The Embedded I/O Company

TEWS TECHNOLOGIES

TAMC900-SW-82

Linux Device Driver

8 High Speed ADCs

Version 1.0.x

User Manual

Issue 1.0.1

January 2011

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

TAMC900-SW-82

Linux Device Driver

8 High Speed ADCs

Supported Modules:
TAMC900

Issue	Description	Date
1.0.0	First Issue	May 25, 2010
1.0.1	No support for kernel 2.4.x	January 7, 2011

Table of Contents

1 Introduction

The TAMC900-SW-82 Linux device driver allows the operation of the TAMC900 device conforming to the Linux I/O system specification. This includes a device-independent basic I/O interface with *open()*, *close(),*and *ioctl()* functions.

The TAMC900-SW-82 device driver does not support 2.4.x Linux kernel versions. Only the 2.6.x kernel tree is supported.

The TAMC900-SW-82 device driver supports the following features:

- Read current ADC value, and sample multiple values
- Configure ADC channel groups and channels

The TAMC900-SW-82 supports the modules listed below:

TAMC900	8 high Speed ADCs, 105MSps, 14Bit	Advanced Mezzanine Card

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

TAMC900 User manual
TAMC900 Engineering Manual

2 Installation

The directory TAMC900-SW-82 on the distribution media contains the following files:

TAMC900-SW-82-1.0.1.pdf	This manual in PDF format
TAMC900-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 TAMC900-SW-82-SRC.tar.gz contains the following files and directories:

Directory path 'tamc900':

tamc900.c	Driver source code
tamc900def.h	Driver include file
tamc900.h	Driver include file for application program
Makefile	Device driver make file
makenode	Script for device node creation
example/tamc900exa.c	Example application
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 TAMC900-SW-82-SRC.tar.gz to the desired target directory. The command 'tar -xzvf TAMC900-SW-82-SRC.tar.gz' will extract the files into the local directory. Additionally, copy *tamc900.h* into your include path.

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 tamc900drv

- 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 TAMC900 device found. The first TAMC900 device can be accessed with device node /dev/tamc900_0, the second module with device node /dev/tamc900_1 and so on.

The assignment of device nodes to physical TAMC900 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 tamc900drv

If your kernel has enabled devfs or sysfs (udev), all /dev/tamc900_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 "*tamc900drv: 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 TAMC900 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 tamc900def.h, change the following symbol to appropriate value and enter `make install` to create a new driver.

TAMC900_MAJOR	Valid numbers are in range between 0 and 255. A value of 0 means dynamic number allocation.

Example:

```
#define TAMC900_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 Device Input/Output functions

This chapter describes the interface to the device driver I/O system.

3.1 open()

NAME

open() opens a file descriptor.

SYNOPSIS

#include <fcntl.h>

int open (*const char *filename, int flags*)

DESCRIPTION

The **open** function creates and returns a new file descriptor for the file named by *filename*. The *flags* argument controls how the file is to be opened. This is a bit mask. Create the value by the bitwise OR of the appropriate parameters (using the | operator in C). See also the GNU C Library documentation for more information about the open function and open flags.

EXAMPLE

```
int fd;

fd = open(“/dev/tamc900_0”, O_RDWR);
if (fd == -1) {
   /* handle error condition */
}
```

RETURNS

The normal return value from **open** is a non-negative integer file descriptor. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during open. For more information about open error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.2 close()

NAME

close() closes a file descriptor.

SYNOPSIS

#include <unistd.h>

int **close** (int *filedes*)

DESCRIPTION

The **close** function closes the file descriptor *filedes*.

EXAMPLE

```
int fd;

if (close(fd) != 0) {
  /* handle close error conditions */
}
```

RETURNS

The normal return value from **close** is 0. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code.

ERRORS

ENODEV	The requested minor device does not exist.

This is the only error code returned by the driver, other codes may be returned by the I/O system during close. For more information about close error codes, see the *GNU C Library description – Low-Level Input/Output*.

SEE ALSO

GNU C Library description – Low-Level Input/Output

3.3 ioctl()

NAME

ioctl() device control functions

SYNOPSIS

#include <sys/ioctl.h>

int **ioctl**(*int filedes, int request [, void *argp])*

DESCRIPTION

The **ioctl** function sends a control code directly to a device, specified by *filedes*, causing the corresponding device to perform the requested operation.

The argument *request* specifies the control code for the operation. The optional argument *argp* depends on the selected request and is described for each request in detail later in this chapter.

The following ioctl codes are defined in *tamc900.h*:

Value	Meaning
TAMC900_IOCG_READBUF	Read a number of ADC samples
TAMC900_IOCG_ADCREAD	Read current ADC data
TAMC900_IOCG_DESCRIPTORCFG_GET	Read DMA descriptor setup
TAMC900_IOCS_DESCRIPTORCFG_SET	Setup DMA descriptor count for each channel
TAMC900_IOCG_CHANGROUPCFG_GET	Read configuration of Channel Group
TAMC900_IOCS_CHANGROUPCFG_SET	Configure Channel Group
TAMC900_IOCG_CHANCFG_GET	Read ADC Channel configuration
TAMC900_IOCS_CHANCFG_SET	Configure ADC Channel
TAMC900_IOCG_DCMCFG_GET	Read DCM configuration
TAMC900_IOCS_DCMCFG_SET	Configure DCM
TAMC900_IOCG_MODULESTATUS	Read Module Status
TAMC900_IOCS_SWTRIGGER_RAISE	Raise software trigger
TAMC900_IOCT_ARMCHANNELS	Arm specified ADC channels for data acquisition
TAMC900_IOCS_REGREAD	Read module register content

See below for more detailed information on each control code.

To use these TAMC900 specific control codes the header file *tamc900.h* must be included in the application.

RETURNS

On success, zero or a value greater than zero is returned. In case of an error, a value of –1 is returned. The global variable *errno* contains the detailed error code. Special ioctl functions may return a specific result.

ERRORS

EINVAL	Invalid argument. This error code is returned if the requested ioctl function is unknown. Please check the argument *request*.

Other function dependant error codes will be described for each ioctl code separately. Note, the TAMC900 device driver always returns standard Linux error codes.

SEE ALSO

ioctl man pages

3.3.1 TAMC900_IOCG_READBUF

NAME

TAMC900_IOCG_READBUF – Read a number of ADC samples

DESCRIPTION

This function starts the sampling of ADC data. Data is sampled if the corresponding channel is armed (refer to ioctl function TAMC900_IOCT_ARMCHANNELS), and the configured trigger signal has been detected. The sampled data is stored in the user buffer using DMA. For this direct access the user memory portion is mapped into kernel space. A pointer to the caller's data buffer (*TAMC900_READBUF_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int            Channel;
   int            nbytes;
   int            timeout;
   unsigned char  flags;
   unsigned char  *pData;
} TAMC900_READBUF_STRUCT;
```

Channel

This parameter specifies the desired ADC channel to be used for this operation. Valid values are 0 to 7.

nbytes

This parameter specifies the size of the allocated memory section in bytes.

timeout

This parameter specifies the amount of time to wait for completion of the data transfer. This timeout includes the time to wait for the trigger event. The timeout is specified in milliseconds.

flags

This parameter specifies the size of the allocated memory section in bytes.

pData

This parameter specifies a pointer to a user data buffer large enough to receive the requested ADC data.

EXAMPLE

```
#include <tamc900.h>

int                     fd;
int                     result;
TAMC900_READBUF_STRUCT  ReadStruct;

/*
** allocate enough memory to hold 2000 ADC samples,
** wait up to 10s for the ADC data on first channel.
*/
ReadStruct.pData   = malloc( 2000 * sizeof(unsigned short) );
ReadStruct.nbytes  = 2000 * sizeof(unsigned short);
ReadStruct.timeout = 10000;
ReadStruct.Channel = 0;

result = ioctl(fd, TAMC900_IOCG_READBUF, &ReadStruct);

if (result < 0) {
  /* handle ioctl error */
}
free(ReadStruct.pData);
```

ERRORS

EFAULT	Error while copying data to or from user space.
EINVAL	Specified Channel is invalid.
EBUSY	There is another active job.
ETIME	The specified timeout occurred.

Other returned error codes are system error conditions.

3.3.2 TAMC900_IOCG_ADCREAD

NAME

TAMC900_IOCG_ADCREAD – Read current ADC data

DESCRIPTION

This function reads the current ADC value of the specified channel. A pointer to the caller's data buffer (*TAMC900_ADC_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int              Channel;
   unsigned short   AdcValue;
} TAMC900_ADC_STRUCT;
```

Channel

This parameter specifies the desired ADC channel to be used for this operation. Valid values are 0 to 7.

AdcValue

This parameter returns the current ADC value.

EXAMPLE

```
#include <tamc900.h>

int                  fd;
int                  result;
TAMC900_ADC_STRUCT   AdcStruct

/*
** Read ADC value of channel 7
*/
AdcStruct.Channel = 7;
result = ioctl(fd, TAMC900_IOCG_ADCREAD, &AdcStruct);

if (result < 0) {
   /* handle ioctl error */
} else {
   printf("Current ADC value: 0x%04X\n", AdcStruct.AdcValue);
}
```

ERRORS

EFAULT	Error while copying data to or from user space.
EINVAL	Specified Channel is invalid.

Other returned error codes are system error conditions.

3.3.3 TAMC900_IOCG_DESCRIPTORCFG_GET

NAME

TAMC900_IOCG_DESCRIPTORCFG_GET – Read DMA descriptor setup

DESCRIPTION

This ioctl function returns the number of assigned DMA descriptors for the ADC channels. The descriptor memory is located in the TAMC900 local memory, which is the reason why the number of descriptors is limited. A pointer to the caller's data buffer (*TAMC900_DESCRIPTORSETUP_STRUCT*) is passed by the parameter *argp* to the driver.

typedef struct {
  int NumberOfDescriptors[8];
} TAMC900_DESCRIPTORCFG_STRUCT;

NumberOfDescriptors

This parameter is an array of integer values describing the number of DMA descriptors for each ADC channel. The total number of descriptors must not exceed the available descriptor memory. Up to 680 descriptors can be used in sum.

EXAMPLE

```
#include <tamc900.h>

int                              fd;
int                              result;
TAMC900_DESCRIPTORCFG_STRUCT     DescriptorCfg;

/*
** Read current DMA descriptor assignment
*/
result = ioctl(fd, TAMC900_IOCG_DESCRIPTORCFG_GET, &DescriptorCfg);
if (result >= 0) {
   for (channel=0; channel<8; channel++)
   {
      printf("  Channel %d: %d DMA descriptors\n", channel,
                              DescriptorCfg.NumberOfDescriptors[channel]);
   }
} else {
   /* handle ioctl error */
}
```

ERRORS

EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.4 TAMC900_IOCS_DESCRIPTORCFG_SET

NAME

TAMC900_IOCS_DESCRIPTORCFG_SET – Setup DMA descriptor count for each channel

DESCRIPTION

This ioctl function assigns a number of DMA descriptors to the ADC channels. The descriptor memory is located in the TAMC900 local memory, which is the reason why the number of descriptors is limited. A pointer to the caller's data buffer (*TAMC900_DESCRIPTORSETUP_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int NumberOfDescriptors[8];
} TAMC900_DESCRIPTORCFG_STRUCT;
```

NumberOfDescriptors

This parameter is an array of integer values describing the number of DMA descriptors for each ADC channel. The total number of descriptors must not exceed the available descriptor memory. Up to 680 descriptors can be used in sum.

EXAMPLE

```
#include <tamc900.h>

int                             fd;
int                             result;
TAMC900_DESCRIPTORCFG_STRUCT    DescriptorCfg;

/*
** Assign 40 descriptors to each ADC channel
*/
DescriptorCfg.NumberOfDescriptors[0] = 40;
DescriptorCfg.NumberOfDescriptors[1] = 40;
DescriptorCfg.NumberOfDescriptors[2] = 40;
DescriptorCfg.NumberOfDescriptors[3] = 40;
DescriptorCfg.NumberOfDescriptors[4] = 40;
DescriptorCfg.NumberOfDescriptors[5] = 40;
DescriptorCfg.NumberOfDescriptors[6] = 40;
DescriptorCfg.NumberOfDescriptors[7] = 40;
result = ioctl(fd, TAMC900_IOCS_DESCRIPTORCFG_SET, &DescriptorCfg);
if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid number of descriptors.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.5 TAMC900_IOCG_CHANGROUPCFG_GET

NAME

TAMC900_IOCG_CHANGROUPCFG_GET – Read configuration of Channel Group

DESCRIPTION

This ioctl function returns the specified channel group configuration.

A pointer to the caller's data buffer (*TAMC900_CHANGROUPCFG_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
    int             ChannelGroup;
    unsigned char   SamClock;
    int             SamClockEna;
    unsigned char   TriggerMode;
    unsigned char   TriggerInput;
    unsigned char   TriggerEvent;
} TAMC900_CHANGROUPCFG_STRUCT;
```

ChannelGroup

This parameter describes the desired channel group for this operation. Valid values are 0 to 1.

SamClock

This parameter specifies the clock source to be used for data sampling. Valid values are:

Value	Description
TAMC900_SAMCLOCK_AMCREFCLK	AMC Reference Clock (100MHz), can also be Spread Spectrum Clock
TAMC900_SAMCLOCK_EXTCLK0	External Clock Input 0
TAMC900_SAMCLOCK_EXTCLK1	External Clock Input 1
TAMC900_SAMCLOCK_EXTCLK2	External Clock Input 2
TAMC900_SAMCLOCK_DCM0	Clock generated by DCM 0
TAMC900_SAMCLOCK_DCM1	Clock generated by DCM 1

SamClockEna

This parameter describes if the selected sample clock is enabled or disabled. If this value is FALSE (0), the sample clock is disabled. If it is TRUE (1), the sample clock is enabled.

TriggerMode

This parameter specifies the trigger mode used for data sampling. Valid values are:

Value	Description
TAMC900_TRIGGERMODE_DISA	Disable trigger functionality (no data sampling)
TAMC900_TRIGGERMODE_AROUND	Sample data before and after trigger event.

TriggerInput

This parameter specifies the trigger input used for data sampling. Valid values are:

Value	Description
TAMC900_TRIGGER_INPUT0	Use Trigger Input 0.
TAMC900_TRIGGER_INPUT1	Use Trigger Input 1.
TAMC900_TRIGGER_INPUT2	Use Trigger Input 2.

TriggerEvent

This parameter specifies the trigger event used for data sampling. Valid values are:

Value	Description
TAMC900_TRIGGER_RISING	Trigger on rising edge.
TAMC900_TRIGGER_FALLING	Trigger on falling edge.

EXAMPLE

```
#include <tamc900.h>

int                          fd;
int                          result;
TAMC900_CHANGROUPCFG_STRUCT  ChanGrpCfg;

/*
** Read configuration of Channel Group 0 (channel 0 to 3)
*/
ChanGrpCfg.ChannelGroup = 0;

result = ioctl(fd, TAMC900_IOCG_CHANGROUPCFG_GET, &ChanGrpCfg);
if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid channel group specified.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.6 TAMC900_IOCS_CHANGROUPCFG_SET

NAME

TAMC900_IOCS_CHANGROUPCFG_SET – Setup configuration of Channel Group

DESCRIPTION

This ioctl function configures the specified channel group.

A pointer to the caller's data buffer (*TAMC900_CHANGROUPCFG_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
    int             ChannelGroup;
    unsigned char   SamClock;
    int             SamClockEna;
    unsigned char   TriggerMode;
    unsigned char   TriggerInput;
    unsigned char   TriggerEvent;
} TAMC900_CHANGROUPCFG_STRUCT;
```

ChannelGroup

This parameter describes the desired channel group for this operation. Valid values are 0 to 1.

SamClock

This parameter specifies the clock source to be used for data sampling. Valid values are:

Value	Description
TAMC900_SAMCLOCK_AMCREFCLK	AMC Reference Clock (100MHz), can also be Spread Spectrum Clock
TAMC900_SAMCLOCK_EXTCLK0	External Clock Input 0
TAMC900_SAMCLOCK_EXTCLK1	External Clock Input 1
TAMC900_SAMCLOCK_EXTCLK2	External Clock Input 2
TAMC900_SAMCLOCK_DCM0	Clock generated by DCM 0
TAMC900_SAMCLOCK_DCM1	Clock generated by DCM 1

SamClockEna

This parameter enables or disables the selected sample clock. Use FALSE (0) to disable and TRUE (1) to enable the sample clock.

TriggerMode

This parameter specifies the trigger mode used for data sampling. Valid values are:

Value	Description
TAMC900_TRIGGERMODE_DISA	Disable trigger functionality (no data sampling)
TAMC900_TRIGGERMODE_AROUND	Sample data before and after trigger event

TriggerInput

This parameter specifies the trigger input used for data sampling. Valid values are:

Value	Description
TAMC900_TRIGGER_INPUT0	Use Trigger Input 0.
TAMC900_TRIGGER_INPUT1	Use Trigger Input 1.
TAMC900_TRIGGER_INPUT2	Use Trigger Input 2.

TriggerEvent

This parameter specifies the trigger event used for data sampling. Valid values are:

Value	Description
TAMC900_TRIGGER_RISING	Trigger on rising edge.
TAMC900_TRIGGER_FALLING	Trigger on falling edge.

EXAMPLE

```
#include <tamc900.h>

int                          fd;
int                          result;
TAMC900_CHANGROUPCFG_STRUCT  ChanGrpCfg;

/*
** Configure Channel Group 0 (channel 0 to 3):
** Use External Clock 0, trigger on rising edge of external trigger input 0
** Sample data around trigger event.
*/
ChanGrpCfg.ChannelGroup = 0;
ChanGrpCfg.SamClock     = TAMC900_SAMCLOCK_EXTCLK0;
ChanGrpCfg.SamClockEna  = 1;
ChanGrpCfg.TriggerMode  = TAMC900_TRIGGERMODE_AROUND;
ChanGrpCfg.TriggerInput = TAMC900_TRIGGER_INPUT0;
ChanGrpCfg.TriggerEvent = TAMC900_TRIGGER_RISING;
```

```
result = ioctl(fd, TAMC900_IOCS_CHANGROUPCFG_SET, &ChanGrpCfg);

if (result < 0) {
   /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid channel group specified.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.7 TAMC900_IOCG_CHANCFG_GET

NAME

TAMC900_IOCG_CHANCFG_GET – Read ADC Channel configuration

DESCRIPTION

This ioctl function reads the current configuration of the specified channel.

A pointer to the caller's data buffer (*TAMC900_CHANCFG_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int          Channel;
   int          UseClockDutyCycleStabilizer;
   int          AdcOutputFormat;
} TAMC900_CHANCFG_STRUCT;
```

Channel

This parameter describes the desired channel for this operation. Valid values are 0 to 7.

UseClockDutyCycleStabilizer

This parameter specifies if the internal clock duty cycle stabilizing mechanism should be used. Use FALSE (0) to disable and TRUE (1) to enable the stabilizer.

AdcOutputFormat

This parameter specifies the ADC output format. Valid values are:

Value	Description
TAMC900_CHANCFG_FORMAT_BINARY	Use binary output format.
TAMC900_CHANCFG_FORMAT_TWOSCOMP	Use two's complement output format.

EXAMPLE

```
#include <tamc900.h>

int                          fd;
int                          result;
TAMC900_CHANCFG_STRUCT       ChanCfg;

/*
** Read Configuration of Channel 1
*/
ChanCfg.Channel  = 1;

result = ioctl(fd, TAMC900_IOCG_CHANCFG_GET, &ChanCfg);
if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid channel group specified.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.8 TAMC900_IOCS_CHANCFG_SET

NAME

TAMC900_IOCS_CHANCFG_SET – Configure ADC Channel

DESCRIPTION

This ioctl function configures the specified channel.

A pointer to the caller's data buffer (*TAMC900_CHANCFG_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int          Channel;
   int          UseClockDutyCycleStabilizer;
   int          AdcOutputFormat;
} TAMC900_CHANCFG_STRUCT;
```

Channel

This parameter describes the desired channel for this operation. Valid values are 0 to 7.

UseClockDutyCycleStabilizer

This parameter specifies if the internal clock duty cycle stabilizing mechanism should be used. Use FALSE (0) to disable and TRUE (1) to enable the stabilizer.

AdcOutputFormat

This parameter specifies the ADC output format. Valid values are:

Value	Description
TAMC900_CHANCFG_FORMAT_BINARY	Use binary output format.
TAMC900_CHANCFG_FORMAT_TWOSCOMP	Use two's complement output format.

EXAMPLE

```
#include <tamc900.h>

int                          fd;
int                          result;
TAMC900_CHANCFG_STRUCT       ChanCfg;

/*
** Configure Channel 1:
** Use Clock Duty Cycle Stabilizer, and two’s complement.
*/
ChanCfg.Channel                       = 1;
ChanCfg.UseClockDutyCycleStabilizer   = 1;
ChanCfg.AdcOutputFormat               = TAMC900_CHANCFG_FORMAT_TWOSCOMP;

result = ioctl(fd, TAMC900_IOCS_CHANCFG_SET, &ChanCfg);
if (result < 0) {
  /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid channel group specified.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.9 TAMC900_IOCG_DCMCFG_GET

NAME

TAMC900_IOCG_DCMCFG_GET – Read DCM configuration

DESCRIPTION

This ioctl function returns the current configuration of the specified DCM. The resulting DCM frequency is calculated as follows:

$$f = 50MHz \cdot \frac{(mult+1)}{(div+1)}$$

The valid frequency range depends on the Virtex-5 FPGA and is limited from 32MHz to 105MHz (maximum ADC sampling rate).

A pointer to the caller's data buffer (*TAMC900_DCMCFG_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int             dcmno;
   unsigned char   mult;
   unsigned char   div;
} TAMC900_DCMCFG_STRUCT;
```

dcmno

This parameter describes the desired DCM for this operation. Valid values are 0 to 1.

mult

This parameter describes the multiplier value. Valid values are 1 to 31.

div

This parameter describes the divider value. Valid values are 0 to 31.

EXAMPLE

```
#include <tamc900.h>

int                     fd;
int                     result;
TAMC900_DCMCFG_STRUCT   DcmCfg;
```

```
/*
** Check DCM0 configuration
*/
DcmCfg.dcmno = 0;
result = ioctl(fd, TAMC900_IOCG_DCMCFG_GET, &DcmCfg);
if (result >= 0) {
  /* handle ioctl error */
} else {
  printf("mult = %d      div = %d\n", DcmCfg.mult, DcmCfg.div);
}
```

ERRORS

EINVAL	Invalid DCM specified.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.10 TAMC900_IOCS_DCMCFG_SET

NAME

TAMC900_IOCS_DCMCFG_SET – Configure DCM

DESCRIPTION

This ioctl function configures the specified DCM. The resulting DCM frequency is calculated as follows:

$$f = 50MHz \cdot \frac{(mult+1)}{(div+1)}$$

The valid frequency range depends on the Virtex-5 FPGA and is limited from 32MHz to 105MHz (maximum ADC sampling rate). Because of jitter problems, please note that the resulting clock should not be used for high-accuracy data sampling.

A pointer to the caller's data buffer (*TAMC900_DCMCFG_STRUCT*) is passed by the parameter *argp* to the driver.

```
typedef struct {
   int             dcmno;
   unsigned char   mult;
   unsigned char   div;
} TAMC900_DCMCFG_STRUCT;
```

dcmno

This parameter describes the desired DCM for this operation. Valid values are 0 to 1.

mult

This parameter describes the multiplier value. Valid values are 1 to 31.

div

This parameter describes the divider value. Valid values are 0 to 31.

EXAMPLE

```
#include <tamc900.h>

int                     fd;
int                     result;
TAMC900_DCMCFG_STRUCT   DcmCfg;

/*
** Setup DCM1 to 75MHz: mult=2, div=1
** 75MHz = 50Mhz * (2+1) / (1+1)
*/
DcmCfg.dcmno  = 1;
DcmCfg.mult   = 2;
DcmCfg.div    = 1;
result = ioctl(fd, TAMC900_IOCS_DCMCFG_SET, &DcmCfg);

if (result >= 0) {
  /* handle ioctl error */
}
```

ERRORS

EINVAL	Invalid DCM specified.
EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.11 TAMC900_IOCG_MODULESTATUS

NAME

TAMC900_IOCG_MODULESTATUS – Read Module Status

DESCRIPTION

This ioctl function returns the current module status. A pointer to an *unsigned char* value must be passed to the driver by the parameter *argp*.

Following values (binary OR'ed) are possible:

Value	Description
TAMC900_MODULESTATUS_OPERATING	TAMC900 is operating
TAMC900_MODULESTATUS_QDRREADY	QDR is ready
TAMC900_MODULESTATUS_DCM0LOCKED	DCM 0 locked properly
TAMC900_MODULESTATUS_DCM1LOCKED	DCM 1 locked properly
TAMC900_MODULESTATUS_COREPLLLOCKED	Virtex-5 Core PLL locked properly

If one of the above values is NOT set, something is wrong with either the TAMC900 or the DCM configuration.

EXAMPLE

```
#include <tamc900.h>

int           fd;
int           result;
unsigned char ModuleStatus;

/* Read TAMC900 Module Status */
result = ioctl(fd, TAMC900_IOCG_MODULESTATUS, &ModuleStatus);

if (result < 0) {
   /* handle ioctl error */
} else {
   /* show the current Module Status */
  printf("  Module Status: 0x%02X\n", ModuleStatus);
}
```

ERRORS

EFAULT	Error while copying data to or from user space.

Other returned error codes are system error conditions.

3.3.12 TAMC900_IOCS_SWTRIGGER_RAISE

NAME

TAMC900_IOCS_SWTRIGGER_RAISE – Raise software trigger

DESCRIPTION

This ioctl function raises the software trigger event. No additional parameter is required.

EXAMPLE

```
#include <tamc900.h>

int           fd;
int           result;

/* Raise Software Trigger */
result = ioctl(fd, TAMC900_IOCS_SWTRIGGER_RAISE, 0);

if (result < 0) {
  /* handle ioctl error */
}
```

3.3.13 TAMC900_IOCT_ARMCHANNELS

NAME

TAMC900_IOCT_ARMCHANNELS – Arm specified ADC channels for data acquisition

DESCRIPTION

This ioctl function arms specified ADC channels for data acquisition. This function must be called to enable the detection of the selected trigger signal. The argument specifies a bitmask, where bit 0 corresponds to the first ADC channel, bit 1 corresponds to the second ADC channel and so on.

EXAMPLE

```
#include <tamc900.h>

int           fd;
int           result;

/* ARM first two ADC channels */
result = ioctl(fd, TAMC900_IOCT_ARMCHANNELS, (1 << 1) | (1 << 0));

if (result < 0) {
   /* handle ioctl error */
}
```

4 Diagnostic

If the TAMC900 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 TAMC900 driver (see also the proc man pages).

```
# lspci -v
  …
0f:00.0 Signal processing controller: TEWS Datentechnik GmbH Unknown device
8384
        Subsystem: TEWS Datentechnik GmbH Unknown device 8384
        Flags: bus master, fast devsel, latency 0, IRQ 19
        Memory at ff6fe000 (32-bit, non-prefetchable) [size=8K]
        Expansion ROM at ff500000 [disabled] [size=1M]
        Capabilities: [40] Power Management version 3
        Capabilities: [48] Message Signalled Interrupts: Mask- 64bit+
        Queue=0/0 Enable-
        Capabilities: [60] Express Endpoint IRQ 0
…

# cat /proc/devices
Character devices:
  1 mem
  2 pty
  …
248 tamc900drv
  …

# cat /proc/iomem
00000000-0009fbff : System RAM
  …
ff400000-ff7fffff : PCI Bus #08
  ff400000-ff6fffff : PCI Bus #09
    ff400000-ff6fffff : PCI Bus #0a
      ff400000-ff6fffff : PCI Bus #0b
        ff400000-ff6fffff : PCI Bus #0f
          ff500000-ff5fffff : 0000:0f:00.0
          ff6fe000-ff6fffff : 0000:0f:00.0
          ff6fe000-ff6fffff : TAMC900
  …
```