TIP610-SW-95
QNX-Neutrino Device Driver
TIP610 – 16/20 Channel Digital I/O on SBS PCI40 Carrier

Version 1.0.x

Reference Manual
Issue 1.0

January 2002

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7
D-25469 Halstenbek
Germany
Tel.: +49 (0)4101 4058-0
Fax.: +49 (0)4101 4058-19
http://www.tews.com
e-mail: info@tews.com

TIP610-SW-95
16/20 Channel Digital I/O
QNX-Neutrino Device Driver

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

This product has been designed to operate with IndustryPack® compatible carriers. Connection to incompatible hardware is likely to cause serious damage.

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

Issue	Description	Date
1.0	First Issue	31. January 2002

Table of Contents

1 Introduction

The TIP610-SW-95 QNX-Neutrino device driver allows the operation of a TIP610 16/20 Channel digital I/O IP on QNX-Neutrino operating systems.

The TIP610 device driver is basically implemented as an user installable Resource Manager. The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

Supported features:

- Read and write actual port values
- Read and write actual port directions
- Read and write actual port polarities
- Read the port values when a port event occurs or wait for the event

This driver will need an initializing of the carrier board, (e.g. SBS-PCI40). This driver should also announce the physical base addresses and the interrupt vector of the IP-slots. An example using the SBS-PCI40 is attached to the driver. This initialization software must be run before the driver is started.

2 Installation

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

Following driver specific files are located on the diskette:

/driver/tip610.c	Driver source code
/driver/tip610.h	Driver interface definitions and data structures
/driver/tip610def.h	Device driver include
/driver/node.h	Queue management definitions
/driver/node.c	Queue management source code
/example/example.c	Example application
/pci40/*	SBS-PCI40 installation example
TIP610-SW-95.pdf	This manual in PDF format

For installation create a new directory (e.g. *../tip610*) in the */usr/src* directory and copy the complete */driver* and */example* directories (with sub-directories and all files) from the distribution diskette into the new created project directory.

Note

It's absolute important to create the tip610 project directory in the /usr/src directory otherwise the automatic build with make will fail.

2.1 Build the device driver

1. Change to the */usr/src/tip610/driver* directory
2. Execute the Makefile

```
# make install
```

After successful completion the driver binary will be installed in the */bin* directory.

2.2 Build the example application

1. Change to the */usr/src/tip610/example* directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (*t610exam*) will be installed in the */bin* directory.

2.3 Build the carrier board initialization example

1. Change to the */usr/src/pci40* directory
2. Execute the Makefile

```
# make install
```

After successful completion the example binary (*pci40*) will be installed in the */bin* directory.

2.4 Start the driver process

The carrier board initialization must be called before the driver is started. For example call the SBS-PCI40 initialization.

```
pci40
```

This initialization will printout the base addresses of I/O-, memory space and interrupt vector for each IP-slot.

To start the TIP610 device driver respective the TIP610 resource manager you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tip610 –A<IOaddress> -I<Interrupt> &
```

This will start the TIP610 resource manager with one module mounted at the specified <*IOaddress*> with the interrupt <*Interrupt*>. (The address and Interrupt depend on the system, this address is printed out by the SBS-PCI40 initialization example).
For starting the TIP610 resource manager with more than one module, you have simply to add the additional IO-addresses behind the *–A* flag and the additional interrupts behind the *-I*

```
tip610 –A<IOaddress_0>,<IOaddress_1>,…,<IOaddress_n>
      -I<Interrupt_0>,<Interrupt_1>,…,<Interrupt_n> &
```

The TIP610 Resource Manager registers created devices in the Neutrinos pathname space under following names.

```
/dev/tip610_0
/dev/tip610_1
…
/dev/tip610_x
```

This pathname must be used in the application program to open a path to the desired TIP610 device.

```
fd = open(“/dev/tip610_0”, O_RDWR);
```

For debugging you can start the TIP610 Resource Manager with the –v option. Now the Resource Manager will print versatile information about TIP610 configuration and command execution on the terminal window.

```
tip610 –v –A<IOaddress> &
```

3 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() - open a file descriptor

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int **open** (*const char *pathname, int flags*)

DESCRIPTION

The **open** function creates and returns a new file descriptor for the TIP610 named by *pathname.*
The flags argument controls how the file is to be opened. TIP610 devices must be opened *O_RDWR*.

EXAMPLE

```
int    fd;

fd = open("/dev/tip610_0", O_RDWR);
```

RETURNS

The normal return value from open is a non-negative integer file descriptor. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Returns only Neutrino specific error codes, see Neutrino Library Reference.

SEE ALSO

Library Reference - open()

3.2 close()

NAME

close() – close a file descriptor

SYNOPSIS

#include <unistd.h>

int **close** *(int filedes)*

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int   fd;

…

if (close(fd) != 0)
{
  /* handle close error conditions */
}
```

RETURNS

The normal return value from close is 0. In the case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

Returns only Neutrino specific error code, see Neutrino Library Reference.

SEE ALSO

Library Reference - close()

3.3 devctl()

NAME

devctl() – device control functions

SYNOPSIS

```
#include <sys/types.h>
#include <unistd.h>
#include <devctl.h>

int devctl( int filedes,
            int dcmd,
            void * data_ptr,
            size_t n_bytes,
            int * dev_info_ptr );
```

DESCRIPTION

The **devctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.
The argument *dcmd* specifies the control code for the operation.
The arguments *data_ptr* and *n_bytes* depends on the command and will be described for each command in detail later in this chapter. Usually *data_ptr* points to a buffer that passes data between the user task and the driver and *n_bytes* defines the size of this buffer.
The argument *dev_info_ptr* is unused for the TIP610 driver and should be set to NULL.

The following devctl command codes are defined in *TIP610.h* :

Value	**Meaning**
DCMD_T610_READ	Read actual port values
DCMD_T610_WRITE	Write new port values
DCMD_T610_POLREAD	Read actual polarity setting of the ports
DCMD_T610_POLWRITE	Change polarity setting of the ports
DCMD_T610_DIRREAD	Read actual direction setting of the ports
DCMD_T610_DIRWRITE	Change direction setting of the ports
DCMD_T610_EVREAD	Wait for event and read input ports

See behind for more detailed information on each control code.

Note

To use these TIP610 specific control codes the header file TIP610.h must be included in the application

RETURNS

On success, EOK is returned. In the case of an error, the appropriate error code is returned by the function (not in errno!).

ERRORS

ENOTTY	Inappropriate I/O control operation. This error code is returned if the requested devctl function is unknown. Please check the argument *dcmd*.

Other function dependant error codes will be described for each devoctl code separately. Note, the TIP610 driver always returns standard QNX error codes.

SEE ALSO

Library Reference - devctl()

3.3.1 DCMD_T610_READ

NAME

DCMD_T610_READ - Read actual value of the I/O ports

DESCRIPTION

This devctl function reads the actual value of the I/O ports. A pointer to the callers read buffer (*T610_READ_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T610_READ_BUF* structure has the following layout:

```
typedef struct
{
   /* OUT */
   unsigned char        portA;
   unsigned char        portB;
   unsigned char        portC;
} T610_READ_BUF, *PT610_READ_BUF;
```

portA
portB
portC
In these arguments the actual values of the ports A, B and C will be returned. The values will be also read, if a port is programmed as output.

EXAMPLE

```
int              fd;
int              result;
T610_READ_BUF    ReadBuf;

…

/* Read actual values */
result = devctl  (fd,
                  DCMD_T610_READ,
                  &ReadBuf,
                  sizeof(ReadBuf),
                  NULL);
if (result == EOK)
{
    /* Read of port values successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.3.2 DCMD_T610_WRITE

NAME

DCMD_T610_WRITE - Change output value of ports

DESCRIPTION

This devctl function writes new values to the ports. A pointer to the callers read buffer (*T610_WRITE_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device. Writing to input registers is allowed, but will have no effect until the direction of the port is changed.

The *T610_WRITE_BUF* structure has the following layout:

```
typedef struct
{
    /* IN */
    unsigned char       portA;
    unsigned char       portB;
    unsigned char       portC;
    unsigned char       enab;
} T610_WRITE_BUF ,*PT610_WRITE_BUF;
```

portA
portB
portC
In these arguments the new values of the ports A, B and C must be specified.

enab
This argument is used to specify ports that shall be changed. The ports write is enabled with a ORed value of the following flags:

TIP610_FL_ENA_PA	Enable writing to port A
TIP610_FL_ENA_PB	Enable writing to port B
TIP610_FL_ENA_PC	Enable writing to port C

EXAMPLE

```
int              fd;
int              result;
T610_WRITE_BUF   WriteBuf;

…

/* Set Port A to 0x23, do not touch portB, set portC to 0x01 */
WriteBuf.portA = 0x23;
WriteBuf.portC = 0x01;
WriteBuf.enab =  TIP610_FL_ENA_PA | TIP610_FL_ENA_PC;
result = devctl  (fd,
                  DCMD_T610_WRITE,
                  &WriteBuf,
                  sizeof(WriteBuf),
                  NULL);
if (result == EOK)
{
    /* Write of port values successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.3.3 DCMD_T610_POLREAD

NAME

DCMD_T610_POLREAD - Read actual value of the I/O port polarity setting

DESCRIPTION

This devctl function reads the actual value of the I/O ports polarity setting. A pointer to the callers read buffer (*T610_READ_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T610_READ_BUF* structure has the following layout:

```
typedef struct
{
   /* OUT */
   unsigned char        portA;
   unsigned char        portB;
   unsigned char        portC;
} T610_READ_BUF, *PT610_READ_BUF;
```

portA
portB
portC
In these arguments the actual values of the polarity settings of port A, B and C will be returned. If a bit is set the value of the respective I/O line is inverted.

EXAMPLE

```
int              fd;
int              result;
T610_READ_BUF    ReadBuf;

…

/* Read actual port polarities */
result = devctl  (fd,
                  DCMD_T610_POLREAD,
                  &ReadBuf,
                  sizeof(ReadBuf),
                  NULL);
if (result == EOK)
{
    /* Read of port polarities successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.3.4 DCMD_T610_POLWRITE

NAME

DCMD_T610_WRITE - Change polarity setting of the ports

DESCRIPTION

This devctl function changes the polarity settings of the I/O ports. A pointer to the callers read buffer (*T610_WRITE_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T610_WRITE_BUF* structure has the following layout:

```
typedef struct
{
    /* IN */
    unsigned char       portA;
    unsigned char       portB;
    unsigned char       portC;
    unsigned char       enab;
} T610_WRITE_BUF ,*PT610_WRITE_BUF;
```

portA
portB
portC
In these arguments the new values of the ports A, B and C polarity must be specified. A set bit will invert the respective I/O line.

enab
This argument is used to specify the ports polarities that shall be changed. The changing writes are enabled with a ORed value of the following flags:

TIP610_FL_ENA_PA	Enable polarity changing for port A
TIP610_FL_ENA_PB	Enable polarity changing for port B
TIP610_FL_ENA_PC	Enable polarity changing for port C

EXAMPLE

```
int              fd;
int              result;
T610_WRITE_BUF   WriteBuf;

…

/* Invert Port B, do not change port A and C */
WriteBuf.portB = 0xFF;
WriteBuf.enab =  TIP610_FL_ENA_PB;
result = devctl  (fd,
                  DCMD_T610_POLWRITE,
                  &WriteBuf,
                  sizeof(WriteBuf),
                  NULL);
if (result == EOK)
{
    /* Write of ports polarity successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.3.5 DCMD_T610_DIRREAD

NAME

DCMD_T610_DIRREAD - Read actual value of the I/O ports direction setting

DESCRIPTION

This devctl function reads the actual value of the I/O ports direction setting. A pointer to the callers read buffer (*T610_READ_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T610_READ_BUF* structure has the following layout:

```
typedef struct
{
   /* OUT */
   unsigned char        portA;
   unsigned char        portB;
   unsigned char        portC;
} T610_READ_BUF, *PT610_READ_BUF;
```

portA
portB
portC
In these arguments the actual values of the ports A, B and C will be returned. If a bit is set, the I/O line will be used as an input, otherwise, if the bit is reset, the I/O line will be used as an output.

EXAMPLE

```
int              fd;
int              result;
T610_READ_BUF    ReadBuf;

…

/* Read actual port directions */
result = devctl  (fd,
                  DCMD_T610_DIRREAD,
                  &ReadBuf,
                  sizeof(ReadBuf),
                  NULL);
if (result == EOK)
{
     /* Read of port direction successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.3.6 DCMD_T610_DIRWRITE

NAME

DCMD_T610_DIRWRITE - Change the port direction

DESCRIPTION

This devctl function reads the actual value of the I/O ports. A pointer to the callers read buffer (*T610_WRITE_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

The *T610_WRITE_BUF* structure has the following layout:

```
typedef struct
{
    /* IN */
    unsigned char       portA;
    unsigned char       portB;
    unsigned char       portC;
    unsigned char       enab;
} T610_WRITE_BUF ,*PT610_WRITE_BUF;
```

portA
portB
portC
In these arguments the new values of the ports A, B and C direction must be specified. A set bit will configure the respective I/O line as an input, a reset bit configures an output line. Remember that the ports direction must match to the hardware settings (See User Manual of the TIP610).

enab
In these argument is used to specify ports where the direction shall be changed. The ports direction change is enabled with a ORed value of the following flags:

TIP610_FL_ENA_PA	Enable direction changing for port A
TIP610_FL_ENA_PB	Enable direction changing for port B
TIP610_FL_ENA_PC	Enable direction changing for port C

EXAMPLE

```
int              fd;
int              result;
T610_WRITE_BUF   WriteBuf;

…

/* Set Port A as output, do not change port A and C */
WriteBuf.portA = 0x00;
WriteBuf.enab =  TIP610_FL_ENA_PA;
result = devctl  (fd,
                  DCMD_T610_DIRWRITE,
                  &WriteBuf,
                  sizeof(WriteBuf),
                  NULL);
if (result == EOK)
{
    /* Write of port directions successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.3.7 DCMD_T610_EVREAD

NAME

DCMD_T610_EVREAD - Wait for specified event and read port values

DESCRIPTION

This devctl function waits on a specified event and than reads the ports contents. A pointer to the callers event read buffer (*T610_EVREAD_BUF*) and the size of this structure is passed by the parameters *data_ptr* and *n_bytes* to the device.

A special characteristic of the CIO Z8536, which is used on the TIP610, is the functionality of the ports. In addition to the normal port read function, the driver can wait for a special event, before the ports are read. The event can be specified by special bit patterns of the ports A and B or by any transition of port bit of port A or B. The ports are used for the purpose as bit port in OR mode (see also the TECHNICAL MANUAL Z8536 CIO which is part of the TIP610 engineering kit).
The *T610_READ_BUF* structure has the following layout:

```
typedef struct
{
    /* IN */
    unsigned long       ev_mode;
    long                timeout;
    unsigned char       matchA;
    unsigned char       matchB;
    unsigned char       maskA;
    unsigned char       maskB;
    /* OUT */
    unsigned char       portA;
    unsigned char       portB;
    unsigned char       portC;
} T610_EVREAD_BUF, *PT610_EVREAD_BUF;
```

ev_mode
This parameter selects the kind of event to wait for. The following events are defined:

TIP610_EV_MATCH	In this mode the I/O request terminates if all bits, which are masked in the arguments ***maskA*** and ***maskB***, have the state defined in the parameters ***matchA*** and ***matchB***.
TIP610_EV_TRANS_HIGH	In this mode the I/O request terminates if a single port bit, which is specified in the call parameters ***maskA*** and ***maskB*** has a low to high transition. The parameters ***matchA*** and ***matchB*** are don't care. Note that only one bit shall be selected in ***maskA*** or ***maskB*** in this mode.
TIP610_EV_TRANS_LOW	In this mode the I/O request terminates if a single port bit, which is specified in the call parameters ***maskA*** and ***maskB*** has a high to low transition. The parameters ***matchA*** and ***matchB*** are don't care. Note that only one bit shall be selected in ***maskA*** or ***maskB*** in this mode.
TIP610_EV_TRANS_ANY	In this mode the I/O request terminates if a single port bit, which is specified in the call parameters ***maskA*** and ***maskB*** has any transition. The parameters ***matchA*** and ***matchB*** are don't care. Note that only one bit shall be selected in ***maskA*** or ***maskB*** in this mode.

The call parameter **timeout** selects the time in seconds, the I/O request have to wait before it times out.

portA
portB
portC
In these arguments the actual values of the ports A, B and C will be returned. The values will be also read, if a port is programmed as output.

EXAMPLE

```
int              fd;
int              result;
T610_EVREAD_BUF  EventBuf;

…

/* Wait for a match of port B to 0x12, timeout after 30sec. */
EventBuf.ev_mode =    TIP610_EV_MATCH;
EventBuf.timeout =    30;
EventBuf.matchA =     0x00;
EventBuf.matchB =     0x12;
EventBuf.maskA =      0x00;
EventBuf.maskB =      0xFF;
result = devctl  (fd,
                  DCMD_T610_EVREAD,
                  &EventBuf,
                  sizeof(EventBuf),
                  NULL);
if (result == EOK)
{
    /* Wait for match event successful */
}

…
/* Wait for a high transition of bit3 of port B, */
/* timeout after 30sec.                          */
EventBuf.ev_mode =    TIP610_EV_TRANS_HIGH;
EventBuf.timeout =    30;
EventBuf.maskA =      0;
EventBuf.maskB =      (1 << 3);
result = devctl  (fd,
                  DCMD_T610_EVREAD,
                  &EventBuf,
                  sizeof(EventBuf),
                  NULL);
if (result == EOK)
{
    /* Wait for high transition event successful */
}

…
```

ERRORS

EINVAL	Invalid argument. This error code is returned if the size of the message buffer is too small.

SEE ALSO

Library Reference - devctl()

3.4 Step by Step Driver Initialization

The following code example illustrates how the module must be setup with default hardware setup, to make Port A to an output port and Port B to an input port.

```
/*
**  ( 1.) Set I/O Driver direction
*/
WriteBuf.PortC = 0x01;
WriteBuf.enab = TIP610_FL_ENA_PC;
result = devctl( fd,
                 DCMD_T610_WRITE,
                 &WriteBuf,
                 sizeof(WriteBuf),
                 NULL);


/*
**  ( 2.) Setup port direction
*/
WriteBuf.PortA = 0x00;
WriteBuf.PortB = 0xFF;
WriteBuf.enab = TIP610_FL_ENA_PA | TIP610_FL_ENA_PB;
result = devctl( fd,
                 DCMD_T610_DIRWRITE,
                 &WriteBuf,
                 sizeof(WriteBuf),
                 NULL);
```