SIS3153
Ethernet API Reference

SIS GmbH
Harksheider Str. 102A
22399 Hamburg
Germany

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

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

Version: sis3153-M-eth-V101-windows_Api_Reference.doc
as of 03.08.2017

Revision Table:

Revision	Date	Modification
0.02	14.12.16	cosmetics
1.01	03.08.2017	Update

- Table of contents

1 Overview

This section provides an overview over the implemented SIS3153API functions.
See `sis3153ETH_vme_class.cpp and sis3153ETH_vme_class.h`

API Function Name	Description
Default Class Constructor	Open a device handle with default values
Standard Class Constructor	Open a device handle
get_UdpSocketStatus	Return the UDP socket status
get_UdpSocketPort	Return the port number
set_UdpSocketOptionTimeout	Define the UDP socket timeout span
set_UdpSocketOptionBufSize	Define the UDP socket buffer size
set_UdpSocketBindToDevice	LINUX :Bind the socket to a real ETH device
set_UdpSocketBindMyOwnPort	WIN: Bind the socket to a real ETH device
set_UdpSocketSIS3153_IpAddress	Set the IP address of the SIS3153
UdpSocketNofReadMaxLWords	Define number of long words which are read
UdpSocketNofWriteMaxLWords	Define number of long words which are write
UdpSocketReceiveNofPackagesAtOnce	Define number of packages which are receive
UdpSocketGapValue	Define the gap time
UdpSocketEnableJumboFrame	Enable data transfer with jumbo frames
UdpSocketDisableJumboFrame	Disable data transfer with jumbo frames
reset_cmd	Reset the SIS3153 UDP stack
register_read/write	Single cycle access to internal address space
register_dma_read/write	Block access to internal address space
vme_read/write	Single cycle access to VME address space
vme_dma_read/write	Block access to VME address space
Additional VME wrapper functions	**Several functions for access VME address space**
`vme_A16D8_read`	
`vme_A16D16_read`	
`vme_A16D32_read`	
`..`	
`..`	
`vme_A32BLT32_read`	
`vme_A32MBLT64_read`	
`..`	
`..`	
`vme_A16D8_write`	
`vme_A16D16_write`	
`vme_A16D32_write`	
`..`	

1.1 Sis3153 Default Class Constructor

Syntax:

```
sis3153eth(
       sis3153eth **eth_interface,
       char *device_ip
);
```

Description:
Tries to open a device handle for a card with the supplied device handle structure pointer and the target device IP.
This constructor initializes the class variables with default values, e.g. the default LAN interface will be used. If it is necessary to adapt the values, use the standard constructor and the following functions for the modifications.

Arguments:

eth_interface
: Pointer to a device handle structure.

device_ip
: Pointer to a char array which holds the target device IP.

Usage:

```
       sis3153eth *eth_interface[MAX_SOCKETS];
       sis3153eth(eth_interface, "212.60.16.200");
```

or

```
       strcpy(sis3153_ip_addr_string, "212.60.16.200"); // SIS3153 IP address
       sis3153eth *vme_crate;
       sis3153eth(&vme_crate, sis3153_ip_addr_string);
```

1.2 Sis3153 Standard Class Constructor

Syntax:

```
sis3153eth(
       void
);
```

Description:
Tries to open a device handle for a card. It is required to set up the values for different variables e.g. for the device IP. Notice the example below.

Arguments:
    none

Usage:

```
sis3153eth *eth_interface = new sis3153eth;

#ifdef WINDOWS
       WORD wVersionRequested;
       WSADATA wsaData;
       wVersionRequested = MAKEWORD(2, 1);
       WSAStartup( wVersionRequested, &wsaData );
#endif

eth_interface->set_UdpSocketOptionBufSize(0x2000000);

strcpy(pc_ip_addr_string,"");
eth_interface->set_UdpSocketBindMyOwnPort(pc_ip_addr_string);

eth_interface->set_UdpSocketSIS3153_IpAddress(device_ip);

…
```

1.3 get_UdpSocketStatus

Syntax:

```
INTEGER
get_UdpSocketStatus (
       void
);
```

Description:
Returns the socket status.

Arguments:
    none

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	A SOCKET_ERROR has occurred

Usage:

```
sis3153eth *eth_interface[MAX_SOCKETS];
sis3153eth(eth_interface, "212.60.16.200");

if(eth_interface[0]->get_UdpSocketStatus() != 0)
{
       printf("ERROR: Can't open UDP soket!");
       return -1;
}
```

1.4 get_UdpSocketPort

Syntax:

```
INTEGER
get_UdpSocketPort (
       void
);
```

Description:
Returns the number of the used port.

Arguments:
none

Usage:

```
…

printf("Connection establisched on port: %d\n",eth_interface[0]->get_UdpSocketPort());
```

1.5 set_UdpSocketOptionTimeout

Syntax:

```
this->recv_timeout_sec  = 0 ;
this->recv_timeout_usec = 50000; // default 50ms
status = this->set_UdpSocketOptionTimeout( ) ;
```

Description:
Defines the time span in which an Ethernet action must be finished.

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	A SOCKET_ERROR has occurred

Usage:

```
eth_interface[0]->recv_timeout_sec  = 0 ;
eth_interface[0]->recv_timeout_usec = 50000; // default 50ms
status = eth_interface[0]->set_UdpSocketOptionTimeout( ) ;

if(status != 0){
	printf("Error in 'set_UdpSocketOptionTimeout': %x\n", status);
}
```

1.6 set_UdpSocketOptionBufSize

Syntax:

```
INTEGER
int set_UdpSocketOptionBufSize(
        int sockbufsize
);
```

Description:
Defines the socket buffer size.

Arguments:

sockbufsize

Pointer to an opened device

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	A SOCKET_ERROR has occurred

Usage:

```
…

status = eth_interface[0]->set_UdpSocketOptionBufSize(0x2000000);

if(status != 0){
        printf("Error in 'set_UdpSocketOptionBufSize': %x\n", status);
}
```

1.7 set_UdpSocketBindToDevice

Syntax:

```
	int set_UdpSocketBindMyOwnPort( char* pc_ip_addr_string);
```

Description:
For Linux: Defines which Ethernet host device(e.g. eth0) is used for the connection to the SIS3153.

Arguments:

eth_device

Pointer to an opened device

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	A SOCKET_ERROR has occurred

Usage:

```
…

#ifdef LINUX
	status = eth_interface[0]->set_UdpSocketBindToDevice("eth0");
#else
	status = 0;
#endif

if(status != 0){
	printf("Error in 'set_UdpSocketBindToDevice': %x\n", status);
}
```

1.8 *set_UdpSocketBindMyOwnPort*

Syntax:

```
INTEGER
set_UdpSocketBindMyOwnPort(
       char* pc_ip_addr_string
);
```

Description:
For Windows: Defines which Ethernet host device (e.g IP address of the second LAN interface) is used for the connection to the SIS3153.

Arguments:

pc_ip_addr_string
: Pointer to an opened device

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	A SOCKET_ERROR has occurred

Usage:

```
…

#ifdef WINDOWS
status = eth_interface[0]->set_UdpSocketSIS3153_IpAddress(device_ip);
#else
       status = 0;
#endif

if(status != 0){
       printf("Error in 'set_UdpSocketSIS3153_IpAddress': %x\n", status);
}
```

1.9 set_UdpSocketSIS3153_IpAddress

Syntax:

```
INTEGER
set_UdpSocketSIS3153_IpAddress(
       char* sis3153_ip_addr_string
);
```

Description:
Defines the Ethernet address of the SIS3153.

Arguments:
sis3153_ip_addr_string
Pointer to an opened device

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

```
…

char* device_ip = "212.60.16.200";

status = eth_interface[0]->set_UdpSocketSIS3153_IpAddress(device_ip);

if(status != 0){
       printf("Error in 'set_UdpSocketSIS3153_IpAddress': %x\n", status);
}
```

1.10 get_UdpSocketNofReadMaxLWords

Syntax:

```
INTEGER
get_UdpSocketNofReadMaxLWords(
       void
);
```

Description:
Returns the number of long words, which are read at once.

Arguments:
none

Usage:

```
…

printf("Read %d LWords at once.\n", eth_interface[0]->get_UdpSocketNofReadMaxLWords()
);
```

1.11 set_UdpSocketNofReadMaxLWords

Syntax:

```
INTEGER
set_UdpSocketNofReadMaxLWords(
	UINT nofMaxLWords
);
```

Description:
Defines the maximum number of long words, which are read at once.

Arguments:
    nofMaxLWords

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

```
…

status = eth_interface[0]-> set_UdpSocketNofReadMaxLWords(0x10000);

if(status != 0){
	printf("Error in 'set_UdpSocketNofReadMaxLWords': %x\n", status);
}
```

1.12 get_UdpSocketNofWriteMaxLWords

Syntax:

```
INTEGER
get_UdpSocketNofWriteMaxLWords(
        void
);
```

Description:
Returns the number of long words, which are written at once.

Arguments:
none

Usage:

```
…

printf("Write %d LWords at once.\n", eth_interface[0]-> UdpSocketNofWriteMaxLWords()
);
```

1.13 set_UdpSocketNofWriteMaxLWords

Syntax:

```
INTEGER
set_UdpSocketNofWriteMaxLWords(
	UINT nofMaxLWords
);
```

Description:
Defines the maximum number of long words, which are written at once.

Arguments:
	nofMaxLWords

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

```
…

status = eth_interface[0]->set_UdpSocketNofWriteMaxLWords (0x100);

if(status != 0){
	printf("Error in 'set_UdpSocketNofReadMaxLWords': %x\n", status);
}
```

1.14 get_UdpSocketReceiveNofPackagesAtOnce

Syntax:

```
INTEGER
get_UdpSocketReceiveNofPackagesAtOnce(
	void
);
```

Description:
Returns the number of packages, which are received at once.

Arguments:
	none.

Usage:

```
…

printf("Read %d packages at once.\n", eth_interface[0]->
get_UdpSocketReceiveNofPackagesAtOnce()
);
```

1.15 set_UdpSocketReceiveNofPackagesAtOnce

Syntax:

```
INTEGER
set_UdpSocketReceiveNofPackagesAtOnce(
       UINT nofAtOnce
);
```

Description:
Defines the maximum number of packages, which are received at once.
Arguments:
    nofMaxLWords

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

```
…

status = eth_interface[0]-> set_UdpSocketReceiveNofPackagesAtOnce(0xFF);

if(status != 0){
       printf("Error in 'set_UdpSocketReceiveNofPackagesAtOnce': %x\n", status);
}
```

1.16 get_UdpSocketGapValue

Syntax:

```
INTEGER
get_UdpSocketGapValue(
        void
);
```

Description:
Returns the break length between two transmissions.

Arguments:
none

Usage:

```
…

printf("GapValue setup: 0x%02x\n", eth_interface[0]->get_UdpSocketGapValue());
```

1.17 set_UdpSocketGapValue

Syntax:

```
INTEGER
set_UdpSocketGapValue(
       UINT gapValue
);
```

Description:
Defines the gap length in between of two transmissions. This value can be chosen in fixed steps between 256ns and 57us. Refer to the main manual of the SIS3153 for details.

Arguments:
gapValue

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

```
…

status = eth_interface[0]->set_UdpSocketGapValue(0x2); // Insert a gap of 1us


if(status != 0){
       printf("Error in 'set_UdpSocketGapValue': %x\n", status);
}
```

1.18 get_UdpSocketJumboFrameStatus

Syntax:

```
INTEGER
get_UdpSocketJumboFrameStatus(
        void
);
```

Description:
Returns the status of the jumbo frame setup.

Arguments:
none

Return Codes:

Code	Description
0	Jumbo frames are disabled
1	Jumbo frames are enabled

Usage:

```
…

if(eth_interface[0]->get_UdpSocketJumboFrameStatus())
{
        printf("Jumboframes are enabled.\n");
}
else
{
        printf("Jumboframes are disabled.\n");
}
```

1.19 set_UdpSocketEnableJumboFrame

Syntax:

```
INTEGER
set_UdpSocketEnableJumboFrame(
        void
);
```

Description:
Enables jumbo frame data transmission.

Arguments:
none

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

```
…

status = eth_interface[0]->set_UdpSocketEnableJumboFrame();

if(status == 0){
        if(eth_interface[0]->get_UdpSocketJumboFrameStatus())
        {
                printf("Jumboframes are enabled.\n");
        }
        else
        {
                printf("Jumboframes are disabled.\n");
        }
}
else
{
        printf("Error in 'set_UdpSocketEnableJumboFrame': %x\n", status);
}
```

1.20 set_UdpSocketDisableJumboFrame

Syntax:

```
INTEGER
set_UdpSocketDisableJumboFrame(
	void
);
```

Description:
Disables jumbo frames transmission.

Arguments:
none

Return Codes:

Code	Description
0	The function returned successfully
-1	An error has occurred

Usage:

…

```
status = eth_interface[0]->set_UdpSocketDisableJumboFrame();

if(status == 0){
	if(eth_interface[0]->get_UdpSocketJumboFrameStatus())
	{
		printf("Jumboframes are enabled.\n");
	}
	else
	{
		printf("Jumboframes are disabled.\n");
	}
}
else
{
	printf("Error in 'set_UdpSocketDisableJumboFrame': %x\n", status);
}
```

1.21 udp_reset_cmd

Syntax:

```
INTEGER
udp_reset_cmd(
        void
);
```

Description:
Reset the SIS3153 Ethernet stack.

Arguments:
none

Return Codes:

Code	Description
`0`	The function returned successfully
`Otherwise`	A SOCKET_ERROR has occurred

1.22 udp_sis3153_register_read

Syntax:

```
INTEGER
udp_sis3153_register_read (
        UINT addr,
        UINT* data
);
```

Description:
Reads from the device internal control register space.

Arguments:

addr
: register offset to read from

data
: Pointer to an unsigned 32bit variable holding the read data

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	An error has occurred

Usage:

```
…

addr = 0x1;
return_code = eth_interface[0]->udp_sis3153_register_read(addr, &data);

printf("sis3153_Register_Single_Read: addr = %08X   data = %08X    return_code =  %08X \n",
addr, data, return_code);

…
```

1.23 udp_sis3153_register_write

Syntax:

```
INTEGER
udp_sis3153_register_write (
        UINT addr,
        UINT data
);
```

Description:
Writes to the internal register space of the device.

Arguments:

addr
: register offset to write to

data
: Unsigned 32bit variable holding the data to be written

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	An error has occurred

Usage:

```
SIS1100W_STATUS status;

status = sis3153w_VmeSysreset(
                        usbDevice
                        );

if(status != Stat3153Success){
      printf("Error in 'sis3153w_VmeSysreset': %x\n", status);
}
```

1.24 udp_sis3153_register_dma_read

Syntax:

```
INTEGER
udp_sis3153_register_dma_read (
       UINT addr,
       UINT* data,
       UINT request_nof_words,
       UINT* got_nof_words
);
```

Description:
Reads from the internal register space of the device in block transfer access, .

Arguments:

addr
: register offset to read from

data
: Pointer to an unsigned 32bit buffer which holds the data

request_nof_words
: Requested number of 32bit word to read

got_nof_words
: Pointer to an unsigned 32bit value to hold the number of 32bit words transferred

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	An error has occurred

Usage:

```
unsigned int status;
unsigned int start_addr= 0x1000;          // internal RAM space;
unsigned int read_buffer[0x1000];         // 4k Lwords
unsigned int req_read_length = 0x1000;
unsigned int ack_read_length;

status = eth_interface[0]->udp_sis3153_register_dma_read(start_addr, read_buffer,
req_read_length, &ack_read_length);

if(status != 0)
       printf("ERROR udp_sis3153_register_dma_read: req_read_length = %08X   ack_read_length
%08X   return_code =  %08X \n",req_read_length, ack_read_length, status);
else
       printf("udp_sis3153_register_dma_read: req_read_length = %08X   ack_read_length %08X
return_code =  %08X \n",req_read_length, ack_read_length, status);
```

1.25 udp_sis3153_register_dma_write

Syntax:

```
INTEGER
udp_sis3153_register_dma_write (
       UINT addr,
       UINT *data,
       UINT request_nof_words,
       UINT* written_nof_words
);
```

Description:
Writes to internal register space of the device in block transfer access. .

Arguments:

addr	register offset to write to
data	Pointer to an unsigned 32bit buffer which holds the data
request_nof_words	Requested number of 32bit word to write
written_nof_words	Pointer to an unsigned 32bit value to hold the number of 32bit words transferred

Return Codes:

Code	Description
0	The function returned successfully
Otherwise	An error has occurred

Usage:

```
unsigned int status;
unsigned int start_addr= 0x1000;                // internal RAM space;
unsigned int write_buffer[0x1000];              // 4k Lwords
unsigned int req_write_length = 0x1000;
unsigned int ack_write_length;

/* ... create write data ... */

status = eth_interface[0]->udp_sis3153_register_dma_write(start_addr, write_buffer,
req_read_length, &ack_read_length);

if(status != 0)
       printf("ERROR udp_sis3153_register_dma_write: req_read_length = %08X   ack_read_length
%08X    return_code =  %08X \n",req_read_length, ack_read_length, status);
else
       printf("udp_sis3153_register_dma_write: req_read_length = %08X   ack_read_length %08X
return_code =  %08X \n",req_read_length, ack_read_length, status);
```

2 Additional VME wrapper functions/methods

Based on the main VME read/write functions (single/DMA) several VME wrapper functions are provided. These functions allow access to the VME bus with standardised setup for Mode and Size:

Example Open/Close:

```
sis3153eth *vme_crate;
sis3153eth(&vme_crate, sis3153_ip_addr_string); // open Vme Interface device
vme_crate->vmeopen();  // open Vme interface
vme_crate->get_vmeopen_messages(char_messages, sizeof(char_messages), &nof_found_devices);  // open Vme interface
..
..
vme_crate->vmeclose();
```

Example with “vme_A16D16_write”:

```
unsigned int vme_address = 0x3800 ;  // A16 Address
return_code = vme_crate->vme_A16D16_write(vme_address, ushort_data);   //

return_codes:
      0x111: no Ethernet connection
      0x211: VME Buss Error
      0x212: VME Retry
      0x214: VME Arbitration Timeout
```

Example with ”vme_A16D16_sgl_random_burst_write”:

This function executes N vme_A16D16_write cycles with random addresses (not continuously addressing) with one UDP request.
Note: 1 =< N =< 64

```
unsigned int vme_address_array[64];
unsigned short ushort_vme_data_array[64];

nof_writes = 4;

vme_address_array[0] = 0x3800 ;
vme_address_array[1] = 0x3900 ;
vme_address_array[2] = 0x3A00 ;
vme_address_array[3] = 0x3000 ;

ushort_vme_data_array[0] = 0x1234 ;
ushort_vme_data_array[1] = 0x1111 ;
ushort_vme_data_array[2] = 0x2222 ;
ushort_vme_data_array[3] = 0x3333 ;
nof_writes = 4;

return_code = vme_crate->vme_A16D16_sgl_random_burst_write ( nof_writes, vme_address_array, ushort_vme_data_array);   //


return_codes:
       0x111: no Ethernet connection
       0x211: VME Buss Error
       0x212: VME Retry
       0x214: VME Arbitration Timeout
```

Open/Close functions/methods:

```
int vmeopen ( void );
int vmeclose( void );
int get_vmeopen_messages( CHAR* messages, UINT* nof_found_devices );
int get_vmeopen_messages(CHAR* messages, size_t size_of_messages, UINT* nof_found_devices);
```

VME read functions/methods:

```
int vme_IRQ_Status_read( UINT* data ) ;
int vme_IACK_D8_read (UINT vme_irq_level, UCHAR* data);

int vme_CRCSR_D8_read(UINT addr, UCHAR* data);
int vme_CRCSR_D16_read(UINT addr, USHORT* data);
int vme_CRCSR_D32_read(UINT addr, UINT* data);

int vme_A16D8_read(UINT addr, UCHAR* data);
int vme_A16D16_read(UINT addr, USHORT* data);
int vme_A16D32_read(UINT addr, UINT* data);

int vme_A24D8_read (UINT addr, UCHAR* data);
int vme_A24D16_read (UINT addr, USHORT* data);
int vme_A24D32_read (UINT addr, UINT* data);

int vme_A32D8_read (UINT addr, UCHAR* data);
int vme_A32D16_read (UINT addr, USHORT* data);
int vme_A32D32_read (UINT addr, UINT* data);
int vme_A32DMA_D32_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32BLT32_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32MBLT64_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2EVME_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2ESST160_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2ESST267_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2ESST320_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );

int vme_A32DMA_D32FIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32BLT32FIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32MBLT64FIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2EVMEFIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2ESST160FIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2ESST267FIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );
int vme_A32_2ESST320FIFO_read (UINT addr, UINT* data, UINT request_nof_words, UINT* got_nof_words );

int vme_A16D8_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, UCHAR* data_ptr);
int vme_A16D16_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, USHORT* data_ptr);

int vme_A24D8_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, UCHAR* data_ptr);
int vme_A24D16_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, USHORT* data_ptr);
int vme_A24D32_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, UINT* data_ptr);

int vme_A32D8_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, UCHAR* data_ptr);
int vme_A32D16_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, USHORT* data_ptr);
int vme_A32D32_sgl_random_burst_read(UINT nof_reads, UINT* addr_ptr, UINT* data_ptr);
```

VME write functions/methods:

```
int vme_CRCSR_D8_write(UINT addr, UCHAR data);
int vme_CRCSR_D16_write(UINT addr, USHORT data);
int vme_CRCSR_D32_write(UINT addr, UINT data);

int vme_A16D8_write(UINT addr, UCHAR data);
int vme_A16D16_write(UINT addr, USHORT data);
int vme_A16D32_write(UINT addr, UINT data);

int vme_A24D8_write (UINT addr, UCHAR data);
int vme_A24D16_write (UINT addr, USHORT data);
int vme_A24D32_write (UINT addr, UINT data);

int vme_A32D8_write (UINT addr, UCHAR data);
int vme_A32D16_write (UINT addr, USHORT data);
int vme_A32D32_write (UINT addr, UINT data);

int vme_A32DMA_D32_write (UINT addr, UINT* data, UINT request_nof_words, UINT* written_nof_words );
int vme_A32BLT32_write (UINT addr, UINT* data, UINT request_nof_words, UINT* written_nof_words );
int vme_A32MBLT64_write (UINT addr, UINT* data, UINT request_nof_words, UINT* written_nof_words );
int vme_A32DMA_D32FIFO_write (UINT addr, UINT* data, UINT request_nof_words, UINT* written_nof_words );
int vme_A32BLT32FIFO_write (UINT addr, UINT* data, UINT request_nof_words, UINT* written_nof_words );
int vme_A32MBLT64FIFO_write (UINT addr, UINT* data, UINT request_nof_words, UINT* written_nof_words );


int vme_A16D8_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, UCHAR* data_ptr);
int vme_A16D16_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, USHORT* data_ptr);

int vme_A24D8_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, UCHAR* data_ptr);
int vme_A24D16_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, USHORT* data_ptr);
int vme_A24D32_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, UINT* data_ptr);

int vme_A32D8_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, UCHAR* data_ptr);
int vme_A32D16_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, USHORT* data_ptr);
int vme_A32D32_sgl_random_burst_write(UINT nof_writes, UINT* addr_ptr, UINT* data_ptr);
```

3 Index