The Embedded I/O Company

TEWS TECHNOLOGIES

TIP675-SW-42

VxWorks Device Driver

48 TTL I/O Lines with Interrupts

Version 2.0.x

User Manual

Issue 2.0.0

July 2010

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

TIP675-SW-42

VxWorks Device Driver

48 TTL I/O Lines with Interrupts

Supported Modules:
TIP675

Issue	Description	Date
1.0	First Issue	September 12, 2002
1.1.0	IPAC Carrier Driver Support	January 11, 2006
1.2.0	New Address TEWS TECHNOLOGIES LLC ChangeLog.txt added to file list, new error code for tip675DevCreate()	December 5, 2006
1.2.1	Carrier Driver description added	June 24, 2008
2.0.0	SMP Support, IPAC carrier interface functions removed, return value for ioctl function changed	July 26, 2010

Table of Contents

1 INTRODUCTION 4
1.1 Device Driver 4
1.2 IPAC Carrier Driver 5
2 INSTALLATION 6
2.1 Include device driver in VxWorks projects 6
2.2 System resource requirement 6
3 I/O SYSTEM FUNCTIONS 7
3.1 tip675Drv() 7
3.2 tip675DevCreate() 9
4 I/O FUNCTIONS 11
4.1 open() 11
4.2 close() 13
4.3 ioctl() 15
4.3.1 FIO_T675_READ 17
4.3.2 FIO_T675_WRITE 18
4.3.3 FIO_T675_SET_DIR 19
4.3.4 FIO_T675_INSTALL_ISF 20
4.3.5 FIO_T675_REMOVE_ISF 23
4.3.6 FIO_T675_ENA_EXCLK 25
4.3.7 FIO_T675_DIS_EXCLK 26

1 Introduction

1.1 Device Driver

The TIP675-SW-42 VxWorks device driver software allows the operation of the TIP675 IP conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with open(), close() and ioctl() functions.

The TIP675 driver includes the following functions:

- reading the input register
- writing the output register
- programming direction of every I/O line
- configure simultaneous update feature
- connect interrupt callback functions to every I/O line
- Support for legacy and VxBus IPAC carrier driver
- SMP Support

The TIP675-SW-42 supports the modules listed below:

TIP675-10	48 TTL I/O Lines with Interrupts	IndustryPack® compatible

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

TIP675 User manual
TIP675 Engineering Manual
CARRIER-SW-42 IPAC Carrier 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 might cause problems when accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which shall 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-42 is part of this TIP675-SW-42 distribution. It is located in the directory '/CARRIER-SW-42/' 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-42 User Manual for a detailed description on how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

The following files and directories are located on the distribution media:

Directory path './TIP675-SW-42/':

tip675drv.c	TIP675 device driver source
tip675def.h	TIP675 driver include file
tip675.h	TIP675 include file for driver and application
tip675exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP675-SW-42-2.0.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include device driver in VxWorks projects

For including the TIP675-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path. (For example: TIP675)

(2) Add the device drivers C-files to your project.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 System resource requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	< 1 KB
Semaphores	---	---

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

3 I/O system functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tip675Drv()

NAME

tip675Drv() - installs the TIP675 driver in the I/O system

SYNOPSIS

#include “tip675.h”

STATUS tip675Drv(void)

DESCRIPTION

This function initializes the TIP675 driver and installs it in the I/O system.

The call of this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tip675.h”
…
status = tip675Drv();

if (status == ERROR)
{
     /* Error handling */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip675DevCreate()

NAME

tip675DevCreate() – Add a TIP675 device to the VxWorks system

SYNOPSIS

#include “tip675.h”

```
STATUS tip675DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This routine adds a TIP675 device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the desired device instance beginning by 0. This parameter is 0 for the first TIP675 in the system, 1 for the second TIP675 and so forth. The order of TIP675 modules depends on the search order of the IPAC carrier driver.

funcType

This parameter is unused and should be set to 0.

pParam

This parameter is unused and should be set to NULL.

EXAMPLE

```
#include "tip675.h”
…
STATUS    result;


/*-------------------------------------------------------
  Create the device "/tip675/0" for the first TIP675 module
  -------------------------------------------------------*/

result = tip675DevCreate("/tip675/0", 0, 0, NULL);

if (result == ERROR)
{
     /* Error occurred when creating the device */
}…
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes can be read with the function *errnoGet()*.

Error code	Description
S_ioLib_NO_DRIVER	Driver not installed, call tip675Drv() first.
S_tip675Drv_IARG	The parameter devIdx is out of range
S_tip675Drv_EXISTS	The TIP675 device specified by the parameter devIdx has been created already.
ENXIO	No TIP675 module found for the specified parameter devIdx. If this error occurred for parameter devIdx=0, no TIP675 module at all was recognized.

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4 I/O Functions

4.1 open()

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char   *name,
        int          flags,
        int          mode
)
```

DESCRIPTION

Before I/O can be performed to the TIP675 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened, the name specified in tip675DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int  fd;

/*----------------------------------------
  Open the device named "/tip675/0" for I/O
  ----------------------------------------*/

fd = open("/tip675/0", 0, 0);

if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

A device descriptor number or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
int close
(
      int     fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int       fd;
STATUS    result;

/*----------------
  close the device
  ----------------*/

result = close(fd);

if (result == ERROR)
{
     /* Handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 ioctl()

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

#include “tip675.h”

```
int ioctl
(
        int     fd,
        int     request,
        int     arg
)
```

DESCRIPTION

Special I/O operations that do not fit to the standard basic I/O calls will be performed by calling the *ioctl* function with a specific function code and an optional function dependent argument.

For details of supported *ioctl* functions see VxWorks Reference Manual: VxWorks Programmer’s Guide: I/O system.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed. Following functions are defined:

Function	Description
FIO_T675_READ	Read from input buffer
FIO_T675_WRITE	Write to output buffer
FIO_T675_SET_DIR	Set direction of I/O lines
FIO_T675_INSTALL_ISF	Install user supplied I/O service function at specified I/O line
FIO_T675_REMOVE_ISF	Remove user supplied I/O service function from specified I/O line
FIO_T675_ENA_EXCLK	Enable simultaneous update feature
FIO_T675_DIS_EXCLK	Disable simultaneous update feature

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error code can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual). Function specific error codes will be described with the function.

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.3.1 FIO_T675_READ

This function reads the current contents of the input registers of the TIP675 associated with the device descriptor *fd* into the read buffer of type T675_READ_STR.

The function specific control parameter **arg** is a pointer to a T675_READ_STR structure.

```
typedef struct
{
        unsigned short          in_1_16;
        unsigned short          in_17_32;
        unsigned short          in_33_48;
} T675_READ_STR;
```

in_1_16

Returns the contents of the input register for I/O-line 1 up to 16. (Line 1 is assigned to bit 0, line 2 to bit 1 and so on.)

in_17_32

Returns the contents of the input register for I/O-line 17 up to 32. (Line 17 is assigned to bit 0, line 18 to bit 1 and so on.)

in_33_48

Returns the contents of the input register for I/O-line 33 up to 48. (Line 33 is assigned to bit 0, line 34 to bit 1 and so on.)

EXAMPLE

```
#include      "tip675.h"
…
int               fd;
int               retval;
T675_READ_STR     rBuf;


/*-------------------------------------------------------
  read the input registers of the device associated with fd
  -------------------------------------------------------*/
retval = ioctl (fd, FIO_T675_READ, (int)&rBuf);

if (retval == OK)
{
     printf( “Input  1-16: %04X\n”, rBuf.in_1_16 );
     printf( “Input 17-32: %04X\n”, rBuf.in_17_32 );
     printf( “Input 33-48: %04X\n”, rBuf.in_33_48 );
}
```

4.3.2 FIO_T675_WRITE

This function writes to the output registers of the TIP675 associated with the device descriptor *fd* from a write buffer of type T675_WRITE_STR.

The function specific control parameter **arg** is a pointer to a T675_WRITE_STR structure.

```
typedef struct
{
        unsigned short          out_1_16;
        unsigned short          out_17_32;
        unsigned short          out_33_48;
} T675_WRITE_STR;
```

out_1_16

Specifies the output value that will be written into the output register for I/O-lines 1 up to 16 (Line 1 is assigned to bit 0, line 2 to bit 1 and so on).

out_17_32

Specifies the output value that will be written into the output register for I/O-lines 17 up to 32 (Line 17 is assigned to bit 0, line 18 to bit 1 and so on).

out_33_48

Specifies the output value that will be written into the output register for I/O-lines 33 up to 48 (Line 33 is assigned to bit 0, line 34 to bit 1 and so on).

EXAMPLE

```
#include      "tip675.h"
…
int               fd;
int               retval
T675_WRITE_STR    wBuf;

/*-------------------------------------------------------
  write to the output registers
  -------------------------------------------------------*/
wBuf.out_1_16 = 0x9abc;
wBuf.out_17_32 = 0x5678;
wBuf.out_33_48 = 0x1234;

retval = ioctl(fd, FIO_T675_WRITE, (int)&wBuf);

if (retval == ERROR)
{
     /* handle function specific error conditions */
}
```

4.3.3 FIO_T675_SET_DIR

With this ioctl function, each of the 48 I/O lines may be individually set as input or output. After driver startup, all I/O lines are set to be inputs. To enable a certain output line, set the corresponding bit in the mask to 1.

The function specific control parameter **arg** is a pointer to a T675_DIRECTION_STR structure.

```
typedef struct
{
        unsigned short          dir_1_16;
        unsigned short          dir_17_32;
        unsigned short          dir_33_48;
} T675_DIRECTION_STR;
```

dir_1_16

Specifies the line direction for I/O-lines 1 up to 16 (Line 1 is assigned to bit 0, line 2 to bit 1 and so on).

dir_17_32

Specifies the line direction for I/O-lines 17 up to 32 (Line 17 is assigned to bit 0, line 18 to bit 1 and so on).

dir_33_48

Specifies the line direction for I/O-lines 33 up to 48 (Line 33 is assigned to bit 0, line 34 to bit 1 and so on).

EXAMPLE

```
#include      "tip675.h"
…
int                     fd;
int                     retval;
T675_DIRECTION_STR      dBuf;

/*---------------------------------
  Set direction: Line 1-24  – output
                 Line 25-32 – input
  ---------------------------------*/
dBuf.out_1_16 = 0xFFFF;
dBuf.out_17_32 = 0x00FF;
dBuf.out_33_48 = 0x0000;

retval = ioctl(fd, FIO_T675_SET_DIR, (int)&dBuf);
if (retval == ERROR)
{
     /* handle function specific error conditions */
}
```

4.3.4 FIO_T675_INSTALL_ISF

This ioctl function connects a user supplied function to an interrupt at the specified I/O line. Each time an interrupt occurred with the specified transition at the specified I/O line, this function will be called. Only one callback function can be installed per I/O line and transition at the same time.

The callback function will be invoked in supervisor mode at interrupt level. Be sure that only suitable system functions will be called and no wait conditions do occur.

The function specific control parameter **arg** is a pointer to a T675_INSTALL_ISF_STR structure

```
typedef struct
{
        unsigned short          io_line;
        unsigned short          edge;
        unsigned long           arg;
        VOIDFUNCPTR             funcPtr;
} T675_INSTALL_ISF_STR;
```

io_line

Specifies the I/O-line at which the callback function will be installed. Valid I/O lines are in range from 1 to 48.

edge

Specifies the transition (positive, negative) at which the callback function will be installed. Valid values are:

Value	Description
T675_POS_EDGE	An interrupt will be generated on a Low to High transition
T675_NEG_EDGE	An interrupt will be generated on a High to Low transition

arg

This user defined argument will be passed in the 4th argument to the user supplied callback function. This argument could be a semaphore ID, a message ID or something like that.

funcPtr

This parameter contains a pointer to a function to be called if the interrupt at the connected I/O line occurred. The routine is invoked in supervisor mode at interrupt level. A proper C environment is established, the necessary registers saved, and the stack is set up.

The function prototype of the callback function has the following arguments described below.

```
void callback
(
        unsigned long           base,
        unsigned long           line_info,
        T675_READ_STR           *inBuf,
        unsigned long           arg
);
```

base

Base address of the modules I/O space. This argument is available for compatibility purposes only and it is not recommended to be used.

line_info

The lower 16-bit contains the number of the I/O-line (1...48) at which the event occurred. The upper 16-bit contains a flag which specifies the transition on which this interrupt was generated. The corresponding macros are the same as used for the parameter edge in the T675_INSTALL_ISF_STR structure. Please note these flags are left shifted by 16 bits.

inBuf

This argument is a pointer to a read buffer of type T675_READ_STR. The buffer is filled with the contents of the line input registers directly on entrance of the drivers interrupt service routine. The read buffer is placed on the interrupt stack and only available in the interrupt context.
Due to the fact that the input registers aren't latched, the input buffer doesn't reflect the state of the input lines at the moment of the transition. The read is delayed by the interrupts latency which depends on the system performance and other functions which may block the interrupt processing.

arg

User supplied argument.

EXAMPLE

```
#include      "tip675.h"
…
/*-----------------------------
  CALLBACK FUNCTION (INTERRUPT)
  -----------------------------*/
static void callback
(
     unsigned long base,
     unsigned long line_info,
     T675_READ_STR *pvalue,
     unsigned long arg
)
{
     int line;

     line = line_info & 0xffff;   /* mask lower 16 bit */

     if (((line_info>>16) & T675_POS_EDGE) != 0)
     {
          /* yes, this was a rising edge */
     }
     /* do whatever necessary to handle this interrupt */
}
```

```
/*----------------
  USER APPLICATION
  ----------------*/

…

int                     fd;
int                     retval;
T675_INSTALL_ISF_STR    IsfPar;

/*-------------------------------------------------------
  Install a callback function for line 12 interrupts. The
  function will be called if a rising edge occurred. The
  user argument is not used (0).
  -------------------------------------------------------*/
IsfPar.io_line = 12;
IsfPar.edge = T675_POS_EDGE;
IsfPar.funcPtr = callback;
IsfPar.arg = 0;

retval = ioctl(fd, FIO_T675_INSTALL_ISF, (int)&IsfPar);

if (retval == ERROR)
{
     /* handle function specific error conditions */
}
```

ERRORS CODES

S_t675Drv_ILINE	Specified I/O line out of range.
S_t675Drv_IPARAM	No valid transition parameter
S_t675Drv_BUSY	This I/O line and edge combination is already occupied. Call the ioctl function *FIO_T675_REMOVE_ISF* to remove the previous installed callback function.

4.3.5 FIO_T675_REMOVE_ISF

This ioctl function removes a previously installed callback function from a specified I/O line and transition edge.

The function specific control parameter **arg** is a pointer to a T675_INSTALL_ISF_STR structure.

```
typedef struct
{
        unsigned short          io_line;
        unsigned short          edge;
} T675_REMOVE_ISF_STR;
```

io_line

Specifies the I/O-line number from which the callback function shall be removed.
Valid I/O lines are in range from 1 to 48.

edge

Specifies the transition edge from which the callback function shall be removed.
Valid values are:

Value	Description
T675_POS_EDGE	Rising edge (low to high transition)
T675_NEG_EDGE	Falling edge (high to low transition)

EXAMPLE

```
#include "tip675.h"
…
int                     fd;
int                     retval;
T675_REMOVE_ISF_STR     RsfPar;

/*-------------------------------------------------------
  Remove user supplied callback function from I/O line 24
  -------------------------------------------------------*/
RsfPar.io_line = 24;
RsfPar.edge = T675_POS_EDGE;

retval = ioctl(fd, FIO_T675_REMOVE_ISF, (int)&RsfPar);
if (retval == ERROR)
{
     /* handle function specific error conditions */
}
```

ERROR CODES

S_t675Drv_ILINE	Specified I/O line out of range.
S_t675Drv_IPARAM	No valid transition parameter

4.3.6 FIO_T675_ENA_EXCLK

This ioctl function enables the simultaneous update feature of the TIP675. Setting the ioctl argument **arg** to *T675_POS_EDGE* will cause the inputs and outputs to be latched on the rising edge of the external clock, while setting **arg** to *T675_NEG_EDGE* will latch the inputs and outputs on the falling edge.

EXAMPLE

```
#include "tip675.h"

…

int      fd;
int      retval;

/*-----------------------------------------
  Enable simultaneous update on falling edge
  -----------------------------------------*/

retval = ioctl(fd, FIO_T675_ENA_EXCLK, T675_NEG_EDGE);

if (retval == ERROR) {
     /* handle function specific error conditions */
}
```

4.3.7 FIO_T675_DIS_EXCLK

This ioctl function disables the simultaneous update feature of the TIP675. After driver startup the simultaneous update feature is disabled.

The optional argument **arg** shall be set to 0.

EXAMPLE

```
#include "tip675.h"

…

int       fd;
int       retval;

/*-------------------------------------------------
  Disable simultaneous update of inputs and outputs
  -------------------------------------------------*/

retval = ioctl(fd, FIO_T675_DIS_EXCLK, 0);

if (retval == ERROR)
{
     /* handle function specific error conditions */
}
```