SIS1100/3100
LINUX Driver

Programmers Manual

SIS GmbH
Harksheider Str. 102A
22399 Hamburg
Germany

Phone: ++49 (0) 40 60 87 305 0
Fax: ++49 (0) 40 60 87 305 20

email: info@struck.de
http://www.struck.de

Version: 1.30pre as of 12.07.04

Revision Table:

Revision	Date	Modification
0.1	09.10.01	Generation
1.0	25.10.01	First official release
1.01	17.01.02	Minor bug fixes
1.02	07.02.02	Bug fix in open file rights
1.03	22.02.02	Bug fix in s3100_sdram_read/write return code
1.10	10.06.02	2e routines (require SIS3100 firmware major 1 minor 6 single word DMA routines, V1.0 driver
1.11	13.06.02	Reference to driver FAQ page added
1.12	19.08.02	Bug fix in mknod example for multiple modules
1.13	27.02.03	Add configure for driver versions 1.0 and higher
1.20	09.11.03	1.0 and 1.3 driver installation
1.21	29.01.04	Hint for devfs users
1.30		V2.xx driver

1 Table of contents

1 Table of contents 3
2 Introduction 5
2.1 Acknowledgements 5
2.2 Copyright and Liability 5
3 Concept 6
4 Getting started 7
4.1 Hardware Installation 7
4.1.1 Installation of SIS1100 7
4.1.2 Verify installation of SIS1100 8
4.1.3 Installation of SIS3100 8
4.2 Fibre Installation and Handling 9
4.2.1 Fibre cabling/installation 9
4.2.2 Protection against dust and mechanical stress 9
4.3 LINUX Driver installation (below 2.0x) 10
4.3.2 LINUX Driver load at boot time 13
4.4 Compilation of examples 13
4.5 2.0x driver, SIS3100 VME and SIS5100 CAMAC support 14
4.5.1 SIS1100 driver 14
4.5.2 SIS3100 library lib_sis3100.a 16
4.5.3 SIS5100 library lib_sis5100.a 16
4.5.4 VME examples 17
4.5.5 CAMAC examples 17
5 Subroutines 18
5.1 Summary of routines 18
5.2 Routines to handle VME, SDRAM and SHARC environment 20
5.2.1 open 20
5.2.2 close 20
5.3 SIS3100 control read/write routines 21
5.3.1 s3100_control_read 21
5.3.2 s3100_control_write 21
5.4 VME infrastructure routines(s) 22
5.4.1 vmesysreset 22
5.5 VME single word read/write routines 23
5.5.1 vme_A16D8_read VME A16 D8 read 23
5.5.2 vme_A16D8_write VME A16 D8 write 23
5.5.3 vme_A16D16_read VME A16 D16 read 24
5.5.4 vme_A16D16_write VME A16 D16 write 24
5.5.5 vme_A16D32_read VME A16 D32 read 25
5.5.6 vme_A16D32_write VME A16 D32 write 25
5.5.7 vme_A24D8_read VME A24 D8 read 26
5.5.8 vme_A24D8_write VME A24 D8 write 26
5.5.9 vme_A24D16_read VME A24 D16 read 27
5.5.10 vme_A24D16_write VME A24 D16 write 27
5.5.11 vme_A24D32_read VME A24 D32 read 28
5.5.12 vme_A24D32_write VME A24 D32 write 28
5.5.13 vme_A32D8_read VME A32 D8 read 29
5.5.14 vme_A32D8_write VME A32 D8 write 29
5.5.15 vme_A32D16read VME A32 D16 read 30
5.5.16 vme_A32D16_write VME A32 D16 write 30
5.5.17 vme_A32D32_read VME A32 D32 read 31
5.5.18 vme_A32D32_write VME A32 D32 write 31
5.6 VME single word DMA routines 32
5.6.1 vme_A24DMA_D32_read 32
5.6.2 vme_A24DMA_D32_write 32
5.6.3 vme_A32DMA_D32_read 33
5.6.4 vme_A32DMA_D32_write 33
5.6.5 vme_A32DMA_D32FIFO_read 34

5.7 VME block read/write routines 35
5.7.1 vme_A24BLT32_read VME A32 BLT32 block read 35
5.7.2 vme_A24BLT32FIFO_read VME A24 BLT32 block read from FIFO 36
5.7.3 vme_A24BLT32_write VME A24 BLT32 block write 36
5.7.4 vme_A24MBLT64_read VME A24 MBLT64 block read 37
5.7.5 vme_A24MBLT64FIFO_read VME A24 MBLT64 block read from FIFO 37
5.7.6 vme_A24MBLT64_write VME A24 MBLT64 block write 38
5.7.7 vme_A32BLT32_read VME A32 BLT32 block read 38
5.7.8 vme_A32BLT32FIFO_read VME A32 BLT32 block read from FIFO 39
5.7.9 vme_A32BLT32_write VME A32 BLT32 block write 39
5.7.10 vme_A32MBLT64_read VME A32 MBLT64 block read 40
5.7.11 vme_A32MBLT64FIFO_read VME A32 MBLT64 block read from FIFO 40
5.7.12 vme_A32MBLT64_write VME A32 MBLT64 block write 41
5.8 2e VME block transfer routines 42
5.8.1 vme_A32_2EVME_read 42
5.8.2 vme_A32_”2EVMEFIFO_read 42
6 SDRAM access routines 43
6.1 SDRAM single word read/write routines 43
6.1.1 s3100_sdram_read 43
6.1.2 s3100_sdram_write 43
7 DSP access routines 44
7.1 DSP single word read/write routines 44
7.1.1 s3100_sharc_read 44
7.1.2 s3100_sharc_write 45
8 Return/Error codes 46
9 Index 47

2 Introduction

As we are aware, that no manual is perfect, we appreciate your feedback and will try to incorporate proposed changes and corrections as quickly as possible. The most recent version of this manual can be obtained from http://www.struck.de/linux1100.htm. The current version of the driver can be downloaded from http://www.struck.de/linux1100.htm also. Have a look at the http://www.struck.de/linux1100faq.htm in case of problems with driver installation.

2.1 Acknowledgements

The SIS1100/3100 is a joined development between the ZEL department of the FZ Jülich and SIS GmbH. The SIS1100 is manufactured by SIS under license of the FZ Jülich. The underlying driver for the high level calls were written by Dr. Peter Wüstner from the ZEL department of the FZ Jülich.

2.2 Copyright and Liability

```
* Copyright (c) 2001
* Peter Wuestner and Matthias Drochner ZEL FZ Juelich  All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the above copyright
*    notice, this list of conditions, and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*    notice, this list of conditions and the following disclaimer in the
*    documentation and/or other materials provided with the distribution.
*
* THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
*/
```

3 Concept

The LINUX driver for the SIS1100/3100 PCI to VME interface consists basically of three layers.

The lowest level of the driver is in charge of the actual communication with the Gigabit link hardware over PCI. It uses the link protocol that was implemented on the SIS1100 and the SIS3100.

The second layer of the driver makes use of the first layer through IOCTL commands.

The third layer of the driver are high level routines, which are the actual contents of this manual. The main focus of the routines is readability of VME sequences without the knowledge of the actual content of a variable. An example is given below.
The current implementation uses separate read and write routines e.g.:

```
int vme_A32D32_write(int dev, u_int32_t vme_adr, u_int32_t vme_data );
int vme_A32D32_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data );
```

instead of a combination where the status of rw has to be known:

```
int vme_A32D32(int dev, u_int32_t vme_adr, u_int32_t* vme_data, int rw );
```

The actual implementation of the third layer makes use of the functionality provided by the ioctl's of the second layer, like illustrated with the A32 D32 read below.

```
int vme_A32D32_read(int p, u_int32_t vme_adr, u_int32_t* vme_data )
{
struct sis1100_vme_req req;

  req.size=4; /* driver does not change any field except data */
  req.am=0x9; /* "" */
  req.addr= vme_adr;
  if (ioctl(p, SIS3100_VME_READ, &req)<0)  return req.error ;
  *vme_data = req.data;
  return 0 ;
}
```

The end user may want to use the source code of these high level commands to design routines, which are more suited to the C calling nomenclature of a given experiment, or which combine a number of calls at the cost of additional variables and harder readability (as shown above).
The LINUX driver for the SIS1100/3100 is capable of handling multi interfaces in one PC.

4 Getting started

4.1 Hardware Installation

Many users of the SIS1100/3100 PCI to VME interface will not have to go through the complete documentation of the card combination unless they want to write custom driver software or have a closer look at the underlying protocol. We recommend to start with the following steps to get going:

- go to section installation of SIS1100 in this document
- go to section installation of SIS3100 in this document
- go to section fibre installation and handling in this document
- go to section getting started in the LINUX driver documentation, compile and install the driver and the provided example/test code.
- modify one of the examples to match your first minimum application (like switch on/off a user LED on a VME slave, or check VME access on a VME display).
- start implementing a real application (referring to the driver manual for a comprehensive list of implemented calls)

4.1.1 Installation of SIS1100

Although the SIS1100 card (consisting of the SIS1100-CMC carrier card and the SIS1100-OPT Gigabit link CMC) has several jumpers, no modification of jumper settings prior to installation will be required.
Following steps are required to install the SIS1100:

- power down the computer prior to installation of the SIS1100
- open the computer housing
- install the SIS1100 in a PCI slot
- make sure that the card is properly seated
- secure the front panel bracket with a screw
- close the computer housing
- go to section installation of SIS3100

Note: Misalignment of contacts of the SIS1100 can result in serious damage of your computer. Please make sure, that the card is properly seated and secured with the front panel screw.

4.1.2 Verify installation of SIS1100

With the comand **`pcitweak -l`** or with comand **lspci** you can verifiy, that the card was detected properly under LINUX. Pcitweak or lspci has to be run as root, example output is shown below. The SIS1100-CMC card will be reported under chip 1796,0001 (i.e. vendor Id. 0x1796) and card 1796,1100.

```
mki@mki:~ > su
Password:
root@mki:/home/mki > pcitweak -l
PCI: Probing config type using method 1
PCI: Config type is 1
PCI: PCI scan (all values are in hex)
PCI: 00:00:0: chip 8086,7180 card 0000,0000 rev 03 class 06,00,00 hdr 00
PCI: 00:01:0: chip 8086,7181 card 0000,0000 rev 03 class 06,04,00 hdr 01
PCI: 00:07:0: chip 8086,7110 card 0000,0000 rev 01 class 06,01,00 hdr 80
PCI: 00:07:1: chip 8086,7111 card 0000,0000 rev 01 class 01,01,80 hdr 00
PCI: 00:07:2: chip 8086,7112 card 0000,0000 rev 01 class 0c,03,00 hdr 00
PCI: 00:07:3: chip 8086,7113 card 0000,0000 rev 01 class 06,80,00 hdr 00
PCI: 00:0f:0: chip 10b7,9050 card 0000,0000 rev 00 class 02,00,00 hdr 00
PCI: 00:12:0: chip 1796,0001 card 1796,1100 rev 01 class 07,80,00 hdr 00
PCI: 01:01:0: chip 1023,9750 card 1023,9750 rev f3 class 03,00,00 hdr 00
PCI: End of PCI scan
root@mki:/home/mki >
```

4.1.3 Installation of SIS3100

The SIS3100 VME sequencer has a number of jumpers to configure part of its functionality. For the standard single SIS3100/crate installation you should be fine with the factory default setting which includes:

- VME system controller /16 MHz clock enabled
- VME slave disabled
- Power on reset results in VME SYSRESET
- FPGA reset results in VME SYSRESET
- NIM Reset Input results in FPGA reset (where I/O option is installed)

Please refer to the chapter jumpers if you would like to alter these settings.

Following steps are required to install the SIS3100 in the VME crate:

- power down the VME crate (unless you have a VME64x backplane)
- install the SIS3100 in slot 1 of the crate
- make sure, that the card is properly seated
- go to section fibre installation and handling

4.2 Fibre Installation and Handling

4.2.1 Fibre cabling/installation

The connection between the SIS1100 card and the SIS3100 board is made with two optical fibres. The length of the fibres is limited to 450 m (1300 feet) with the standard multimode lasers and fibres. Standard shipments come with a duplex fibre with LC connectors on both ends. . The fibres are mounted in the duplex housing/latch in a fashion, that sending and receiving fibre are crossed. Before installation of the duplex fibre the small white protection caps have to removed carefully from the four ends of the fibres. After removal of the black elastic protection plugs from the link media of the SIS1100 and SIS3100 the twin LC connector assemblies can be connected to the link media. You will hear a clicking sound when the two latches of the connector come into place.
Proper optical connection can be verified by the green link LED (L) on the front panel of the SIS3100 once the SIS1100 and 3100 have been powered up. The green link upstream (LU) and link downstream (LD) LEDs can be used to track down a problem to one of the two fibres when the L LED remains off.
Push down the grey button (with the two letters A and B on it) on the twin LC connector assembly to release the two latches and to disconnect the fibre from the link medium.

Note: Please retain the protective caps of the fibres and the link media for later use (see below)

4.2.2 Protection against dust and mechanical stress

You should use the dust caps, that come with your SIS1100/3100 shipment, to protect unused fibres and optical link media against dust. Dust or dirt can block the optical path or reduce transmission. The four small white caps should be plugged carefully onto the fibre ends of all fibres, the two black elastic plugs should be inserted into the optical link media of the SIS1100 and the SIS3100.
Fibres should be installed with a minimum bending radius of 10 cm (4 inches) and protected from accidental mechanical damage (step, office/lab chairs, ...).

4.3 LINUX Driver installation (below 2.0x)

The LINUX operating system is subject to continuous evolution. Part of the changes between different Kernel revisions are incompatible unfortunately. In the transition between 2.4 and 2.6 kernels and the use of 2.6 features in later 2.4 kernels even the number of parameters in the call to subroutines is changed in some cases. The current SIS1100/3100 driver distribution consists of the V1.0 and the V1.3 driver to take this into account (you may want to erase the directory which is not used on your machine after installation).

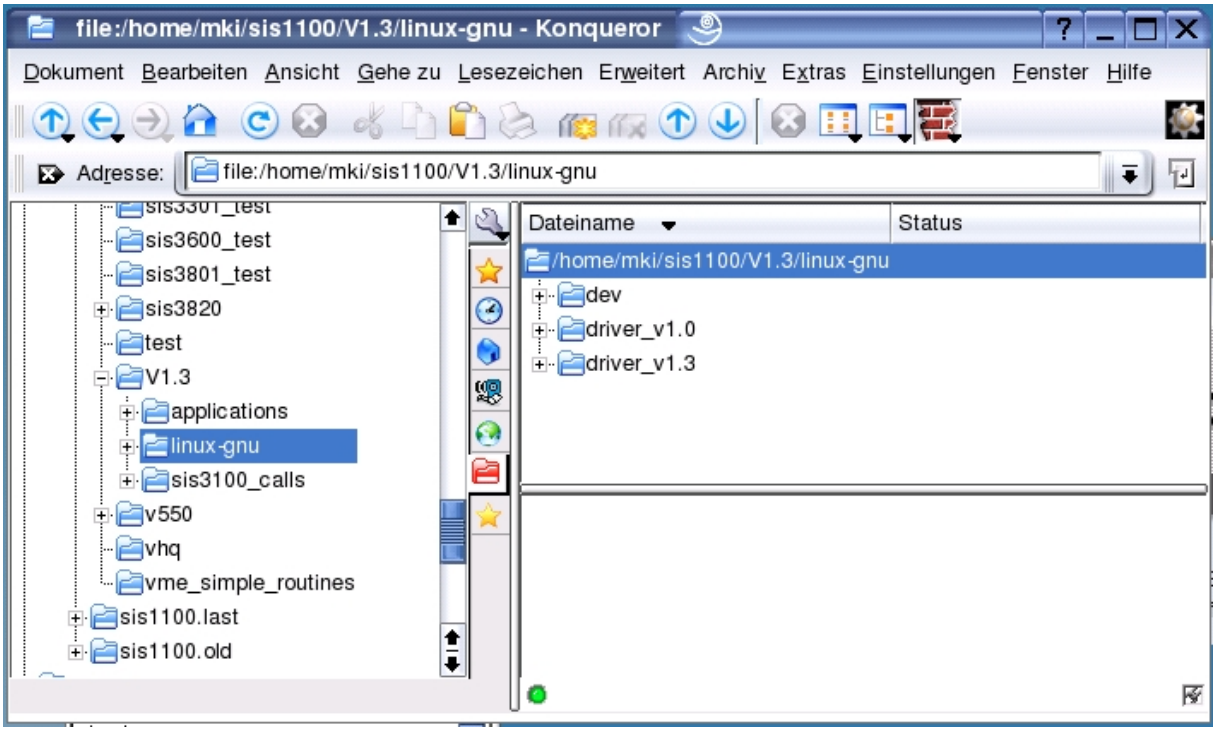
Kernel	Driver
2.4.4 – 2.4.18 (and slightly beyond?)	1.0
about 2.4.21 onwards	1.3
2.4.? and 2.6.? with SIS5100CAMAC support	2.0x

Kernel revisions below 2.4.4 are not supported

Unpack the driver to the directory of your choice (mydir)

```
tar -xzf sis1100.tar.gz
```

A screendump (with mydir=/homemki/sis1100) is shown below. Change to the directory `linux-gnu` and further down into the directory of the driver of your choice. Compile the driver as described below.



driver_v1.0

```
./configure
make depend
make
insmod sis1100.o (as root)
```

driver_v1.3

```
make
insmod sis1100.o (as root)
```

4.3.1.1 Single SIS1100/3100 installation

With a single SIS1100 installed you will have three devices related to the SIS1100/3100 PCI to VME interface:

```
# cat /proc/devices |grep SIS*
252 SIS3100sharc
253 SIS3100sdram
254 SIS1100
```

`mknod /tmp/sis1100 c 254 0` (as root, if above returned number for the SIS1100 was 254)

If you want to use the SDRAM option:

```
mknod /tmp/sis3100sdram c 253 0
```

If you want to use the SHARC DSP option:

```
mknod /tmp/sis3100sharc c 252 0
```

Note: The major device Id. may change if the machine is rebooted, hence it may make sense to handle the required steps with a script (like the load_module script, that can be found in the V1.3 driver directory)

Change the file/node attributes if you want to give VME access to all users (as illustrated with the VME device in the line below:

```
chmod 666 /tmp/sis1100
```

With device file system:

You should see the directory /dev/sis1100

```
0 -> vme Interface
0sd -> SDRAM
0sh -> SHARC

~$ ls -l /dev/sis1100/
total 0
crw-rw-rw- 1 root root 8, 1 Jan 1 1970 0
crw-rw-rw- 1 root root 8, 2 Jan 1 1970 0sd
crw-rw-rw- 1 root root 8, 3 Jan 1 1970 0sh
```

In case you do not see the /dev/sis1100 you may not have the devfs mounted, try (as root):

```
mkdir /devices
mount -t devfs none /devices
```

4.3.1.2 Multi SIS1100/3100 installation

The selection of one SIS1100 of multi of them is controlled by the **minor** number

selection of the first SIS1100:

```
mknod /tmp/sis1100 c 254 0
```

selection of the second SIS1100:

```
mknod /tmp/sis1100_2 c 254 1
```

Note: the driver can be unloaded with the command `rmmod sis1100`

4.3.2 LINUX Driver load at boot time

Copy the driver (sis1100.o) to the new directory `/lib/modules/`uname -r`/vme`
An entry of the form
`/sbin/insmod /lib/modules/`uname -r`/vme/sis1100.o`
has to be added to the file:
`/etc/rc.d/boot.local`

Note: `uname -r` returns the kernel release (2.4.4 e.g)

4.4 Compilation of examples

The examples that come with the driver can be compiled as described below.

Change to the directory `mydir/sis1100/test`
```
make depend
make
```

The examples and the Makefile in the `mydir/sis1100/test` directory should be a good starting point for your software development.

4.5 2.0x driver, SIS3100 VME and SIS5100 CAMAC support

The compressed tar file is generated from the directory /home/vme.
If you unpack it to the same directory on your machine you should be able to use all Makefiles without modification.

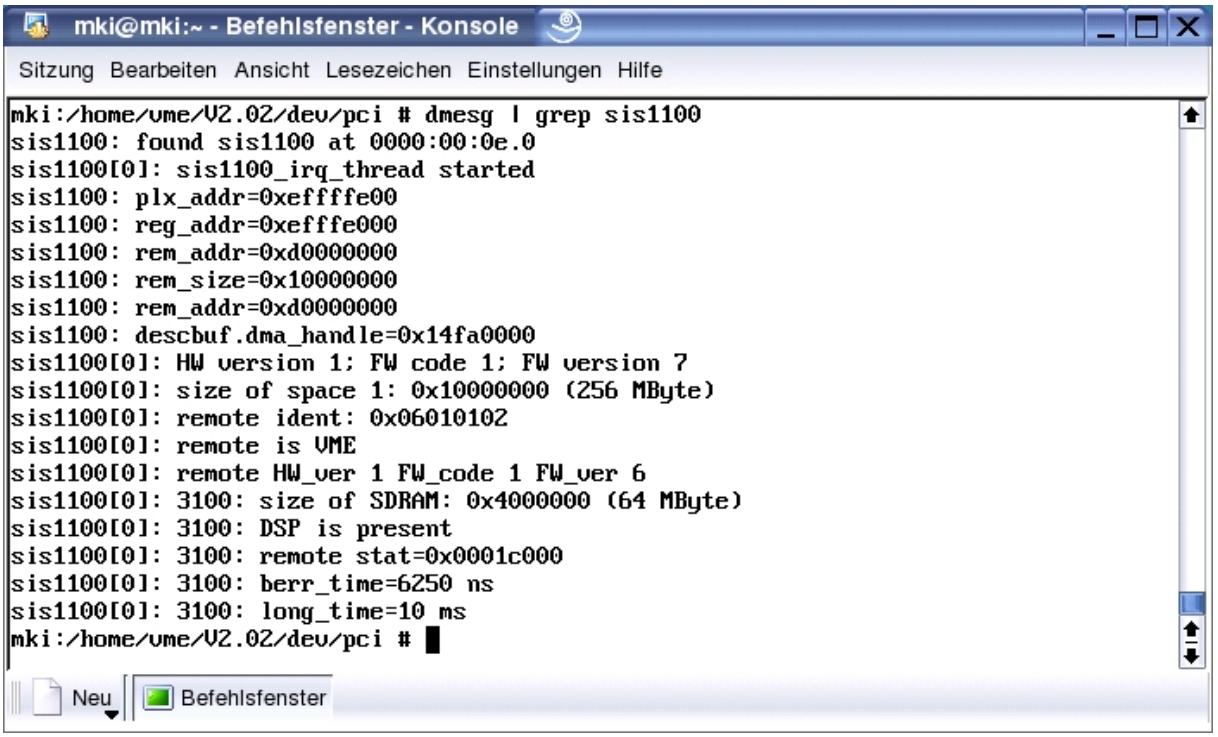
4.5.1 SIS1100 driver

```
cd /home/vme
tar -xzf V2.02.tar.gz
cd dev/pci
make
```

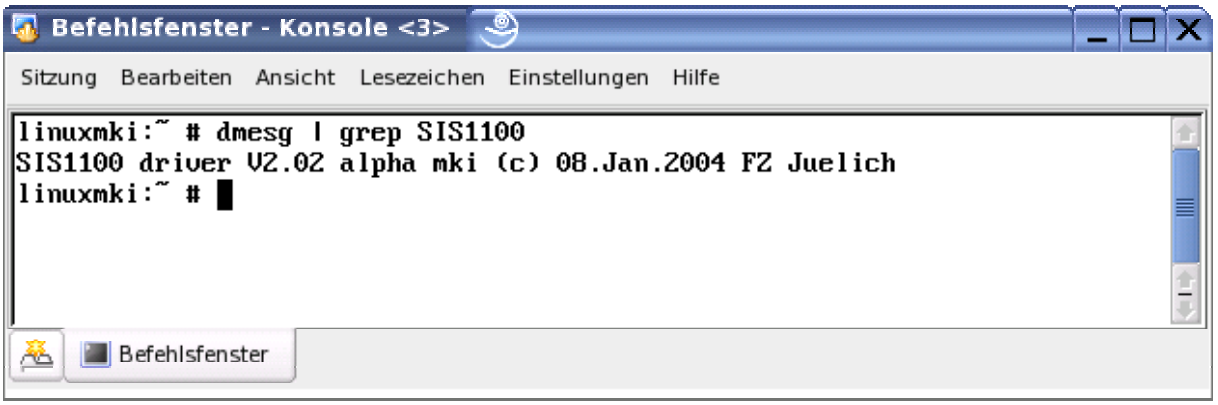
this should compile and result in the loadable module sis1100.o
Run the load_module script to load the driver and to generate the device descriptors.

```
./load_module
```

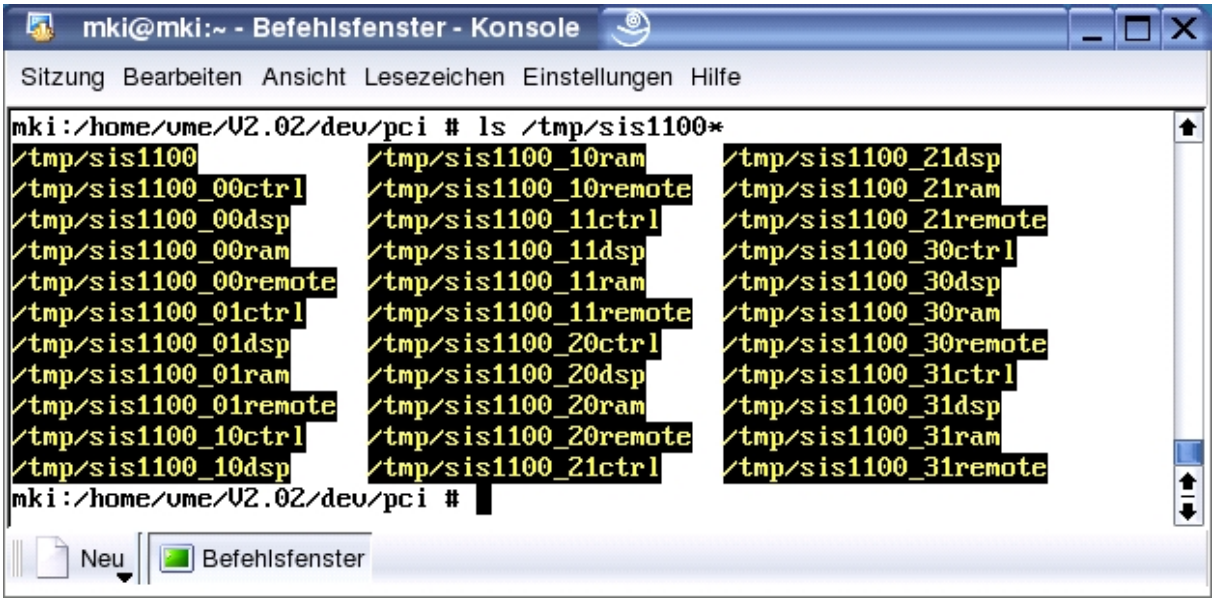
Execution of `dmesg | grep sis1100` should give you similar output to what is shown below (in this case a SIS3100 with SDRAM and DSP is connected to the SIS1100 and powered up.



The driver version can be seen as shown below:

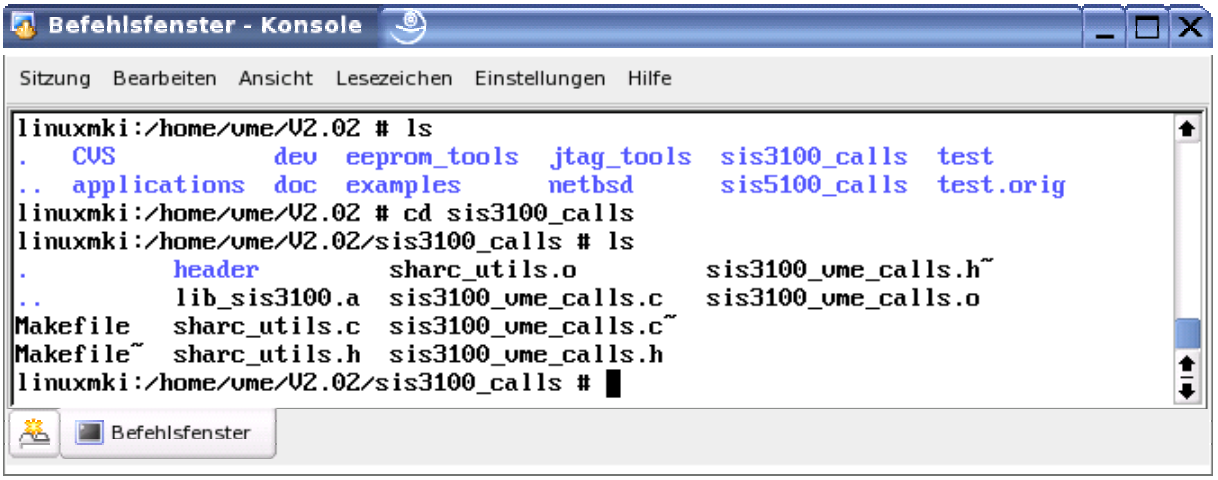


The descriptors are generated by default in the /tmp directory.



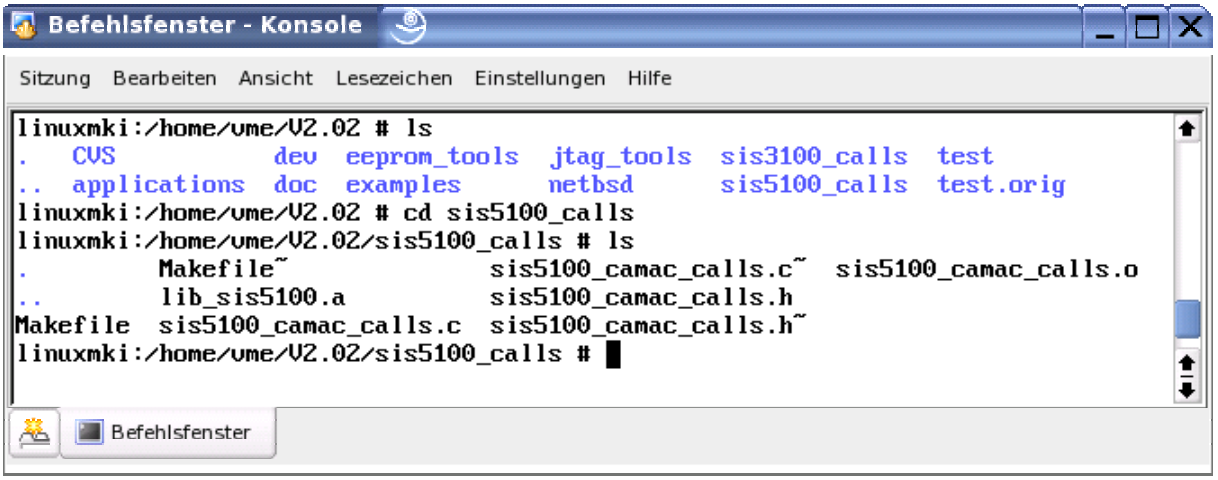
4.5.2 SIS3100 library lib_sis3100.a

Run the makefile in the sis3100_calls directory to obtain the library lib_sis3100.a as shown below. This library contains the VME calls.



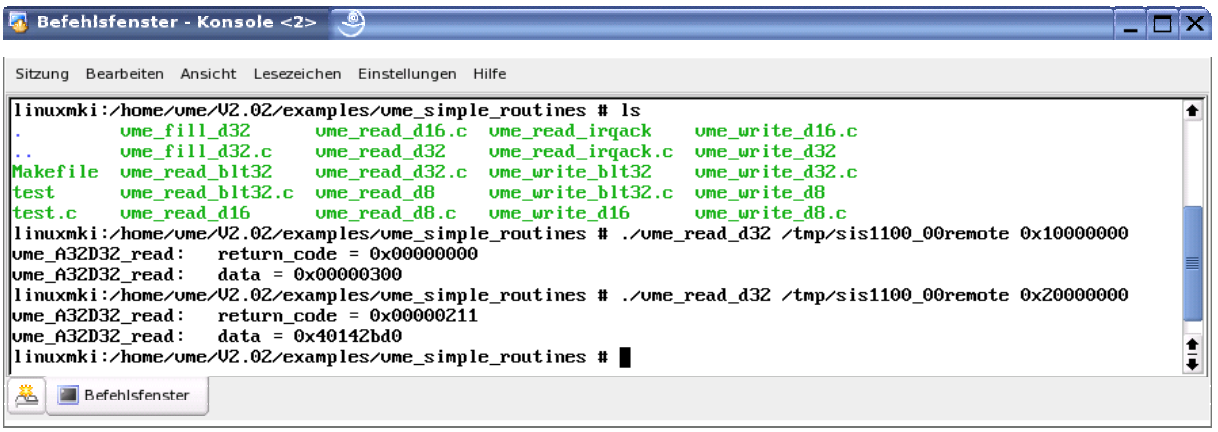
4.5.3 SIS5100 library lib_sis5100.a

Run the makefile in the sis5100_calls directory to obtain the library lib_sis5100.a as shown below. This library contains the CAMAC calls.



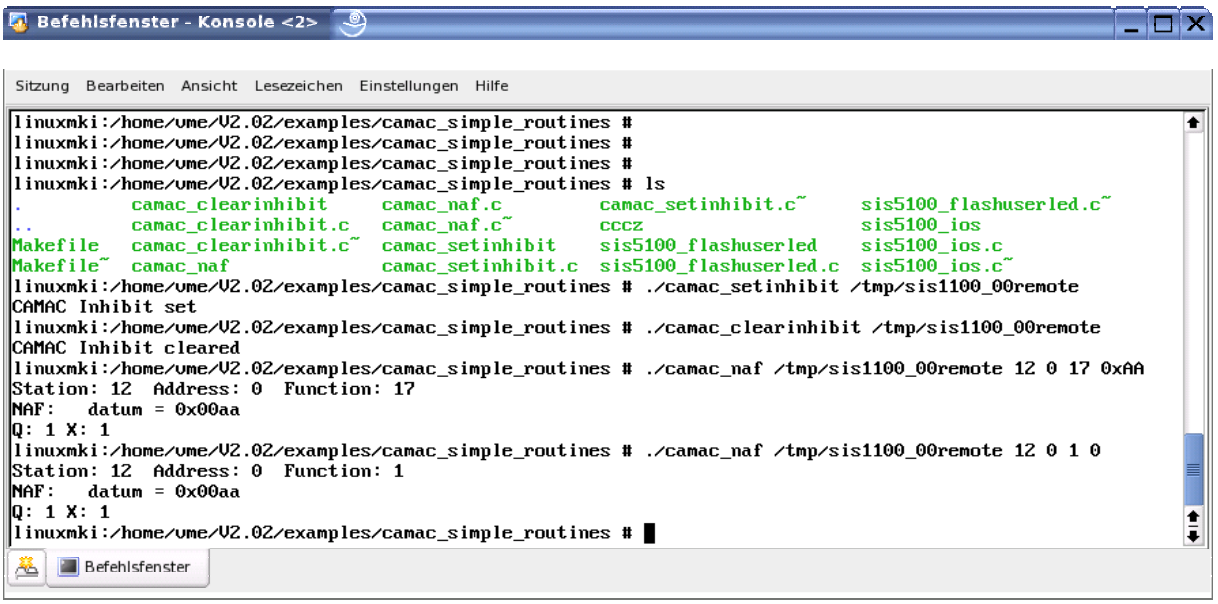
4.5.4 VME examples

The VME examples can be found in the examples/vme_simple_routines directory as shown below. Run the Makefile to generate the executables. The examples can be executed from the command line as illustrated in the screen shot (please note, that the 2nd call with a return code of 0x211 is a VME bus error).



4.5.5 CAMAC examples

The CAMAC examples can be found in the examples/camac_simple_routines directory as shown below. Run the Makefile to generate the executables. The examples can be executed from the command line as illustrated in the screen shot.



5 Subroutines

5.1 Summary of routines

The table below summarizes the implemented calls. Refer to the current version of the sis3100_vme_calls.h include file also in case routines were not added to the documentation yet.

VME environment handling calls	
open	5.2.1
close	5.2.2
SIS1100 control calls	
s1100_control_read	not implemented yet
s1100_control_write	not implemented yet
SIS3100 control calls	
s3100_control_read	5.3.1
s3100_control_write	5.3.2
VME routines	
Single word VME routines	
vme_A16D8_read	5.5.1
vme_A16D8_write	5.5.2
vme_A16D16_read	5.5.5
vme_A16D16_write	5.5.6
vme_A16D32_read	5.5.5
vme_A16D32_write	5.5.6
vme_A24D8_read	5.5.7
vme_A24D8_write	5.5.8
vme_A24D16_read	5.5.9
vme_A24D16_write	5.5.10
vme_A24D32_read	5.5.11
vme_A24D32_write	5.5.12
vme_A32D8_read	5.5.13
vme_A32D8_write	5.5.14
vme_A32D16_read	5.5.16
vme_A32D16_write	5.5.16
vme_A32D32_read	5.5.17
vme_A32D32_write	5.5.18
Single word DMA routines	
vme_A24DMA_D32_read	5.6.1
vme_A24DMA_D32_write	5.6.2
vme_A32DMA_D32_read	5.6.3
vme_A32DMA_D32_write	5.6.4
vme_A32DMA_D32FIFO_read	5.6.5
Block transfer routines	
vme_A24BLT32_read	5.7.1
vme_A24BLT32FIFO_read	5.7.2
vme_A24BLT32_write	5.7.3
vme_A24MBLT64_read	5.7.4
vme_A24MBLT64FIFO_read	5.7.5
vme_A24MBLT64_write	5.7.6
vme_A32BLT32_read	5.7.7
vme_A32BLT32FIFO_read	5.7.8
vme_A32BLT32_write	5.7.9
vme_A32MBLT64_read	5.7.10
vme_A32MBLT64FIFO_read	5.7.11

vme_A32MBLT64_write	5.7.12
2e block transfer routines	
vme_A32_2EVME_read	5.8.1
vme_A32_2EVMEFIFO_read	5.8.2
Interrupt handling routines	
SDRAM calls	
s3100_sdram_read	6.1.1
s3100_sdram_write	6.1.2
SHARC DSP option	
s3100_sharc_read	7.1.1
s3100_sharc_write	7.1.2

5.2 Routines to handle VME, SDRAM and SHARC environment

The open and close command to get a device descriptor for VME access, for access to the SDRAM option and to the DSP option are actually the standard LINUX open and close routines.

5.2.1 open

Function	open VME environment
Prototype	int open(const char* file, int oflag,int minor)
Parameters	
file	filename of device
oflag	flags, set to O_RDWR
minor	minor device Id., used to distinguish multiple SIS1100's
returncode	descriptor (positive) upon successful completion, negative otherwise
Include files	
#include <sys/ioctl.h>	

Example:

```
crate1= open("/tmp/sis1100", O_RDWR,0 );
crate2= open("/tmp/sis1100", O_RDWR,1 );
```

For SDRAM:

```
sdram1= open("/tmp/sis3100sdram ", O_RDWR,0 );
```

For SHARC DSP:

```
sharc1= open("/tmp/sis3100sharc ", O_RDWR,0 );
```

Note: this example assumes, that the device for the SIS1100 was linked to /tmp/sis1100 as described in the driver installation section 4.3

5.2.2 close

Function	close VME environment
Prototype	int close(int fd)
Parameters	
returncode	0 upon successful completion, non zero otherwise
fd	VME device descriptor
Include files	
#include <sys/ioctl.h>	

Example:

```
returncode= close(crate1);
```

5.3 SIS3100 control read/write routines

These routines allow you to access resources, like the control register e.g. on the SIS3100.

5.3.1 s3100_control_read

Function	SIS3100 control register read
Prototype	int s3100_control_read(int dev, int offset, u_int32_t* data)
Parameters	
offset	offset to SIS3100 base
data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode= s3100_control_read(dev, int offset, &controldatum);
```

5.3.2 s3100_control_write

Function	SIS3100 control register write
Prototype	int s3100_control_write(int dev, int offset, u_int32_t data)
Parameters	
offset	offset to SIS3100 base
data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
printf("switch on SIS3100 user LED\n");
offset = 0x00000100;
return_code = s3100_control_write(p, offset, 0x00000080) ;
if (return_code != 0) printf("s3100_control_write: return_code =
                              0x%08x\n", return_code );
```

5.4 VME infrastructure routines(s)

5.4.1 vmesysreset

Function	Issues VME SYSRESET on VME crate connected as device dev
Prototype	int vmesysreset(int dev);
Parameters	
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmesysreset(crate1);
```

5.5 VME single word read/write routines

5.5.1 vme_A16D8_read VME A16 D8 read

Function	single word A16 D8 read
Prototype	int vme_A16D8_read(int dev, u_int32_t vme_adr, u_int8_t* data8)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA16D8_read(crate1,adc1base,&adc1bytedata);
```

5.5.2 vme_A16D8_write VME A16 D8 write

Function	single word A16 D8 write
Prototype	int vme_A16D8_write(int dev, u_int32_t vme_adr, u_int8_t data8)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA16D8_write(crate1,adc1base,adc1bytedata);
```

5.5.3 vme_A16D16_read VME A16 D16 read

Function	single word A16 D16 read
Prototype	int vme_A16D16_read(int dev, u_int32_t vme_adr, u_int16_t* data16)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA16D16_read(crate1,adc1base,&adc1worddata);
```

5.5.4 vme_A16D16_write VME A16 D16 write

Function	single word A16 D16 write
Prototype	int vme_A16D16_write(int dev, u_int32_t vme_adr, u_int16_t data16)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA16D16_write(crate1,adc1base,adc1worddata);
```

5.5.5 vme_A16D32_read VME A16 D32 read

Function	single word A16 D32 read
Prototype	int vme_A16D32_read(int dev, u_int32_t vme_adr, u_int32_t* data32)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA16D32_read(crate1,adc1base,&adc1worddata);
```

5.5.6 vme_A16D32_write VME A16 D32 write

Function	single word A16 D32 write
Prototype	int vme_A16D32_write(int dev, u_int32_t vme_adr, u_int32_t data32)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA16D32_write(crate1,adc1base,adc1worddata);
```

5.5.7 vme_A24D8_read VME A24 D8 read

Function	single word A24 D8 read
Prototype	int vme_A24D8_read(int dev, u_int32_t vme_adr, u_int8_t* data8)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA24D8_read(crate1,adc1base,&adc1bytedata);
```

5.5.8 vme_A24D8_write VME A24 D8 write

Function	single word A24 D8 write
Prototype	int vme_A24D8_write(int dev, u_int32_t vme_adr, u_int8_t data8)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA24D8_write(crate1,adc1base,adc1bytedata);
```

5.5.9 vme_A24D16_read VME A24 D16 read

Function	single word A24 D16 read
Prototype	int vme_A24D16_read(int dev, u_int32_t vme_adr, u_int16_t* data16)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA24D16_read(crate1,adc1base,&adc1worddata);
```

5.5.10 vme_A24D16_write VME A24 D16 write

Function	single word A24 D16 write
Prototype	int vme_A24D16_write(int dev, u_int32_t vme_adr, u_int16_t data16)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA24D16_write(crate1,adc1base,adc1worddata);
```

5.5.11 vme_A24D32_read VME A24 D32 read

Function	single word A24 D32 read
Prototype	int vme_A24D32_read(int dev, u_int32_t vme_adr, u_int32_t* data32)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA24D32_read(crate1,adc1base,&adc1worddata);
```

5.5.12 vme_A24D32_write VME A24 D32 write

Function	single word A24 D32 write
Prototype	int vme_A24D32_write(int dev, u_int32_t vme_adr, u_int32_t data32)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA24D32_write(crate1,adc1base,adc1worddata);
```

5.5.13 vme_A32D8_read VME A32 D8 read

Function	single word A32 D8 read
Prototype	int vme_A32D8_read(int dev, u_int32_t vme_adr, u_int8_t* data8)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA32D8_read(crate1,adc1base,&adc1bytedata);
```

5.5.14 vme_A32D8_write VME A32 D8 write

Function	single word A32 D8 write
Prototype	int vme_A32D8_write(int dev, u_int32_t vme_adr, u_int8_t data8)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA32D8_write(crate1,adc1base,adc1bytedata);
```

5.5.15 vme_A32D16read VME A32 D16 read

Function	single word A32 D16 read
Prototype	int vme_A32D16_read(int dev, u_int32_t vme_adr, u_int16_t* data16)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA32D16_read(crate1,adc1base,&adc1worddata);
```

5.5.16 vme_A32D16_write VME A32 D16 write

Function	single word A32 D16 write
Prototype	int vme_A32D16_write(int dev, u_int32_t vme_adr, u_int16_t data16)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA32D16_write(crate1,adc1base,adc1worddata);
```

5.5.17 vme_A32D32_read VME A32 D32 read

Function	single word A32 D32 read
Prototype	int vme_A32D32_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to datum to read
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA32D32_read(crate1,adc1base,&adc1data);
```

5.5.18 vme_A32D32_write VME A32 D32 write

Function	single word A32 D32 write
Prototype	int vme_A32D32_write(int dev, u_int32_t vme_adr, u_int32_t vme_data)
Parameters	
vme_adr	VME address to write to
vme_data	datum to write
returncode	0 upon successful completion, non zero otherwise
dev	VME device handle

Example:

```
returncode=vmeA32D32_write(crate1,adc1base,adc1data);
```

5.6 VME single word DMA routines

Single word DMA routines are a unique feature of the SIS1100/3100. The transfer from the SIS1100 is done in direct memory access mode, while the VME slave is addressed with single word access. These routines yield a substantial gain in performance on VME slaves with lack of block read capabilities as no additional operating system overhead is involved for the individual read within the block. The number of data words does not necessarily have to be know before the transfer is started as the number of transferred words up to error condition (bus error e.g.) occurrence is stored. Routines with "FIFO" in the name are non address incrementing (to allow for readout from a FIFO memories with fixed address).

5.6.1 vme_A24DMA_D32_read

Function	single word A24 D32 read with DMA (and VME address increment)
Prototype	int vme_A24DMA_D32_read(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_no_of_lwords) ;
Parameters	
vme_adr	A24 VME address to read from
vme_data	Pointer to array to receive the data
req_num_of_lwords	Number of requested longwords
got_no_of_lwords	Number of transferred longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

5.6.2 vme_A24DMA_D32_write

Function	single word A24 D32 write with DMA (and VME address increment)
Prototype	int vme_A24DMA_D32_write(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* put_num_of_lwords) ;
Parameters	
vme_adr	A24 VME address to write to
vme_data	Pointer to data array
req_num_of_lwords	Number of requested longwords
Put_num_of_lwords	Number of written longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. put_num_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

5.6.3 vme_A32DMA_D32_read

Function	single word A32 D32 read with DMA (and VME address increment)
Prototype	int vme_A32DMA_D32_read(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_no_of_lwords) ;
Parameters	
vme_adr	A32 VME address to read from
vme_data	Pointer to array to receive the data
req_num_of_lwords	Number of requested longwords
got_no_of_lwords	Number of transferred longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

5.6.4 vme_A32DMA_D32_write

Function	single word A32 D32 write with DMA (and VME address increment)
Prototype	int vme_A32DMA_D32_write(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* put_num_of_lwords) ;
Parameters	
vme_adr	A32 VME address to write to
vme_data	Pointer to data array
req_num_of_lwords	Number of requested longwords
got_no_of_lwords	Number of transferred longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. put_num_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

5.6.5 vme_A32DMA_D32FIFO_read

Function	single word 32 D32 read with DMA (from constant VME address)
Prototype	int vme_A32DMA_D32_read(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_no_of_lwords) ;
Parameters	
vme_adr	A32 VME address to read from
vme_data	Pointer to array to receive the data
req_num_of_lwords	Number of requested longwords
got_no_of_lwords	Number of transferred longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

5.7 VME block read/write routines

All block transfer routines focus on the efficient movement of data from or to VME slaves. As many front end modules tend to have a variable (in many cases unknown) amount of data the user will want to read a blocksize that is in any case longer than the amount of data that is actually available from the particular slave. The slave will terminate the transfer by actively issuing a bus error or the bus error condition will be detected as soon as the predefined bus error timeout on the SIS3100 has expired. In nay case the bus error can not be regarded as an error condition on these block reads and the user will have to verify success of the operation by checking the actual number of read/written words.

5.7.1 vme_A24BLT32_read VME A32 BLT32 block read

Function	A24 BLT32 block read (with address auto increment)
Prototype	int vme_A24BLT32_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first A24 VME address to read from
vme_data	pointer to the array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA24BLT32_read(crate1,memorybase,*readdata,0x100,wordsread);
```

5.7.2 vme_A24BLT32FIFO_read VME A24 BLT32 block read from FIFO

This routine is the non address incrementing version of vme_A24BLT32_read. It is used to read data from FIFO memories.

Function	A24 BLT32 block read (without address auto increment)
Prototype	int vme_A24BLT32FIFO_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	A24 VME address to read from
vme_data	pointer to the array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA24BLT32FIFO_read(crate1,fifobase,*readdata,wordstowrite,word
sread);
```

5.7.3 vme_A24BLT32_write VME A24 BLT32 block write

Function	A24 BLT32 block write (with address auto increment)
Prototype	int vme_A24BLT32_write(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first A24 VME address to write to
vme_data	pointer to the data array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of written longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA24BLT32_write(crate1,memorybase,*writedata,wordstowrite,words
written);
```

5.7.4 vme_A24MBLT64_read VME A24 MBLT64 block read

Function	A24 MBLT64 block read (with address auto increment)
Prototype	int vme_A24MBLT64_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first A24 VME address to read from
vme_data	Pointer to the array
req_num_of_lwords	Number of longwords to transfer
got_num_of_lwords	Actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA24MBLT64_read(crate1,memorybase,*readdata,0x100,wordsread);
```

5.7.5 vme_A24MBLT64FIFO_read VME A24 MBLT64 block read from FIFO

This routine is the non address incrementing version of vme_A32MBLT64_read. It is used to read data from FIFO memories.

Function	A24 MBLT64 block read (without address auto increment)
Prototype	int vme_A24MBLT64FIFO_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	A24 VME address to read from
vme_data	Pointer to the array
req_num_of_lwords	Number of longwords to transfer
got_num_of_lwords	Actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA24MBLT64FIFO_read(crate1,fifobase,*readdata,wordstowrite,wor
dsread);
```

5.7.6 vme_A24MBLT64_write VME A24 MBLT64 block write

Function	A24 MBLT64 block write (with address auto increment)
Prototype	int vme_A24MBLT64_write(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first A24 VME address to write to
vme_data	pointer to the data array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of written longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA24MBLT64_write(crate1,memorybase,*writedata,wordstowrite,word
swritten);
```

5.7.7 vme_A32BLT32_read VME A32 BLT32 block read

Function	A32 BLT32 block read (with address auto increment)
Prototype	int vme_A32BLT32_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first VME address to read from
vme_data	Pointer to the array
req_num_of_lwords	Number of longwords to transfer
got_num_of_lwords	actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA32BLT32_read(crate1,memorybase,*readdata,0x100,wordsread);
```

5.7.8 vme_A32BLT32FIFO_read VME A32 BLT32 block read from FIFO

This routine is the non address incrementing version of vme_A32BLT32_read. It is used to read data from FIFO memories.

Function	A32 BLT32 block read (without address auto increment)
Prototype	int vme_A32BLT32FIFO_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	VME address to read from
vme_data	Pointer to the array
req_num_of_lwords	Number of longwords to transfer
got_num_of_lwords	actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA32BLT32FIFO_read(crate1,fifobase,*readdata,wordstowrite,word
sread);
```

5.7.9 vme_A32BLT32_write VME A32 BLT32 block write

Function	A32 BLT32 block write (with address auto increment)
Prototype	int vme_A32BLT32_write(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first VME address to write to
vme_data	pointer to the data array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of written longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA32BLT32_write(crate1,memorybase,*writedata,wordstowrite,words
written);
```

5.7.10 vme_A32MBLT64_read VME A32 MBLT64 block read

Function	A32 MBLT64 block read (with address auto increment)
Prototype	int vme_A32MBLT64_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first VME address to read from
vme_data	pointer to the array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA32MBLT64_read(crate1,memorybase,*readdata,0x100,wordsread);
```

5.7.11 vme_A32MBLT64FIFO_read VME A32 MBLT64 block read from FIFO

This routine is the non address incrementing version of vme_A32MBLT64_read. It is used to read data from FIFO memories.

Function	A32 MBLT64 block read (without address auto increment)
Prototype	int vme_A32MBLT64FIFO_read(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	VME address to read from
vme_data	pointer to the array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of read longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA32MBLT64FIFO_read(crate1,fifobase,*readdata,wordstowrite,wor
dsread);
```

5.7.12 vme_A32MBLT64_write VME A32 MBLT64 block write

Function	A32 MBLT64 block write (with address auto increment)
Prototype	int vme_A32MBLT64_write(int dev, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords)
Parameters	
vme_adr	first VME address to write to
vme_data	pointer to the data array
req_num_of_lwords	number of longwords to transfer
got_num_of_lwords	actual number of written longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

Example:

```
returncode=vmeA32MBLT64_write(crate1,memorybase,*writedata,wordstowrite,word
swritten);
```

5.8 2e VME block transfer routines

The SIS3100 is one of the first VME masters on the market to implement 2e (2 edge) VME functionality. It can be use in conjunction with the SIS3300 and SIS3301 digitizers to reach readout performance in excess of 80 Mbytes/s.

5.8.1 vme_A32_2EVME_read

Function	2eVME A32 block transfer read (and VME address increment)
Prototype	int vme_A32_2EVME_read(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords) ;
Parameters	
vme_adr	A32 VME address to read from
vme_data	Pointer to array to receive the data
req_num_of_lwords	Number of requested longwords
got_no_of_lwords	Number of transferred longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

5.8.2 vme_A32_"2EVMEFIFO_read

Function	2eVME A32 block transfer read (no VME address increment)
Prototype	int vme_A32_2EVMEFIFO_read(int p, u_int32_t vme_adr, u_int32_t* vme_data, u_int32_t req_num_of_lwords, u_int32_t* got_num_of_lwords) ;
Parameters	
vme_adr	A32 VME address to read from
vme_data	Pointer to array to receive the data
req_num_of_lwords	Number of requested longwords
got_no_of_lwords	Number of transferred longwords
returncode	0 upon successful completion, non zero otherwise Upon non zero completion (with error code = 0x211, bus error e.g.) part of the data (i.e. got_no_of_lwords) may have been transferred before the error condition occurred.
dev	VME device handle

6 SDRAM access routines

These routines handle communication with the SDRAM option of the SIS3100 board.

6.1 SDRAM single word read/write routines

These routines allow data transfer between the PCI side of the link and memory locations of the SDRAM memory strip. Although they are in principle single cycle routines they allow the transfer of a number of longwords in a non DMA transfer

6.1.1 s3100_sdram_read

Function	read data from SIS3100 SDRAM strip
Prototype	int s3100_sdram_read(int sdram_dev, u_int32_t byte_adr, u_int32_t* data, u_int32_t num_of_lwords)
Parameters	
byte_adr	address offset (within SDRAM)
data	pointer to data array to read to
num_of_lwords	number of longwords to read
returncode	number of written bytes upon successful completion, negative otherwise
sdram_dev	SDRAM device handle

Example:

```
returncode=s3100_sdram_read(crate1_sdram,sdrambase,*data,one word);
```

6.1.2 s3100_sdram_write

Function	write data to SIS3100 SDRAM strip
Prototype	int s3100_sdram_write(int sdram_dev, u_int32_t byte_adr, u_int32_t* data, u_int32_t num_of_lwords)
Parameters	
byte_adr	address offset (within SDRAM)
data	pointer to data array that holds the data to write
num_of_lwords	number of longwords to write
returncode	number of written bytes upon successful completion, negative otherwise
sdram_dev	SDRAM device handle

Example:

```
returncode=s3100_sdram_write(crate1_sdram,sdrambase,*data,oneword);
```

7 DSP access routines

These routines handle access to memory locations of the SIS3100 boards SIS9200 DSP option.

7.1 DSP single word read/write routines

These routines allow data transfer between the PCI side of the link and resources on the SIS9200 DSP piggy. Although they are in principle single cycle routines they allow the transfer of a number of longwords in a non DMA transfer

7.1.1 s3100_sharc_read

Function	read data from SHARC DSP
Prototype	int s3100_sharc_read(int sharc_dev, u_int32_t byte_adr, u_int32_t* data, u_int32_t num_of_lwords)
Parameters	
byte_adr	address offset (within SIS9200)
data	pointer to data array to read to
num_of_lwords	number of longwords to read
returncode	0 upon successful completion, non zero otherwise
sharc_dev	SHARC device handle

Example:

```
returncode=s3100_sharc_read(crate1_sharc,sharcbase,*data,one word);
```

7.1.2 s3100_sharc_write

Function	write data to SHARC DSP
Prototype	int s3100_sharc_write(int sharc_dev, u_int32_t byte_adr, u_int32_t* data, u_int32_t num_of_lwords)
Parameters	
byte_adr	address offset (within SIS9200)
data	pointer to data array that holds the data to write
num_of_lwords	number of longwords to write
returncode	0 upon successful completion, non zero otherwise
sharc_dev	SHARC device handle

Example (SHARC program load):

```
printf("loading SHARC DSP\n");
currentaddress=SHARCRAM;
while (loadcount<count) {
   addr = D48REG;
   data = tempword[loadcount];
   return_code = s3100_sharc_write(p_sharc, addr, &data, 0x1);
   if (return_code != 0)
      printf("s3100_sharc_write: return_code = 0x%08x\n",return_code );
   loadcount++;
   addr = currentaddress;
   data = ((tempword[loadcount+1] << 16 ) & 0xFFFF0000)+
           (tempword[loadcount] & 0x0000FFFF);
   return_code = s3100_sharc_write(p_sharc, addr, &data, 0x1);
   if (return_code != 0)
      printf("s3100_sharc_write: return_code = 0x%08x\n",return_code );
   currentaddress+=4;
   loadcount+=2;
  }
```

8 Return/Error codes

Following error codes are defined.

Name	Code	Condition
RE_NRDY	0x202	remote station not ready (status of control register bit READY)
RE_PROT	0x206	protocol error (illegal request e.g.)
RE_TO	0x207	protocol timeout
RE_BERR	0x211	VME bus error
RE_RETRY	0x212	VME retry
RE_ARB	0x214	Arbitration timeout, SIS3100 could not get bus mastership

Note: The VME retry line a previously reserved line (connector P2 pin B3). It is not properly terminated on some older VME crates. VME retry functionality is active by default on the SIS3100 up to firmware revision major 1 minor 5, what will result in retry errors in such a crate. SIS3100 firmware revisions major 1 minor 6 and higher have the retry inactive as power up default to avoid this problem source. User that want to make use of retry functionality will have to activate the feature in the control register of the SIS3100.

9 Index

<sys/ioctl.h> 20
16 MHz 8
A32 6
arbitration timeout 46
backplane 8
bending 9
BERR 46
block read 35
block write 35
BLT32 35
bus error 35, 46
bus mastership 46
card 8
cat 11
chip 8
chmod 12
close 20
CMC 7
configure 11
connector
  LC 9
copyright 5
crate 8
D32 6
devfs 12
device file system 12
device id. 20
  minor 20
driver
  load 11
  unload 12
DSP 43, 44
duplex 9
dust 9
error code 46
examples 13
  CAMAC 17
  VME 17
FAQ 5
fibre 9
  duplex 9
  minimum bending radius 9
FIFO 32
insmod 11
introduction 5
ioctl 6
latch 9
LC 9
LED
  L 9
  LD 9
  LU 9
Liability 5
lib_sis3100.a 16
lib_sis5100.a 16
load_module 11
Makefile 13
mastership 46
mknod 11, 12
open 20
PCI 7
pcitweak 8
protection 9
protocol error 46
protocol timeout 46
retry 46
return code 46
rmmod 12
s3100_control_read 21
s3100_control_write 21
s3100_sdram_read 43
s3100_sdram_write 43
s3100_sharc_read 44
s3100_sharc_write 45
sdram 20
SDRAM 43
sharc 20
SHARC 43, 44
SIS1100
  installation 7
SIS1100-CMC 7
SIS1100-OPT 7
SIS3100
  installation 8
SYSRESET 8, 22
system clock 8
vendor Id. 8
VME
  retry 46
  slave 8
vme_A16D16_read 24
vme_A16D16_write 24
vme_A16D32_read 25
vme_A16D32_write 25
vme_A16D8_read 23
vme_A16D8_write 23
vme_A24BLT32_read 35
vme_A24BLT32_write 36
vme_A24BLT32FIFO_read 36
vme_A24D16_read 27
vme_A24D16_write 27
vme_A24D32_read 28
vme_A24D32_write 28
vme_A24D8_read 26
vme_A24D8_write 26
vme_A24DMA_D32_read 32
vme_A24DMA_D32_write 32
vme_A24MBLT64_read 37
vme_A24MBLT64_write 38
vme_A24MBLT64FIFO_read 37
vme_A32_2EVME_read 42
vme_A32_2EVMEFIFO_read 42
vme_A32BLT32_fiforead 39
vme_A32BLT32_read 38

vme_A32BLT32_write....39
vme_A32D16_read....30
vme_A32D16_write....30
vme_A32D32_read....31
vme_A32D32_write....31
vme_A32D8_read....29
vme_A32D8_write....29
vme_A32DMA_D32_read....33
vme_A32DMA_D32_write....33
vme_A32DMA_D32FIFO_read....34
vme_A32MBLT64_fiforead....40
vme_A32MBLT64_read....40
vme_A32MBLT64_write....41
VME64x....8
vmesysreset....22