The Embedded I/O Company

TEWS TECHNOLOGIES

TAMC532-SW-82

Linux Device Driver

32 x 12/14 Bit 50/75 Msps ADC for MTCA.4 Rear-I/O

User Manual

Issue 1.0.0

December 2019

powerBridge Computer

Ehlbeek 15a
30938 Burgwedel
fon 05139-9980-0
fax 05139-9980-49

www.powerbridge.de
info@powerbridge.de

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
(0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
info@tews.com www.tews.com

TAMC532-SW-82

Linux Device Driver

32 x 12/14 Bit 50/75 Msps ADC
for MTCA.4 Rear-I/O

Supported Modules:
TAMC532

Issue	Description	Date
1.0.0	First Issue	December 13, 2019

Table of Contents

1 INTRODUCTION 4

2 INSTALLATION 5

2.1 Build and Install the Device Driver 5
2.2 Uninstall the Device Driver 6
2.3 Install Device Driver into the running Kernel 6
2.4 Remove Device Driver from the running Kernel 6
2.5 Change Major Device Number 7

3 API DOCUMENTATION 8

3.1 General Functions 8
3.1.1 tamc532Open 8
3.1.2 tamc532Close 10
3.1.3 tamc532GetPciInfo 12
3.1.4 tamc532GetModuleStatus 14
3.2 Configuration Functions 16
3.2.1 tamc532SetSampleCountLimit 16
3.2.2 tamc532SetPreTriggerCount 18
3.2.3 tamc532SetTriggerSource 20
3.3 I²C Functions 22
3.3.1 tamc532I2CWriteRead 22
3.4 Data Acquisition Functions 26
3.4.1 tamc532AdcReadChannel 26
3.4.2 tamc532AdcReadSample 28
3.4.3 tamc532AdcReadBuffers 30
3.4.4 tamc532ArmCsptUnits 34
3.4.5 tamc532RaiseSoftwareTrigger 36

4 DIAGNOSTIC 38

1 Introduction

The TAMC532-SW-82 Linux device driver allows the operation of the TAMC532 devices conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close(),*and *ioctl()* functions.

The TAMC532-SW-82 device driver supports the following features:

- Configure the ADC data acquisition (trigger source, number of ADC samples)
- Read Single ADC Channel Data
- Sample ADC data using DMA transfers

The TAMC532-SW-82 supports the modules listed below:

TAMC532	32 x 12/14 Bit 50/75 Msps ADC for MTCA.4 Rear-I/O	MTCA.4

In this document all supported modules and devices will be called TAMC532. Specials for a certain device will be advised.

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

TAMC532 User Manual

2 Installation

The directory TAMC532-SW-82 on the distribution media contains the following files:

TAMC532-SW-82-1.0.0.pdf	This manual in PDF format
TAMC532-SW-82-SRC.tar.gz	GZIP compressed archive with driver source code
ChangeLog.txt	Release history
Release.txt	Information about the Device Driver Release

The GZIP compressed archive TAMC532-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tamc532':

tamc532.c	Driver source code
tamc532def.h	Driver include file
tamc532.h	Driver include file for application program
Makefile	Device driver make file
makenode	Script for device node creation
COPYING	Copy of the GNU Public License (GPL)
api/tamc532api.h	API include file
api/tamc532api.c	API source file
example/tamc532exa.c	Example application
example/tamc532dma.c	Example application for DMA data acquisition
example/Makefile	Example application makefile
include/tpmodule.c	Driver independent library
include/tpmodule.h	Driver independent library header file
include/tpxxxhwdep.h	HAL library header file
include/tpxxxhwdep.c	HAL library source file

In order to perform an installation, extract all files of the archive TAMC532-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TAMC532-SW-82-SRC.tar.gz' will extract the files into the local directory.

- Login as *root* and change to the target directory
- Copy tamc532.h to */usr/include*

2.1 Build and Install the Device Driver

- Login as *root*
- Change to the target directory
- To create and install the driver in the module directory */lib/modules/<version>/misc* enter:

  # make install

- To update the device driver's module dependencies, enter:

  # **depmod -aq**

2.2 Uninstall the Device Driver

- Login as *root*
- Change to the target directory
- To remove the driver from the module directory */lib/modules/<version>/misc* enter:

  # make uninstall

2.3 Install Device Driver into the running Kernel

- To load the device driver into the running kernel, login as root and execute the following commands:

  # modprobe tamc532drv

- After the first build or if you are using dynamic major device allocation it is necessary to create new device nodes on the file system. Please execute the script file *makenode* to do this. If your kernel has enabled a device file system (devfs or sysfs with udev) then you have to skip running the *makenode* script. Instead of creating device nodes from the script the driver itself takes creating and destroying of device nodes in its responsibility.

  # sh makenode

On success the device driver will create a minor device for each TAMC532 device found. The first TAMC532 device can be accessed with device node /dev/tamc532_0, the second module with device node /dev/tamc532_1 and so on.

The assignment of device nodes to physical TAMC532 modules depends on the search order of the PCI bus driver.

2.4 Remove Device Driver from the running Kernel

- To remove the device driver from the running kernel login as root and execute the following command:

  # modprobe –r tamc532drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tamc532_x nodes will be automatically removed from your file system after this.

Be sure that the driver isn't opened by any application program. If opened you will get the response "*tamc532drv: Device or resource busy*" and the driver will still remain in the system until you close all opened files and execute *modprobe –r* again.

2.5 Change Major Device Number

This paragraph is only for Linux kernels without dynamic device file system installed. The TAMC532 driver uses dynamic allocation of major device numbers per default. If this isn't suitable for the application it is possible to define a major number for the driver.

To change the major number, edit the file tamc532def.h, change the following symbol to appropriate value, and enter `make install` to create a new driver.

TAMC532_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TAMC532_MAJOR   122
```

Be sure that the desired major number isn't used by other drivers. Please check *proc/devices* to see which numbers are free.

3 API Documentation

3.1 General Functions

3.1.1 tamc532Open

NAME

tamc532Open – opens a device.

SYNOPSIS

```
TAMC532_HANDLE tamc532Open
(
        char            *DeviceName
)
```

DESCRIPTION

Before I/O can be performed to a device, a device 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 following device naming must be used:

Device Number	Device Name
1	/dev/tamc532_0
2	/dev/tamc532_1

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE   hdl;

/*
** open the specified device
*/
hdl = tamc532Open("/dev/tamc532_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 tamc532Close

NAME

tamc532Close – closes a device.

SYNOPSIS

```
TAMC532_STATUS tamc532Close
(
      TAMC532_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 <tamc532api.h>

TAMC532_HANDLE   hdl;
TAMC532_STATUS   result;

/*
** close the device
*/
result = tamc532Close(hdl);
if (result != TAMC532_OK)
{
   /* handle close error */
}
```

RETURNS

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.3 tamc532GetPciInfo

NAME

tamc532GetPciInfo – get information of the module PCI header

SYNOPSIS

```
TAMC532_STATUS tamc532GetPciInfo
(
    TAMC532_HANDLE          hdl,
    TAMC532_PCIINFO_BUF     *pPciInfoBuf
)
```

DESCRIPTION

This function returns information of the module PCI header in the provided data buffer.

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 TAMC532_PCIINFO_BUF that receives information of the module PCI header.

```
typedef struct
{
    unsigned short      vendorId;
    unsigned short      deviceId;
    unsigned short      subSystemId;
    unsigned short      subSystemVendorId;
    int                 pciBusNo;
    int                 pciDevNo;
    int                 pciFuncNo;
} TAMC532_PCIINFO_BUF;
```

vendorId

PCI module vendor ID.

deviceId

PCI module device ID

subSystemId
PCI module sub system ID

subSystemVendorId
PCI module sub system vendor ID

pciBusNo
Number of the PCI bus, where the module resides.

pciDevNo
PCI device number

pciFuncNo
PCI function number

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE        hdl;
TAMC532_STATUS        result;
TAMC532_PCIINFO_BUF   pciInfoBuf

/*
** get module PCI information
*/
result = tamc532GetPciInfo( hdl, &pciInfoBuf );

if (result != TAMC532_OK)
{
   /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

3.1.4 tamc532GetModuleStatus

NAME

tamc532GetModuleStatus – get current status of the module

SYNOPSIS

```
TAMC532_STATUS tamc532GetModuleStatus
(
    TAMC532_HANDLE              hdl,
    TAMC532_MODULESTATUS_BUF    *pModuleStatusBuf
)
```

DESCRIPTION

This function returns status information of the module in the provided data buffer.

PARAMETERS

hdl

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

pModuleStatusBuf

This argument is a pointer to the structure TAMC532_MODULESTATUS_BUF that receives information of the current module status.

```
typedef struct
{
    unsigned int    ModuleStatus;
    unsigned int    I2CBridge;
    unsigned int    DmaController[4];
    unsigned int    ApplicationStatus;
    unsigned int    FirmwareId;
} TAMC532_MODULESTATUS_BUF;
```

ModuleStatus

Holds the current value of the Module Status Register.

I2CBridge

Holds the current value of the I²C Bridge Status Register.

DmaController

Holds the current values of the DMA Controller x Status registers.

Array Index	DMA Controller Number
0	0
1	1
2	2
3	3

ApplicationStatus

Holds the current value of the Application Status Register.

FirmwareId

Holds the current value of the Firmware Identification Register.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE              hdl;
TAMC532_STATUS              result;
TAMC532_MODULESTATUS_BUF    moduleStatusBuf

/*
** get module status information
*/
result = tamc532GetModuleStatus( hdl, &moduleStatusBuf );

if (result != TAMC532_OK)
{
  /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

3.2 Configuration Functions

3.2.1 tamc532SetSampleCountLimit

NAME

tamc532SetSampleCountLimit – Configure Sample Count Limit of a CSPT Unit

SYNOPSIS

```
TAMC532_STATUS tamc532SetSampleCountLimit
(
    TAMC532_HANDLE      hdl,
    int                 csptUnit,
    unsigned int        SampleCountLimit
)
```

DESCRIPTION

This function configures the total number of ADC samples to be stored by the selected CSPT unit.

PARAMETERS

hdl

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

csptUnit

This argument specifies the CSPT unit which shall be affected.

SampleCountLimit

This argument specifies the total number of samples which shall be acquired.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE     hdl;
TAMC532_STATUS     result;

/*
** Configure CSPT Unit #0 to acquire a total of 10000 ADC Samples.
*/
result = tamc532SetSampleCountLimit( hdl, 0, 10000 );
if (result != TAMC532_OK)
{
   /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid
TAMC532_ERR_INVAL	The specified CSPT unit is invalid

3.2.2 tamc532SetPreTriggerCount

NAME

tamc532SetPreTriggerCount – Configure number of samples stored before the trigger

SYNOPSIS

```
TAMC532_STATUS tamc532SetPreTriggerCount
(
    TAMC532_HANDLE      hdl,
    int                 csptUnit,
    unsigned int        PreTriggerCount
)
```

DESCRIPTION

This function configures the number of ADC samples to be stored by the selected CSPT unit before the trigger arrives.

PARAMETERS

hdl

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

csptUnit

This argument specifies the CSPT unit which shall be affected.

PreTriggerCount

This argument specifies the number of samples which shall be stored before the trigger arrives.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE     hdl;
TAMC532_STATUS     result;

/*
** Configure CSPT Unit #0 to acquire 1000 ADC Samples before the trigger.
*/
result = tamc532SetPreTriggerCount( hdl, 0, 1000 );
if (result != TAMC532_OK)
{
   /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid
TAMC532_ERR_INVAL	The specified CSPT unit is invalid

3.2.3 tamc532SetTriggerSource

NAME

tamc532SetTriggerSource – Configure the trigger source

SYNOPSIS

```
TAMC532_STATUS tamc532SetPreTriggerCount
(
    TAMC532_HANDLE      hdl,
    int                 csptUnit,
    unsigned int        TriggerSource,
    int                 Polarity
)
```

DESCRIPTION

This function configures the trigger source and polarity for the selected CSPT unit.

PARAMETERS

hdl

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

csptUnit

This argument specifies the CSPT unit which shall be affected.

TriggerSource

This argument specifies the trigger source. Possible values are:

Value	Description
TAMC532_TRIGGER_DISABLED	No External Trigger. Using Software Trigger is possible.
TAMC532_TRIGGER_RTM_D5_DIFF	RTM D5 (Differential)
TAMC532_TRIGGER_RTM_D6_DIFF	RTM D6 (Differential)
TAMC532_TRIGGER_RTM_D7_DIFF	RTM D7 (Differential)
TAMC532_TRIGGER_RTM_D8_DIFF	RTM D8 (Differential)
TAMC532_TRIGGER_AMC_PORT_17_RX	AMC Port 17 Rx
TAMC532_TRIGGER_AMC_PORT_17_TX	AMC Port 17 Tx
TAMC532_TRIGGER_AMC_PORT_18_RX	AMC Port 18 Rx
TAMC532_TRIGGER_AMC_PORT_18_TX	AMC Port 18 Tx

TAMC532_TRIGGER_AMC_PORT_19_RX	AMC Port 19 Rx
TAMC532_TRIGGER_AMC_PORT_19_TX	AMC Port 19 Tx
TAMC532_TRIGGER_AMC_PORT_20_RX	AMC Port 20 Rx
TAMC532_TRIGGER_AMC_PORT_20_TX	AMC Port 20 Tx

Polarity

This argument specifies the trigger polarity. Possible values are:

Value	Description
TAMC532_TRIGGER_POLARITY_RISINGEDGE	Trigger on Rising Edge
TAMC532_TRIGGER_POLARITY_FALLINGEDGE	Trigger on Falling Edge

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE     hdl;
TAMC532_STATUS     result;

/*
** Configure CSPT Unit #0 to trigger on rising edge of RTM D5
*/
result = tamc532SetTriggerSource( hdl,
                                  0,
                                  TAMC532_TRIGGER_RTM_D5_DIFF,
                                  TAMC532_TRIGGER_POLARITY_RISINGEDGE
                                 );
if (result != TAMC532_OK)
{
   /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid
TAMC532_ERR_INVAL	At least one of the specified parameters is invalid.

3.3 I²C Functions

3.3.1 tamc532I2CWriteRead

NAME

tamc532I2CWriteRead – Write and/or Read data from the onboard I²C busses

SYNOPSIS

```
TAMC532_STATUS tamc532I2CWriteRead
(
        TAMC532_HANDLE      hdl,
        int                 busNo,
        unsigned char       i2cAddr,
        unsigned char       *pWbuf,
        int                 nbytesWrite,
        unsigned char       *pRbuf,
        int                 nbytesRead,
        int                 *pWrDone,
        int                 *pRdDone,
        int                 timeout_ms
)
```

DESCRIPTION

This function writes the specified number of bytes to the specified I²C device, and reads the specified amount of data afterwards.

PARAMETERS

hdl

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

busNo

This argument specifies the bus number of the desired I²C bus. The following values are possible:

Value	Description
0	BCC I2C Bus
1	RTM I2C Bus (TEWS RTM Special)
2	Payload I2C Bus (ADC, SI5338 etc.)
3	SFP Slot #0
4	SFP Slot #1

i2cAddr

This argument specifies the desired I²C target address. Specify a 7bit address here.

pWbuf

This argument specifies a pointer to the data buffer, which shall be written to the device. If no write action shall be performed, set this parameter to NULL.

nbytesWrite

This argument specifies the number of data bytes which shall be written. If no write action shall be performed, set this parameter to 0.

pRbuf

This argument specifies a pointer to the data buffer, where the data read from the device will be returned. If no read action shall be performed, set this parameter to NULL.

nbytesRead

This argument specifies the number of data bytes which shall be read. If no read action shall be performed, set this parameter to 0.

pWrDone

This argument specifies a pointer to an int value where the number of successfully written data bytes will be returned. If this value is not of interest, set this parameter to NULL.

pRdDone

This argument specifies a pointer to an int value where the number of successfully read data bytes will be returned. If this value is not of interest, set this parameter to NULL.

timeout_ms

This argument specifies the timeout for this I²C action, specified in milliseconds. If the function shall block indefinitely, specify -1.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE   hdl;
TAMC532_STATUS   result;
unsigned char    dataWrite[2];
unsigned char    dataRead[2];
int              nbytesWrite, nbytesRead;
int              wrDone, rdDone;

/*
** read "Configuration DIP Switches Register" (Offset 0x0A) from
** Device 0x68 on bus 0. Wait up to 1 second for completion.
*/
dataWrite[0] = 0x0A;
nbytesWrite  = 1;
nbytesRead   = 1;
result = tamc532I2CWriteRead(
                  hdl,
                  0,
                  0x68,
                  dataWrite,
                  nbytesWrite,
                  dataRead,
                  nbytesRead,
                  &wrDone,
                  &rdDone,
                  1000
                );

if ( (result == TAMC532_OK) && (rdDone == 1) )
{
  printf("Configuration DIP Switch Reg = 0x%02X\n", dataRead[0]);
} else {
  /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid
TAMC532_ERR_INVAL	At least one of the specified parameters is invalid.

3.4 Data Acquisition Functions

3.4.1 tamc532AdcReadChannel

NAME

tamc532AdcReadChannel – Read ADC Data of one specific Channel

SYNOPSIS

```
TAMC532_STATUS tamc532AdcReadChannel
(
    TAMC532_HANDLE          hdl,
    int                     channelNo,
    unsigned short          *pData
)
```

DESCRIPTION

This function reads the most recent value of one ADC channel. This function uses single register access.

PARAMETERS

hdl

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

channelNo

This parameter specifies the channel number. Valid values are 0 to 31.

pData

This parameter specifies a pointer to an unsigned short (16bit) data buffer, where the ADC value is returned. For compatibility reasons regarding different ADC resolutions, the returned ADC value is Most Significant Bit aligned.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE     hdl;
TAMC532_STATUS     result;
unsigned short     adcData;

/*
** read current ADC data of channel #0
*/
result = tamc532AdcReadChannel(hdl, 0, &adcData);
if (result == TAMC532_OK)
{
  printf( "ADC[0] = %d (%04Xh)\n", (signed short)adcData, adcData );
} else {
  /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid
TAMC532_ERR_INVAL	The specified channel is invalid.

3.4.2 tamc532AdcReadSample

NAME

tamc532AdcReadSample – Read values of all ADC channels

SYNOPSIS

```
TAMC532_STATUS tamc532AdcReadSample
(
    TAMC532_HANDLE          hdl,
    TAMC532_ADCSAMPLE_BUF   *pAdcSampleBuf
)
```

DESCRIPTION

This function reads the most recent values of all ADC channels. This function uses single register accesses.

PARAMETERS

hdl

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

pAdcSampleBuf

This argument is a pointer to the structure TAMC532_ADCSAMPLE_BUF that receives the ADC data.

```
typedef struct
{
    unsigned short channeldata[32];
} TAMC532_ADCSAMPLE_BUF;
```

channeldata

ADC data of each channel. Array index 0 corresponds to the first ADC channel, array index 1 corresponds to the second ADC channel and so on. For compatibility reasons regarding different ADC resolutions, the ADC values are Most Significant Bit aligned.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE          hdl;
TAMC532_STATUS          result;
TAMC532_ADCSAMPLE_BUF   AdcSampleBuf;

/*
** Read the most recent data of all 32 ADC channels
*/
result = tamc532AdcReadSample(hdl, &AdcSampleBuf);
if (result == TAMC532_OK)
{
   int i;
   for (i=0; i<32; i++)
   {
     printf("ADC[%d]=%d\n", i, (signed short)AdcSampleBuf.channeldata[i]);
   }
} else {
   /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

3.4.3 tamc532AdcReadBuffers

NAME

tamc532AdcReadBuffers – Read a number of samples using DMA

SYNOPSIS

```
TAMC532_STATUS tamc532AdcReadBuffers
(
    TAMC532_HANDLE      hdl,
    unsigned int        dmaUnitsMask,
    unsigned char       *pAdcDataBuf[4],
    unsigned long       nBytes,
    unsigned long       nBytesValid[4],
    int                 timeout_ms
)
```

DESCRIPTION

This function reads a number of Samples using the specified DMA channel groups, consisting of eight ADC channels per DMA channel. This function uses DMA for data transfer. The function will block until the requested ADC data is transferred, or the specified timeout occurred.

PARAMETERS

hdl

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

dmaUnitsMask

This parameter specifies the DMA units linked to an ADC channel group. Bit #0 refers to ADC A (Channels 0 to 7), bit #1 refers to ADC B (Channels 8 to 15) and so on. The DMA channels depend on each other.

pAdcDataBuf[]

This parameter specifies an array of pointers to unsigned char (8bit) data buffers, which receive the sampled ADC data. The buffers must be large enough to hold the requested amount of data bytes necessary for a complete ADC channel group. Each value of an ADC channel is stored as a 16bit value. For compatibility reasons regarding different ADC resolutions, the returned ADC value is Most Significant Bit aligned. The endianness is automatically adapted to the host architecture, so using 16bit pointers is possible on all platforms. The ADC values are stored as follows:

Index / Offset	16bit Index							
	0	1	2	3	4	5	6	7
0x00	Ch#1	Ch#0	Ch#3	Ch#2	Ch#5	Ch#4	Ch#7	Ch#6
0x10	Ch#1	Ch#0	Ch#3	Ch#2	Ch#5	Ch#4	Ch#7	Ch#6
0x20	Ch#1	Ch#0	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

nBytes

This parameter specifies the number of bytes for the ADC data. A multiple of a complete channel set must be specified. This value can be calculated as follows:

nBytes = numSamples * 8 * 2

nBytesValid[]

This argument returns the number of valid bytes returned in the ADC data buffers. This should normally be the same as requested in nBytes.

timeout_ms

This argument specifies the timeout for this action, specified in milliseconds. If the function shall block indefinitely, specify -1.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE          hdl;
TAMC532_STATUS          result;
signed short            *pAdcDataBufArray[4];
signed short            *pAdcDataBuf;
int                     numSamples;
unsigned long           nBytes;
unsigned long           nBytesValid[4];

/*
** Read ADC data for 10000 samples of all DMA-Units.
** Wait indefinitely.
*/
numSamples  = 10000;
nBytes      = (numSamples * 8 * 2);
pAdcDataBufArray[0] = (signed short*)malloc( nBytes );
pAdcDataBufArray[1] = (signed short*)malloc( nBytes );
…
/* handle allocation error ... */

result = tamc532AdcReadBuffers(
                                hdl,
                                (1 << 3) | (1 << 2) | (1 << 1) | (1 << 0),
                                pAdcDataBufArray,
                                nBytes,
                                nBytesValid,
                                -1
                            );
if (result == TAMC532_OK)
{
  /* show some sample data */
  pAdcDataBuf = pAdcDataBufArray[0];
  printf("ADC[0]=%d (%04Xh)\n", pAdcDataBuf[1],
                                (unsigned short)pAdcDataBuf[1]);
  printf("ADC[1]=%d (%04Xh)\n", pAdcDataBuf[0],
                                (unsigned short)pAdcDataBuf[0]);
  printf("ADC[2]=%d (%04Xh)\n", pAdcDataBuf[3],
                                (unsigned short)pAdcDataBuf[3]);
  printf("ADC[3]=%d (%04Xh)\n", pAdcDataBuf[2],
                                (unsigned short)pAdcDataBuf[2]);
} else {
  /* handle error */
```

```
}
free(pAdcDataBufArray[0]);
free(pAdcDataBufArray[1]);
…
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

3.4.4 tamc532ArmCsptUnits

NAME

tamc532ArmCsptUnits – Activate Acquisition Units for Trigger

SYNOPSIS

```
TAMC532_STATUS tamc532ArmCsptUnits
(
    TAMC532_HANDLE          hdl,
    unsigned int            unitMask
)
```

DESCRIPTION

This function activates (arms) the specified CSPT units, so they start data acquisition upon detecting a trigger signal. This function can also be used for deactivation of the units.

PARAMETERS

hdl

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

unitMask

This parameter specifies the units to be activated. Bit #0 corresponds to CSPT Unit #0, bit #1 corresponds to CSPT Unit #1 and so on. Multiple units can be activated simultaneously.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE     hdl;
TAMC532_STATUS     result;
unsigned short     adcData;

/*
** Arm all CSPT Units
*/
result = tamc532ArmCsptUnits(hdl, 0x0F);
if (result != TAMC532_OK)
{
  /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

3.4.5 tamc532RaiseSoftwareTrigger

NAME

tamc532RaiseSoftwareTrigger – Raise a Software Trigger to initiate Data Acquisition

SYNOPSIS

```
TAMC532_STATUS tamc532RaiseSoftwareTrigger
(
        TAMC532_HANDLE          hdl,
        unsigned int            unitMask
)
```

DESCRIPTION

This function raises a software trigger to start data acquisition on the specified CSPT units.

PARAMETERS

hdl

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

unitMask

This parameter specifies the units to be triggered. Bit #0 corresponds to CSPT Unit #0, bit #1 corresponds to CSPT Unit #1 and so on. Multiple units can be triggered simultaneously.

EXAMPLE

```
#include <tamc532api.h>

TAMC532_HANDLE     hdl;
TAMC532_STATUS     result;
unsigned short     adcData;

/*
** Trigger all CSPT Units
*/
result = tamc532RaiseSoftwareTrigger(hdl, 0x0F);
if (result != TAMC532_OK)
{
  /* handle error */
}
```

RETURN VALUE

On success, TAMC532_OK is returned. In the case of an error, the appropriate error code is returned by the function.

ERROR CODES

Error Code	Description
TAMC532_ERR_INVALID_HANDLE	The specified device handle is invalid

4 Diagnostic

If the TAMC532 does not work properly it is helpful to get some status information from the driver respective kernel.

The Linux */proc* file system provides information about kernel, resources, driver, devices, and so on. The following screen dumps displays information of a correct running TAMC532 driver (see also the proc man pages).

```
# lspci -v
  …
03:00.0 Signal processing controller: TEWS Technologies GmbH Device 8214
        Subsystem: TEWS Technologies GmbH Device 800b
        Physical Slot: 5
        Flags: bus master, fast devsel, latency 0, IRQ 16
        Memory at f7c00000 (32-bit, non-prefetchable) [size=4K]
        Capabilities: [40] Power Management version 3
        Capabilities: [48] MSI: Enable- Count=1/1 Maskable- 64bit+
        Capabilities: [60] Express Endpoint, MSI 00
        Capabilities: [100] Device Serial Number 00-00-00-00-00-00-00-00
        Kernel driver in use: TEWS TECHNOLOGIES - TAMC532 Device Driver
        Kernel modules: tamc532drv
  …


# cat /proc/devices
Character devices:
  1 mem
  2 pty
  …
248 tamc532drv
  …

# cat /proc/iomem
00010000-0009ebff : System RAM
  …
dfa00000-feafffff : PCI Bus 0000:00
  …
  f7800000-f7bfffff : 0000:00:02.0
  f7c00000-f7dfffff : PCI Bus 0000:01
    f7c00000-f7cfffff : PCI Bus 0000:02
      f7c00000-f7cfffff : PCI Bus 0000:03
        f7c00000-f7c00fff : 0000:03:00.0
          f7c00000-f7c00fff : TAMC532
  …
```