.

The name of the device information declaration (info-block-name) must match to an existing C structure in the file *tip570_info.c*.

This example shows a driver entry with one major device:

```
#   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:tip570:\
    :t570open:t570close:t570read:::::t570install:t570uninstall
D:TIP570 1:t570a_info::
N:t570a1:0
```

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

/dev/t570a1

3 TIP570 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 (TIP570 device) named in *path* for reading and writing. The value of *oflags* indicates the intended use of the file. In case of a TIP570 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 TIP570 device already exists this argument is ignored.

EXAMPLE

```
int  fd

...

/*
**  open the device named "/dev/t570a1" for I/O
*/
fd = open ("/dev/t570a1", O_RDWR);

...
```

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;

...

/*
**  close the device
*/
result = close(fd);

...
```

RETURNS

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

SEE ALSO

LynxOS System Call - close()

3.3 read()

NAME

read() - read from a file

SYNOPSIS

```
#include <tip570.h>
```

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

DESCRIPTION

The read function reads an analog input value from the specified channel. A pointer to the callers message buffer (*T570_READ_BUF*) and the size of this structure, is passed by the parameters **buff** and **count** to the device.

The function will use the fastest possible mode. If it is necessary to wait a settling time after changing channel, gain or input interface, the driver will do it, if the parameters do not change, the settling time will be ignored.

The *T570_READ_BUF* structure has the following layout:

```
typedef struct
{
    int          channel;
    int          gain;
    int          flags;
    long         value;
} T570_READ_BUF, *PT570_READ_BUF;
```

channel

This parameter specifies the ADC channel number for conversion. Allowed channel numbers are 1 up to 8 for differential mode and 1 up to 16 for single-ended mode.

gain

This value specifies the input gain. Allowed values are 1, 2, 5 and 10.

flags

This parameter is ORed value of the defines, that enables or disables special features.

Input interface	<i>T570_FL_SINGL</i>	The conversion will be executed on a single-ended input
	<i>T570_FL_DIFF</i>	The conversion will be executed on a differential input
Data correction	<i>T570_FL_RAWDATA</i>	Returns the converted data without any data correction
	<i>T570_FL_CORRECTION</i>	Make a data correction with the converted data and the factory stored correction data. Return the corrected value.
Pipeline mode	<i>T570_FL_NOPIPELINE</i>	The pipeline mode is disabled.
	<i>T570_FL_PIPELINE</i>	The pipeline mode is enabled.

value

This parameter returns the converted data value. Values are between -2048 and +2047.

EXAMPLE

```
int          fd;
int          result;
T570_READ_BUF ReadBuf;

...

ReadBuf.channel = 5;           /* Use channel 5 */
ReadBuf.gain = 2;             /* Gain: 2 (max. 5V) */
ReadBuf.flags = T570_FL_RAWDATA; /* Do not make data correction */
ReadBuf.flags |= T570_FL_DIFF;  /* Use differential interface */
ReadBuf.flags |= T570_FL_NOPIPELINE; /* The pipeline mode is disabled */

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

/*
** Check the result of the last device I/O operation
*/
if( result == sizeof(T570_READ_BUF))
{
    /* OK */
    printf("Input value: %d\n", ReadBuf.value);
}
else
{
    printf( "\nRead failed --> Error = %d.\n", errno );
}

...
```

RETURNS

When *read* succeeds, the size of the read value 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)

SEE ALSO

LynxOS System Call - read()

TIP570 example application

3.4 write()

NAME

write() – write to a file

SYNOPSIS

```
#include <tip570.h>
```

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

DESCRIPTION

The write function writes a new value to a specified DAC output channel. A pointer to the callers message buffer (*T570_WRITE_BUF*) and the size of this structure, is passed by the parameters **buff** and **count** to the device.

The *T570_WRITE_BUF* structure has the following layout:

```
typedef struct
```

```
{
    int          channel;
    int          flags;
    long         value;
} T570_WRITE_BUF, *PT570_WRITE_BUF;
```

channel

This parameter specifies the DAC channel number for conversion. Allowed channel numbers are 1 up to 8.

flags

This parameter is ORed value of the defines, that enables or disables special features.

Data correction	<i>T570_FL_RAWDATA</i>	Write the specified new output value without correction.
	<i>T570_FL_CORRECTION</i>	Use the factory stored correction data to correct the specified new output data.
Latched mode	<i>T570_FL_NOLATCH</i>	Write the specified data in transparent mode.
	<i>T570_FL_LATCH</i>	Write the specified data into the DAC register, but don not output the new value, use the latched mode.
	<i>T570_FL_LATCH_CONV</i>	Use the latched mode and output all last written values.

value

This specifies the new output data. Valid values are between -2048 and +2047.

EXAMPLE

```
int          fd;
int          result;
T570_WRITE_BUF WriteBuf;

...

WriteBuf.channel = 5;           /* Use channel 5 */
WriteBuf.flags =  T570_FL_RAWDATA; /* Do not make data correction */
WriteBuf.value =  0x400;        /* output ~5V */

result = write(fd, (char*)&WriteBuf, sizeof(WriteBuf));

/*
** Check the result of the last device I/O operation
*/
if( result == sizeof(T570_WRITE_BUF))
{
    /* OK */
}
else
{
    printf( "\nWrite failed --> Error = %d.\n", errno );
}

...
```

RETURNS

When *write* succeeds, the size of the write value 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)

SEE ALSO

LynxOS System Call - write()

TIP570 example application

4 Debugging and Diagnostic

This driver was successful tested on a Motorola MVME2306-900 board with a Windows Cross development with LynxOS V4.0.0.

If the driver will not work properly please enable debug outputs by removing the comments around the symbol *DEBUG*.

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 debug output displays the device information data for the current major device, a memory dump of the IP ID space (contents of the ID-PROM) and a memory dump of the IP I/O space (registers) like this, followed by the factory programmed correction data for the ADC and DAC channels.

TIP570 Device Driver Install

IP I/O Virtual Address = CFFF8000

IP ID Virtual Address = CFFF8080

IP ID space (ID-PROM)...

```
CFFF8080 : 00 49 00 50 00 41 00 43 00 B3 00 2C 00 10 00 00
CFFF8090 : 00 00 00 00 00 0D 00 08 00 0A 00 FF 00 FF 00 FF
CFFF80A0 : 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
CFFF80B0 : 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
```

IP I/O space (Registers)...

```
CFFF8000 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8040 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8070 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
CFFF8080 : 00 49 00 50 00 41 00 43 00 B3 00 2C 00 10 00 00
CFFF8090 : 00 00 00 00 00 0D 00 08 00 0A 00 FF 00 FF 00 FF
CFFF80A0 : 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
CFFF80B0 : 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF 00 FF
CFFF80C0 : 00 21 00 00 00 21 00 00 00 21 00 00 00 21 00 00
CFFF80D0 : 00 21 00 00 00 21 00 00 00 21 00 00 00 21 00 00
CFFF80E0 : 00 21 00 00 00 21 00 00 00 21 00 00 00 21 00 00
CFFF80F0 : 00 21 00 00 00 21 00 00 00 21 00 00 00 21 00 00
```

...

...

Correctiondata: [gain/offset]

```
ADC[0]    4/-2
ADC[1]    4/-2
ADC[2]    4/-2
ADC[3]    7/-2
DAC[0]   -7/-1
DAC[1]    0/-4
DAC[2]   -1/-1
DAC[3]   -2/-3
DAC[4]    1/1
DAC[5]   -5/-2
DAC[6]   -2/-1
DAC[7]   -4/-3
```

The debug output above is only an example. Debug output on other systems may be different for addresses and data in some locations.