The Embedded I/O Company

TEWS TECHNOLOGIES

TPMC118-SW-42

VxWorks Device Driver

6 Channel Motion Control PMC

Version 3.0.x

User Manual

Issue 3.0.0

September 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

TPMC118-SW-42

VxWorks Device Driver

6 Channel Motion Control PMC

Issue	Description	Date
1.0	First Issue	April 2000
1.1	General Revision	November 2003
2.0.0	New Parameters for DeviceCreate(), new files	October 15, 2004
2.0.1	New Address TEWS LLC, general revision	November 25, 2008
3.0.0	API documentation added	September 10, 2010

Table of Contents

1 Introduction

The TPMC118-SW-42 VxWorks device driver software allows the operation of the modules supported by TPMC118 conforming to the VxWorks I/O system specification.

The TPMC118-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 TPMC118-SW-42 device driver supports the following features:

- read digital inputs
- set the DAC outputs
- setup the encoder channels
- read the input value of the encoder channels
- use the synchronous function of the TPMC118
- use digital input interrupts for signalizing semaphores.

The TPMC118-SW-42 supports the modules listed below:

TPMC118-10	6 Channel Motion Control	(PMC)

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

TPMC118 User manual
TPMC118 Engineering Manual

2 Installation

The following files are located on the distribution media:

Directory path 'TPMC118-SW-42':

TPMC118-SW-42-3.0.0.pdf	PDF copy of this manual
TPMC118-SW-42-VXBUS.zip	Zip compressed archive with VxBus driver sources
TPMC118-SW-42-LEGACY.zip	Zip compressed archive with legacy driver sources
ChangeLog.txt	Release history
Release.txt	Release information

The archive TPMC118-SW-42-VXBUS.zip contains the following files and directories:

Directory path './tews/tpmc118':

tpmc118drv.c	TPMC118 device driver source
tpmc118def.h	TPMC118 driver include file
tpmc118.h	TPMC118 include file for driver and application
tpmc118api.c	TPMC118 API file
Makefile	Driver Makefile
40tpmc118.cdf	Component description file for VxWorks development tools
tpmc118.dc	Configuration stub file for direct BSP builds
tpmc118.dr	Configuration stub file for direct BSP builds
include/tvxbHal.h	Hardware dependent interface functions and definitions
apps/tpmc118exa.c	Example application

The archive TPMC118-SW-42-LEGACY.zip contains the following files and directories:

Directory path './tpmc118':

tpmc118drv.c	TPMC118 device driver source
tpmc118def.h	TPMC118 driver include file
tpmc118.h	TPMC118 include file for driver and application
tpmc118pci.c	TPMC118 device driver source for x86 based systems
tpmc118api.c	TPMC118 API file
tpmc118exa.c	Example application
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!)

TEWS TECHNOLOGIES recommends not using the VxBus Driver before VxWorks release 6.6. In previous releases required header files are missing and the support for 3rd-party drivers may not be available.

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 TPMC118-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 TPMC118 device driver is located in the vendor and driver-specific directory *installDir/vxworks-6.x/target/3rdparty/tews/tpmc118.*

At this point the TPMC118 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 processor (CPU) and build tool (TOOL) must be built in the following way:

- Open a VxWorks development shell (e.g. C:\WindRiver\wrenv.exe -p vxworks-6.7)
- Change into the driver installation directory
  installDir/vxworks-6.x/target/3rdparty/tews/tpmc118
- 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\tpmc118
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 TPMC118 driver with the VxWorks development tools (Workbench), the component configuration file *40tpmc118.cdf* must be copied to the directory *installDir/vxworks-6.x/target/config/comps/VxWorks*.

```
C:> cd \WindRiver\vxworks-6.7\target\3rdparty\tews\tpmc118
C:> copy 40tpmc118.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 TPMC118 driver can be included in VxWorks projects by selecting the *"TEWS TPMC118 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 TPMC118 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\tpmc118
C:> copy tpmc118.dc \Windriver\vxworks-6.7\target\config\comps\src\hwif
C:> copy tpmc118.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 VxWorks projects

For including the TPMC118-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

- Extract all files from the archive TPMC118-SW-42-LEGACY.zip to your project directory.
- 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 tpmc118 directory can be selected.
- Now the driver is included in the project and will be built with the project.

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

2.3.2 Special installation for Intel x86 based targets

The TPMC118 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 TPMC118 PCI memory spaces prior the MMU initialization (*usrMmuInit()*) is done.

The C source file **tpmc118pci.c** contains the function *tpmc118PciInit().* This routine finds out all TPMC118 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 *tpmc118PciInit()* is at the end of the function *sysHwInit()* in **sysLib.c** (it can be opened from the project *Files* window):

```
tpmc118PciInit();
```

Be sure that the function is called prior to MMU initialization or otherwise the TPMC118 PCI spaces remain unmapped and an access fault occurs during driver initialization.

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

2.3.3 BSP dependent adjustments

The driver includes a file called *include/tdhal.h* which contains functions and definitions for BSP adaptation. It may be necessary to modify them for BSP specific settings. Most settings can be made automatically by conditional compilation set by the BSP header files, but some settings must be configured manually. There are two ways of modification, first you can change the *include/tdhal.h* and define the corresponding definition and its value, or you can do it using the command line option *–D*.

There are 3 offset definitions (*USERDEFINED_MEM_OFFSET*, *USERDEFINED_IO_OFFSET*, and *USERDEFINED_LEV2VEC*) that must be configured if a corresponding warning message appears during compilation. These definitions always need values. Definition values can be assigned by command line option *-D<definition>=<value>*.

definition	**description**
USERDEFINED_MEM_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI memory space access
USERDEFINED_IO_OFFSET	The value of this definition must be set to the offset between CPU-Bus and PCI-Bus Address for PCI I/O space access
USERDEFINED_LEV2VEC	The value of this definition must be set to the difference of the interrupt vector (used to connect the ISR) and the interrupt level (stored to the PCI header)

Another definition allows a simple adaptation for BSPs that utilize a *pciIntConnect()* function to connect shared (PCI) interrupts. If this function is defined in the used BSP, the definition of *USERDEFINED_SEL_PCIINTCONNECT* should be enabled. The definition by command line option is made by *-D<definition>.*

Please refer to the BSP documentation and header files to get information about the interrupt connection function and the required offset values.

2.4 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	---	6

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 tpmc118Open()

Name

tpmc118Open() – opens a device.

Synopsis

```
TPMC118_DEV tpmc118Open
(
      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 TPMC118 device is named “/tpmc118/0”, the second device is named “/tpmc118/1” and so on.

Example

```
#include “tpmc118.h”

TPMC118_DEV   pDev;

/*
** open file descriptor to device
*/
pDev = tpmc118Open(“/tpmc118/0”);
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 tpmc118Close()

Name

tpmc118Close() – closes a device.

Synopsis

```
int tpmc118Close
(
     TPMC118_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 “tpmc118.h”

TPMC118_DEV   pDev;
int           result;

/*
** close file descriptor to device
*/
result = tpmc118Close(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 tpmc118DigitalInputRead

Name

tpmc118DigitalInputRead – read state of input lines

Synopsis

```
STATUS tpmc118DigitalInputRead
(
    TPMC118_DEV             pDev,
    unsigned char           *pInputBuf
)
```

Description

This function reads the current state of the input lines.

Parameters

pDev

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

pInputBuf

This argument points to a buffer where the value will be returned. Bit 0 of the value corresponds to the digital input line of the first motion controller channel, bit 1 corresponds to the second channel and so on.

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;
unsigned char      in_value;

/*
** read current state of Input lines
*/
result = tpmc118DigitalInputRead(pDev,
                                 &in_value);
if (result == ERROR)
{
     /* handle error */
}
else
{
     printf(“value: 0x%02X\n”, in_value);
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EINVAL	A NULL pointer is referenced for an input value
EBADF	The device handle is invalid

3.2.2 tpmc118DacWrite

Name

tpmc118DacWrite – write output value to DAC channel

Synopsis

```
STATUS tpmc118DacWrite
(
        TPMC118_DEV             pDev,
        int                     Channel,
        unsigned short          DacValue
)
```

Description

This function writes a new output value to a specific DAC channel.

Parameters

pDev

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

Channel

This parameter specifies the DAC channel to be used. Allowed channel numbers are 1 to 6.

DacValue

This argument specifies the new DAC output value.

Data Value	**Analog Output Voltage**	
0x7FFF	Full Scale – 1 LSB	9.9997V
0x4000	¾ Scale	5V
0x0001	Midscale + 1 LSB	0.0003V
0x0000	Midscale	0V
0xFFFF	Midscale – 1LSB	-0.0003V
0xC000	¼ Scale	-5V
0x8000	Negative Full Scale	-10V

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;

/*
** Set channel 3 to 0x6000 (+ 7.500 Volts)
*/
result = tpmc118DacWrite(    pDev,
                             3,
                             0x6000);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc118Drv_ICHAN	Invalid channel specified.

3.2.3 tpmc118SetClockMode

Name

tpmc118SetClockMode – Change encoder clock

Synopsis

```
STATUS tpmc118SetClockMode
(
    TPMC118_DEV         pDev,
    int                 Channel,
    unsigned long       ClockMode
)
```

Description

This function changes the encoder clock generation.

Parameters

pDev

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

Channel

This parameter specifies the encoder channel to be used. Allowed channel numbers are 1 to 6.

ClockMode

This argument specifies the new encoder clock mode. Possible values are:

Value	Description
TPMC118_RES_OFF	disable Channel
TPMC118_RES_X1	1x – single
TPMC118_RES_X2	2x – double
TPMC118_RES_X4	4x – quadruple

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Set the encoder clock generation of channel 3 to x4
  -------------------------------------------------------*/
result = tpmc118SetClockMode(    pDev,
                                 3,
                                 TPMC118_RES_X4);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc118Drv_ICHAN	Invalid channel specified.
S_tpmc118Drv_IRES	Invalid clock mode specified.

3.2.4 tpmc118SetReferenceMode

Name

tpmc118SetReferenceMode – Change encoder reference mode

Synopsis

```
STATUS tpmc118SetReferenceMode
(
    TPMC118_DEV         pDev,
    int                 Channel,
    unsigned long       ReferenceMode
)
```

Description

This function changes the encoder reference mode.

Parameters

pDev

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

Channel

This parameter specifies the encoder channel to be used. Allowed channel numbers are 1 to 6.

ReferenceMode

This argument specifies the new encoder reference mode. Possible values are:

Value	Description
TPMC118_NONEREF_MODE	None Reference Mode
TPMC118_REF_MODE	Reference Mode
TPMC118_AUTOREF_MODE	Auto Reference Mode
TPMC118_INDEX_MODE	Index Mode

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Set the encoder of channel 3 into index mode
  -------------------------------------------------------*/
result = tpmc118SetReferenceMode(    pDev,
                                     3,
                                     TPMC118_INDEX_MODE);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc118Drv_ICHAN	Invalid channel specified.
S_tpmc118Drv_IMODE	Invalid reference mode specified.

3.2.5 tpmc118SetPreloadValue

Name

tpmc118SetPreloadValue – Set encoder preload value

Synopsis

```
STATUS tpmc118SetPreloadValue
(
    TPMC118_DEV         pDev,
    int                 Channel,
    unsigned long       PreloadValue,
    unsigned long       Flags
)
```

Description

This function sets the encoder preload value.

Parameters

pDev

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

Channel

This parameter specifies the encoder channel to be used. Allowed channel numbers are 1 to 6.

PreloadValue

This argument specifies the new preload value which will be loaded to the encoder counter register, if the mode dependent event occurs.

Flags

This argument specifies if the preload value shall be loaded immediately or not. This will be done if the flag *TPMC118_IMM_PRLD* is set. Otherwise the value will be first loaded if the mode dependent event occurs.

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  * Set the encoder preload value of channel 3 to 0
    and make an immediately load
  -------------------------------------------------------*/
result = tpmc118SetPreloadValue( pDev,
                                 3,
                                 0,
                                 TPMC118_IMM_PRLD);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc118Drv_ICHAN	Invalid channel specified.

3.2.6 tpmc118ReadEncoderValue

Name

tpmc118ReadEncoderValue – Read value of encoder channel

Synopsis

```
STATUS tpmc118ReadEncoderValue
(
    TPMC118_DEV         pDev,
    int                 Channel,
    unsigned long       *pEncoderValue
)
```

Description

This function reads the current encoder value of the specified channel.

Parameters

pDev

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

Channel

This parameter specifies the encoder channel to be used. Allowed channel numbers are 1 to 6.

pEncoderValue

This parameter points to a buffer where the current value of the encoder register will be returned.

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;
unsigned long      EncoderValue;

/*-------------------------------------------------------
  Read the encoder value of channel 3
  -------------------------------------------------------*/
result = tpmc118ReadEncoderValue(    pDev,
                                     3,
                                     &EncoderValue);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
     printf(“Encoder Value: 0x%08X\n”, EncoderValue);
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc118Drv_ICHAN	Invalid channel specified.

3.2.7 tpmc118EnableSynchronization

Name

tpmc118EnableSynchronization – Define synchronized channels

Synopsis

```
STATUS tpmc118EnableSynchronization
(
    TPMC118_DEV         pDev,
    unsigned long       Channels
)
```

Description

This function enables synchronous read for the specified channels.

Parameters

pDev

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

Channels

This argument specifies which channels shall be synchronously read. Each set bit specifies a channel. If one channel is read, all others must be read too. Now new encoder values can be read from any of the synchronized channels.

Bit	Channel
0	1
1	2
2	3
3	4
4	5
5	6

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Synchronize Channel 1, 3 and 5
  -------------------------------------------------------*/
result = tpmc118EnableSynchronization(     pDev,
                                           (1 << 4) | (1 << 2) | (1 << 0));
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid

3.2.8 tpmc118DisableSynchronization

Name

tpmc118DisableSynchronization – Disable synchronized read

Synopsis

```
STATUS tpmc118DisableSynchronization
(
    TPMC118_DEV                pDev
)
```

Description

This function disables synchronous read on different channels.

Parameters

pDev

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

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;

/*-------------------------------------------------------
  Disable synchronization
  -------------------------------------------------------*/
result = tpmc118DisableSynchronization(pDev);
if (result == ERROR)
{
    /* handle error */
}
else
{
    /* function succeeded */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid

3.2.9 tpmc118ReadSynchronizedEncoderValues

Name

tpmc118ReadSynchronizedEncoderValues – Read value of encoder channel

Synopsis

```
STATUS tpmc118ReadSynchronizedEncoderValues
(
    TPMC118_DEV             pDev,
    unsigned long           pEncoderValueArray[]
)
```

Description

This function reads the values of the synchronized channels.

Parameters

pDev

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

pEncoderValueArray

This parameter points to an array of buffers where the encoder values will be returned. The array has six elements, but only the elements for the enabled channels will return valid values. The array with index 0 returns the value of channel 1, the array element with index 1 returns the value of channel 2 and so on.

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;
int                i;
unsigned long      EncoderValueArray[6];

/*-------------------------------------------------------
  Read synchronized channels and display all values
  -------------------------------------------------------*/
result = tpmc118ReadSynchronizedEncoderValues(   pDev,
                                                 3,
                                                 EncoderValueArray
                                               );
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
     for(i = 0; i < 6; i++)
     {
          printf(“Channel %d: %d\n”, i + 1, EncoderValueArray [i]);
     }
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid

3.2.10 tpmc118RegisterEvent

Name

tpmc118RegisterEvent – Set semaphore on input event

Synopsis

```
STATUS tpmc118RegisterEvent
(
    TPMC118_DEV         pDev,
    int                 Channel,
    unsigned long       Flags,
    SEM_ID              Sema
)
```

Description

This function gives a semaphore ID for a specified digital input interrupt. This semaphore will be signalized if the specified interrupt occurs.

Parameters

pDev

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

Channel

This parameter specifies the channel to be used. Allowed channel numbers are 1 to 6.

Flags

This argument specifies the kind of the event. Only one of the following flags may be set:

Value	Description
TPMC118_HIGHTRANS	Generate interrupt on high transition
TPMC118_LOWTRANS	Generate interrupt on low transition
TPMC118_NOTRANS	Do not generate interrupts on any transition. Semaphore ID is no longer valid.

Sema

This argument specifies the semaphore ID of the user created semaphore. The interrupt function of the driver will signalize to this semaphore if the specified event occurs.

Example

```
#include “tpmc118.h”

TPMC118_DEV        pDev;
STATUS             result;
SEM_ID             Sema;

/* create semaphore */
Sema = semBCreate(SEM_Q_FIFO, SEM_EMPTY);

/*------------------------------------------------------
  Wait for a high transition on the digital input of
  channel 4
  ------------------------------------------------------*/
result = tpmc118RegisterEvent(   pDev,
                                 4,
                                 TPMC118_HIGHTRANS,
                                 Sema);
if (result == ERROR)
{
     /* handle error */
}
else
{
     /* function succeeded */
     /* Wait for event, or 1000 Ticks */
     retval = semTake(Sema, 1000);

     /* unregister the semaphore again */
     result = tpmc118RegisterEvent(  pDev,
                                     4,
                                     TPMC118_NOTRANS,
                                     Sema);
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and 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 set code described below. Function specific error codes will be described with the function.

Error code	Description
EBADF	The device handle is invalid
S_tpmc118Drv_ICHAN	Invalid channel specified.

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 TPMC118 driver. For the VxBus-enabled TPMC118 driver, the driver will be installed automatically in the I/O system and devices will be created as needed for detected modules.

4.1 tpmc118Drv()

NAME

tpmc118Drv() - installs the TPMC118 driver in the I/O system

SYNOPSIS

#include “tpmc118.h”

STATUS tpmc118Drv(void)

DESCRIPTION

This function searches for devices on the PCI bus, installs the TPMC118 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 "tpmc118.h”

/*-------------------
  Initialize Driver
  -------------------*/
status = tpmc118Drv();
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()*.

ENXIO	There was no module found supported by this driver

SEE ALSO

VxWorks Programmer's Guide: I/O System

4.2 tpmc118DevCreate()

NAME

tpmc118DevCreate() – Add a TPMC118 device to the VxWorks system

SYNOPSIS

```
#include “tpmc118.h”

STATUS tpmc118DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

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.

devIdx

This index number specifies the device to add to the system.
If modules of the same type are installed, the index numbers, starting with 0, will be assigned in the order the VxWorks *pciFindDevice()* function will find the devices.

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter is unused and should be set to *NULL*.

EXAMPLE

```
#include "tpmc118.h”

STATUS            result;

/*-----------------------------------------------------
  Create the device "/tpmc118/0" for the first device
  -----------------------------------------------------*/
result = tpmc118DevCreate(   "/tpmc118/0",
                             0,
                             0,
                             NULL);
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

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

S_ioLib_NO_DRIVER	Driver has not been installed
ENXIO	Specified device number is not available

SEE ALSO

VxWorks Programmer’s Guide: I/O System

4.3 tpmc118PciInit()

NAME

tpmc118PciInit() – Generic PCI device initialization

SYNOPSIS

void tpmc118PciInit()

DESCRIPTION

This function is required only for Intel x86 VxWorks platforms. The purpose is to setup the MMU mapping for all required TPMC118 PCI spaces (base address register) and to enable the TPMC118 device for access.

The global variable *tpmc118Status* 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 successfully completed. The value of tpmc118Status is equal to the number of mapped PCI spaces
0	No TPMC118 device found
< 0	Initialization failed. The value of (tpmc118Status & 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 tpmc118Init();

…

tpmc118PciInit();

…
```

4.4 tpmc118Init()

NAME

tpmc118Init() – initialize TPMC118 driver and devices

SYNOPSIS

#include “tpmc118.h”

STATUS tpmc118Init(void)

DESCRIPTION

This function is used by the TPMC118 example application to install the driver and to add all available devices to the VxWorks system.

See also 3.1.1 tpmc118Open() for the device naming convention for legacy devices.

After calling this function it is not necessary to call tpmc118Drv() and tpmc118DevCreate() explicitly.

EXAMPLE

```
#include "tpmc118.h”

STATUS    result;


result = tpmc118Init();

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

RETURNS

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

ERROR CODES

Error codes are only set by system functions. The error codes are stored in *errno* and can be read with the function *errnoGet()*.

See 4.1 and 4.2 for a description of possible error codes.

5 Basic I/O Functions

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 TPMC118 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 tpmc118DevCreate() must be used

flags

Not used

mode

Not used

EXAMPLE

```
int fd;

/*----------------------------------------
  Open the device named "/tpmc118/0" for I/O
  ----------------------------------------*/
fd = open("/tpmc118/0", 0, 0);
if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

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

ERROR CODES

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

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

SEE ALSO

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

5.2 close()

NAME

close() – close a device or file

SYNOPSIS

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

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

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

EXAMPLE

```
int  fd;
int  retval;

/*----------------
  close the device
  ----------------*/
retval = close(fd);
if (retval == ERROR)
{
     /* Handle error */
}
```

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()*.

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

SEE ALSO

ioLib, basic I/O routine - close()

5.3 ioctl()

NAME

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

SYNOPSIS

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

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls 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_TPMC118_INPUTREAD	Read Digital input Lines
FIO_TPMC118_DACWRITE	Write Analog Output Voltage
FIO_TPMC118_SETRES	Change encoder clock
FIO_TPMC118_SETMODE	Change encoder reference mode
FIO_TPMC118_SETPRELD	Set encoder preload value
FIO_TPMC118_READENC	Read value of encoder channel
FIO_TPMC118_SYNCON	Define synchronized channels
FIO_TPMC118_SYNCOFF	Disable synchronized read
FIO_TPMC118_SYNCREAD	Read synchronized channels
FIO_TPMC118_SETINTR	Set semaphore on interrupt

arg

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

RETURNS

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

ERROR CODES

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

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

Error code	Description
S_tp118Drv_ICMD	Unknown ioctl() command specified

SEE ALSO

ioLib, basic I/O routine - ioctl()

5.3.1 FIO_TPMC118_INPUTREAD

This I/O control function reads the current digital input value. The function dependent argument **arg** points to an *unsigned char* buffer. Bit 0 corresponds to the first input line, bit 1 corresponds to the second input line and so on.

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
unsigned char DigitalInput;

/*-------------------------------------------------------
  Read the current digital input value
  -------------------------------------------------------*/
retval = ioctl(    fd,
                   FIO_TPMC118_INPUTREAD,
                   (int)&DigitalInput);
if (retval == ERROR)
{
     /* Error reading digital input */
}
else
{
     printf(“Current Input State: 0x%02X\n”, DigitalInput);
}
```

5.3.2 FIO_TPMC118_DACWRITE

This I/O control function writes a new output value to a DAC channel. The function dependent argument **arg** points to a *TPMC118_WBUF* data buffer.

```
typedef struct
{
    int             Channel;
    unsigned short  Value;
} TPMC118_WBUF;
```

Channel

This argument specifies the encoder channel. Allowed channels are 1 to 6.

Value

This argument specifies the new analog output value.

Data Value	**Analog Output Voltage**	
0x7FFF	Full Scale – 1 LSB	9.9997V
0x4000	¾ Scale	5V
0x0001	Midscale + 1 LSB	0.0003V
0x0000	Midscale	0V
0xFFFF	Midscale – 1LSB	-0.0003V
0xC000	¼ Scale	-5V
0x8000	Negative Full Scale	-10V

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_WBUF  wBuf;

/*-------------------------------------------------------
  Set channel 3 to 0x6000 (+ 7.500 Volts)
  -------------------------------------------------------*/
wBuf.Channel  = 3;
wBuf.Value    = 0x6000;

retval = ioctl(    fd,
                   FIO_TPMC118_DACWRITE,
                   (int)&wBuf);
if (retval == ERROR)
{
     /* Error writing analog output value */
}
```

ERROR CODES

Error code	Description
S_tpmc118Drv_ICHAN	Invalid channel specified.

5.3.3 FIO_TPMC118_SETRES

This I/O control function changes the encoder clock generation. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *ResMode* which is defined as a structure *TPMC118_SetResMode*.

```
typedef union
{
        TPMC118_SetResMode      ResMode;
        TPMC118_SetPreLd        PreLd;
        TPMC118_EncRead         EncRead;
        TPMC118_SetSync         SetSync;
        TPMC118_ReadSync        ReadSync;
        TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
        unsigned long           Mode;
        int                     Channel;
} TPMC118_SetResMode;
```

Mode

This argument specifies the new encoder clock mode. Possible values are:

Value	Description
TPMC118_RES_OFF	disable Channel
TPMC118_RES_X1	1x – single
TPMC118_RES_X2	2x – double
TPMC118_RES_X4	4x – quadruple

Channel

This argument specifies the encoder channel. Allowed channels are 1 to 6.

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;

/*-------------------------------------------------------
  Set the encoder clock generation of channel 3 to x4
  -------------------------------------------------------*/
io_par.ResMode.Channel  = 3;
io_par.ResMode.Mode     = TPMC118_RES_X4;

retval = ioctl(    fd,
                   FIO_TPMC118_SETRES,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Error changing encoder counter generation */
}
```

ERROR CODES

Error code	Description
S_tpmc118Drv_ICHAN	Invalid channel specified.
S_tpmc118Drv_IRES	Invalid clock mode specified.

5.3.4 FIO_TPMC118_SETMODE

This function changes the encoder reference mode. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *ResMode* which is defined as a structure *TPMC118_SetResMode*.

```
typedef union
{
        TPMC118_SetResMode      ResMode;
        TPMC118_SetPreLd        PreLd;
        TPMC118_EncRead         EncRead;
        TPMC118_SetSync         SetSync;
        TPMC118_ReadSync        ReadSync;
        TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
        unsigned long           Mode;
        int                     Channel;
} TPMC118_SetResMode;
```

Mode

This argument specifies the new encoder reference mode. Possible values are:

Value	Description
TPMC118_NONEREF_MODE	None Reference Mode
TPMC118_REF_MODE	Reference Mode
TPMC118_AUTOREF_MODE	Auto Reference Mode
TPMC118_INDEX_MODE	Index Mode

Channel

This argument specifies the encoder channel. Allowed channels are 1 to 6.

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;

/*-------------------------------------------------------
  Set the encoder of channel 3 into index mode
  -------------------------------------------------------*/
io_par.ResMode.Channel  = 3;
io_par.ResMode.Mode     = TPMC118_INDEX_MODE;

retval = ioctl(    fd,
                   FIO_TPMC118_SETMODE,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Error changing encoder reference mode */
}
```

ERROR CODES

Error code	Description
S_tpmc118Drv_ICHAN	Invalid channel specified.
S_tpmc118Drv_IMODE	Invalid reference mode specified.

5.3.5 FIO_TPMC118_SETPRELD

This function sets the encoder preload value. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *PreLd* which is defined as a structure *TPMC118_SetPreLd*.

```
typedef union
{
        TPMC118_SetResMode      ResMode;
        TPMC118_SetPreLd        PreLd;
        TPMC118_EncRead         EncRead;
        TPMC118_SetSync         SetSync;
        TPMC118_ReadSync        ReadSync;
        TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
        unsigned long           Value;
        int                     Channel;
        unsigned long           Flags;
} TPMC118_SetPreLd;
```

Value

This argument specifies the new preload value which will be loaded to the encoder counter register, if the mode dependent event occurs.

Channel

This argument specifies the encoder channel. Allowed channels are 1 to 6.

Flags

This argument specifies if the preload value shall be loaded immediately or not. This will be done if the flag *TPMC118_IMM_PRLD* is set. Otherwise the value will be first loaded if the mode dependent event occurs.

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;

/*-------------------------------------------------------
  * Set the encoder preload value of channel 3 to 0
  * Set the encoder preload value of channel 4 to 1000
    and make an immediately load
  -------------------------------------------------------*/
io_par.PreLd.Channel    = 3;
io_par.PreLd.Value      = 0;
io_par.PreLd.Flags      = 0;

retval = ioctl(    fd,
                   FIO_TPMC118_SETPRELD,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Error setting encoder preload value */
}

io_par.PreLd.Channel    = 4;
io_par.PreLd.Value      = 1000;
io_par.PreLd.Flags      = TPMC118_IMM_PRLD;

retval = ioctl(    fd,
                   FIO_TPMC118_SETPRELD,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Error setting encoder preload value */
}
```

ERROR CODES

Error code	Description
S_tpmc118Drv_ICHAN	Invalid channel specified.

5.3.6 FIO_TPMC118_READENC

This function reads the actual encoder value of the specified channel. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *EncRead* which is defined as a structure *TPMC118_EncRead*.

```
typedef union
{
        TPMC118_SetResMode      ResMode;
        TPMC118_SetPreLd        PreLd;
        TPMC118_EncRead         EncRead;
        TPMC118_SetSync         SetSync;
        TPMC118_ReadSync        ReadSync;
        TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
        unsigned long           Value;
        int                     Channel;
} TPMC118_EncRead;
```

Value

This parameter returns the current value of the encoder register.

Channel

This argument specifies the encoder channel. Allowed channels are 1 to 6.

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;

/*-------------------------------------------------------
  Read the encoder value of channel 3
  -------------------------------------------------------*/
io_par.EncRead.Channel  = 3;
retval = ioctl(    fd,
                   FIO_TPMC118_READENC,
                   (int)&io_par);
if (retval == OK)
{
     printf(“Encoder Value: %d\n”, io_par.EncRead.Value);
}
else
{
     /* Error reading encoder value */
}
```

ERROR CODES

Error code	Description
S_tpmc118Drv_ICHAN	Invalid channel specified.

5.3.7 FIO_TPMC118_SYNCON

This function enables synchronous read for the specified channels. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *SetSync* which is defined as a structure *TPMC118_SetSync*.

```
typedef union
{
      TPMC118_SetResMode      ResMode;
      TPMC118_SetPreLd        PreLd;
      TPMC118_EncRead         EncRead;
      TPMC118_SetSync         SetSync;
      TPMC118_ReadSync        ReadSync;
      TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
      unsigned long           Channels;
} TPMC118_SetSync;
```

Channel

This argument specifies which channels shall be read synchronously. Each set bit specifies a channel. If one channel is read, all others must be read too. Now new encoder values can be read from any of the synchronized channels.

Bit	Channel
0	1
1	2
2	3
3	4
4	5
5	6

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;

/*-------------------------------------------------------
  Synchronize Channel 1, 3 and 5
  -------------------------------------------------------*/
io_par.SetSync.Channels = (1 << 4) | (1 << 2) | (1 << 0);    /* 0x15 */
retval = ioctl(    fd,
                   FIO_TPMC118_SYNCON,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Error enabling synchronous read */
}
```

5.3.8 FIO_TPMC118_SYNCOFF

This function disables synchronous read on different channels. The function dependent argument **arg** is unused.

EXAMPLE

```
#include “tpmc118.h”

int  fd;
int  retval;

/*-------------------------------------------------------
  Disable synchronization
  -------------------------------------------------------*/
retval = ioctl(    fd,
                   FIO_TPMC118_SYNCOFF,
                   0);
if (retval == ERROR)
{
     /* Error disabling synchronous read */
}
```

5.3.9 FIO_TPMC118_SYNCREAD

This function reads the values of the synchronized channels. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *ReadSync* which is defined as a structure *TPMC118_ReadSync*.

```
typedef union
{
      TPMC118_SetResMode      ResMode;
      TPMC118_SetPreLd        PreLd;
      TPMC118_EncRead         EncRead;
      TPMC118_SetSync         SetSync;
      TPMC118_ReadSync        ReadSync;
      TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
      unsigned long           Value[MAXCHANMOD];
} TPMC118_ReadSync;
```

Value

The encoder values of the synchronized channels are returned in this array. The array has six elements, but only the elements for the enabled channels will return valid values. The array with index 0 returns the value of channel 1, the array element with index 1 returns the value of channel 2 and so on.

EXAMPLE

```
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;
int           i;

/*-------------------------------------------------------
  Read synchronized channels and display all values
  -------------------------------------------------------*/
retval = ioctl(    fd,
                   FIO_TPMC118_SYNCREAD,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Error enabling synchronous read */
}
else
{
     for(i = 0; i < MAXCHANMOD; i++)
     {
          printf(“Channel %d: %d\n”,
                 i + 1,
                 io_par.ReadSync.Channel[i]);
     }
}
```

5.3.10 FIO_TPMC118_SETINTR

This function gives a semaphore ID for a specified digital input interrupt. This semaphore will be signalized if the specified interrupt occurs. The function dependent argument **arg** points to a union *TPMC118_IOBUF*. For this function use the selection *SetIntr* which is defined as a structure *TPMC118_SetIntr*.

```
typedef union
{
    TPMC118_SetResMode      ResMode;
    TPMC118_SetPreLd        PreLd;
    TPMC118_EncRead         EncRead;
    TPMC118_SetSync         SetSync;
    TPMC118_ReadSync        ReadSync;
    TPMC118_SetInt          SetIntr;
} TPMC118_IOBUF;

typedef struct
{
    SEM_ID                  Sema;
    int                     Channel;
    unsigned long           Flags;
} TPMC118_SetIntr;
```

Sema

This argument specifies the semaphore ID of the user created semaphore. The interrupt function of the driver will signalize to this semaphore if the specified event occurs.

Channel

This argument specifies the digital input channel where the event shall be detected. Allowed values are 1 to 6.

Flags

This argument specifies the kind of the event. Only one of the following flags may be set:

Value	Description
TPMC118_HIGHTRANS	Generate interrupt on high transition
TPMC118_LOWTRANS	Generate interrupt on low transition
TPMC118_NOTRANS	Do not generate interrupts on any transition. Semaphore ID is no longer valid.

EXAMPLE

```
#include <semLib.h>
#include “tpmc118.h”

int           fd;
int           retval;
TPMC118_IOBUF io_par;

/* create semaphore */
semID = semBCreate(SEM_Q_FIFO, SEM_EMPTY);

/*------------------------------------------------------
  Wait for a high transition on the digital input of
  channel 4
  ------------------------------------------------------*/
io_par.SetIntr.Channel  = 4;
io_par.SetIntr.Sema     = semID;
io_par.SetIntr.Flags    = TPMC118_HIGHTRANS;
retval = ioctl(    fd,
                   FIO_TPMC118_SETINTR,
                   (int)&io_par);
if (retval == ERROR)
{
     /* Setting semaphore on interrupt */
}

/* Wait for event, or 1000 Ticks */
retval = semTake(semID, 1000);
```