"ATLAS ROD Busy Module"
Technical description and users manual.

The ROD Busy Modules gather and monitor the busy state of all the ATLAS Experiment Read Out Drivers. The sum of all Busy signals is sent to the Central Trigger Processor in order to control the experiment level one trigger rate.

DRAFT DRAFT DRAFT

Per Gällnö CERN/EP/ATE/dq
per.gallno@cern.ch
August 13, 2001

SPECIFICATIONS.......... 5
GENERAL DESCRIPTION.......... 8
INTRODUCTION.......... 8
ATLAS Experiment Dead Time Control.......... 8
SYSTEM DESCRIPTION.......... 8
Read-Out Driver Busy Handling.......... 8
THE ATLAS ROD-BUSY MODULE DESCRIPTION.......... 9
Basic Operation.......... 9
Manual Operation Mode.......... 10
Circular Buffer Operation Mode.......... 10
Additional features.......... 11
Picture of the ROD Busy Module.......... 12
DESIGN DESCRIPTION.......... 13
Input signal receivers and test drivers.......... 13
Output signal drivers.......... 13
Busy Input masking and summing.......... 13
Duration Counting.......... 13
Duration Count Buffering and Read-Out.......... 13
Duration Counter/Buffer Sequencer.......... 14
Global Busy Time-Out Service Requester.......... 14
VMEbus Data Bus Interface.......... 14
VMEbus Interrupt generator.......... 14
Module Configuration EEPROM.......... 15
ISP Module Firmware programming.......... 15
System Clock generation and distribution.......... 15
REFERENCE LITERATURE.......... 16
USER'S GUIDE.......... 17
FRONT PANEL FUNCTIONS.......... 17
Indicator LED's.......... 17
BUSY Inputs.......... 17
BUSY Outputs.......... 17
INSTALLATION PROCEDURE.......... 19
IMPORTANT WARNINGS.......... 19
Printed Circuit Board Lay-out.......... 19
Selecting VME base address.......... 20
Programming the Configuration EEPROM.......... 20
Interconnecting cables.......... 20
TEST PROCEDURES.......... 21
Diagnostics Headers.......... 21
Module acceptance test procedures.......... 22
PROGRAMMING MODEL.......... 23
Register offset map.......... 23
Register bit mapping.......... 23
Configuration EEPROM memory map.......... 26
HARDWARE MANUAL.......... 27
PARTS LIST.......... 27
CIRCUIT DIAGRAMS.......... 29
VHDL SOURCE CODES.......... 30
ip_reg_structure.vhd.......... 30
quad_cnt_struct.vhdl.......... 32
fifo_sequencer.vhd.......... 34
decod_fifo.vhd.......... 38
sreq_timer_struct.vhdl.......... 39
vme_if.vhd.......... 43
ISP FIRMWARE PROGRAMMING.......... 49
JEDEC Chain chip order.......... 49
ANNEX A.......... 50
COMPONENT DATA SHEETS.......... 50

SPECIFICATIONS

- **Front panel input signal levels**

  TRUE is 0 V and FALSE is + 0.8 V. The input is equipped with a 50Ω resistive Thévenin network resulting in an idle input voltage of 0.8 V.

- **Busy and Busy Carry-In Inputs**

  16 coaxial connectors (LEMO® #00). Any Busy input may be used as a Busy Carry-In input from another ROD Busy module.

- **Front panel output signal levels**

  TRUE is 0 V and FALSE is + 5.0 V. The drivers are of FAST-TTL open-collector type, able to sink up to 64 mA.

- **Busy Out Outputs**

  4 coaxial connectors (LEMO® #00), two are used to drive following ROD Busy modules or the Central Trigger Processor Busy input and the other two for monitoring purposes.

- **Input Test Register**

  A 16 bit VME register feeding the Busy Inputs via O/C drivers.

- **Input Monitoring**

  All 16 inputs may be monitored by reading the
  Busy State Register.

- **Input Enable**

  Each Busy Input can be enabled/disabled by setting bits in the Busy Masking Register.

- **Busy Duration Monitor**

  Max duration: $2^{16} * 1/10 * 10^{-6} = 6.55$ ms,
  (i.e. 16 bit counters incremented at 10 MHz)
  Counter outputs feed Busy Duration Buffer.
  Counter reset via a global reset command.
  Counters not affected by the state of the Input
  Masking Register bits.

- **Busy Duration Buffers**

  By FIFO 16x512 written every ≈ 6.5 ms.
  FIFO’s are full after ≈ 3.3 sec.
  FIFO’s are reset by global command.

Each of the 16 FIFO’s are readable from VME.

- **Software Busy Generation**

  A bit in a register is implemented in order to generate a global test busy under program control.

- **Busy Out Generation**

  The sum of all enabled Busy Inputs.

- **Busy Out Time-Out**

  An interrupt request may be generated when the Busy Out has been asserted longer than a programmable time-out.

- **Busy Out Monitoring**

  State reflected by a bit in the Status Register.

- **Internal clock generator frequency**

  10.00 MHz, 100 ppm

- **Back plane protocol**

  VME Slave: A24,A16/D16 (only VME connector P1 used)

- **Address Modifiers**

  Standard: 39, 3A, 3D, 3E
  Short : 29, 2D

- **VME Interrupter**

  ROAK type (release on acknowledge),
  programmable IRQ priority level (1 to 7),
  programmable STATUS I/D D08(odd)

- **Configuration ROM**

  EEPROM to store manufacturer/board/revision ID

- **Power Requirements**

  1.2 A @ + 5 V
  1.6 mA @ + 12 V
  1.0 mA @ - 12 V

- **Module PCB size**

  233.4 * 160.0 mm (height * width)

- **Front panel size**

  261.9 * 20.0 mm (6U * 4TE)

GENERAL DESCRIPTION

INTRODUCTION

ATLAS Experiment Dead Time Control

The data flow in the ATLAS sub-detector acquisition systems needs to be controlled in order to prevent information losses in the case the data buffers in the Front End, Read Out Drivers (ROD) or Read Out Buffers (ROB) get saturated.

Three different mechanisms to control the data flow will be implemented:

By ***Back pressure*** using a XON/XOFF protocol on the read-out links between the ROD's and the ROB's.
By ***Throttling*** to slow down the level one (LVL1) trigger rate from the CTP when the ROD data buffers are nearly filled.
By ***Prevention*** introducing a constant dead-time combined with one set by a pre-programmed algorithm in the CTP in order to avoid buffer overflow in the Front End. The constant dead-time is chosen to be 4 BC's after each LVL1 and the algorithm, called "leaky bucket", limits the number of LVL1 to 8 in any window of 80µs.

The introduction of a dead-time by a *throttling* mechanism is based on a ROD busy signaling scheme informing the Central Trigger Processor about the state of the ROD data buffers as each ROD is able to produce a ROD-Busy signal when its buffer is filled up. The busy signals from each ROD are summed and monitored in ROD-Busy Modules connected in a tree structure to finally produce a veto signal for the CTP. The ROD Busy signaling scheme and associated hardware will be described in this context.

SYSTEM DESCRIPTION

Read-Out Driver Busy Handling

The Read-Out Drivers (ROD), of which there will be several hundred in the ATLAS experiment, buffer, process and format the data from the Front End electronics before being sent to the Read-Out Buffers (ROB).

If the data buffers in the ROD are close to get filled up the Level-1 trigger rate must be reduced. A way of achieving this is to send a busy flag to the CTP to introduce a dead-time.

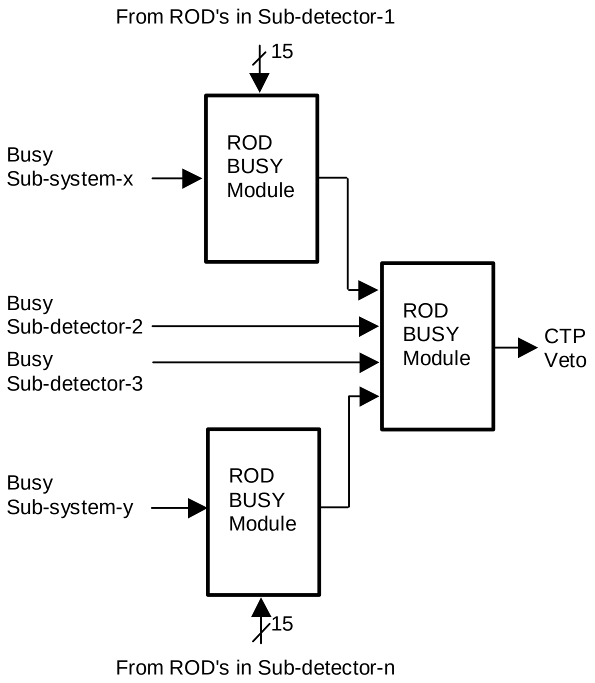


Figure 1. The ROD -Busy tree structure

Each ROD produces a Busy signal, which is sent to a ROD-Busy module together with Busy signals from other ROD's in the same sub-system. The ROD-Busy module sums the incoming Busy signals to produce one Busy signal of the particular sub-system. In turn the sub-system Busy signal is summed with other sub-system Busy signals in another Busy module to form a sub-detector Busy signal. Finally all sub-detector Busy signals are gathered to form a Busy input to the CTP.

THE ATLAS ROD-BUSY MODULE DESCRIPTION

Basic Operation

- The ROD-Busy module has been designed to perform the following functionality:
- Collect and make a logical OR of up to 16 Busy input signals.
- Monitor the state of any input Busy signal.
- Mask off any input Busy signal in the case a ROD is generating a false Busy state.

- Measure the integrated duration any Busy input is asserted for a given time period.
- Store a history of the integrated Busy duration for each input.
- Generate an interrupt if any Busy input is asserted for longer than a pre-set time limit.
- Generate a Busy output serving as an input for a subsequent ROD-Busy module in the tree structure or as a veto for the CTP.

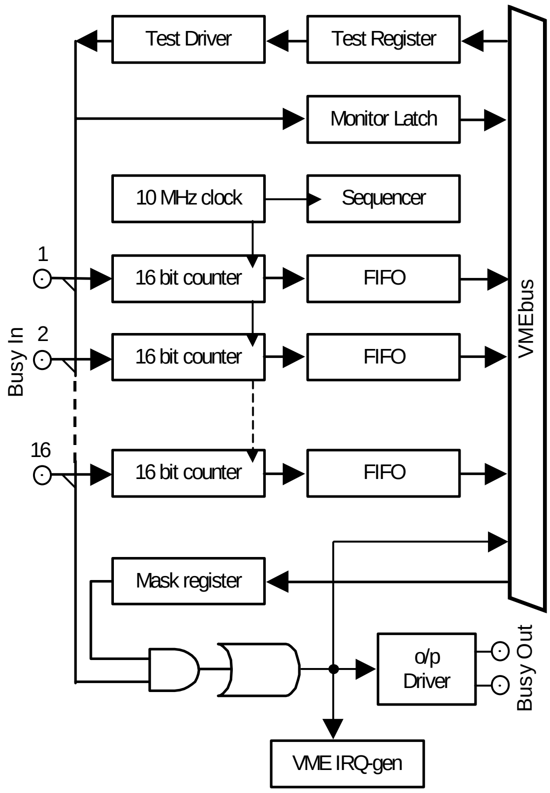


Figure 2. ROD -Busy module block diagram

Manual Operation Mode

In this mode of operation are the resetting and enabling of the counters, as well as resetting, writing and reading of the FIFO buffers done entirely under program control. The FIFO empty and full status flags for each FIFO are available to the VMEbus.

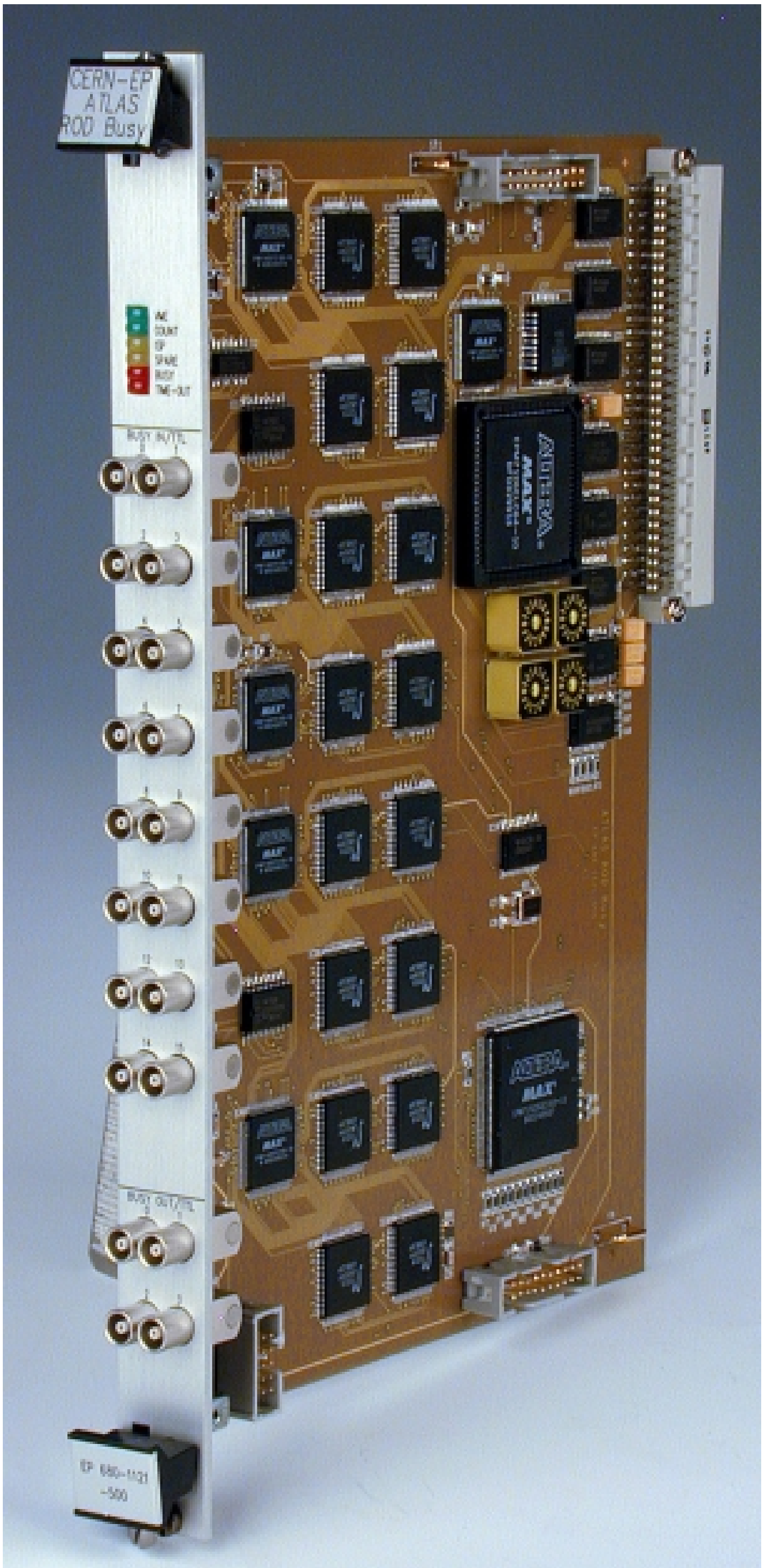
Circular Buffer Operation Mode

In this mode of operation is the transfer of data from the counters to the FIFO controlled by a timed sequencer. Bits may be set in a register in order to allow a circular buffer operation, i.e. a word is read out from the FIFO for each word written when the FIFO full flag is present. The maximum time between two consecutive data transfers from counter to FIFO is 6.55 ms. This time may be adjusted in a 16 bit VME register.

Additional features

- Each input path may be tested from bits in a VME test register.
- A status bit reflects the state of the Busy Out.
- A bit may be set in a control register in order to turn on a global Busy signal on all Busy Outputs.
- The Busy Time-Out service requester may be controlled by software functions, i.e. enable, disable, set and clear of the service request.
- The VMEbus interrupter may be tested with a software function.
- The module may be globally reset by a software function.

Picture of the ROD Busy Module



DESIGN DESCRIPTION

Input signal receivers and test drivers

The inputs are terminated with a Thévenin network resulting in a 50Ω resistive input impedance and calculated to give a + 0.8 V idle voltage. A Busy TRUE input corresponds to a 0 V level and a Busy FALSE to a + 0.8 V level. The input voltage threshold is set to + 0.4 V and the ultra fast input comparators have an internal hysteresis circuit producing clean input signals even when receiving data over long lines. All inputs may be monitored by reading a 16 bit input status VME register. Each input may be tested by being pulled down by an internal open-collector driver connected in turn to a 16 bit VME test register.

Output signal drivers

The four Busy Out outputs are driven by FAST TTL open-collector drivers. The outputs have the following characteristics and usage:

0. Pulled up to + 5 V by 10 kΩ and should be used to drive a following Busy Input or the CTP Busy Input.
1. Same as 0.
2. Pulled up to + 5 V by 510 Ω and should be used for monitoring purposes, i.e. oscilloscope etc.
3. Same as 2.

Busy Input masking and summing

The cleaned up input signals drive the Busy Summing circuit and the Busy Duration counters. The input signals to the Summing circuit may be masked off in order to isolate faulty ROD units. The Summing circuit produces a global Busy signal which is fed to the four Busy Out outputs. A control bit may be set to produce a global Busy Out for system test purposes. This block is implemented in a FPGA named ***ip_reg_structure***.

Duration Counting

The 16 bit duration counters increment at a speed of 10 MHz as long as there are Busy In signals on the inputs. There are global counter enable and reset functions generated by either accessing VME control bits or by the Buffer Sequencer. The sixteen counters are implemented in four FPGA's named ***quad_count_struct***.

Duration Count Buffering and Read-Out

The 512 word deep FIFO's buffer the Duration Counter data until read out by the VMEbus. There are global FIFO write cycle and reset functions generated by either accessing VME control bits or by the Buffer Sequencer. The FIFO read cycles are either done by the VMEbus or by the Buffer Sequencer. Control bits enable the FIFO's to be configured as circular buffer, i.e. they maintain always the history of the 512 last entered Duration Count figures. If not configured as circular buffers the FIFO's only will contain the first 512 entered Duration Count figures.

Duration Counter/Buffer Sequencer

The sequencer, when enabled, handles the control of the Duration counters and the FIFO's. A 16 bit down counter with a VME programmable shadow register, clocked by the 10 MHz clock, is used to set the rate for transferring the duration counts to the FIFO's. This block is implemented in a FPGA named ***fifo_sequencer***.

Global Busy Time-Out Service Requester

The Time-Out circuit monitors the duration of the global busy signal and generates a service request if a certain time limit is reached. Two 16 bit counters, magnitude comparators and VME programmable registers are used for this monitoring circuitry. An Interval counter/comparator/register circuit sets the frequency when the two counters are reset. The Limit counter/comparator/register circuit, where the counter increments during the Busy is true, generates a service request if the preprogrammed level is attained before being reset by the Interval circuit. Both counters are incremented at 10 MHz. The Time-Out service request may be programmed to trigger a VMEbus interrupt. This block is implemented in a FPGA named ***sreq_timer_struct***.

VMEbus Data Bus Interface

The VME bus slave interface is of conventional type and accepts only 16 bit word data cycles (D16). The addressing can either be standard or short (A24 or A16). Address pipelining and address only cycles are accepted. Four hexa-decimal switches are used for setting the module's base address. This block is implemented in a FPGA named ***vme_if***.

VMEbus Interrupt generator

An VMEbus interrupt can be generated when a Time-Out service request occurs. The interrupt generator is controlled by a control register where the VME Interrupt Request level is programmed and the interrupter is enabled. Another register contains the Status/ID information in an 8 bit format (D<7..0>). This block is also implemented in the FPGA named ***vme_if***.

Module Configuration EEPROM

Manufacturer identification, module identification and serial number, as well as module revision number should be stored in this non-volatile memory chip. There are spare locations for storing supplementary information. A strap must be installed in order to program this memory chip.

ISP Module Firmware programming

All ALTERA® FPGA chips, except for the VMEbus interface chip, are programmed with an In-System Programming scheme, using a *"Byte-Blaster"* adapter connected to a PC, where the ALTERA MAX-PLUS® programming software is installed.

System Clock generation and distribution

An internal 10 MHz system clock generator is used and a clock driver fan-out chip is used to drive the seven impedance matched clock lines, each terminated with a series RC network.

REFERENCE LITERATURE

- ATLAS Level-1 TDR Chapter 20
  http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/TDR/tdr.html
- R. Spiwoks: "Dead-time Generation in the Level-1 Central Trigger Processor", ATLAS Internal Note
- P. Gällnö:"The ATLAS ROD Busy Module" ATLAS ROD Workshop, University of Geneva, Nov. 1998
  http://mclaren.home.cern.ch/mclaren/atlas/conferences/ROD/programme.htm
- ALTERA® IN-System Programmability Handbook

USER'S GUIDE

FRONT PANEL FUNCTIONS

Indicator LED's

The six front panel indicator LED's show the state or activity of a function:

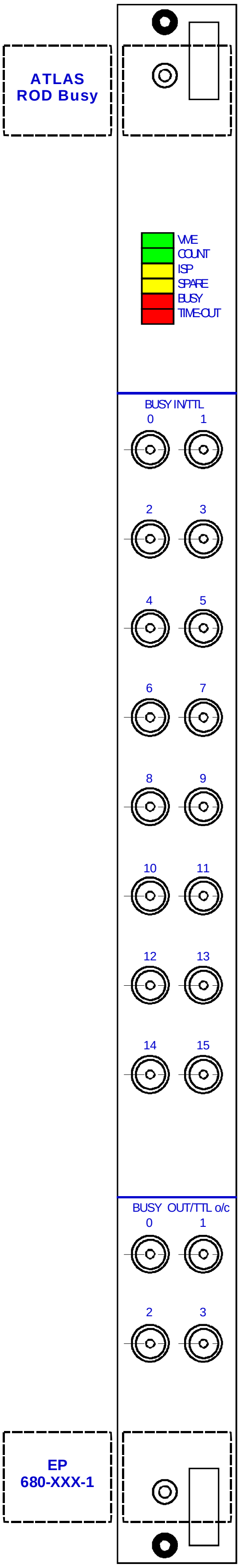
- **VME** - The module responds to a VME access
- **COUNT** - The duration counters are enabled
- **ISP** - In-System Programming is active
- **SPARE** - Data transfer from counters to FIFO's
- **BUSY** - BUSY Out is active
- **TIMEOUT** - There is a service request present

BUSY Inputs

16 Busy inputs where input 0 corresponds to D<0> and input 15 to D<15>in the VME data word. (NB on the prototype corresponds input 14 to D<0> and input 1 to D<15>)

BUSY Outputs

Use outputs 0 and 1 to drive following BUSY modules or the CTP and outputs 2 and 3 to drive external equipment or instruments.


ATLAS
ROD Busy
VME
COUNT
ISP
SPARE
BUSY
TIME-OUT
BUSY IN/TTL
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
BUSY OUT/TTL o/c
0
1
2
3
EP
680-XXX-1

INSTALLATION PROCEDURE

IMPORTANT WARNINGS

1. ***The VME crate must be powered down before inserting or extracting the ROD BUSY module.***
2. ***The module must be thoroughly pushed into the VME crate and secured with the top and bottom fixing screws, in order to assure proper operation.***
3. ***Some components on the printed circuit board are sensitive to electro-static discharges. To avoid damage, minimize handling and take appropriate precautions against static discharges.***
4. ***Any modification to the pre-set adjustments or the firmware of the module must only be carried out by a specialist in a laboratory environment.***

Printed Circuit Board Lay-out

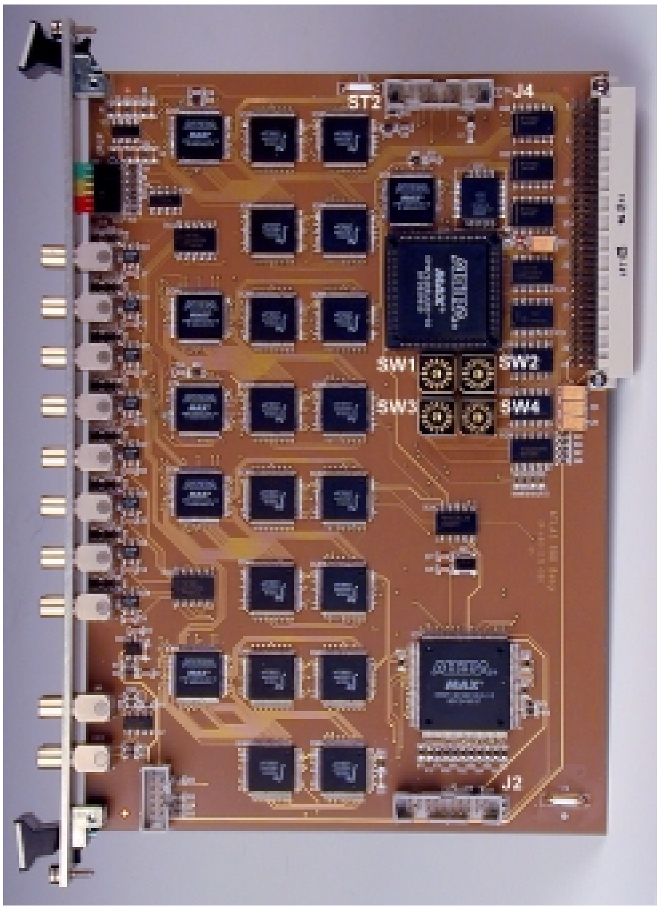


Figure 3. Board Lay-Out

Selecting VME base address

The module VME base address is selected by the hexa-decimal switches SW<1..4> and correspond to the following VME address bits:

SW1 23..20
SW2 19..16
SW3 15..12
SW4 11..08

Programming the Configuration EEPROM

The non-volatile memory chip for storing module identification data can only be programmed when strap ST2 is installed. After each write access to this EEPROM a wait cycle of at least 5 ms must be introduced.

Interconnecting cables

Cables between different equipment in the ROD BUSY tree structure must not exceed 100 m in order to assure proper operation. Low loss high quality 50Ω coaxial cables are recommended to be used, like the SCEM 04.69.11.XXX.X or the SCEM 04.61.11.145.5

TEST PROCEDURES

Diagnostics Headers

There are two Test Headers J2 and J4 mounted on the printed circuit board to be used for test and debugging purposes. The test headers fit the Hewlett Packard 100 kΩ Termination Adapter (part no. 01650-90920). A suitable Logic State Analyser is the HP 16500 series, for which a number of acquisition set-ups already exist.

TESTPAD J2

Counter/FIFO Sequencer related signals

SIGNAL-NAME	***PIN***	***POD***
NC	1	
NC	2	
CLK 10 MHz	3	CLK
CLK 10 MHz	4	D15
from test point (spare pin)	5	D14
from test point (spare pin)	6	D13
from test point (spare pin)	7	D12
from test point (spare pin)	8	D11
NC	9	D10
NC	10	D09
RST_L	11	D08
FIFFF0_L	12	D07
FIFEF0_L	13	D06
FIFREN0_L	14	D05
FIFOE0_L	15	D04
FIFWEN0_L	16	D03
FIFRST_L	17	D02
CNTEN_L	18	D01
CNTRST_L	19	D00
GND	20	GND

TESTPAD J4

Counter/FIFO Sequencer related signals

SIGNAL-NAME	*PIN*	*POD*
NC	1	
NC	2	
CLK 10 MHz	3	CLK
CLK 10 MHz	4	D15
from test point (spare pin)	5	D14
from test point (spare pin)	6	D13
IACKOUT_L	7	D12
IACKIN_L	8	D11
IACK_L	9	D10
SREQ_L	10	D09
HIADDR_L	11	D08
LOADDR_L	12	D07
DTACK_L	13	D06
WR_L	14	D05
DS0_L	15	D04
AS_L	16	D03
SVAR_2	17	D02
SVAR_1	18	D01
SVAR_0	19	D00
GND	20	GND

Module acceptance test procedures

To be determined. A test software exist for the functional debugging and testing of the module.

PROGRAMMING MODEL

Register offset map

Addr. offset	Register	R/W	Access	Remarks	
EE 1110-1110	FIFEF	R	W 16	FIFO empty flags	FIFO's + FIFO SEQUENSER
EC 1110-1100	FIFFF	R	W 16	FIFO full flags	
EA 1110-1010	SEQREG	R/W	W 16	Sequencer transfer interval register	
E8 1110-1000	FIFRCR	R/W	W 16	FIFO read control register	
E6 1110-0110	FIFWCR	R/W	W 2	Counter and FIFO write control register	
E4 1110-0100	FIFWEN	W	W -	Transfer all counters to FIFO's	
E2 1110-0010	FIFRST	W	W -	Reset all FIFO's	
E0 1110-0000	CNTRST	W	W -	Reset all counters	
DE 1101-1110	FIFO-READ-15	R	W 16	Busy duration FIFO 15	
DC 1101-1100	FIFO-READ-14	R	W 16	Busy duration FIFO 14	
DA 1101-1010	FIFO-READ-13	R	W 16	Busy duration FIFO 13	
D8 1101-1000	FIFO-READ-12	R	W 16	Busy duration FIFO 12	
D6 1101-0110	FIFO-READ-11	R	W 16	Busy duration FIFO 11	
D4 1101-0100	FIFO-READ-10	R	W 16	Busy duration FIFO 10	
D2 1101-0010	FIFO-READ-9	R	W 16	Busy duration FIFO 9	
D0 1101-0000	FIFO-READ-8	R	W 16	Busy duration FIFO 8	
CE 1100-1110	FIFO-READ-7	R	W 16	Busy duration FIFO 7	
CC 1100-1100	FIFO-READ-6	R	W 16	Busy duration FIFO 6	
CA 1100-1010	FIFO-READ-5	R	W 16	Busy duration FIFO 5	
C8 1100-1000	FIFO-READ-4	R	W 16	Busy duration FIFO 4	
C6 1100-0110	FIFO-READ-3	R	W 16	Busy duration FIFO 3	
C4 1100-0100	FIFO-READ-2	R	W 16	Busy duration FIFO 2	
C2 1100-0010	FIFO-READ-1	R	W 16	Busy duration FIFO 1	
C0 1100-0000	FIFO-READ-0	R	W 16	Busy duration FIFO 0	
9E 1001-1110					I/P
9C 1001-1100					
9A 1001-1010	BUSYMASK	R/W	W 16	SET/CLR Busy mask	
98 1001-1000	BUSYSTATE	R/W	W 16	Read i/p busy lines / Write test bits	
96 1001-0110	IVALREG	R/W	W 16	Interval Count Register	SREQ
94 1001-0100	LIMREG	R/W	W 16	Limit Count Register	
92 1001-0010	SREQSETCLR	W	W 2	Service Request Set/Clear Functions	
90 1001-0000	SREQCSR	R/W	W 4	Busy/Service Requester CSR	
8E 1000-1110					
8C 1000-1100					
8A 1000-1010					
88 1000-1000					
86 1000-0110	INTID	R/W	W 16	VME Interrupter Status/ID register	VME
84 1000-0100	INTCSR	R/W	W 16	VME Interrupter CSR	
82 1000-0010	SWIRQ	W	W -	VME Interrupt by soft function	
80 1000-0000	SWRST	W	W -	Reset module / data-less function	
00 0000-0000	Config. EEPROM	R/(W)	W 16	LSBytes in every Long Word. See specs.	

Register bit mapping

<u>VMEbus Interrupter</u>

SWRST - Software Reset Function (offset $82):

Writing any data to this address will cause the entire module to be reset to its initial state.

SWIRQ - Software Interrupt Function (offset $84):

Writing any data to this address will cause a VMEbus interrupt to be generated depending on the settings of the Interrupter Control Register.

INTID - Status ID Register (offset $86):

Data bits D7..D0 have read/write access and represent the Status ID or Interrupt Vector Number. Data bits D15..D8 are unused.

INTCSR - Interrupter Control Register (offset $84):

Bit	**3** (r/w)	**2** (r/w)	**1** (r/w)	**0** (r/w)
Usage	IRQ enable if '1'	Interrupt Request Level IRQ[7..1]		

Data bits D15..D4 are unused.

NB It must be avoided to use the non-existing IRQ[0] as this, with the present implementation, will cause the interrupt generator to hang. A software reset will bring the state machine back to initial state.

<u>Service Requester and Timer</u>

SREQCSR - Service Requester Control Register (offset $90):

Bit	**3** (r)	**2** (r/w)	**1** (r/w)	**0** (r)
Usage	SREQ active if '1'	SREQ enable if '1'	SW Busy set if '1'	Busy status active if '1'

Data bits D15..D4 are unused.

SREQSETCLR - Service Request Set/Clear Functions (offset $92):

Bit	**1** (w)	**0** (w)
Usage	Set SREQ if "2" Clear SREQ if "1" NOP if "3" or "0"	

Data bits D15..D2 are unused.

LIMREG - Limit Count Register (offset $94):

Data bits D15..D0 have read /write access. This register contains the limit number for clock pulses counted during the integrated assertion time of the summed busy signals. Reaching the limit count before being reset by the Interval Counter circuit causes the generation of a SREQ, if the counter is enabled by the SREQ enable bit.

IVALREG - Interval Count Register (offset $96):

Data bits D15..D0 have read /write access. This register contains the number at which point a reset is generated to the Limit Counter. The associated counter is incremented by the claock and enabled by the SREQ enable bit.

Input, Mask and Test Registers

BUSYSTATE - Busy State and Test Register (offset $98):

Data bits D15..D0 have read /write access. Reading the register show the immediate state of the Busy [15..0] inputs before masking. Reading a '1' mean "Busy asserted". However, writing ones to this register causes the Test outputs to drive the Busy inputs. The Busy inputs and the Test outputs are connected in a "wired OR" fashion.

BUSYMASK - Busy Masking Register (offset $9A):

Data bits D15..D0 have read /write access. Setting bits to '1' in this register enable the Busy inputs to be summed.

Counter and FIFO Sequencer

CNTRST - Counter Reset Function (offset $E0):

Writing any data to this address will cause all the counters to be reset to zero, if FIFWCR(1) = '0'.

FIFRST - FIFO Reset Function (offset $E2):

Writing any data to this address will cause all the FIFO pointers to be reset in any mode of operation.

FIFWEN - Transfer Data Counter_2_FIFO Function (offset $E4):

Writing any data to this address will cause the contents of all counter to be transferred to the FIFO's, if FIFWCR(1) = '0'.

FIFWCR - Counter/FIFO Write Control Register (offset $E6):

Bit / **Value**	**1** (r/w)	**0** (r/w)	**Usage**
0	0	0	Sequencer + Counters disabled, CNTRST + FIFRST + FIFWEN enabled
1	0	1	Sequencer disabled, Counters + CNTRST + FIFRST + FIFWEN enabled ***(Manual op.)***
2	1	0	Sequencer idle, Counters + CNTRST + FIFWEN disabled, FIFRST enabled
3	1	1	Sequencer + Counters + FIFRST enabled, CNTRST + FIFWEN disabled ***(Autom. op.)***

FIFRCR - FIFO Read Control Register (offset $E8):

Data bits D15..D0 have read /write access. When a bit D(n) is set to '1' the sequencer will read the first location in the FIFO(n) if the full flag is present. When the bit D(n) is set to '0' only the VME will be able to read the FIFO(n) at a corresponding address offset. (see memory map)

SEQREG - Sequencer Transfer Interval Register (offset $EA):

Data bits D15..D0 have read /write access. This is a shadow register to a down counter. When the count reaches zero the contents of the Busy Duration Counters are transferred to the FIFO's. The counter is decremented by the clock.

FIFFF - FIFO Full Flag Register (offset $EC):

Data bits D15..D0 have read only access. Bits reflecting the state of the FIFO full flags. Reading a '1' means corresponding FIFO is full.

FIFEF - FIFO Empty Flag Register (offset $EE):

Data bits D15..D0 have read only access. Bits reflecting the state of the FIFO empty flags. Reading a '1' means corresponding FIFO is empty.

Configuration EEPROM memory map

VME Address Offsets					
MSBYTE			LSBYTE		
31 24	23 16	15 8	7 0		
20	21	22	23		
24	25	26	27	Manufacturer ID (CERN)	MSBYTE
28	29	2A	2B	Manufacturer ID (CERN)	
2C	2D	2E	2F	Manufacturer ID (CERN)	LSBYTE
30	31	32	33	Board ID / Serial No.	MSBYTE
34	35	36	37	Board ID / Serial No.	
38	39	3A	3B	Board ID / Serial No.	
3C	3D	3E	3F	Board ID / Serial No.	
40	41	42	43	Board Revision No.	MSBYTE
44	45	46	47	Board Revision No.	
48	49	4A	4B	Board Revision No.	
4C	4D	4E	4F	Board Revision No.	LSBYTE

The IEEE Manufacturer ID# for CERN is: 080030 (hex)

HARDWARE MANUAL

PARTS LIST

PART	TYPE, VALUE, TOLERANCE	MANUFACTURER	SUPPLIER	SCEM/Cmd No	Qty
Integrated Circuit FPGA isp	EPM7192SQC160-10 PQFP	ALTERA			1
Integrated Circuit FPGA isp	EPM7160STC100-10 TQFP	ALTERA			6
Integrated Circuit FPGA	EPM7128ELC84-10 PLCC	ALTERA			1
Integrated Circuit Clock Driver	CDC341DW	Texas Instrument	Spoerle		1
Integrated Circuit EEPROM	AT28C16 PLCC 32	Atmel			1
Integrated Circuit FIFO	IDT72215LB TQFP 64	Integrated Device Technology			16
Integrated Circuit OP-Amp	µA741CD SOIC8	Texas Instrument	Radio Spares	277-1928	1
Integrated Circuit Oscillator	IQXO-70 10MHz	IQD	Radio Spares	190-0175	1
Integrated Circuit Comparator	LT1720CS8	Linear Technology			8
Integrated Circuit TTL Fast	74F07D hex o/c buffer	Philips Semiconductors			2
Integrated Circuit TTL Fast	74F521D octal eq. comparator			08.56.96.521.6	2
Integrated Circuit TTL Fast	74F545D octal tranceiver			08.56.96.545.8	3
Integrated Circuit TTL Fast	74F573D octal latch			08.56.96.573.4	2
Integrated Circuit TTL Fast	74F756D octal inv o/c buffer	Philips Semiconductors			2
Integrated Circuit TTL Fast	74F760D octal o/c buffer	Philips Semiconductors			1
Integrated Circuit CMOS	74HC365D hex TS buffer	Philips Semiconductors			1
Diode SMD	Si PMLL4448		CERN	08.51.10.010.8	1
LED double	Green Type: 1802-8832	MENTOR	Novitronic SA		1
LED double	Red Type: 1802-2232	MENTOR	Novitronic SA		1
LED double	Yellow Type: 1802-7732	MENTOR	Novitronic SA		1
Capacitor SMD	1206 330pF 50V cl 2		CERN	10.03.04.233.2	7
Capacitor SMD	1206 10nF 50V cl2		CERN	10.03.04.400.5	6
Capacitor SMD	1206 47nF 50V cl2		CERN	10.03.04.447.0	162
Capacitor SMD tantal	10µF 25V		CERN	10.82.01.570.0	2
Capacitor SMD tantal	47µF 10V		CERN	10.82.01.290.5	2
Resistor SMD	1206 10K 1%		CERN	11.24.05.400.3	13
Resistor SMD	1206 10M 1%		CERN		6
Resistor SMD	1206 15K 1%		CERN	11.24.05.415.6	1
Resistor SMD	1206 1K 1%		CERN	11.24.05.300.6	4
Resistor SMD	1206 270 1%		CERN	11.24.05.227.8	4
Resistor SMD	1206 300 1%		CERN	11.24.05.230.3	16
Resistor SMD	1206 330 1%		CERN	11.24.05.233.0	2
Resistor SMD	1206 4.7K 1%		CERN	11.24.05.347.1	16
Resistor SMD	1206 510 1%		CERN	11.24.05.251.8	3
Resistor SMD	1206 62 1%		CERN	11.24.05.162.8	23
Connector Coaxial	Duplex #00 for VME modules	LEMO SA, CH	CERN	09.46.11.188.8	10
Connector Header	male 2x5 pins type: 2510-6002	3M	Radio Spares	120-7230	1
Connector Header	male 2x10 pins type: 2520-6002	3M	Radio Spares	120-7268	2
Connector VMEbus	male 3x32 pins 90° DIN 41612		CERN	09.61.33.315.7	1
Socket PLCC	84 pins Type 284.7166.75.1157.SMT	3M	Radio Spares	203-9498	1
Jumper	for scope probe grounding		CERN	07.88.24.516.1	2
Straps	2X1 cut to size		CERN	09.55.10.708.9	4
Switch Hexadecimal	230057GB	EECO			4
Front Panel + extractor handles	machined & engraved to specs.		CERN	06.61.64.704.3	1

CIRCUIT DIAGRAMS

The circuit diagrams are presented in a hierarchical fashion, with an explanatory block diagram at the top level followed by the detailed design drawings or sub-blocks. ***(to be added later, please consult the item page for the ROD Busy Module in EDMS)***

VHDL SOURCE CODES

ip_reg_structure.vhd

```
-------------------------------------------------------------------------
-- File                 : ~gallno/rod_busy/vhdl/ip_reg_mask/ip_reg_structure.vhd
-- Title                : ATLAS Busy module input test, read and mask FPGA
-- Author               : Per Gallno ATE/EP/CERN
-- Date                 : 99 11 08
-- Updates              : 00 09 21 (comments added)
--                        00 11 07 (added LPM's)
-- Description          : IP_REG Structure
--
-- Comments             : Contains VME port, registers and masks.
--
--                        BUSY_IN and TEST_OUT are in posistive logic ie
--                        inversion is done outside device.
--
--                        Inputs are read when        ADDR(1) = '0'
--                        Test are written when       ADDR(1) = '0'
--                        Mask register R/W when      ADDR(1) = '1'
--
--                        Inputs are read by VME before masking.
--
--                        Mask bits must be '1' to enable the BUSY_IN inputs.
--
--                        Synthesized in SYNPLIFY and simulated in MAXPLUS-9.6
--                        on 00 11 15
--
--                        As LPM's are not always synthesized properly in
--                        SYNPLIFY, especially counters, a switch was made
--                        to use LEONARDO5J in the future. The lib/use statements
--                        have to be modified a bit and one can no longer
--                        use the same model for LEAPFROG and LEONARDO due to this.
--                        The LEONARDO give the same results when running
--                        MAXPLUS and simuation works as well. (00 11 17)
--
-- Device  :             Min EPM7096LC84 72% LC's and 88% I/O used
--                        Target EMP7160STC100 41% LC's and 71% I/O used
-------------------------------------------------------------------------
-- Packages:
   library IEEE;
--  library LPM;
      use IEEE.std_logic_1164.all;
      use WORK.lpm_components.all;
--          use lpm_components.all;
-------------------------------------------------------------------------
 entity IP_REG is

        port    (
                VMEDATA         : inout std_logic_vector (15 downto 0);
                ADDR            : in        std_logic;  -- VME address bit(1)
                CS_L            : in        std_logic;  -- Chip Select
                WR_L            : in        std_logic;  -- VME write line
                CLK             : in        std_logic;
                RST_L           : in        std_logic;
                BUSY_IN         : in    std_logic_vector (15 downto 0);
                TEST_OUT        : out   std_logic_vector (15 downto 0);
                BUSY_OUT_L      : out       std_logic
                );

        end IP_REG;
-------------------------------------------------------------------------
 architecture STRUCTURE of IP_REG is

-- SIGNAL DECLARATIONS:

  signal   VMEDIN, VMEDOUT,
                INPUTS, MASK                : std_logic_vector (15 downto 0);
  signal   CEN_TEST, CEN_MASK,
                CEN_READ , RST              : std_logic;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
 begin

-- TRI-STATE OUTPUTS:

               VMEDATA        <= VMEDOUT when (CS_L = '0') and (WR_L = '1')
                                       else (others => 'Z');

-- CONCURRENT STATEMENTS:

               VMEDIN         <= VMEDATA;

               RST            <= not RST_L;

               CEN_TEST       <= '1' when (CS_L = '0') and (WR_L = '0')
                                      and (ADDR = '0')
                                      else '0';

               CEN_MASK       <= '1' when (CS_L = '0') and (WR_L = '0')
                                      and (ADDR = '1')
                                      else '0';

               CEN_READ       <= '0' when (CS_L = '0') and (WR_L = '1')
                                       and (ADDR = '0')
                                      else '1';

               with ADDR select VMEDOUT <=
                                      INPUTS when '0',
                                      MASK   when '1',
                                      MASK   when others;

               BUSY_OUT_L     <= '0' when ((MASK and BUSY_IN) /= X"0000")
                                      else '1';

-- COMPONENT INSTANTIATION:


TSTREG  :  LPM_FF

               generic map    (lpm_width      => 16)
               port    map    (data           => VMEDIN,
                               clock          => CLK,
                               enable         => CEN_TEST,
                               aclr           => RST,
                               q              => TEST_OUT);
MASKREG :  LPM_FF

               generic map    (lpm_width      => 16)
               port    map    (data           => VMEDIN,
                               clock          => CLK,
                               enable         => CEN_MASK,
                               aclr           => RST,
                               q              => MASK);

INPREG   :  LPM_FF

               generic map    (lpm_width      => 16)
               port    map    (data           => BUSY_IN,
                               clock          => CLK,
                               enable         => CEN_READ,
                               aclr           => RST,
                               q              => INPUTS);

 end STRUCTURE;
---------------------------------------------------------------------------
```

quad_cnt_struct.vhdl

```
---------------------------------------------------------------------------
-- File                : ~gallno/rod_busy/vhdl/quad_counter/quad_cnt_struct.vhdl
-- Title               : ATLAS Busy module quad 16-bit counter
-- Author              : Per Gallno ATE/EP/CERN
-- Date                : 99 11 10
-- Updates             : 99 11 15 FF sclr implemented, clock always present to chip
--                       00 11 08 New Leapfrog LPM lib; dummy signals removed
--
-- Description         : QUAD_COUNT Structure containing 4 synchronisation F/F
--                       and four 16-bit binary counters with parallell outputs.
--                       The counter clocking is enabled by the synchronised
--                       BUSY signals. The reset of the counters are handled
--                       by the FIFO_SEQUENCER chip.
--
--                       Synthesized in SYNPLIFY on 00 11 15 but F/F's not found
--                       when running MAXPLUS, ie BUSY inputs unused !?!?!?
--
--                       As LPM's are not always synthesized properly in
--                       SYNPLIFY, especially counters, a switch was made
--                       to use LEONARDO5J in the future. The lib/use statements
--                       have to be modified a bit and one can no longer
--                       use the same model for LEAPFROG and LEONARDO due to this.
--                       The LEONARDO give the same results when running
--                       MAXPLUS and simuation works as well. (00 11 17)
--
--                       Target EPM7160STC100-10:
--
--                       Total dedicated input pins used:                 2/4  ( 50%)
--                       Total I/O pins used:                            73/80  ( 91%)
--                       Total logic cells used:                         68/160 ( 42%)
--                       Total shareable expanders used:                 0/160 (  0%)
--                       Total Turbo logic cells used:                          68/160  ( 42%)
--                       Total shareable expanders not available (n/a):  0/160 (  0%)
--                       Average fan-in:                                 11.05
--                       Total fan-in:                                   752
--
--
-- Comments            : Using LPM's
---------------------------------------------------------------------------
-- Packages:
                        library IEEE;
                        use IEEE.std_logic_1164.all;
                        use WORK.lpm_components.all;
---------------------------------------------------------------------------
 entity QUAD_COUNT is

        port    (
                CLK,RST_L,EN_L   : in    std_logic;
                BUSY             : in    std_logic_vector (3 downto 0);
                QA, QB, QC, QD   : out   std_logic_vector (15 downto 0)
                );

 end QUAD_COUNT;
---------------------------------------------------------------------------
 architecture STRUCTURE of QUAD_COUNT is

-- SIGNAL DECLARATIONS:

  signal   IBUSY    : std_logic_vector (3 downto 0);
  signal   RST       : std_logic;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

 begin

-- CONCURRENT STATEMENTS:

        RST     <= not RST_L;

-- COMPONENT INSTANTIATION:

 FF4 :  LPM_FF                          -- BUSY synchronisation F/F

                generic map      (lpm_width         => 4)
```

```
                port   map          (data             => BUSY,
                                     sclr             => EN_L,  -- clear FF EN_L = 1
                                     clock            => CLK,
                                     aclr             => RST,
                                     q                => IBUSY);

CNT0 : LPM_COUNTER
                generic map         (lpm_width        => 16)
                port   map          (clock            => CLK,
                                     cnt_en           => IBUSY(0),
                                     aclr             => RST,
                                     q                => QA);
CNT1 : LPM_COUNTER
                generic map         (lpm_width        => 16)
                port   map          (clock            => CLK,
                                     cnt_en           => IBUSY(1),
                                     aclr             => RST,
                                     q                => QB);
CNT2 : LPM_COUNTER
                generic map         (lpm_width        => 16)
                port   map          (clock            => CLK,
                                     cnt_en           => IBUSY(2),
                                     aclr             => RST,
                                     q                => QC);
CNT3 : LPM_COUNTER
                generic map         (lpm_width        => 16)
                port   map          (clock            => CLK,
                                     cnt_en           => IBUSY(3),
                                     aclr             => RST,
                                     q                => QD);

end STRUCTURE;
---------------------------------------------------------------------------
```

fifo_sequencer.vhd

```
---------------------------------------------------------------------------
-- File              : ~gallno/rod_busy/vhdl/sequencer/fifo_sequencer.vhd
-- Title             : ATLAS Busy module FIFO Sequencer
-- Author            : Per Gallno ATE/EP/CERN
-- Date              : 99 11 16
-- Updates           : 99 11 24 State decoding moved outside if/CLK statement,
--                              as otherwise f/f's are created.
--                     99 11 26 Status register for FIFO FULL/EMPTY implemented
--                     00 11 30 Continuing on project / using LEONARDO5J
--                     00 12 01 Remove comparator, do count down, since errors
--                          in maxplus
--                     01 06 19 FIFO flag read faulty address offset corrected
--                              "Manual VME" operation modified
-- Description: FIFO Sequencer Structure
-- Comments          : Contains VME port, registers and control logic
--
---------------------------------------------------------------------------
-- FIFWCR register bit map:
-- (1) : FIFO write by sequenser if '1', write by VME function if '0'
-- (0) : Counter[15:0] enabled if '1', disable counting if '0'
---------------------------------------------------------------------------
-- Packages:
  library IEEE;
        use IEEE.std_logic_1164.all;
        use WORK.lpm_components.all;
---------------------------------------------------------------------------
 entity  FIFO_SEQ is

        port    (
                VMEDATA         : inout   std_logic_vector (15 downto 0);
                ADDR            : in      std_logic_vector (5 downto 0);
                CS_L            : in      std_logic;
                WR_L            : in      std_logic;
                RST_L           : in      std_logic;
                CLK             : in      std_logic;
                FIFFF_L         : in      std_logic_vector (15 downto 0);
                FIFEF_L         : in      std_logic_vector (15 downto 0);
                FIFWEN_L        : out     std_logic;
                FIFOE_L         : out     std_logic_vector (15 downto 0);
                FIFRST_L        : out     std_logic;
                CNTRST_L        : out     std_logic;
                CNTEN_L         : out     std_logic;
                FIFREN_L        : out     std_logic_vector (15 downto 0);
                DUMMY_OUT       : out     std_logic
                );

        end  FIFO_SEQ;
---------------------------------------------------------------------------
 architecture STRUCTURE of  FIFO_SEQ is

-- CONSTANT DECLARATIONS:

        constant  cCNTRST       : std_logic_vector (7 downto 0) := x"20";
        constant  cFIFRST       : std_logic_vector (7 downto 0) := x"22";
        constant  cFIFWEN       : std_logic_vector (7 downto 0) := x"24";
        constant  cFIFWCR       : std_logic_vector (7 downto 0) := x"26";
        constant  cFIFRCR       : std_logic_vector (7 downto 0) := x"28";
        constant  cSEQREG       : std_logic_vector (7 downto 0) := x"2A";
        constant  cFIFFF        : std_logic_vector (7 downto 0) := x"2C";
        constant  cFIFEF        : std_logic_vector (7 downto 0) := x"2E";


-- SIGNAL DECLARATIONS:

        signal    IADDR                         : std_logic_vector (7 downto 0);
        signal    VMEDOUT,VMEDIN,FIFO           : std_logic_vector (15 downto 0);
        signal    FIFRCR,SEQREG,SEQCNT          : std_logic_vector (15 downto 0);
        signal    FIFWCR16,FIFEF,FIFFF          : std_logic_vector (15 downto 0);
        signal    FIFWCR                        : std_logic_vector (1 downto 0);
        signal    CS,RST,CNTRST                 : std_logic;
        signal    FIFWCR_CEN,FIFRCR_CEN         : std_logic;
        signal    FIFFF_CEN,FIFEF_CEN           : std_logic;
        signal    FIFWEN,FIFREN,CNTEQZERO       : std_logic;
```

```
        signal      SEQREG_CEN,SEQCNT_CEN           : std_logic;

-- COMPONENT DECLARATIONS:

        component DECOD_FIFO
                port        (
                            address             : in  std_logic_vector (7 downto 0);
                            fifo_no             : out std_logic_vector (15 downto 0)
                            );
        end component;

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

 begin

-- TRI-STATE OUTPUTS:

                VMEDATA         <= VMEDOUT      when (CS_L = '0') and (WR_L = '1')
                                                        and (IADDR >= cCNTRST)
                                                else (others => 'Z');

-- CONCURRENT STATEMENTS:

                VMEDIN          <= VMEDATA;     -- int VME data i/p bus

                IADDR           <= "00" & ADDR (5 downto 1) & '0';
                                        -- make a byte wide internal address bus

                DUMMY_OUT       <= ADDR(0); -- for easy simulation

        with IADDR select                       -- VME o/p data bus mux
                VMEDOUT         <=      FIFWCR16        when cFIFWCR,
                                        FIFRCR          when cFIFRCR,
                                        SEQREG          when cSEQREG,
                                        not FIFFF       when cFIFFF,
                                        not FIFEF               when cFIFEF,

                                        (others => '-')         when others;

                FIFWCR16 (1 downto 0)
                                <= FIFWCR;

                FIFWCR16 (15 downto 2)
                                <= (others => '-');

                CNTRST_L        <= '0' when (((CS_L = '0') and (WR_L = '0')
                                        and (IADDR = cCNTRST) and (FIFWCR(1) = '0'))
                                    or ((CNTRST = '1') and (FIFWCR(1) = '1')))
                                    or (RST_L = '0')
                                    else '1';

                RST             <= not RST_L;

                FIFRST_L        <= '0' when ((CS_L = '0') and (WR_L = '0')
                                        and (IADDR = cFIFRST))
                                    or (RST_L = '0')
                                    else '1';

                FIFWCR_CEN      <= '1' when ((CS_L = '0') and (WR_L = '0')
                                        and (IADDR = cFIFWCR))
                                    else '0';

                FIFRCR_CEN      <= '1' when ((CS_L = '0') and (WR_L = '0')
                                        and (IADDR = cFIFRCR))
                                    else '0';

                SEQREG_CEN      <= '1' when ((CS_L = '0') and (WR_L = '0')
                                        and (IADDR = cSEQREG))
                                    else '0';

                FIFFF_CEN       <= '0' when ((CS_L = '0') and (WR_L = '1')
                                        and (IADDR = cFIFFF))
                                    else '1'; -- freezes data during read cycle
                FIFEF_CEN       <= '0' when ((CS_L = '0') and (WR_L = '1')
                                        and (IADDR = cFIFEF))
```

```
                                    else '1';

                CNTEN_L         <= '0' when (((SEQCNT_CEN = '1') and (FIFWCR(1) = '1'))
                                        or    ((FIFWCR(0) = '1') and (FIFWCR(1) = '0')))
                                    else '1';

                CNTEQZERO       <= '1' when (SEQCNT = x"0000") else '0';

                FIFOE_L         <= not FIFO when ((CS_L = '0') and (WR_L = '1'))
                                    else (others => '1');

                FIFWEN_L        <= '0' when (((FIFWEN = '1') and (FIFWCR(1) ='1'))
                                        or ((CS = '1') and (IADDR = cFIFWEN)
                                        and (WR_L = '0') and (FIFWCR(1) ='0')))
                                    else '1';

 GFIFREN : for I in 15 downto 0 generate
                FIFREN_L(I)     <= '0' when (((FIFREN = '1') and (FIFRCR(I) = '1')
                                        and (FIFFF_L(I) = '0'))
                                        or ((CS = '1') and (FIFO(I) = '1')
                                        and (WR_L = '1') and (FIFRCR(I) ='0')))
                                    else '1';
                end generate;

-- SEQUENTIAL STATEMENTS:

 CSPULSER: process (CS_L,CLK,RST)
                 variable PULS : std_logic_vector (1 downto 0); -- := "00";
                begin
                  if RST = '1' then PULS := "00";
                  elsif rising_edge(CLK) then
                        case PULS is
                         when "00" =>
                          if    CS_L = '0' then         PULS := "01";
                       end if;
                         when "01" =>                   PULS := "10";
                         when "10" =>
                          if    CS_L = '1' then         PULS := "00";
                       end if;
                         when others =>                 PULS := "00";
                        end case;
                  end if;
                 CS <= PULS(0);
          end process;

 SEQFSM :  process (CLK,FIFWCR(0),CNTEQZERO,RST)
                 variable STATE : std_logic_vector (1 downto 0); -- := "00";
                begin
                  if RST = '1' then STATE := "00";
                  elsif rising_edge(CLK) then
                        case STATE is
                         when "00" =>                           -- idle, counters stopped + reset
                          if FIFWCR(0) = '1' then       STATE := "01";
                          end if;
                         when "01" =>                           -- counters and sequenser enabled
                           if CNTEQZERO = '1' then  STATE := "10";
                           end if;
                         when "10" =>                           -- read FIFOs
                                                        STATE := "11";
                         when "11" =>                           -- write FIFOs
                                                        STATE := "00";
                         when others =>                 STATE := "00";
                        end case;
                  end if;
                        case STATE is
                         when "00" =>       -- idle, counters stopped + reset
                                SEQCNT_CEN      <= '0';
                                CNTRST          <= '1';
                                FIFWEN          <= '0';
                                FIFREN          <= '0';
                         when "01" =>       -- counters and sequenser enabled
                                SEQCNT_CEN      <= '1';
                                CNTRST          <= '0';
                                FIFWEN          <= '0';
                                FIFREN          <= '0';
```

```
                         when "10" =>         -- read FIFOs
                                SEQCNT_CEN       <= '0';
                                CNTRST           <= '0';
                                FIFWEN           <= '0';
                                FIFREN           <= '1';
                         when "11" =>         -- write FIFOs
                                SEQCNT_CEN       <= '0';
                                CNTRST           <= '0';
                                FIFWEN           <= '1';
                                FIFREN           <= '0';
--                       when others => null;
                        end case;

                end process;

-- COMPONENT INSTANTIATION:

  DECODER:      DECOD_FIFO
                        port map
                                (address         => IADDR,
                                 fifo_no         => FIFO);

  LFIFWCR:      LPM_FF
                        generic map (lpm_width   => 2)
                        port    map (data        => VMEDIN (1 downto 0),
                                 clock           => CLK,
                                 enable          => FIFWCR_CEN,
                                 q               => FIFWCR);

 LFIFRCR:       LPM_FF
                        generic map (lpm_width   => 16)
                        port    map (data        => VMEDIN,
                                clock            => CLK,
                                enable           => FIFRCR_CEN,
                                q                => FIFRCR);


 LSEQREG:       LPM_FF
                        generic map (lpm_width   => 16)
                        port    map (data        => VMEDIN,
                                 clock           => CLK,
                                 enable          => SEQREG_CEN,
                                 q               => SEQREG);

 LSEQCNT:       LPM_COUNTER
                        generic map (lpm_width   => 16,
                                 lpm_direction
                                                 => "DOWN",
                                 LPM_HINT        => "CARRY_CNT_EN")
                        port   map   (data       => SEQREG,
                                 clock           => CLK,
                                 cnt_en          => SEQCNT_CEN,
                                 sload           => CNTRST,
                                 q               => SEQCNT);

 LFIFFF: LPM_FF
                        generic map (lpm_width   => 16)
                        port    map (data        => FIFFF_L,
                                 clock           => CLK,
                                 enable          => FIFFF_CEN,
                                 q               => FIFFF);

 LFIFEF:        LPM_FF
                        generic map (lpm_width   => 16)
                        port    map (data        => FIFEF_L,
                                 clock           => CLK,
                                 enable          => FIFEF_CEN,
                                 q               => FIFEF);

end STRUCTURE;
---------------------------------------------------------------------------
```

decod_fifo.vhd

```
---------------------------------------------------------------------------
-- File            : ~gallno/rod_busy/vhdl/sequencer/decod_fifo.vhd
-- Title           : ATLAS Busy module FIFO Sequencer - FIFO decoder
-- Author          : Per Gallno ATE/EP/CERN
-- Date            : 99 11 23
-- Updates         : 99 11 24 Rewritten as prev version wouldn't synthesise!
-- Description     : FIFO decoder bahavior
-- Comments        : Created due to problems between LPM and IEEE.numeric
--                   packages. (this version should work in all cases)
--
-- Problems        :        The full story: A decoder was implemented in the
--                          style that was proposed in the DOULOS course (ex10)
--                          and was implemented as a process in the top structure.
--                          There were however conflicts between packages.
--                          (see problems in fifo_sequencer.vhd) To cure the
--                          problem the decoder was moved to a separate entity,
--                          without LPM elements and then instatiated as a
--                          component in FIFO_SEQ. This tree structure compiled
--                          OK in Leapfrog and Synplify, only with a warning
--                          that the Address lines not connected. Checking the
--                          schematics in Synplify revealed that the DECODER
--                          box was empty !!!
--
-- Cure            :        A traditional (see below) address decoder was
--                          implemented.
---------------------------------------------------------------------------
-- Packages:
  library IEEE;
        use IEEE.std_logic_1164.all;
---------------------------------------------------------------------------
 entity DECOD_FIFO is

        port    (
                ADDRESS         : in    std_logic_vector (7 downto 0);
                FIFO_NO         : out   std_logic_vector (15 downto 0)
                );

        end  DECOD_FIFO;
---------------------------------------------------------------------------
 architecture BEHAVIOR of DECOD_FIFO is

 begin
                with ADDRESS select

                FIFO_NO <=      b"0000_0000_0000_0001" when x"00",
                                b"0000_0000_0000_0010" when x"02",
                                b"0000_0000_0000_0100" when x"04",
                                b"0000_0000_0000_1000" when x"06",

                                b"0000_0000_0001_0000" when x"08",
                                b"0000_0000_0010_0000" when x"0A",
                                b"0000_0000_0100_0000" when x"0C",
                                b"0000_0000_1000_0000" when x"0E",

                                b"0000_0001_0000_0000" when x"10",
                                b"0000_0010_0000_0000" when x"12",
                                b"0000_0100_0000_0000" when x"14",
                                b"0000_1000_0000_0000" when x"16",

                                b"0001_0000_0000_0000" when x"18",
                                b"0010_0000_0000_0000" when x"1A",
                                b"0100_0000_0000_0000" when x"1C",
                                b"1000_0000_0000_0000" when x"1E",
                                b"0000_0000_0000_0000" when others;

 end BEHAVIOR;
---------------------------------------------------------------------------
```

sreq_timer_struct.vhdl

```
---------------------------------------------------------------------------
-- File          : ~gallno/rod_busy/vhdl/sreq_timer_1/sreq_timer_struct.vhdl
-- Title         : ATLAS Busy module service request timer
-- Author        : Per Gallno ATE/EP/CERN
-- Date          : 00 11 03
-- Updates       : 00 11 21 preserve signal attr. on LPM_CMP eq o/p
--                          and split comparators to make job fit
-- Description: SREQ_TIMER Structure
-- Comments      : Using LPM's
--                 One bit vectors since data/q to f/f must be vectors
---------------------------------------------------------------------------
-- Packages:
                library IEEE;
                use IEEE.std_logic_1164.all;
                use work.lpm_components.all;
---------------------------------------------------------------------------
 entity SREQ_TIMER is

        port    (
                CLK,RST_L       : in     std_logic;
                VMEDATA         : inout  std_logic_vector (15 downto 0);
                ADDR            : in     std_logic_vector (2 downto 1);
                CS_L            : in     std_logic;  -- Chip Select
                WR_L            : in     std_logic;  -- VME write line
                BUSY_IN_L       : in     std_logic;
                BUSY_OUT_L      : out    std_logic;
                SREQ_L          : out    std_logic
                );

 end SREQ_TIMER;
---------------------------------------------------------------------------
 architecture STRUCTURE of SREQ_TIMER is

-- CONSTANT DECLARATIONS:

-- Corresponding to the internal registers VME address offsets,
-- when addressing during simulation these values should be divided by 2

  constant      cINTVALREG      : std_logic_vector (3 downto 0) := x"6";
  constant      cLIMITREG       : std_logic_vector (3 downto 0) := x"4";
  constant      cSREQCLR        : std_logic_vector (3 downto 0) := x"2";
  constant      cCSR            : std_logic_vector (3 downto 0) := x"0";

-- CSR Bit Map

--      bit 3:  SREQ Status set if '1'          R
--      bit 2:  SREQ Enable if '1'              R/W
--      bit 1:  SWBUSY Set if '1'               R/W
--      bit 0:  BUSY_OUT Status set if '1'      R

-- SREQ set & clear functions

--      bit 1:  SREQ set if '1' (for testing)   W
--      bit 0:  SREQ clear if '1'               W

-- SIGNAL DECLARATIONS:

  signal  IADDR               : std_logic_vector (3 downto 0);
  signal  VMEDIN,VMEDOUT: std_logic_vector (15 downto 0);
  signal  RST                 : std_logic;
  signal  LIMIT_REG           : std_logic_vector (15 downto 0);
  signal  LIMIT_CNT           : std_logic_vector (15 downto 0);
  signal  LIMITREG_CEN        : std_logic;
  signal  LIMIT_EQ            : std_logic;
  signal  LIMIT_EQ_T          : std_logic;
  signal  LIMIT_EQ_B          : std_logic;
  signal  INTERVAL_REG        : std_logic_vector (15 downto 0);
  signal  INTERVAL_CNT        : std_logic_vector (15 downto 0);
  signal  INTVALREG_CEN       : std_logic;
  signal  CSR                 : std_logic_vector (3 downto 0);
  signal  CSR16               : std_logic_vector (15 downto 0);
  signal  CSR_WCEN            : std_logic;
  signal  CSR_RCEN            : std_logic;
```

```
  signal   SW_BUSY          : std_logic_vector (0 downto 0);
  signal   IBUSY            : std_logic_vector (0 downto 0);
  signal   SREQ             : std_logic_vector (0 downto 0);
  signal   SREQ_EN          : std_logic_vector (0 downto 0);
  signal   SREQ_D           : std_logic_vector (0 downto 0);
  signal   SREQ_D1          : std_logic;
  signal   CLR_SREQ         : std_logic;
  signal   SET_SREQ         : std_logic;
  signal   CNT_ARST         : std_logic;
  signal   CNT_SRST         : std_logic;
  signal   CNT_SRST_T       : std_logic;
  signal   CNT_SRST_B       : std_logic;


-- EXEMPLAR/LEONARDO ATTRIBUTES:

  attribute preserve_signal                     : BOOLEAN;

  attribute preserve_signal of CNT_SRST_T       : signal is TRUE;
  attribute preserve_signal of CNT_SRST_B       : signal is TRUE;
  attribute preserve_signal of LIMIT_EQ_T       : signal is TRUE;
  attribute preserve_signal of LIMIT_EQ_B       : signal is TRUE;
  attribute preserve_signal of SREQ_D1          : signal is TRUE;


  -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

 begin

-- TRI-STATE OUTPUTS:

                VMEDATA         <= VMEDOUT when (CS_L = '0') and (WR_L = '1')
                                        else (others => 'Z');

-- CONCURRENT STATEMENTS:

                RST             <= not RST_L;

                VMEDIN          <= VMEDATA;

                IADDR           <= '0' & ADDR & '0'; -- make nibble wide int ADDR bus

                CSR16 (3 downto 0)
                                <= CSR;

                CSR16 (15 downto 4)
                                <= (others => '-');

                CSR(2)          <= SREQ_EN(0);

                CSR(1)          <= SW_BUSY(0);

                IBUSY(0)        <= SW_BUSY(0) or not(BUSY_IN_L);

                BUSY_OUT_L      <= not IBUSY(0);

                CNT_ARST        <= RST or not(SREQ_EN(0)) or CLR_SREQ;

                CNT_SRST        <= CNT_SRST_T and CNT_SRST_B;

                SREQ_L          <= not SREQ(0);

                SREQ_D1         <= LIMIT_EQ or SREQ(0);

                SREQ_D(0)       <= SREQ_D1;

                LIMIT_EQ        <= LIMIT_EQ_T and LIMIT_EQ_B;

        with IADDR select   -- VME o/p data bus mux
                VMEDOUT         <=      CSR16           when cCSR,
                                        LIMIT_REG       when cLIMITREG,
                                        INTERVAL_REG    when cINTVALREG,
                                        (others => '-') when others;
                CSR_WCEN
                        <= '1' when (CS_L = '0') and (WR_L = '0')
```

```
                      and (IADDR = cCSR)
                            else '0';

        CSR_RCEN
                <= '0' when (CS_L = '0') and (WR_L = '1')
                      and (IADDR = cCSR)
                            else '1';

        LIMITREG_CEN
                <= '1' when (CS_L = '0') and (WR_L = '0')
                      and (IADDR = cLIMITREG)
                            else '0';

        INTVALREG_CEN
                <= '1' when (CS_L = '0') and (WR_L = '0')
                      and (IADDR = cINTVALREG)
                            else '0';

        CLR_SREQ
                <= '1' when (CS_L = '0') and (WR_L = '0')
                      and (IADDR = cSREQCLR)
                                 and (VMEDIN (1 downto 0) = "01")
                            else '0';

        SET_SREQ
                <= '1' when (CS_L = '0') and (WR_L = '0')
                      and (IADDR = cSREQCLR)
                                 and (VMEDIN (1 downto 0) = "10")
                            else '0';

-- COMPONENT INSTANTIATION:


CSR3            :  LPM_FF          -- SREQ status bit in CSR

                generic map        (lpm_width          => 1)
                port    map        (data               => SREQ,
                                    clock              => CLK,
                                    enable             => CSR_RCEN,
                                    aclr               => RST,
                                    q                  => CSR(3 downto 3));

CSR2            :  LPM_FF          -- SREQ_EN control bit in CSR

                generic map        (lpm_width          => 1)
                port    map        (data               => VMEDIN(2 downto 2),
                                    clock              => CLK,
                                    enable             => CSR_WCEN,
                                    aclr               => RST,
                                    q                  => SREQ_EN);

CSR1            :  LPM_FF          -- SW_BUSY control bit in CSR

                generic map        (lpm_width          => 1)
                port    map        (data               => VMEDIN(1 downto 1),
                                    clock              => CLK,
                                    enable             => CSR_WCEN,
                                    aclr               => RST,
                                    q                  => SW_BUSY);

CSR0            :  LPM_FF          -- BUSY status bit in CSR

                generic map        (lpm_width          => 1)
                port    map        (data               => IBUSY(0 downto 0),
                                    clock              => CLK,
                                    enable             => CSR_RCEN,
                                    aclr               => RST,
                                    q                  => CSR(0 downto 0));


LIMCNT          : LPM_COUNTER      -- Limit Counter

                generic map        (lpm_width          => 16)
                port    map        (clock              => CLK,
                                    cnt_en             => IBUSY(0),
```

```
                                aclr            => CNT_ARST,
                                sclr            => CNT_SRST,
                                q               => LIMIT_CNT);

INTVALCNT       : LPM_COUNTER   -- Interval Counter

                generic map     (lpm_width      => 16)
                port    map     (clock          => CLK,
                                aclr            => CNT_ARST,
                                sclr            => CNT_SRST,
                                q               => INTERVAL_CNT);


LIMREG          : LPM_FF-- Limit Register

                generic map     (lpm_width      => 16)
                port    map     (data           => VMEDIN,
                                clock           => CLK,
                                enable          => LIMITREG_CEN,
                                q               => LIMIT_REG);

INTVALREG       : LPM_FF-- Interval Register

                generic map     (lpm_width      => 16)
                port    map     (data           => VMEDIN,
                                clock           => CLK,
                                enable          => INTVALREG_CEN,
                                q               => INTERVAL_REG);

LIMCMPT         : LPM_COMPARE   -- Limit Comparator top

                        generic map (lpm_width  => 8)
                        port    map (dataa      => LIMIT_CNT (15 downto 8),
                                     datab      => LIMIT_REG (15 downto 8),
                                     aeb        => LIMIT_EQ_T);

LIMCMPB         : LPM_COMPARE   -- Limit Comparator bottom

                        generic map (lpm_width  => 8)
                        port    map (dataa      => LIMIT_CNT (7 downto 0),
                                     datab      => LIMIT_REG (7 downto 0),
                                     aeb        => LIMIT_EQ_B);


INTVALCMPT      : LPM_COMPARE   -- Interval Comparator top

                        generic map (lpm_width  => 8)
                        port    map (dataa      => INTERVAL_CNT (15 downto 8),
                                     datab      => INTERVAL_REG (15 downto 8),
                                     aeb        => CNT_SRST_T);

INTVALCMPB      : LPM_COMPARE   -- Interval Comparator bottom

                        generic map (lpm_width  => 8)
                        port    map (dataa      => INTERVAL_CNT (7 downto 0),
                                     datab      => INTERVAL_REG (7 downto 0),
                                     aeb        => CNT_SRST_B);

SRQ_FF          :  LPM_FF       -- SREQ flip/flop

                generic map     (lpm_width      => 1,
                                lpm_avalue      => "1")
                port    map     (data           => SREQ_D (0 downto 0),
                                clock           => CLK,
                                aclr            => CNT_ARST,
                                aset            => SET_SREQ,
                                q               => SREQ (0 downto 0));


end STRUCTURE;
----------------------------------------------------------------------------
```

vme_if.vhd

```
--------------------------------------------------------------------------
-- File              : ~gallno/rod_busy/vhdl/vme_if/vme_if.vhd
-- Title             : ATLAS Busy module VMEbus interface
-- Author            : Per Gallno ATE/EP/CERN
-- Date              : 99 12 06
-- Updates           : 00 11 01 continuing on project
-- Description       : VME slave interface and interrupt generator
-- Comments          :
--------------------------------------------------------------------------
-- Packages:
  library IEEE;
     use IEEE.std_logic_1164.all;
--------------------------------------------------------------------------
 entity  VME_IF is

        port      (
                  VMEDATA          : inout std_logic_vector (7 downto 0);
                  VMEADDR          : in         std_logic_vector (7 downto 0);
                  AM               : in         std_logic_vector (5 downto 0);
                  SYSRST_L         : in         std_logic;
                  POWUPRST_L       : in         std_logic;
                  CHIPRST_L        : in         std_logic;          -- dedicated reset pin
                  CLK              : in         std_logic;
                  WR_L             : in         std_logic;
                  AS_L             : in         std_logic;
                  DS1_L            : in         std_logic;
                  DS0_L            : in         std_logic;
                  LW_L             : in         std_logic;
                  IACK_L           : in         std_logic;
                  IACKIN_L         : in         std_logic;
                  HIADDR_L         : in         std_logic;
                  LOADDR_L         : in         std_logic;
                  ID_ENWR_L        : in         std_logic;
                  SREQ_L           : in         std_logic;
                  DTACK_L          : out        std_logic;
                  IACKOUT_L        : out        std_logic;
                  IRQ_L            : out        std_logic_vector (7 downto 1);
                  RST_OUT_L        : out        std_logic;          -- connected to CHIPRST_L
                  RST_MODUL_L      : inout      std_logic;
                  EN_DBUFF_L       : out        std_logic;
                  LATCH_IP_L       : out        std_logic;
                  ID_CS_L          : out        std_logic;
                  ID_OE_L          : out        std_logic;
                  STATEVAR         : out         std_logic_vector (2 downto 0);
                  BUSY_CS_L        : out        std_logic;
                  TIMER_CS_L       : out        std_logic;
                  DUMMY_OUT        : out        std_logic; -- to keep unused i/p A0,AM2
                  SEQ_CS_L         : out        std_logic
                  );

 end  VME_IF;
--------------------------------------------------------------------------
 architecture BEHAVIOR of VME_IF is

-- CONSTANT DECLARATIONS:

        constant  cIDPROM_LL       : std_logic_vector (7 downto 0) := x"00";
        constant  cIDPROM_UL       : std_logic_vector (7 downto 0) := x"7E";

        constant  cSW_RST          : std_logic_vector (7 downto 0) := x"80";
        constant  cSW_IRQ          : std_logic_vector (7 downto 0) := x"82";
        constant  cINT_CSR         : std_logic_vector (7 downto 0) := x"84";
        constant  cINT_VECT        : std_logic_vector (7 downto 0) := x"86";

        constant  cTIMER_CSR       : std_logic_vector (7 downto 0) := x"90";
        constant  cTIMER_CLR       : std_logic_vector (7 downto 0) := x"92";
        constant  cTIMER_LIM       : std_logic_vector (7 downto 0) := x"94";
        constant  cTIMER_IVAL      : std_logic_vector (7 downto 0) := x"96";

        constant  cBUSY_STATE      : std_logic_vector (7 downto 0) := x"98";
        constant  cBUSY_MASK       : std_logic_vector (7 downto 0) := x"9A";
        constant  cSEQ_LL          : std_logic_vector (7 downto 0) := x"C0";
        constant  cSEQ_UL          : std_logic_vector (7 downto 0) := x"EE";
```

```
-- SIGNAL DECLARATIONS:

        signal   iADDR,iAM                : std_logic_vector (7 downto 0);
        signal   VMEDOUT,VMEDIN           : std_logic_vector (7 downto 0);
        signal   VECT_REG                 : std_logic_vector (7 downto 0);
        signal   CSR_REG                  : std_logic_vector (3 downto 0);
        signal   STATE                    : std_logic_vector (2 downto 0);
        signal   RST,IRQ,SREQ,IRQ_EN      : std_logic;
        signal   iAS,iDS,iDS0             : std_logic;
        signal   AMSTD,AMSHORT            : boolean;
        signal   SLAV_DTACK               : boolean;
        signal   SW_RST,SW_IRQ            : boolean;
        signal   IDROM_CHIP,SEQ_CHIP      : boolean;
        signal   TIMER_CHIP               : boolean;
        signal   BUSY_CHIP,RD,WR,DS       : boolean;
        signal   VME_OK,OE_DATA           : boolean;
        signal   INT_VECT,INT_CSR         : boolean;
        signal   INT_MATCH,ADDR_OK        : boolean;
        signal   EN_VECT,INT_DTACK        : boolean;
        signal   IRQ_RELEASE              : boolean;
        signal   DEL                      : std_logic_vector(0 to 3);
        signal   IRQ_LEV                  : std_logic_vector(2 downto 0);
        signal   iIRQ_L                   : std_logic_vector(7 downto 1);

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

 begin
                 ------------------------------------------
                 -- V M E   S L A V E  I N T E R F A C E --
                 ------------------------------------------

-- RESET GENERATION:

        -- This output should be connected to the dedicated RESET input
        -- of the chip called here CHIPRST_L:

                 RST_OUT_L      <= '0' when  (SYSRST_L = '0') or (POWUPRST_L ='0')
                                          else '1';

        -- Chip internal global reset line:

                 RST            <= not CHIPRST_L;

        -- Module general reset line to other chips on board:
        -- (should be buffered outside chip to  increase fan-out)

                 RST_MODUL_L    <= '0' when (CHIPRST_L = '0') or SW_RST
                                          else '1';

-- EXTERNAL/INTERNAL SIGNAL/FUNCTION INTERFACING:

        -- Tri-state data output drive:

                 VMEDATA                  <= VMEDOUT when
                                                  (RD  and (INT_CSR or INT_VECT))
                                                  or EN_VECT
                                                  else (others => 'Z');

        -- Internal data out bus mux:

                 VMEDOUT                  <= VECT_REG when INT_VECT or EN_VECT
                                                  else "00" & SREQ & IRQ & CSR_REG;

        -- Internal data in bus:

                 VMEDIN                   <= VMEDATA;

        -- Control signal interface and synchronisation:

                 RD             <= (WR_L = '1');                      -- boolean
                 WR             <= (WR_L = '0');                      -- boolean
                 DS             <= (DS1_L = '0') and (DS0_L = '0');   -- boolean
CTLSYNC:         process (CLK,DS1_L,DS0_L,AS_L)
                 begin
```

```
                    if rising_edge(CLK) then
                          iAS      <= not AS_L;
                          iDS      <= (not DS1_L) and (not DS0_L);
                          iDS0     <= not DS0_L;
                    end if;
                  end process CTLSYNC;

          -- Delay generator implemented as a shift register w synchr clear:

DELAY:    process (VME_OK,CLK)
          begin
                   if rising_edge(CLK) then
                    if (not VME_OK) then DEL <= "0000";
                    else
                          DEL(0) <= '1';
                          DEL(1) <= DEL(0);
                          DEL(2) <= DEL(1);
                          DEL(3) <= DEL(2);
                    end if;
                   end if;
                  end process DELAY;

-- DTACK GENERATION:

                  SLAV_DTACK      <= TRUE when
                                          (IDROM_CHIP      and (DEL(3) = '1')) or
                                          (SEQ_CHIP        and (DEL(2) = '1')) or
                                          (BUSY_CHIP       and (DEL(2) = '1')) or
                                          (TIMER_CHIP      and (DEL(2) = '1')) or
                                          (SW_RST          and (DEL(3) = '1')) or
                                          (SW_IRQ          and (DEL(1) = '1')) or
                                          (INT_VECT        and (DEL(2) = '1')) or
                                          (INT_CSR         and (DEL(2) = '1'))
                                    else FALSE;

                  DTACK_L         <= '0' when (INT_DTACK and (DS0_L = '0')) or
                                    (SLAV_DTACK and (DS0_L = '0') and (DS1_L = '0'))
                                          else '1';

-- ADDRESS BUS AND AM CODE HANDLING AND COMPARASION:

          -- Make byte wide internal address and AM busses:

                  iADDR(7 downto 0)
                                  <= VMEADDR(7 downto 1) & '0';

                  iAM             <= "00" & AM(5) & AM(4) & AM(3) & '0' & AM(1) & AM(0);

                  DUMMY_OUT       <= VMEADDR(0) or AM(2);              -- waisting 2 pins for the
                                                                       -- sake of easy simulation

          -- Responding AM codes:

                  with iAM select AMSTD       <= TRUE  when x"09"|x"0A"|x"0D"|x"0E"|
                                                            x"39"|x"3A"|x"3D"|x"3E",
                                                       FALSE when others;

                  with iAM select AMSHORT     <= TRUE  when x"29"|x"2D",
                                                       FALSE when others;

          -- Adress comparasion:

                  ADDR_OK         <= TRUE when
                                          (AMSTD and (HIADDR_L = '0') and (LOADDR_L = '0'))
                                          or (AMSHORT and (LOADDR_L = '0'))
                                          else FALSE;

-- EXTERNAL CONTROL, ADDRESS AND DATA BUS CONTROLS:

                  EN_DBUFF_L      <= '0' when OE_DATA else '1';

                  LATCH_IP_L      <= '0' when OE_DATA else '1';
-- FUNCTION (BOOLEAN) ADDRESS DECODING:

                  SW_RST          <= VME_OK and DS and (iADDR = cSW_RST) and WR;
```

```
                SW_IRQ          <= VME_OK and DS and (iADDR = cSW_IRQ) and WR;

                INT_VECT        <= VME_OK and DS and (iADDR = cINT_VECT);

                INT_CSR         <= VME_OK and DS and (iADDR = cINT_CSR);

                IDROM_CHIP      <= VME_OK and DS and
                                   ((iADDR >= cIDPROM_LL) and (iADDR <= cIDPROM_UL));

                SEQ_CHIP        <= VME_OK and DS and
                                   ((iADDR >= cSEQ_LL) and (iADDR <= cSEQ_UL));

                BUSY_CHIP       <= VME_OK and DS and
                                   ((iADDR = cBUSY_STATE) or (iADDR = cBUSY_MASK));

                TIMER_CHIP      <= VME_OK and DS and
                                   ((iADDR = cTIMER_CSR) or
                                    (iADDR = cTIMER_CLR) or
                                    (iADDR = cTIMER_LIM) or
                                    (iADDR = cTIMER_IVAL));

-- EXTERNAL CHIP SELECT GENERATION:

                ID_CS_L         <= '0' when IDROM_CHIP and (iADDR(1) = '1') and
                                        (RD or (WR and (ID_ENWR_L ='0')
                                        and (DEL(0) = '1') and (DEL(3) = '0')))
                                        else '1';

                ID_OE_L         <= '0' when IDROM_CHIP and RD
                                        else '1';

                BUSY_CS_L       <= '0' when BUSY_CHIP and
                                        (RD or
                                        (WR and (DEL(0) = '1') and (DEL(2) = '0')))
                                        else '1';

                SEQ_CS_L        <= '0' when SEQ_CHIP and
                                        (RD or
                                        (WR and (DEL(0) = '1') and (DEL(2) = '0')))
                                        else '1';

                TIMER_CS_L      <= '0' when TIMER_CHIP and
                                        (RD or
                                        (WR and (DEL(0) = '1') and (DEL(2) = '0')))
                                        else '1';

                ------------------------------------
                -- V M E   I N T E R R U P T E R --
                ------------------------------------

-- INTERRUPTER VECTOR AND CONTROL REGISTERS

VECTREG:        process (CLK,RST,VMEDIN,WR,INT_VECT)
                begin
                  if (RST = '1') then VECT_REG <= x"00";
                  elsif rising_edge(CLK) then
                    if (WR and INT_VECT and (DEL(0) = '1') and (DEL(1) = '0'))
                            then VECT_REG <= VMEDIN;
                    end if;
                  end if;
                end process VECTREG;

CSRREG:         process (CLK,RST,VMEDIN,WR,INT_CSR)
                begin
                  if (RST = '1') then CSR_REG <= x"0";
                  elsif rising_edge(CLK) then
                    if (WR and INT_CSR and (DEL(0) = '1') and (DEL(1) = '0'))
                            then CSR_REG <= VMEDIN (3 downto 0);
                    end if;
                  end if;
                end process CSRREG;

                IRQ_EN   <= CSR_REG(3);
```

```
                IRQ_LEV <= CSR_REG(2 downto 0);

-- INTERRUPT REQUEST FLIP/FLOP

INTREQ:  process (CLK,SREQ,IRQ_EN,SW_IRQ,IRQ_RELEASE,RST_MODUL_L)
                begin
                  if (RST_MODUL_L = '0')
                          or (IRQ_EN = '0')
                          or  IRQ_RELEASE
                           then IRQ <= '0';
                  elsif rising_edge(CLK) then
                    if (SREQ = '1') or SW_IRQ
                           then IRQ <= '1';
                    end if;
                  end if;
                end process INTREQ;

-- DRIVING INTERRUPT REQUEST LINES:

                with IRQ_LEV select iIRQ_L <=      "1111110" when o"1",
                                                   "1111101" when o"2",
                                                   "1111011" when o"3",
                                                   "1110111" when o"4",
                                                   "1101111" when o"5",
                                                   "1011111" when o"6",
                                                   "0111111" when o"7",
                                                   "1111111" when others;

                IRQ_L           <= iIRQ_L when (IRQ = '1') else "1111111";

-- MATCHING REQUEST LEVEL AND ADDRESS LINES:

                INT_MATCH       <= (IRQ_LEV = iADDR(3 downto 1)) and (IRQ = '1');

-- MISC:

                SREQ            <= not SREQ_L;

                ------------------------------------------
                -- V M E   S L A V E  +  I R Q   F S M --
                ------------------------------------------

-- STATE MACHINE:

IRQFSM :        process (CLK,RST,iAS,iDS,iDS0,LW_L,IACK_L,IACKin_L,ADDR_OK,
                     SLAV_DTACK,INT_MATCH,IRQ)
                begin
                  if RST = '1' then STATE <= o"0";
                  elsif rising_edge(CLK) then
                          case STATE is

                           when o"0" =>                    -- idle state
                           ---------
                            if (iAS = '1') and (LW_L = '1') and (IACK_L = '1')
                              and ADDR_OK
                                 then STATE <= o"2";       -- do slave cycle !
                            elsif (iAS = '1') and (LW_L = '1') and (IACK_L = '0')
                                 then STATE <= o"1";       -- do IACK cycle !
                            end if;

                           when o"2" =>                    -- slave: DS assert ?
                           ---------
                            if (iAS = '0') and (iDS = '0')
                                  then STATE <= o"0";      -- was an address only cycle !
                            elsif (iDS = '1')
                                  then STATE <= o"6";      -- continue in slave cycle !
                            end if;

                           when o"6" =>                    -- slave: DTACK assert ?
                           ---------
                            if (iDS = '0')
                                  then STATE <= o"2";      -- problem: BERR - go back !
                            elsif SLAV_DTACK
                                  then STATE <= o"4";      -- this is OK - go on !
                            end if;
```

```
                when o"4" =>                        -- slave: AS*DS negate ?
                ---------
                 if (iAS = '0') and (iDS = '0')
                        then STATE <= o"0";         -- end cycle - go back to idle !
                 end if;

                when o"1" =>                        -- irq: match ? / IACKin ?
                ---------
                 if (IACKin_L = '0') and INT_MATCH
                        then STATE <= o"3";         -- this irq cycle is for me !
                 elsif (iAS = '0')
                        then STATE <= o"0";         -- this irq cycle wasn't for me !
                 end if;

                when o"3" =>                        -- irq: DS0 assert ?
                ---------
                 if (iDS0 = '1')
                        then STATE <= o"7";         -- continue in cycle !
                 end if;

                when o"7" =>                        -- irq: DS0 * AS negate ?
                ---------
                 if (iAS = '0') and (iDS0 = '0')
                        then STATE <= o"5";         -- continue in cycle !
                 end if;

                when o"5" =>                        -- irq: IRQ negate ?
                ---------
                 if (IRQ = '0')
                        then STATE <= o"0";         -- end cycle - go back to idle !
                 end if;

                when others => STATE <= o"0";
                -----------

               end case;
          end if;
        end process IRQFSM;

-- STATE VARIABLE TO OUTPUT DECODING:

        with STATE select VME_OK        <= TRUE when o"6"|o"4",
                                                FALSE when others;

        with STATE select OE_DATA              <= TRUE when o"2"|o"6"|o"4"|o"3"|o"7",
                                                FALSE when others;

        with STATE select EN_VECT       <= TRUE when o"3"|o"7",
                                                FALSE when others;

        with STATE select INT_DTACK     <= TRUE when o"7",
                                                FALSE when others;

        with STATE select IRQ_RELEASE   <= TRUE when o"5",
                                                FALSE when others;

        IACKout_L       <= '0' when (IACKin_L = '0') and (STATE = o"1")
                                and not INT_MATCH
                           else '1';

        STATEVAR        <= STATE;       -- to ext monitor state machine

 end BEHAVIOR;
 ------------------------------------------------------------------------
```

ISP FIRMWARE PROGRAMMING

JEDEC Chain chip order

The files in the JTAG Chain File (.jcf) must have the following order:

```
[JTAG_CHAIN]
DEVICE_1=EPM7160S ip_reg_s.pof
DEVICE_2=EPM7160S sreq_tim.pof
DEVICE_3=EPM7192S fifo_seq.pof
DEVICE_4=EPM7160S quad_cnt.pof
DEVICE_5=EPM7160S quad_cnt.pof
DEVICE_6=EPM7160S quad_cnt.pof
DEVICE_7=EPM7160S quad_cnt.pof
```

ANNEX A

COMPONENT DATA SHEETS

(to be added later, please consult the item page for the ROD Busy Module in EDMS)