

# TCPS118-SW-95

## QNX - Neutrino Device Driver

8 Channel SSI or Incremental Encoder and Counter Interface,  
RS-422/TTL I/O, 8x 24 V Input, 8x 24 V Output

Version 1.0.x

## User Manual

Issue 1.0.0

May 2025

## TCPS118-SW-95

QNX - Neutrino Device Driver

8 Channel SSI or Incremental Encoder and Counter Interface, RS-422/TTL I/O, 8x 24 V Input, 8x 24 V Output

Supported Modules:  
TCPS118

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

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

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

©2025 by TEWS Technologies GmbH

Issue	Description	Date
1.0.0	First Issue	May 27, 2025

# Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>5</b>
<b>2</b>	<b>INSTALLATION.....</b>	<b>6</b>
	<b>2.1 Building Executables on Native Systems .....</b>	<b>6</b>
	2.1.1 Build the Device Driver.....	6
	2.1.2 Build the API Library .....	6
	2.1.3 Build the Example Application .....	7
	<b>2.2 Building Executables with Momentics IDE (5.0) .....</b>	<b>8</b>
	2.2.1 Build the Device Driver.....	8
	2.2.2 Build the API Library .....	8
	2.2.3 Build the Example Application .....	8
	2.2.4 Integrate the Device Driver Files to a QNX-Image .....	9
	<b>2.3 Building Executables with Momentics IDE (7.0) .....</b>	<b>10</b>
	2.3.1 Build the Device Driver.....	10
	2.3.2 Build the API Library .....	10
	2.3.3 Build the Example Application .....	11
	2.3.4 Integrate the Device Driver Files to a QNX-Image .....	11
	<b>2.4 Start the Driver Process .....</b>	<b>12</b>
<b>3</b>	<b>API DOCUMENTATION .....</b>	<b>13</b>
	<b>3.1 General Functions.....</b>	<b>13</b>
	3.1.1 tcps118Open .....	13
	3.1.2 tcps118Close .....	15
	3.1.3 tcps118GetPciInfo .....	17
	3.1.4 tcps118GetBoardInfo .....	19
	3.1.5 tcps118GetBoardHealth.....	22
	<b>3.2 Device Access Functions.....</b>	<b>24</b>
	3.2.1 tcps118ReadDigitalInput .....	24
	3.2.2 tcps118ConfigDigitalInput .....	26
	3.2.3 tcps118WaitDigitalInput .....	28
	3.2.4 tcps118SetDigitalOutput .....	30
	3.2.5 tcps118TriggerDigitalOutput .....	32
	3.2.6 tcps118GetDigitalOutputStatus.....	34
	3.2.7 tcps118ConfigDigitalOutput .....	37
	3.2.8 tcps118ReadTimer .....	39
	3.2.9 tcps118WaitTimer .....	41
	3.2.10 tcps118StartTimer .....	43
	3.2.11 tcps118StopTimer.....	45
	3.2.12 tcps118ReadSSIValue.....	47
	3.2.13 tcps118ConfigSSI .....	49
	3.2.14 tcps118DisableSSI .....	53
	3.2.15 tcps118GetSSIStatus .....	55
	3.2.16 tcps118ReadCounterValue.....	57
	3.2.17 tcps118ConfigCounter .....	59
	3.2.18 tcps118DisableCounter .....	63
	3.2.19 tcps11SetCounterPreload.....	65
	3.2.20 tcps118LoadCounter .....	67
	3.2.21 tcps118ResetCounter .....	69
	3.2.22 tcps118WaitCounterMatch .....	71
	3.2.23 tcps118WaitCounterControl.....	73
	3.2.24 tcps118ReadMulti .....	75
	3.2.25 tcps118ConfigMulti .....	77

---

<b>4</b>	<b>APPENDIX.....</b>	<b>79</b>
	<b>4.1 Build Error Message String.....</b>	<b>79</b>
	<b>4.2 Timeout Granularity.....</b>	<b>79</b>

# 1 Introduction

The TCSP118-SW-95 QNX-Neutrino device driver allows the operation of the TCPS118 product family on QNX-Neutrino operating systems.

The TCPS118 device driver is basically implemented as a user installable resource manager. The standard file (I/O) functions (open, close and devctl) provide the basic interface for opening and closing a file descriptor and for performing device I/O and control operations.

The TCPS118-SW-95 device driver supports the following features:

- Reading value and status from SSI channels
- Reading value and status from counter channels
- Configure channels in different modes (SSI, SSI Listen-Only and counter mode)
- Read multiple channels
- Get and set Input and Output values
- Configure I/O behavior, like watchdog and debounce functionality
- Support of on-board interval timer
- Wait for supported events

The TCPS118-SW-95 device driver supports the modules listed below:

TCPS118-10R	8 Channel SSI or Incremental Encoder and Counter Interface, RS-422/TTL I/O, 8x 24 V Input, 8x 24 V Output	(Compact PCI Express)
TCPS118-20R	8 Channel Incremental Encoder and Counter Interface, 24 V I/O, 8x 24 V Input, 8x 24 V Output	(Compact PCI Express)

To get more information about the features and use of TCPS118 devices it is recommended to read the manuals listed below.

TCPS118 User Manual
---------------------

## 2 Installation

Following files are located in the directory TCPS118-SW-95 on the distribution media:

TCPS118-SW-95-SRC.tar.gz	GZIP compressed archive with driver source code
TCPS118-SW-95-1.0.0.pdf	This manual in PDF format
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TCPS118-SW-95-SRC.tar.gz contains the following files and directories:

Directory path 'tcps118':

/driver/tcps118.c	Driver source code
/driver/tcps118.h	Definitions and data structures for driver and application
/driver/tcps118def.h	Device driver include
/driver/node.c	Queue management source code
/driver/node.h	Queue management definitions
/api/tcps118.c	API Library
/api/tcps118.h	API Library include file
/api/nto/*	API Library Build path
/example/tcps118.c	Example application
/example/nto/*	Example application Build path

### 2.1 Building Executables on Native Systems

For installation copy the tar-archive into the /usr/src directory and unpack it (e.g. `tar -xvzf TCPS118-SW-95-SRC.tar.gz`). After that the necessary directory structure for the automatic build and the source files are available underneath the new directory called *tcps118*.

**It is absolutely important to extract the TCPS118 tar archive in the /usr/src directory. Otherwise the automatic build with make will fail.**

#### 2.1.1 Build the Device Driver

Change to the /usr/src/tcps118/driver directory

Execute the Makefile:

```
# make install
```

After successful completion the driver binary (tcps118) will be installed in the /bin directory.

#### 2.1.2 Build the API Library

Change to the /usr/src/tcps118/api directory

Execute the Makefile:

```
# make install
```

After successful completion the API Library will be installed and is available for later usage.

---

### 2.1.3 Build the Example Application

Change to the `/usr/src/tcps118/example` directory

Execute the Makefile:

```
# make install
```

After successful completion the example binary (*tcps118exa*) will be installed in the `/bin` directory.

## 2.2 Building Executables with Momentics IDE (5.0)

This chapter gives just a simple description how to build the drivers with the Momentics IDE (5.0), for more detailed information please refer to the appropriate documentation.

For installation unpack the tar-archive into the desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tcps118*.

### 2.2.1 Build the Device Driver

Create a new project ("Makefile Project with Existing Code") in your workspace:

- Select a "Project Name" (e.g. TCPS118)
- Select the path "tcps118\driver" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")

Now the device driver can be built by "Building the Project".

After successful completion the IDE shows a "Binaries"-path containing the built binary of tcps118 device driver. (e.g. "tcps118 – [x86/le]")

### 2.2.2 Build the API Library

Create a new project ("Makefile Project with Existing Code") in your workspace:

- Select a "Project Name" (e.g. TCPS118-API)
- Select the path "tcps118\api" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")

Now the API Library can be built by "Building the Project".

### 2.2.3 Build the Example Application

Create a new project ("Makefile Project with Existing Code") in your workspace:

- Select a "Project Name" (e.g. TCPS118-Example)
- Select the path "tcps118/example" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")
- Copy the TCPS118 API Library binary file (libtcps118api.a) into the local QNX library path

Now the example can be built by "Building the Project".

After successful completion the IDE shows a "Binaries"-path containing the built binary of tcps118 example application. (e.g. "tcps118exa – [x86/le]")

## 2.2.4 Integrate the Device Driver Files to a QNX-Image

To add the device driver file and the example application file to a QNX-Image, just a few steps are necessary.

Copy the desired binary files of the device driver and example project into “sbin” beneath the “install”-path of the target project using the Momentics-IDE.

Add the filenames of the added files into the build-file (e.g. “x86-generic.build”) in “images”. For example the filenames (e.g. tcps118, tcps118exa) can be inserted behind the serial driver names (insert each filename in a separate line).

After a rebuild of the QNX-Image, the driver files will be available on the disk and can be used after booting.

## 2.3 Building Executables with Momentics IDE (7.0)

This chapter gives just a simple description how to build the drivers with the Momentics IDE (7.0), for more detailed information please refer to the appropriate documentation.

For installation unpack the tar-archive into the desired working directory.

After that the necessary directory structure for the automatic build and the source files are available beneath the new directory called *tcps118*.

### 2.3.1 Build the Device Driver

Create a new project ("Makefile Project with Existing Code") in your workspace:

- Select a "Project Name" (e.g. TCPS118)
- Select the path "tcps118\driver" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")

Now we have to specify the name of the driver executable and additional libraries needed for the driver. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:

- NAME = tcps118
- LIBS = pci

Now the device driver can be built by "Building the Project".

After successful completion the IDE shows a "Binaries"-path containing the built binaries of tcps118 device driver of the enabled configurations (e.g. "tcps118 – [x86/le]" and "tcps118 – [x86\_64/le]").

### 2.3.2 Build the API Library

Create a new project ("Makefile Project with Existing Code") in your workspace:

- Select a "Project Name" (e.g. TCPS118-API)
- Select the path "tcps118\api" in the working directory as "Existing Code Location"
- Select the "Toolchain for Indexer Settings" (e.g. "QNX Multi-toolchain")

Now we have to specify the name of the driver API library. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:

- NAME = tcps118api

Now the API Library can be built by "Building the Project".

### 2.3.3 Build the Example Application

Create a new project (“Makefile Project with Existing Code”) in your workspace:

- Select a “Project Name” (e.g. TCPS118-Example)
- Select the path “tcps118\example” in the working directory as “Existing Code Location”
- Select the “Toolchain for Indexer Settings” (e.g. “QNX Multi-toolchain”)
- Copy the TCPS118 API Library binary file (libtcps118api.a) into the local QNX library path

Now we have to specify the name of the driver example executable. Open the projects properties (Alt+Enter), select C/C++ Build→Environment and add the following environment variables and values to the necessary configurations:

- NAME = tcps118exa
- LIBS = tcps118api

Now the example can be built by “Building the Project”.

After successful completion the IDE shows a “Binaries”-path containing the built binaries of tcps118 example application of the enabled configurations. (e.g. “tcps118exa – [x86/le]” and “tcps118exa – [x86\_64/le]”)

### 2.3.4 Integrate the Device Driver Files to a QNX-Image

To add the device driver file and the example application file to a QNX-Image, just a few steps are necessary.

Copy the desired binary files of the device driver and example project into “sbin” beneath the “install”-path of the target project using the Momentics-IDE.

Add the filenames of the added files into the build-file (e.g. “x86-generic.build”) in “images”. For example the filenames (e.g. tcps118, tcps118exa) can be inserted behind the serial driver names (insert each filename in a separate line).

After a rebuild of the QNX-Image, the driver files will be available on the disk and can be used after booting.

## 2.4 Start the Driver Process

To start the TCPS118 device driver, you have to enter the process name with optional parameter from the command shell or in the startup script.

```
tcps118 [-v] &
```

The TCPS118 Resource Manager creates one device for each supported module, and registers the created devices in the Neutrino's pathname space under following names.

```
/dev/tcps118_0  
/dev/tcps118_1  
...  
/dev/tcps118_x
```

The pathname must be used in the application program to open a path to the desired TCPS118 device.

For debugging, you can start the TCPS118 Resource Manager with the `-v` option. Now the Resource Manager will print versatile information about TCPS118 configuration and command execution on the terminal window.

**Make sure that only one instance of the device driver process is started.**

# 3 API Documentation

## 3.1 General Functions

### 3.1.1 tcps118Open

#### NAME

tcps118Open – open a device.

#### SYNOPSIS

```
TCPS118_HANDLE tcps118Open  
(  
    char      *DeviceName  
)
```

#### DESCRIPTION

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

**The tcps118Open function can be called multiple times (e.g. in different tasks).**

#### PARAMETERS

*DeviceName*

This parameter points to a null-terminated string that specifies the name of the device. The first TCPS118 device is named “/dev/tcps118\_0” the second device is named “/dev/tcps118\_1” and so on.

---

## EXAMPLE

```
#include <tcps118api.h>

TCPS118_HANDLE    hdl;

/*
** open the specified device
*/
hdl = tcps118Open("/dev/tcps118_0");
if (hdl == NULL)
{
    /* handle open error */
}
```

## RETURNS

A device handle, 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 tcps118Close

### NAME

tcps118Close – closes a device.

### SYNOPSIS

```
TCPS118_STATUS tcps118Close
(
    TCPS118_HANDLE    hdl
)
```

### DESCRIPTION

This function closes previously opened devices.

### PARAMETERS

*hdl*

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

### EXAMPLE

```
#include <tcps118api.h>

TCPS118_HANDLE    hdl;
TCPS118_STATUS    result;

/*
** close the device
*/
result = tcps118Close(hdl);
if (result != TCPS118_OK)
{
    /* handle close error */
}
```

---

## RETURNS

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.1.3 tcps118GetPciInfo

#### NAME

tcps118GetPciInfo – get PCI information of the module

#### SYNOPSIS

```
TCPS118_STATUS tcps118GetPciInfo
(
    TCPS118_HANDLE          hdl,
    TCPS118_PCIINFO_BUF    *pPciInfoBuf
)
```

#### DESCRIPTION

This function returns PCI information about the module, e.g. PCI location.

#### PARAMETERS

*hdl*

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

*pPciInfoBuf*

This argument is a pointer to the structure TCPS118\_PCIINFO\_BUF that receives information with the PCI identifiers and PCI location.

```
typedef struct
{
    unsigned short    vendorId;
    unsigned short    deviceId;
    unsigned short    subSystemId;
    unsigned short    subSystemVendorId;
    int               pciBusNo;
    int               pciDevNo;
    int               pciFuncNo;
} TCPS118_PCIINFO_BUF;
```

*vendorId*

Returns the PCI vendor ID of the board.

*deviceId*

Returns the PCI device ID of the board.

*subSystemId*

Returns the PCI subsystem ID of the board.

*subSystemVendor*

Returns the PCI subsystem vendor ID of the board.

*pciBusNo*

Returns the PCI bus number.

*pciDevNo*

Returns the PCI device number.

*pciFuncNo*

Returns the PCI function number.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
TCPS118_PCIINFO_BUF pciInfo;

/*
** get module PCI information
*/
result = tcps118GetPciInfo(hdl, &pciInfo);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("PCI-Location: %d:%d:%d.\n",
       pciInfo.pciBusNo, pciInfo.pciDevNo, pciInfo.pciFuncNo);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.1.4 tcps118GetBoardInfo

#### NAME

tcps118GetBoardInfo – get hardware information from the module

#### SYNOPSIS

```
TCPS118_STATUS tcps118GetBoardInfo
(
    TCPS118_HANDLE          hdl,
    TCPS118_BOARDINFO_BUF  *pBoardInfoBuf
)
```

#### DESCRIPTION

This function returns information about the module, e.g. firmware id (version).

#### PARAMETERS

*hdl*

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

*pBoardInfoBuf*

This argument is a pointer to the structure TCPS118\_BOARDINFO\_BUF that receives information about the hardware.

```
typedef struct
{
    unsigned int    firmwareId;
    unsigned int    channelConfig[TCPS118_MAX_CHANS];
} TCPS118_MODULEINFO_BUF;
```

*firmwareId*

Returns the firmware version used on the board. The version is returned in 32 bit word in the following format:

MMmmRRbb	(hex-format)
MM	major version,
mm	minor version,
RR	revision
bb	build version

e.g. Firmware ID 1.2.3.4 will be returned as 0x01020304

### *channelConfig[]*

An array where the detected channel configurations will be returned. The index specifies the assigned channel number. The following channel configurations are defined:

<b>Channel Configuration</b>	<b>Description</b>
TCPS118_FL_RS422TTL	RS-422/TTL-I/O
TCPS118_FL_24V	24V

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
TCPS118_BOARDINFO_BUF boardInfo;
int                 chan;

/*
** get module board information
*/
result = tcps118GetBoardInfo(hdl, &boardInfo);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("Firmware-ID: %02d.%02d.%02d Build: %02d.\n",
       (moduleInfo.firmwareId >> 24) & 0xFF,
       (moduleInfo.firmwareId >> 16) & 0xFF,
       (moduleInfo.firmwareId >> 8) & 0xFF,
       moduleInfo.firmwareId & 0xFF);
printf("Channel Configuration:\n");
for (chan = 0; chan < TCPS118_MAX_CHANS; chan++)
{
    printf("#%d: %s\n", chan,
           (boardInfo.chanConfig[chan] == TCPS118_FL_RS422TTL) ?
           "RS-422/TTL" : "24 V");
}
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.1.5 tcps118GetBoardHealth

#### NAME

tcps118GetBoardHealth – get health information from the module

#### SYNOPSIS

```
TCPS118_STATUS tcps118GetBoardInfo
(
    TCPS118_HANDLE          hdl,
    TCPS118_HEALTH_BUF     *pHealthBuf
)
```

#### DESCRIPTION

This function returns information about the module health, e.g. the on-board temperature of the XADC.

#### PARAMETERS

*hdl*

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

*pHealthBuf*

This argument is a pointer to the structure TCPS118\_HEALTH\_BUF that receives information about the hardware health.

```
typedef struct
{
    unsigned int    powerGood;
    int             temperature;
    unsigned int    sensorAlarm;
} TCPS118_HEALTH_BUF;
```

*powerGood*

This value returns the “Power Good” state. The following values are valid:

Power Good State	Description
TCPS118_FL_POWERGOOD	The board signals “Power Good”
TCPS118_FL_POWERFAILURE	The board signals a failure with the onboard power

### *temperature*

This value returns the on-chip temperature of the XADC on the TCPS118. The returned temperature is scaled to  $1/1000$  °C.

### *sensorAlarm*

This value indicates if internal sensors are out of bounds.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
TCPS118_HEALTH_BUF  healthBuf;

/*
** get module health information
*/
result = tcps118GetBoardHealth(hdl, &healthBuf);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("XADC-temperature: %8.4f °C\n", healthBuf.temperature / 1000.0);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

## 3.2 Device Access Functions

### 3.2.1 tcps118ReadDigitalInput

#### NAME

tcps118ReadDigitalInput – read state of the digital input lines

#### SYNOPSIS

```
TCPS118_STATUS tcps118ReadDigitalInput  
(  
    TCPS118_HANDLE          hdl,  
    unsigned int             *pDigitalInput  
)
```

#### DESCRIPTION

This function returns the current state of the digital input lines.

#### PARAMETERS

*hdl*

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

*pDigitalInput*

This argument is a pointer to an unsigned int value, where the current state of the input lines will be stored to. Bit 0 is assigned to the digital input line of channel 0, bit 1 is assigned to the digital input line of channel 1, and so on.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         digIn;

/*
** get state of the digital input lines
*/
result = tcps118ReadDigitalInput(hdl, &digIn);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("digital inputs: 0x%02X\n", digIn);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

## 3.2.2 tcps118ConfigDigitalInput

### NAME

tcps118ConfigDigitalInput – configure and enable digital input debouncer

### SYNOPSIS

```
TCPS118_STATUS tcps118ConfigDigitalInput
(
    TCPS118_HANDLE          hdl,
    unsigned int            enabelMask,
    unsigned int            timeBase,
    unsigned int            timeVal
)
```

### DESCRIPTION

This function configures the debouncer behavior and enables or disables digital input channels to use the debouncer.

### PARAMETERS

*hdl*

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

*enabelMask*

This parameter specifies a mask, which enables or disables channel to use the debouncer. A set bit will enable and a unset bit will reset the debouncer function on the corresponding input line. Bit 0 is assigned to the digital input line of channel 0, bit 1 is assigned to the digital input line of channel 1, and so on.

*timeBase*

This argument specifies the time base the interval counter work on. The following time bases are defined for this function:

Time Base	Description
TCPS118_FL_TIMEBASE_30NS	The time base will be 30 ns.
TCPS118_FL_TIMEBASE_100NS	The time base will be 100 ns.
TCPS118_FL_TIMEBASE_1US	The time base will be 1 μs.
TCPS118_FL_TIMEBASE_1MS	The time base will be 1 ms.

*timeVal*

This parameter specifies the time a signal must be stable before it will be accepted as a new valid state. The time for the debouncer is calculated in the following way:

$$\langle \text{debouncer time} \rangle = \langle \text{timeBase} \rangle * \langle \text{timeVal} \rangle$$

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         chanMask;

/*
** enable debouncer for digital input line 1 and 2
** the debouncer time shall be 5 µs
*/
chanMask = (1 << 1) | (1 << 2);
result = tcps118ConfigDigitalInput(hdl, chanMask,
                                   TCPS118_FL_TIMEBASE_1US, 5);

if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. the value of the timeBase, an invalid channel mask, ...)

### 3.2.3 tcps118WaitDigitalInput

#### NAME

tcps118WaitDigitalInput – wait for a specified digital input event

#### SYNOPSIS

```
TCPS118_STATUS tcps118WaitDigitalInput
(
    TCPS118_HANDLE          hdl,
    unsigned int            transLowMask,
    unsigned int            transHighMask,
    int                     timeout
)
```

#### DESCRIPTION

This function waits for a specified digital input event. The function will return if at least one of the specified events occurs.

#### PARAMETERS

*hdl*

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

*transLowMask*

This argument specifies a mask where the input lines are specified which will generate an event on (high-to-)low transition. Bit 0 of the value is assigned to the digital input line of channel 0, bit 1 is assigned to the digital input line of channel 1, and so on.

*transHighMask*

This argument specifies a mask where the input lines are specified which will generate an event on (low-to-)high transition. Bit 0 of the value is assigned to the digital input line of channel 0, bit 1 is assigned to the digital input line of channel 1, and so on.

*timeout*

This argument specifies the maximum time to wait for an specified event. The value is specified in ms.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         lowTrMask;
unsigned int         highTrMask;

/*
** Wait for an input event (max. 60 s)
** low transition on input line #0 or #1
** or
** high transition on input line #0 or #2
*/
lowTrMask = (1 << 0) | (1 << 1);
highTrMask = (1 << 0) | (1 << 2);
result = tcps118WaitDigitalInput(hdl, lowTrMask, highTrMask, 60000);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_BUSY	Waiting for interval timer event is already in use
TCPS118_ERR_TIMEOUT	The specified time to wait has passed before the event occurred.
TCPS118_ERR_INVAL	A specified parameter is invalid (e.g. the value of the masks, ...)

## 3.2.4 tcps118SetDigitalOutput

### NAME

tcps118SetDigitalOutput – set the state of the digital output lines

### SYNOPSIS

```
TCPS118_STATUS tcps118SetDigitalOutput
(
    TCPS118_HANDLE          hdl,
    unsigned int            digitalOutput,
    unsigned int            digitalOutputMask
)
```

### DESCRIPTION

This function sets the state of selected and enabled digital output lines.

### PARAMETERS

*hdl*

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

*digitalOutput*

This parameter specifies the new state of the digital output lines. Bit 0 is assigned to the digital output line of channel 0, bit 1 is assigned to the digital output line of channel 1, and so on.

*digitalOutputMask*

This parameter masks the output lines that shall be set- only output lines with a set bit in this mask will be set. Bit 0 is assigned to the digital output line of channel 0, bit 1 is assigned to the digital output line of channel 1, and so on.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** set state of the digital output lines
** set output line #2 and reset line #4
*/
result = tcps118SetDigitalOutput(hdl, 0x04, 0x14);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. the value or mask)

## 3.2.5 tcps118TriggerDigitalOutput

### NAME

tcps118TriggerDigitalOutput – trigger the digital output watchdog timer

### SYNOPSIS

```
TCPS118_STATUS tcps118TriggerDigitalOutput
(
    TCPS118_HANDLE      hdl
)
```

### DESCRIPTION

This function restarts the digital watchdog timer. The watchdog timer will start again.

### PARAMETERS

*hdl*

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

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;

/*
** trigger the digital output watchdog timer
*/
result = tcps118TriggerDigitalOutput (hdl);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.2.6 tcps118GetDigitalOutputStatus

#### NAME

tcps118GetDigitalOutputStatus – reads the digital output status

#### SYNOPSIS

```
TCPS118_STATUS tcps118GetDigitalOutputStatus
(
    TCPS118_HANDLE          hdl,
    unsigned int            *pWatchdogStatus,
    unsigned int            *pDigOutStatus
)
```

#### DESCRIPTION

This function reads the digital output status.

#### PARAMETERS

*hdl*

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

*pWatchdogStatus*

This argument points to an unsigned int value where the status of the digital output watchdog is returned. The returned value is an ORed value of the following flags

Watchdog Status Flags	Description
TCPS118_FL_WD_STD	Watchdog has expired and has disabled all 24 V Digital Output channels. For a more detailed description see the User Manual of the module.
TCPS118_FL_WD_LOCK	Watchdog has expired and has disabled all 24 V Digital Output channels. For a more detailed description see the User Manual of the module.

### *watchdogEnable*

This argument returns the digital output status. The lower eight bits show the “output status indicators”. The other states are flags ORed into the returned value.

Output Status Flag	Description
TCPS118_MA_DOSx	This mask can be used to extract the “output status indicators”.
TCPS118_FL_WRN0_3	“VDD2 Low-Voltage Warning” for channels 0-3.
TCPS118_FL_FLT0_3	Fault indicator for channels 0-3.
TCPS118_FL_WRN4_7	“VDD2 Low-Voltage Warning” for channels 4-7-
TCPS118_FL_FLT4_7	Fault indicator for channels 4-7.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         digOutStat;
unsigned int         wdStat;

/*
** enable all output lines and the watchdog with a time of 100µs
*/
result = tcps118GetDigitalOutputStatus (hdl, &wdStat, &digOutStat);
if (result != TCPS118_OK)
{
    /* handle error */
}
printf("Watchdog Status [STD/LOCK]: %d/%d\n",
      (watchdogStatus & TCPS118_FL_WD_STD) ? 1 : 0,
      (watchdogStatus & TCPS118_FL_WD_LOCK) ? 1 : 0);
printf("Output Status Indicators:  0x%02X\n",
      digOutStatus & TCPS118_MA_DOSx);
printf("WRN/FLT 0-3:                %d/%d\n",
      (digOutStatus & TCPS118_FL_WRN0_3) ? 1 : 0,
      (digOutStatus & TCPS118_FL_FLT0_3) ? 1 : 0);
printf("WRN/FLT 4-7:                %d/%d\n",
      (digOutStatus & TCPS118_FL_WRN4_7) ? 1 : 0,
      (digOutStatus & TCPS118_FL_FLT4_7) ? 1 : 0);
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

## 3.2.7 tcps118ConfigDigitalOutput

### NAME

tcps118ConfigDigitalOutput – configure the behavior of the digital output set the state of the digital output lines

### SYNOPSIS

```
TCPS118_STATUS tcps118ConfigDigitalOutput
(
    TCPS118_HANDLE          hdl,
    unsigned int            outputEnable,
    unsigned int            watchdogEnable,
    unsigned int            watchdogTimeVal
)
```

### DESCRIPTION

This function sets the state of selected and enabled digital output lines.

### PARAMETERS

#### *hdl*

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

#### *outputEnable*

This parameter specifies which digital output should be enabled. The enable is assigned to groups of four digital output lines. The following flags specify the groups which shall be enabled. An unspecified flag will disable the outputs.

Enable Flags	Description
TCPS118_FL_ENA_OUT0_3	Enable digital output lines of channel 0 up to channel 3.
TCPS118_FL_ENA_OUT4_7	Enable digital output lines of channel 4 up to channel 7.

#### *watchdogEnable*

This parameter specifies if the digital output watchdog shall be enabled or not. The following flag can be assigned to enable the watchdog function.

Watchdog Flag	Description
TCPS118_FL_ENA_WATCHDOG	Enable the digital output watchdog function.

#### *watchdogTimeVal*

This parameter specifies the time within the watchdog must be triggered, before the watchdog disables the digital output lines. The time is specified in  $\mu$ s.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** enable all output lines and the watchdog with a time of 100 µs
*/
result = tcps118ConfigDigitalOutput (hdl,
                                     TCPS118_FL_ENA_OUT0_3 | TCPS118_FL_ENA_OUT4_7,
                                     TCPS118_FL_ENA_WATCHDOG,
                                     100);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid

## 3.2.8 tcps118ReadTimer

### NAME

tcps118ReadTimer – read current counter value if the interval timer

### SYNOPSIS

```
TCPS118_STATUS tcps118ReadTimer
(
    TCPS118_HANDLE          hdl,
    unsigned int             *pTimerVal
)
```

### DESCRIPTION

This function returns the current counter value of the interval timer.

### PARAMETERS

*hdl*

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

*pDigitalInput*

This argument is a pointer to an unsigned int value, where the current counter value of the interval timer will be stored to.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         timerCount;

/*
** get count of the interval timer
*/
result = tcps118ReadTimer(hdl, &timerCount);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("interval timer count: %d\n", timerCount);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

## 3.2.9 tcps118WaitTimer

### NAME

tcps118WaitTimer – wait for the next interval timer event

### SYNOPSIS

```
TCPS118_STATUS tcps118WaitTimer
(
    TCPS118_HANDLE    hdl,
    int                timeout
)
```

### DESCRIPTION

This function waits for the next interval timer event.

### PARAMETERS

*hdl*

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

*timeout*

This argument specifies the maximum time to wait for the next event. The value is specified in ms.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE    hdl;
TCPS118_STATUS    result;

/*
** Wait for an interval timer event (max. 500 ms)
*/
result = tcps118WaitTimer(hdl, 500);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_BUSY	Waiting for interval timer event is already in use
TCPS118_ERR_TIMEOUT	The specified time to wait has passed before the event occurred

### 3.2.10 tcps118StartTimer

#### NAME

tcps118StartTimer – setup and start the interval timer

#### SYNOPSIS

```
TCPS118_STATUS tcps118StartTimer
(
    TCPS118_HANDLE          hdl,
    unsigned int            timeBase,
    unsigned int            preloadVal
)
```

#### DESCRIPTION

This function sets up the interval timer and starts it afterwards.

#### PARAMETERS

*hdl*

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

*timeBase*

This argument specifies the time base the interval counter works on. The following time bases are defined for this function:

Time Base	Description
TCPS118_FL_TIMEBASE_100NS	The time base will be 100 ns.
TCPS118_FL_TIMEBASE_1US	The time base will be 1 μs.
TCPS118_FL_TIMEBASE_1MS	The time base will be 1 ms.
TCPS118_FL_TIMEBASE_1S	The time base will be 1 s.

*preloadVal*

This parameter specifies the preload value of the interval timer which defines the time for one timer interval. The time for one timer interval is calculated in the following way:

$$\langle \text{interval} \rangle = \langle \text{timeBase} \rangle * \langle \text{preloadVal} \rangle$$

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** start the interval timer with an interval time of 750 ms
*/
result = tcps118StartTimer(hdl, TCPS118_FL_TIMEBASE_1MS, 750);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVAL	A specified parameter is invalid (e.g. the value of the timeBase)

## 3.2.11 tcps118StopTimer

### NAME

tcps118StopTimer – stops the interval timer

### SYNOPSIS

```
TCPS118_STATUS tcps118StopTimer
(
    TCPS118_HANDLE          hdl
)
```

### DESCRIPTION

This function stops the interval timer.

### PARAMETERS

*hdl*

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

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** stop the interval timer
*/
result = tcps118StopTimer(hdl);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid

### 3.2.12 tcps118ReadSSIValue

#### NAME

tcps118ReadSSIValue – start SSI transfer and read the value

#### SYNOPSIS

```
TCPS118_STATUS tcps118ReadTimer
(
    TCPS118_HANDLE          hdl,
    unsigned int            ssiChan,
    int                     timeout,
    unsigned int            *pSSIVal,
    unsigned int            *pSSIStatus
)
```

#### DESCRIPTION

This function starts an SSI transfer, waits until it completes and returns the value and status of the transfer. This function works for channels which are configured for SSI or SSI Listen Only mode.

#### PARAMETERS

*hdl*

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

*ssiChan*

This argument specifies the channel number of the channel that shall be used.

*timeout*

This argument specifies the maximum time to wait for completion of the SSI transfer. The value is specified in ms.

*pSSIVal*

This argument is a pointer to an unsigned int value, where the SSI value will be stored to.

*pSSIStatus*

This argument is a pointer to an unsigned int value, where the SSI transfer status will be stored to. The following status flags are defined and will be returned in an ORed mask.

SSI Status Flag	Description
TCPS118_FL_SSI_READ_PARITY_ERR	A parity error has been detected in the SSI transfer.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         ssiValue;
unsigned int         ssiStatus;

/*
** get SSI value from channel 3
** - timeout after 1 s
*/
result = tcps118ReadSSIValue(hdl, 3, 1000, &ssiValue, &ssiStatus);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("SSI value: 0x%08x (status: 0x%x)\n", ssiValue, ssiStatus);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVAL	A specified parameter is invalid, e.g. channel number
TCPS118_ERR_BUSY	Waiting for a completion of an active read
TCPS118_ERR_TIMEOUT	The specified time to wait has passed before the read completed
TCPS118_ERR_CONFIG	The channel is not configured as an SSI channel

### 3.2.13 tcps118ConfigSSI

#### NAME

tcps118ConfigSSI – configure a channel as SSI channel

#### SYNOPSIS

```
TCPS118_STATUS tcps118StartTimer
(
    TCPS118_HANDLE          hdl,
    TCPS118_SSI_CONFIG_BUF *pSsiConfigBuf
)
```

#### DESCRIPTION

This function configures the specified channel to work as an SSI or SSI-listen-only channel. The function configures the data transmission parameter.

#### PARAMETERS

*hdl*

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

*pSsiConfigBuf*

This argument is a pointer to the structure TCPS118\_SSI\_CONFIG\_BUF that specifies the parameters necessary to setup the channel for SSI transfers.

```
typedef struct
{
    unsigned int    channel;
    unsigned int    ssiMode;
    unsigned int    ssiClockRate;
    unsigned int    ssiStartMode;
    unsigned int    ssiDelay;
    unsigned int    dataBits;
    unsigned int    zeroBits;
    unsigned int    grayCode;
    unsigned int    parityMode;
} TCPS118_SSI_CONFIG_BUF;
```

*channel*

This argument specifies the channel number which shall be configured. Valid values are 0 up to 7.

*ssiMode*

This argument specifies the SSI mode that shall be configured. Depending on the mode some of the following arguments may not be used.

SSI Mode	Description
TCPS118_FL_SSI_MODE_STANDARD	Configure standard SSI interface mode
TCPS118_FL_SSI_MODE_LISTENONLY	Configure SSI Listen-Only mode

*ssiClockRate*

This argument specifies the SSI clock rate in kHz. Valid clock rates are 1 kHz up to 4095 kHz.

This value is not used and ignored in listen-only mode.

*ssiStartMode*

This argument specifies the SSI start mode. The following values are defined for the different SSI start modes.

SSI Start Mode	Description
TCPS118_FL_SSI_START_MANUAL	The SSI transfer is triggered by a call reading the SSI data value
TCPS118_FL_SSI_START_TIMER	The SSI transfer is triggered by the boards interval timer
TCPS118_FL_SSI_START_DELAYFROMSTART	Configure a Back-to-back transmission, starting after a specified time after the last transmission has been started
TCPS118_FL_SSI_START_DELAYFROMEND	Configure a Back-to-back transmission, starting after a specified time after the last transmission has been completed

*ssiDelay*

This argument specifies the SSI delay if a Back-to-back transmission is configured for the channel. The value is specified in steps of 10µs. The maximum delay is 310 µs.

*dataBits*

This argument specifies the number of databits that will be transferred. Valid numbers of databits are 1 up to 32.

*zeroBits*

This argument specifies the number of zerobits that will be transferred. Valid numbers of zerobits are 0 up to 7.

*grayCode*

This argument specifies if the received data shall be interpreted as graycode or simple binary. A TRUE (1) specifies to use graycode, a FALSE (0) specifies the use of binary values.

### *parityMode*

This argument specifies if and how a parity bit will be built and checked. The following values are defined and can be used and ORed to configure a mixture of the modes.

<b>SSI Parity Mode</b>	<b>Description</b>
TCPS118_FL_SSI_PAR_DISABLE	Do not build a parity bit
TCPS118_FL_SSI_PAR_ONDATABITS	Databits are used to build the parity flag
TCPS118_FL_SSI_PAR_ONZEROBITS	Zerobits are used to build the parity flag
TCPS118_FL_SSI_PAR_EVEN	Build an even parity TCPS118_FL_SSI_PAR_ODD cannot be specified at the same time
TCPS118_FL_SSI_PAR_ODD	Build an odd parity TCPS118_FL_SSI_PAR_EVEN cannot be specified at the same time

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
TCPS118_SSI_CONFIG_BUF ssiConf;

/*
** configure channel 2 for SSI manually triggered at a clock rate of 100
kHz
** 24 data and 2 zerobits
** build a parity over data and zerobits
*/
ssiConf.channel = 2;
ssiConf.ssiMode = TCPS118_FL_SSI_MODE_STANDARD;
ssiConf.ssiClockRate = 100; /* 100 kHz */
ssiConf.ssiStartMode = TCPS118_FL_SSI_START_MANUAL;
/* ssiConf.ssiDelay = 0; not used */
ssiConf.dataBits = 24;
ssiConf.zeroBits = 2;
ssiConf.grayCode = FALSE; /* binary coded */
ssiConf.parityMode = TCPS118_FL_SSI_PAR_ONDATABITS |
                    TCPS118_FL_SSI_PAR_ONZEROBITS |
                    TCPS118_FL_SSI_PAR_ODD; /* odd parity on db and zb */

result = tcps118ConfigSSI(hdl, &ssiConf);
if (result != TCPS118_OK)
{
    /* handle error */
}
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid, e.g. number of databits
TCPS118_ERR_BUSY	The channel is busy and already configured, it must be disabled first

## 3.2.14 tcps118DisableSSI

### NAME

tcps118DisableSSI – disable channel and leave SSI mode

### SYNOPSIS

```
TCPS118_STATUS tcps118DisableSSI
(
    TCPS118_HANDLE          hdl,
    unsigned int            ssiChan
)
```

### DESCRIPTION

This function stops the SSI mode for the specified channel and disables it.

### PARAMETERS

*hdl*

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

*ssiChan*

This argument specifies the number of the channel that shall be disabled.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** disable SSI channel 3
*/
result = tcps118DisableSSI(hdl, 3);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. the specified channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a SSI channel

### 3.2.15 tcps118GetSSIStatus

#### NAME

tcps118GetSSIStatus – reads the SSI status

#### SYNOPSIS

```
TCPS118_STATUS tcps118GetSSIStatus
(
    TCPS118_HANDLE          hdl,
    unsigned int            ssiChan,
    unsigned int            *pSSIChanStatus
)
```

#### DESCRIPTION

This function simply returns the current value of the “SSI Mode Status Register”.

#### PARAMETERS

*hdl*

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

*ssiChan*

This argument specifies the number of the channel that shall be disabled.

*pSSIChanStatus*

This argument points to an unsigned int value where the value of the “SSI Mode Status Register” will be returned. A full description of the register can be found in the User Manual of the hardware.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         ssiStat;

/*
** read the SSI status of channel 3
*/
result = tcps118GetSSIStatus (hdl, 3, &ssiStat);
if (result != TCPS118_OK)
{
    /* handle error */
}
printf("SSI Mode Status Register: 0x%08d\n", ssiStat);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVAL	A specified parameter is invalid (e.g. the specified channel number)

## 3.2.16 tcps118ReadCounterValue

### NAME

tcps118ReadCounterValue – read the value and status of a counter

### SYNOPSIS

```
TCPS118_STATUS tcps118ReadCounterValue
(
    TCPS118_HANDLE          hdl,
    unsigned int            counterChan,
    unsigned int            *pCounterVal,
    unsigned int            *pCounterStatus
)
```

### DESCRIPTION

This function reads the current value and status of the specified counter channel. This function works for channels which are configured in counter mode.

### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the channel that shall be used.

*pCounterVal*

This argument is a pointer to an unsigned int value, where the counter value will be stored to.

*pCounterStatus*

This argument is a pointer to an unsigned int value, where the counter status will be stored to. For a more detailed description of the status flags have a look at the User Manual of the hardware.

The following status flags are defined and will be returned in an ORed mask.

Counter Status Flag	Description
TCPS118_FL_CNT_READ_BOR	A counter value underflow occurred
TCPS118_FL_CNT_READ_CAR	A counter value overflow occurred
TCPS118_FL_CNT_READ_MAT	A counter match has occurred
TCPS118_FL_CNT_READ_SGN	This flag shows the last sign change of counter value (BORROW or CARRY (reset/set))
TCPS118_FL_CNT_READ_DIR	This flag shows the current direction of the counter (set: counts up / reset: counts down)

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         countValue;
unsigned int         countStatus;

/*
** read the current counter value from channel 1
*/
result = tcps118ReadSSIValue(hdl, 1, &countValue, &countStatus);
if (result != TCPS118_OK)
{
    /* handle error */
}

printf("Counter value: 0x%08x (status: 0x%x)\n", countValue, countStatus);
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVAL	A specified parameter is invalid, e.g. channel number
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

## 3.2.17 tcps118ConfigCounter

### NAME

tcps118ConfigCounter – configure a channel as counter channel

### SYNOPSIS

```
TCPS118_STATUS tcps118ConfigCounter
(
    TCPS118_HANDLE                hdl,
    TCPS118_COUNT_CONFIG_BUF     *pCounterConfigBuf
)
```

### DESCRIPTION

This function configures the specified channel to work in counter mode. The function configures the data transmission parameter.

### PARAMETERS

*hdl*

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

*pCountConfigBuf*

This argument is a pointer to the structure TCPS118\_COUNT\_CONFIG\_BUF that specifies the parameters necessary to setup the channel for SSI transfers.

```
typedef struct
{
    unsigned int    channel;
    unsigned int    countMode;
    unsigned int    preloadValue;
    unsigned int    intClockRate;
    unsigned int    specialMode;
    unsigned int    indexMode;
    unsigned int    polA;
    unsigned int    polB;
    unsigned int    poll;
} TCPS118_COUNT_CONFIG_BUF;
```

*channel*

This argument specifies the channel number which shall be configured. Valid value are 0 up to 7.

*countMode*

This argument specifies the counter mode that shall be configured. Depending on the mode some of the following arguments may not be used.

SSI Mode	Description
TCPS118_FL_CNT_MODE_TIMERUP	Configure "Timer Mode Up"
TCPS118_FL_CNT_MODE_TIMERDOWN	Configure "Timer Mode Down"
TCPS118_FL_CNT_MODE_DIRECTION	Configure "Direction Count"
TCPS118_FL_CNT_MODE_UPDOWN	Configure "Up/Down Count"
TCPS118_FL_CNT_MODE_QUAD1X	Configure "Quadrature Count 1x"
TCPS118_FL_CNT_MODE_QUAD2X	Configure "Quadrature Count 2x"
TCPS118_FL_CNT_MODE_QUAD4X	Configure "Quadrature Count 4x"

*preloadValue*

This argument specifies the preload value in counter mode.

*intClockRate*

This argument specifies the internal base clock rate. The following values are defined for the internal base clock rate.

Value	Description
1	Internal Base Clock: 1 MHz
2	Internal Base Clock: 2 MHz
4	Internal Base Clock: 4 MHz
5	Internal Base Clock: 5 MHz
10	Internal Base Clock: 10 MHz
20	Internal Base Clock: 20 MHz
50	Internal Base Clock: 50 MHz
100	Internal Base Clock: 100 MHz

*specialMode*

This argument specifies the special count mode. The following values are defined for the special count modes.

SSI Start Mode	Description
TCPS118_FL_CNT_SPECMODE_NONE	No special count mode / cycling count mode
TCPS118_FL_CNT_SPECMODE_DIVBYN	Single Cycle
TCPS118_FL_CNT_SPECMODE_SNGLCYC	Divide-by-N

*indexMode*

This argument specifies the index control mode. The following values are defined for the index control mode.

<b>Index Control Mode</b>	<b>Description</b>
TCPS118_FL_CNT_INDEX_IGNORE	Ignore I-input
TCPS118_FL_CNT_INDEX_LOAD	Non-Reference Mode: Load on I
TCPS118_FL_CNT_INDEX_LATCH	Non-Reference Mode: Latch on I
TCPS118_FL_CNT_INDEX_GATE	Non-Reference Mode: Gate on I
TCPS118_FL_CNT_INDEX_RESET	Non-Reference Mode: Reset on I
TCPS118_FL_CNT_INDEX_REFMODE	Reference mode
TCPS118_FL_CNT_INDEX_AUTOREFMODE	Auto Reference Mode
TCPS118_FL_CNT_INDEX_INDEXMODE	Index Mode

*polA*

This argument specifies the polarity of input A. The following values are defined.

<b>Polarity</b>	<b>Description</b>
TCPS118_FL_CNT_POL_HIGH	Input is high active
TCPS118_FL_CNT_POL_LOW	Input is low active

*polB*

This argument specifies the polarity of input B. The following values are defined.

<b>Polarity</b>	<b>Description</b>
TCPS118_FL_CNT_POL_HIGH	Input is high active
TCPS118_FL_CNT_POL_LOW	Input is low active

*polI*

This argument specifies the polarity of input I. The following values are defined.

<b>Polarity</b>	<b>Description</b>
TCPS118_FL_CNT_POL_HIGH	Input is high active
TCPS118_FL_CNT_POL_LOW	Input is low active

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;
TCPS118_COUNT_CONFIG_BUF countConf;

/*
** configure channel 1 for Counter "Timer Down"
**   clkRate = 10 MHz, preload = 10000
**   DIV-by-N
**/
countConf.channel = 1;
countConf.countMode = TCPS118_FL_CNT_MODE_TIMERDOWN;
countConf.preloadValue = 10000;
countConf.intClockRate = 10;
countConf.specialMode = TCPS118_FL_CNT_SPECMODE_DIVBYN;
countConf.indexMode = TCPS118_FL_CNT_INDEX_IGNORE;
/* polA, polB and polI are not used */

result = tcps118ConfigCounter (hdl, &countConf);
if (result != TCPS118_OK)
{
    /* handle error */
}

```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVAL	A specified parameter is invalid, e.g. counter or index mode
TCPS118_ERR_BUSY	The channel is busy and already configured, it must be disabled first

## 3.2.18 tcps118DisableCounter

### NAME

tcps118DisableCounter – disable channel and leave counter mode

### SYNOPSIS

```
TCPS118_STATUS tcps118DisableCounter
(
    TCPS118_HANDLE      hdl,
    unsigned int        counterChan
)
```

### DESCRIPTION

This function stops the counter mode for the specified channel and disables it.

### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the channel that shall be disabled.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;

/*
** disable counter channel 1
*/
result = tcps118DisableCounter(hdl, 1);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. the specified channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

## 3.2.19 tcps11SetCounterPreload

### NAME

tcps11SetCounterPreload – set a new counter preload value

### SYNOPSIS

```
TCPS118_STATUS tcps11SetCounterPreload
(
    TCPS118_HANDLE          hdl,
    unsigned int            counterChan,
    unsigned int            preloadValue
)
```

### DESCRIPTION

This function sets a new preload value for the specified channel. The value will be used starting with the next load event.

### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the channel that shall be disabled.

*preloadValue*

This parameter specifies the new preload value.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;

/*
** set preload value of channel 1 to 100000
*/
result = tcps118SetCounterPreload (hdl, 1, 100000);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVAL	A specified parameter is invalid (e.g. the channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

## 3.2.20 tcps118LoadCounter

### NAME

tcps118LoadCounter – set counter value to the preload value

### SYNOPSIS

```
TCPS118_STATUS tcps118LoadCounter
(
    TCPS118_HANDLE          hdl,
    unsigned int            counterChan
)
```

### DESCRIPTION

This function loads the preload value into the counter.

### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the counter channel that shall be loaded.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** load preload value into counter channel 1
*/
result = tcps118LoadCounter (hdl, 1);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. the specified channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

## 3.2.21 tcps118ResetCounter

### NAME

tcps118ResetCounter – reset the value of the specified counter

### SYNOPSIS

```
TCPS118_STATUS tcps118ResetCounter
(
    TCPS118_HANDLE      hdl,
    unsigned int         counterChan
)
```

### DESCRIPTION

This function resets the value of the specified counter.

### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the counter channel that shall be reset.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;

/*
** reset counter channel 1
*/
result = tcps118ResetCounter (hdl, 1);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. the specified channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

## 3.2.2 tcps118WaitCounterMatch

### NAME

tcps118WaitCounterMatch – wait for counter match event

### SYNOPSIS

```
TCPS118_STATUS tcps118WaitCounterMatch
(
    TCPS118_HANDLE          hdl,
    unsigned int             counterChan,
    unsigned int             matchValue,
    int                     timeout
)
```

### DESCRIPTION

This function waits for a counter match event on the specified channel.

### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the counter channel.

*matchValue*

This argument specifies the value, which must match the counter value to generate the event.

*timeout*

This argument specifies the maximum time to wait for the event. The value is specified in ms.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** Wait until counter channel 1 matches 5000 (max. 2000 ms)
*/
result = tcps118WaitCounterMatch(hdl, 1, 5000, 2000);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_BUSY	Waiting for interval timer event is already in use
TCPS118_ERR_TIMEOUT	The specified time to wait has passed before the event occurred
TCPS118_ERR_INVAL	A specified parameter is invalid (e.g. the specified channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

### 3.2.23 tcps118WaitCounterControl

#### NAME

tcps118WaitCounterControl – wait for counter control event

#### SYNOPSIS

```
TCPS118_STATUS tcps118WaitCounterControl  
(  
    TCPS118_HANDLE          hdl,  
    unsigned int            counterChan,  
    int                     timeout  
)
```

#### DESCRIPTION

This function waits for a counter control event, which depends on the index control mode.

#### PARAMETERS

*hdl*

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

*counterChan*

This argument specifies the number of the counter channel.

*timeout*

This argument specifies the maximum time to wait for the event. The value is specified in ms.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** Wait until a control event occurs on counter channel 1 (max. 2000 ms)
*/
result = tcps118WaitCounterControl (hdl, 1, 2000);
if (result != TCPS118_OK)
{
    /* handle error */
}
```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_BUSY	Waiting for interval timer event is already in use
TCPS118_ERR_TIMEOUT	The specified time to wait has passed before the event occurred
TCPS118_ERR_INVAL	A specified parameter is invalid (e.g. the specified channel number)
TCPS118_ERR_CONFIG	The channel is not configured as a counter channel

## 3.2.24 tcps118ReadMulti

### NAME

tcps118ReadMulti – read value of multiple channels

### SYNOPSIS

```
TCPS118_STATUS tcps118ReadMulti
(
    TCPS118_HANDLE          hdl,
    int                     timeout,
    unsigned int            *pValidMask,
    unsigned int            *pValueBuf
)
```

### DESCRIPTION

This function starts a read on multiple channels and returns the values of the enabled channels. The read will be delayed for at least the duration the slowest channel needs to complete the read.

### PARAMETERS

*hdl*

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

*timeout*

This argument specifies the maximum time to wait for completion of the multiple read. The value is specified in ms.

*pValidMask*

This argument is a pointer to an unsigned int value, where a mask showing the active channels will be returned. Bit 0 specifies channel 0, bit 1 specifies channel 1 and so on. A set bit shows, that the channel is active and a valid value will be returned in buffer.

*pValueBuf*

This argument is a pointer to an array of unsigned int values, where the read values will be returned in. The value at index 0 will contain the value of channel 0, at index 1 the value of channel 1, and so on. Only channels marked as active in pValidMask will return a valid value.

## EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE      hdl;
TCPS118_STATUS      result;
unsigned int         chMask;
unsigned int         values[TCPS118_MAX_CHANS];
int^                 ch;

/*
** get values of multiple channels
** - timeout after 1 s
*/
result = tcps118ReadMulti(hdl, 1000, &chMask, values);
if (result != TCPS118_OK)
{
    /* handle error */
}

for (ch = 0; ch < TCPS118_MAX_CHANS; ch ++)
{
    if (chMask & (1 << ch))
    {
        /* Value is valid */
        printf("#%d - Value: 0x%08X\n", ch, values[ch]);
    }
}

```

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_BUSY	Waiting for a completion of an active read
TCPS118_ERR_TIMEOUT	The specified time to wait has passed before the read completed
TCPS118_ERR_CONFIG	No active channel for multiple read

## 3.2.25 tcps118ConfigMulti

### NAME

tcps118ConfigMulti – select channels for multiple reads

### SYNOPSIS

```
TCPS118_STATUS tcps118ConfigMulti
(
    TCPS118_HANDLE          hdl,
    unsigned int            selMultiChan
)
```

### DESCRIPTION

This function selects the channels, that shall be used for multiple channel reads.

### PARAMETERS

*hdl*

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

*selMultiChan*

This parameter sets a mask for enabled channels for multiple channel reads. The selected channels must be configured in SSI or counter mode. A set bit 0 enables channel 0 for multiple read, a set bit 1 selects channel 1, and so on.

### EXAMPLE

```
#include "tcps118api.h"

TCPS118_HANDLE          hdl;
TCPS118_STATUS          result;

/*
** enable channel 1, 2, and 5 for multiple channel read
*/
result = tcps118ConfigMulti(hdl, (1<<1) | (1<<2) | (1<<5));
if (result != TCPS118_OK)
{
    /* handle error */
}
```

---

## RETURN VALUE

On success, TCPS118\_OK is returned. In the case of an error, the appropriate error code is returned by the function.

## ERROR CODES

Error Code	Description
TCPS118_ERR_INVALID_HANDLE	The specified device handle is invalid
TCPS118_ERR_INVALID	A specified parameter is invalid (e.g. invalid channel mask)
TCPS118_ERR_CONFIG	At least one of the selected channels is not configured as a counter or SSI channel

---

## 4 Appendix

### 4.1 Build Error Message String

#### NAME

tcps118ErrorMessage – build error message string

#### SYNOPSIS

```
char* tcps118ErrorMessage  
(  
    TCPS118_STATUS    status  
)
```

#### DESCRIPTION

This function returns a character string containing the TCPS118 error message of the specified status.

#### PARAMETERS

*status*

This argument specifies the error code returned from the TCPS118 device driver function.

#### RETURN VALUE

Returns a null-terminated character string containing the error code name.

### 4.2 Timeout Granularity

The granularity of the timeout check can be modified in the tcps118def.h file. The #define of TIMEOUT\_GRANULARITY\_MS specifies the granularity the driver will use to test the pending jobs for a timeout. The default value is 100 ms.

Additionally to the specified TIMEOUT\_GRANULARITY\_MS, the granularity depends on the system tick but finally the granularity will be at least the specified time. Therefore, the specified timeout for an API call will be at least the specified timeout.