The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC851-SW-42

VxWorks Device Driver

Multifunction I/O (16 bit ADC/DAC, TTL I/O, Counter)

Version 2.0.x

User Manual

Issue 2.0.1

March 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

TPMC851-SW-42

VxWorks Device Driver

Multifunction I/O
(16 bit ADC/DAC, TTL I/O, Counter)

Supported Modules:
TPMC851-10

Issue	Description	Date
1.0.0	First Issue	January 12, 2005
1.0.1	Smaller Corrections	January 24, 2005
1.1.0	General review, tp851DrvRemove removed, tp851PciInit() added, file list changed, compilation hints for VxWorks 5.4 added	May 19, 2006
1.1.1	New Address TEWS LLC	October 4, 2006
1.2.0	File list changed	August 20, 2007
2.0.0	Support for VxBus and API description added, general revision	January 22, 2010
2.0.1	Legacy vs. VxBus Driver modified	March 26, 2010

Table of Contents

1 Introduction

The TPMC851-SW-42 release contains independent driver sources for the old legacy (pre-VxBus) and the new VxBus-enabled driver model. The VxBus-enabled driver is recommended for new developments with later VxWorks 6.x releases and mandatory for VxWorks SMP systems.

Both drivers, legacy and VxBus, share the same application programming interface (API) and device-independent basic I/O interface with open(), close() and ioctl() functions. The basic I/O interface is only for backward compatibility with existing applications and should not be used for new developments.

Both drivers invoke a mutual exclusion and binary semaphore mechanism to prevent simultaneous requests by multiple tasks from interfering with each other.

The TPMC851-SW-42 device driver supports the following features:

- Reading an ADC input value from a specified channel
- Configuring and using the ADC input sequencer
- Setting a DAC output value for a specified channel
- Configuring and using the DAC output sequencer
- Reading from digital I/O input register
- Writing to digital I/O output register
- Waiting for I/O input event (high, low or any transition on input line)
- Configuring I/O line direction
- Reading counter value
- Reset counter value
- Setting counter preload and match value
- Configuring counter mode
- Wait for counter match and control event

The TPMC851-SW-42 supports the modules listed below:

TPMC851	16 bit ADC/DAC, TTL I/O, Counter	(PMC)

To get more information about the features and use of TPMC851 devices it is recommended to read the manuals listed below.

TPMC851 User manual

TPMC851 Engineering Manual

2 Installation

Following files are located on the distribution media:

Directory path 'TPMC851-SW-42':

TPMC851-SW-42-2.0.1.pdf	PDF copy of this manual
TPMC851-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC851-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC851-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc851':

tpmc851drv.c	TPMC851 device driver source
tpmc851def.h	TPMC851 driver include file
tpmc851.h	TPMC851 include file for driver and application
tpmc851api.c	TPMC851 API file
Makefile	Driver Makefile
40tpmc851.cdf	Component description file for VxWorks development tools
tpmc851.dc	Configuration stub file for direct BSP builds
tpmc851.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc851exa.c	Example application

The archive TPMC851-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc851':

tpmc851drv.c	TPMC851 device driver source
tpmc851def.h	TPMC851 driver include file
tpmc851.h	TPMC851 include file for driver and application
tpmc851api.c	TPMC851 API file
tpmc851pci.c	TPMC851 PCI MMU mapping for Intel x86 based targets
tpmc851exa.c	Example application
tpmc851init.c	Legacy driver initialization
include/tdhal.h	Hardware dependent interface functions and definitions

2.1 Legacy vs. VxBus Driver

In later VxWorks 6.x releases, the old VxWorks 5.x legacy device driver model was replaced by VxBus-enabled device drivers. Legacy device drivers are tightly coupled with the BSP and the board hardware. The VxBus infrastructure hides all BSP and hardware differences under a well defined interface, which improves the portability and reduces the configuration effort. A further advantage is the improved performance of API calls by using the method interface and bypassing the VxWorks basic I/O interface.

VxBus-enabled device drivers are the preferred driver interface for new developments.

The checklist below will help you to make a decision which driver model is suitable and possible for your application:

Legacy Driver	VxBus Driver
▪ VxWorks 5.x releases ▪ VxWorks 6.5 and earlier releases ▪ VxWorks 6.x releases without VxBus PCI bus support	▪ VxWorks 6.6 and later releases with VxBus PCI bus ▪ SMP systems (only the VxBus driver is SMP safe!)

2.2 VxBus Driver Installation

Because Wind River doesn't provide a standard installation method for 3rd party VxBus device drivers the installation procedure needs to be done manually.

In order to perform a manual installation extract all files from the archive TPMC851-SW-42-VXBUS.zip to the typical 3rd party directory *installDir/vxworks-6.x/target/3rdparty* (whereas *installDir* must be substituted by the VxWorks installation directory).

After successful installation the TPMC851 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc851.*

At this point the TPMC851 driver is not configurable and cannot be included with the kernel configuration tool in a Wind River Workbench project. To make the driver configurable the driver library for the desired processer (CPU) and build tool (TOOL) must be built in the following way:

(1) Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)

(2) Change into the driver installation directory
installDir/vxworks-6.x/target/3rdparty/tews/tpmc851

(3) Invoke the build command for the required processor and build tool
make CPU=cpuName TOOL=tool

For Windows hosts this may look like this:

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc851
C:> make CPU=PENTIUM4 TOOL=diab
```

To compile SMP-enabled libraries, the argument VXBUILD=SMP must be added to the command line

```
C:> make CPU=PENTIUM4 TOOL=diab VXBUILD=SMP
```

To integrate the TPMC851 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc851.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc851
C:> copy 40tpmc851.cdf \Windriver\vxworks-6.7\target\config\comps\vxWorks
```

In VxWorks 6.7 and newer releases the kernel configuration tool scans the CDF file automatically and updates the *CxrCat.txt* cache file to provide component parameter information for the kernel configuration tool as long as the timestamp of the copied CDF file is newer than the one of the *CxrCat.txt*. If your copy command preserves the timestamp, force to update the timestamp by a utility, such as *touch*.

In earlier VxWorks releases the CxrCat.txt file may not be updated automatically. In this case, remove or rename the original *CxrCat.txt* file and invoke the make command to force recreation of this file.

```
C:> cd \Windriver\vxworks-6.7\target\config\comps\vxWorks
C:> del CxrCat.txt
C:> make
```

After successful completion of all steps above and restart of the Wind River Workbench, the TPMC851 driver can be included in VxWorks projects by selecting the *"TEWS TPMC851 Driver"* component in the *"hardware (default) - Device Drivers"* folder with the kernel configuration tool.

2.2.1 Direct BSP Builds

In development scenarios with the direct BSP build method without using the Workbench or the vxprj command-line utility, the TPMC851 configuration stub files must be copied to the directory *installDir/vxworks-6.x/target/config/comps/src/hwif.* Afterwards the *vxbUsrCmdLine.c* file must be updated by invoking the appropriate make command.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc851
C:> copy tpmc851.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc851.dr \Windriver\vxworks-6.7\target\config\comps\src\hwif

C:> cd \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> make vxbUsrCmdLine.c
```

2.3 Legacy Driver Installation

2.3.1 Include device driver in Tornado IDE project

For Including the TPMC851-SW-42 device driver into a Tornado IDE project follow the steps below:

(1) Extract all files from the archive TPMC851-SW-42-LEGACY.zip to your project directory.

(2) Add the device drivers C-files to your project.
Make a right click to your project in the 'Workspace' window and use the 'Add Files ...' topic.
A file select box appears, and the driver files in the tpmc851 directory can be selected.

(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 Tornado User's Guide.

2.3.2 Special installation for Intel x86 based targets

The TPMC851 device driver is fully adapted for Intel x86 based targets. This is done by conditional compilation directives inside the source code and controlled by the VxWorks global defined macro **CPU_FAMILY**. If the content of this macro is equal to *I80X86* special Intel x86 conforming code and function calls will be included.

The second problem for Intel x86 based platforms can't be solved by conditional compilation directives. Due to the fact that some Intel x86 BSP's doesn't map PCI memory spaces of devices which are not used by the BSP, the required device memory spaces can't be accessed.

To solve this problem a MMU mapping entry has to be added for the required TPMC851 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc851pci.c** contains the function *tpmc851PciInit().* This routine finds out all TPMC851 devices and adds MMU mapping entries for all used PCI memory spaces. Please insert a call to this function after the PCI initialization is done and prior to MMU initialization (*usrMmuInit()*).

The right place to call the function *tpmc851PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window).

Be sure that the function is called prior to MMU initialization otherwise the TPMC851 PCI spaces remains unmapped and an access fault occurs during driver initialization.

Please insert the following call at a suitable place in **sysLib.c**:

```
tpmc851PciInit();
```

Modifying the sysLib.c file will change the sysLib.c in the BSP path. Remember this for future projects and recompilations.

2.4 Changing driver parameters

The maximum number of wait jobs is limited for the driver not for a device. The default value is 10. If necessary, this number can be changed. Therefore change the definition of *TP851_IO_NUMEVENTOBJ* in tpmc851def.h. Each of the wait job objects allocates memory and a semaphore, please remember this when increasing the maximum number.

2.5 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	---
Semaphores	<maxNumberOfEvents>	1

<maxNumberOfEvents> = value of definition *TP851_IO_NUMEVENTOBJ*

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 API Documentation

3.1 General Functions

3.1.1 tpmc851Open()

Name

tpmc851Open() – opens a device.

Synopsis

```
TPMC851_DEV tpmc851Open
(
      char          *DeviceName
);
```

Description

Before I/O can be performed to a device, a file descriptor must be opened by a call to this function.

Parameters

DeviceName

This parameter points to a null-terminated string that specifies the name of the device. The first TPMC851 device is named “/tpmc851/0”, the second device is named “/tpmc851/1” and so on.

Example

```
#include “tpmc851.h”
TPMC851_DEV     pDev;

/*
** open file descriptor to device
*/
pDev = tpmc851Open( “/tpmc851/0” );
if (pDev == NULL)
{
     /* handle open error */
}
```

RETURNS

A device descriptor pointer, or NULL if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno.*

The error code is a standard error code set by the I/O system.

3.1.2 tpmc851Close()

Name

tpmc851Close() – closes a device.

Synopsis

```
int tpmc851Close
(
      TPMC851_DEV        pDev
);
```

Description

This function closes previously opened devices.

Parameters

pDev

This value specifies the file descriptor pointer to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc851.h”
TPMC851_DEV pDev;
int result;

/*
** close file descriptor to device
*/
result = tpmc851Close( pDev );
if (result < 0)
{
     /* handle close error */
}
```

RETURNS

Zero, or -1 if the function fails. An error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno*.

The error code is a standard error code set by the I/O system.

3.2 Device Access Functions

3.2.1 tpmc851AdcRead()

Name

tpmc851AdcRead() – Read value from ADC channel

Synopsis

```
STATUS tpmc851AdcRead
(
        TPMC851_DEV     pDev,
        int             channel,
        int             gain,
        unsigned long   flags,
        short           *pAdcValue
);
```

Description

This function starts an ADC conversion with specified parameters, waits for completion and returns the value.

The ADC sequencer must be stopped for single ADC conversions.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

channel

Specifies the ADC channel number. Valid values are 1..16 for differential input and 1..32 for single-ended input.

gain

Specifies the input gain. Valid gain values are 1, 2, 4, and 8.

flags

This is an ORed value of the following flags:

flag	description
TP851_F_CORR	If set the function will return a corrected value of the input data in *adcValue*. Factory set and module dependent correction data is used for correction. If not set, the raw value read from the module will be returned in *adcValue*.
TP851_F_IMMREAD	If set the driver will start the conversion without waiting for settling time. This should only be used if the previous conversion has used the same interface parameters (channel, gain, differential/single-ended). If not set the driver will use the automatic mode, which sets interface configuration, waits settling time and then starts the conversion.
TP851_F_DIFF	If set the input channel will be a differential input. If not set the input channel will be a single-ended input.

pAdcValue

This parameter specifies a pointer to a *short* value which receives the current ADC value.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;
short              AdcValue;

/*---------------------------------------------------------------------
  Read a corrected value from differential channel 2 using gain of 4
  ---------------------------------------------------------------------*/
result = tpmc851AdcRead(
              pDev,
              2,                                /* Channel    */
              4,                                /* Gain       */
              TP851_F_CORR | TP851_F_DIFF,      /* Flags      */
              &AdcValue );                      /* ADC value  */
if (result != ERROR)
{
     /* function succeeded */
     printf("    ADC-value: %d", AdcValue);
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC is busy, the ADC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid ADC channel number specified
S_tp851Drv_IGAIN	invalid gain specified
S_tp851Drv_TIMEOUT	ADC conversion timed out

3.2.2 tpmc851AdcSeqConfig()

Name

tpmc851AdcSeqConfig() – Configure ADC sequencer channel

Synopsis

```
STATUS tpmc851AdcSeqConfig
(
        TPMC851_DEV     pDev,
        int             channel,
        int             enable,
        int             gain,
        unsigned long   flags
);
```

Description

This function enables and configures, or disables an ADC channel for sequencer use.

The ADC sequencer must be stopped to execute this function.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

channel

Specifies the ADC channel number to configure. Valid values are 1..16 for differential input and 1..32 for single-ended input.

enable

Specifies if the channel shall be used in sequencer mode or not. (0 disables the channel, any other value will enable the channel)

gain

Specifies the input gain. Valid gain values are 1, 2, 4, and 8.

flags

Is an ORed value of the following flags:

flag	description
TP851_F_CORR	If set the sequencer will return a corrected value for the specified channel. Factory set and module dependent correction data is used for correction. If not set, the raw value read from the module will be returned.
TP851_F_DIFF	If set the input channel will be a differential input. If not set the input channel will be a single-ended input.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Configure single-ended channel 3, using a gain of 4 and
  returning corrected data when the sequencer is running
  -------------------------------------------------------*/
result = tpmc851AdcSeqConfig(
              pDev,
              3,                                /* Channel    */
              1,                                /* Enable     */
              4,                                /* Gain       */
              TP851_F_CORR );                   /* Flags      */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC is busy, the ADC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid ADC channel number specified
S_tp851Drv_IGAIN	invalid gain specified

3.2.3 tpmc851AdcSeqStart()

Name

tpmc851AdcSeqStart() – Start ADC Sequencer

Synopsis

```
STATUS tpmc851AdcSeqStart
(
    TPMC851_DEV                 pDev,
    unsigned short              cycTime,
    unsigned long               flags,
    TP851_ADC_SEQDATA_BUF       *pAdcSeqBuf
);
```

Description

This function configures the ADC sequencer time and starts the ADC sequencer.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

cycTime

Specifies the ADC sequencer cycle time. The sequencer time is specified in 100µs steps. With a value of 0, the “Sequencer Continuous Mode” is selected.

flags

One of the following optional flags is possible:

flag	description
TP851_F_EXTTRIGSRC	If set the ADC sequencer is triggered with digital I/O line 0. If not set, the ADC sequencer uses the ADC cycle counter.
TP851_F_EXTTRIGOUT	If set the ADC trigger is used as output on digital I/O line 0.

adcSeqBuf

Points to an ADC sequencer buffer. The buffer is used to store the ADC data. The ADC sequencer buffer is defined as a structure named *TP851_ADC_SEQDATA_BUF*. This structure is defined in *tpmc851.h*. The size of the buffer is variable, therefore a macro *TP851_CALC_SIZE_ADC_SEQDATA_BUF(s)* is defined in *tpmc851.h*, that calculates the size to allocate for the buffer. The macro-parameter *s* specifies the size of the *buffer* array of the structure.

```
typedef struct
{
    long        putIdx;
    long        getIdx;
    long        bufSize;
    long        seqState;
    short       buffer[1];
} TP851_ADC_SEQDATA_BUF, *PTP851_ADC_SEQDATA_BUF;
```

putIdx

Specifies the index into *buffer* where the next data will be stored to. This index is handled by the driver and should only be read by the application to check if data is available. The driver initializes this index to 0 when sequencer starts.

getIdx

Specifies the index into *buffer* where the next input data can be read from. This index must be handled by the application and is only be read by the driver to check a FIFO overflow. The driver initializes this index to 0 when sequencer starts.

bufSize

Specifies the array size of *buffer*. This value must be the same as used for *s* in *TP851_CALC_SIZE_ADC_SEQDATA_BUF(s)* when calculating the allocation size for *adcSeqBuf*.

seqState

Returns the sequencer state. This is an ORed value of the following status flags:

flag	**description**
TP851_SF_SEQACTIVE	If set the ADC sequencer is started. If not set, the ADC sequencer stopped.
TP851_SF_SEQOVERFLOWERR	If set the ADC sequencer has detected an overflow error. (Hardware detected)
TP851_SF_SEQTIMERERROR	If set the ADC sequencer has detected a timer error. (Hardware detected)
TP851_SF_SEQIRAMERROR	If set the ADC sequencer has detected an instruction RAM error. (Hardware detected)
TP851_SF_SEQFIFOOVERFLOW	If set the application supplied FIFO (*buffer*) has overrun. Data got lost.

buffer

Array used for ADC sequencer data FIFO.
The ADC data is stored by the sequencer into this FIFO. The assignment from data to channel is done as follows. The first data will be from the lowest enabled channel, the second from the next enabled channel and so on. There will be no data stored for disabled channels. If the end of *buffer* is reached the next data will be stored again at the beginning of the buffer.

Example:
Enabled channels: 1, 2, 5
Buffer Size: 10
The table shows the index the data is stored to for channel and cycle.

sequencer cycle	channel 1	channel 2	channel 5
1st	0	1	2
2nd	3	4	5
3rd	6	7	8
4th	9	0	1
5th	2	3	4
...	...	...	...

Example

```
#include “tpmc851.h”

TPMC851_DEV                 pDev;
STATUS                      result;
PTP851_ADC_SEQDATA_BUF      seqBuf;
long                        realBufSize;

/*-------------------------------
  allocate Buffer (100 word FIFO)
  -------------------------------*/
realBufSize = TP851_CALC_SIZE_ADC_SEQDATA_BUF(100);
seqBuf = (PTP851_ADC_SEQDATA_BUF)malloc(realBufSize);

seqBuf->bufSize = 100;

/*-------------------------------------------------
  Start sequencer with a buffer of 100 words and
  a cycle time of 100 ms, do not use external trigger
  -------------------------------------------------*/
result = tpmc851AdcSeqStart(
              pDev,
              1000,                     /* Cycle Time (in 100us)    */
              0,                        /* Flags                    */
              seqBuf );                 /* Sequencer buffer         */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC sequencer is already started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_IFLAGCOMB	invalid flag combination is specified
S_tp851Drv_NULL	the buffer pointer is NULL

3.2.4 tpmc851AdcSeqStop()

Name

tpmc851AdcSeqStop() – Stop ADC Sequencer

Synopsis

```
STATUS tpmc851AdcSeqStop
(
    TPMC851_DEV                 pDev
);
```

Description

This function stops the ADC sequencer. All sequencer channel configurations are still valid after stopping.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc851.h”

TPMC851_DEV                 pDev;
STATUS                      result;

/*------------------
  Stop ADC sequencer
  ------------------*/
result = tpmc851AdcSeqStop( pDev );

if (result != ERROR)
{
    /* function succeeded */
} else {
    /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_NOTBUSY	the ADC sequencer is not started

3.2.5 tpmc851DacWrite()

Name

tpmc851DacWrite() – Write to DAC channel

Synopsis

```
STATUS tpmc851DacWrite
(
        TPMC851_DEV     pDev,
        int             channel,
        unsigned long   flags,
        short           DacValue
);
```

Description

This function writes a value to the DAC register and starts the conversion if specified.

The DAC sequencer must be stopped for single DAC writes.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

channel

Specifies the DAC channel number. Valid values are 1..8.

flags

Is an ORed value of the following flags:

flag	description
TP851_F_CORR	If set the function will correct the *dacValue* before writing to DAC channel. Factory set and module dependent correction data is used for correction. If not set, *dacValue* is written to the DAC channel.
TP851_F_NOUPDATE	If set the DACs will not update after changing the DAC value. The output voltage will change with the next write with unset *TP851_F_NOUPDATE* flag. If not set the DAC will immediately convert and output the new voltage.

DacValue

This value is written to the DAC channel.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;
short              AdcValue;

/*-------------------------------------------------------------
  Write uncorrected 0x4000 to DAC channel 5, immediate convert
  -------------------------------------------------------------
result = tpmc851DacWrite(
              pDev,
              5,                                /* Channel    */
              0,                                /* Flags      */
              0x4000 );                         /* DAC value  */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the DAC is busy, the DAC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid DAC channel number specified

3.2.6 tpmc851DacSeqConfig()

Name

tpmc851DacSeqConfig() – Configure DAC sequencer channel

Synopsis

```
STATUS tpmc851DacSeqConfig
(
        TPMC851_DEV     pDev,
        int             channel,
        int             enable,
        unsigned long   flags
);
```

Description

This function enables and configures, or disables a DAC channel for sequence use.

The DAC sequencer must be stopped to execute this function.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

channel

Specifies the DAC channel number to configure. Valid values are 1..8.

enable

Specifies if the channel shall be used in sequencer mode or not. (0 disables the channel, any other value will enable the channel)

flags

The following optional flag is possible:

flag	description
TP851_F_CORR	If set the function will correct the dacValue before writing to DAC channel. Factory set and module dependent correction data is used for correction. If not set, dacValue is written to the DAC channel.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-----------------------------------------------------
  Configure DAC channel 1, using corrected data while
  the sequencer is running
  -----------------------------------------------------*/
result = tpmc851DacSeqConfig(
              pDev,
              1,                                /* Channel    */
              1,                                /* Enable     */
              TP851_F_CORR );                   /* Flags      */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC is busy, the ADC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid ADC channel number specified

3.2.7 tpmc851DacSeqStart()

Name

tpmc851DacSeqStart() – Start DAC Sequencer

Synopsis

```
STATUS tpmc851DacSeqStart
(
    TPMC851_DEV                 pDev,
    unsigned short              cycTime,
    unsigned long               flags,
    TP851_DAC_SEQDATA_BUF       *pDacSeqBuf
);
```

Description

This function configures the DAC sequencer time and starts the DAC sequencer.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

cycTime

Specifies the DAC sequencer cycle time. The sequencer time is specified in 100µs steps. With a value of 0, the “Sequencer Continuous Mode” is selected.

flags

Is an ORed value of the following flags:

flag	description
TP851_F_EXTTRIGSRC	If set the DAC sequencer is triggered with digital I/O line 1. If not set, the DAC sequencer uses the DAC cycle counter.
TP851_F_EXTTRIGOUT	If set the DAC trigger is used as output on digital I/O line 1.
TP851_F_DACSEQREPEAT	If set the DAC will repeat data when the end of the buffer is reached, the error *TP851_SF_SEQFIFOUNDERFLOW* is suppressed.

TP851_F_EXTTRIGSRC* and *TP851_F_EXTTRIGOUT cannot be used at the same time.

dacSeqBuf

Points to a DAC sequencer buffer. The buffer is used to supply the DAC data to the driver. The DAC sequencer buffer is defined as a structure named *TP851_DAC_SEQDATA_BUF*. This structure is defined in *tpmc851.h*. The size of the buffer is variable, therefore a macro *TP851_CALC_SIZE_DAC_SEQDATA_BUF(s)* is defined in *tpmc851.h*, that calculates the size to allocate for the buffer. The macro-parameter *s* specifies the size of the *buffer* array of the structure.

```
typedef struct
{
    long        putIdx;
    long        getIdx;
    long        bufSize;
    long        seqState;
    short       buffer[1];
} TP851_DAC_SEQDATA_BUF, *PTP851_DAC_SEQDATA_BUF;
```

putIdx

Specifies the index into *buffer* where the next data shall be stored. This index must be handled by the application and is only be read by the driver to check a FIFO underrun.

getIdx

Specifies the index into *buffer* where the next data will be read from. This index is handled by the driver and should only be read by the application to check if there is space for new data.

bufSize

Specifies the array size of *buffer*. This value must be the same as used for *s* in *TP851_CALC_SIZE_ADC_SEQDATA_BUF(s)* when calculating the allocation size for *adcSeqBuf*.

seqState

Returns the sequencer state. This is an ORed value of the following status flags:

flag	**description**
TP851_SF_SEQACTIVE	If set the DAC sequencer is started. If not set, the DAC sequencer stopped.
TP851_SF_SEQUNDERFLOWERR	If set the DAC sequencer has detected an underrun error. (Hardware detected)
TP851_SF_SEQFIFOUNDERFLOW	If set the application supplied FIFO (*buffer*) is empty and the sequencer could not read new data from FIFO.

buffer

Array used for DAC sequencer data FIFO.
The DAC data is stored by the application into this FIFO. The assignment from data to channel is done as follows. The first data will be used for the lowest enabled channel, the second from the next enabled channel and so on. There will be no data used for disabled channels. If the end of *buffer* is reached the next data will be read again from the beginning of the buffer.

Example:
Enabled channels: 1, 2, 5
Buffer Size: 10
The table shows the index the data is used for channel and cycle.

sequencer cycle	channel 1	channel 2	channel 5
1st	0	1	2
2nd	3	4	5
3rd	6	7	8
4th	9	0	1
5th	2	3	4
…	…	…	…

Example

```
#include “tpmc851.h”

TPMC851_DEV                 pDev;
STATUS                      result;
PTP851_DAC_SEQDATA_BUF      seqBuf;
long                        realBufSize;

/*------------------------------
  allocate Buffer (100 word FIFO)
  ------------------------------*/
realBufSize = TP851_CALC_SIZE_DAC_SEQDATA_BUF(100);
seqBuf = (PTP851_DAC_SEQDATA_BUF)malloc(realBufSize);

seqBuf->bufSize = 100;

/*-----------
  Fill buffer
  -----------*/
seqBuf->buffer[0] = …;
seqBuf->buffer[1] = …;
seqBuf->buffer[2] = …;

…
```

```
/*-------------------------------------------------------------
  Start sequencer with a buffer of 100 words and
  a cycle time of 100 ms, do not use external trigger, repeat data
  -------------------------------------------------------------*/
result = tpmc851DacSeqStart(
             pDev,
             1000,                        /* Cycle Time (in 100us)    */
             TP851_F_DACSEQREPEAT,        /* Flags                    */
             seqBuf );                    /* Sequencer buffer         */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the DAC sequencer is already started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_IFLAGCOMB	invalid flag combination is specified
S_tp851Drv_NULL	the buffer pointer is NULL

3.2.8 tpmc851DacSeqStop()

Name

tpmc851DacSeqStop() – Stop DAC Sequencer

Synopsis

```
STATUS tpmc851DacSeqStop
(
    TPMC851_DEV                 pDev
);
```

Description

This function stops the DAC sequencer. All sequencer channel configurations are still valid after stopping.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include "tpmc851.h"

TPMC851_DEV                 pDev;
STATUS                      result;

/*------------------
  Stop DAC sequencer
  ------------------*/
result = tpmc851DacSeqStop( pDev );

if (result != ERROR)
{
    /* function succeeded */
} else {
    /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_NOTBUSY	the DAC sequencer is not started

3.2.9 tpmc851IoRead()

Name

tpmc851IoRead() – Read from digital I/O

Synopsis

```
STATUS tpmc851IoRead
(
        TPMC851_DEV     pDev,
        unsigned short  *pIoValue
);
```

Description

This function reads the current value of digital I/O input.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

pIoValue

This parameter specifies a pointer to an *unsigned short* value which receives the current I/O value. Bit 0 corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;
unsigned short     IoValue;

/*--------------------
  Read I/O input value
  --------------------*/
result = tpmc851IoRead(
              pDev,
              &IoValue );                      /* I/O value  */

if (result != ERROR)
{
     /* function succeeded */
     printf("    I/O-value: 0x%04X", IoValue);
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.10 tpmc851IoWrite()

Name

tpmc851IoWrite() – Write to digital I/O

Synopsis

```
STATUS tpmc851IoWrite
(
        TPMC851_DEV     pDev,
        unsigned short  IoValue
);
```

Description

This function writes a value to digital I/O output.

Only I/O lines configured for output will be affected. Please refer to chapter 3.2.11 tpmc851IoConfig().

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

IoValue

This value is written to the I/O output. Bit 0 corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*------------------------------
  Write I/O output value 0x1234
  ------------------------------*/
result = tpmc851IoWrite(
              pDev,
              0x1234 );                          /* I/O value  */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.11 tpmc851IoConfig()

Name

tpmc851IoConfig() – Configure direction of digital I/O

Synopsis

```
STATUS tpmc851IoConfig
(
    TPMC851_DEV     pDev,
    unsigned short  Direction
);
```

Description

This function configures the direction (input/output) of the digital I/O lines.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Direction

Specifies the new direction setting for digital I/O. A bit set to 1 enables output, a 0 means that the I/O line is input. Bit 0 corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Enable line 0,2,8 for output, all other lines are input
  -------------------------------------------------------*/
result = tpmc851IoConfig(
              pDev,
              (1 << 0) | (1 << 2) | (1 << 8) ); /* Direction */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.12 tpmc851IoDebConfig()

Name

tpmc851IoDebConfig() – Configure digital I/O (input) debouncer

Synopsis

```
STATUS tpmc851IoDebConfig
(
        TPMC851_DEV     pDev,
        unsigned short  EnableMask,
        unsigned short  DebounceTime
);
```

Description

This function configures the digital I/O input debouncing mechanism to avoid detection of invalid signal changes in noisy environments.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

EnableMask

Specifies digital I/O lines to be filtered by the debouncing mechanism. A bit set to 1 enables the debouncer, and a 0 disables the debouncer for the corresponding I/O line. Bit 0 corresponds to the first I/O line, bit 1 corresponds to the second I/O line and so on.

DebounceTime

Specifies the debounce time. The time is specified in 100ns steps, using the following formula:
Debounce duration = (DebounceTimeValue * 100ns) + 100ns

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Enable Debouncer for line 1 and 2 (debounce time 1ms)
  -------------------------------------------------------*/
result = tpmc851IoDebConfig(
              pDev,
              (1 << 1) | (1 << 2),    /* EnableMask                    */
              10000 );                /* DebounceTime (in 100ns steps) */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.13 tpmc851IoEventWait()

Name

tpmc851IoEventWait() – Wait for I/O event

Synopsis

```
STATUS tpmc851IoEventWait
(
        TPMC851_DEV     pDev,
        int             IoLine,
        unsigned long   flags,
        long            timeout
);
```

Description

This function waits for an I/O input event.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

IoLine

Specifies the digital I/O line where the event shall occur. Valid values are 0..15.

flags

Specifies the event that shall occur. This is an ORed value of the following flags:

flag	description
TP851_F_HI2LOTRANS	If set, the function will return after a high to low transition occurs.
TP851_F_LO2HITRANS	If set, the function will return after a low to high transition occurs.

At least one flag must be specified.

timeout

Specifies the maximum time the function will wait for the specified event. The time is specified in system ticks.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------------
  Wait for any transition on I/O line 12 (max wait 10000 ticks)
  -------------------------------------------------------------*/
result = tpmc851IoEventWait(
              pDev,
              12,                                          /* IoLine     */
              TP851_F_HI2LOTRANS | TP851_F_LO2HITRANS,     /* Flags      */
              10000 );                                     /* Timeout    */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid I/O line specified
S_tp851Drv_NOFREEEVENT	no free event object available
S_tp851Drv_TIMEOUT	timeout has occurred

3.2.14 tpmc851CntRead()

Name

tpmc851CntRead() – Read counter/timer value

Synopsis

```
STATUS tpmc851CntRead
(
        TPMC851_DEV     pDev,
        unsigned long   *pCounterValue,
        unsigned long   *pCounterStatus
);
```

Description

This function reads the value of the counter.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

pCounterValue

This parameter is a pointer to an *unsigned long* data buffer where the current counter value is stored.

pCounterStatus

This parameter is a pointer to an *unsigned long* data buffer where the counter status is returned. If possible the flags are cleared after read. This is an ORed value of the following flags.

flag	description
TP851_SF_CNTBORROW	Counter borrow bit set (current state)
TP851_SF_CNTCARRY	Counter carry bit set (current state)
TP851_SF_CNTMATCH	Counter match event has occurred since last read.
TP851_SF_CNTSIGN	Counter sign bit (current state)
TP851_SF_CNTDIRECTION	If set, counter direction is upward. If not set, counter direction is downward.
TP851_SF_CNTLATCH	Counter value has been latched.
TP851_SF_CNTLATCHOVERFLOW	Counter latch overflow has occurred.
TP851_SF_CNTSNGLCYC	Counter Single Cycle is active

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;
unsigned long      CounterValue;
unsigned long      CounterStatus;


/*------------------
  Read counter value
  ------------------*/
result = tpmc851CntRead(
              pDev,
              &CounterValue,                  /* Counter Value   */
              &CounterStatus );               /* Counter Status  */

if (result != ERROR)
{
     /* function succeeded */
     printf("    Counter: %ld", CounterValue);
     printf("    State:   %lXh", CounterStatus);
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.15 tpmc851CntConfig()

Name

tpmc851CntConfig() – Configure counter

Synopsis

```
STATUS tpmc851CntConfig
(
        TPMC851_DEV     pDev,
        unsigned long   inputMode,
        int             clockDivider,
        unsigned long   countMode,
        unsigned long   controlMode,
        unsigned long   invFlags
);
```

Description

This function configures the counter.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

inputMode

Specifies the counter input mode. The following modes are defined and valid:

flag	**description**
TP851_M_CNTIN_DISABLE	Counter disabled
TP851_M_CNTIN_TIMERUP	Timer Mode Up
TP851_M_CNTIN_TIMERDOWN	Timer Mode Down
TP851_M_CNTIN_DIRCOUNT	Direction Count
TP851_M_CNTIN_UPDOWNCOUNT	Up/Down Count
TP851_M_CNTIN_QUAD1X	Quadrature Count 1x
TP851_M_CNTIN_QUAD2X	Quadrature Count 2x
TP851_M_CNTIN_QUAD3X	Quadrature Count 4x

clockDivider

Specifies clock divider for Timer Mode. Allowed clock divider values are:

clock divider value	clock input frequency
1	40 MHz
2	20 MHz
4	10 MHz
8	5 MHz

countMode

Specifies the count mode. The following modes are defined and valid:

flag	description
TP851_M_CNT_CYCLE	Cycling Counter
TP851_M_CNT_DIVN	Divide-by-N
TP851_M_CNT_SINGLE	Single Cycle

controlMode

Specifies the counter control mode. These events can generate counter control events. The following modes are defined and valid:

flag	description
TP851_M_CNTCTRL_NONE	No Control Mode
TP851_M_CNTCTRL_LOAD	Load Mode
TP851_M_CNTCTRL_LATCH	Latch Mode
TP851_M_CNTCTRL_GATE	Gate Mode
TP851_M_CNTCTRL_RESET	Reset Mode

invFlags

Specifies if counter input lines shall be inverted or not. This is an ORed value of the following flags:

flag	description
TP851_F_CNTINVINP2	If set, input line 2 is low active If not set, input line 2 is high active
TP851_F_CNTINVINP3	If set, input line 3 is low active If not set, input line 3 is high active
TP851_F_CNTINVINP4	If set, input line 4 is low active If not set, input line 4 is high active

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Setup counter for direction count, clock divider 1,
  cycling count, no control mode and all lines high active
  -------------------------------------------------------*/
result = tpmc851CntConfig(
              pDev,
              TP851_M_CNTIN_DIRCOUNT,      /* inputMode       */
              1,                           /* clockDivider    */
              TP851_M_CNT_CYCLE,           /* countMode       */
              TP851_M_CNTCTRL_NONE,        /* controlMode     */
              0 );                         /* invFlags        */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_IMODE	invalid mode specified
S_tp851Drv_IFLAG	invalid flag specified

3.2.16 tpmc851CntReset()

Name

tpmc851CntReset() – Reset counter

Synopsis

```
STATUS tpmc851CntReset
(
     TPMC851_DEV                pDev
);
```

Description

This function resets the counter value to 0x00000000.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

Example

```
#include “tpmc851.h”

TPMC851_DEV                 pDev;
STATUS                      result;

/*--------------------
  Reset counter value
  --------------------*/
result = tpmc851CntReset( pDev );

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.17 tpmc851CntSetPreload()

Name

tpmc851CntSetPreload() – Set counter preload value

Synopsis

```
STATUS tpmc851CntSetPreload
(
        TPMC851_DEV     pDev,
        unsigned long   PreloadValue,
        unsigned long   PreloadFlags
);
```

Description

This function sets the counter preload value, either immediately or on the next preload condition.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

PreloadValue

Specifies the new counter preload value.

PreloadFlags

The following flag is optional:

flag	description
TP851_F_IMMPRELOAD	If set, the function will immediately load the preload value into the counter If not set, preload value will be used for the next preload condition.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;


/*-----------------------------------------
  Immediately load 0x11223344 into the counter
  and preload register
  -----------------------------------------*/
result = tpmc851CntSetPreload(
              pDev,
              0x11223344,                    /* Preload Value   */
              TP851_F_IMMPRELOAD );          /* Flags           */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_IFLAG	invalid flag specified

3.2.18 tpmc851CntSetMatch()

Name

tpmc851CntSetMatch() – Set counter match value

Synopsis

```
STATUS tpmc851CntSetMatch
(
    TPMC851_DEV     pDev,
    unsigned long   MatchValue
);
```

Description

This function sets the counter match value. If counter and match value are the same, a match event occurs. The driver can wait for this event.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

MatchValue

Specifies the new counter match value.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;


/*--------------------------
  Set match value to 0x10000
  --------------------------*/
result = tpmc851CntSetMatch(
              pDev,
              0x10000 );              /* MatchValue */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error code is a standard error code set by the I/O system.

3.2.19 tpmc851CntMatchWait()

Name

tpmc851CntMatchWait() – Wait for counter match event

Synopsis

```
STATUS tpmc851CntMatchWait
(
        TPMC851_DEV     pDev,
        long            timeout
);
```

Description

This function waits for a counter match event.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

timeout

Specifies the maximum time the function will wait for the counter match event. The time is specified in system ticks.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-------------------------------------------------
  Wait for counter match event (max wait 10000 ticks)
  -------------------------------------------------*/
result = tpmc851CntMatchWait(
              10000 );                                  /* Timeout    */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_NOFREEEVENT	no free event object available
S_tp851Drv_TIMEOUT	timeout has occurred

3.2.20 tpmc851CntCtrlWait()

Name

tpmc851CntCtrlWait() – Wait for counter control event

Synopsis

```
STATUS tpmc851CntCtrlWait
(
    TPMC851_DEV     pDev,
    long            timeout
);
```

Description

This function waits for counter control event. The event to wait for is chosen with API function *tpmc851CntConfig(),* specifying the parameter *controlMode*.

Parameters

pDev

This parameter specifies the device descriptor to the hardware module retrieved by a call to the corresponding open-function.

timeout

Specifies the maximum time the function will wait for the counter control event. The time is specified in system ticks.

Example

```
#include “tpmc851.h”

TPMC851_DEV        pDev;
STATUS             result;

/*-----------------------------------------------------
  Wait for counter control event (max wait 10000 ticks)
  -----------------------------------------------------*/
result = tpmc851CntCtrlWait(
              10000 );                                  /* Timeout    */

if (result != ERROR)
{
     /* function succeeded */
} else {
     /* handle error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

Error code	Description
S_tp851Drv_NOFREEEVENT	no free event object available
S_tp851Drv_TIMEOUT	timeout has occurred

4 Legacy I/O system functions

This chapter describes the legacy 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.

The legacy I/O system functions are only relevant for the legacy TPMC851 driver. For the VxBus-enabled TPMC851 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tp851Drv()

NAME

tp851Drv() - installs the TPMC851 driver in the I/O system and initializes the driver.

SYNOPSIS

#include “tpmc851.h”

STATUS tp851Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC851 driver 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 "tpmc851.h”

/*-------------------
  Initialize Driver
  -------------------*/
status = tp851Drv();
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 codes are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_tp851Drv_NOMEM	memory allocation failed
S_tp851Drv_NXIO	no TPMC851 found

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tp851DevCreate()

NAME

tp851DevCreate() – Add a TPMC851 device to the VxWorks system and initializes device hardware

SYNOPSIS

```
#include “tpmc851.h”

STATUS tp851DevCreate
(
    char        *name,
    int         globModNo
)
```

DESCRIPTION

This routine adds the selected 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.

glbModNo

This index number specifies the device to add to the system. The *glbModNo* is assigned when searching for the TPMC851 in *tp851Drv()*. The first module that has been found will be identified with 0, the second with 1, and so on. The search order depends on the BSP.

EXAMPLE

```
#include "tpmc851.h"

STATUS            result;

/*-----------------------------------------------------
  Create the device "/tpmc851/0" for the first device
  -----------------------------------------------------*/
result = tp851DevCreate(    "/tpmc851/0",
                            0);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* 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 are stored in *errno* and can be read with the function *errnoGet()*.

Error code	Description
S_tp851Drv_NODRV	the TPMC851 driver has not been installed
S_tp851Drv_NXIO	no TPMC851 found
S_tp851Drv_EXISTS	a device for the specified TPMC851 has already been created

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.3 tp851PciInit()

NAME

tp851PciInit() – Generic PCI device initialization

SYNOPSIS

void tp851PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC851 PCI spaces (base address register) and to enable the TPMC851 device for access.

The global variable *tp851Status* obtains the result of the device initialization and can be polled later by the application before the driver will be installed.

Value	Meaning
> 0	Initialization successful completed. The value of tp851Status is equal to the number of mapped PCI spaces
0	No Tp851 device found
< 0	Initialization failed. The value of (tp851Status & 0xFF) is equal to the number of mapped spaces until the error occurs. Possible cause: Too few entries for dynamic mappings in sysPhysMemDesc[]. Remedy: Add dummy entries as necessary (syslib.c).

EXAMPLE

```
extern void tp851PciInit();

tp851PciInit();
```

5 Basic I/O Functions

The VxWorks basic I/O interface functions are useable with the TPMC851 legacy and VxBus-enabled driver in a uniform manner.

5.1 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 TPMC851 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 tp851DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tpmc851/0" for I/O
  ----------------------------------------*/
fd = open("/tpmc851/0", 0, 0);
if (fd == ERROR)
{
     /* Error handling */
}
```

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()*

5.2 close()

NAME

close() – close a device or file

SYNOPSIS

```
STATUS 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    retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == 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()*.

SEE ALSO

ioLib, basic I/O routine - close()

5.3 ioctl()

NAME

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

SYNOPSIS

#include “tpmc851.h”

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

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls (read, write) will be performed by calling the ioctl() function.

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_TP851_ADC_READ	Read value from ADC channel
FIO_TP851_ADC_SEQCONFIG	Configure ADC sequencer channel
FIO_TP851_ADC_SEQSTART	Start ADC Sequencer
FIO_TP851_ADC_SEQSTOP	Stop ADC Sequencer
FIO_TP851_DAC_WRITE	Write to DAC channel
FIO_TP851_DAC_SEQCONFIG	Configure DAC sequencer channel
FIO_TP851_DAC_SEQSTART	Start DAC Sequencer
FIO_TP851_DAC_SEQSTOP	Stop DAC Sequencer

FIO_TP851_IO_READ	Read from digital I/O
FIO_TP851_IO_WRITE	Write to digital I/O
FIO_TP851_IO_EVENTWAIT	Wait for I/O event
FIO_TP851_IO_CONFIG	Configure digital I/O
FIO_TP851_IO_DEBCONFIG	Configure digital I/O (input) debouncer
FIO_TP851_CNT_READ	Read counter/timer value
FIO_TP851_CNT_MATCHWAIT	Wait for counter match event
FIO_TP851_CNT_CTRLWAIT	Wait for counter control event
FIO_TP851_CNT_CONFIG	Configure counter
FIO_TP851_CNT_RESET	Reset counter
FIO_TP851_CNT_SETPRELD	Set counter preload value
FIO_TP851_CNT_SETMATCH	Set counter match value

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) or a driver specific code described below. Function specific error codes will be described with the function.

Error code	Description
S_tp851Drv_ICMD	An illegal request code has been specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TP851_ADC_READ

This function starts an ADC conversion with specified parameters, waits for completion and returns the value.

The ADC sequencer must be stopped for single ADC conversions.

For this operation the argument **arg** points to a structure named *TP851_ADC_READ_BUF*.

```
typedef struct
{
        int             channel;
        int             gain;
        unsigned long   flags;
        short           adcValue;
} TP851_ADC_READ_BUF, *PTP851_ADC_READ_BUF;
```

channel

Specifies the ADC channel number. Valid values are 1..16 for differential input and 1..32 for single-ended input.

gain

Specifies the input gain. Valid gain values are 1, 2, 4, and 8.

flags

Is an ORed value of the following flags:

flag	**description**
TP851_F_CORR	If set the function will return a corrected value of the input data in *adcValue*. Factory set and module dependent correction data is used for correction. If not set, the raw value read from the module will be returned in *adcValue*.
TP851_F_IMMREAD	If set the driver will start the conversion without waiting for settling time. This should only be used if the previous conversion has used the same interface parameters (channel, gain, differential/single-ended). If not set the driver will use the automatic mode, which sets interface configuration, waits settling time and then starts the conversion.
TP851_F_DIFF	If set the input channel will be a differential input. If not set the input channel will be a single-ended input.

adcValue

This value will return the read ADC value.

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;
TP851_ADC_READ_BUF      adcReadBuf;

/*---------------------------------------------------------------------
  Read a corrected value from differential channel 2 use a gain of 4
  ---------------------------------------------------------------------*/
adcReadBuf.channel = 2;
adcReadBuf.gain    = 4;
adcReadBuf.flags   = TP851_F_CORR | TP851_F_DIFF;

printf("Read from ADC ... ");
retval = ioctl(    fd,
                   FIO_TP851_ADC_READ,
                   (int)&adcReadBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
     printf("    ADC-value: %d", adcReadBuf.adcValue);
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC is busy, the ADC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid ADC channel number specified
S_tp851Drv_IGAIN	invalid gain specified
S_tp851Drv_TIMEOUT	ADC conversion timed out

5.3.2 FIO_TP851_ADC_SEQCONFIG

This function enables and configures, or disables an ADC channel for sequencer use.

The ADC sequencer must be stopped to execute this function.

For this operation the argument **arg** points to a structure named *TP851_ADC_SEQCONFIG_BUF*.

```
typedef struct
{
        int             channel;
        int             enable;
        int             gain;
        unsigned long   flags;
} TP851_ADC_SEQCONFIG_BUF, *PTP851_ADC_SEQCONFIG_BUF;
```

channel

Specifies the ADC channel number to configure. Valid values are 1..16 for differential input and 1..32 for single-ended input.

enable

Specifies if the channel shall be used in sequencer mode or not. (0 disables the channel, any other value will enable the channel)

gain

Specifies the input gain. Valid gain values are 1, 2, 4, and 8.

flags

Is an ORed value of the following flags:

flag	description
TP851_F_CORR	If set the sequencer will return a corrected value for the specified channel. Factory set and module dependent correction data is used for correction. If not set, the raw value read from the module will be returned.
TP851_F_DIFF	If set the input channel will be a differential input. If not set the input channel will be a single-ended input.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_ADC_SEQCONFIG_BUF     adcSeqConfBuf;

/*-------------------------------------------------------
  Configure single-ended channel 3, using a gain of 4 and
  returning corrected data when the sequencer is running
  -------------------------------------------------------*/
adcSeqConfBuf.channel   = 3;
adcSeqConfBuf.enable    = TRUE;
adcSeqConfBuf.gain      = 4;
adcSeqConfBuf.flags     = TP851_F_CORR;

printf("Configure channel for Sequencer ... ");
retval = ioctl(    fd,
                   FIO_TP851_ADC_SEQCONFIG,
                   (int)&adcSeqConfBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC is busy, the ADC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid ADC channel number specified
S_tp851Drv_IGAIN	invalid gain specified

5.3.3 FIO_TP851_ADC_SEQSTART

This function configures the ADC sequencer time and starts the ADC sequencer.

For this operation the argument **arg** points to a structure named *TP851_ADC_SEQSTART_BUF*.

```
typedef struct
{
    unsigned short              cycTime;
    unsigned long               flags;
    PTP851_ADC_SEQDATA_BUF      adcSeqBuf;
} TP851_ADC_SEQSTART_BUF, *PTP851_ADC_SEQSTART_BUF;
```

cycTime

Specifies the ADC sequencer cycle time. The sequencer time is specified in 100µs steps. With a value of 0, the “Sequencer Continuous Mode” is selected.

flags

One of the following optional flags is possible:

flag	**description**
TP851_F_EXTTRIGSRC	If set the ADC sequencer is triggered with digital I/O line 0. If not set, the ADC sequencer uses the ADC cycle counter.
TP851_F_EXTTRIGOUT	If set the ADC trigger is used as output on digital I/O line 0.

adcSeqBuf

Points to an ADC sequencer buffer. The buffer is used to store the ADC data. The ADC sequencer buffer is defined as a structure named *TP851_ADC_SEQDATA_BUF*. This structure is defined in *tpmc851.h*. The size of the buffer is variable, therefore a macro *TP851_CALC_SIZE_ADC_SEQDATA_BUF(s)* is defined in *tpmc851.h*, that calculates the size to allocate for the buffer. The macro-parameter *s* specifies the size of the *buffer* array of the structure.

```
typedef struct
{
    long        putIdx;
    long        getIdx;
    long        bufSize;
    long        seqState;
    short       buffer[1];
} TP851_ADC_SEQDATA_BUF, *PTP851_ADC_SEQDATA_BUF;
```

putIdx

Specifies the index into *buffer* where the next data will be stored to. This index is handled by the driver and should only be read by the application to check if data is available. The driver initializes this index to 0 when sequencer starts.

getIdx

Specifies the index into *buffer* where the next input data can be read from. This index must be handled by the application and is only be read by the driver to check a FIFO overflow. The driver initializes this index to 0 when sequencer starts.

bufSize

Specifies the array size of *buffer*. This value must be the same as used for *s* in *TP851_CALC_SIZE_ADC_SEQDATA_BUF(s)* when calculating the allocation size for *adcSeqBuf*.

seqState

Returns the sequencer state. This is an ORed value of the following status flags:

flag	**description**
TP851_SF_SEQACTIVE	If set the ADC sequencer is started. If not set, the ADC sequencer stopped.
TP851_SF_SEQOVERFLOWERR	If set the ADC sequencer has detected an overflow error. (Hardware detected)
TP851_SF_SEQTIMERERROR	If set the ADC sequencer has detected a timer error. (Hardware detected)
TP851_SF_SEQIRAMERROR	If set the ADC sequencer has detected an instruction RAM error. (Hardware detected)
TP851_SF_SEQFIFOOVERFLOW	If set the application supplied FIFO (*buffer*) has overrun. Data got lost.

buffer

Array used for ADC sequencer data FIFO.
The ADC data is stored by the sequencer into this FIFO. The assignment from data to channel is done as follows. The first data will be from the lowest enabled channel, the second from the next enabled channel and so on. There will be no data stored for disabled channels. If the end of *buffer* is reached the next data will be stored again at the beginning of the buffer.

Example:
Enabled channels: 1, 2, 5
Buffer Size: 10
The table shows the index the data is stored to for channel and cycle.

sequencer cycle	**channel 1**	**channel 2**	**channel 5**
1st	0	1	2
2nd	3	4	5
3rd	6	7	8
4th	9	0	1
5th	2	3	4
…	…	…	…

EXAMPLE

```
#include “tpmc851.h”

int                          fd;
int                          retval;
TP851_ADC_SEQSTART_BUF       adcSeqStartBuf;
PTP851_ADC_SEQDATA_BUF       seqBuf;
long                         realBufSize;

/*------------------------------
  allocate Buffer (100 word FIFO)
  ------------------------------*/
realBufSize = TP851_CALC_SIZE_ADC_SEQDATA_BUF(100);
seqBuf = (PTP851_ADC_SEQDATA_BUF)malloc(realBufSize);

seqBuf->bufSize = 100;

/*-------------------------------------------------
  Start sequencer with a buffer of 100 words and
  a cycle time of 100 ms, do not use external trigger
  -------------------------------------------------*/
adcSeqStartBuf.cycTime       = 1000;
adcSeqStartBuf.flags         = 0;
adcSeqStartBuf.adcSeqBuf     = seqBuf;

printf("Start Sequencer ... ");
retval = ioctl(    fd,
                   FIO_TP851_ADC_SEQSTART,
                   (int)&adcSeqStartBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the ADC sequencer is already started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_IFLAGCOMB	invalid flag combination is specified
S_tp851Drv_NULL	the buffer pointer is NULL

5.3.4 FIO_TP851_ADC_SEQSTOP

This function stops the ADC sequencer. All sequencer channel configurations are still valid after stopping.

For this operation the argument **arg** is not used and should be 0

EXAMPLE

```
#include “tpmc851.h”

int                fd;
int                retval;

/*------------------
  Stop ADC sequencer
  ------------------*/
printf("Stop ADC sequencer ... ");
retval = ioctl(    fd,
                   FIO_TP851_ADC_SEQSTOP,
                   0);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_NOTBUSY	the ADC sequencer is not started

5.3.5 FIO_TP851_DAC_WRITE

This function writes a value to the DAC register and starts the conversion if specified.

The DAC sequencer must be stopped for single DAC writes.

For this operation the argument **arg** points to a structure named *TP851_DAC_WRITE_BUF*.

```
typedef struct
{
        int             channel;
        unsigned long   flags;
        short           dacValue;
} TP851_DAC_WRITE_BUF, *PTP851_DAC_WRITE_BUF;
```

channel

Specifies the DAC channel number. Valid values are 1..8.

flags

Is an ORed value of the following flags:

flag	description
TP851_F_CORR	If set the function will correct the *dacValue* before writing to DAC channel. Factory set and module dependent correction data is used for correction. If not set, *dacValue* is written to the DAC channel.
TP851_F_NOUPDATE	If set the DACs will not update after changing the DAC value. The output voltage will change with the next write with unset *TP851_F_NOUPDATE* flag. If not set the DAC will immediately convert and output the new voltage.

dacValue

This value is written to the DAC channel.

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;
TP851_DAC_WRITE_BUF     dacWriteBuf;

/*-------------------------------------------------------------
  Write uncorrected 0x4000 to DAC channel 5, immediate convert
  -------------------------------------------------------------
dacWriteBuf.channel     = 5;
dacWriteBuf.flags       = 0;
dacWriteBuf.dacValue    = 0x4000;

printf("Write to DAC ... ");
retval = ioctl(    fd,
                   FIO_TP851_DAC_WRITE,
                   (int)&dacWriteBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the DAC is busy, the DAC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid DAC channel number specified

5.3.6 FIO_TP851_DAC_SEQCONFIG

This function enables and configures, or disables a DAC channel for sequence use.

The DAC sequencer must be stopped to execute this function.

For this operation the argument **arg** points to a structure named *TP851_DAC_SEQCONFIG_BUF*.

```
typedef struct
{
        int             channel;
        int             enable;
        unsigned long   flags;
} TP851_DAC_SEQCONFIG_BUF, *PTP851_DAC_SEQCONFIG_BUF;
```

channel

Specifies the DAC channel number to configure. Valid values are 1..8.

enable

Specifies if the channel shall be used in sequencer mode or not. (0 disables the channel, any other value will enable the channel)

flags

The following optional flag is possible:

flag	description
TP851_F_CORR	If set the function will correct the dacValue before writing to DAC channel. Factory set and module dependent correction data is used for correction. If not set, dacValue is written to the DAC channel.

EXAMPLE

```
#include "tpmc851.h"

int                         fd;
int                         retval;
TP851_DAC_SEQCONFIG_BUF     dacSeqConfBuf;

/*-----------------------------------------------------
  Configure DAC channel 1, using corrected data while
  the sequencer is running
  -----------------------------------------------------*/
dacSeqConfBuf.channel   = 1;
dacSeqConfBuf.enable    = TRUE;
dacSeqConfBuf.flags     = TP851_F_CORR;

…
```

…

```
printf("Configure channel for Sequencer ... ");
retval = ioctl(    fd,
                   FIO_TP851_DAC_SEQCONFIG,
                   (int)&dacSeqConfBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the DAC is busy, the DAC sequencer is started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid DAC channel number specified

5.3.7 FIO_TP851_DAC_SEQSTART

This function configures the DAC sequencer time and starts the DAC sequencer.

For this operation the argument **arg** points to a structure named *TP851_DAC_SEQSTART_BUF*.

```
typedef struct
{
    unsigned short              cycTime;
    unsigned long               flags;
    PTP851_DAC_SEQDATA_BUF      dacSeqBuf;
} TP851_DAC_SEQSTART_BUF, *PTP851_DAC_SEQSTART_BUF;
```

cycTime

Specifies the DAC sequencer cycle time. The sequencer time is specified in 100µs steps. With a value of 0, the “Sequencer Continuous Mode” is selected.

flags

Is an ORed value of the following flags:

flag	**description**
TP851_F_EXTTRIGSRC	If set the DAC sequencer is triggered with digital I/O line 1. If not set, the DAC sequencer uses the DAC cycle counter.
TP851_F_EXTTRIGOUT	If set the DAC trigger is used as output on digital I/O line 1.
TP851_F_DACSEQREPEAT	If set the DAC will repeat data when the end of the buffer is reached, the error *TP851_SF_SEQFIFOUNDERFLOW* is suppressed.

TP851_F_EXTTRIGSRC* and *TP851_F_EXTTRIGOUT cannot be used at the same time.

dacSeqBuf

Points to a DAC sequencer buffer. The buffer is used to supply the DAC data to the driver. The DAC sequencer buffer is defined as a structure named *TP851_DAC_SEQDATA_BUF*. This structure is defined in *tpmc851.h*. The size of the buffer is variable, therefore a macro *TP851_CALC_SIZE_DAC_SEQDATA_BUF(s)* is defined in *tpmc851.h*, that calculates the size to allocate for the buffer. The macro-parameter *s* specifies the size of the *buffer* array of the structure.

```
typedef struct
{
    long        putIdx;
    long        getIdx;
    long        bufSize;
    long        seqState;
    short       buffer[1];
} TP851_DAC_SEQDATA_BUF, *PTP851_DAC_SEQDATA_BUF;
```

putIdx

Specifies the index into *buffer* where the next data shall be stored. This index must be handled by the application and is only be read by the driver to check a FIFO underrun.

getIdx

Specifies the index into *buffer* where the next data will be read from. This index is handled by the driver and should only be read by the application to check if there is space for new data.

bufSize

Specifies the array size of *buffer*. This value must be the same as used for *s* in *TP851_CALC_SIZE_ADC_SEQDATA_BUF(s)* when calculating the allocation size for *adcSeqBuf*.

seqState

Returns the sequencer state. This is an ORed value of the following status flags:

flag	description
TP851_SF_SEQACTIVE	If set the DAC sequencer is started. If not set, the DAC sequencer stopped.
TP851_SF_SEQUNDERFLOWERR	If set the DAC sequencer has detected an underrun error. (Hardware detected)
TP851_SF_SEQFIFOUNDERFLOW	If set the application supplied FIFO (*buffer*) is empty and the sequencer could not read new data from FIFO.

buffer

Array used for DAC sequencer data FIFO.
The DAC data is stored by the application into this FIFO. The assignment from data to channel is done as follows. The first data will be used for the lowest enabled channel, the second from the next enabled channel and so on. There will be no data used for disabled channels. If the end of *buffer* is reached the next data will be read again from the beginning of the buffer.

Example:
Enabled channels: 1, 2, 5
Buffer Size: 10
The table shows the index the data is used for channel and cycle.

sequencer cycle	channel 1	channel 2	channel 5
1st	0	1	2
2nd	3	4	5
3rd	6	7	8
4th	9	0	1
5th	2	3	4
…	…	…	…

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_DAC_SEQSTART_BUF      dacSeqStartBuf;
PTP851_DAC_SEQDATA_BUF      seqBuf;
long                        realBufSize;

/*-------------------------------
  allocate Buffer (100 word FIFO)
  -------------------------------*/
realBufSize = TP851_CALC_SIZE_DAC_SEQDATA_BUF(100);
seqBuf = (PTP851_DAC_SEQDATA_BUF)malloc(realBufSize);

seqBuf->bufSize = 100;

/*-----------
  Fill buffer
  -----------*/
seqBuf->buffer[0] = …;
seqBuf->buffer[1] = …;

/*-----------------------------------------------------------------
  Start sequencer with a buffer of 100 words and
  a cycle time of 100 ms, do not use external trigger, repeat data
  -----------------------------------------------------------------*/
dacSeqStartBuf.cycTime      = 1000;
dacSeqStartBuf.flags        = TP851_F_DACSEQREPEAT;
dacSeqStartBuf.dacSeqBuf    = seqBuf;

printf("Start Sequencer ... ");
retval = ioctl(    fd,
                   FIO_TP851_DAC_SEQSTART,
                   (int)&dacSeqStartBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
} else {
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_BUSY	the DAC sequencer is already started
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_IFLAGCOMB	invalid flag combination is specified
S_tp851Drv_NULL	the buffer pointer is NULL

5.3.8 FIO_TP851_DAC_SEQSTOP

This function stops the DAC sequencer. All sequencer channel configurations are still valid after stopping.

For this operation the argument **arg** is not used and should be 0

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;

/*------------------
  Stop DAC Sequencer
  ------------------*/
printf("Stop DAC sequencer ... ");
retval = ioctl(    fd,
                   FIO_TP851_DAC_SEQSTOP,
                   0);
if (retval == OK)
{
     printf("OK\n");
}
else
{
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_NOTBUSY	the DAC sequencer is not started

5.3.9 FIO_TP851_IO_READ

This function reads the current value of digital I/O input.

For this operation the argument **arg** points to a structure named *TP851_IO_BUF*.

typedef struct
{
    unsigned short   value;
} TP851_IO_BUF, *PTP851_IO_BUF;

value

Returns the current digital I/O input value.

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;
TP851_IO_BUF            ioBuf;

/*--------------------
  Read I/O input value
  --------------------*/
printf("Read I/O input value ... ");
retval = ioctl(    fd,
                   FIO_TP851_IO_READ,
                   (int)&ioBuf);

if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
     printf("    I/O input: %04X", ioBuf.value);
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

5.3.10 FIO_TP851_IO_WRITE

This function writes a value to digital I/O output.

For this operation the argument **arg** points to a structure named *TP851_IO_BUF*.

typedef struct
{
    unsigned short   value;
} TP851_IO_BUF, *PTP851_IO_BUF;

value

Specifies the new digital I/O output value.

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;
TP851_IO_BUF            ioBuf;

/* Write 0x1234 to I/O */
ioBuf.value = 0x1234;

/*----------------------
  Write I/O output value
  ----------------------*/
printf("Write I/O output value ... ");
retval = ioctl(    fd,
                   FIO_TP851_IO_WRITE,
                   (int)&ioBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

5.3.11 FIO_TP851_IO_EVENTWAIT

This function waits for an I/O input event.

For this operation the argument **arg** points to a structure named *TP851_IO_EVENTWAIT_BUF*.

```
typedef struct
{
        int             ioLine;
        unsigned long   flags;
        long            timeout;
} TP851_IO_EVENTWAIT_BUF, *PTP851_IO_EVENTWAIT_BUF;
```

ioLine

Specifies the digital I/O line where the event shall occur. Valid values are 0..15.

flags

Specifies the event that shall occur. This is an ORed value of the following flags:

flag	description
TP851_F_HI2LOTRANS	If set, the function will return after a high to low transition occurs.
TP851_F_LO2HITRANS	If set, the function will return after a low to high transition occurs.

At least one flag must be specified.

timeout

Specifies the maximum time the function will wait for the specified event. The time is specified in system ticks.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_IO_EVENTWAIT_BUF      waitBuf;

/*-------------------------------------------------------------
  Wait for a transition on I/O line 12 (max wait 10000 ticks)
  -------------------------------------------------------------*/
waitBuf.ioLine =   12;
waitBuf.flags =    TP851_F_HI2LOTRANS | TP851_F_LO2HITRANS;
waitBuf.timeout =  10000;

…
```

…

```
printf("Wait for an I/O event ... ");
result = ioctl(    fd,
                   FIO_TP851_IO_EVENTWAIT,
                   (int)&waitBuf);
if (result == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_IFLAG	invalid flag specified
S_tp851Drv_ICHAN	invalid I/O line specified
S_tp851Drv_NOFREEEVENT	no free event object available
S_tp851Drv_TIMEOUT	timeout has occurred

5.3.12 FIO_TP851_IO_CONFIG

This function configures digital I/O, the I/O direction is set.

For this operation the argument **arg** points to a structure named *TP851_IO_CONF_BUF*.

typedef struct
{
    unsigned short   direction;
} TP851_IO_CONF_BUF, *PTP851_IO_CONF_BUF;

direction

Specifies the new direction setting for digital I/O. A bit set to 1 enables output, a 0 means that the I/O line is input.

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;
TP851_IO_CONF_BUF       ioConfBuf;

/*-------------------------------------------------------
  Enable line 1,2,8,9 for output, all other lines are input
  -------------------------------------------------------*/
ioConfBuf.direction = (1 << 1) | (1 << 2) | (1 << 8) | (1 << 9);

printf("Set new I/O configuration ... ");
retval = ioctl(    fd,
                   FIO_TP851_IO_CONFIG,
                   (int)&ioConfBuf);

if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

5.3.13 FIO_TP851_IO_DEBCONFIG

This function configures digital I/O debouncer.

For this operation the argument **arg** points to a structure named *TP851_IO_DEBCONF_BUF*.

typedef struct
{
    unsigned short enableMask;
    unsigned short debTime;
} TP851_IO_DEBCONF_BUF, *PTP851_IO_DEBCONF_BUF;

enableMask

Specifies digital I/O lines which shall be used with debouncer. A bit set to 1 enables the debouncer, and a 0 disables the debouncer for the corresponding I/O line.

debTime

Specifies the debounce time. The time is specified in 100ns steps.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_IO_DEBCONF_BUF        ioDebConfBuf;

/*-------------------------------------------------------
/* Enable Debouncer for line 1 and 2 (debounce time 1ms) */
  -------------------------------------------------------*/
ioDebConfBuf.enableMask =   (1 << 1) | (1 << 2);
ioDebConfBuf.debTime =      10000;

printf("Set debouncer  configuration ... ");
retval = ioctl(    fd,
                   FIO_TP851_IO_DEBCONFIG,
                   (int)&ioDebConfBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
} else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

5.3.14 FIO_TP851_CNT_READ

This function reads the value of the counter.

For this operation the argument **arg** points to a structure named *TP851_CNT_READ_BUF*.

```
typedef struct
{
        unsigned long       count;
        unsigned long       state;
} TP851_CNT_READ_BUF, *PTP851_CNT_READ_BUF;
```

count

Returns the current counter value.

state

Returns the counter state. If possible the flags are cleared after read. This is an ORed value of the following flags.

flag	description
TP851_SF_CNTBORROW	Counter borrow bit set (current state)
TP851_SF_CNTCARRY	Counter carry bit set (current state)
TP851_SF_CNTMATCH	Counter match event has occurred since last read.
TP851_SF_CNTSIGN	Counter sign bit (current state)
TP851_SF_CNTDIRECTION	If set, counter direction is upward. If not set, counter direction is downward.
TP851_SF_CNTLATCH	Counter value has been latched.
TP851_SF_CNTLATCHOVERFLOW	Counter latch overflow has occurred.
TP851_SF_CNTSNGLCYC	Counter Single Cycle is active

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;
TP851_CNT_READ_BUF      cntBuf;

/*------------------
  Read counter value
  ------------------*/
printf("Read counter ... ");
retval = ioctl(    fd,
                   FIO_TP851_CNT_READ,
                   (int)&cntBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
     printf("    Counter: %ld", cntBuf.counter);
     printf("    State:   %lXh", cntBuf.state);
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

5.3.15 FIO_TP851_CNT_MATCHWAIT

This function waits for a counter match event.

For this operation the argument **arg** points to a structure named *TP851_CNT_WAIT_BUF*.

typedef struct
{
    long   timeout;
} TP851_CNT_WAIT_BUF, *PTP851_CNT_WAIT_BUF;

timeout

Specifies the maximum time the function will wait for the counter match event. The time is specified in system ticks.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_CNT_WAIT_BUF          waitBuf;

/*-----------------------------------------------------
  Wait for counter match event (max wait 10000 ticks)
  -----------------------------------------------------*/
waitBuf.timeout =  10000;

printf("Wait for counter match event ... ");
retval = ioctl(    fd,
                   FIO_TP851_CNT_MATCHWAIT,
                   (int)&waitBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_NOFREEEVENT	no free event object available
S_tp851Drv_TIMEOUT	timeout has occurred

5.3.16 FIO_TP851_CNT_CTRLWAIT

This function waits for counter control event. The event to wait is chosen with ioctl() function *FIO_TP851_CNT_CONFIG* specifying the parameter *controlMode*.

For this operation the argument **arg** points to a structure named *TP851_CNT_WAIT_BUF*.

```
typedef struct
{
      long                    timeout;
} TP851_CNT_WAIT_BUF, *PTP851_CNT_WAIT_BUF;
```

timeout

Specifies the maximum time the function will wait for the counter control event. The time is specified in system ticks.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_CNT_WAIT_BUF          waitBuf;

/*-------------------------------------------------------
/* Wait for counter control event (max wait 10000 ticks) */
  -------------------------------------------------------*/
waitBuf.timeout =  10000;

printf("Wait for counter control event ... ");
result = ioctl(    fd,
                   FIO_TP851_CNT_CTRLWAIT,
                   (int)&waitBuf);
if (retval == OK)
{
     /* function succeeded */
     retval ("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_NOFREEEVENT	no free event object available
S_tp851Drv_TIMEOUT	timeout has occurred

5.3.17 FIO_TP851_CNT_CONFIG

This function configures the counter.

For this operation the argument **arg** points to a structure named *TP851_CNT_CONFIG_BUF*.

```
typedef struct
{
        unsigned long       inputMode;
        int                 clockDivider;
        unsigned long       countMode;
        unsigned long       controlMode;
        unsigned long       invFlags;
} TP851_CNT_CONFIG_BUF, *PTP851_CNT_CONFIG_BUF;
```

inputMode

Specifies the counter input mode. The following modes are defined and valid:

flag	**description**
TP851_M_CNTIN_DISABLE	Counter disabled
TP851_M_CNTIN_TIMERUP	Timer Mode Up
TP851_M_CNTIN_TIMERDOWN	Timer Mode Down
TP851_M_CNTIN_DIRCOUNT	Direction Count
TP851_M_CNTIN_UPDOWNCOUNT	Up/Down Count
TP851_M_CNTIN_QUAD1X	Quadrature Count 1x
TP851_M_CNTIN_QUAD2X	Quadrature Count 2x
TP851_M_CNTIN_QUAD3X	Quadrature Count 4x

clockDivider

Specifies clock divider for Timer Mode. Allowed clock divider values are:

clock divider value	**clock input frequency**
1	40 MHz
2	20 MHz
4	10 MHz
8	5 MHz

countMode

Specifies the count mode. The following modes are defined and valid:

flag	**description**
TP851_M_CNT_CYCLE	Cycling Counter
TP851_M_CNT_DIVN	Divide-by-N
TP851_M_CNT_SINGLE	Single Cycle

controlMode

Specifies the counter control mode. These events can generate counter control events. The following modes are defined and valid:

flag	description
TP851_M_CNTCTRL_NONE	No Control Mode
TP851_M_CNTCTRL_LOAD	Load Mode
TP851_M_CNTCTRL_LATCH	Latch Mode
TP851_M_CNTCTRL_GATE	Gate Mode
TP851_M_CNTCTRL_RESET	Reset Mode

invFlags

Specifies if counter input lines shall be inverted or not. This is an ORed value of the following flags:

flag	description
TP851_F_CNTINVINP2	If set, input line 2 is low active If not set, input line 2 is high active
TP851_F_CNTINVINP3	If set, input line 3 is low active If not set, input line 3 is high active
TP851_F_CNTINVINP4	If set, input line 4 is low active If not set, input line 4 is high active

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_CNT_CONFIG_BUF        cntConfBuf;

/*-------------------------------------------------------
  Setup counter for direction count, clock divider 1,
  cycling count, no control mode and all lines high active
  -------------------------------------------------------*/
cntConfBuf. inputMode =     TP851_M_CNTIN_DIRCOUNT;
cntConfBuf. clockDivider =  1;
cntConfBuf. countMode =     TP851_M_CNT_CYCLE;
cntConfBuf. controlMode =   TP851_M_CNTCTRL_NONE;
cntConfBuf. invFlags =      0;

printf("Set counter configuration ... ");
retval = ioctl(    fd,
                   FIO_TP851_CNT_CONFIG,
                   (int)&cntConfBuf);
```

```
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_IMODE	invalid mode specified
S_tp851Drv_IFLAG	invalid flag specified

5.3.18 FIO_TP851_CNT_RESET

This function resets the counter value to 0x00000000.

For this operation the argument **arg** is not used and should be 0

EXAMPLE

```
#include “tpmc851.h”

int                     fd;
int                     retval;

/*-------------------
  Reset counter value
  -------------------*/
printf("Reset counter value ... ");
retval = ioctl(    fd,
                   FIO_TP851_CNT_RESET,
                   0);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

5.3.19 FIO_TP851_CNT_SETPRELD

This function sets the counter preload value.

For this operation the argument **arg** points to a structure named *TP851_CNT_SETPRELD_BUF*.

```
typedef struct
{
        unsigned long           value;
        unsigned long           flags;
} TP851_CNT_SETPRELD_BUF, *PTP851_CNT_SETPRELD_BUF;
```

value

Specifies the new counter preload value.

flags

Is an ored value of the following flags:

flag	description
TP851_F_IMMPRELOAD	If set, the function will immediately load the preload value into the counter If not set, preload value will be used for the next preload condition.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_CNT_SETPRELD_BUF      cntPrldBuf;

/*----------------------------------------
  Immediately load 0x11223344 into the counter
  and preload register
  ----------------------------------------*/
cntPrldBuf. value       = 0x11223344;
cntPrldBuf.flags        = TP851_F_IMMPRELOAD;

printf("Set preload value ... ");
retval = ioctl(    fd,
                   FIO_TP851_CNT_SETPRELD,
                   (int)&cntPrldBuf);

…
```

```
…

if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```

ERROR CODES

Error code	Description
S_tp851Drv_IFLAG	invalid flag specified

5.3.20 FIO_TP851_CNT_SETMATCH

This function sets the counter match value. If counter and match value are the same a match event occurs. The driver can wait for this event.

For this operation the argument **arg** points to a structure named *TP851_CNT_SETMATCH_BUF*.

typedef struct
{
    unsigned long   value;
} TP851_CNT_SETMATCH_BUF, *PTP851_CNT_SETMATCH_BUF;

value

Specifies the new counter match value.

EXAMPLE

```
#include “tpmc851.h”

int                         fd;
int                         retval;
TP851_CNT_SETMATCH_BUF      cntMatchBuf;

/*--------------------------
  Set match value to 0x10000
  --------------------------*/
cntMatchBuf.value       = 0x10000;

printf("Set counter match value ... ");
retval = ioctl(    fd,
                   FIO_TP851_CNT_SETMATCH,
                   (int)&cntMatchBuf);
if (retval == OK)
{
     /* function succeeded */
     printf("OK\n");
}
else
{
     /* handle the error */
     printf("FAILED (%08Xh)\n", errnoGet());
}
```