EUROPEAN SOUTHERN OBSERVATORY

Organisation Européenne pour des Recherches Astronomiques dans l'Hémisphère Austral

Europäische Organisation für astronomische Forschung in der südlichen Hemisphäre

VERY LARGE TELESCOPE

VLT Software

CCS-LCU

Analog I/O Driver

User Manual

Doc.No. VLT-MAN-ESO-17210-0462

Issue 1.3

Date 27.09.1994

Prepared S.Sandrock ....
Name Date Signature

Approved B.Gustafsson ....
Name Date Signature

Released M.Ziebell ....
Name Date Signature

Change Record

Issue/Rev.	Date	Section/Page affected	Reason/Initiation/Document/Remarks
1.0	04.10.1993		First release
1.1	-		(not released)
1.2	-		(not released)
1.3	27.09.1994	3.5, 4, 5 All	Changed sections for driver version 1.3. Converted from WordPerfect to FrameMaker. Updated version numbers, cross-references, and imported files and man-pages. Corrected wrong literals etc.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose

This document[1] provides the User Manual of the VMIVME-3111 Analog I/O Board Driver software module.

It is intended to provide all the necessary information to use the **aio** Driver from the LCU Common Software or higher level applications.

The manual assumes that the reader has a good knowledge of UNIX, C-language, the VxWorks operation system and is familiar with the VxWorks development environment.

In addition to the **Introduction** section, this manual contains three major sections:

- **Users's Guide:** describes the tasks that can be done by the driver, including examples.
- **Reference:** a list of the functions available.
- **Installation Guide:** how to install the driver and make it ready for use.

At the end, the **Error Messages and Recovery** section provides a complete list of errors and diagnostic messages and possible recovery actions.

1.2 Scope

This manual describes the software module **aio** in its version 1.3.

The **aio** Driver is developed for the VMIVME-3111 Analog I/O Board to set analog output signals and to read analog input signals.

This implementation includes the following C library:

- **aio** - library of routines to access **aio** module

The following hardware and software environment is required to run the driver software described:

- a standard 6U VMEbus chassis with bus backplane and power supply
- at least one VMIVME-3111 Analog I/O board, any version
- at least one Motorola MVME167 CPU board, any version
- VxWorks 5.1 operating system
- **lcudrv** for common driver functions, version 1.3
- **lculog** for internal logging, version 1.4

1. The document is prepared according to the IEEE 1063-1987 (Standard for SW User Documentation) as recommended in sect. 4.2.14 page 34 of the SW Management Plan - Issue 2.0, 21/05/92.

1.3 Reference Documents

The following documents contain additional information and are referenced in the text.

[1] VxWorks Version 5.1 Programmer's Guide
Wind River Systems

[2] VxWorks Version 5.1 Reference Manual
Wind River Systems

[3] VLT Software Programming Standards
VLT-PRO-ESO-10000-0228, Issue 1.0, 10/03/93

[4] VMIVME-3111
48 Analog-to-Digital Inputs 2-Channel Digital-to-Analog Output Board
Instruction Manual
Document No. 500-003111-000

[5] CCS-LCU Analog I/O Driver Specification
VLT-SPE-ESO-17210-0355, Issue 1.0, 16/12/92

[6] VLT SW Rel1 - Overview and Installation
VLT-MAN-ESO-17200-0642, Issue 1.1, 16/06/94

[7] CCS-LCU Driver Development Guide
VLT-SPE-ESO-17210-0375, Issue 1.0, 12/09/94

1.4 Abbreviations And Acronyms

The following abbreviations and acronyms are used in this document:

AIO	Analog I/O Interface Module
CCS	Central Control Software
CDT	Command Descriptor Table
DCT	Driver Channel Table
EOC	End of analog-to-digital conversion
HW	Hardware
I/O	Input/Output
ISR	Interrupt Service Routine
LAN	Local Area Network
LCC	LCU Common Software
LCU	Local Control Unit
N/A	Not Applicable
OS	Operating System
SW	Software
TBD	To Be Defined
VLT	Very Large Telescope
WS	Workstation

1.5 Stylistic Conventions

The following styles are used:

bold - in the text, for commands, filenames, prefixes/suffixes as they have to be typed.

italic - in the text, for parts that have to be substituted with the real content before typing.

`teletype` - for examples.

`<name>` in the examples, for parts that have to be substituted with the real content before typing.

bold and *italic* are also used to highlight words.

Items which are subject to change in future versions are marked in this way (AuthorRemark).

✗Very important items are marked in this way (Warning).

Changes to the previous issue of this document are marked with **Change-Bars**.

1.6 Naming Conventions

This implementation follows the naming conventions as outlined in the VLT Programming Standards [3].

1.7 Problem Reporting/Change Request

To:

- report a problem/error encountered using the software and/or the documentation,
- suggest changes in the software or documentation

fill an SPR-form[1] (shown in the following page) and send it with any other additional material that you think could be helpful, to ESO:

- by mail or fax for the attention of:

  VLT/ELE- SOFTWARE GROUP/Software Configuration Control Manager

- by e-mail to:

  vltsccm@eso.org

Your request will be checked-in the SPR data base and processed according to ESO change control procedure. The registration number will be communicated to you for further reference and you will be kept informed of the outcome of your SPR.

1. The utility `getTemplate` provides such form as editable file.

```
==============================================================================
  E. S. O.       ---- VERY LARGE TELESCOPE PROJECT  ----
------------------------------------------------------------------------------
SOFTWARE PROBLEM REPORT/CHANGE REQUEST FORM             DATE: <dd/mm/yy>
------------------------------------------------------------------------------
From: . . . . <yourAddress> . . . .
      . . . . . . . . . . . . . . .
------------------------------------------------------------------------------
Software/Document identification
module name / Title       : . . . . . . . . . . . . . . . . . . . . . . .
Version/Doc.NR-issue-date: . . . . . . . . . . . . . . . . . . . . . . .
------------------------------------------------------------------------------
 Please mark one of the following categories

     _ software error
     _ error in the documentation
     _ change request
------------------------------------------------------------------------------
Description:


>>>>> Please delete these lines and provide:
>>>>>     for software errors:
>>>>>         - the description of the anomaly, uncluding outputs,
>>>>>         - how to reproduce it,
>>>>>         - if existing, the temporary solution,
>>>>>         - if known, the reference to where it was specified correctly
>>>>>         - if known, the way to fix it.
>>>>>
>>>>>     for errors in the documentation:
>>>>>         - the exact location of the error (DocNr, issue, page, etc.)
>>>>>         - the text that is wrong
>>>>>         - if known, to what should it be changed.
>>>>>
>>>>>     for change request
>>>>>         - the description of what you would like to have
>>>>>         - if available, possible implementation hints.
>>>>>
>>>>>

==============================================================================
```

2 USER'S GUIDE

This part of the document provides a functional description of the module.

2.1 General about drivers

In general, a software which controls a specific hardware is called a driver and the hardware to be controlled is called device. In case of the VLT LCUs the devices are VMEbus printed circuit boards. The driver runs on the CPU-board of the VMEbus system and is able to control several devices of the same type. To the user's side the driver provides a set of functions which can be invoked through the VxWorks I/O system and allow access to a device from higher level software applications. Thereby this access provides a more logical view of the device as needed for programming purposes and hides the hardware related details.

2.2 Overview

The program controlling hardware is usually called driver, and the specific hardware to control is called a device. The driver can control several devices simultaneously, where each device consists of the same type of hardware, using the same code but with different sets of data for each device. The set of data defining the device and containing the status of the device is called device descriptor table. The driver concept of VxWorks is described in [1].

The software interface of the analog I/O board consists of a set of registers to set output signals, to read input signals, to configure interrupts at the end of a conversion sequence and to get the status of the board. The board can also do autocalibration of the input signals.

The analog I/O board contains 16 input signals and 2 output signals. The board is defined as one device with 16 input and 2 output signals.

The driver supports several analog I/O board in the same system, where each board is one device. These devices share the same driver code and only have different sets of data (device descriptor tables).

2.3 User Interface to the Driver

The **aio** driver provides the standard VxWorks I/O system interface to the user, i.e. the open, close and ioctl system calls.

To get access to the device a user has to open a channel to it by calling the open routine with a file name, specifiying the device and an open mode. The filename of a **aio** device is made up of two parts:

- the driver name **aio** and
- the device unit **0, 1, 2** ...

so, for instance **/aio0** would specify the first **aio** device in a LCU.

The open mode specifies the access rights to the device which are controlled by the driver. The open mode can be :

- Read-Only access: on a channel opened in this mode only read commands can be issued.
- Exclusive read/write access: write access is granted only to the user that performed the open with this option. Requesting Exclusive access to a device already opened in Exclusive or Shared access mode gives back an error.
- Shared read/write access: write access is granted to the user that performed the open and to any other task that later on will request a similar open. Requesting Shared access to a device already opened in Exclusive access mode gives back an error.
- Test read/write access: write access is granted to the user that performed the open with this option without limitation to the present opening status. Requesting Test access to a device already opened in Test access mode gives back an error, i.e. Test access mode is only granted to one user. This mode shall only be used by test and maintenance software. The idea is that test and maintenance software can access the driver without interfering with other users of the driver.

The open call returns a file handle with which the channel is identified in any further driver calls.

An open channel to a device can be closed with the **close** system call.

All functions performed by the **aio** driver are activated with the **ioctl** call. The **ioctl** call has a parameter specifying the action requested by the driver. This parameter is called *ioctl control command* throughout this document.

The ioctl control commands are grouped into *read* and *write* commands. *Read* commands are used to read values or status from the board, while *write* commands are used to write values or perform some action.

In addition an ioctl control command has a further parameter which is an argument of the command. If the command needs multiple arguments then this parameter is the address of a data structure which contains these multiple arguments. All needed data structures are provided in the Module Interface File *aioCommands.h*.

The driver calls are mutually exclusive, that is the device accessed by a user is protected from access from any other authorized user until the executing driver call has terminated. This means that when an user is executing a driver call and another user tries to access the driver for the same device, then the second user task will be blocked until the driver call of the first one has terminated. The blocked task will be put on a queue. The tasks on the queue will be scheduled according to their priority. The driver uses the VxWorks *mutex semaphore* for this purpose to provide *priority inheritance*, see reference [1]. [1]

A task, blocked for access of a device will have a timeout. The timeout value is a general driver parameter and can be set/asked with the *aioSet/GetTimeout* command. After the timeout interval has expired without the task having got access to the device the call is rejected with an error code and the task is resumed.

1. *There should be one semaphore per device, but up to version 1.3 of the driver there is only one semaphore for all devices controlled by the aio driver. This means that there might be more blocking than necessary.*

2.4 Interrupts

Two methodes are available for controlling the analog to digital conversion:

- polling, where the driver runs through a loop adapted to the duration of a analog-digital conversion sequence
- using interrupt response, whereby the board provides an interrupt at the end of each conversion sequence (EOC-Interrupt)

The EOC-Interrupt is used as default by the driver. [1]

Using the interrupt mechanism can at any time be disabled or enabled with the control commands **aioCMD_DISABLE_EOC_INTERRUPT** respectively **aioCMD_ENABLE_EOC_INTERRUPT**. Only one of these methods for one device can be used, that means different adjustment of single input signals of one device is not possible.

Using interrupt response does not load the CPU like polling, but can possibly be slower than polling because of interrupt latency.

All interrupt handling is done completely inside the driver, user-defined ISRs are not necessary (and not possible).

1. This is the case since driver version 1.3.

3 REFERENCE

3.1 Introduction

This chapter provides a detailed description of the **aio** Driver module interface to the user, namely functions, commands, include files and tools.

The functions are described in UNIX man-page format and are organized in a functional order. Below is an index of the routines in the same order.

- **open** to open a channel
- **close** to close a channel
- **ioctl** to issue a control command
- **aioDrv** to install the **aio** driver
- **aioDevCreate** to add a **aio** device to the driver

Each of these calls returns a status code which signals the success of the call. A zero value always means "successful completion" while a negative value indicates an error. The value -1 stands for "general error" and origines in VxWorks while other values signal a specific error reason and are returned by the driver itself. Literals of all error codes can be found in the include file *aioErrors.h* and in [7].

3.2 Functions

As mentioned in the "User Manual" part of this document, the **aio** driver uses the standard VxWorks I/O system calls **open**, **close** and **ioctl** to interface to the user. These functions are described in the following sections with respect to their special meaning in the **aio** context. For more general information see the "VxWorks Programmers Guide" [1].

3.2.1 open

NAME

open - open a channel to a **aio** device

SYNOPSIS

```
#include "aio.h"
int open (char *deviceName, int openMode, int *status)
```

DESCRIPTION

This operation opens the device corresponding to *deviceName* in read-only mode or in one of the read/write modes described below. The read access is granted in all open modes. The argument *status* receives the completion status of the routine.

* *deviceName* specifies the device (e.g. "/aio0")

* *openMode* can be:

- -lcudrvOPEN_READONLY

  Read only mode.

- -lcudrvOPEN_SHARED

  Shareable read and write mode. Write access is granted to a defined number of tasks to use this open mode. Exceeding this number or requesting this mode for a device already opened in Exclusive mode returns an error.

- -lcudrvOPEN_EXCLUSIVE

  Exclusive read write mode. Write access is granted to the task using this open mode. Requesting this mode for a device already opened in Exclusive or Shared mode returns an error.

- -lcudrvOPEN_TEST

  Test read and write open mode. Write access is granted to the task using this option mode regardless of the status of the device. Requesting this mode for a device already opened in Test mode returns an error.

RETURN VALUES

* A positive *file descriptor* number, which must be used for subsequent **close** and **ioctl** operations.

* A value of -1 signals a general error. The specific error reason is provided in the argument *status* which can be

- -lcudrvERROR_INVALID_DEVICE

If the specified device name is invalid.

-lcudrvERROR_INVALID_OPEN_MODE

If the open mode is invalid.

-lcudrvERROR_NO_MORE_CHANNEL

No more channel is available.

-lcudrvERROR_ACCESS_CONFLICT

The requested open mode is conflicting with the open modes of already existing channels. This error is caused by one of the following reasons:

+ Shared mode requested and device already open in Exclusive mode.

+ Exclusive mode requested and the device is already open in Exclusive or Shared mode.

+ Test mode requested and the device is already open in Test mode.

* The zero value should never be returned.

EXAMPLE

```
int fd, status;
......
fd = open ("/aio0", lcudrvOPEN_READONLY, (int)&status);
if (fd <= 0)
  {
  /* error processing */
  ......
  }
else
  {
  /* normal processing */
  ......
  }
```

CAUTIONS

* Device names shall not be longer than 15 characters. They are defined by use of the function **aioDevCreate** at startup.

* This function is not queued by the driver, it always returns immediately with the file descriptor or an error code.

* The returned file descriptor can be shared by several applications.

* Applications can have the same device opened several times, using different file descriptors and with different open modes.

* As only one task can open a device for *Exclusive* mode at any one time, and the number of

channels is limited, applications should issue a **close** call if no more action on the device has to be performed.

* The type of the third parameter is declared as *int* in VxWorks. Therefore a cast is necessary.

VxWorks

* The VxWorks include file "**configAll.h**" contains two literals used by iosInit function: NUM_DRIVERS which defines the maximum number of drivers allowed, and NUM_FILES, the maximum number of simultanously open files.
* The VxWorks command "**iosDrvShow**" shows all drivers of the system and their basic I/O function entry-points. The command "**iosFdShow**" shows all currently used file descriptors, and "**iosDevShow**" (or "**devs**") all installed devices.

3.2.2 close

NAME

close - close a channel to a **aio** device

SYNOPSIS

```
#include "aio.h"
int close (int fileDescriptor)
```

DESCRIPTION

This operation closes a channel to an opened device and frees the file descriptor. *fileDescriptor* must correspond to one of those obtained previously be an open call.

RETURN VALUES

- lcudrvOK if successfully done.
- Other error codes returned by the driver are described in chapter 5.

EXAMPLE

```
int fd, status;
int closeError;
fd = open ("/aio0", lcudrvOPEN_TEST, (int)&status);
......
closeError = close (fd);
if (closeError != lcudrvOK)
  {
  /* error processing */
  ......
  }
else
  {
  /* normal processing */
  ......
  }
```

CAUTIONS

- This function is not queued by the driver, it always returns on of the above two status codes immediatly.

3.2.3 ioctl

NAME

ioctl - send a control command to a **aio** device

SYNOPSIS

#include "aio.h"

int ioctl (int fileDescriptor, int command, void *argument)

DESCRIPTION

This function commands the driver to perform the specified operation on the device. The commands are devided into read and write commands. Read commands are allowed in each open mode while write commands will be rejected if the channel was not previously opened in *Shared*, *Exclusive* or *Test* mode.

* *fileDescriptor* must correspond to one of those obtained previously by an open operation.
* *command* is a number, identifying the operation to be performed by the driver. Literals of all commands are provided in *aioCommands.h*. The following commands are supported:

Read commands:
- aioCMD_GET_BOARD_ID
- aioCMD_READ_VALUE
- aioCMD_GET_INPUT_CONFIG
- aioCMD_GET_OUTPUT_CONFIG
- aioCMD_GET_GAIN
- aioCMD_GET_TIMEOUT

Write commands:
- aioCMD_RESET
- aioCMD_WRITE_VALUE
- aioCMD_SET_INPUT_CONFIG
- aioCMD_SET_OUTPUT_CONFIG
- aioCMD_SET_GAIN
- aioCMD_ENABLE_EOC_INTERRUPT
- aioCMD_DISABLE_EOC_INTERRUPT
- aioCMD_AUTOCALIBRATION
- aioCMD_SET_TIMEOUT
- aioCMD_FREE_DEVICE
- aioCMD_WAIT

* *argument* is the address of the command argument, or NULL if no argument is used. For commands, needing multiple arguments it is the address of a data structure which contains those. All argument data structures are defined in the include file *aioCommands.h*.

RETURN VALUES

* lcudrvOK if the command is successfully transmitted and performed by the device.

- lcudrvERROR_ACCESS_CONFLICT if the write command can not be executed by use of a file descriptor previously obtained with a read-only open mode.
- lcudrvERROR_INVALID_ARGUMENT if command code is invalid.
- lcudrvERROR_TIMEOUT if access protection timed out.
- Other command specific values returned by the driver are described in chapter 5.

EXAMPLE

```
int fd, status;
int arg;
int ioctlError;
fd = open ("/aio0", lcudrvOPEN_TEST, (int)&status);
......
arg = 30;
ioctlError = ioctl (fd, aioCMD_SET_TIMEOUT, (int)&arg);
if (ioctlError != lcudrvOK)
  {
  /* error processing */
  ......
  }
else
  {
  /* normal processing */
  ......
  }
```

CAUTIONS

- This function is protected by a semaphore, and commands are queued by application *priorities* rather than by *First-in First-out*.
- A watchdog timer is implemented to ensure the permissible maximum execution delay of a command. This delay, expressed in ticks (currently there are 100 ticks in 1 second) and can be set/got at run-time by two additional commands: *aioCMD_SET/GET_TIMEOUT*.
- The type of the third parameter is declared as *int* in VxWorks. Therefore a cast is necessary.

3.2.4 aioDrv(3)

NAME

```
aioDrv - Install the AIO Device Driver
```

SYNOPSIS

```
#include "aio.h"
int aioDrv(int devices, int maxChannels, int timeout)
```

DESCRIPTION

```
This routine installs the AIO device driver. It is called on startup
and initializes the channel and device management. The routine must
be called before any I/O request to the driver and before installing
a AIO device.
```

RETURN VALUES

```
lcudrvOK                      if successful completion
lcudrvERROR_DRIVER_EXISTS     if driver already installed
lcudrvERROR_INVALID_ARGUMENT  if invalid channel count
lcudrvERROR_NO_MEMORY         if no memory for internal data
lcudrvERROR_NO_SEMAPHORE      if mutex-semaphore cannot be created
```

CAUTIONS

```
This routine must only be called once and before installing a AIO
device.
```

SEE ALSO

```
aioDevCreate(3)
```

```
- - - - - -
Last change:  27/09/94-08:09
```

3.2.5 aioDevCreate(3)

NAME

```
aioDevCreate - Add a AIO device to the AIO driver.
```

SYNOPSIS

```
#include "aio.h"
int aioDevCreate(char *deviceName,
                 int baseAddr,
                 int interruptNumber,
                 int interruptLevel,
                 int boardID)
```

DESCRIPTION

```
This function is called at startup once for each device to be
installed. It adds a device to the driver, making it available
for subsequent open operations.
```

RETURN VALUES

```
lcudrvOK                       if successful completion
lcudrvERROR_NO_DRIVER          if driver not yet installed
lcudrvERROR_INVALID_DEVICE     if invalid device name
aioERROR_INVALID_BOARD_ID      if invalid board ID
lcudrvERROR_INVALID_ARGUMENT   if no board is installed at baseAddr
lcudrvERROR_DEVICE_EXISTS      if device already installed
lcudrvERROR_NO_SEMAPHORE       if failed to assign semaphore
lcudrvERROR_TIMEOUT            if semaphore-timeout
```

CAUTIONS

```
The AIO driver must be installed before using this routine.
```

SEE ALSO

```
aioDrv(3)
```

```
- - - - - -
Last change:  27/09/94-08:09
```

3.3 IOCTL control commands

This section describes all control commands which can be invoked by use of the VxWorks **ioctl** function. The parameter *command* specifies which operation has to be performed by the driver while the parameter *argument* passes the address of an argument to the command handler. In cases where multiple arguments are needed the address of a structure which contains those is passed.

All command literals and argument data structures are defined in the include file *aioCommands.h*.

The supported commands can be divided into two groups: Read and Write Commands. Read Commands are allowed in each open mode, while Write Commands are allowed only in *Shared*, *Exclusive* or *Test* mode.

This section describes all control commands supported by the driver, namely their purpose, arguments and return codes.

3.3.1 Read Commands

- **aioCMD_GET_BOARD_ID - Get board ID**

  Returns the board ID.

  Argument: address of an integer value to which the board ID is written by the driver.

  Return Values:

  - **lcudrvOK** - successful completion

- **aioCMD_READ_VALUE - Read signal value**

  Returns the value of a specified input signal in Volts.

  Argument:address of an *aioSIGNAL* structure, providing fields for the

  - signal number (*.number* [integer]);
    Range: 0 to 15
  - value to be returned by this command (*.value* [real])
  - gain of the specified signal (*.gain* [integer])
  - configured range of the specified signal (*.config* [integer])

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_NOT_CONFIGURED** - specified signal is not configured
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range

- **aioCMD_GET_INPUT_CONFIG - Get input signal configuration**

  Returns the configuration value for a specified input signal.

  Argument:address of an *aioSIGNAL* structure, providing fields for the

  - signal number (*.number* [integer]);
    Range: 0 to 15
  - configuration value, which will be returned by this command (*.config* [integer]), where the value:

    - **aioCONFIG_2x10V** corresponds to the input range -10 V to +10 V
    - **aioCONFIG_10V** corresponds to the input range 0 V to +10 V
    - **aioCONFIG_2x5V** corresponds to the input range -5 V to +5 V
    - **aioCONFIG_5V** corresponds to the input range 0 V to +5 V

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range

- **aioCMD_GET_OUTPUT_CONFIG - Get output signal configuration**

  Returns the configuration value of a specified output signal.

  Argument:address of an *aioSIGNAL* structure, providing fields for the

  - signal number (*.number* [integer]);
    Range: 0 to 1
  - configuration value, which will be returned by this command (*.config* [integer]), where the value:
    - **aioCONFIG_2x10V** corresponds to the input range -10 V to +10 V
    - **aioCONFIG_10V** corresponds to the input range 0 V to +10 V
    - **aioCONFIG_2x5V** corresponds to the input range -5 V to +5 V
    - **aioCONFIG_5V** corresponds to the input range 0 V to +5 V

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range

- **aioCMD_GET_GAIN - Get the gain**

  Returns the gain of a specified input signal.

  Argument:address of an *aioSIGNAL* structure, providing fields for the

  - signal number (*.number* [integer]);
    Range: 0 to 15
  - gain of the specified signal, which will be returned by this command. (*.gain* [integer])

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range

- **aioCMD_GET_TIMEOUT - Get the driver timeout**

  Returns the timeout of the driver in ticks, i.e. the maximum time the user's task has to wait for device access before it will be timeout.

  Argument:address of an integer, recieving the timeout value

  Return Values:

  - **lcudrvOK** - successful completion

3.3.2 Write Commands

- **aioCMD_RESET - Reset the Device**
  This command sets all output signals to the value 0 and the gain of all input signals to 1.
  Argument:none
  Return Values:
  - **lcudrvOK** - successful completion

- **aioCMD_WRITE_VALUE - Write value to signal**
  This command writes a value to the specified output signal.
  Argument:address of an *aioSIGNAL* structure, containing the
  - signal number (*.number* [integer]);
    Range: 0 to 1
  - the value to be set in Volts (*.value* [real])

  Return Values:
  - **lcudrvOK** - successful completion
  - **aioERROR_NOT_CONFIGURED** - specified signal is not configured
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range
  - **aioERROR_INVALID_SIGNAL_VALUE** - signal value out of range

- **aioCMD_SET_INPUT_CONFIG - Configure input signal**
  With this command the signal range of an input signal is specified. Note that the configuration is defined by jumper settings on the board [4], it is not programmable. This command is only to inform the driver about the jumper selection.
  Argument:address of an *aioSIGNAL* structure, containing the
  - signal number (*.number* [integer])
    Range: 0 to 15
  - configuration value, to be set (*.config* [integer]), where the value:
    - **aioCONFIG_2x10V** corresponds to the input range -10 V to +10 V
    - **aioCONFIG_10V** corresponds to the input range 0 V to +10 V
    - **aioCONFIG_2x5V** corresponds to the input range -5 V to +5 V
    - **aioCONFIG_5V** corresponds to the input range 0 V to +5 V

  Return Values:
  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range
  - **aioERROR_INVALID_SIGNAL_CONFIG** - invalid signal configuration value

- **aioCMD_SET_OUTPUT_CONFIG - Configure output signal**

  This command sets the output signal range. Note that the configuration is defined by jumper settings on the board [4], it is not programmable. This command is only to inform the driver about the jumper selection.

  Argument:address of an *aioSIGNAL* structure, containing the

  - signal number (*.number* [integer])
    Range: 0 to 1
  - configuration value, to be set (*.config* [integer]), where the value:
    - **aioCONFIG_2x10V** corresponds to the input range -10 V to +10 V
    - **aioCONFIG_10V** corresponds to the input range 0 V to +10 V
    - **aioCONFIG_2x5V** corresponds to the input range -5 V to +5 V
    - **aioCONFIG_5V** corresponds to the input range 0 V to +5 V

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range
  - **aioERROR_INVALID_SIGNAL_CONFIG** - invalid signal configuration value

- **aioCMD_SET_GAIN - Set gain of an input signal**

  This command sets the gain for a specified input signal. The gain factor amplifies the input signal by the given value, before it is converted to a digital value. This is to increase the resolution of the channel. The driver then divides the converted value with the gain factor to provide the true value in Volts.

  Argument:address of an *aioSIGNAL* structure, containing the

  - -signal number (*.number* [integer])
    Range: 0 to 15
  - -gain value to be set (*.gain* [integer]);
    Values allowed: 1, 10, 100, 500

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_SIGNAL_NUMBER** - signal number out of range
  - **aioERROR_INVALID_SIGNAL_GAIN** - gain value to be set not allowed

- **aioCMD_ENABLE_EOC_INTERRUPT - Enable interrupt on EOC**

  This command enables interrupt response on EOC and disables the polling procedure.

  Argument: none

  Return Values:

  - **lcudrvOK** - successful completion

- **aioCMD_DISABLE_EOC_INTERRUPT - Disable interrupt on EOC**

  This command disables interrupt response on EOC and enables the polling procedure.

Argument: none

Return Values:

- **lcudrvOK** - successful completion

- **aioCMD_AUTOCALIBRATION - Set to autozero mode**

  This command causes the board to perform autozero of each input.

  Argument: none

  Return Values:

  - **lcudrvOK** - successful completion

- **aioCMD_SET_TIMEOUT - Set driver timeout value**

  This command sets the timeout of the driver. The value must be specified in system clock ticks. The driver timeout specifies how long a control command is waiting for access to the device before it is timed out.

  Argument: address of an integer, containing the timeout value in system clock ticks [1]

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_INVALID_TIMEOUT** - timeout value out of range

- **aioCMD_FREE_DEVICE - Free the Device**

  This command frees the access to the driver when it is blocked by a task which has opened it in exclusive mode and terminated without closing or hang-up, etc. In this case it is not possible to give any write command to the driver from other tasks. The *'free device'* command cleans this situation by closing the exclusive channel. The intention is that this command should be used manually by an operator only and not from other tasks which just want to give write commands.

  Argument: none

  Return Values:

  - **lcudrvOK** - successful completion
  - **aioERROR_NO_EXCLUSIVE** - no exclusive channel was opened

1. Note: Before driver version 1.3 the argument was passed by value, not by reference through a pointer. However, the User Manual always stated it this way. Now code and manual are consistent.

3.4 Include files

Applications using the **aio** must include **aio.h**. The **aio.h** include file includes everything required to use the procedures described in the previous section.

- The needed VxWorks header files
- "*aioErrors.h*" which defines literals of **aio** Driver errors
- "*aioCommands.h*" which defines literals of the **aio** Driver commands and the data structures for multiple argument commands as used with the *ioctl* procedure.

3.5 Tools

The **aio** Driver includes a number of tools which can be used from the VxWorks shell. These tools are intended to be used by a system manager to ease maintenance and troubleshooting.

- **aioVersion** ()
  to print the version number of the **aio** driver
- **aioDevShow** ()
  to print the list of devices controlled by the **aio** driver with their respective base addresses
- **aioPrintRegs** (unitNumber)
  prints the device register contents in readable format, the device is specified by its unit number
- **aioPrintDevice** (unitNumber)
  prints the status of a **aio** device in readable format, the device is specified by its unit number
- **aioPrintError** (errorCode)
  decodes an error code to the corresponding text
- **aioPrintCommand** (commandCode)
  decodes a command code to the corresponding text

All the tools require that the **aio** driver is already installed and at least one device is added.

All output is made to the standard output, i.e. either to the system console or to a host terminal. Therefore the output can be redirected to a file for logging purposes.

All tools return a status code, where *lcudrvOK* means "successful completion" and *lcudrvERROR* means "ERROR".

More detailed information for the usage of the tools can be accessed through the on-line man pages.

4 INSTALLATION

The installation of the **aio** Driver is done at system start-up time by script files and shall not be changed at run-time. It is composed by the installation of the driver code, the installation of the devices for that driver and the connection of the device to the driver.

4.1 Installation Prerequisites

The following hardware and software prerequisities must be fulfilled for a successful installation of the driver.

4.1.1 Hardware Requirements

- a standard 6U VMEbus chassis with bus backplane and power supply
- Motorola MVME167 CPU board, any version
- VMIVME-3111 Analog I/O board, any version
- Ethernet network
- LCU console (terminal attached to the serial line of the VMEbus CPU)

4.1.2 Software Requirements

- VxWorks 5.1 operating system
- **lcudrv** for common driver functions, version 1.3
- **lculog** for internal logging, version 1.4
- the **aio** module installed on a host machine

4.2 Building the Software

The **aio** module is delivered as a UNIX tar file, either on disk or on tape. The file expands in a directory structure as defined in [3].

```
aio/
      |
      |------include/                  (include files)
      |------src/                      (source files and Makefile)
      |------object/                   (target dir for make)
      |------doc/                      (target dir for make)
      |------test/                     (test sources and includes)
      |------bin/                      (target dir for make)
      |------man/                      (target dir for make)
```

If it is required to build or rebuild the software the Makefile provided shall be used. To use the Makefile a number of environment variables must be defined. An example to set up the environment for VxWorks and the GNU-C-compiler can be found in [6].

Before using the make files the user should make sure that GNU make is defined before the make supplied by the vendor in the search path. This can be checked by issuing the UNIX command **which make**. As defined in [3] VLT software should be built using GNU make in order to avoid discrepancies between different vendors implementation of *make*. To build the software follow the procedures below:

1. Move to **aio src** directory (cd ./aio/src)
2. Type **'make clean all man install'** to
   a. remove everything which can be made, thus enforcing recompilation of the entire code.
   b. compile and link everything. The result, the VxWorks object-module ***aio*** and the installation script ***aioInstall*** will be stored in directory ***aio/bin***.
   c. move the executables and include files required by external modules to their target directories.

4.3 Installation Procedure

This section describes the steps to install the **aio** driver and the first device. A script file ***aioInstall*** is prepared to perform these steps. The utility ***vltMan*** can be used to access man-page informations for the aio-functions.

1. Load the **aio** driver code to the target system
2. The driver is installed by invoking the function *aioDrv()* with the following parameters (see Reference):
   - the maximum number of supported **aio** devices
   - the maximum number of open channels
   - the access timeout in ticks
3. A device is installed by invoking the function *aioDevCreate()* with the following parameters (see Reference):
   - the device name: **"/aio<*i*>"** where **<*i*>** is the device unit number, starting with 0 and incrementing by one for each additional device.
   - the base address of the device in the CPU's address space (see hardware manual)
   - the interrupt vector number (see hardware manual)
   - the interrupt level (see hardware manual)
   - the board ID
4. Initializes driver internal working areas
5. Connects the device to the driver (enabling interrupts, defining vector number, connecting Driver ISR).

An installation script file *aioInstall* is prepared in directory <*drvRoot*>/aio/bin which performs the above described installation procedure with default parameters. It is strongly recommended to use this script only to install the **aio** driver. A listing of the installation script is provided in the following section. To install the **aio** driver do the following from the VxWorks shell of the target system:

- *putenv "VWROOT=<drvRoot>")* to set the environment variable **VWROOT**; <*drvRoot*> is the root directory into which the module **aio** is installed
- invoke the installation script <*drvRoot*>**/bin/aioInstall**

4.4 Installation Script Example

The following installation script shall be called at system start-up time either included in a system start-up file or called from here.

```
#************************************************************************
#    NAME
#    aioInstall - install the AIO driver and device on target LCU
#
#    SYNOPSIS
#    aioInstall
#
#    DESCRIPTION
#    This script is used to load the AIO driver module object from the host
#    to the LCU target system and to configure the driver. This script must
#    run on the LCU target system under the VxWorks shell.
#
#    FILES
#    aio must exist on the host system with group read access
#
#    ENVIRONMENT
#    The environment variable VWROOT can be set to the LCU drivers
#    directory using the VxWorks function putenv("VWROOT=.......").
#    The driver module is taken from $VWROOT/bin.
#    If VWROOT is undefined then ./bin is used.
#
#    RETURN VALUES
#    none
#
#    CAUTIONS
#    This script must be run only once, at LCU system start-up time
#
#    EXAMPLES
#    vxworks> cd "/vlt/vw"
#    vxworks> < bin/aioInstall
#
#    SEE ALSO
#    AIO Driver User Manual
#
#------------------------------------------------------------------------
#

# load driver object
cd getenv("VWROOT")
cd "bin"
ld < aio
cd ".."

# install and configure the AIO driver
aioDrv(1,10,100)

# install and configure the AIO device
aioDevCreate("/aio0", 0xffff8000, 65, 1, 0x0d)
```

4.5 Installation Verification

Functions performed during installation phase always log OK or ERROR messages to the console.

The tools described in a previous section can be used to test that the driver and the device have been installed correctly. From the VxWorks shell issue the following commands:

- aioVersion
  This should print the expected version number on the console.
- aioPrintDevice(0)
  This prints status and configuration information about the installed **aio** device. This information should be consistent with the configured parameters.
- aioPrintRegs(0)
  This should dump the register contents of the specified device unit on the console. The values are from minor interest but the fact that this command causes no bus error indicates that the device registers are probably mapped correctly.

5 ERROR MESSAGES AND RECOVERY

The following list shows all possible errors which can be returned by the **aio** driver. There are three types of errors:

- VxWorks system errors
- errors from the common driver library
- errors from the **aio** driver

The errors are shown in alphabetical order by their literals as defined in *aioErrors.h* (**aio** specific errors with the prefix *aio*) and and in [7] (common errors with the prefix *lcudrv*) together with a description of their meaning and reason. Errors corresponding to each function are depicted in the section 3.2.

Note: Additional errors, not described here can be returned when the VxWorks I/O system rejects a call before invoking the driver.

5.1 Common Driver Errors

The following list shows common driver errors as defined in [7].

- **lcudrvOK**
  Indicates successful completion.
- **lcudrvERROR**
  Indicates general error.
- **lcudrvERROR_ACCESS_CONFLICT**
  The open mode conflicts with the open modes of already existing channels or the requested action is not allowed with this open mode.
- **lcudrvERROR_CHANNEL_NOT_OPEN**
  The requested channel is not open.
- **lcudrvERROR_DEVICE_EXISTS**
  Attempt to create an already existing device a further time.
- **lcudrvERROR_DRIVER_EXISTS**
  Attempt to install an already existing driver a further time.
- **lcudrvERROR_INVALID_ARGUMENT**
  The value of one of the arguments was not in a valid range.
- **lcudrvERROR_INVALID_COMMAND**
  The ioctl command code matches no valid command.
- **lcudrvERROR_INVALID_DEVICE**
  The device name does not specify a valid device.
- **lcudrvERROR_INVALID_OPEN_MODE**
  The requested open mode is unknown.
- **lcudrvERROR_NO_CHANNEL**
  No more channel to a device is available or invalid channel number.
- **lcudrvERROR_NO_DRIVER**
  Attempt to create a device without having the driver installed yet.
- **lcudrvERROR_NO_MEMORY**
  Not enough memory available to allocate internal data structures.

- **lcudrvERROR_NO_MORE_CHANNEL**
  No more channel to a device can be opened.
- **lcudrvERROR_NO_SEMAPHORE**
  Failed to create the access protection semaphore.
- **lcudrvERROR_TIMEOUT**
  The call is timed-out because access to the device was pending for longer than the *driver timeout* parameter.

5.2 aio specific Errors

- **aioERROR_INVALID_BOARD_ID**
  The specified board ID is invalid.
- **aioERROR_INVALID_IR_LEVEL**
  The specified interrupt level is out of range.
- **aioERROR_INVALID_IR_NUMBER**
  The specified interrupt number is out of range.
- **aioERROR_INVALID_SIGNAL_CONFIG**
  The specified signal configuration value is out of range.
- **aioERROR_INVALID_SIGNAL_GAIN**
  The specified gain value is not allowed.
- **aioERROR_INVALID_SIGNAL_NUMBER**
  The specified signal number is out of range.
- **aioERROR_INVALID_TIMEOUT**
  The specified timeout value is out of range.
- **aioERROR_NOT_CONFIGURED**
  The signal to be read/written is not configured.
- **aioERROR_NOT_SETTLED**
  The signal to be read has not settled within the maximum timeout.
- **aioERROR_NO_EXCLUSIVE**
  There is no exclusive channel open on a free device command.

___oOo___