SIS3153
USB3.0/Ethernet to VME Interface

Ethernet UDP Addendum

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: SIS3153-M-eth-1-V107-ethernet-addendum.doc as of 14.11.2017

Revision Table:

Revision	Date	Modification
1.00	26.11.2014	First official release
1.01	01.12.2014	Minor touch up
1.02	05.06.2015	DHCP description, minor touch up
1.03	08.09.2015	release related to Firmware Version - V3153-1603 (0x3153-1603) modification: - Jumbo Frame support - add "Read Last Packet again" command (Retry) - change Ethernet UDP protocol (add packet identifier byte) **Note:** software has to be adapted (see: `sis3153ETH_vme_class`)
1.04	17.03.2016	Minor touch up in the Header description
1.05	20.03.2017	Minor touch up in the Header description: Status
1.06	04.08.2017	release related to Firmware Version - V3153-1604 (0x3153-1604)
1.07	14.11.2017	release related to Firmware Version - V3153-1605 (0x3153-1605) - increase Jumbo Frame packet size from 3200 Bytes to 7168 Bytes - add "List-Buffering"

- Table of contents

1 SIS3153 Ethernet interface

The SIS3153 provides the interface options USB3.0 and Ethernet to access the on board resources and the VME Bus via read and write access to the control space. Possible future extensions are Optical and VXS access.
This manual describes the **Ethernet interface**.

The SIS3153 USB3.0/Ethernet to VME interface is a single width 6U VME card.

Photograph of SIS3153

As we are aware, that no manual is perfect, we appreciate your feedback and will incorporate proposed changes and corrections as quickly as possible. The most recent version of this manual can be obtained by email from info@struck.de.
Feel free to apply for an account for our Dokuwiki documentation web page also.

The SIS3153 firmware page is at www.struck.de/sis3153firm.html .

Information on SIS3153 applications, firmware news and other related issues will be posted on our DAQ blog at www.struck.de/blog also.

ATTENTION
Observe Precautions for Handling
Electrostatic Sensitive Devices

1.1 Functionality

The SIS3153 interfaces the popular Universal Serial Bus (USB) and Ethernet to the VMEbus. The modules functionality comprises:

- USB (USB3/USB2/USB1) connectivity (Described in ‘SIS3153 – USB3 manual’)

- Ethernet (UDP) connectivity:
  - 1 Gbit/sec only
  - Jumbo frame support
  - DHCP support

- VME master read cycles:
  - IACK, CRCSR, A16/A24/A32
  - D8/D16/D32/BLT32/MBLT64/2eVME/2eSST160/2eSST267/2eSST320

- VME master write cycles:
  - CRCSR, A16/A24/A32
  - D8/D16/D32/BLT32/MBLT64/2eVME

- Direct List operation:
  - a list of VME read/write cycles can be send with one command

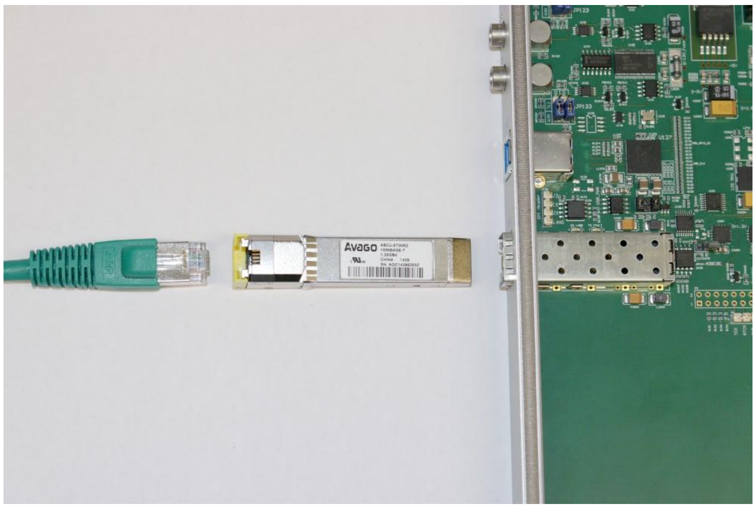
- Stack List operation (autonomous operation):
  - 8 lists of VME read/write cycles can be stored in a 8K x 32 Stack memory
  - Each stored list can be triggered by one of the following conditions: IRQ7-IRQ1, Input 1, Input 2, Timer 1, Timer 2 and software command
  - Incoming data (event) will be received via a second UDP-socket

- 2 digital front panel inputs (NIM or TTL level, select by jumper or register bits)

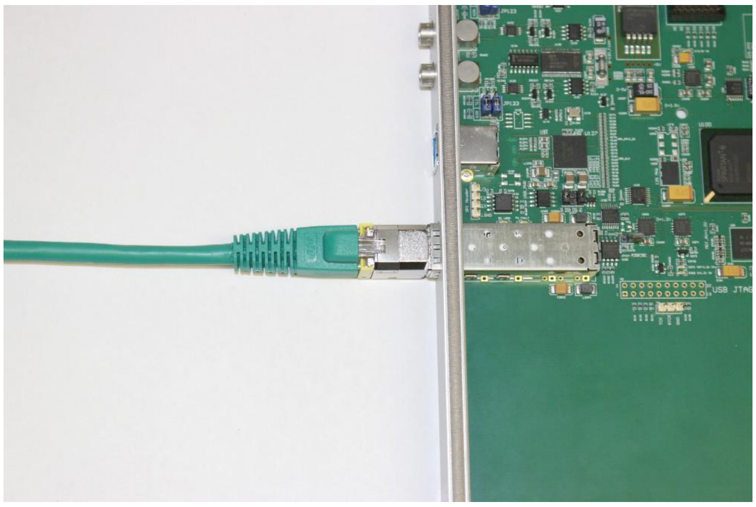
- 2 digital front panel outputs (NIM or TTL level , select by jumper or register bits)

2 Hardware prerequisite

2.1 1000BASE-T Copper SFP



A 1000BASE-T copper SFP transceiver has to be installed into the SFP cage of the SIS3153. Avago ABCU-5730RZ or compatible SFPs can be used, the Struck part number is 04333.



Switch **SW160-2** is used to activate/deactivate auto-negotiation for the Ethernet link. The switch has to be in **off** position for a copper connection or an optical connection over a media converter (like TP-LINK part number MC220L).

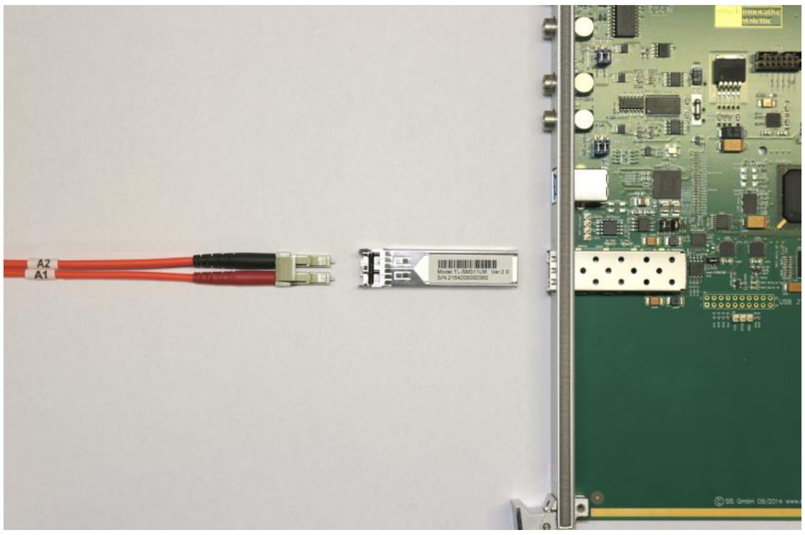
Note: the Ethernet interface supports 1Gbit/sec, only!

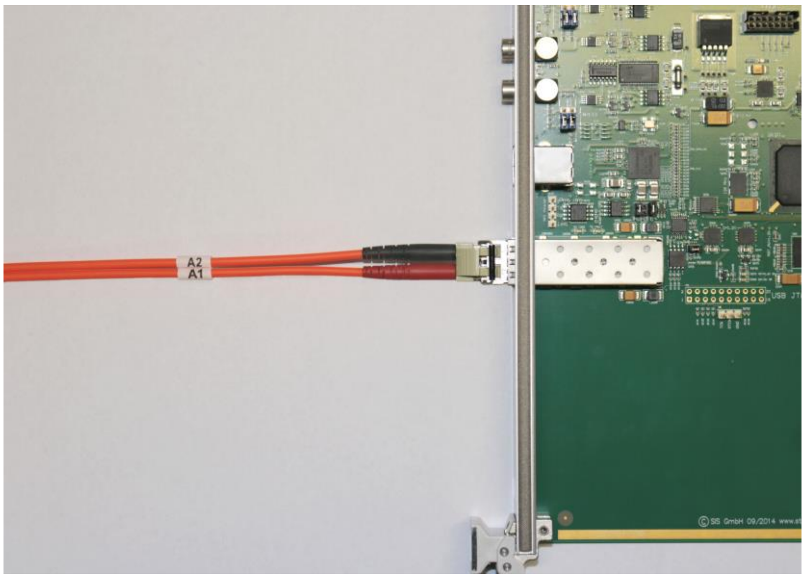
2.2 1000BASE-T Optical SFP

Optical decoupling of data acquisition devices from the readout system is mandatory in some applications (like installation on a platform under high voltage). In case of the SIS3153ETH interface optical decoupling can be established with readout over an optical Ethernet connection.

Starting with FPGA firmware revision V3153-xx05 Switch **SW160-2** is used to activate/deactivate auto-negotiation for the Ethernet link. The switch has to be in **on** position for an **optical connection** and in **off** position for a **copper connection** or an **optical connection over a media converter** (like TP-LINK part number MC220L).

The tests were performed with a DELOCK PCI Express 1x SFP Slot Gigabit LAN card (DELOCK part number 89368), two SFP link media (TP-LINK part number TL-SM311LM or Finisar FTLF8524P2BNV, Struck part number 03145) and a standard LC-LC multimode duplex50/125 µm fiber.





3 Computer UDP-socket connection

Each device (computer) on a network has an **IP address**, but the device’s network interfaces have **MAC addresses**. The UDP socket library (Linux and Windows) requires **IP addresses** (own computer IP address and Server IP address) to establish communication between the computer and the SIS3153.

Therefore the SIS3153 MAC Address has to be assigned to an IP address on the network.
This can be done manually with a **static ARP entry** on the computer, switch or router or automatically with a **DHCP (Dynamic Host Configuration Protocol)** request.

The DHCP request mode can be enabled/disabled by the Switch SW162-4.
The behavior of the Link (L) and Access (A) LEDs can be defined by the Switch SW162-3.
Please refer to the SIS3153 User manuall for further information's.

Note: The Link LED indicates after Power-Up and booting the FPGA the Ethernet Link Status:

- blinking with 4 Hz (every 0.25 sec) indicates no Ethernet Link
- blinking with 0.8 Hz (every 1.25 sec) indicates DHCP request busy
- blinking with **ping** access

The MAC address of the SIS3153 is: **00:00:56:15:3x:xx** (xxx: hex. serial number)

3.1 Static ARP entry example (manually IP assignment)

3.1.1 On a Windows computer

3.1.1.1 Example 1 (connected with a network)

SIS3153 serial number:	25 (0x19):
Desired SIS3153 IP address:	212.60.16.200 (SIS network)

Note: The used IP address must be locked on the DHCP-server in case of a "company network".

To add a static ARP entry for a local UDP node with an IP address of **212.60.16.200** that resolves to a MAC address of **00:00:56:15:30:19**, type the following at an **administrator** command prompt:

```
>arp –s  212.60.16.200  00-00-56-15-30-19
```
This command will bind the SIS3153 to the default LAN-interface.

```
>arp –s  212.60.16.200   00-00-56-15-30-19  -N 212.60.16.24
```
This command will bind the SIS3153 to the own (further) LAN-interface with the IP-address 212.60.16.24.

The following command shows all ARP-entries:
```
>arp -a
```

Helpful utility command: ipconfig and ping

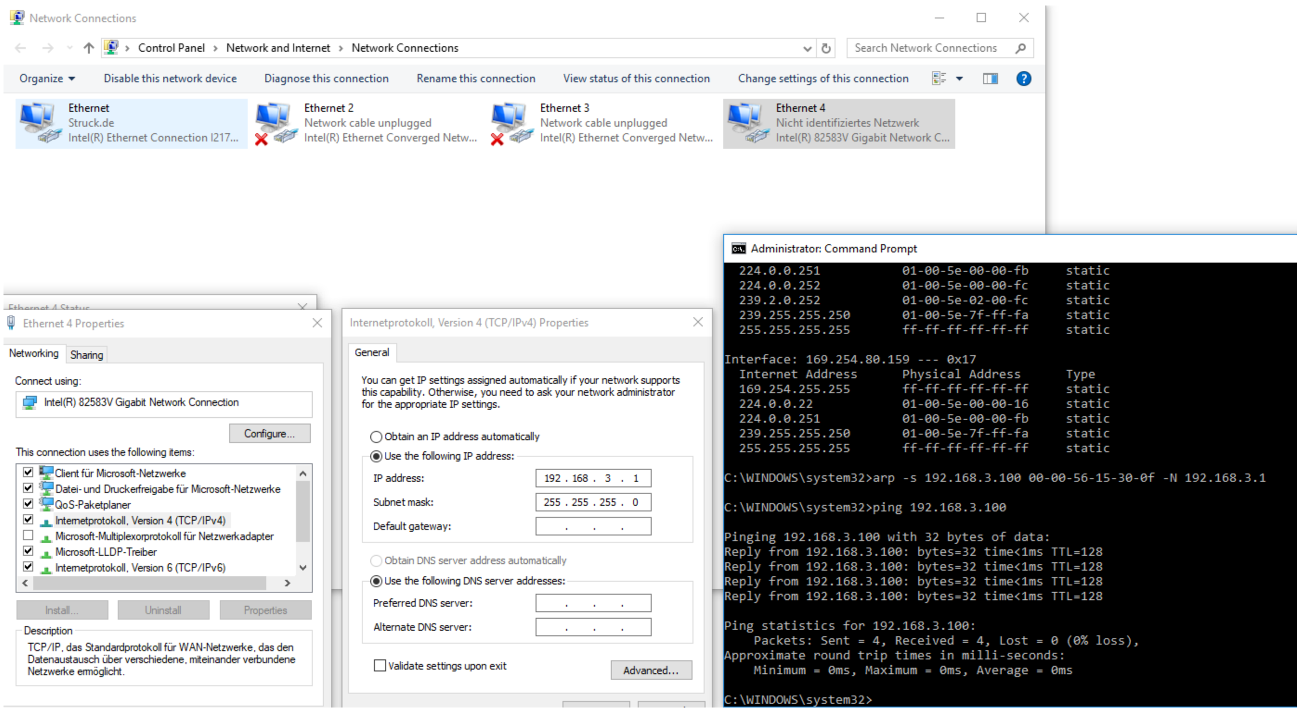
Ping is a computer network administration software utility used to test the reachability of a host (SIS3153) on an Internet Protocol network.

```
>ping 212.60.16.200
```
The SIS3153 will answer and the LED L will flash up.

3.1.1.2 Example 2 (direct connected with an additional network adapter)

SIS3153 serial number:	15 (0xF):
Desired SIS3153 IP address:	192.168.3.100

First configure the additional network adapter with a static IP-address.
For example network adapter "Ethernet 4" with the IP 192.168.3.1 (see screenshot below).


Network Connections
Control Panel > Network and Internet > Network Connections
Search Network Connections
Organize
Disable this network device
Diagnose this connection
Rename this connection
View status of this connection
Change settings of this connection
Ethernet 4 Status
Ethernet 4 Properties
Networking
Sharing
Configure...
Install...
Uninstall
Properties
Internetprotokoll, Version 4 (TCP/IPv4) Properties
General
Advanced...
Administrator: Command Prompt

Ethernet
Struck.de
Intel(R) Ethernet Connection I217...

Ethernet 2
Network cable unplugged
Intel(R) Ethernet Converged Netw...

Ethernet 3
Network cable unplugged
Intel(R) Ethernet Converged Netw...

Ethernet 4
Nicht identifiziertes Netzwerk
Intel(R) 82583V Gigabit Network C...

Connect using:
Intel(R) 82583V Gigabit Network Connection

This connection uses the following items:
- [✓] Client für Microsoft-Netzwerke
- [✓] Datei- und Druckerfreigabe für Microsoft-Netzwerke
- [✓] QoS-Paketplaner
- [✓] Internetprotokoll, Version 4 (TCP/IPv4)
- [ ] Microsoft-Multiplexorprotokoll für Netzwerkadapter
- [✓] Microsoft-LLDP-Treiber
- [✓] Internetprotokoll, Version 6 (TCP/IPv6)

Description
TCP/IP, das Standardprotokoll für WAN-Netzwerke, das den Datenaustausch über verschiedene, miteinander verbundene Netzwerke ermöglicht.

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

○ Obtain an IP address automatically
● Use the following IP address:

IP address:	192 . 168 . 3 . 1
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	. . .

○ Obtain DNS server address automatically
● Use the following DNS server addresses:

Preferred DNS server:	. . .
Alternate DNS server:	. . .

[ ] Validate settings upon exit

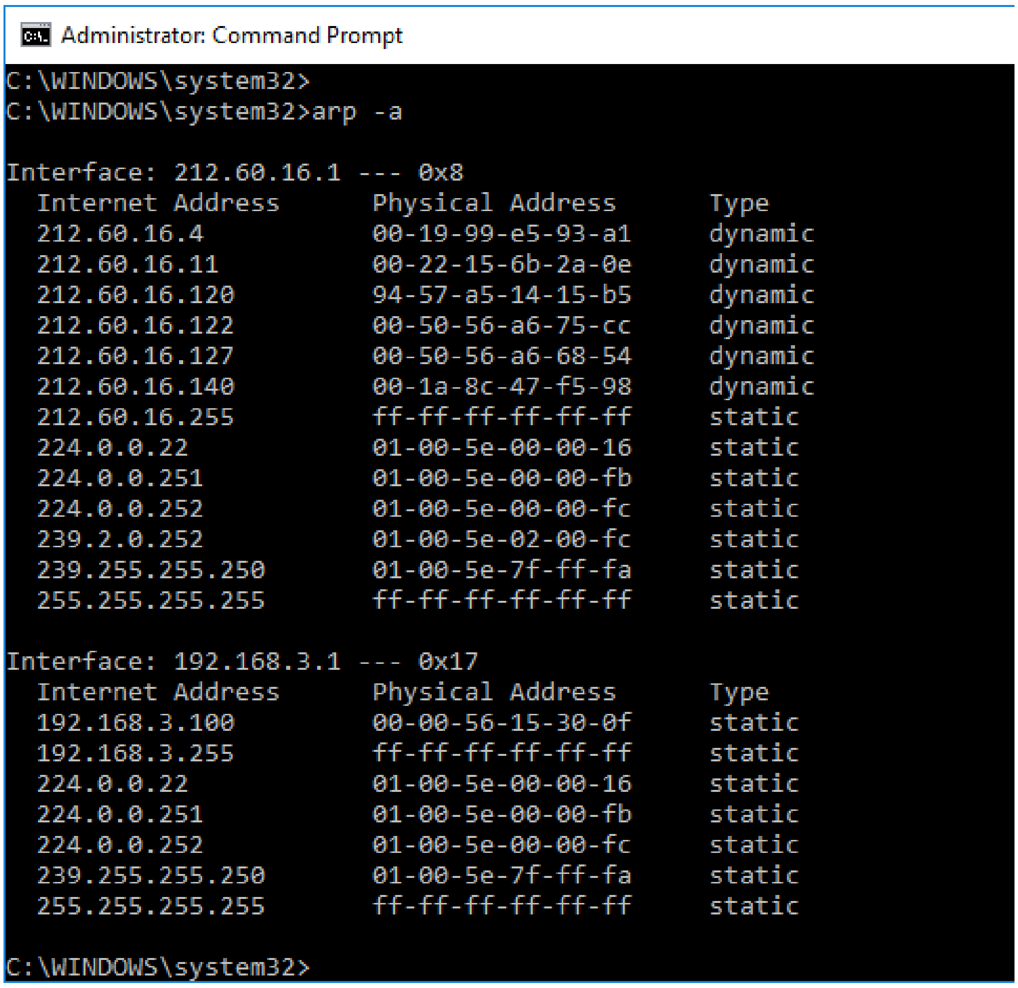
```
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.2.0.252           01-00-5e-02-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 169.254.80.159 --- 0x17
  Internet Address      Physical Address      Type
  169.254.255.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\WINDOWS\system32>arp -s 192.168.3.100 00-00-56-15-30-0f -N 192.168.3.1

C:\WINDOWS\system32>ping 192.168.3.100

Pinging 192.168.3.100 with 32 bytes of data:
Reply from 192.168.3.100: bytes=32 time<1ms TTL=128
Reply from 192.168.3.100: bytes=32 time<1ms TTL=128
Reply from 192.168.3.100: bytes=32 time<1ms TTL=128
Reply from 192.168.3.100: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.3.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\WINDOWS\system32>
```

To add a static ARP entry for a local UDP node with an IP address of **192.168.3.100** that resolves to a MAC address of **00:00:56:15:30:0f**, type the following at an **administrator** command prompt:

>arp –s 192.168.3.100 00-00-56-15-30-0f -N 192.168.3.1
This command will bind the SIS3153 to the network adapter (LAN-interface) with the IP-address **192.168.3.1**.

Ping is a computer network administration software utility used to test the reachability of a host (SIS3153) on an Internet Protocol network.
>ping 192.168.3.100
The SIS3153 will answer and the LED L will flash up.

The following command shows all ARP-entries:
>arp –a


Administrator: Command Prompt

```
C:\WINDOWS\system32>
C:\WINDOWS\system32>arp -a

Interface: 212.60.16.1 --- 0x8
  Internet Address      Physical Address      Type
  212.60.16.4           00-19-99-e5-93-a1     dynamic
  212.60.16.11          00-22-15-6b-2a-0e     dynamic
  212.60.16.120         94-57-a5-14-15-b5     dynamic
  212.60.16.122         00-50-56-a6-75-cc     dynamic
  212.60.16.127         00-50-56-a6-68-54     dynamic
  212.60.16.140         00-1a-8c-47-f5-98     dynamic
  212.60.16.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.2.0.252           01-00-5e-02-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

Interface: 192.168.3.1 --- 0x17
  Internet Address      Physical Address      Type
  192.168.3.100         00-00-56-15-30-0f     static
  192.168.3.255         ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static

C:\WINDOWS\system32>
```

3.1.2 On a Linux computer

SIS3153 serial number:	25 (0x19):
Desired SIS3153 IP address:	212.60.16.200 (SIS network)

Note: The used IP address must be locked on the DHCP-server in case of a “network”.

To add a static ARP entry for a local UDP node with an IP address of **212.60.16.200** that resolves to a MAC address of **00:00:56:15:30:19**, type the following at an administrator command prompt:

#arp -s 212.60.16.200 00:00:56:15:30:19
or
#arp –i eth0 -s 212.60.16.200 00:00:56:15:30:19
This command will bind the SIS3153 to the default LAN-interface eth0.

The following command shows all ARP-entires:
$arp -a

Helpful utility commands: ifconfig and ping

Ping is a computer network administration software utility used to test the reachability of a host (SIS3153) on an Internet Protocol network.
$ping 212.60.16.200
The SIS3153 will answer and the LED L will flash up.

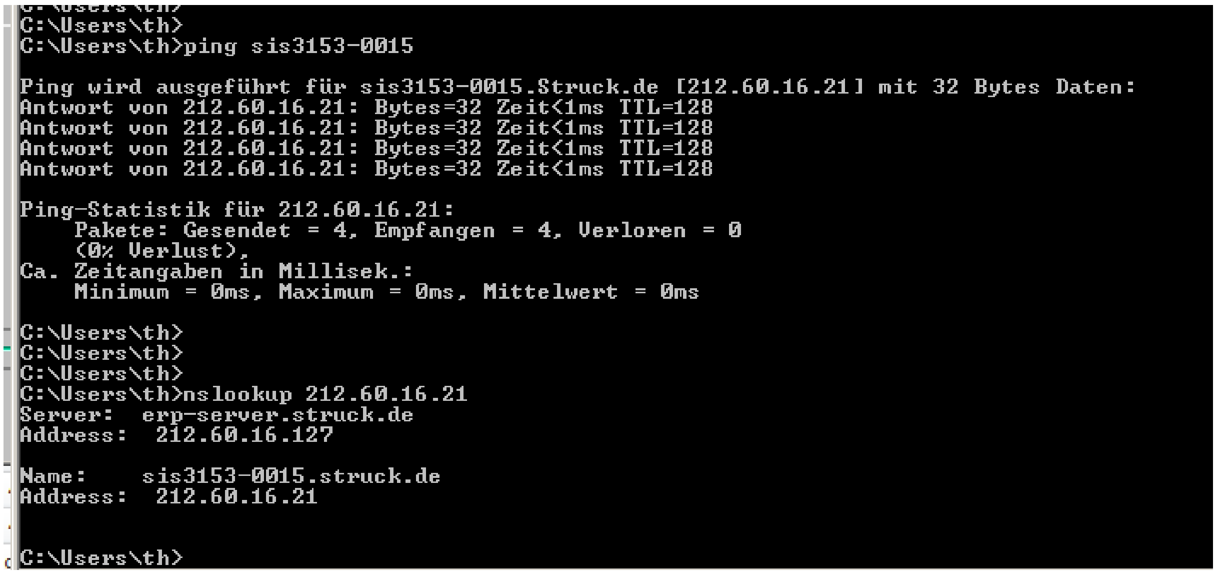
3.2 DHCP (automatically IP assignment)

If the DHCP option is enabled then the SIS3153 will request automatically an IP-address from the DHCP server. The LED L is blinking every 1.25 second while the DHCP request process is busy.

If a Name server is enabled it is possible to access the SIS3153 with its name and serial number: ping sis3153-nnnn (nnnn: four digits, decimal serial number).

Example with serial number 15:
> ping sis3153-0015

Windows:

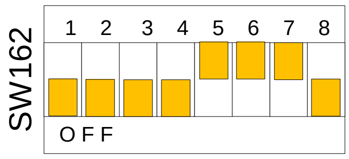


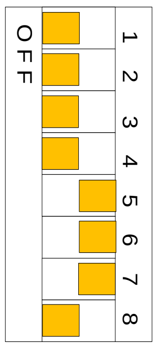
```
C:\Users\th>
C:\Users\th>ping sis3153-0015

Ping wird ausgeführt für sis3153-0015.Struck.de [212.60.16.21] mit 32 Bytes Daten:
Antwort von 212.60.16.21: Bytes=32 Zeit<1ms TTL=128
Antwort von 212.60.16.21: Bytes=32 Zeit<1ms TTL=128
Antwort von 212.60.16.21: Bytes=32 Zeit<1ms TTL=128
Antwort von 212.60.16.21: Bytes=32 Zeit<1ms TTL=128

Ping-Statistik für 212.60.16.21:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0
    (0% Verlust),
Ca. Zeitangaben in Millisek.:
    Minimum = 0ms, Maximum = 0ms, Mittelwert = 0ms

C:\Users\th>
C:\Users\th>
C:\Users\th>
C:\Users\th>nslookup 212.60.16.21
Server:  erp-server.struck.de
Address:  212.60.16.127

Name:    sis3153-0015.struck.de
Address:  212.60.16.21


C:\Users\th>
```

3.3 SW162 Dip switch System Controller/Reset Behavior

The 8 switches of SW162 are in charge of system controller function, reset behaviour and slave addressing (not implemented yet) as listed in the table below. Factory default settings are illustrated on the left hand side of the table.

Factory setting:

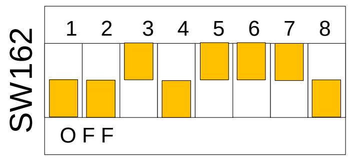


SW162	Off Function	On Function
	**1: Disable A32 slave addressing***	**Enable A32 slave addressing***
	2: Deactivate auto-negotiation (copper)	**Activate auto-negotiation (optical) ****
	3: Led L indicates USB activities	**Led L indicates Ethernet activities**
	4: DHCP disable	**DHCP enable (Ethernet)**
	5: System controller disable	**System controller enable**
	6: Disconnect FPGA reset to VME SYSRESET	**Connect FPGA reset to VME SYSRESET**
	7: Watchdog disable	**Watchdog enable**
	8: Disconnect VME SYSRESET from FPGA reset	**Connect VME SYSRESET to FPGA reset**

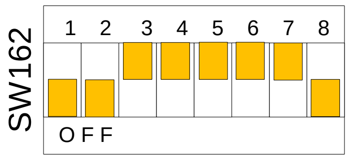
* reserved for future slave use
** with Firmware version V3153-1605 and higher

Note 1: do not set switch 6 and 8 to the on position at the same time (deadlock). The recommended setting is switch 6 on/8 off for a master/system controller and switch 6 off/8 on for non system controller units.

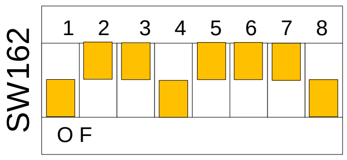
Led L indicates Ethernet activities:



Led L indicates Ethernet activities and DHCP enable:



Led L indicates Ethernet activities and “Optical SFP”:
(auto-negotiation enabled)



4 SIS3153 Internal Registers

The following register description gives only an overview of the Ethernet “additional” registers. The function of the remaining registers (blue) is described in detail in the SIS3153USB manual.

4.1 Register Space Address Map

Offset	R/W	Function/Register
0x0	R/W	USB Control/Status register
0x1	R	Module Id. and firmware version register
0x2	R	Serial Number register
0x3	R/W	LEMO IO control register
0x4	R/W	UDP protocol configuration register
0x10	R/W	VME Master Status/Control register
0x11	R	VME Master Cycle Status Register
0x12	R	VME Interrupt Status Register
0x100	KA	Key reset all
0x 0000 1000 .. 0x 0000 1FFF	R/W	Internal RAM
0x 0010 0000 .. 0x 001F FFFF	R	Address/Data Test space: Read Data = Read Address
0x 0020 0000 .. 0x 002F FFFF	R	Speed Test space: Read Data = Speed Counter The Speed Counter increments every 8ns (125 MHz)

The shorthand KA stands for key address. Write access with arbitrary data to a key address initiates the specified function

0x01000000	R/W	STACK-LIST 1 Configuration register
0x01000001	R/W	STACK-LIST 1 Trigger Source Select Register
..		..
..		..
0x0100000E	R/W	STACK-LIST 8 Configuration register
0x0100000F	R/W	STACK-LIST 8 Trigger Source Select Register
0x01000010	R/W	STACK-LIST Control register
0x01000011	W	STACK-LIST Trigger Command register
0x01000014	R/W	STACK-LIST Timer 1 Configuration register
0x01000015	R/W	STACK-LIST Timer 2 Configuration register
0x01000016	R/W	STACK-LIST “Dynamical Block Sizing for Reads” Configuration register
0x01000017	R	STACK-LIST “Dynamical Block Sizing for Reads” Test Readback register
0x 0180 0000	R/W	Stack-List RAM 8K x 32
..		
0x 0180 1FFF		

4.2 Register description

4.2.1 UDP Protocol Configuration register

```
#define SIS3153ETH_UDP_PROTOCOL_CONFIG          0x4          /* rd/wr D32 */
```

This register is used to control the UDP data packets**.**

Bit	31 - 9	8	7 - 5	4	3 - 0
Function	reserved	UDP transmit Data packet format bit	reserved	UDP transmit jumbo Packet enable bit	UDP transmit packet gap bits

UDP transmit Data packet format bit	Bit order into the UDP Data section
0	Little-endian (default, standard for the SIS Software)
1	Big-endian

UDP transmit packet gap value	Gap time between UDP packets
0	256 ns
1	512 ns
2	1 us
3	2 us
4	4 us
5	8 us
6	10 us
7	12 us
8	14 us
9	16 us
0xA	20 us
0xB	28 us
0xC	32 us
0xD	41 us
0xE	50 us
0xF	57 us

The power up default value reads 0x 00000000

4.2.2 UDP Stack-List N Configuration register

```
#define SIS3153ETH_STACK_LIST1_CONFIG         0x01000000 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST2_CONFIG         0x01000002 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST3_CONFIG         0x01000004 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST4_CONFIG         0x01000006 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST5_CONFIG         0x01000008 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST6_CONFIG         0x0100000A /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST7_CONFIG         0x0100000C /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST8_CONFIG         0x0100000E /* rd/wr D32 */
```

Bit	31 - 16	15 – 13	12 – 0 (8 K x 32)
Function	Stack List length - 1	reserved	Stack List Start address

4.2.3 UDP Stack-List N Trigger Source Select register

```
#define SIS3153ETH_STACK_LIST1_TRIGGER_SOURCE  0x01000001 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST2_TRIGGER_SOURCE  0x01000003 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST3_TRIGGER_SOURCE  0x01000005 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST4_TRIGGER_SOURCE  0x01000007 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST5_TRIGGER_SOURCE  0x01000009 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST6_TRIGGER_SOURCE  0x0100000B /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST7_TRIGGER_SOURCE  0x0100000D /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST8_TRIGGER_SOURCE  0x0100000F /* rd/wr D32 */
```

Bit	31 - 4	3 - 0
Function	reserved	Stack List Trigger Source Select value

Stack List Trigger Source Select value	Stack List Trigger Source
0	No Stack List Trigger (disabled)
1	VME IRQ 1
2	VME IRQ 2
3	VME IRQ 3
4	VME IRQ 4
5	VME IRQ 5
6	VME IRQ 6
7	VME IRQ 7
8	Timer 1
9	Timer 2
10 (0xA)	Stack List Trigger command
11 (0xB)	reserved
12 (0xC)	Input 1 rising edge
13 (0xD)	Input 1 falling edge
14 (0xE)	Input 2 rising edge
15 (0xF)	Input 2 falling edge

A write to one of these registers will save the IP-address and the UDP-port of the UDP-socket.
The reading data (events) of a List execution will be transmitted to this saved UDP-socket address.

4.2.4 UDP Stack-List Control register

```
#define SIS3153ETH_STACK_LIST_CONTROL          0x01000010 /* rd/wr D32 */
```

The control register is in charge of the control of most of the basic properties of the SIS3153 board in write access. It is implemented via a selective J/K register, a specific function is enabled by writing a 1 into the set/enable bit, the function is disabled by writing a 1 into the clear/disable bit (which location is 16-bit higher in the register). An undefined toggle status will result from setting both the enable and disable bits for a specific function at the same time.

Bit	Write Function	Read Function
31	Clear “List Multi Event Buffering Enable” bit	0
30	reserved	0
29	reserved	0
28	Clear “Force to send rest of Buffer Enable” bit	0
27	reserved	List Multi Event buffer word counter bit 11
26	reserved	List Multi Event buffer word counter bit 10
25	reserved	..
24	reserved	..
23	reserved	..
22	reserved	..
21	Clear “Incoming ICMP Disable” bit	..
20	Clear “Incoming Broadcast Disable” bit	..
19	reserved	..
18	Clear “Timer2 Operation Enable” bit	..
17	Clear “Timer1 Operation Enable” bit	List Multi Event buffer word counter bit 1
16	Clear “Stack List Operation Enable” bit	List Multi Event buffer word counter bit 0
15	Set “List Multi Event Buffering Enable” bit	Status “List Multi Event Buffering Enable” bit
14	reserved	Status
13	reserved	Status
12	Set “Force to send rest of Buffer Enable” bit	Status “Force to send rest of Buffer Enable” bit
11	reserved	Status reserved
10	reserved	Status reserved
9	reserved	Status reserved
8	reserved	Status reserved
7	reserved	Status reserved
6	reserved	Status reserved
5	Set “Incoming ICMP Disable” bit	Status “Incoming ICMP Disable” bit
4	Set “Incoming Broadcast Disable” bit	Status “Incoming Broadcast Disable” bit
3	reserved	Status reserved
2	Set “Timer2 Operation Enable” bit	Status “Timer2 Operation Enable” bit
1	Set “Timer1 Operation Enable” bit	Status “Timer1 Operation Enable” bit
0	Set “Stack List Operation Enable” bit	Status ”Stack List Operation Enable” bit

The power up value is 0x0

For example: enable (start) List operation

```
printf("enable stack-list operation\n");
// enable stack operation
data = 0x1;          // enable stack operation
data = data + 0x2;  // enable Timer 1 operation
data = data + 0x4;  // enable Timer 2 operation
if (multi_event_buffering_flag == 1) {
       data = data + 0x8000;  // enable List Multi Event Buffering mode
}
return_code = vme_crate->udp_sis3153_register_write(SIS3153ETH_STACK_LIST_CONTROL, data);
```

For example: disable (stop) List operation

```
printf("disable stack operation\n");
// disable stack operation
data = 0xf0000;              // disable stack operation and Timer
data = data + 0x80000000;   // disable List Multi Event Buffering mode
return_code = vme_crate->udp_sis3153_register_write(SIS3153ETH_STACK_LIST_CONTROL, data); //

// in case of List Multi Event Buffering mode
if (multi_event_buffering_flag == 1) {
   printf("force to push List-Buffer\n");
   data = 0x1000;                       // set "force to send/push rest of List-Buffer"
   vme_crate->udp_sis3153_register_write(SIS3153ETH_STACK_LIST_CONTROL, data); //
   data = 0x10000000;                          // clr "force to send/push rest of List-Buffer"
   vme_crate->udp_sis3153_register_write(SIS3153ETH_STACK_LIST_CONTROL, data); //

   do {
      vme_crate->udp_sis3153_register_read(SIS3153ETH_STACK_LIST_CONTROL, &data);
      rest_list_buffer_length = ((data >> 16) & 0xfff);// List Multi Event buffer word counter
   } while (rest_list_buffer_length > 0);
}
```

4.2.5 UDP Stack-List Trigger Command register

```
#define SIS3153ETH_STACK_LIST_TRIGGER_CMD 0x01000011 /* rd/wr D32 */
```

Bit	31 - 4	3 - 0
Function	reserved	Stack List Trigger command value

Stack List Trigger command value	command
0	Trigger Stack List 1 *
1	Trigger Stack List 2 *
2	Trigger Stack List 3 *
3	Trigger Stack List 4 *
4	Trigger Stack List 5 *
5	Trigger Stack List 6 *
6	Trigger Stack List 7 *
7	Trigger Stack List 8 *
8	reserved
..	..
..	..
15	Force to flush List-Buffer

* If “Stack List Trigger command” with the corresponding register “Stack-List N Trigger Source Select“ is selected .

4.2.6 UDP Stack-List Timer N Configuration register

```
#define SIS3153ETH_STACK_LIST_TIMER1           0x01000014 /* rd/wr D32 */
#define SIS3153ETH_STACK_LIST_TIMER2           0x01000015 /* rd/wr D32 */
```

Timer

Bit	31	30 - 16	15 - 0
Function	Watchdog enable	reserved	(Timer value – 1) x 100us

Timer value	Repetition Time
0	100 us
1	200 us
2	300 us
.	.
.	.
65635	6,5635 sec

Watchdog enable = 0: The List (enabled for Timer N) will be executed every (Timer value – 1) x 100us.

Watchdog enable = 1: Restart Timer with each transmitted packet
The List (enabled for Timer N) will be executed only if no other List was executed within the Timer N.

Example:

```
// setup Timer 1
// data = 10000 - 1; // 10.000 * 100us -> 1 sec
// data = 2000 - 1; // 2.000 * 100us -> 0.2 sec
data = 1000 - 1; // 1.000 * 100us -> 100 msec
return_code = vme_list->udp_sis3153_register_write(SIS3153ETH_STACK_LIST_TIMER1, data); //
```

4.2.7 UDP Stack-List "Dynamical Block Sizing for Reads" Configuration register

```
#define SIS3153ETH_STACK_LIST_DYN_BLK_SIZING_CONFIG  0x01000016 /* rd/wr */
```

Bit	31 - 26	25 - 24	23 - 0
Function	reserved	Shift-Left value	Bit-Mask

The value of a "marked VME Single Cycle" read will be masked with the "Bit-Mask" and then shift to the left by the "Shift-Level value" to get the desired length in Bytes for the next "marked VME Blocktransfer Read"(BLT32, MBL64, …).

Note:

- the Byte length must be a multiple of 4 Bytes (0x4, 0x8, 0xC, ..) for BLT32 Read
- the Byte length must be a multiple of 8 Bytes (0x8, 0x10, 0x14, .) for MBLT64/e2VME Read

Example, read from a VME Slave register a Fifo word counter which represents the number of 32-bit valid values:

```
- Conf register:  0x 0200 0fff  (0x 0200 0ffe for MBLT64 !)
- Read value:     0x YYYY Y040 -> "AND" with Mask = 0x00 0040 (64 x 32-bit words)
                               -> "Shift" with "2"= 0x00 0100 (256 Bytes)
```

The length of the next "marked VME Blocktransfer Read" is 256 Bytes.

The Write of this Configuration register can be done one time within a "Setup" with an "Internal Register Write cycle" (->udp_sis3153_register_write) or it can be part of the "List" (->list_generate_add_register_write), also.

4.2.8 UDP Stack-List "Dynamical Block Sizing for Reads" Test Readback register

```
#define SIS3153ETH_STACK_LIST_DYN_BLK_SIZING_TEST    0x01000017 /* rd only */
```

For test purpose, it is possible to read back the last "latched dynamical block sizing length" value in bytes.

5 FPGA Ethernet UDP protocol implementation

The XILINX Soft Tri-Mode Ethernet MAC core is used in the XC6SLX45T-3FGG484C Spartan 6 FPGA for the interface implementation.

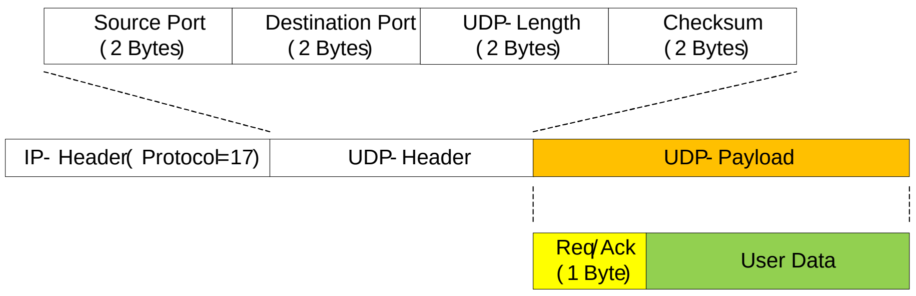
Note 1: the Ethernet interface supports 1Gbit/sec, only!

The Ethernet interface of the SIS3153 (server, acknowledge) is based on the UDP protocol for communication with the PC (client, requester).

The PC (client) sends a “request command” (Req), which is answered by the SIS3153 (server) with an “acknowledge” (Ack).

With FPGA Firmware V_1604 is a “Stack-List” logic implemented, which sends automatically, upon a trigger condition, “acknowledge” packets, only.

Ethernet protocol:



5.1 Request/Acknowledge access protocol

The data of the UDP-Payload packet is used for the communication with the SIS3153.
Request commands (Req) are implemented as listed in the table below:

Req	Command	Methods in `sis3153ETH_vme_class .cpp` and `.h`

0x20	Single Cycle read from register/VME space
	Single Cycle write to register/VME space
		`int udp_single_read(..)`
		`int udp_single_write(..)`

0x30	DMA (block) read from register/VME space
	DMA (block) write to register/VME space
		`int udp_sub_DMA_read(..)`
		`int udp_sub_DMA_write (..)`

0x40	Send a “Direct List” “Direct List” is a set of Single Cycle reads, Single Cycle writes, DMA (block) reads and DMA (block) writes.	`int udp_send_direct_list (..)`
0xEE	Read Last Packet again	`int udp_retransmit_cmd(..)`
0xFF	Reset command	`int udp_reset_cmd(void)`

All Register and VME access cycles are built on the requests 0x20 and 0x30, thus on :

```
private:
      int udp_single_read(unsigned int nof_read_words, UINT* addr_ptr, UINT* data_ptr);
      int udp_single_write(unsigned int nof_write_words, UINT* addr_ptr,
                                             UINT*  data_ptr);

      int udp_DMA_read ( unsigned int nof_read_words, UINT  addr, UINT* data_ptr,
                             UINT* got_nof_words );
      int udp_sub_DMA_read ( unsigned int nof_read_words, UINT  addr,
                                             UINT* data_ptr, UINT* nof_got_words);

      int udp_DMA_write(unsigned int nof_write_words, UINT addr, UINT* data_ptr,
                                             UINT* written_nof_words );
      int udp_sub_DMA_write(unsigned int nof_write_words,UINT addr,
                                             UINT* data_ptr,UINT* nof_written_words);
```

The public "Register calls" and "VME calls" are based on the above "private methods" which are part of sis3153ETH_vme_class.cpp and sis3153ETH_vme_class.h, also:

```
public:
	int udp_sis3153_register_read (UINT addr, UINT* data);
	int udp_sis3153_register_write (UINT addr, UINT data);
	int udp_sis3153_register_dma_read (UINT addr, UINT* data,
	                                       UINT request_nof_words,
	                                       UINT* got_nof_words );
	int udp_sis3153_register_dma_write (UINT addr, UINT *data,
	                                       UINT request_nof_words,
	                                       UINT* written_nof_words);

	..


	int vme_A32D32_read (UINT addr, UINT* data);
	int vme_A32D32_write (UINT addr, UINT data);

	int vme_A32_2EVMEFIFO_read (UINT addr, UINT* data,
	                         UINT request_nof_words, UINT* got_nof_words );

	..
```

Please have a look to sis3153ETH_vme_class.cpp and sis3153ETH_vme_class.h to see all defined VME Cycle methods.

These VME Cycle methods define the required "protocol Header" and interpret the "protocol Status" of the Response/Acknowledge packet(s).

Possible Return codes are:

```
/***************************************************************************************/
/* udp_single_write                                                                    */
/* possible ReturnCodes:                                                               */
/*       0     : OK                                                                    */
/*       0x111 : PROTOCOL_ERROR_CODE_TIMEOUT                                           */
/*       0x122 : wrong received Packet ID after N Retransmit and M Request commands    */
/*       0x211 : PROTOCOL_VME_CODE_BUS_ERROR                                           */
/*       0x2xx : PROTOCOL_VME_CODES                                                    */
/***************************************************************************************/

/***************************************************************************************/
/* udp_DMA_read, udp_sub_DMA_read                                                      */
/* possible ReturnCodes:                                                               */
/*       0     : OK                                                                    */
/*       0x111 : PROTOCOL_ERROR_CODE_TIMEOUT                                           */
/*       0x120 : wrong received Packet CMD after N Retransmit                          */
/*       0x122 : wrong received Packet ID after N Retransmit                           */
/*       0x124 : PROTOCOL_ERROR_CODE_HEADER_STATUS                                     */
/*                                                                                     */
/*       0x211 : PROTOCOL_VME_CODE_BUS_ERROR                                           */
/*                                                                                     */
/*       0x311 : PROTOCOL_DISORDER_AND_VME_CODE_BUS_ERROR                              */
/***************************************************************************************/
```

Read Last Packet again (`int udp_retransmit_cmd(..)`):

Lost packets are standard with UDP transfers and depend on Operating System, hardware (PC Ethernet interface, router hubs, …) and network load. So far, we observed "lost packets" with Window PCs in networks, only and only received packets are lost. Normally, a "Read Last Packet again" command will recover the "lost packet" behaviour. The retry received packet identifier equates to the requested packet identifier.
So far, we never observed "lost packets" with Linux and MAC PCs and with a point to point connection.
This command is used in the four functions/methods `udp_single_read, udp_single_write, udp_sub_DMA_read and udp_sub_DMA_write`, already. The number of "retries" is defined with the "`#define UDP_RETRANSMIT_RETRY      2`" in `sis3153ETH_vme_class.h` .

5.1.1 SIS3153 single register/VME space access protocol

0x20	Single Cycle read from register/VME space
	Single Cycle write to register/VME space
		`int udp_single_read(..)`
		`int udp_single_write(..)`

SIS3153 Read Request command (PC write)

- Read Request of N register values (max. 64)

Req = 0x20	Identifier (1 Byte)	Length* - 1 (2 Bytes)	Header (8 Bytes)	Address 0 (4 Bytes)	Address 1 (4 Bytes)		Address N-1 (4 Bytes)

Data section**

Protocol section

SIS3153 Read Response (PC read)

Ack = 0x2X	Identifier (1 Byte)	Status (1 Byte)	Data 0 (4 Bytes)	Data 1 (4 Bytes)		Data N-1 (4 Bytes)

SIS3153 Write Request command (PC write)

- Write Request of N register values (max. 64)

Req = 0x20	Identifier (1 Byte)	Length* - 1 (2 Bytes)	Header (8 Bytes)	Address 0 (4 Bytes)	Data 0 (4 Bytes)		Address N-1 (4 Bytes)	Data N-1 (4 Bytes)

Data section**

Protocol section

SIS3153 Write Response (PC read)

Ack = 0x2X	Identifier (1 Byte)	Status (1 Byte)	Dummy / VME Status (4 Bytes)

**) Lenght in 32-bit words of the protocol section.*

Length* - 1 (2 Bytes)	
Lower bits [7:0]	Upper bits [15:8]

***) Byetorder in the data section:*

Byte num.			
0	1	2	3

LSB MSB

Example:
unsigned int Address 0x12345678 →

78	56	34	12

5.1.2 SIS3153 DMA(block) register/VME space access protocol

0x30	DMA (block) read from register/VME space
	DMA (block) write to register/VME space
		`int udp_sub_DMA_read(..)`
		`int udp_sub_DMA_write (..)`

SIS3153 DMA Read Request command (PC write)

- Read Request of N values (N is to be defined in the Header length field, max. 262.144 Bytes)

Req = 0x30	Identifier (1 Byte)	Length* - 1 (2 Bytes)	Header (8 Bytes)	Address (4 Bytes)

Protocol section

SIS3153 DMA Read Response (PC read)

Ack = 0x3X	Identifier (1 Byte)	Status (1 Byte)	Data 0 (4 Bytes)	Data 1 (4 Bytes)	Data X-1 (4 Bytes)	packet 0
Ack = 0x3X	Identifier (1 Byte)	Status (1 Byte)	Data X (4 Bytes)	Data X+1 (4 Bytes)	Data Y-1 (4 Bytes)	packet 1
			: :			
Ack = 0x3X	Identifier (1 Byte)	Status (1 Byte)	Data Y (4 Bytes)	Data Y+1 (4 Bytes)	Data N-1 (4 Bytes)	packet n

Data section**

SIS3153 DMA Write Request command (PC write)

- Write Request of N values (max. 256)

Req = 0x30	Identifier (1 Byte)	Length* - 1 (2 Bytes)	Header (8 Bytes)	Address (4 Bytes)	Data 0 (4 Bytes)	Data N-1 (4 Bytes)

Data section**

Protocol section

SIS3153 DMA Write Response (PC read)

Ack = 0x3X	Identifier (1 Byte)	Status (1 Byte)	Dummy / VME Status (4 Bytes)

**) Lenght in 32-bit words of the protocol section.*

Length* - 1 (2 Bytes)	
Lower bits [7:0]	Upper bits [15:8]

***) Byetorder in the data section:*

Byte num.			
0	1	2	3

LSB MSB

Example:
unsigned int Address 0x12345678 →

78	56	34	12

5.1.3 Request Header description

The VME Cycle methods define/build the required "protocol Request Header",
see method `int vme_head_add(unsigned int nof_word)` in sis3153ETH_vme_class.cpp and sis3153ETH_vme_class.h for example .

The Header consists of 8 Bytes:

Header

Bit 7 Bit 0

Byte		
0	Length*[23:16]	
1	SPACE [3:0]	CTRL [3:0]
2	0xAA	
3	0xAA	
4	Length* [7:0]	
5	Length* [15:8]	
6	Mode [7:0]	
7	Mode [15:8]	

* Length in Bytes

Definition of SPACE[3:0]:

SPACE [3 : 0]	space
1	Register space
4	VME space
8	List Marker
9	List Header (start of List)
0xA	List Trailer (end of List)

Definition of CTRL[3:0]:

Bit	**Bit**	**Comment**
3	**WR** write request	0: read 1: write
2	**AUTO ADDRESS INCREMENT DISABLE** (FIFO Access)	0: auto address increment 1: no address increment
1-0	Data Size (Address Increment offset)	00: VME Byte, address increment by 1 01: VME 16-bit Word, address increment by 2 10: VME 32-bit Word, address increment by 4 11: address increment by 8

Definition of Mode [15:0]:

Bit	Bit	Comment
15	**reserved**	
14	**IACK**	0: normal VME read cycle 1: VME Interrupt Acknowledge read cycle
13	**Save dynamically sized length word (List Mode)**	0: normal VME read cycle 1: normal VME read cycle and save data for using as dynamically length during a "Dynamically sized VME BLT/MBLT Read"
12	**Dynamically sized VME BLT/MBLT Read (List Mode)**	0: VME BLT/MBLT read cycle with fixed length 1: VME BLT/MBLT read cycle with dynamically length
11	**Random Address Burst Read/Write cycles**	0: single VME read/write cycle 1: multiple VME read/write cycles with random addresses
10	**SWAP D32 words during MBLT64 read**	0: normal 32-bit word order during MBLT64 read 1: swapped 32-bit word order during MBLT64 read
9-8	**2eSST speed**	00: 160 Mbyte/sec 01: 267 Mbyte/sec 10: 320 Mbyte/sec 11: reserved
7	**reserved**	
6	**2eSST**	0: disable 2eSST (XAM code) 1: enable 2eSST (XAM code)
5-0	**VME Address modifier**	VME Address modifier (VME AM code)

5.1.4 Response Ack. and Status description

The VME Cycle methods interpret the "protocol Status" of the Response/Acknowledge packet(s) (sis3153ETH_vme_class.cpp).

Ack

During a VME cycle:

bit 3-0:	0x0 indicates a normal packet
	0x2 indicates a packet without valid data (Zero packet)
	0x4 indicates the last packet with valid data (Stop packet)

Status

bit 7:	1-bit Req. counter (toggles with each Req)
bit 6:	1 in case of protocol error (Request command packet error)
bit 5:	1 in case of access timeout
bit 4:	1 if the Ethernet interface has no grant
bits 3-0:	4-bit packet counter

Identifier

Request and response have the same packet identifier number. Therefore an unambiguous assignment of a response to a request is possible.

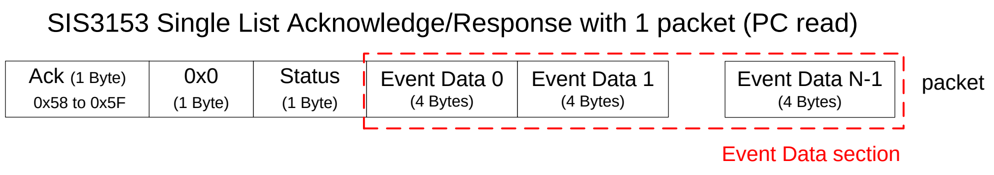
5.2 Stack-List "Acknowledge/Response" protocol

A triggered Stack List operation (autonomous operation) generates an "Event data" packet. The data structure and length of the "Event data" packet depend on the stored execution list (earlier programmed list of commands/cycles).
At least the "Event data" packet consists of two 32-bit words, a Header and a Trailer word. But it is also possible that an "Event data" packet contains a "large" number of 32-bit words which can' t send with one UDP-packet. In this case the "Event data" packet will be sent with multiple UDP-packets.

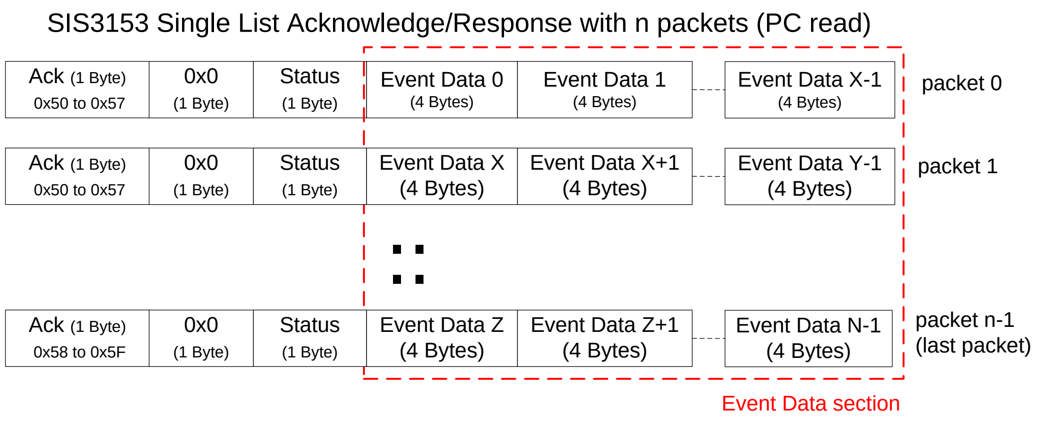
To decrease the number of sending "short Event data packets" a "Multi-Event Buffering" mode is implemented. If this mode is enabled (see "List Multi Event Buffering Enable" bit) the logic will buffer the "short Events" until an incoming event exceeds the UDP-packet size (1140 Bytes or 7168 Bytes in case of Jumbo frame is enabled) or if a command "Force to flush List-Buffer" is executed from a "Watchdog-Timer-List" for example.

5.2.1 Single List Event Acknowledge/Response

An "Event data" packet with a length less than the "UDP-packet" size (1140 Bytes or 7168 Bytes in case of Jumbo frame is enabled) will be sent with one UDP-packet.
The "Ack" Byte will be 0x58 (List 1) to 0x5F (List 8). These "Ack" values indicate a "Last-packet".



An "Event data" packet with a length greater than the "UDP-packet" size will be sent with multiple UDP-packets.
The "Ack" Byte will be 0x50 (List 1) to 0x57 (List 8) for the "not Last-packets". The "Ack" value for the "Last-packet" will be 0x58 (List 1) to 0x5F (List 8).

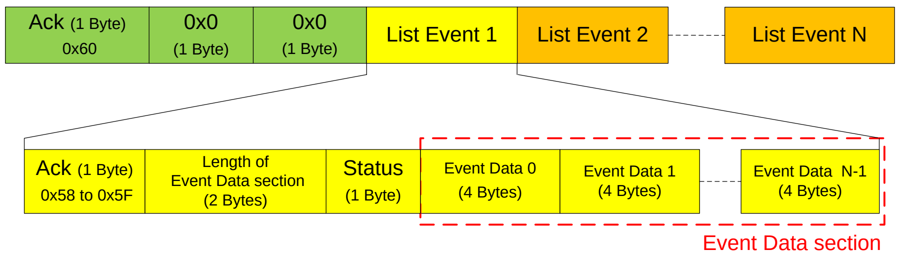


The length of a received UDP-packet is known. Therefore, the length of the "Event Data section" of each UDP-packet is known, also:
"Event Data section" length = UDP-packet length – 3 (in Bytes)

5.2.2 Multiple List Event Acknowledge/Response

The "Ack" Byte will be 0x60, which indicates a Multiple Event packet.
Each "List Event N" packet of the Multiple Event packet contains an "Ack" Byte again, which indicates the "triggered List" (0x58 -> List1, … 0x5F -> List8) following from 16-bit Length counter of the Event Data section. The length counter indicates the number of 32-bit words.

SIS3153 Multiple List Acknowledge/Response (PC read)



6 Stack-List operation

An 8K x 32 Stack memory is implemented to store up to 8 different Execution-Lists. An Execution-List is a sequence of VME read/write cycles, internal register read/write cycles and/or special cycles, like “write Marker word”. Each Execution-List can be triggered by one selected condition.
A triggered List pushes the “Event Data” via UDP immediately to the PC or it will be buffered until the size of a UDP packet is reached to reduce the number of UDP packets and to increase the Readout Data rate.

6.1 *Stack-List generation*

An Execution-List has to be embedded into a Header- and a Trailer. This means that the first List command has to be a “Header” command and the last List command has to be a “Trailer” command !

```
stack_list_buffer_ptr = 0; // start pointer of pc stack_list_buffer
stack_offset = 0;  // List 1

// start with Header !
vme_list->list_generate_add_header(&stack_list_buffer_ptr, uint_stack_list_buffer);

       // begin of user list cycles
       ....
       // for example: read VME A32D32
       addr = 0x31000004;  // SIS3316 version register
       vme_list->list_generate_add_vmeA32D32_read(&stack_list_buffer_ptr,
                                              uint_stack_list_buffer, addr);
       ....
       // end of user list cycles

// stop with Trailer !
vme_list->list_generate_add_trailer(&stack_list_buffer_ptr, uint_stack_list_buffer);
```

After a List sequence is generated, it has to be written into the 8K x 32 Stack Memory. The programmer has to care about that no Lists are overlaps and that each list are embedded into a “Header” and “Trailer”.

```
// write created list to stack memory
stack_addr = SIS3153ETH_STACK_RAM_START_ADDR + stack_offset;
stack_list_length = stack_list_buffer_ptr; //
vme_list->udp_sis3153_register_dma_write(stack_addr, uint_stack_list_buffer,
                                         stack_list_length, &written_nof_words); //
```

And the corresponding Stack-List Configuration register has to be programmed with Stack-List length and the address of the Stack Memory:

```
// configure Stack-List 1 Configuration register
data = stack_offset; // Stack-List 1 memory start address
data = data + (((stack_list_length - 1) & 0xffff) << 16); // stack list length
vme_list->udp_sis3153_register_write(SIS3153ETH_STACK_LIST1_CONFIG, data); //
```

6.2 Stack-List triggering

Each stored list can be triggered by one of the following conditions:

- VME IRQ7-IRQ1
- Lemo Input 1 or Input 2
- Timer 1 or Timer 2
- software command from PC

Please have a look to the description of the eight registers "UDP Stack-List N Trigger Source Select".

A write to one of these registers will save the IP-address and the UDP-port of the UDP-socket.
The reading data (event data) of a List execution will be transmitted to this saved UDP-socket address.

For example:

```
// Stack-List 3: select trigger source
data = 0xE; // enable trigger IN2 rising edge
vme_list->udp_sis3153_register_write(SIS3153ETH_STACK_LIST3_TRIGGER_SOURCE, data); //
```

6.3 *Stack-List Event Data section*

At least the “Event data” packet consists of two 32-bit words, a Header and a Trailer word. But it is also possible that an “Event data” packet contains a “large” number of 32-bit words which can’ t send with one UDP-packet. In this case the “Event data” packet will be sent with multiple UDP-packets.

Event Data Section

Bit 31 Bit 0

32-bit word				
Header 0	0xBB	List execution counter [23:0]		
1	depend on user-list			
..	..			
N-2	depend on user-list			
Trailer N-1	0xEE	VME blocktransfer read buserror counter [7:0]	VME read buserror counter [7:0]	VME write buserror counter [7:0]

The upper byte of the Header word is 0xBB. It indicates the “begin”. The lower 24-bits are a List execution counter. The counter will be incremented with each executed List. It can be used to check that no “Event data” packet are lost.

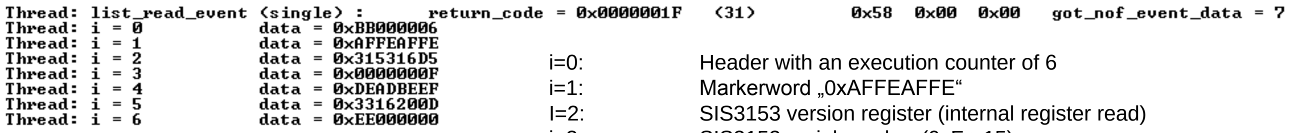
The upper byte of the Trailer word is 0xEE. It indicates the “end”. The following 3 bytes are VME BusError counters. These counters indicate the number of VME BusErrors during the execution of this List.

6.4 Stack-List Examples

6.4.1 Internal Register read/write, Marker word, VME D32 read

```
/**********************/
/*   List 1  (Timer)  */
/**********************/
stack_list_buffer_ptr = 0; // start pointer of pc stack_list_buffer
stack_offset = 0;  // List 1

// start with Header !
vme_list->list_generate_add_header(&stack_list_buffer_ptr, uint_stack_list_buffer); //

// begin of user list cycles
 // add marker word
 marker_word = 0xAFFEAFFE;
 vme_list->list_generate_add_marker(&stack_list_buffer_ptr, uint_stack_list_buffer, marker_word);

 // write internal register
 addr = SIS3153ETH_CONTROL_STATUS;
 data = 0x1; // set Led A
 vme_list->list_generate_add_register_write(&stack_list_buffer_ptr,uint_stack_list_buffer,addr,data);

 // read internal register
 addr = SIS3153ETH_MODID_VERSION;
 vme_list->list_generate_add_register_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // read internal register
 addr = SIS3153ETH_SERIAL_NUMBER_REG;
 vme_list->list_generate_add_register_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // add marker word
 marker_word = 0xDEADBEEF;
 vme_list->list_generate_add_marker(&stack_list_buffer_ptr, uint_stack_list_buffer, marker_word);

 // read VME A32D32
 addr = 0x31000004;  // SIS3316 version register
 vme_list->list_generate_add_vmeA32D32_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // write VME A32D32
 addr = 0x31000000;  // SIS3316 Control register
 data = 0x10001; // toggle Led U
 vme_list->list_generate_add_vmeA32D32_write(&stack_list_buffer_ptr,uint_stack_list_buffer,addr,data);

 // write internal register
 addr = SIS3153ETH_CONTROL_STATUS;
 data = 0x10000; // clr Led A
 vme_list->list_generate_add_register_write(&stack_list_buffer_ptr,uint_stack_list_buffer,addr,data);

 // write internal register
 addr = SIS3153ETH_STACK_LIST_CONTROL;
 data = 0x1000; // Set "Force to send rest of Buffer Enable" bit
 vme_list->list_generate_add_register_write(&stack_list_buffer_ptr,uint_stack_list_buffer,addr,data);
// end of user list cycles

// stop with Trailer !
vme_list->list_generate_add_trailer(&stack_list_buffer_ptr, uint_stack_list_buffer); //
```

The execution of the above List sequence (List 1) generates the following "Event Data" packet:



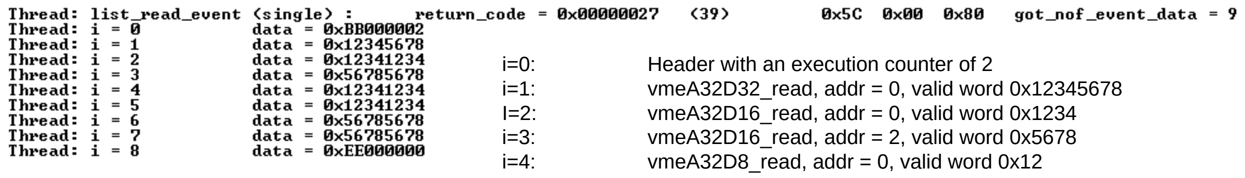
```
Thread: list_read_event (single) :      return_code = 0x0000001F   (31)         0x58  0x00  0x00   got_nof_event_data = 7
Thread: i = 0             data = 0xBB000006
Thread: i = 1             data = 0xAFFEAFFE
Thread: i = 2             data = 0x315316D5
Thread: i = 3             data = 0x0000000F
Thread: i = 4             data = 0xDEADBEEF
Thread: i = 5             data = 0x3316200D
Thread: i = 6             data = 0xEE000000
```

i=0:	Header with an execution counter of 6
i=1:	Markerword „0xAFFEAFFE“
I=2:	SIS3153 version register (internal register read)
i=3:	SIS3153 serial number (0xF->15)
i=4:	Markerword „0xDEADBEEF“
i=5:	SIS3316 version register (VME read)
i=6:	Trailer (no VME BusError detected)

6.4.2 VME D32 write and VME D32/D16/D8 reads

```
// start with Header !
vme_list->list_generate_add_header(&stack_list_buffer_ptr, uint_stack_list_buffer); //

// begin of user list cycles
 // write VME A32D32
 addr = 0x0;  // VME Memory module
 data = 0x12345678; // test 32-bit word
 vme_list->list_generate_add_vmeA32D32_write(&stack_list_buffer_ptr,uint_stack_list_buffer,addr,data);

 // read VME A32D32
 addr = 0x0;  // VME Memory module
 vme_list->list_generate_add_vmeA32D32_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // read VME A32D16
 addr = 0x0;  // VME Memory module
 vme_list->list_generate_add_vmeA32D16_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // read VME A32D16
 addr = 0x2;  // VME Memory module
 vme_list->list_generate_add_vmeA32D16_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //


 // read VME A32D8
 addr = 0x0;  // VME Memory module
 vme_list->list_generate_add_vmeA32D8_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // read VME A32D8
 addr = 0x1;  // VME Memory module
 vme_list->list_generate_add_vmeA32D8_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // read VME A32D8
 addr = 0x2;  // VME Memory module
 vme_list->list_generate_add_vmeA32D8_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // read VME A32D8
 addr = 0x3;  // VME Memory module
 vme_list->list_generate_add_vmeA32D8_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

// end of user list cycles


// stop with Trailer !
vme_list->list_generate_add_trailer(&stack_list_buffer_ptr, uint_stack_list_buffer); //
```

The execution of the above List sequence (List 5) generates the following "Event Data" packet:



```
Thread: list_read_event (single) :      return_code = 0x00000027   (39)         0x5C  0x00  0x80   got_nof_event_data = 9
Thread: i = 0             data = 0xBB000002
Thread: i = 1             data = 0x12345678
Thread: i = 2             data = 0x12341234
Thread: i = 3             data = 0x56785678
Thread: i = 4             data = 0x12341234
Thread: i = 5             data = 0x12341234
Thread: i = 6             data = 0x56785678
Thread: i = 7             data = 0x56785678
Thread: i = 8             data = 0xEE000000
```

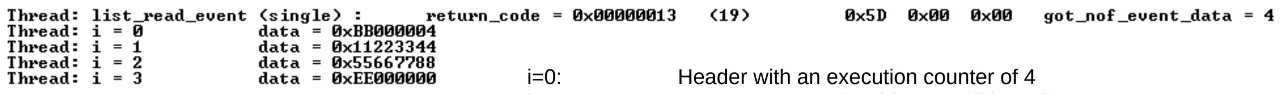
i=0:	Header with an execution counter of 2
i=1:	vmeA32D32_read, addr = 0, valid word 0x12345678
I=2:	vmeA32D16_read, addr = 0, valid word 0x1234
i=3:	vmeA32D16_read, addr = 2, valid word 0x5678
i=4:	vmeA32D8_read, addr = 0, valid word 0x12
i=5:	vmeA32D8_read, addr = 1, valid word 0x34
i=6:	vmeA32D8_read, addr = 2, valid word 0x56
i=7:	vmeA32D8_read, addr = 3, valid word 0x78
i=8:	Trailer (no VME BusError detected)

Note:
In case of VME D8 read cycles, the programmer has to know the lowest address bit (A0) of corresponding read cycle to take the "valid" data byte.

6.4.3 VME D16/D8 writes and VME D32 reads

```
// start with Header !
vme_list->list_generate_add_header(&stack_list_buffer_ptr, uint_stack_list_buffer); //

// begin of user list cycles

 // write VME A32D16
 addr = 0x0;  // VME Memory module
 data16 = 0x1122; //
 vme_list->list_generate_add_vmeA32D16_write(&stack_list_buffer_ptr,uint_stack_list_buffer,addr, data16);

 // write VME A32D16
 addr = 0x2;  // VME Memory module
 data16 = 0x3344; //
 vme_list->list_generate_add_vmeA32D16_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data16);


 // write VME A32D8
 addr = 0x4;  // VME Memory module
 data8 = 0x55; //
 vme_list->list_generate_add_vmeA32D8_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data8);

 // write VME A32D8
 addr = 0x5;  // VME Memory module
 data8 = 0x66; //
 vme_list->list_generate_add_vmeA32D8_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data8);

 // write VME A32D8
 addr = 0x6;  // VME Memory module
 data8 = 0x77; //
 vme_list->list_generate_add_vmeA32D8_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data8);

 // write VME A32D8
 addr = 0x7;  // VME Memory module
 data8 = 0x88; //
 vme_list->list_generate_add_vmeA32D8_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data8);


 // read VME A32D32
 addr = 0x0;  // VME Memory module
 vme_list->list_generate_add_vmeA32D32_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr);

 // read VME A32D32
 addr = 0x4;  // VME Memory module
 vme_list->list_generate_add_vmeA32D32_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr);

// end of user list cycles

// stop with Trailer !
vme_list->list_generate_add_trailer(&stack_list_buffer_ptr, uint_stack_list_buffer); //
```

The execution of the above List sequence (List 6) generates the following "Event Data" packet:



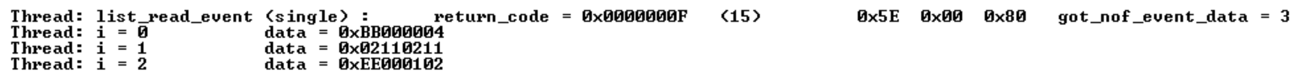
```
Thread: list_read_event (single) :      return_code = 0x00000013   (19)         0x5D  0x00  0x00   got_nof_event_data = 4
Thread: i = 0            data = 0xBB000004
Thread: i = 1            data = 0x11223344
Thread: i = 2            data = 0x55667788
Thread: i = 3            data = 0xEE000000
```

i=0:	Header with an execution counter of 4
i=1:	vmeA32D32_read, addr = 0, valid word 0x11223344
I=2:	vmeA32D32_read, addr = 4, valid word 0x55667788
i=3:	Trailer (no VME BusError detected)

6.4.4 VME D32 read/write with VME BusErrors

```
// start with Header !
vme_list->list_generate_add_header(&stack_list_buffer_ptr, uint_stack_list_buffer); //

// begin of user list cycles

 // write VME A32D32
 addr = 0x0;  // VME Memory module
 data = 0x12345678; // test 32-bit word
 vme_list->list_generate_add_vmeA32D32_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data);

 // read VME A32D32
 addr = 0xf0000000;  // not valid vme address !
 vme_list->list_generate_add_vmeA32D32_read(&stack_list_buffer_ptr, uint_stack_list_buffer, addr); //

 // write VME A32D32
 addr = 0xf0000000;  // not valid vme address !
 data = 0x12345678; // test 32-bit word
 vme_list->list_generate_add_vmeA32D32_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data);

 // write VME A32D32
 vme_list->list_generate_add_vmeA32D32_write(&stack_list_buffer_ptr, uint_stack_list_buffer, addr, data);


// end of user list cycles

// stop with Trailer !
vme_list->list_generate_add_trailer(&stack_list_buffer_ptr, uint_stack_list_buffer); //
```

The execution of the above List sequence (List 7) generates the following “Event Data” packet:



```
Thread: list_read_event (single) :      return_code = 0x0000000F   (15)        0x5E  0x00  0x80   got_nof_event_data = 3
Thread: i = 0           data = 0xBB000004
Thread: i = 1           data = 0x02110211
Thread: i = 2           data = 0xEE000102
```

i=0:	Header with an execution counter of 4
i=1:	vmeA32D32_read, addr = 0xF0000000, word 0x02110211 indicates VME BusError
I=2:	Trailer, indicates 1 VME read BusError and 2 VME write BusErrors

7 VME address modifier AM and XAM table

7.1 AM

AM	Function
0x3F	A24 supervisory block transfer (BLT)
0x3E	A24 supervisory program access
0x3D	A24 supervisory data access
0x3C	A24 supervisory 64-bit block transfer (MBLT)
0x3B	A24 non privileged block transfer (BLT)
0x3A	A24 non privileged program access
0x39	A24 non privileged data access
0x38	A24 non privileged 64-bit block transfer (MBLT)
0x37	A40BLT
0x36	Reserved
0x35	A40 lock command (LCK)
0x34	A40 access
0x33	Reserved
0x32	A24 lock command(LCK)
0x31	Reserved
0x30	Reserved
0x2F	Configuration ROM/Control&Status Register (CR/CSR)
0x2E	Reserved
0x2D	A16 supervisory access
0x2C	A16 lock command (LCK)
0x2B	Reserved
0x2A	Reserved
0x29	A16 non privileged access
0x28	Reserved
0x27	Reserved
0x26	Reserved
0x25	Reserved
0x24	Reserved
0x23	Reserved
0x22	Reserved
0x21	2e VME 3U
0x20	2e VME 6U
0x1F	User defined
0x1E	User defined
0x1D	User defined
0x1C	User defined
0x1B	User defined

0x1A	User defined
0x19	User defined
0x18	User defined
0x17	User defined
0x16	User defined
0x15	User defined
0x14	User defined
0x13	User defined
0x12	User defined
0x11	User defined
0x10	User defined
0xF	A32 supervisory block transfer (BLT)
0xE	A32 supervisory program access
0xD	A32 supervisory data access
0xC	A32 supervisory 64-bit block transfer (MBLT)
0xB	A32 non privileged block transfer (BLT)
0xA	A32 non privileged program access
0x9	A32 non privileged data access
0x8	A32 non privileged 64-bit block transfer (MBLT)
0x7	Reserved
0x6	Reserved
0x5	A32 lock command (LCK)
0x4	A64 lock command (LCK)
0x3	A64 block transfer (BLT)
0x2	Reserved
0x1	A64 single transfer access
0x0	A64 64-bit block transfer (MBLT)

7.2 6U 2eVME Extended Address Modifier Codes

XAM Code	Function
00	Reserved
01	A32/D64 2eVME Transfer
02	A64/D64 2eVME Transfer
03-FF	Reserved

7.3 3U 2eVME Extended Address Modifier Codes

XAM Code	Function
00	Reserved
01	A32/D32 2eVME Transfer
02	A40/D32 2eVME Transfer
03-FF	Reserved

Index