STR340

Struck Fastbus Interface

April 1996

Titel :	Struck Fastbus Interface
Projekt :	STR340/SFI
Autor :	Tino Haeupke (email: th@struck.de)
Version :	1.0
Erstellungsdatum :	1.8.95
letzte Änderung :	15.07.96 14:45
Filename :	S340_MAN.DOC

CONTENS

1 GENERAL INFORMATION 5

1.1 Purpose 5

1.2 Product Assistance 5

1.3 Note 5

2 SFI INTRODUCTION 6

2.1 FASTBUS transactions 8

2.2 Auxiliary Connector 8

2.3 Future extension 9

2.4 FASTBUS Master Port 10

3 SFI VME SLAVE ADDRESS MAP 12

4 SFI VME SLAVE REGISTER 13

4.1 Sequencer Status register (R) 13

4.2 FASTBUS Status register 1 (R) 14

4.3 FASTBUS Status register 2 (R) 15

4.4 FASTBUS Last Primary Address Store register (R) 15

4.5 Last Sequencer Protocol (Sequencer Keyaddress) register (R) 16

4.6 Next Sequencer Ram Address register (R/W) 17

4.7 FASTBUS Timeout register (R/W) 18

4.8 FASTBUS Arbitration Level register (R/W) 18

4.9 FASTBUS Protocol signal register (R) 19

4.10 Sequencer Fifo Flag and ECL/NIM Input register (R) 20

4.11 Out-Signal register (W) 21
4.11.1 VME Out-Signal register 21
4.11.2 Sequencer Out-Signal register (writeable from the sequencer !) 22

4.12 VME IRQ Source and MASK register 23

4.13 VME IRQ Level and Vector register 24

5 VME INTERRUPTS 25

6 FASTBUS MASTER SEQUENCER 26

6.1 Sequencer Enable (arm) 26

6.2 Sequencer FiFo Key Addresses 27
6.2.1 FASTBUS actions 27
6.2.1.1 Mode register and Limit counter (Sequencer Load) 28
6.2.1.2 DMA Status register and Word Counter (Sequencer Store) 28
6.2.1.3 FASTBUS command key address assignments 29
6.2.1.4 FASTBUS cycle command list examples 30
6.2.2 Sequencer Control actions 31

6.3 Sequencer RAM 32
6.3.1 Sequencer RAM Loading 32
6.3.2 Sequencer RAM List execution 32

6.4 Sequencer Programming Examples 33
6.4.1 FASTBUS initialization routine 33
6.4.2 Single FWC routine 33
6.4.3 Single FRD routine 34
6.4.4 Single FRDB routine 35
6.4.5 Load Next EVENT and FRDB List execution example 36
6.4.6 Load a Command List in the Sequencer Ram example 37
6.4.7 Event Readout example using a Ram List 38

7 AUXILIARY CONNECTOR INTERFACE 40

7.1 Auxiliary Connector schematic 40

7.2 User Trigger interface 41

7.3 FASTBUS blocktransfer Direct Mode interface 43

8 FRONT PANEL 44

8.1 LEDs 44

8.2 Inputs/Outputs 45
8.2.1 Schematic differential ECL In/Output 46
8.2.2 Schematic NIM In/Output 47

9 POWER REQUIREMENTS FOR THE SFI 48

10 JUMPER 49

1 GENERAL INFORMATION

1.1 PURPOSE

The **SFI** (**S**TRUCK **F**ASTBUS **I**nterface) is a FASTBUS Master based on a standard VMEbus CPU. That allows the user to choose a wellknown CPU and operating system and to drive FASTBUS with that.

1.2 PRODUCT ASSISTANCE

We afford assistance to SFI users to allow for a short implementation time for user specifier application by the following means:

* Example code for OS9 (K&R style C) is supplied with SFI

* Address questions relating SFI hard- and software to **SFI@struck.de**

* Send an email containing your SFI's serial number and the used VMEbus CPU and operating system to SFI@struck.de . We would like to use your email address for information exchange with other users (only with your permission).

1.3 NOTE

***** The cover sheet is hard anodized so that the surface is insulating. Additional, the inside VME area has been provided with a plastic foil in addition. Please, slide in (or out) carefully the left VME module to protect the foil.

***** **VME BUS Daisy chain**
If no VMEbus module is installed in the second VMEbus Slot (right VMEbus slot side) of the SFI2, so you have to place a „dummy“ Daisy Chain VMEBus connector on the P1 connector (upper one).

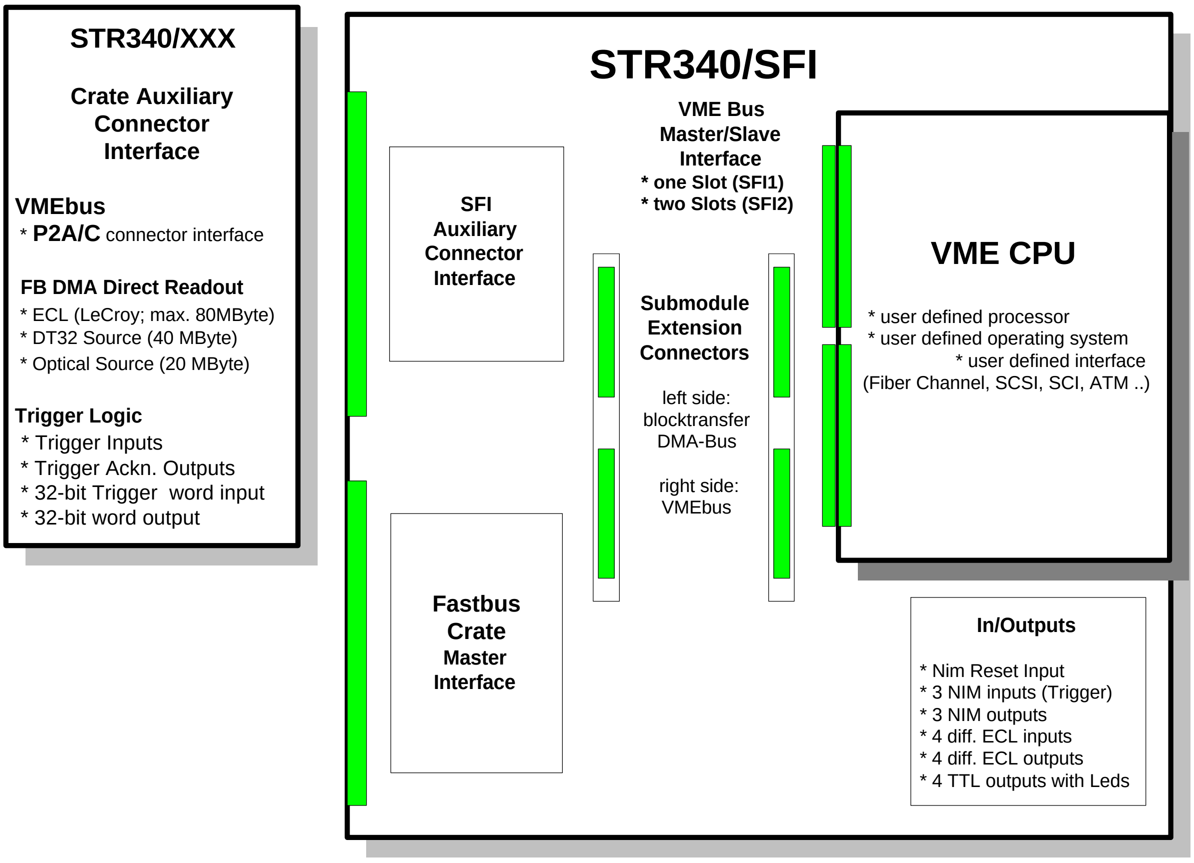
***** **Auxiliary Connector**
The Auxiliary connector is **used** on the SFI. Make sure that no incompatible auxiliary board is plugged in.

2 SFI INTRODUCTION

The SFI is a powerful and simple Fastbus Master and Readout Controller **using commercial off-the-shelf 6U VME CPU** (or Master) module as local intelligence. Using already established real time operating systems on these modules (like OS-9, VxWorks, LynxOS or even MS-Windows Derivates), programming of FASTBUS applications is simplified and allow direct code portation.

One or two (more on request) 6U VME modules can be plugged into the SFI FASTBUS module. For example, you can operate a powerful VME configuration consisting of a CPU (68040, R3000, Power-PC or DSP based) together with a communication interface module (like Fibre channel, ATM, VICbus etc.) locally and directly in a FASTBUS crate.
Another possibility is to operate a single VME processor with an on-board communication interface based for instance upon PCI local bus mezzanines (available from vendors are PCI-ATM, PCI-Fibre channel, PCI-SCI etc.).

The interface between the VME-CPU and the FASTBUS Segment guarantees highest possible performance in speed, flexibility and reliability.


STR340/XXX
Crate Auxiliary Connector Interface
VMEbus
* P2A/C connector interface
FB DMA Direct Readout
* ECL (LeCroy; max. 80MByte)
* DT32 Source (40 MByte)
* Optical Source (20 MByte)
Trigger Logic
* Trigger Inputs
* Trigger Ackn. Outputs
* 32-bit Trigger word input
* 32-bit word output
STR340/SFI
VME Bus Master/Slave Interface
* one Slot (SFI1)
* two Slots (SFI2)
SFI Auxiliary Connector Interface
Submodule Extension Connectors
left side: blocktransfer DMA-Bus
right side: VMEbus
VME CPU
* user defined processor
* user defined operating system
* user defined interface
(Fiber Channel, SCSI, SCI, ATM ..)
Fastbus Crate Master Interface
In/Outputs
* Nim Reset Input
* 3 NIM inputs (Trigger)
* 3 NIM outputs
* 4 diff. ECL inputs
* 4 diff. ECL outputs
* 4 TTL outputs with Leds

file: s340_blk.vsd

Blockdiagram: SFI

Blockdiagram: **Data Flow**

2.1 FASTBUS TRANSACTIONS

FASTBUS transactions are executed by a high speed hardware sequencer logic. Using a simple programming model FASTBUS transactions are defined by a list of VME key address cycles. In order to gain in speed VME key address lists and data are passed through a FIFO to the sequencer. Applying this technique FASTBUS cycles and instructions can be executed already during the write list phase in an overlap mode. Synchronisation of list execution with external NIM/ECL signals is implemented too.

FASTBUS blocktransfer read sequences are transformed into VME blocktransfer sequences or single cycles (if the **VME mode** is enabled). In this case, the SFI on-board DMA controller passes the FASTBUS data through a derandomizing FASTBUS-data FIFO directly into the VME-CPU memory (partitioned in multiple 256 byte blocks in the case of large blocks). **In this case, the SFI controller acts as the (internal) "VME Master".**

Another important feature is the **direct mode** via the FASTBUS Auxiliary connector. In this mode, the FASTBUS data, reading in blocktransfer mode, are pushed through Buffer/Latches to the FASTBUS Auxiliary connector controlled via several protocol lines. A simple logic on an auxiliary module converts the FASTBUS data into strobed differential ECL outputs (LeCroy ECLine compatible), into serial optical data stream (STR333/S) or into a private data stream format. Protocol compatibility to LeCroy ECLine offers to pass the data stream into buffer memories and/or DSP based Input FiFos (e.g. Struck VME STR8090 or FASTBUS STR370 DSP96002 Systems).

The **VME mode and the direct mode** can also enabled at the same time. In this case data are pushed into the VME slave memory (CPU or other VME module) **and** to the Auxiliary connector interface (**SPY Mode**).

A complete FASTBUS crate scan and readout sequence consisting of an arbitrary mixture of FASTBUS transactions (typically address locks, secondary address writes, multi block reads, single block reads, local broadcasts etc.) is executed by the list sequencer in one "stream".

2.2 AUXILIARY CONNECTOR

The Auxiliary connector provides the following features:

* connection to the VME P2 A/C connector from the leftside VME modul
* FASTBUS blocktransfer DIRECT Mode
* user Trigger interface (write/read 32-bit word, receive signals, assert signals)

2.3 FUTURE EXTENSION

In order to offer even higher flexibility and speed, VMEbus and FASTBUS DMA Bus submodule extension SMD connectors are implemented. These SMD connectors allow to add different kind of user defined logic.
For example a local memory buffer with up to 128Mbyte of fast dual ported DRAM, which is able to accept FASTBUS data streams at a sustained rate of 40Mbyte/s. Using this buffer it is possible to cover up to three seconds of data taking at highest FB speed for instance during a accelerator beam spill. Emptying of the buffer, for instance during a spill pause, can be performed by the VME CPU with D64 blocktransfer read cycles, passing the data into communication channels like Fibre Channel, ATM etc., or for example a memory look-up table to filter and convert the data, eg..

Blockdiagram: example of a look-up table to filter and convert data

2.4 FASTBUS MASTER PORT

Due to the special attributes and complexity of FASTBUS operations, especially compound cycles like <*address lock, secondary address write, data read, address release*> and also complex status responses on SS-lines, the SFI does not use a direct, synchronous transformation method between VME action(s) and FASTBUS action(s). In this case route/mapping look up tables and SS-response handling methods would cause many complications.
Instead, the SFI implements a transformation method between VME and FASTBUS actions using so called **Key Address Technique**.
Key Address Technique means that certain VME address keys are used to execute ("to open") certain types of FASTBUS transactions. The actual transaction is defined in detail by accompanying data (for instance the key address operation to establish an address lock has to be accompanied by the geographical or logical address in the data part of the VME cycle.) VME Key address cycles are even used to program and define the SFI DMA controller (Setting word/limit counter, address pointer). A set of successive key address cycles forms a **list of SFI/FASTBUS transactions**, which is interpreted and executed by the sequencer logic.

In order to increase the FASTBUS system performance the VME key address cycles and the FASTBUS list interpreter are not directly coupled (or synchronized). Instead, the key address information together with its data information are written into a decoupling FIFO (or RAM, see below) at highest VME speed. As soon as the first words have rippled through the FIFO, the list sequencer starts the corresponding FASTBUS or internal actions in an overlapping mode.

Another advantage of this technique is, that error conditions (Time out, or SS-response >0) are not handled by special CPU Bus Errors or Traps, instead the FASTBUS Master sequencer handles the exceptions (also SI support) and informs the VME CPU by polling a register or an VME Interrupt. FASTBUS and Sequencer status registers can be read to achieve the required status information.

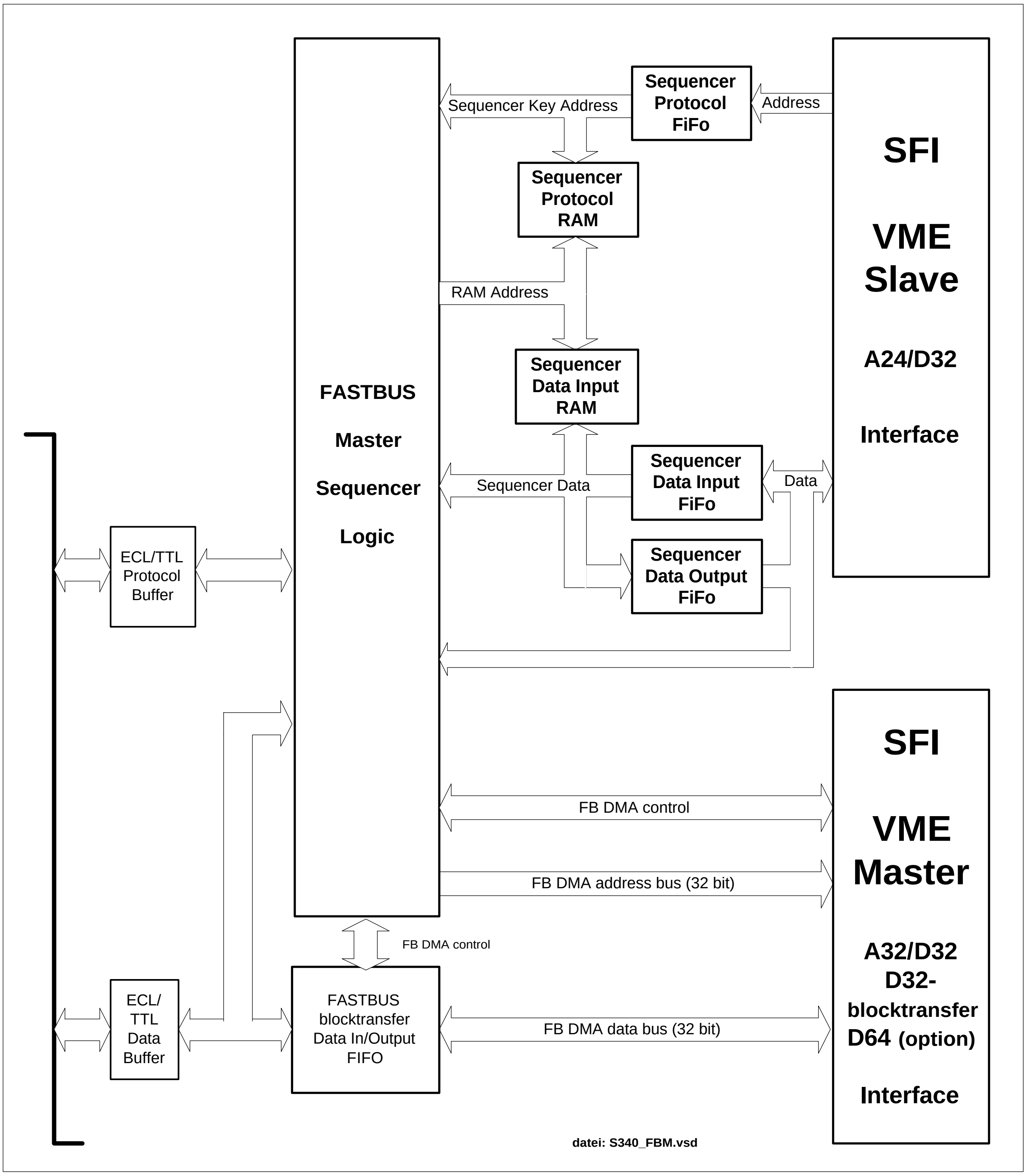
Data, which are read from FASTBUS in single cycle mode (not with the autonomous blocktransfer), are passed directly into the sequencer Data Output FIFO (SEQ2VME Fifo).

With the sequencer FIFO key addresses it is also possible to execute the following special functions:

- start autonomous blocktransfer (High speed FB blocktransfer, DK(t) to DS(t) =35ns to 45 ns)
- load DMA limitcounter (=max. blocklength)
- load DMA address pointer (VME Slave address; A32)
- store DMA wordcounter, Timeout information and SS-response in the Data Output Fifo (SEQ2VME Fifo)
- store only DMA wordcounter
- generate VME interrupt
- set/clear external signals (NIM, ECL)
- set/clear Leds

Sequencer RAM

In parallel to the FIFO a Sequencer List RAM is implemented in order to store a complete list and to start the execution of the entire list without rewriting the list elements into the FIFO. This feature allows for instance to initialize a compound FB crate readout sequence of multiple TDC´s and ADC´s with one single VME cycle.



Blockdiagram: **FASTBUS Master Sequencer**

3 SFI VME SLAVE ADDRESS MAP

The SFI VME Slave logic can be addressed with the VME Address Modifier AM=39 and AM=3D **(A24/D32).**
The VME Base Address (b) is to choose with the 4-bit switch SW850.
Factory setting: SW850 = E ==> $E00000.
For processor data cache and Compiler optimizer problems it is helpful to use different addresses (different x) for read and write cycles !

A24 address (A24/D32)	R/W	function (x = don't care)
$ b0 1x 00	**R**	**Internal FASTBUS I/O Bus**
$ b0 1x 04	**R**	**FASTBUS Last Primary Address Store register**
$ b0 1x 00	**W**	**Write VME Out-Signal register (LEDs, ECL, NIM, AUX)**
$ b0 1x 04	**W**	**Key Address: clear VME Out-Signal register**
$ b0 1x 10	**W**	**Write to internal AUX-port register**
$ b0 1x 14	**W**	**Key Address: generate AUX_B40 pulse**
$ b0 2x 00	**R**	**FASTBUS Timeout register**
$ b0 2x 04	**R**	**FASTBUS Arbitration Level register**
$ b0 2x 08	**R**	**FASTBUS Protocol Signal register**
$ b0 2x 0C	**R**	**Sequencer Fifo Flag and ECL/NIM Input register**
$ b0 2x 10	**R**	**VME IRQ Level and Vector register**
$ b0 2x 14	**R**	**VME IRQ Source and Mask register**
$ b0 2x 18	**R**	**Next Sequencer Ram Address register**
$ b0 2x 1C	**R**	**Last Sequencer Protocol (Sequencer Keyaddress) register**
$ b0 2x 20	**R**	**Sequencer Status register**
$ b0 2x 24	**R**	**FASTBUS Status register 1 (Arbitration/Primary Status)**
$ b0 2x 28	**R**	**FASTBUS Status register 2 (Data cycle/DMA Status)**
$ b0 2x 00	**W**	**FASTBUS Timeout register**
$ b0 2x 04	**W**	**FASTBUS Arbitration Level register**
$ b0 2x 08	**W**	**reserved**
$ b0 2x 0C	**W**	**reserved**
$ b0 2x 10	**W**	**VME IRQ Level and Vector register**
$ b0 2x 14	**W**	**VME IRQ Mask register**
$ b0 2x 18	**W**	**Next Sequencer RAM Address register**
$ b0 2x 1C	**W**	**Key Address : Reset register group LCA2 (xx 20 xx)**
$ b0 2x 20	**W**	**Key Address: Sequencer Enable**
$ b0 2x 24	**W**	**Key Address: Sequencer Disable**
$ b0 2x 28	**W**	**Key Address: Sequencer Ram Load Enable**
$ b0 2x 2C	**W**	**Key Address: Sequencer Ram Load Disable**
$ b0 2x 30	**W**	**Key Address: Sequencer Reset**
$ b0 2x 38	**W**	**Key Address: Clear Sequencer CMD Flag**
$ b0 4x xx	**R**	**Read SEQ2VME Fifo**
$ b1 xx xx	**W**	**Write VME2SEQ Fifo**

4 SFI VME Slave Register

4.1 Sequencer Status register (R)

This register is a read only register for the Sequencer Status information.

bit	name	function
31	reserved	“1“
..	..	..
16	reserved	“1“
15	SEQ_DONE	indicates that the Sequencer is in idle Loop or Sequencer is disabled
14	SEQ_BUSY	indicates that the Sequencer is BUSY (valid only if SEQ_ENABLE)
13	SEQ_IDLE	indicates that the Sequencer is IDLE (Enabled but no new Command)
12	reserved	“0“
11	CMD_CTR	indicates that the Sequencer works on a Control command
10	CMD_PRIM	indicates that the Sequencer works on a Arbitration or Primary Address command
9	CMD_DATA	indicates that the Sequencer works on a FB Data Cycle
8	CMD_DMA	indicates that the Sequencer works on a FB DMA Blocktransfer
7	SEQ_DMA_ERROR	indicates an error during FB DMA Blocktransfer Cycle
6	SEQ_DATA_ERROR	indicates an error during FB Data Cycle
5	SEQ_PRIM_ERROR	indicates an error during Arbitration or Primary Address Cycle
4	SEQ_CMD_ERROR	indicates an invalid Sequencer Command (undefined Sequencer Key address)
3	SEQ_WAIT	Sequencer waits for an external or internal event (reserved)
2	SEQ_RAM_LOAD	Sequencer is in the Sequencer RAM Load Mode
1	SEQ_RAM_ENABLE	Sequencer is enabled and in Sequencer Ram Mode
0	SEQ_ENABLE	Sequencer is enabled (Fifo or RAM Mode)

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2“ : 0xFFFF0000

4.2 FASTBUS STATUS REGISTER 1 (R)

This register is a read only register. In case of a Sequencer Primary Address cycle error (bit SEQ_PRIM_ERROR in the Sequencer Status register is set), this register gives more detailed information about the reason.

bit	name	function
31	reserved	“1“
..	..	..
16	reserved	“1“
15	reserved	“1“
14	reserved	“1“
13	reserved	“1“
12	reserved	“1“
11	reserved	“0“
10	PRIM_AK_TIMEOUT2	Primary Address Cycle (AK) Timeout and WT is actice --> SI (Segment Interconnect) Farside Timeout
9	PRIM_AK_TIMEOUT1	normal Primary Address cycle (AK) Timeout
8	reserved	“0“
7	PRIM_SS_STOP	indicates that the last Primary Address cycle stops with SS-response < > 0
6	PRIM_SS2	SS2 bit of last Primary Address cycle
5	PRIM_SS1	SS1 bit of last Primary Address cycle
4	PRIM_SS0	SS0 bit of last Primary Address cycle
3	PRIM_ERROR3	set AS Timeout; not possible to set AS (WT)
2	PRIM_ERROR2	pending Master but other Master still active (Longtime Timeout)
1	ARB_ERROR1	Arbitration Timeout
0	PRIM_ERROR1	try to set AS but already ASAK Lock is set

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2“ : 0xFFFFF000

4.3 FASTBUS STATUS REGISTER 2 (R)

This register is a read only register. In case of a Sequencer Data or DMA(blocktransfer) cycle error (bit SEQ_DATA_ERROR or bit SEQ_DMA_ERROR in the Sequencer Status register is set), this register gives more detailed information about the reason.

bit	name	function
31	reserved	“1“
.:	.:	.:
16	reserved	“1“
15	DMA_DONE	indicates that the DMA is finished
14	DMA_BUSY	indicates that the DMA is BUSY
13	DMA_VME_TIMEOUT	indicates a VME TIMEOUT during last DMA
12	DMA_FB_TIMEOUT	indicates a FASTBUS Timeout during last DMA
11	DMA_LIMIT	indicates that the last DMA stops by Limit Counter
10	DMA_SS2	shows the SS2 bit from last DMA
9	DMA_SS1	shows the SS1 bit from last DMA
8	DMA_SS0	shows the SS0 bit from last DMA
7	DATA_SS_STOP	indicates that the last Data Cycle stops with SS-response < > 0
6	DATA_SS2	SS2 bit of last Data Cycle
5	DATA_SS1	SS1 bit of last Data Cycle
4	DATA_SS0	SS0 bit of last Data Cycle
3	DATA_ERROR4	DK Timeout
2	DATA_ERROR3	set DS Timeout (not possible to set DS, WT)
1	DATA_ERROR2	DK is set !!!
0	DATA_ERROR1	no ASAK Lock

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2“ : 0xFFFF0000

4.4 FASTBUS LAST PRIMARY ADDRESS STORE REGISTER (R)

This register is a 32-bit read only register and it is updated with the Fastbus AD lines (primary address) by each Primary Address cycle of the SFI. This register is helpfull for error debuging by executing a FASTBUS command list to find out in an error case with which FASTBUS slave the error occured.

Value after Power-up Reset or Reset : 0x00000000

4.5 LAST SEQUENCER PROTOCOL (SEQUENCER KEYADDRESS) REGISTER (R)

This register is a read only register and it is updated with sequencer protocol command by each reading of the next sequencer protocol command from the VME2SEQ sequencer protocol Fifo. In case of a sequencer command error (bit SEQ_CMD_ERROR in the Sequencer Status register is set), this register stores the last protocol command.

bit	name	function
31	reserved	“1“
..	..	..
16	reserved	“1“
15	L_SEQ_A15	last sequencer key address bit A15
14	L_SEQ_A14	last sequencer key address bit A14
13	L_SEQ_A13	last sequencer key address bit A13
12	L_SEQ_A12	last sequencer key address bit A12
11	L_SEQ_A11	last sequencer key address bit A11
10	L_SEQ_A10	last sequencer key address bit A10
9	L_SEQ_A9	last sequencer key address bit A9
8	L_SEQ_A8	last sequencer key address bit A8
7	L_SEQ_A7	last sequencer key address bit A7
6	L_SEQ_A6	last sequencer key address bit A6
5	L_SEQ_A5	last sequencer key address bit A5
4	L_SEQ_A4	last sequencer key address bit A4
3	L_SEQ_A3	last sequencer key address bit A3
2	L_SEQ_A2	last sequencer key address bit A2
1	reserved	“1“
0	reserved	“1“

Value after Power-up Reset or Reset : 0xFFFF0003

4.6 NEXT SEQUENCER RAM ADDRESS REGISTER (R/W)

This register stores the Sequencer Ram address pointer. It is used either for loading a command list or for executing a command list.
In case of loading a command list this register will be loaded with the ram address via the VME CPU. Then the CPU has to enable the ram load mode. Now the CPU can write the command list into the VME2SEQ Fifo. The sequencer ram load logic reads the Fifo (if not empty) and writes the reading data (protocol and data) into the sequencer ram and increments the address pointer automatically. After the CPU has written the full command list into the Fifo (indirect to the sequencer ram) it is possible to read the address pointer to check or to store the end address.
In case of executing a command list this register will be loaded with the start ram address via the sequencer. The sequencer can only load addresses on a 0x100 boundary (0x0, 0x100, 0x200, ...). The sequencer increments also the address pointer automatically. After executing the command list it is possible to read the address pointer via the CPU to check the end address or in an error case it is possible to find out on which command the error occured.

It is **only possible** to load this register (write to this register) from the VME CPU with a new sequencer ram address pointer **if the sequencer is not enabled** and **if the sequencer is not in the sequencer ram load mode.**

It is **only allowed to read** the address pointer if the sequencer is **not in ram load mode** or if the sequencer is **not in the ram mode.**

bit	name	R/W	function
31	reserved	read only	“1“
..	..	..	..
16	reserved	read only	“1“
15	SEQ_RAM_A15	read/write	no function
14	SEQ_RAM_A14	read/write	sequencer ram address bit A14
13	SEQ_RAM_A13	read/write	sequencer ram address bit A13
12	SEQ_RAM_A12	read/write	sequencer ram address bit A12
11	SEQ_RAM_A11	read/write	sequencer ram address bit A11
10	SEQ_RAM_A10	read/write	sequencer ram address bit A10
9	SEQ_RAM_A9	read/write	sequencer ram address bit A9
8	SEQ_RAM_A8	read/write	sequencer ram address bit A8
7	SEQ_RAM_A7	read/write	sequencer ram address bit A7 (loaded from the seq. always with “0“)
6	SEQ_RAM_A6	read/write	sequencer ram address bit A6 (loaded from the seq. always with “0“)
5	SEQ_RAM_A5	read/write	sequencer ram address bit A5 (loaded from the seq. always with “0“)
4	SEQ_RAM_A4	read/write	sequencer ram address bit A4 (loaded from the seq. always with “0“)
3	SEQ_RAM_A3	read/write	sequencer ram address bit A3 (loaded from the seq. always with “0“)
2	SEQ_RAM_A2	read/write	sequencer ram address bit A2 (loaded from the seq. always with “0“)
1	SEQ_RAM_A1	read/write	sequencer ram address bit A1 (loaded from the seq. always with “0“)
0	SEQ_RAM_A0	read/write	sequencer ram address bit A0 (loaded from the seq. always with “0“)

Value after Power-up Reset or Reset : 0xFFFF0000

Sequencer ram size: 32K x 48 bit

4.7 FASTBUS TIMEOUT REGISTER (R/W)

The Short Timeout is enabled during each primary address and data cycles on FASTBUS if WT is not active.

The Long Timeout is only enabled during FASTBUS arbitration and if WT (wait on FASTBUS) is set during primary address and data cycles.

bit	name	R/W	function
31	reserved	read only	"1"
.:	.:	.:	.:
8	reserved	read only	"1"
7	TIME_LONG_DIS	read/write	Disable Long Timeout
6	TIME_LONG_2	read/write	Long Timeout Bit 2
5	TIME_LONG_1	read/write	Long Timeout Bit 1
4	TIME_LONG_0	read/write	Long Timeout Bit 0
3	TIME_SHORT_DIS	read/write	Disable Short Timeout
2	TIME_SHORT_2	read/write	Short Timeout Bit 2 (not used ; reserved)
1	TIME_SHORT_1	read/write	Short Timeout Bit 1
0	TIME_SHORT_0	read/write	Short Timeout Bit 0

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2" : 0xFFFFFF00

TIME_x2	TIME_x1	TIME_x0	Short Timeout	Long Timeout
0	0	0	1.6 us	51.2 us
0	0	1	3.2 us	205 us
0	1	0	6.4 us	1.6 ms
0	1	1	12.8 us	12.8 ms
1	0	0		102 ms
1	0	1		8182 ms
1	1	0		6.4 sec
1	1	1		24.8 sec

(valid for 40 MHZ Source Clock, 32 MHZ are used !!)

4.8 FASTBUS ARBITRATION LEVEL REGISTER (R/W)

bit	name	R/W	function
31	reserved	read only	"1"
.:	.:	.:	.:
8	reserved	read only	"1"
7	ABIT	read/write	Assured Access (FAIR)
6	reserved	read/write	reserved
5	AL5	read/write	FASTBUS Arbitration Level Bit 5
4	AL4	read/write	FASTBUS Arbitration Level Bit 4
3	AL3	read/write	FASTBUS Arbitration Level Bit 3
2	AL2	read/write	FASTBUS Arbitration Level Bit 2 (set after reset)
1	AL1	read/write	FASTBUS Arbitration Level Bit 1
0	AL0	read/write	FASTBUS Arbitration Level Bit 0

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2" : 0xFFFFFF04

4.9 FASTBUS PROTOCOL SIGNAL REGISTER (R)

This register is a read only register and gives status information about some FASTBUS signals.

bit	name	function
31	reserved	“1“
.:	.:	.:
16	reserved	“1“
15	FB_RB	shows the actual status of FASTBUS signal Reset Bus (RB)
14	FB_BH	shows the actual status of FASTBUS signal Bus Halt (BH)
13	FB_SR	shows the actual status of FASTBUS signal Service Request (SR)
12	ISMINE_L	shows a actual status of the Arbitation Logic
11	FB_GK	shows the actual status of FASTBUS signal Grant Acknowledge (GK)
10	FB_AG	shows the actual status of FASTBUS signal Arbitration Grant (AG)
9	FB_AI	shows the actual status of FASTBUS signal Arbitration Request Inhibit (AI)
8	FB_AR	shows the actual status of FASTBUS signal Arbitration Request (AR)
7	FB_EG	shows the actual status of FASTBUS signal Enable Geographical (EG)
6	reserved	“1“
5	FB_AK	shows the actual status of FASTBUS signal Address Acknowledge (AK)
4	FB_DK	shows the actual status of FASTBUS signal Data Acknowledge (DK)
3	FB_WT	shows the actual status of FASTBUS signal Wait (WT)
2	FB_SS2	shows the actual status of FASTBUS signal Slave Status Bit2 (SS2)
1	FB_SS1	shows the actual status of FASTBUS signal Slave Status Bit 1 (SS1)
0	FB_SS0	shows the actual status of FASTBUS signal Slave Status Bit 0 (SS0)

4.10 SEQUENCER FIFO FLAG AND ECL/NIM INPUT REGISTER (R)

This register is a read only register and gives status informations about the AUX_B42, NIM and ECL inputs and about the Sequencer Fifo Flags.

bit	name	function
31	reserved	“1“
..	..	..
16	reserved	“1“
15	IN_AUX_B42	AUXILIARY Connector B42 Input
14	IN_NIM3	Level Input NIM3
13	IN_NIM2	Level Input NIM2
12	IN_NIM1	Level Input NIM1
11	IN_ECL4	Level Input ECL4
10	IN_ECL3	Level Input ECL3
9	IN_ECL2	Level Input ECL2
8	IN_ECL1	Level Input ECL1
7	SEQ2VME_FF	Sequencer to VME Fifo **Full** Flag (synchronous)
6	SEQ2VME_HF	Sequencer to VME Fifo **Half Full** Flag (asynchronous)
5	SEQ2VME_AE	Sequencer to VME Fifo **Almost Empty** Flag (asynchronous)
4	SEQ2VME_EF	Sequencer to VME Fifo **Empty** Flag (synchronous)
3	VME2SEQ_FF	VME to Sequencer Fifo **Full** Flag (synchronous)
2	VME2SEQ_HF	VME to Sequencer Fifo **Half Full** Flag (asynchronous)
1	VME2SEQ_AE	VME to Sequencer Fifo **Almost Empty** Flag (asynchronous)
0	VME2SEQ_EF	VME to Sequencer Fifo **Empty** Flag (synchronous)

(default used Fifos IDT72225: 1K Fifo , Almost Empty is valid if words < 128)

4.11 OUT-SIGNAL REGISTER (W)

On the SFI are **two** Out-Signal registers implemented. The first register will be set/cleared by VME cycles and the second will be set/cleared by the Sequencer. The Output Level of the signals NIM_OUTx, ECL_Ox and Lx of both register are ored.
Each register is a selective set/clear register. A „0“ written into the corresponding bit has no effect. A „1“ written into the corresponding bit activates the specified function. A „1“ written into both of the set and clear bit is not allowed in the same time (generates toggle function).

4.11.1 VME Out-Signal register

bit	name	write a “1“ function
31	reserved	no
30	CLR_AUX_A45	clear Auxiliary Connector Output A45
29	CLR_AUX_A28	clear Auxiliary Connector Output A28
28	CLR_AUX_A10	clear Auxiliary Connector Output A10
27	reserved	no
26	CLR_NIM_OUT 3	clear NIM Output O3
25	CLR_NIM_OUT 2	clear NIM Output O2
24	CLR_NIM_OUT 1	clear NIM Output O1
23	CLR_ECL_O1	clear diff. ECL Output O1
22	CLR_ECL_O2	clear diff. ECL Output O2
21	CLR_ECL_O3	clear diff. ECL Output O3
20	CLR_ECL_O4	clear diff. ECL Output O4
19	CLR_L4	clear Led L4 and Testpin output T4 (TTL, low active)
18	CLR_L3	clear Led L3 and Testpin output T3 (TTL, low active)
17	CLR_L2	clear Led L2 and Testpin output T2 (TTL, low active)
16	CLR_L1	clear Led L1 and Testpin output T1 (TTL, low active)
15	reserved	no
14	SET_AUX_A45	set Auxiliary Connector Output A45
13	SET_AUX_A28	set Auxiliary Connector Output A28
12	SET_AUX_A10	set Auxiliary Connector Output A10
11	reserved	no
10	SET_NIM_OUT 3	set NIM Output O3
9	SET_NIM_OUT 2	set NIM Output O2
8	SET_NIM_OUT 1	set NIM Output O1
7	SET_ECL_O4	set diff. ECL Output O4
6	SET_ECL_O3	set diff. ECL Output O3
5	SET_ECL_O2	set diff. ECL Output O2
4	SET_ECL_O1	set diff. ECL Output O1
3	SET_L4	set Led L4 and Testpin output T4 (TTL, low active)
2	SET_L3	set Led L3 and Testpin output T3 (TTL, low active)
1	SET_L2	set Led L2 and Testpin output T2 (TTL, low active)
0	SET_L1	set Led L1 and Testpin output T1 (TTL, low active)

4.11.2 Sequencer Out-Signal register (writeable from the sequencer !)

bit	name	write a “1“ function
31	reserved	no
30	reserved	no
29	reserved	no
28	reserved	no
27	reserved	no
26	CLR_NIM_OUT 3	clear NIM Output O3
25	CLR_NIM_OUT 2	clear NIM Output O2
24	CLR_NIM_OUT 1	clear NIM Output O1
23	CLR_ECL_O1	clear diff. ECL Output O1
22	CLR_ECL_O2	clear diff. ECL Output O2
21	CLR_ECL_O3	clear diff. ECL Output O3
20	CLR_ECL_O4	clear diff. ECL Output O4
19	CLR_L4	clear Led L4 and Testpin output T4 (TTL, low active)
18	CLR_L3	clear Led L3 and Testpin output T3 (TTL, low active)
17	CLR_L2	clear Led L2 and Testpin output T2 (TTL, low active)
16	CLR_L1	clear Led L1 and Testpin output T1 (TTL, low active)
15	reserved	no
14	reserved	no
13	reserved	no
12	reserved	no
11	reserved	no
10	SET_NIM_OUT 3	set NIM Output O3
9	SET_NIM_OUT 2	set NIM Output O2
8	SET_NIM_OUT 1	set NIM Output O1
7	SET_ECL_O1	set diff. ECL Output O1
6	SET_ECL_O2	set diff. ECL Output O2
5	SET_ECL_O3	set diff. ECL Output O3
4	SET_ECL_O4	set diff. ECL Output O4
3	SET_L4	set Led L4 and Testpin output T4 (TTL, low active)
2	SET_L3	set Led L3 and Testpin output T3 (TTL, low active)
1	SET_L2	set Led L2 and Testpin output T2 (TTL, low active)
0	SET_L1	set Led L1 and Testpin output T1 (TTL, low active)

4.12 VME IRQ Source and MASK register

Eight interrupt sources are implemented. Each of these interrupt sources can be individually enabled/disabled (cleared) via this J/K register. A J/K register is a selective set/clear register. A “0“ written into the corresponding bit has no effect. A “1“ written into the corresponding bit activates the specified function. A “1“ written into both of the enable-bit and disable (clear)-bit is **not allowed** in the same time (generates a bit toggle function).

bit	read function	write a “1“ function
31	“1“	no
..	..	..
16	“1“	no
15	SEQ_DISABLE_IRQ_FLAG	disable Sequencer DISABLE source
14	SEQ_CMD_FLAG	disable Sequencer CMD Flag source
13	SR_IRQ_FLAG	disable SR source (FASTBUS Service Request)
12	AUX_B42_IRQ_FLAG	disable/clear AUX_B42 source
11	ECL1_IRQ_FLAG	disable/clear ECL1 IRQ source
10	NIM3_IRQ_FLAG	disable/clear NIM3 IRQ source
9	NIM2_IRQ_FLAG	disable/clear NIM2 IRQ source
8	NIM1_IRQ_FLAG	disable/clear NIM1 IRQ source
7	SEQ_DISABLE_ENABLE	enable Sequencer DISABLE source
6	SEQ_CMD_FLAG_ENABLE	enable Sequencer CMD Flag source
5	SR_ENABLE	enable SR source (FASTBUS Service Request)
4	AUX_B42_ENABLED	enable AUX_B42 source
3	ECL1_IRQ_ENABLED	enable ECL1 IRQ source
2	NIM3_IRQ_ENABLED	enable NIM3 IRQ source
1	NIM2_IRQ_ENABLED	enable NIM2 IRQ source
0	NIM1_IRQ_ENABLED	enable NIM1 IRQ source

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2“ : 0xFFFF0000

4.13 VME IRQ LEVEL AND VECTOR REGISTER

This read/write register controls the VME interrupt control logic on the SFI. It is possible to enable/disable the generation of a VME interrupt if an internal VME_IRQ is set, to determine the VME IRQ Level (IRQ[7..1]) and to determine the 8-bit Status/ID (Interrupt Vector). The type of the Interrupter is **D08(O).**

bit	name	R/W	function
31	reserved	read only	“1“
.:	.:	.:	.:
16	reserved	read only	“1“
15	VME_IRQ	read only	“1“: VME IRQ is set (internal VME IRQ is set and VME IRQ is enabled)
14	I_VME_IRQ	read only	“1“: internal VME IRQ is set
13	reserved	read only	“0“
12	reserved	read only	“0“
11	VME_IRQ_ENABLE	read/write	“0“: VME IRQ is disabled ; “1“: VME IRQ is enabled ;
10	VME_IRQ_LEV2	read/write	determines the VME IRQ Level on VME
9	VME_IRQ_LEV1	read/write	determines the VME IRQ Level on VME
8	VME_IRQ_LEV0	read/write	determines the VME IRQ Level on VME
7	Status/ID_bit7	read/write	Status/ID bit 7; placed on D7 during VME IRQ acknowledge cycle
6	Status/ID_bit6	read/write	Status/ID bit 6; placed on D6 during VME IRQ acknowledge cycle
5	Status/ID_bit5	read/write	Status/ID bit 5; placed on D5 during VME IRQ acknowledge cycle
4	Status/ID_bit4	read/write	Status/ID bit 4; placed on D4 during VME IRQ acknowledge cycle
3	Status/ID_bit0	read/write	Status/ID bit 3; placed on D3 during VME IRQ acknowledge cycle
2	Status/ID_bit0	read/write	Status/ID bit 2; placed on D2 during VME IRQ acknowledge cycle
1	Status/ID_bit0	read/write	Status/ID bit 1; placed on D1 during VME IRQ acknowledge cycle
0	Status/ID_bit0	read/write	Status/ID bit 0; placed on D0 during VME IRQ acknowledge cycle

Example for VME IRQ Level 5 (IRQ5): VME_IRQ_LEV2 = 1
VME_IRQ_LEV1 = 0
VME_IRQ_LEV0 = 1

Value after Power-up Reset, Reset or Key Address „Reset register group LCA2“ : 0xFFFF0000

6 FASTBUS MASTER SEQUENCER

This chapter is intended to give a short introduction in the VME - Fastbus master port interface technique of the SFI.

As described in much more detail in the following chapters, Fastbus cycles can be executed as VME A24 /D32 cycles which are decoded by a Sequencer FIFO logic to the VME Fastbus operations. The lower 16 address bits of the VME (A24), called Sequencer FIFO key addresses, specify the Fastbus Protocol (and special functions) whilst the VME D32 data specify the FASTBUS data during a Fastbus write operation.

With the Sequencer FIFO key addresses it is also possible to execute the following special functions:

* start autonomous blocktransfer (High speed Fastbus blocktransfer, DK(t) to DS(t) = 35 to 45ns)
* load DMA limitcounter (max. blocklength)
* load DMA address pointer (VME Slave address, where the data has to transfered)
* store DMA wordcounter and SS-response in Sequencer Data Output Fifo to build Event directory
* set/clear external signals (NIM, ECL, LEDs)

6.1 SEQUENCER ENABLE (ARM)

After Power-On reset the sequencer is disabled and all the Fifos are empty. It is possible to enable the sequencer with the VME key address **Sequencer Enable.** The enabled sequencer checks continuously the empty condition of the VME2SEQ Fifo. If the sequencer logic detects that the Fifo is not empty, then the sequencer logic reads the Sequencer Protocol and the Data Input Fifos (VME2SEQ Fifo) and tries to execute the sequencer command, which was written from the VME CPU to the Fifo. In case of an error (exeption) the sequencer will automatically disable and stop and the used FASTBUS lines will be cleared in the right order.

Four error condition flags are defined and readable via the Sequencer Status register:

* SEQ_CMD_ERROR: This error flag indicates that an invalid sequencer command was written to the VME2SEQ Fifo (Protocol Fifo). For debugging it is possible to read back the last sequencer command via the **Last Sequencer Protocol register.**
* SEQ_PRIM_ERROR: This error flag indicates that an exeption has occurred during executing a FASTBUS primary address cycle. For debugging it is now possible to read the **FASTBUS Status register 1** to get more information about the reason.
* SEQ_DATA_ERROR: This error flag indicates that an exeption has occurred during executing a FASTBUS data cycle (secondary address, random data). For debugging it is possible to read the **FASTBUS Status register 2** to get more information about the reason.
* SEQ_DMA_ERROR : This error flag indicates that an exception has occurred during executing a FASTBUS DMA blocktransfer cycle. For debugging it is possible to read the **FASTBUS Status register 2** to get more information about the reason.

In case of an error the sequencer has to be reseted to clear the error flags and to clear the Fifos. Then the sequencer can be enabled again.

5 VME INTERRUPTS

Eight interrupt sources are implemented to generate a VME interrupt. Each interrupt source can be individually enabled/disabled . The VMEbus interrupt Level (IRQ7 to 1) and the VME Interrupt Vector (Status/Id) is specified in the **VME IRQ Level and Vector register**.

The interrupt sources are enabled/disabled and cleared by writing to the **VME IRQ Source and Mask register**.

The following interrupt sources are implemented:

* Input NIM1: If the NIM1_IRQ is enabled then the NIM1_IRQ_FLAG is set with the leading edge* of the NIM1 input. This flag is cleared with the corresponding disable/clear write cycle.
* Input NIM2: If the NIM2_IRQ is enabled then the NIM2_IRQ_FLAG is set with the leading edge* of the NIM2 input. This flag is cleared with the corresponding disable/clear write cycle.
* Input NIM3: If the NIM3_IRQ is enabled then the NIM3_IRQ_FLAG is set with the leading edge* of the NIM3 input. This flag is cleared with the corresponding disable/clear write cycle.
* Input ECL1: If the ECL1_IRQ is enabled then the ECL1_IRQ_FLAG is set with the leading edge* of the ECL1 input. This flag is cleared with the corresponding disable/clear write cycle.
* Input AUX_B42: If the AUX_B42_IRQ is enabled then the AUX_B42_IRQ_FLAG is set with the high to low transition (negative TTL edge) of the AUX_B42 TTL input. This flag is cleared with the corresponding disable/clear write cycle.
* Service Request: If the SR_IRQ (FASTBUS Service Request) is enabled then a SR on FASTBUS generates an internal IRQ request. In the interrupt service routine has the generation of SR on FASTBUS to be cleared or the Interrupt has to be disabled.
* SEQ_CMD_FLAG: If the SEQ_CMD_FLAG_IRQ is enabled and the SEQ_CMD_FLAG is set from the sequencer then an internal interrupt is generated. The SEQ_CMD_FLAG can be cleared with the VME key address **Clear Sequencer CMD Flag**.
* SEQ_DISABLE: If the SEQ_DISABLE_IRQ is enabled and the sequencer is disabled then an internal interrupt is generated.

(*NIM leading Edge: transition from NIM inactive (0V) to active (-0.7V))

Internal Interrupt	if	NIM1_IRQ_FLAG		
	or	NIM2_IRQ_FLAG		
	or	NIM3_IRQ_FLAG		
	or	ECL1_IRQ_FLAG		
	or	AUX_B42_IRQ_FLAG		
	or	SR_IRQ_FLAG		
	or	SEQ_CMD_FLAG	and	SEQ_DISABLE_IRQ_FLAG
	or	SEQ_DISABLE_IRQ_FLAG		

A VME

6.2 SEQUENCER FIFO KEY ADDRESSES

The sequencer commands are written to the VME2SEQ Fifo. The VME2SEQ Fifo consists of a Sequencer Protocol Fifo and a Sequencer Data Input Fifo. The Sequencer Data Input Fifo stores the data (32-bit) and the Protocol Fifo stores the lower part (A15 to A2) of the VME addresses (A32) during the VME write cycle from the CPU to the VME2SEQ Fifo. The sequencer key address (A15 to A2), stored in the Protocol Fifo, detemines the action of the sequencer.

General definition of the sequencer key address or sequencer command:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
x	x	x	x	x	x	x	x	F3	F2	F1	F0	SEQ_CTR	FB_EN	0	0

SEQ_CTR:	Sequencer Control action Enable Bit
FB_EN:	FASTBUS action Enable Bit
F3, F2, F1, F0:	Function paramter
x:	unction depending

6.2.1 FASTBUS actions

FASTBUS actions are enabled with SEQ_CTR = 0 and FB_EN = 1

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
x	x	x	EG	RD	MS2	MS1	MS0	F3	F2	F1	F0	0	1	0	0

Note that the generation of EG is not vital and even not allowed when trying to pass through Segement interconnects. For test purposes, however, the generation of EG helps to overcome problems due to defective GAC controllers or modules which do not properly take into account the delays of EG when generated by GAC controllers.

F3	F2	F1	F0	Function	Data words defines
0	0	0	0	Primary address cycle (include Arbitration)	FB Primary Address
0	0	0	1	Primary address cycle (include Arbitration and holds mastership)	FB Primary Address
0	0	1	0	Device Release (Disconnect AS-AK lock)	no
0	0	1	1	Device Release (Disconnect AS-AK lock and releases mastership)	no
0	1	0	0	transfer data cycle	FB Write Data
0	1	0	1	transfer data cycle followed by a Device Release (Disconnect)	FB Write Data
0	1	1	0	cleanup data cycle (not impl)	no
0	1	1	1	nop (reserved)	
1	0	0	0	nop (reserved)	
1	0	0	1	load blocktransfer read/write VME address pointer	VME Slave Addr.
1	0	1	0	load limit counter,clear word counter, start autonomous blocktransfer	Mode/Limit counter
1	0	1	1	load limit counter and start autonomous blocktransfer	Mode/Limit counter
1	1	0	0	nop (reserved)	
1	1	0	1	store address pointer in Sequencer Data Input FiFo (SEQ2VME Fifo)	no
1	1	1	0	store DMA status and word counter in Sequencer Data Input FiFo	no
1	1	1	1	store only word counter Sequencer Data Input Fifo (SEQ2VME Fifo)	no

6.2.1.1 Mode register and Limit counter (Sequencer Load)

The sequencer logic loads the Mode register and the Limit counter with the data which are read from the Data Input Fifo (VME2SEQ) during executing the command “load limit counter,clear word counter, start autonomous blocktransfer“ before it starts the blocktransfer.

bit	name	function
31	reserved	no
30	reserved	no
29	reserved	no
28	DIRECT_MODE	enables direct Mode
27	VME_MODE	enables VME Mode
26	B64_MODE	enable VME xBLT64 Blocktransfer (Option, on request)
25	B32_MODE	enable VME 32 bit Blocktransfer
24	D32_MODE	enable VME 32 bit Data cycles
23	LIMIT_CNT23	Limit Counter bit 23 (MSB)
..		
..		
0	LIMIT_CNT0	Limit Counter bit 0 (LSB)

The Limit counter controls the maximal length of the FASTBUS blocktransfer if no other stop condition (SS-response or Timeout) is detected.
The blocktransfer stops after <Limit counter > plus one word.

6.2.1.2 DMA Status register and Word Counter (Sequencer Store)

The sequencer logic stores the DMA status (blocktransfer) and word counter in the Sequencer Data Output Fifo (SEQ2VME). The VME CPU has to read SEQ2VME Fifo to get this word.
Note: The Fifo is only readable, if the Fifo Flag EMPTY is not set. If the Fifo is not empty but the Fifo Flag EMPTY is set, then a read cycle actualize **only** the EMPTY Flag and do not read the data from the Fifo.

bit	name	function
31	reserved	“0“
30	reserved	“0“
29	DMA_VME_TIMEOUT	indicates a VME TIMEOUT during last DMA
28	DMA_FB_TIMEOUT	indicates a FASTBUS Timeout during last DMA
27	DMA_LIMIT	indicates that the last DMA stops by Limit Counter
26	DMA_SS2	shows the SS2 bit from last DMA
25	DMA_SS1	shows the SS1 bit from last DMA
24	DMA_SS0	shows the SS0 bit from last DMA
23	WORD_CNT23	Word counter bit 23 (MSB)
..		
..		
0	WORD_CNT0	Word counter bit 0 (LSB)

6.2.1.3 FASTBUS command key address assignments

A short overview list for better understanding the FASTBUS action command key addresses is given only (refer to file SFI.H for more details or for use).

```
/* constants for primary address sequencer key addresses */
#define PRIM_DSR           (0x0004)    /* Data space logical or geogr.  */
#define PRIM_CSR           (0x0104)    /* CSR space logical or geogr.  */
#define PRIM_DSRM          (0x0204)    /* Data space broadcast          */
#define PRIM_CSRM          (0x0304)    /* CSR space broadcast           */
#define PRIM_AMS4          (0x0404)    /* MS4 primary address          */
#define PRIM_AMS5          (0x0504)    /* MS5 primary address          */
#define PRIM_AMS6          (0x0604)    /* MS6 primary address          */
#define PRIM_AMS7          (0x0704)    /* MS7 primary address          */

/* constants for primary address sequencer key addresses */
#define PRIM_HM_DSR        (0x0014)    /* Data space logical or geogr. and hold Mastership */
#define PRIM_HM_CSR        (0x0114)    /* CSR space logical or geogr. and hold Mastership */
#define PRIM_HM_DSRM       (0x0214)    /* Data space broadcast and hold Mastership */
#define PRIM_HM_CSRM       (0x0314)    /* CSR space broadcast and hold Mastership */

#define PRIM_EG            (0x1000)    /* to add for setting EG during primary addr. cycle */

#define RNDM_R             (0x0844)    /* random data read */
#define RNDM_W             (0x0044)    /* random data write */
#define SECAD_R            (0x0A44)    /* secondary address read */
#define SECAD_W            (0x0244)    /* secondary address write */

#define RNDM_R_DIS         (0x0854)    /* random data read and followed disconnect from FB */
#define RNDM_W_DIS         (0x0054)    /* random data write and followed disconnect from FB */

#define DISCON             (0x0024)    /* disconnect from FASTBUS; release AS-AK lock */
#define DISCON_RM          (0x0034)    /* disconnect from FB; release AS-AK lock  and release
                                          Mastership (if hold) */

#define STORE_FRDB_WC      (0x00E4)    /* store wordcounter into SEQ2VME Fifo */
#define STORE_FRDB_AP      (0x00D4)    /* store „next“ DMA address pointer */

#define START_FRDB_WITH_CLEAR_WORD_COUNTER       (0x08A4)        /* start DMA FRDB */
#define LOAD_DMA_ADDRESS_POINTER                 (0x0094)        /* load „next“ DMA
                                                                 address pointer; VME
                                                                  Slave address */

/* intialize global variables for fast FASTBUS access */
   fastPrimDsr     = GetFastbusPtr(PRIM_DSR);
   fastPrimCsr     = GetFastbusPtr(PRIM_CSR);
   fastPrimDsrM    = GetFastbusPtr(PRIM_DSRM);
   fastPrimCsrM    = GetFastbusPtr(PRIM_CSRM);
   fastPrimHmDsr   = GetFastbusPtr(PRIM_HM_DSR);
   fastPrimHmCsr   = GetFastbusPtr(PRIM_HM_CSR);
   fastPrimHmDsrM  = GetFastbusPtr(PRIM_HM_DSRM);
   fastPrimHmCsrM  = GetFastbusPtr(PRIM_HM_CSRM);
   fastSecadR      = GetFastbusPtr(SECAD_R);
   fastSecadW      = GetFastbusPtr(SECAD_W);
   fastRndmR       = GetFastbusPtr(RNDM_R);
   fastRndmW       = GetFastbusPtr(RNDM_W);
   fastRndmRDis    = GetFastbusPtr(RNDM_R_DIS);
   fastRndmWDis    = GetFastbusPtr(RNDM_W_DIS);
   fastDiscon      = GetFastbusPtr(DISCON);
   fastDisconRm    = GetFastbusPtr(DISCON_RM);
   fastStartFrdbWithClearWc = GetFastbusPtr(START_FRDB_WITH_CLEAR_WORD_COUNTER);
   fastStoreFrdbWc = GetFastbusPtr(STORE_FRDB_WC);
   fastStoreFrdbAp = GetFastbusPtr(STORE_FRDB_AP);
```

6.2.1.4 FASTBUS cycle command list examples

Sequencer command list for one FWC FASTBUS cycle:

```
*fastPrimCsr      =  PAddr;   /* CSR primary address cycle    */
*fastSecadW      =  SAddr;   /* secondary address write cycle */
*fastRndmWDis =  Wdata;  /* random data write cycle           */
```

Sequencer command list for one FRD FASTBUS cycle:

```
*fastPrimDsr      =  PAddr;   /* Data space primary address cycle    */
*fastSecadW      =  SAddr;   /* secondary address write cycle          */
*fastRndmRDis  =  dummy;  /* random data read cycle                     */
```

Sequencer command list for one FWCM FASTBUS cycle and hold Mastership:

```
*fastPrimHmCsrM    =   PAddr;    /* CSR primary address broadcast cycle and hold Mastership
*/
*fastSecadW          =  SAddr;   /* secondary address write cycle */
*fastRndmWDis     =  Wdata;  /* random data write cycle           */
```

Sequencer command list for one FRDB FASTBUS cycle:

```
*fastLoadDmaAddressPointer =  Buffer;                /* VME Slave Memory address */
*fastPrimDsr                          = PAddr;                /* primary address cycle            */
*fastSecadW                          = Saddr;                /* secondary address cycle       */
*fastStartFrdbWithClearWc    = Max_ExpLWord;    /* start dma, load mode and limit counter   */
*fastDiscon                           = reg32;                /*  disconnect   */
*fastStoreFrdbWc                  = reg32;                /* get wordcount; store wordcount command
*/
```

6.2.2 Sequencer Control actions

Sequencer Control actions are enabled with SEQ_CTR = 1 and FB_EN = 0

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
x	RA14	RA13	RA12	RA11	RA10	RA9	RA8	F3	F2	F1	F0	1	0	0	0

RA14:	Sequencer Ram address bit 14
..	
RA8:	Sequencer Ram address bit 8

The Sequencer Ram address bits RA7 to RA0 are loaded with "0" from the sequencer logic. Therefore the list hast to start on 0x100 address boundary.

F3	F2	F1	F0	Function	Data words defines
0	0	0	0	set and clear Sequencer Out Signal register (Leds, ECL, NIM)	set/clr (J/K) data
0	0	0	1	disable sequencer	no
0	0	1	0	enable ram sequencer mode (start list at seq. ram address <RA>)	no
0	0	1	1	disable ram sequencer mode (release ram sequencer mode and change to Fifo sequencer mode)	no
0	1	1	0	set sequencer SEQ_CMD_FLAG (generates VME IRQ)	no
0	1	0	x	no operation (reserved)	
0	1	1	1	no operation (reserved)	

The SEQ_CMD_FLAG is settable from the sequencer and readable from the VME CPU via the VME IRQ Source and MASK register bit 14. The VME CPU can clear this flag with VME Key Address CLR_SEQ_CMD_FLAG. A VME Interrupt is generated if the corresponding interrupt enable bit is set and the SEQ_CMD_FLAG is set.

6.3 SEQUENCER RAM

A sequencer ram is implemented to store and execute with **high speed** often used lists of FASTBUS cycles. The sequencer ram can store 32K (32768) key address commands. The beginning of a list has to store on a 0x100 (hex. $100) address boundary of the sequencer ram. So, it is possible to store up to 128 ($80) different lists. But it is also possible to store a list, which is longer than 0x100. The maximal length of a list is only limited by the absolute end of the sequencer ram.
The command list (set of sequencer key addresses) has to load in the sequencer ram at a user specified sequencer ram address. A stored list can be started with a single sequencer command if the sequencer is enabled.

6.3.1 Sequencer RAM Loading

Flow of Loading the sequencer ram:

1. **Reset** the sequencer to be sure that the VME2SEQ Fifo is empty and the sequencer is diabled.

2. **Load** the sequencer ram address via the Next Sequencer Ram Address register with a user defined ram address on a 0x100 address boundary (between 0x0000 and 0x7F00).

3. **Enable** the **RAM Load Enable Mode** via the VME Key Address **Sequencer Ram Load Enable**.
   This key address also disables the sequencer if the sequencer not yet disabled (did no reset).
   Now the sequencer is in the **RAM Load Mode** and it checks continuously the empty condition of the VME2SEQ Fifo. The VME CPU writes the list of commands in the manner as in the "FiFo Mode" into the VME2SEQ Fifo. If the sequencer logic detects that the Fifo is not empty, then the sequencer logic reads the Sequencer Protocol and the Data Input Fifos (VME2SEQ Fifo) and writes these data into the sequencer ram. Following, the sequencer logic increments the sequencer ram address.

4. **Add and write** an extra sequencer command (last command) to the VME2SEQ FiFo. This last command has to be one of the following sequencer action commands:
   * **disable sequencer** to disable the sequencer after execution of the list
   * **disable ram sequencer mode** to change to normal FiFo mode after execution of the list
   * or **enable ram sequencer** again to start an other ram list after execution of the list

5. **disable** after a little delay (500ns) the RAM Load Mode via the VME Key Address **Sequencer Ram Load Disable.**
   Read the **Next Sequencer Ram Address register** and check the new address pointer:
   <new address> = <old address> + <number of commands>

6. **enable** sequencer again (or call init routine).

6.3.2 Sequencer RAM List execution

To start a stored a command list it is only necessary to write the sequencer action command **enable ram sequencer mode** with the sequencer ram start address of the list to the VME2SEQ FiFo.

6.4 SEQUENCER PROGRAMMING EXAMPLES

6.4.1 FASTBUS initialization routine

```
void InitFastbus(arbReg, timReg)
unsigned long arbReg, timReg;
{
  unsigned long val32out;  /* dummy /*

  val32out = 0L;
  *sfi.sequencerReset = val32out;                    /* reset sequencer */
  *sfi.FastbusArbitrationLevelReg = arbReg;         /* set arbitration level */
  *sfi.FastbusTimeoutReg = timReg;                   /* set timeout register  */
  *sfi.sequencerEnable=val32out;                     /* enable sequencer */

} /* InitFastbus */
```

6.4.2 Single FWC routine

```
unsigned long FWC(PAddr, SAddr,WData)

unsigned long PAddr;
unsigned long SAddr;
unsigned long WData;
{
register unsigned long reg32;
register Return = 0;
register Exit   = 0;

*fastPrimCsr  = PAddr;               /* CSR primary address cycle      */
*fastSecadW   = SAddr;               /* secondary address write cycle  */
*fastRndmWDis = Wdata;               /* data write cycle                */

/* wait for sequencer done */
do
  {
  reg32 = *sfi.sequencerStatusReg;
  switch(reg32 & 0x00008001)
    {
    case 0x00008001:            /* OK */
      Return = 0;
      Exit = 1;
      break;
    case 0x00000001:           /* Not Finished */
      break;
    case 0x00000000:           /* Not Initialized */
      Return = *sfi.sequencerStatusReg & 0x0000ffff;
      Return |= 0x00020000;
      Exit = 1;
      break;
    case 0x00008000:           /* Bad Status is set, we will see */
      Return = *sfi.sequencerStatusReg & 0x0000ffff;
      Return |= 0x00010000;
      Exit = 1;
      break;
    }
  } while(!Exit);
return(Return);
} /* end FWC */
```

6.4.3 Single FRD routine

```
unsigned long FRD(PAddr, SAddr,rData)

unsigned long PAddr;
unsigned long SAddr;
unsigned long *rData;
{
register unsigned long reg32;
register Return = 0;
register Exit   = 0;

*fastPrimDsr  = PAddr; /* primary address cycle   */
*fastSecadW   = SAddr; /* secondary address cycle */
*fastRndmRDis = reg32;

/* wait for sequencer done */
do
  {
  reg32 = *sfi.sequencerStatusReg;

  switch(reg32 & 0x00008001)
    {
    case 0x00008001:            /* OK */
      Return = 0;
      Exit = 1;
     /* read seq fifo flags */
     /* check if EMPTY ; if EMPTY then Dummy read and Loop until not Empty */
     /*                 else if not empty then read fifo */
     reg32 = *sfi.readSeqFifoFlags;
     if((reg32 & 0x00000010) == 0x00000010)
     {
       reg32 = *sfi.readSeq2VmeFifoBase; /* dummy read */
       reg32 = *sfi.readSeqFifoFlags;
       if((reg32 & 0x00000010) == 0x00000010)
         {     /* Error  !!! */
          Return = *sfi.sequencerStatusReg & 0x0000ffff;
          Return |= 0x00100000;
         }
     }
     *rData = *sfi.readSeq2VmeFifoBase;      /* read fifo */
      break;

    case 0x00000001:                /* Not Finished */
      break;
    case 0x00000000:                /* Not Initialized */
      Return = *sfi.sequencerStatusReg & 0x0000ffff;
      Return |= 0x00020000;
      Exit = 1;
      break;
    case 0x00008000:                /* Bad Status is set, we will see */
      Return = *sfi.sequencerStatusReg & 0x0000ffff;
      Return |= 0x00010000;
      Exit = 1;
      break;
    }
  } while(!Exit);
return(Return);
} /* End of FRD */
```

6.4.4 Single FRDB routine

```
unsigned long FRDB(PAddr,SAddr,Buffer,next_buffer,Max_ExpLWord,cnt_RecLWord,Mode)

unsigned long PAddr;              /* Primary Address */
unsigned long SAddr;              /* Secondary Address */
unsigned long Buffer;             /* Address of LWord-Buffer (VME-Slave Address!!! )*/
unsigned long *next_buffer;       /* Address of LWord-Buffer after READ */
unsigned long Max_ExpLWord;       /* Max. Count of 32Bit Datawords  */
unsigned long *cnt_RecLWord;      /* Count of Received 32Bit Datawords  */
unsigned long Mode;               /* Readout-Modus (Direct and/or VME) */
                                  /* 0x10 only Direct           */
                                  /* 0x09 VMED32 Data cycle      */
                                  /* 0x0A VMED32 Blocktransfer   */
                                  /* 0x19 Direct and VME32Datacycl. */
                                  /* 0x1A Direct and VME32Blocktr.  */
{
register unsigned long reg32;
register Return = 0;  register Exit   = 0;

Max_ExpLWord &= 0x00ffffff;
Max_ExpLWord |= (Mode << 24);

*fastLoadDmaAddressPointer        = Buffer;               /* load VME Slave buffer address */
*fastPrimDsr                      = Paddr;                /* primary address cycle   */
*fastSecadW                       = SAddr;                /* secondary address cycle */
*fastStartFrdbWithClearWc         = Max_ExpLWord;         /* start dma    */
*fastDiscon                       = reg32;                /* disconnect   */
*fastStoreFrdbWc                  = reg32;                /* get wordcount*/
*fastStoreFrdbAp                  = reg32;                /* get adr.ptr. */

/* wait for sequencer done */
do
  {
  reg32 = *sfi.sequencerStatusReg;
  switch(reg32 & 0x00008001)
    {
    case 0x00008001:        /* OK */
      Return = 0;
      Exit = 1;
     /* read seq fifo flags */
     /* check if EMPTY ; if EMPTY then Dummy read and Loop until not Empty */
     /*                 else if not empty then read fifo */
     reg32 = *sfi.readSeqFifoFlags;
     if((reg32 & 0x00000010) == 0x00000010)
     {
       reg32 = *sfi.readSeq2VmeFifoBase; /* dummy read */
       reg32 = *sfi.readSeqFifoFlags;
       if((reg32 & 0x00000010) == 0x00000010)
          {    /* Error !!! */
          Return = *sfi.sequencerStatusReg & 0x0000ffff;
          Return |= 0x00100000;
          }
      }
      /* read fifo */
      *cnt_RecLWord = *sfi.readSeq2VmeFifoBase;
      /* Check for FIFO Empty: */
      reg32 = *sfi.readSeqFifoFlags;
      if((reg32 & 0x00000010) == 0x00000010)
         {   /* Error !!! */
         Return = *sfi.sequencerStatusReg & 0x0000ffff;
         Return |= 0x00040000;
         }
      *next_buffer  = *sfi.readSeq2VmeFifoBase;
```

```
         break;

      case 0x00000001:       /* Not Finished */
         break;
      case 0x00000000:        /* Not Initialized */
         Return = *sfi.sequencerStatusReg & 0x0000ffff;
         Return |= 0x00020000;
         Exit = 1;
         break;
      case 0x00008000:        /* Bad Status is set, we will see */
         Return = *sfi.sequencerStatusReg & 0x0000ffff;
         Return |= 0x00010000;
         Exit = 1;
         break;
      }
   } while(!Exit);
return(Return);
} /* end FRDB */
```

6.4.5 Load Next EVENT and FRDB List execution example

```
unsigned long LNE_FRDB(prim1,Buffer,cnt_RecLWord)

unsigned long prim1;           /* Primary Address */
unsigned long Buffer;          /* Address of LWord-Buffer (VME-Slave Address!!! )*/
unsigned long *cnt_RecLWord;   /* Count of Received 32Bit Datawords  */

{
register unsigned long reg32;
register Return = 0;
register Exit   = 0;

/* FWCM */
*fastPrimCsrM = 0x15L;  /* primary address cycle   */
*fastSecadW   = CSR0 ;  /* secondary address cycle */
*fastRndmWDis = LNE  ;

/* FRDB */
*fastLoadDmaAddressPointer = Buffer;
*fastPrimDsr                = prim1;             /* primary address cycle   */
*fastSecadW                 = 0x0L;              /* secondary address cycle */
*fastStartFrdbWithClearWc  = 0x0A007FFFL ; /* start dma, max 8000 words; VME Block    */
*fastDiscon                 = reg32;             /* disconnect   */
*fastStoreFrdbWc            = reg32;             /* get wordcount*/

/*  generate NIM puls (set and clear ) to clear external request */
*fastWriteOutReg          = 0x100L;              /* set NIM1 */
*fastWriteOutReg          = 0x1000000L;       /* clr NIM1 */


/* wait for sequencer done */
do
   {
      same as in FRDB example without read new address pointer
   }
return(Return);
}
```

6.4.6 Load a Command List in the Sequencer Ram example

```
unsigned long LoadRamLNE_FRDB(rStart, prim_start)

unsigned long rStart;          /* Sequencer Ram Startaddress */
unsigned long prim_start;    /* Primary Address */
{
register unsigned long reg32 = 0;
register Return = 0;
unsigned long iloop ;

*sfi.sequencerReset = reg32 ; /* reset sequencer */

/* load sequencer RAM address; befor enable LoadRam Mode !! */
*sfi.sequencerRamAddressReg = rStart;

/* enable Sequencer load ram mode */
*sfi.sequencerRamLoadEnable = reg32 ;
  /* load list */
  /* FWCM */
  *fastPrimHmCsrM              = 0x15L;               /* broadcast CSR with hold Master   */
  *fastSecadW       = CSR0 ;             /* secondary address cycle */
  *fastRndmWDis   = LNE  ;               /* data write cycle */

  /* FRDB; without Buffer address !! */
  *fastPrimDsr                     = prim_start;       /* primary address cycle   */
  *fastSecadW                     = 0x0L;               /* secondary address cycle */
  *fastStartFrdbWithClearWc = 0x0A007FFFL; /* start dma, max 8000 words; VME Block     */
  *fastDisconRm                  = reg32;              /* disconnect and release Master  */
  *fastStoreFrdbWc             = reg32;              /* get wordcount*/

  *fastWriteOutReg           = 0x100L;              /* set NIM1 */
  *fastWriteOutReg           = 0x1000000L;       /* clr NIM1 */

  /* set disable ram mode command to sequencer ram */
  *fastDisableRamMode = reg32  ;

/* wait short moment, dummy loop */
 for (iloop=0;iloop<4;iloop++)
    reg32 ++ ;

/* disable sequencer ram load mode */
*sfi.sequencerRamLoadDisable = reg32;

/* enable sequencer again */
*sfi.sequencerEnable=reg32;

return(Return);
}
```

6.4.7 Event Readout example using a Ram List

```
unsigned long EventReadRamFast(rStart,prim_lpa,Buffer)

unsigned long rStart;
unsigned long prim_lpa;
unsigned long Buffer;
{
  register  FB_Status, reg32;
  register Exit   = 0;
  unsigned long ExpWords;
  unsigned long i,j;
  unsigned long i_errors=0x0L;
  static unsigned long tReg=0x73, aReg=0x15;

  EnableForceA24D32Mode();
  i_errors = 0;

/* wait until NIM1 active (no dead timeout !) */
    reg32 = *sfi.sequencerFifoFlagAndEclNimInputReg ;
    while ((reg32 & 0x00001000) == 0 )
    {
      reg32 = *sfi.sequencerFifoFlagAndEclNimInputReg ;
    }

/*  generate Test ECL outputs (all four outputs) only Tests 10us   */
    *sfi.writeVmeOutSignalReg  =  0xf0L     ;
/* dummy loop */
    for (i=0;i<6;i++)
      *sfi.writeVmeOutSignalReg  =  0x0L ;
      *sfi.writeVmeOutSignalReg  =  0xf00000L ;

    /* sequecer commands; load new VME buffer address and start RAM List */
    *fastLoadDmaAddressPointer = Buffer;     /* load new buffer address */
    /* enable execution of command list in sequencer ram at address */
    *(fastEnableRamSequencer+rStart/4L) = rStart; /* casting ! */


/* wait for sequencer done */
Exit = 0;
do
   {
   reg32 = *sfi.sequencerStatusReg;
   switch(reg32 & 0x00008001)
      {
      case 0x00008001:
         /* OK */
         FB_Status = 0;
         Exit = 1;
        /* read seq fifo flags */
        /* check if EMPTY ; if EMPTY then Dummy read and Loop until not Empty */
        /*                  else if not empty then read fifo */

        reg32 = *sfi.readSeqFifoFlags;

        if((reg32 & 0x00000010) == 0x00000010)
           {
           reg32 = *sfi.readSeq2VmeFifoBase; /* dummy read */
           reg32 = *sfi.readSeqFifoFlags;
           if((reg32 & 0x00000010) == 0x00000010)
              {
              /* Error !!! */
              FB_Status  = *sfi.sequencerStatusReg & 0x0000ffff;
              FB_Status |= 0x00100000;
              }
           }
         /* read fifo */
         ExpWords = *sfi.readSeq2VmeFifoBase;
         break;

      case 0x00000001:
         /* Not Finished */
         break;
      case 0x00000000:
         /* Not Initialized */
         FB_Status  = *sfi.sequencerStatusReg & 0x0000ffff;
         FB_Status |= 0x00020000;
         Exit = 1;
         break;
      case 0x00008000:
         /* Bad Status is set, we will see */
         FB_Status = *sfi.sequencerStatusReg & 0x0000ffff;
         FB_Status |= 0x00010000;
         Exit = 1;
         break;
      }
   } while(!Exit);


  if(FB_Status)
     {
       processError(INTERPRET_STATUS | COUNT_ERROR | WAIT_ON_ERROR,
                    &i_errors,
                    "",
                    FB_Status,
                    *sfi.FastbusStatusReg1,
                    *sfi.FastbusStatusReg2);
       InitFastbus(aReg, tReg);
       i_errors ++;
     }
}

  if (!FB_Status)
    {
      printf("ExpWords:    %08lX\n",ExpWords);
      err_dmareg((ExpWords & 0x3f000000));
      printf("data read:\n");
      for(i=0;i<(ExpWords & 0x00ffffff);i+=8)
       {
         for(j=i; (j<(ExpWords & 0x00ffffff) && ((j-i)<8));j++)
           printf(" %08lX",((long*)Buffer)[j]);
         printf("\n");
       }
/*    WaitForEnter(); */

return(i_errors);

}
```

7 AUXILIARY CONNECTOR INTERFACE

7.1 AUXILIARY CONNECTOR SCHEMATIC

7.2 USER TRIGGER INTERFACE

The Auxiliary connector provides the possibility to add a user trigger logic on an auxiliary module. For setup of the user logic it is possible to write a 32-bit word from the VME CPU to the auxiliary interface. It is also possible to read a 32-bit word from the auxiliary interface, used for example as trigger information. The interface offers also input lines to generate a VME interrupt or to get bit informations and output lines to set/clear user logic and enable/disable data pathes.

Auxiliary signals:

Auxiliary Connector Bit **B42**: **TTL input**
This Input is readable via the „Sequencer Flag and ECL/NIM Input register". An open (not used) signal is high. It is prefered to use this input signal as a low active signal. If the interrupt AUX_B42 is enabled then a TTL high to low transition (down; edge sensitive) sets the interrupt flag AUX_B42_IRQ_FLAG.

Auxiliary Connector Bit **A10**: **TTL output** (low after Reset)
Auxiliary Connector Bit **A28**: **TTL output** (low after Reset)
Auxiliary Connector Bit **A45**: **TTL output** (low after Reset)
These Outputs will be set and cleared by access to the „VME Out-Signal Register".
Note: It is only allowed to set A45 if the sequencer is disabled.

Auxiliary Connector Bit **B40**: **TTL output** pulse (low active)
The VME CPU generates with the Key Address **generate AUX_B40 pulse** a TTL low active pulse on this line.

Auxiliary Connector Bit **B43**: **AUX2FB_OE_L** (**TTL input,** pullup resistor on this line**)**
The AUX2FB_OE_L signal enables (low active) the data path from the auxiliary connector (BUF_AD) to the „internal FB_IO_AD Bus".
Note: It is only allowed to enable if the sequencer is disabled !

Auxiliary Connector Bit **B44**: **AUX2FB_LE_L** (**TTL input,** pullup resistor on this line**)**
The AUX2FB_LE_L signal puts the AUX2FB data buffer either in latch mode (high) or in transparent mode (low).

Auxiliary Connector Bit **B45**: **FB2AUX_OE_L** (**TTL input,** pullup resistor on this line**)**
The FB2AUX_OE_L signal enables (low active) the data path from the „internal FB_IO_AD Bus" to the auxiliary connector (BUF_AD).

Auxiliary Connector Bit **B46**: **FB2AUX_LE_L** (**TTL input,** pullup resistor on this line**)**
The FB2AUX_LE_L signal puts the FB2AUX data buffer either in latch mode (high) or in transparent mode (low).

Auxiliary Connector Bit **A38**: **LCA_RDY_L** (**TTL output)**
This signal indicates that the LCAs on the SFI are configured (low). A high signal indicates a reset condition.

Example: readout a 32-bit Triggerword from the Auxiliary board

Use of B42: Receive external trigger (polling or VME_IRQ)
Use of A10: Clear external trigger
Use of A28: Enable data path from auxiliary connector to internal FB_IO_AD Bus
Set A28 to high to enable data path:
AUX2FB_OE_L = !(A28 & !LCA_RDY_L)

With B44 it is possible to store data. (TTL low : transparent; TTL High : latched)

Flow:
* poll if B42 or IRQ (data are now valid)
* set A28 to enable data path (**Note:** Master sequencer have to disabled or Idle !)
* read „Internal FASTBUS I/O BUS“ ($b0 1x00) with the VME CPU
* clear A28 to disable data path again
* set A10 (clear Trigger)
* clear A10

Example: write a 32-bit Triggerword to the Auxiliary board

Use of B40: Low active latch pulse to store data on user logic
Use of B46: FB2AUX_LE_L is set low (transparent)
Use of A45: Enable data path from internal FB_IO_AD Bus to the auxiliary connector.
Set A45 to high to enable data path:
FB2AUX_OE_L = !(A45 & !LCA_RDY_L)

Flow:
* write the 32-bit word to the **internal Aux-port register**
(implemented in the sequencer LCA)
* set A45 to enable data path (auxiliary buffer and LCA internal buffer)
and to inform the user logic.
(**Note:** Master sequencer has to be disabled !)
* generate pulse on B40 to latch data on user logic
* clear A45 to disable data path (auxiliary buffer and LCA internal buffer)

7.3 FASTBUS BLOCKTRANSFER DIRECT MODE INTERFACE

It is possible to push the read data to a user logic on a auxiliary module if the Direct Mode is enabled during a FASTBUS blocktransfer read cycle.

Auxiliary signals for blocktransfer read (direct mode):

Auxiliary Connector Bit **A40**: **FB2AUX_DMA_WEN_L** (**TTL output)**
The FB2AUX_DMA_WEN_L signal is active (low) during the blocktransfer if the Direct Mode is enabled to arm the user logic.

Auxiliary Connector Bit **A41**: **FB2AUX_DMA_WCLK** (**TTL output)**
With FB2AUX_DMA_CLK signal (low to high) the data on the BUF_AD Bus become valid (setup and holdtime = 15 ns).

Auxiliary Connector Bit **B37**: **FB2AUX_DMA_WAIT_L** (**TTL input,** pullup resistor)
It is possible to delay the blocktransfer read on FASTBUS with a low valid FB2AUX_DMA_WAIT_L signal.
Note: After the user logic has asserted this signal it is possible that the logic pushes still one data word to the auxiliary port.

uxiliary Connector Bit **B45**: **FB2AUX_OE_L** (**TTL input,** pullup resistor on this line**)**
The FB2AUX_OE_L signal enables (low active) the data path from the „internal FB_IO_AD Bus“ to the auxiliary connector (BUF_AD).

Auxiliary Connector Bit **B46**: **FB2AUX_LE_L** (**TTL input,** pullup resistor on this line**)**
The FB2AUX_LE_L signal puts the FB2AUX data buffer either in latch mode (high) or in transparent mode (low).

Auxiliary Connector Bit **A38**: **LCA_RDY_L** (**TTL output)**
This signal indicates that the LCAs on the SFI are configured (low). A high signal indicates a reset condition.

8 FRONT PANEL

The SFI front panel provides connectors for inputs and outputs and LEDs to indicate the module status and to use for debugging.

8.1 LEDS

After Power On Reset or Reset (NIM Input or VME SYSRESET, depending of the Jumper J502) the LCAs will be configured. At configuration time (1 to 2 seconds) all LEDs are on (Led test) except the RDY LED. After configuration only the RDY LED is on.

RDY Green; indicates that the SFI Logic (LCA) is ready
TST Red; indicates that after a Reset a **Module Test design** will be loaded

VSL Yellow; this LED is lit whenever the SFI is attached to VME as a slave
VMA Yellow, this LED is lit whenever the SFI is VME Master (Blocktransfer Read)

FB Yellow, this LED is lit whenever the SFI is FASTBUS Master
DMA Green, this LED is lit whenever the SFI carries out a DMA

SFF Green, this LED is lit whenever the SFI FB Master sequencer is enabled
SRA Green, this LED is lit whenever the SFI FB Master sequencer Ram Mode is enabled

L1 Green, this LED is lit whenever the corresponding bit in the Outsignal registers is set
L2 Red, this LED is lit whenever the corresponding bit in the Outsignal registers is set
L3 Yellow, this LED is lit whenever the corresponding bit in the Outsignal registers is set
L4 Green, this LED is lit whenever the corresponding bit in the Outsignal registers is set

8.2 INPUTS/OUTPUTS

The SFI owns at the front panel four differential ECL compatible Inputs and Outputs, four NIM Inputs, three NIM Outputs and four TTL level Outputs.

The **Output signals** are set/cleared (bit selective set/clear function) from the VME CPU or from the FASTBUS sequencer (see VME/Sequencer Out-Signal register).

The status of the **Input signals** (except the NIM Input Reset) can be read via a register (Sequencer Fifo Flag and ECL/NIM Input register) from the VME CPU.
Furthermore the three NIM Inputs and the ECL Input 1 signals are used for the VME Interrupt generation (edge sensitive logic; see VME Interrupt).

8.2.1 Schematic differential ECL In/Output

8.2.2 Schematic NIM In/Output

9 POWER REQUIREMENTS FOR THE SFI

Voltage	SFI Current	VME module(s) Current
+5 V	5 A	maximum 13 A
-5.2 V	1.5 A	
-2 V	0.3 A	

The Power consumption of the VME modules has to be added to the SFI consumption (5 V). But, take care that the maximum 5V-current **does not excees 18 A**.

On the SFI are voltage regulators to convert +15V/-15V to +12V/-12V for the VME modules. The current of each voltage is limited to 1A.

Voltage	SFI Current	VME module(s) Current
+15 V	max. 1 A	+12V maximum 1 A
-15 V	max. 1 A	-12 V maximum 1A

10 JUMPER

Jumper function

J500 = Selection of LCA Design
J502 = Selection of using VME SYSRESET

J951 = Enable/Disable 50 Ohm Termination on NIM Input 1
J952 = Enable/Disable 50 Ohm Termination on NIM Input 2
J953 = Enable/Disable 50 Ohm Termination on NIM Input 3
J954 = Enable/Disable 50 Ohm Termination on NIM Reset Input

J882 = Selection of VME Requester Level
J883 = VME Arbiter Type

Jumper setting

Jumper J500: **Selection of LCA Design**
closed: SFI Normal operating LCA Design will be loaded after Reset
open: SFI Test LCA Design will be loaded after Reset

Jumper J502: **Selection of using VME SYSRESET**
closed: A VME SYSRESET will reset the SFI and a SFI NIM Reset Input will **not** generate a VME SYSRESET
open: A VME SYSRESET will **not** reset the SFI and a SFI NIM Reset Input and a Power On Reset will generate a VME SYSRESET

Jumper J95x: **Enable/Disable 50 Ohm Termination on NIM Input**
closed: Enable 50 Ohm Termination
open: Disable 50 Ohm Termination

Jumper J883: **VME Arbiter Type**
closed: RWD (Release when done)
open: ROR (Release on Request)

Jumper 882: VME Master Requester Level

Level 3:

```
 20                  11
  O O O O O-O O O O O
  |           | |   |
  O O O O O-O O O O O
  1                  10
```

Level 2:

```
 20                  11
  O O O O O O-O O O O
    |     |     |   |
  O O O O O O-O O O O
  1                  10
```

Level 1:

```
 20                  11
  O O O O O O O O-O O
      |   |   |     |
  O O O O O O O O-O O
  1                  10
```

Level 0:

```
 20                  11
  O O O O O O O O O-O
        | |   | |
  O O O O O O O O O-O
  1                  10
```