

LEON2-FT Processor User's Manual

Version 1.0
28-Oct-2002
GR-LEON2-FT-3

ADVANCE INFORMATION

Jiri Gaisler
Gaisler Research

This document has been produced under ESA contract 15102/01/NL/FM

Gaisler Research

jiri@gaisler.com

The LEON2-FT processor user's manual

Copyright 2002 Gaisler Research.

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

1	Introduction	7
1.1	Overview	7
1.2	Functional overview	8
1.2.1	Integer unit	8
1.2.2	Floating-point unit	8
1.2.3	Cache sub-system	8
1.2.4	Debug support unit	9
1.2.5	Memory interface	9
1.2.6	Timers	9
1.2.7	Watchdog	9
1.2.8	UARTs	9
1.2.9	Interrupt controller	9
1.2.10	Parallel I/O port	9
1.2.11	Watchpoint registers	9
1.2.12	Fault-tolerance	10
1.2.13	Performance	10
2	LEON integer unit	11
2.1	Overview	11
2.2	Instruction pipeline	12
2.3	Multiply instructions	12
2.4	Divide instructions	13
2.5	Processor reset operation	13
2.6	Exceptions	13
2.7	Watch-points	14
2.8	Register file EDAC protection	14
2.8.1	Operation	14
2.8.2	Register file protection control register	15
2.9	Floating-point unit	15
3	Cache sub-system	16
3.1	Overview	16
3.2	Instruction cache	16
3.2.1	Operation	16
3.2.2	Instruction cache flushing	17
3.2.3	Diagnostic cache access	17
3.2.4	Instruction cache parity	17
3.2.5	Instruction cache tag	18
3.3	Data cache	18
3.3.1	Operation	18
3.3.2	Write buffer	18
3.3.3	Data cache flushing	19
3.3.4	Diagnostic cache access	19
3.3.5	Data cache parity	19
3.3.6	Data cache tag	20
3.4	Cache Control Register	20
4	AMBA on-chip buses	22
4.1	Overview	22

4.2	AHB bus	22
4.3	APB bus.....	22
4.4	AHB transfers generated by the processor	22
5	On-chip peripherals	23
5.1	On-chip registers	23
5.2	Interrupt controller	24
5.2.1	Operation	24
5.2.2	Interrupt assignment	25
5.2.3	Control registers	25
5.3	Timer unit	27
5.3.1	Operation	27
5.3.2	Registers	28
5.4	UARTs.....	29
5.4.1	Transmitter operation	29
5.4.2	Receiver operation.....	30
5.4.3	Baud-rate generation	30
5.4.4	Loop back mode	30
5.4.5	Interrupt generation	31
5.4.6	UART registers.....	31
5.5	Parallel I/O port	32
5.6	LEON configuration register	33
5.7	Power-down.....	34
5.8	AHB status register	34
6	External memory access	35
6.1	Memory interface	35
6.2	Memory controller.....	35
6.3	PROM access	36
6.4	Memory mapped I/O	36
6.5	SRAM access	36
6.6	Burst cycles	37
6.7	8-bit PROM and SRAM access.....	38
6.8	8- and 16-bit I/O access.....	38
6.9	Memory EDAC	38
6.10	SDRAM access.....	39
6.10.1	General	39
6.10.2	Address mapping	39
6.10.3	Initialisation.....	39
6.10.4	Configurable SDRAM timing parameters.....	40
6.10.5	Refresh.....	40
6.10.6	SDRAM commands	40
6.10.7	Read cycles.....	40
6.10.8	Write cycles.....	40
6.10.9	Address bus connection.....	41
6.11	Memory configuration register 1 (MCFG1).....	41
6.12	Memory configuration register 2 (MCFG2).....	42
6.13	Memory configuration register 3 (MCFG3).....	43
6.14	Write protection.....	43

6.15	Using BRDYN	44
6.16	Access errors	44
6.17	Attaching an external DRAM controller	45
7	Hardware debug support	46
7.1	Overview	46
7.2	Debug support unit	46
7.2.1	Overview	46
7.2.2	Trace buffer	47
7.2.3	DSU memory map	49
7.2.4	DSU control register	50
7.2.5	DSU breakpoint registers	50
7.2.6	DSU trap register	51
7.3	DSU communication link	51
7.3.1	Operation	51
7.3.2	DSU UART control register	52
7.3.3	DSU UART status register	53
7.3.4	Baud rate generation	53
7.4	Common operations	54
7.4.1	Instruction breakpoints	54
7.4.2	Single stepping	54
7.4.3	Alternative debug sources	54
7.4.4	Booting from DSU	54
7.5	DSU monitor	54
7.6	External DSU signals	54
8	Signals	55
8.1	Memory bus signals	55
8.2	System interface signals	55
8.3	Signal description	56

1 Introduction

1.1 Overview

LEON2-FT is a 32-bit processor conforming to the IEEE P1754 (SPARC V8) architecture. It is designed for embedded applications with the following features on-chip: separate instruction and data caches, hardware multiplier and divider, interrupt controller, debug support unit with trace buffer, two 24-bit timers, two UARTs, power-down function, watchdog, 16-bit I/O port and a flexible memory controller. New modules can easily be added using the on-chip AMBA AHB/APB buses.

This manual describes the functionality of LEON2-FT version 1.0.4. The manual does not describe the PCI controller or PCI arbiter which also are implemented in the processor. The description of these two functions will be added to a later edition of this document.

Note that this document is provided as advance information, and may not accurately describe the final implementation of LEON2-FT.

1.2 Functional overview

A block diagram of LEON2 can be seen in figure 1.

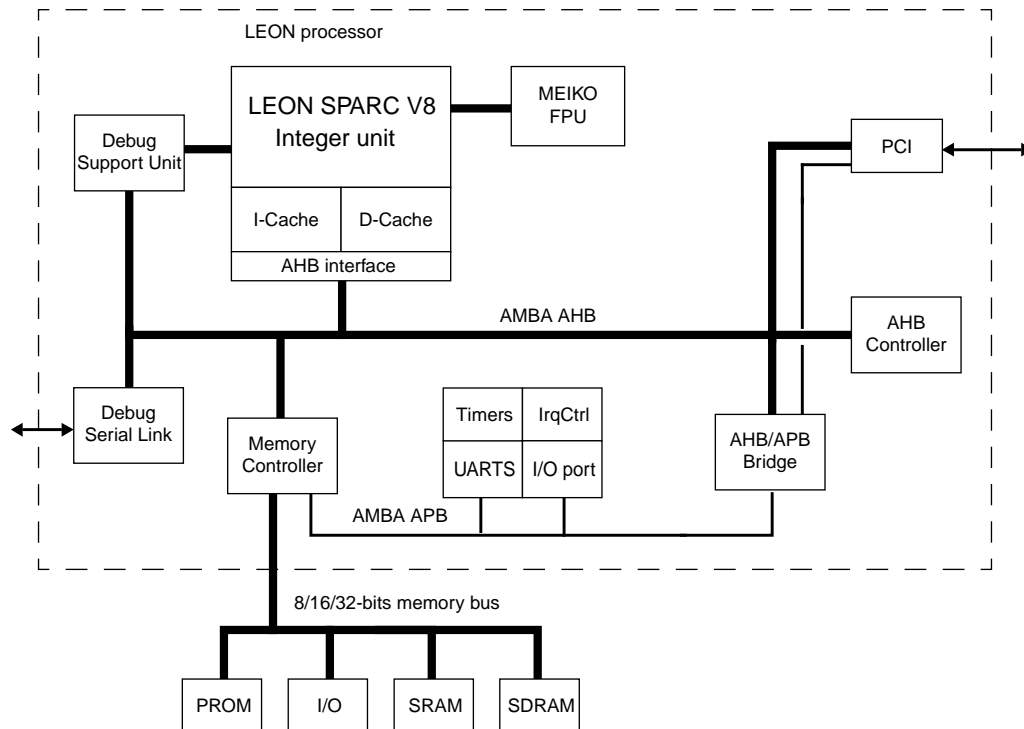


Figure 1: LEON2 block diagram

1.2.1 Integer unit

The LEON integer unit implements the full SPARC V8 standard, including all multiply and divide instructions. The integer unit implements 8 register windows.

1.2.2 Floating-point unit

All SPARC floating-point instructions are implemented, and executed through the Meiko FPU core.

1.2.3 Cache sub-system

Separate instruction and data caches are provided, each consisting of 8 kbyte, with 32 bytes per line. Sub-blocking is implemented with one valid bit per 32-bit word. The instruction cache uses streaming during line-refill to minimise refill latency. The data cache uses write-through policy and implements a double-word write-buffer. The data cache also performs bus-snooping on the AHB bus.

1.2.4 Debug support unit

The on-chip debug support unit (DSU) allows non-intrusive debugging on target hardware. The DSU allows to insert instruction and data watchpoints, and access to all on-chip registers from a remote debugger. A trace buffer is provided to trace the executed instruction flow and/or AHB bus traffic. Communication to an outside debugger (e.g. gdb) is done using a dedicated UART (RS232).

1.2.5 Memory interface

The flexible memory interface provides a direct interface PROM, memory mapped I/O devices, static RAM (SRAM) and synchronous dynamic RAM (SDRAM). The memory areas can be programmed to either 8-, 16- or 32-bit data width, and be protected using an on-chip 32/7 SEC/DED EDAC.

1.2.6 Timers

Two 24-bit timers are provided on-chip. The timers can work in periodic or one-shot mode. Both timers are clocked by a common 10-bit prescaler.

1.2.7 Watchdog

A 24-bit watchdog is provided on-chip. The watchdog is clocked by the timer prescaler. When the watchdog reaches zero, an output signal (WDOG) is asserted. This signal can be used to generate system reset.

1.2.8 UARTs

Two 8-bit UARTs are provided on-chip. The baud-rate is individually programmable and data is sent in 8-bits frames with one stop bit. Optionally, one parity bit can be generated and checked.

1.2.9 Interrupt controller

The interrupt controller manages a total of 15 interrupts, originating from internal and external sources. Each interrupt can be programmed to one of two priority levels.

1.2.10 Parallel I/O port

A 32-bit parallel I/O port is provided. 16 bits are always available and can be individually programmed by software to be an input or an output. An additional 16 bits are only available when the memory bus is configured for 8- or 16-bit operation. Some of the bits have alternate usage, such as UART inputs/outputs and external interrupts inputs.

1.2.11 Watchpoint registers

To aid software debugging, up to four watchpoint registers are provided. Each register can cause a debug-trap on an arbitrary instruction or data address range. If the debug support unit is enabled, the watchpoints can be used to enter debug mode.

1.2.12 Fault-tolerance

The LEON design includes fault-tolerance features to withstand arbitrary single-event upset (SEU) errors without loss of data. The processor register file is protected using a 32-bit SEC/DED EDAC. The cache rams are protected using 2 parity bits/word. All remaining flip-flops and latches are configured in TMR (triple-modular redundancy) mode to correct any soft errors.

1.2.13 Performance

The processor executes approximately 1 Dhrystone-2.1 MIPS/MHz.

2 LEON integer unit

The LEON integer unit (IU) implements SPARC integer instructions as defined in SPARC Architecture Manual version 8.

2.1 Overview

The LEON integer unit has the following features:

- 5-stage instruction pipeline
- Separate instruction and data cache interface
- 8 register windows
- 16x16 multiplier
- Radix-2 divider (non-restoring)

Figure 2 shows a block diagram of the integer unit.

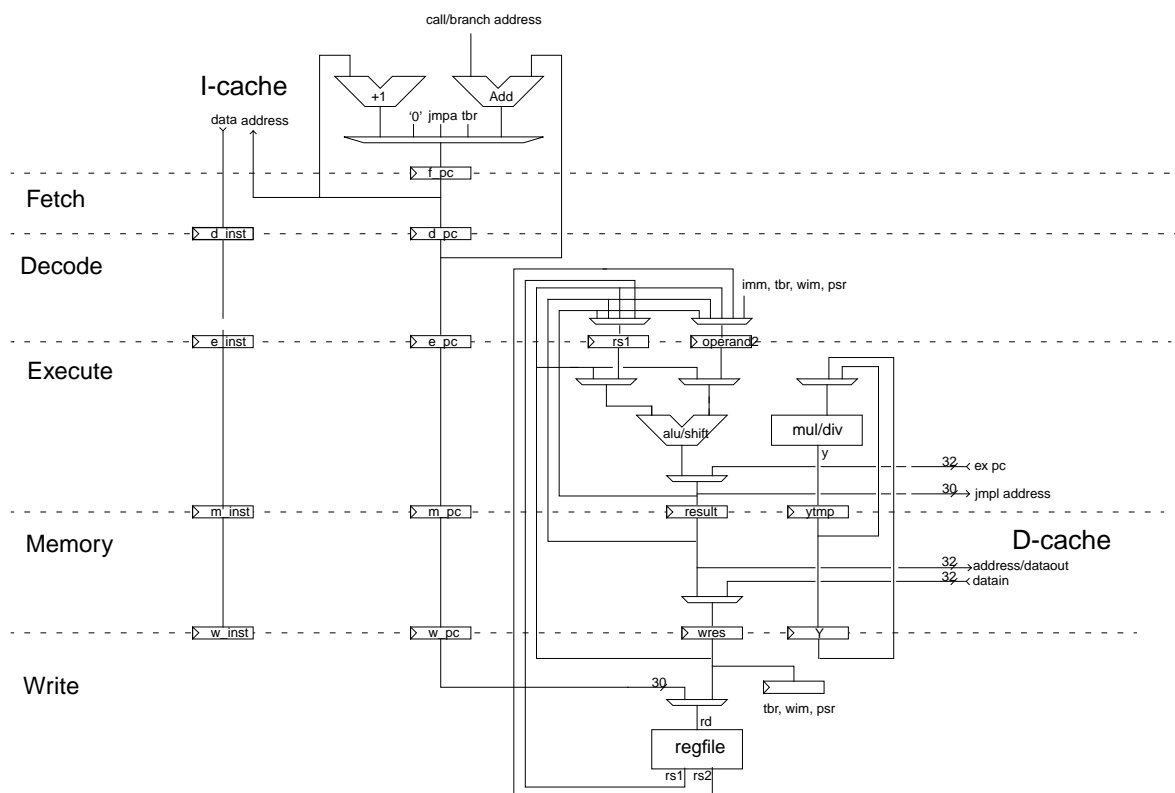


Figure 2: LEON integer unit block diagram

2.2 Instruction pipeline

The LEON integer unit uses a single instruction issue pipeline with 5 stages:

1. FE (Instruction Fetch): If the instruction cache is enabled, the instruction is fetched from the instruction cache. Otherwise, the fetch is forwarded to the memory controller. The instruction is valid at the end of this stage and is latched inside the IU.
2. DE (Decode): The instruction is decoded and the operands are read. Operands may come from the register file or from internal data bypasses. CALL and Branch target addresses are generated in this stage.
3. EX (Execute): ALU, logical, and shift operations are performed. For memory operations (e.g., LD) and for JMPL/RETT, the address is generated.
4. ME (Memory): Data cache is accessed. For cache reads, the data will be valid by the end of this stage, at which point it is aligned as appropriate. Store data read out in the E-stage is written to the data cache at this time.
5. WR (Write): The result of any ALU, logical, shift, or cache read operations are written back to the register file.

Table 1 lists the cycles per instruction (assuming cache hit and no load interlock):

Instruction	Cycles
JMPL	2
Double load	2
Single store	2
Double store	3
SMUL/UMUL	4
SDIV/UDIV	35
Taken Trap	4
Atomic load/store	3
All other instructions	1

Table 1: Instruction timing

2.3 Multiply instructions

The LEON processor supports the SPARC integer multiply instructions UMUL, SMUL, UMULCC and SMULCC. These instructions perform a 32x32-bit integer multiply, producing a 64-bit result. SMUL and SMULCC performs signed multiply while UMUL and UMULCC performs unsigned multiply. UMULCC and SMULCC also set the condition codes to reflect the result.

2.4 Divide instructions

Full support for SPARC V8 divide instructions is provided (SDIV/UDIV/SDIVCC/UDIVCC). The divide instructions perform a 64-by-32 bit divide and produce a 32-bit result. Rounding and overflow detection is performed as defined in the SPARC V8 standard.

2.5 Processor reset operation

The processor is reset by asserting the RESET input for at least one clock cycle. The following table indicates the reset values of the registers which are affected by the reset. All other registers maintain their value (or are undefined). Execution will start from address 0.

Register	Reset value
PC (program counter)	0x0
nPC (next program counter)	0x4
PSR (processor status register)	ET=0, S=1
CCR (cache control register)	0x0

Table 2: Processor reset values

2.6 Exceptions

LEON adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority.

Trap	TT	Pri	Description
reset	0x00	1	Power-on reset
write error	0x2b	2	write buffer error
instruction_access_error	0x01	3	Error during instruction fetch
illegal_instruction	0x02	5	UNIMP or other un-implemented instruction
privileged_instruction	0x03	4	Execution of privileged instruction in user mode
fp_disabled	0x04	6	FP instruction while FPU disabled
cp_disabled	0x24	6	CP instruction while Co-processor disabled
watchpoint_detected	0x0B	7	Instruction or data watchpoint match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
register_hardware_error	0x20	9	register file uncorrectable EDAC error
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
data_access_exception	0x09	13	Access error during load or store instruction
tag_overflow	0x0A	14	Tagged arithmetic overflow
divide_exception	0x2A	15	Divide by zero
interrupt_level_1	0x11	31	Asynchronous interrupt 1
interrupt_level_2	0x12	30	Asynchronous interrupt 2
interrupt_level_3	0x13	29	Asynchronous interrupt 3

Table 3: Trap allocation and priority

Trap	TT	Pri	Description
interrupt_level_4	0x14	28	Asynchronous interrupt 4
interrupt_level_5	0x15	27	Asynchronous interrupt 5
interrupt_level_6	0x16	26	Asynchronous interrupt 6
interrupt_level_7	0x17	25	Asynchronous interrupt 7
interrupt_level_8	0x18	24	Asynchronous interrupt 8
interrupt_level_9	0x19	23	Asynchronous interrupt 9
interrupt_level_10	0x1A	22	Asynchronous interrupt 10
interrupt_level_11	0x1B	21	Asynchronous interrupt 11
interrupt_level_12	0x1C	20	Asynchronous interrupt 12
interrupt_level_13	0x1D	19	Asynchronous interrupt 13
interrupt_level_14	0x1E	18	Asynchronous interrupt 14
interrupt_level_15	0x1F	17	Asynchronous interrupt 15
trap_instruction	0x80 - 0xFF	16	Software trap instruction (TA)

Table 3: Trap allocation and priority

2.7 Watch-points

The integer unit contains four hardware watch-points. Each watch-point consists of a pair of application-specific registers (%asr24/25, %asr26/27, %asr28/30 and %asr30/31) registers; one with the break address and one with a mask:

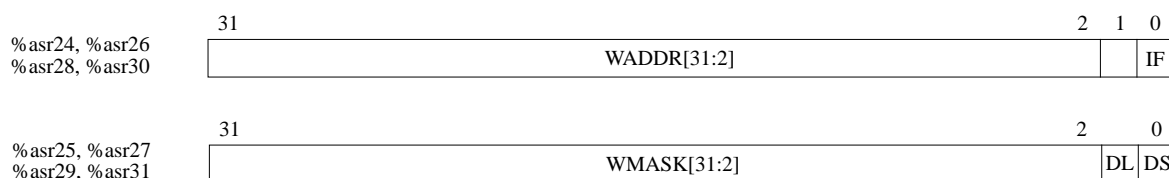


Figure 3: Watch-point registers

Any binary aligned address range can be watched - the range is defined by the WADDR field, masked by the WMASK field ($WMASK[x] = 1$ enables comparison). On a watch-point hit, trap 0x0B is generated. By setting the IF, DL and DS bits, a hit can be generated on instruction fetch, data load or data store. Clearing these three bits will effectively disable the watch-point function.

2.8 Register file EDAC protection

2.8.1 Operation

To prevent erroneous operations from SEU errors in the main register file, each word is protected with a 7-bit EDAC checksum. The EDAC checksums are checked when the register is used as operand in an instruction. Any single-bit error is corrected and written back to the

register file before the instruction is executed. If an un-correctable error is detected, a register error trap (tt=0x20) is generated.

2.8.2 Register file protection control register

The register file protection operation is controlled using application-specific register 16 (%asr16). The register is accessed using the RDASR/WRASR instructions.

31																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Figure 4: Register file protection control register (%asr16)

- [0]: DI - EDAC disable. If set, the EDAC checking function will be disabled.
- [1]: TE - Test enable.
- [8:2] TCB[6:0] - Test checkbits.
- [11:9] CNT[2:0] - Error counter. This field will be incremented for each corrected error.

The protection can be disabled by clearing the DI bit (this bit is set to '1' after reset). By setting the TE bit, errors can be inserted in the register file to test the protection function. When the TE bit is set, the register checksum is XORed with the TCB field before written to the register file. The CNT field is incremented each time a register correction is performed, but saturates at "111".

2.9 Floating-point unit

The LEON processor includes the Meiko floating-point core, thereby providing full floating-point support according to the SPARC-V8 standard. The FPU interface does not implement a floating-point queue, the processor is stopped during the execution of floating-point instructions. This means that QNE bit in the %fsr register always is zero, and any attempts of executing the STDFQ instruction will generate a FPU exception trap. The FPU registers reside in the main register file, and are protected the same EDAC scheme as described in section 2.8 above.

3 Cache sub-system

3.1 Overview

The LEON processor implements a Harvard architecture with separate instruction and data buses, connected to two independent cache controllers. In addition to the address, a SPARC processor also generates an 8-bit address space identifier (ASI), providing up to 256 separate, 32-bit address spaces. During normal operation, the LEON processor accesses instructions and data using ASI 0x8 - 0xB as defined in the SPARC standard. Using the LDA/STA instructions, alternative address spaces can be accessed. The table shows the ASI usage for LEON. Only ASI[3:0] are used for the mapping, ASI[7:4] have no influence on operation.

ASI	Usage
0x0, 0x1, 0x2, 0x3	Forced cache miss (replace if cacheable)
0x4, 0x7	Forced cache miss (update on hit)
0x5	Flush instruction cache
0x6	Flush data cache
0x8, 0x9, 0xA, 0xB	Normal cached access (replace if cacheable)
0xC	Instruction cache tags
0xD	Instruction cache data
0xE	Data cache tags
0xF	Data cache data

Table 4: ASI usage

Access to ASI 4 and 7 will force a cache miss, and update the cache if the data was previously cached. Access with ASI 0 - 3 will force a cache miss, update the cache if the data was previously cached, or allocated a new line if the data was not in the cache and the address refers to a cacheable location. The cacheable areas are by default the prom and ram areas:

Address range	Area	Cached
0x00000000 - 0x1FFFFFFF	PROM	Cacheable
0x20000000 - 0x3FFFFFFF	I/O	Non-cacheable
0x40000000 - 0x7FFFFFFF	RAM	Cacheable
0x80000000 - 0xFFFFFFFF	Internal (AHB)	Non-cacheable

Table 5: Default cache table

3.2 Instruction cache

3.2.1 Operation

The LEON instruction cache is a direct-mapped cache of 8 kbyte. The instruction cache is divided into cache lines with 32 bytes of data. Each line has a cache tag associated with it consisting of a tag field and one valid bit for each 4-byte sub-block. On an instruction cache miss to a cachable location, the instruction is fetched and the corresponding tag and data line updated.

If instruction burst fetch is enabled in the cache control register (CCR), the cache line is filled from main memory starting at the missed address and until the end of the line. At the same time, the instructions are forwarded to the IU (streaming). If the IU cannot accept the streamed instructions due to internal dependencies or multi-cycle instruction, the IU is halted until the line fill is completed. If the IU executes a control transfer instruction (branch/CALL/JMPL/RETT/TRAP) during the line fill, the line fill will be terminated on the next fetch. If instruction burst fetch is enabled, instruction streaming is enabled even when the cache is disabled. In this case, the fetched instructions are only forwarded to the IU and the cache is not updated.

If a memory access error occurs during a line fill with the IU halted, the corresponding valid bit in the cache tag will not be set. If the IU later fetches an instruction from the failed address, a cache miss will occur, triggering a new access to the failed address. If the error remains, an instruction access error trap (tt=0x1) will be generated.

3.2.2 Instruction cache flushing

The instruction cache is flushed by executing the FLUSH instruction, setting the FI bit in the cache control register, or by writing to any location with ASI=0x5. The flushing will take one cycle per cache line during which the IU will not be halted, but during which the instruction cache will be disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register.

3.2.3 Diagnostic cache access

Diagnostic reads of the tags is possible by executing an LDA instruction with ASI=0xC. Address bits making up the cache offset will be used to index the tag to be read, all other address bits are ignored. Similarly, the data sub-blocks may be read by executing an LDA with ASI=0xD. The cache offset indexes the line to be read while A[4:2] indexes which of the sub-blocks to be read.

The tags can be directly written by executing a STA with ASI=0xC. The cache offset will index the tag to be written, and D[31:13] is written into the ATAG filed (see below). The valid bits are written with the D[7:0] of the write data. The data sub-blocks can be directly written by executing a STA with ASI=0xD. The cache offset indexes the cache line and A[4:2] selects the sub-block. Note that diagnostic access to the cache is not possible during a FLUSH operation and will cause a data exception (trap=0x09) if attempted.

3.2.4 Instruction cache parity

Error detection of cache tags and data is implemented using two parity bits per tag and per 4-byte data sub-block. The tag parity is generated from the tag value and the valid bits. The data sub-block parity is derived from the sub-block data. The parity bits are written simultaneously with the associated tag or sub-block and checked on each access. The two parity bits correspond to the parity of odd and even data (tag) bits.

If a tag parity error is detected during a cache access, a cache miss will be generated and the tag (and data) will be automatically updated. All valid bits except the one corresponding to the newly loaded data will be cleared. If a data sub-block parity error occurs, a miss will also be generated but only the failed sub-block will be updated with data from main memory.

3.2.5 Instruction cache tag

A instruction cache tag entry consists of several fields as shown in figure 5:



Figure 5: Cache tag layout (read)

Field Definitions:

- [30:13]: Address Tag (ATAG) - Contains the tag address of the cache line.
- [12:10]: unused, reads as zero.
- [9]: Tag parity (IP) - During read, IP contains the (combined) tag parity.
- [8]: Data parity (DP) - During read DP contains the (combined) parity of the corresponding data sub-block.
- [7:0]: Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits is set when a sub-block is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. A FLUSH instruction will clear all valid bits. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and so on.

3.3 Data cache

3.3.1 Operation

The LEON data cache is a direct-mapped cache of 8 kbyte. The write policy for stores is write-through with no-allocate on write-miss. The data cache is divided into cache lines of 32 bytes. Each line has a cache tag associated with it, containing a tag field and one valid bit per 4-byte sub-block. On a data cache read-miss to a cachable location, 4 bytes of data are loaded into the cache from main memory. If a memory access error occurs during a data load, the corresponding valid bit in the cache tag will not be set. and a data access error trap (tt=0x9) will be generated.

3.3.2 Write buffer

The write buffer (WRB) consists of three 32-bit registers used to temporarily hold store data until it is sent to the destination device. For half-word or byte stores, the stored data replicated into proper byte alignment for writing to a word-addressed device, before being loaded into one of the WRB registers. The WRB is emptied prior to a load-miss cache-fill sequence to avoid any stale data from being read in to the data cache.

Since the processor executes in parallel with the write buffer, a write error will not cause an exception to the store instruction. Depending on memory and cache activity, the write cycle may not occur until several clock cycles after the store instructions has completed. If a write error occurs, the currently executing instruction will take trap 0x2b.

Note: the 0x2b trap handler should flush the data cache, since a write hit would update the cache while the memory would keep the old value due the write error.

3.3.3 Data cache flushing

The data cache can be flushed by executing the FLUSH instruction, setting the FD bit in the cache control register, or by writing any location with ASI=0x6. The flushing will take one cycle per line during which the IU will not be halted, but during which the data cache will be disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register.

3.3.4 Diagnostic cache access

Diagnostic software may read the tags directly by executing a single word load alternate space instructions in ASI space 0xE. The cache offset indexes the tag to be read, all other address bits are ignored. Similarly, the data sub-blocks may be read by executing a single word load alternate space instructions in ASI space 0xF. The cache offset indexes the line to be read while A[4:2] index which of the sub-blocks to be read.

The tags can be directly written by executing single word store alternate space instructions in ASI space 0xE. The cache offset indexes the tag to be written, and A[31:13] is written into the ATAG filed (see below). The valid bits are written with the D[7:0] of the write data.

The data sub-blocks can be directly written by executing single word store alternate space instructions in ASI space 0xF. Address bits The cache offset indexes the cache line and A[4:2] selects the sub-block. The sub-block is written with the write data.

Note that diagnostic access to the cache is not possible during a FLUSH operation. An attempt to perform a diagnostic access during an ongoing flush will cause a data exception trap (trap = 0x09).

3.3.5 Data cache parity

Error detection of cache tags and data is implemented using two parity bits per tag and per 4-byte data sub-block. The tag parity is generated from the tag value and the valid bits The data sub-block parity is derived from the sub-block data. The parity bits are written simultaneously with the associated tag or sub-block and checked on each access. The two parity bits correspond to the parity of odd and even data (tag) bits.

If a tag parity error is detected during a cache access, a cache miss will be generated and the tag (and data) will be automatically updated. All valid bits except the one corresponding to the newly loaded data will be cleared. If a data sub-block parity error occurs, a miss will also be generated but only the failed sub-block will be updated with data from main memory.

3.3.6 Data cache tag

A data cache tag entry consists of several fields as shown in figure 6:



Figure 6: Cache tag layout (read)

Field Definitions:

- [30:13]: Address Tag (ATAG) - Contains the tag address of the cache line.
- [12:10]: unused, reads as zero.
- [9]: Tag parity (IP) - During read, IP contains the (combined) tag parity.
- [8]: Data parity (DP) - During read DP contains the (combined) parity of the corresponding data sub-block.
- [7:0]: Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits is set when a sub-block is filled due to a successful cache miss; a cache fill which results in a memory error will leave the valid bit unset. A FLUSH instruction will clear all valid bits. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and so on.

3.4 Cache Control Register

The operation of the instruction and data caches is controlled through a common Cache Control Register (CCR) (figure 7). Each cache can be in one of three modes: disabled, enabled and frozen. If disabled, no cache operation is performed and load and store requests are passed directly to the memory controller. If enabled, the cache operates as described above. In the frozen state, the cache is accessed and kept in sync with the main memory as if it was enabled, but no new lines are allocated on read misses.

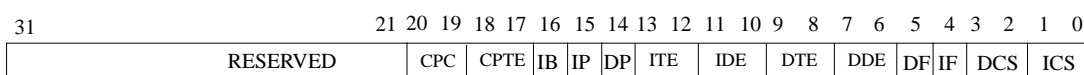


Figure 7: Cache control register

Field Definitions:

- [31:23]: Reserved
- [23] : Data cache snoop enable [DS] - if set, will enable data cache snooping.
- [22]: Flush data cache (FD). If set, will flush the instruction cache. Always reads as zero.
- [21]: Flush Instruction cache (FI). If set, will flush the instruction cache. Always reads as zero.

- [20:19]: Cache parity bits (CPC) - Indicates how many parity bits are used to protect the caches (00=none, 01=1, 10=2)
- [18:17]: Cache parity test bits. (CPTE). These bits are XOR'ed to the data and tag parity bits during diagnostic writes.
- [16]: Instruction burst fetch (IB). This bit enables burst fill during instruction fetch.
- [15]: Instruction cache flush pending (IP). This bit is set when an instruction cache flush operation is in progress.
- [14]: Data cache flush pending (DP). This bit is set when an data cache flush operation is in progress.
- [13:12]: Instruction cache tag error counter (ITE) - This field is incremented every time an instruction cache tag parity error is detected.
- [11:10]: Instruction cache data error counter (IDE) - This field is incremented each time an instruction cache data sub-block parity error is detected.
- [9:8]: Data cache tag error counter (DTE) - This field is incremented every time a data cache tag parity error is detected.
- [7:6]: Data cache data error counter (DDE) - This field is incremented each time an instruction cache data sub-block parity error is detected.
- [5]: Data Cache Freeze on Interrupt (DF) - If set, the data cache will automatically be frozen when an asynchronous interrupt is taken.
- [4]: Instruction Cache Freeze on Interrupt (IF) - If set, the instruction cache will automatically be frozen when an asynchronous interrupt is taken.
- [3:2]: Data Cache state (DCS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.
- [1:0]: Instruction Cache state (ICS) - Indicates the current data cache state according to the following: X0= disabled, 01 = frozen, 11 = enabled.

If the DF or IF bit is set, the corresponding cache will be frozen when an asynchronous interrupt is taken. This can be beneficial in real-time system to allow a more accurate calculation of worst-case execution time for a code segment. The execution of the interrupt handler will not evict any cache lines and when control is returned to the interrupted task, the cache state is identical to what it was before the interrupt.

If a cache has been frozen by an interrupt, it can only be enabled again by enabling it in the CCR. This is typically done at the end of the interrupt handler before control is returned to the interrupted task.

4 AMBA on-chip buses

4.1 Overview

Two on-chip buses are provided: AMBA AHB and APB. The APB bus is used to access on-chip registers in the peripheral functions, while the AHB bus is used for high-speed data transfers.

4.2 AHB bus

LEON uses the AMBA-2.0 AHB bus to connect the processor cache controllers to the memory controller, PCI interface and AHB/APB bridge. Table 6 below shows the address allocation.

Address range	Size	Mapping	Module
0x00000000 - 0x1FFFFFFF	512 M	Prom	Memory controller
0x20000000 - 0x3FFFFFFF	512 M	Memory bus I/O	
0x40000000 - 0x7FFFFFFF	1 G	SRAM and/or SDRAM	
0x80000000 - 0x8FFFFFFF	256 M	On-chip registers	APB bridge
0x90000000 - 0x9FFFFFFF	256 M	Debug support unit	DSU
0xA0000000 - 0xFFFFFFFF	1.5 G	PCI interface	PCI

Table 6: Default AHB address allocation

4.3 APB bus

The APB bridge is connected to the AHB bus as a slave and acts as the (only) master on the APB bus. Most on-chip peripherals are accessed through the APB bus. The address mapping of the APB bus can be seen in table 7.

4.4 AHB transfers generated by the processor

The processor is connected to the AHB bus through the instruction and data cache controllers. Access conflicts between the two cache controllers are resolved locally and only one AHB master interface is connected to the AHB bus. The processor will perform burst transfers to fetch instruction cache lines or reading/writing data as results of double load/store instructions. Byte, half-word and word load/store instructions will perform single (non-sequential accesses). Locked transfers are only performed on LDST and SWAP instructions. Double load/store transfers are however also guaranteed to be atomic since the arbiter will not re-arbitrate the bus during burst transfers.

5 On-chip peripherals

5.1 On-chip registers

A number of system support functions are provided directly on-chip. The functions are controlled through registers mapped APB bus according to the following table:

Address	Register	Address	
0x80000000	Memory configuration register 1	0x800000A0	I/O port input/output register
0x80000004	Memory configuration register 2	0x800000A4	I/O port direction register
0x80000008	Memory configuration register 3	0x800000A8	I/O port interrupt register
0x8000000C	AHB Failing address register		
0x80000010	AHB status register		
0x80000014	Cache control register	0x800000C4	DSU UART status register
0x80000018	Power-down register	0x800000C8	DSU UART control register
0x8000001C	Write protection register 1	0x800000CC	DSU UART scaler register
0x80000020	Write protection register 2		
0x80000024	LEON configuration register		
0x80000040	Timer 1 counter register		
0x80000044	Timer 1 reload register		
0x80000048	Timer 1 control register		
0x8000004C	Watchdog register		
0x80000050	Timer 2 counter register		
0x80000054	Timer 2 reload register		
0x80000058	Timer 2 control register		
0x80000060	Scaler counter register		
0x80000064	Scaler reload register		
0x80000070	Uart 1 data register		
0x80000074	Uart 1 status register		
0x80000078	Uart 1 control register		
0x8000007C	Uart 1 scaler register		
0x80000080	Uart 2 data register		
0x80000084	Uart 2 status register		
0x80000088	Uart 2 control register		
0x8000008C	Uart 2 scaler register		
0x80000090	Interrupt mask and priority register		
0x80000094	Interrupt pending register		
0x80000098	Interrupt force register		
0x8000009C	Interrupt clear register		

Table 7: On-chip registers

5.2 Interrupt controller

The LEON interrupt controller is used to prioritize and propagate interrupt requests from internal or external devices to the integer unit. In total 11 interrupts are handled, divided on two priority levels. Figure 8 shows a block diagram of the interrupt controller.

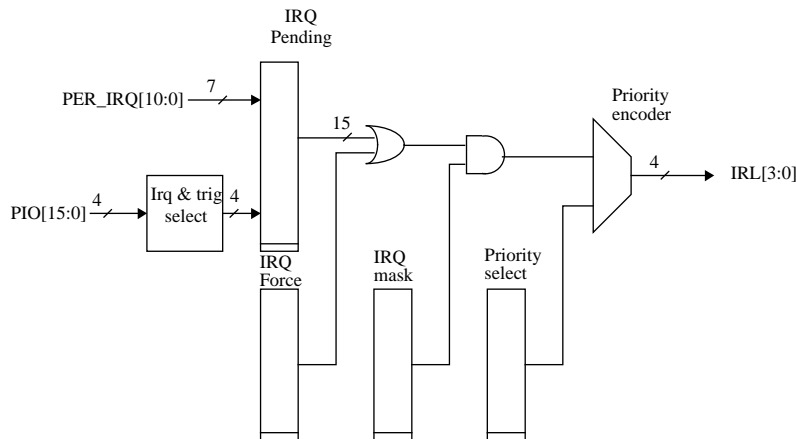


Figure 8: Interrupt controller block diagram

5.2.1 Operation

When an interrupt is generated, the corresponding bit is set in the interrupt pending register. The pending bits are ANDed with the interrupt mask register and then forwarded to the priority selector. Each interrupt can be assigned to one of two levels as programmed in the interrupt level register. Level 1 has higher priority than level 0. The interrupts are prioritised within each level, with interrupt 15 having the highest priority and interrupt 1 the lowest. The highest interrupt from level 1 will be forwarded to the IU - if no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded. When the IU acknowledges the interrupt, the corresponding pending bit will automatically be cleared.

Interrupt can also be forced by setting a bit in the interrupt force register. In this case, the IU acknowledgement will clear the force bit rather than the pending bit.

After reset, the interrupt mask register is set to all zeros while the remaining control registers are undefined.

Interrupts 10, 11 - 12, and 15 are unused.

5.2.2 Interrupt assignment

Table 8 shows the assignment of interrupts.

Interrupt	Source
15	unused
14	PCI
13	unused
12	unused
11	DSU trace buffer
10	unused
9	Timer 2
8	Timer 1
7	Parallel I/O[3]
6	Parallel I/O[2]
5	Parallel I/O[1]
4	Parallel I/O[0]
3	UART 1
2	UART 2
1	AHB error

Table 8: Interrupt assignments

5.2.3 Control registers

The operation of the interrupt controller is programmed through the following registers:

31	17	16	15	1	0
ILEVEL[15:1]				R	IMASK[15:1]
				R	

Figure 9: Interrupt mask and priority register

Field Definitions:

- [31:17]: Interrupt level (ILEVEL[15:1]) - indicates whether an interrupt belongs to priority level 1 (ILEVEL[n]=1) or level 0 (ILEVEL[n]=0).
- [15:1]: Interrupt mask (IMASK[15:0]) - indicates whether an interrupt is masked (IMASK[n]=0) or enabled (IMASK[n]=1).
- [16], [0]: Reserved

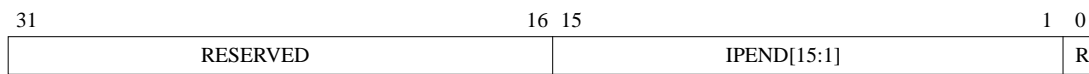


Figure 10: Interrupt pending register

Field Definitions:

- [15:1]: Interrupt pending (IPEND[15:1]) - indicates whether an interrupt is pending (IPEND[n]=1).
- [31:16], [0]: Reserved

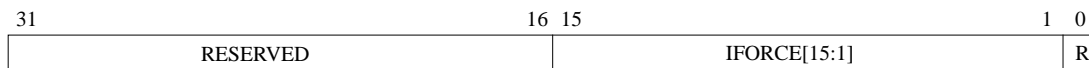


Figure 11: Interrupt force register

Field Definitions:

- [15:1]: Interrupt force (IFORCE[15:1]) - indicates whether an interrupt is being forced (IFORCE[n]=1).
- [31:16], [0]: Reserved

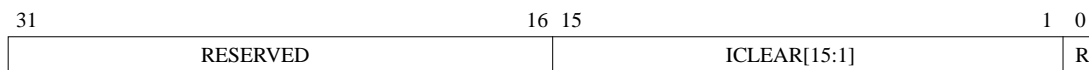


Figure 12: Interrupt clear register

Field Definitions:

- [15:1]: Interrupt clear (ICLEAR[15:1]) - if written with a '1', will clear the corresponding bit(s) in the interrupt pending register. A read returns zero.
- [31:16], [0]: Reserved

5.3 Timer unit

The timer unit implements two 24-bit timers, one 24-bit watchdog and one 10-bit shared prescaler (figure 13).

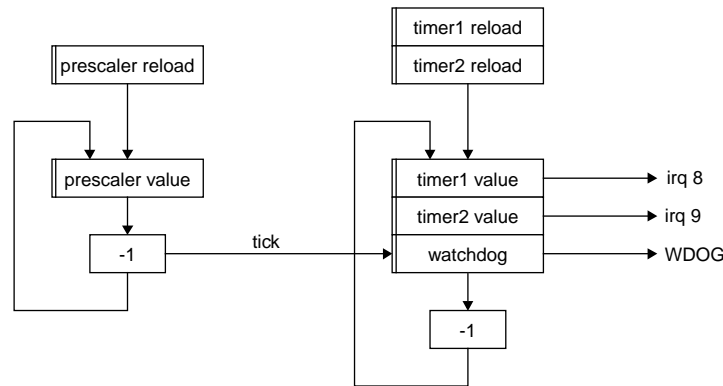


Figure 13: Timer unit block diagram

5.3.1 Operation

The prescaler is clocked by the system clock and decremented on each clock cycle. When the prescaler underflows, it is reloaded from the prescaler reload register and a timer tick is generated for the two timers and watchdog. The effective division rate is therefore equal to prescaler reload register value + 1.

The operation of the timers is controlled through the timer control register. A timer is enabled by setting the enable bit in the control register. The timer value is then decremented each time the prescaler generates a timer tick. When a timer underflows, it will automatically be reloaded with the value of the timer reload register if the reload bit is set, otherwise it will stop (at 0xfffff) and reset the enable bit. An interrupt will be generated after each underflow.

The timer can be reloaded with the value in the reload register at any time by writing a 'one' to the load bit in the control register.

The watchdog operates similar to the timers, with the difference that it is always enabled and upon underflow asserts the external signal WDOG. This signal can be used to generate a system reset.

To minimise complexity, the two timers and watchdog share the same decrements. This means that the minimum allowed prescaler division factor is 4 (reload register = 3).

5.3.2 Registers

Figures 14 to 18 shows the layout of the timer unit registers.



Figure 14: Timer 1/2 and Watchdog counter registers

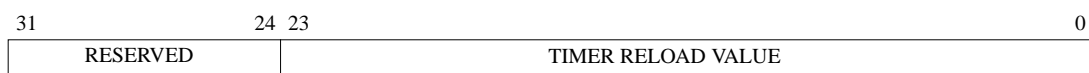


Figure 15: Timer 1/2 reload registers

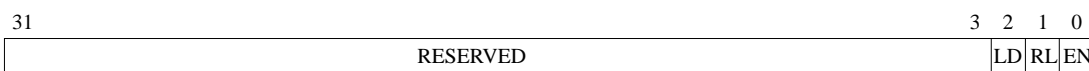


Figure 16: Timer 1/2 control registers

- [2]: Load counter (LD) - when written with 'one', will load the timer reload register into the timer counter register. Always reads as a 'zero'.
- [1]: Reload counter (RL) - if RL is set, then the counter will automatically be reloaded with the reload value after each underflow.
- [0]: Enable (EN) - enables the timer when set.



Figure 17: Prescaler reload register



Figure 18: Prescaler counter register

5.4 UARTs

Two identical UARTs are provided for serial communications. The UARTs support data frames with 8 data bits, one optional parity bit and one stop bit. To generate the bit-rate, each UART has a programmable 12-bits clock divider. Hardware flow-control is supported through the RTSN/CTSN hand-shake signals. Figure 19 shows a block diagram of a UART.

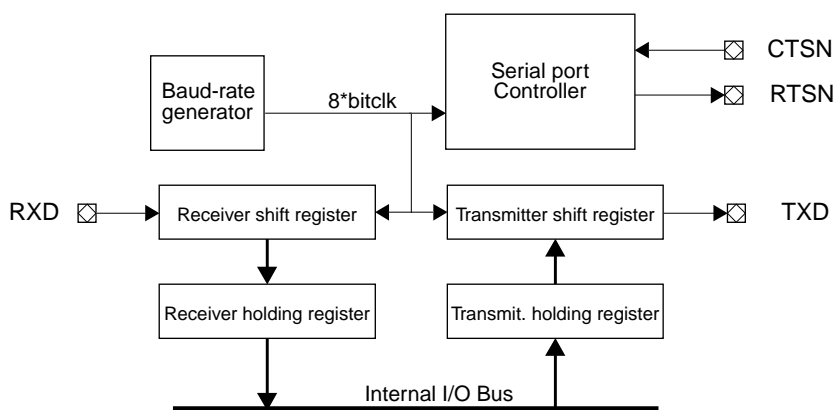


Figure 19: UART block diagram

5.4.1 Transmitter operation

The transmitter is enabled through the TE bit in the UART control register. When ready to transmit, data is transferred from the transmitter holding register to the transmitter shift register and converted to a serial stream on the transmitter serial output pin (TXD). It automatically sends a start bit followed by eight data bits, an optional parity bit, and one stop bits (figure 20). The least significant bit of the data is sent first

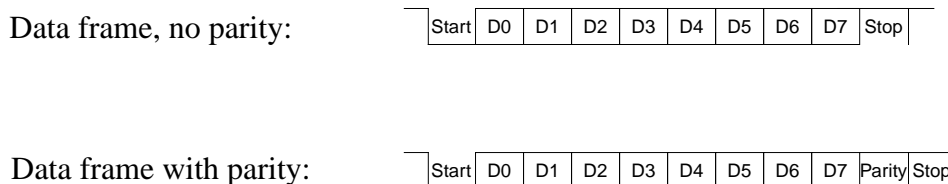


Figure 20: UART data frames

Following the transmission of the stop bit, if a new character is not available in the transmitter holding register, the transmitter serial data output remains high and the transmitter shift register empty bit (TSRE) will be set in the UART control register. Transmission resumes and the TSRE is cleared when a new character is loaded in the transmitter holding register. If the

transmitter is disabled, it will continue operating until the character currently being transmitted is completely sent out. The transmitter holding register cannot be loaded when the transmitter is disabled.

If flow control is enabled, the CTSN input must be low in order for the character to be transmitted. If it is deasserted in the middle of a transmission, the character in the shift register is transmitted and the transmitter serial output then remains inactive until CTSN is asserted again. If the CTSN is connected to a receiver's RTSN, overrun can effectively be prevented.

5.4.2 Receiver operation

The receiver is enabled for data reception through the receiver enable (RE) bit in the USART control register. The receiver looks for a high to low transition of a start bit on the receiver serial data input pin. If a transition is detected, the state of the serial input is sampled a half bit clocks later. If the serial input is sampled high the start bit is invalid and the search for a valid start bit continues. If the serial input is still low, a valid start bit is assumed and the receiver continues to sample the serial input at one bit time intervals (at the theoretical centre of the bit) until the proper number of data bits and the parity bit have been assembled and one stop bit has been detected. The serial input is sampled three times for each bit and averaged to filter out noise.

During this process the least significant bit is received first. The data is then transferred to the receiver holding register (RHR) and the data ready (DR) bit is set in the USART status register. The parity, framing and overrun error bits are set at the received byte boundary, at the same time as the receiver ready bit is set.

If both receiver holding and shift registers contain an un-read character when a new start bit is detected, then the character held in the receiver shift register will be lost and the overrun bit will be set in the UART status register. If flow control is enabled, then the RTSN will be negated (high) when a valid start bit is detected and the receiver holding register contains an un-read character. When the holding register is read, the RTSN will automatically be reasserted again.

5.4.3 Baud-rate generation

Each UART contains a 12-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock and generates a UART tick each time it underflows. The scaler is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate. If the EC bit is set, the scaler will be clocked by the PIO[3] input rather than the system clock. In this case, the frequency of PIO[3] must be less than half the frequency of the system clock.

5.4.4 Loop back mode

If the LB bit in the UART control register is set, the UART will be in loop back mode. In this mode, the transmitter output is internally connected to the receiver input and the RTSN is connected to the CTSN. It is then possible to perform loop back tests to verify operation of receiver, transmitter and associated software routines. In this mode, the outputs remain in the inactive state, in order to avoid sending out data.

5.4.5 Interrupt generation

The UART will generate an interrupt under the following conditions: when the transmitter is enabled, the transmitter interrupt is enabled and the transmitter holding register moves from full to empty; when the receiver is enabled, the receiver interrupt is enabled and the receiver holding register moves from empty to full; when the receiver is enabled, the receiver interrupt is enabled and a character with either parity, framing or overrun error is received.

5.4.6 UART registers

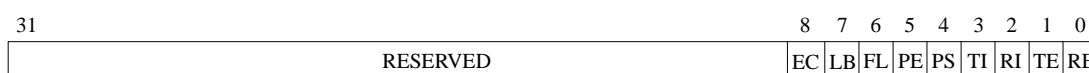


Figure 21: UART control register

- 0: Receiver enable (RE) - if set, enables the receiver.
- 1: Transmitter enable (TE) - if set, enables the transmitter.
- 2: Receiver interrupt enable (RI) - if set, enables generation of receiver interrupt.
- 3: Transmitter interrupt enable (TI) - if set, enables generation of transmitter interrupt.
- 4: Parity select (PS) - selects parity polarity (0 = even parity, 1 = odd parity)
- 5: Parity enable (PE) - if set, enables parity generation and checking.
- 6: Flow control (FL) - if set, enables flow control using CTS/RTS.
- 7: Loop back (LB) - if set, loop back mode will be enabled.
- 8: External Clock - if set, the UART scaler will be clocked by PIO[3]



Figure 22: UART status register

- 0: Data ready (DR) - indicates that new data is available in the receiver holding register.
- 1: Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty.
- 2: Transmitter hold register empty (TH) - indicates that the transmitter hold register is empty.
- 3: Break received (BR) - indicates that a BREAK has been received.
- 4: Overrun (OV) - indicates that one or more character have been lost due to overrun.
- 5: Parity error (PE) - indicates that a parity error was detected.
- 6: Framing error (FE) - indicates that a framing error was detected.



Figure 23: UART scaler reload register

5.5 Parallel I/O port

A partially bit-wise programmable 32-bit I/O port is provided on-chip. The port is split in two parts - the lower 16-bits are accessible via the PIO[15:0] signal while the upper 16-bits uses D[15:0] and can only be used when all areas (rom, ram and I/O) of the memory bus are in 8- or 16-bit mode (see “8-bit PROM and SRAM access” on page 38). If the SDRAM controller is enabled, the upper 16-bits cannot be used.

The lower 16 bits of the I/O port can be individually programmed as output or input, while the high 16 bits of the I/O port only be configured as outputs or inputs on byte basis. Two registers are associated with the operation of the I/O port; the combined I/O input/output register, and I/O direction register. When read, the input/output register will return the current value of the I/O port; when written, the value will be driven on the port signals (if enabled as output). The direction register defines the direction for each individual port bit (0=input, 1=output).

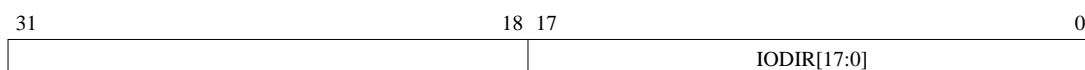


Figure 24: I/O port direction register

- $IODIR_n$ - I/O port direction. The value of $IODIR[15:0]$ defines the direction of I/O ports 15 - 0. If bit n is set the corresponding I/O port becomes an output, otherwise it is an input. $IODIR[16]$ controls D[15:8] while $IODIR[17]$ controls D[7:0]

The I/O ports can also be used as interrupt inputs from external devices. A total of four interrupts can be generated, corresponding to interrupt levels 4, 5, 6 and 7. The I/O port interrupt configuration register (figure 25) defines which port should generate each interrupt and how it should be filtered.

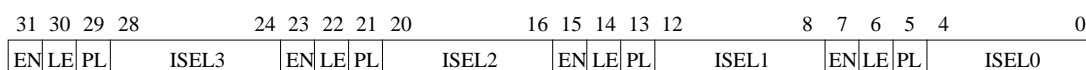


Figure 25: I/O port interrupt configuration register

- $ISEL_n$ - I/O port select. The value of this field defines which I/O port (0 - 31) should generate parallel I/O port interrupt n .
- PL - Polarity. If set, the corresponding interrupt will be active high (or edge-triggered on positive edge). Otherwise, it will be active low (or edge-triggered on negative edge).
- LE - Level/edge triggered. If set, the interrupt will be edge-triggered, otherwise level sensitive.
- EN - Enable. If set, the corresponding interrupt will be enabled, otherwise it will be masked.

To save pins, I/O pins are shared with other functions according to the table below:

I/O port	Function	Type	Description	Enabling condition
PIO[15]	TXD1	Output	UART1 transmitter data	UART1 transmitter enabled
PIO[14]	RXD1	Input	UART1 receiver data	-
PIO[13]	RTS1	Output	UART1 request-to-send	UART1 flow-control enabled
PIO[12]	CTS1	Input	UART1 clear-to-send	-
PIO[11]	TXD2	Output	UART2 transmitter data	UART2 transmitter enabled
PIO[10]	RXD2	Input	UART2 receiver data	-
PIO[9]	RTS2	Output	UART2 request-to-send	UART2 flow-control enabled
PIO[8]	CTS2	Input	UART2 clear-to-send	-
PIO[3]	UART clock	Input	Use as alternative UART clock	-
PIO[2]	EDAC enable	Input	Enable EDAC checking at reset	-
PIO[1:0]	Prom width	Input	Defines prom width at reset	-

Table 9: UART/IO port usage

5.6 LEON configuration register

Since LEON is synthesised from a extensively configurable VHDL model, the LEON configuration register (read-only) is used to indicate which options were enabled during synthesis. Figure 26 shows the layout of the register.

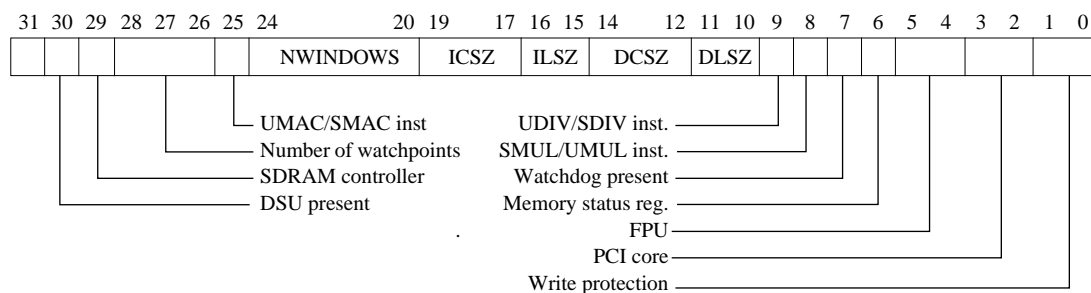


Figure 26: LEON configuration register

- [30]: Debug support unit (1=present)
- [29]: SDRAM controller present (1=present)
- [28:26]: Number of implemented watchpoints (4)
- [25]: UMAC/SMAC instruction implemented (0)
- [24:20]: Number of register windows. The implemented number of SPARC register windows -1. (7)
- [19:17]: Instruction cache size. The size (in Kbytes) of the instruction cache. Cache size = 2^{ICSZ} . (4)
- [16:15]: Instruction cache line size. The line size in 32-bit words of each line. Line size = 2^{ILSZ} . (3)
- [14:12]: Data cache size. The size (in kbytes) of the data cache. Cache size = 2^{DCSZ} . (4)
- [11:10]: Data cache line size. The line size (in 32-bit words) of each line. Line size = 2^{DLSZ} . (3)
- [9]: UDIV/SDIV instruction implemented (1)
- [8]: UMUL/SMUL instruction implemented (1)
- [6]: Memory status and failing address register present (1)
- [5:4]: FPU type (01=Meiko)
- [3:2]: PCI core type (01=InSilicon)
- [1:0]: Write protection type (01=standard)

5.7 Power-down

The processor can be powered-down by writing (an arbitrary) value to the power-down register. Power-down mode will be entered on the next load or store instruction. To enter power-down mode immediately, two consecutive stores to the power-down register should be performed. During power-down mode, the integer unit will effectively be halted. The power-down mode will be terminated (and the integer unit re-enabled) when an unmasked interrupt with higher level than the current processor interrupt level (PIL) becomes pending. All other functions and peripherals operate as nominal during the power-down mode.

5.8 AHB status register

Any access triggering an error response on the AHB bus will be registered in two registers; AHB failing address register and AHB status register. The failing address register will store the address of the access while the memory status register will store the access and error types. The registers are updated when an error occur, and the NE (new error) is set. When the NE bit is set, interrupt 1 is generated to inform the processor about the error. After an error, the NE bit has to be reset by software.

Figure 27 shows the layout of the AHB status register.



Figure 27: AHB status register

- [9]: CE - Correctable EDAC error detected.
- [8]: NE - New error. Set when a new error occurred.
- [7]: RW - Read/Write. This bit is set if the failed access was a read cycle, otherwise it is cleared.
- [6:3]: HMASTER - AHB master. This field contains the HMASTER[3:0] of the failed access.
- [2:0] HSIZE - transfer size. This field contains the HSIZE[2:0] of the failed transfer.

6 External memory access

6.1 Memory interface

The memory bus provides a direct interface to PROM, memory mapped I/O devices, asynchronous static ram (SRAM) and synchronous dynamic ram (SDRAM). Chip-select decoding is done for two PROM banks, one I/O bank, five SRAM banks and two SDRAM banks. Figure 28 shows how the connection to the different device types is made.

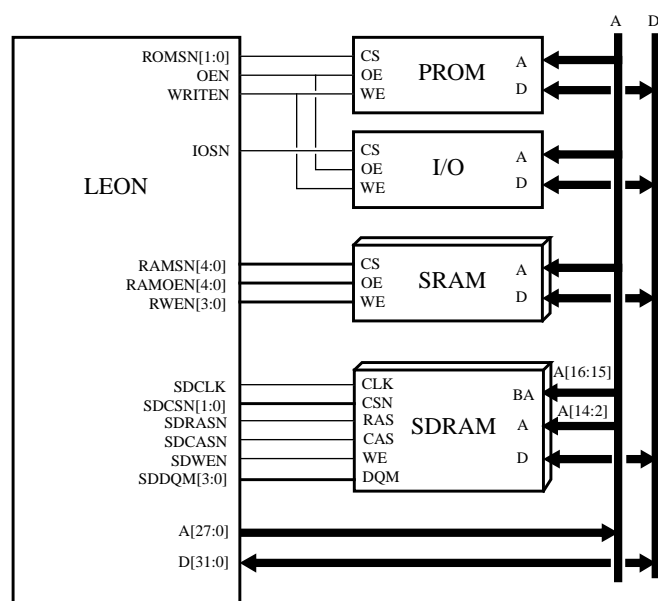


Figure 28: Memory device interface

6.2 Memory controller

The external memory bus is controlled by a programmable memory controller. The controller acts as a slave on the AHB bus. The function of the memory controller is programmed through memory configuration registers 1, 2 & 3 (MCFG1, MCFG2 & MCFG3) through the APB bus. The memory bus supports four types of devices: prom, sram, sdram and local I/O. The memory bus can also be configured in 8-bit mode for applications with low memory and performance demands. The controller decodes a 2 Gbyte address space, divided according to table 10:

Address range	Size	Mapping
0x00000000 - 0x1FFFFFFF	512 M	Prom
0x20000000 - 0x3FFFFFFF	512M	I/O
0x40000000 -0x7FFFFFFF	1 G	SRAM/SDRAM

Table 10: Memory controller address map

6.3 PROM access

Accesses to prom have the same timing as RAM accesses, the differences being that PROM cycles can have up to 15 waitstates.

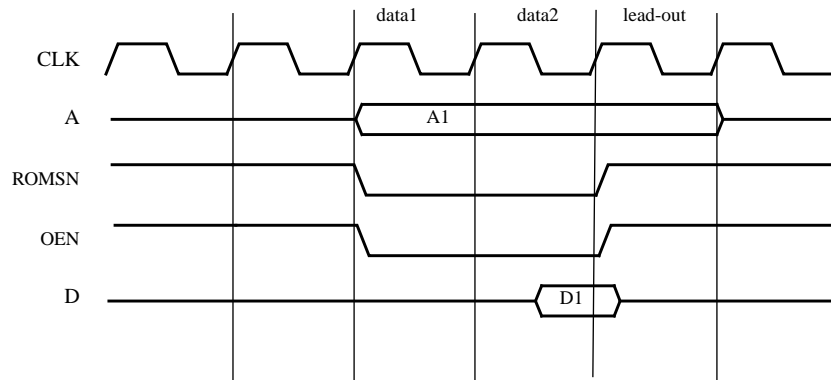


Figure 29: Prom read cycle

Two PROM chip-select signals are provided, ROMSN[1:0]. ROMSN[0] is asserted when the lower half (0 - 0x10000000) of the PROM area as addressed while ROMSN[1] is asserted for the upper half (0x10000000 - 0x20000000).

6.4 Memory mapped I/O

Accesses to I/O have similar timing to ROM/RAM accesses, the differences being that a additional waitstates can be inserted by de-asserting the BRDYN signal. The I/O select signal (IOSN) is delayed one clock to provide stable address before IOSN is asserted.

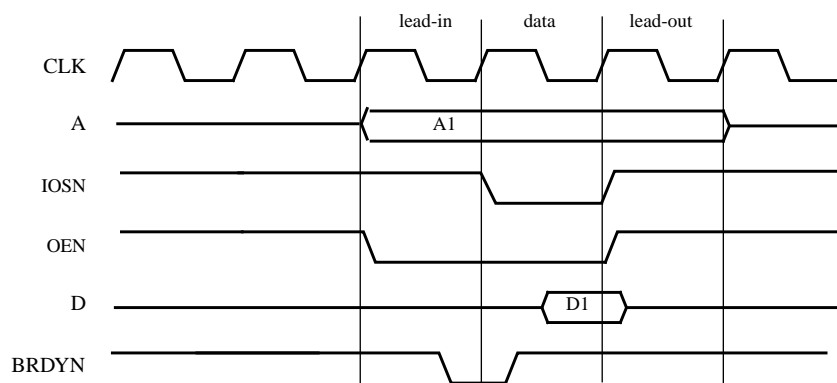


Figure 30: I/O read cycle

6.5 SRAM access

The SRAM area can be up to 1 Gbyte, divided on up to five RAM banks. The size of banks 1-4 (RAMSN[3:0]) is programmed in the RAM bank-size field (MCFG2[12:9]) and can be set in binary steps from 8 Kbyte to 256 Mbyte. The fifth bank (RAMSN[4]) decodes the upper

512 Mbyte. A read access to SRAM consists of two data cycles and between zero and three waitstates. Accesses to RAMSN[4] can further be stretched by de-asserting BRDYN until the data is available. On non-consecutive accesses, a lead-out cycle is added after a read cycle to prevent bus contention due to slow turn-off time of memories or I/O devices. Figure 31 shows the basic read cycle waveform (zero waitstate).

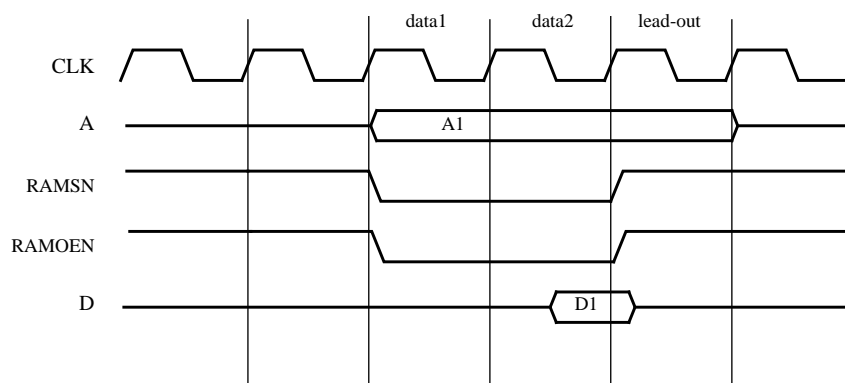


Figure 31: Static ram read cycle (0-waitstate)

For read accesses to RAMSN[4:0], a separate output enable signal (RAMOEN[n]) is provided for each RAM bank and only asserted when that bank is selected. A write access is similar to the read access but has takes a minimum of three cycles:

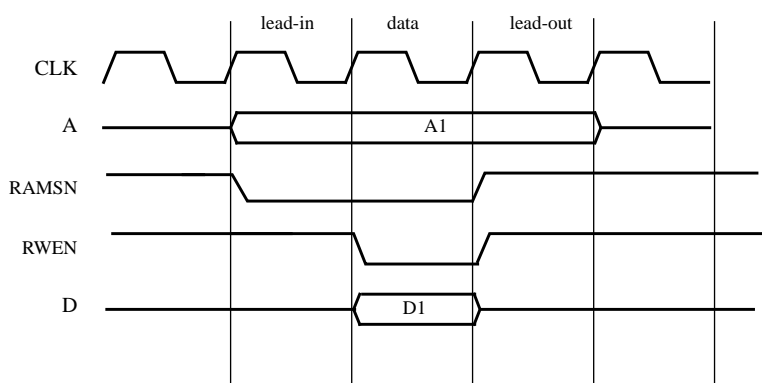


Figure 32: Static ram write cycle

Through an (optional) feed-back loop from the write strobes, the data bus is guaranteed to be driven until the write strobes are de-asserted. Each byte lane has an individual write strobe to allow efficient byte and half-word writes. If you memory used a common write strobe for the full 16- or 32-bit data, set the read-modify-write bit MCFG2 which will enable read-modify-write cycles for sub-word writes.

6.6 Burst cycles

To improve the bandwidth of the memory bus, accesses to consecutive addresses can be performed in burst mode. Burst transfers will be generated when the memory controller is accessed using an AHB burst request. These includes instruction cache-line fills, double

loads and double stores. The timing of a burst cycle is identical to the programmed basic cycle with the exception that during read cycles, the lead-out cycle will only occurs after the last transfer.

6.7 8-bit PROM and SRAM access

To support applications with low memory and performance requirements efficiently, it is not necessary to always have full 32-bit memory banks. The SRAM and PROM areas can be individually configured for 8- operation by programming the ROM and RAM size fields in the memory configuration registers. Since access to memory is always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read cycles. During writes, only the necessary bytes will be written. Figure 33 shows an interface example with 8-bit PROM and 8-bit SRAM.

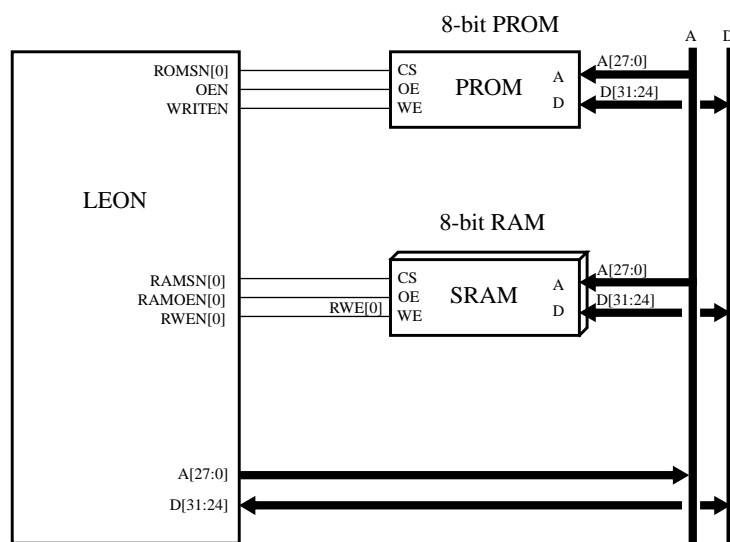


Figure 33: 8-bit memory interface example

6.8 8- and 16-bit I/O access

Similar to the PROM/RAM areas, the I/O area can also be configured to 8-bits mode. However, the I/O device will NOT be accessed by multiple 8 bits accesses as the memory areas, but only with one single access just as in 32-bit mode. To accesses an I/O device on a 8-bit bus, LDUB/STB should be used.

6.9 Memory EDAC

The on-chip memory EDAC can correct one error and detect two errors in a 32-bit word. For each word, a 7-bit checksum is generated according to the equations below. Correction is done on-the-fly and no timing penalty occurs during correction. If an un-correctable error (double-error) is detected, an error response is generated on the AHB bus, which will lead to

a memory exception in the processor. If a correctable error occurs, no error response is generated but the event is registered in the failing address and memory status register, and interrupt 1 is generated. The interrupt can then be attached to a low priority interrupt handler that scrubs the failing memory location. The EDAC can be used during access to PROM or RAM areas by setting the corresponding EDAC enable bits in the MCFG3 (see below). The equations below show how the EDAC checkbits are generated:

$$\begin{aligned} CB0 &= D0 \wedge D4 \wedge D6 \wedge D7 \wedge D8 \wedge D9 \wedge D11 \wedge D14 \wedge D17 \wedge D18 \wedge D19 \wedge D21 \wedge D26 \wedge D28 \wedge D29 \wedge D31 \\ CB1 &= D0 \wedge D1 \wedge D2 \wedge D4 \wedge D6 \wedge D8 \wedge D10 \wedge D12 \wedge D16 \wedge D17 \wedge D18 \wedge D20 \wedge D22 \wedge D24 \wedge D26 \wedge D28 \\ \overline{CB2} &= D0 \wedge D3 \wedge D4 \wedge D7 \wedge D9 \wedge D10 \wedge D13 \wedge D15 \wedge D16 \wedge D19 \wedge D20 \wedge D23 \wedge D25 \wedge D26 \wedge D29 \wedge D31 \\ \overline{CB3} &= D0 \wedge D1 \wedge D5 \wedge D6 \wedge D7 \wedge D11 \wedge D12 \wedge D13 \wedge D16 \wedge D17 \wedge D21 \wedge D22 \wedge D23 \wedge D27 \wedge D28 \wedge D29 \\ CB4 &= D2 \wedge D3 \wedge D4 \wedge D5 \wedge D6 \wedge D7 \wedge D14 \wedge D15 \wedge D18 \wedge D19 \wedge D20 \wedge D21 \wedge D22 \wedge D23 \wedge D30 \wedge D31 \\ CB5 &= D8 \wedge D9 \wedge D10 \wedge D11 \wedge D12 \wedge D13 \wedge D14 \wedge D15 \wedge D24 \wedge D25 \wedge D26 \wedge D27 \wedge D28 \wedge D29 \wedge D30 \wedge D31 \\ CB6 &= D0 \wedge D1 \wedge D2 \wedge D3 \wedge D4 \wedge D5 \wedge D6 \wedge D7 \wedge D24 \wedge D25 \wedge D26 \wedge D27 \wedge D28 \wedge D29 \wedge D30 \wedge D31 \end{aligned}$$

If the memory is configured in 8-bit mode, the EDAC checkbit bus (CB[7:0]) is not used but it is still possible to use EDAC protection. This is done by allocating the top 25% of the memory bank to the EDAC checksums. If the EDAC is enabled, a read access will read the data bytes from the nominal address, and the EDAC checksum from the top part of the bank. A write cycle is performed the same way. In this way, 75% of the bank memory is available as program or data memory, 18.75% is used for checkbits and the top 6.25% is unused. The size of the memory bank is determined from the settings in MCFG1&2.

The operation of the EDAC can be tested through the MCFG3. If the WB (write bypass) bit is set, the value in the TCB field will replace the normal checkbits during memory write cycles. If the RB (read bypass) is set, the memory checkbits of the loaded data will be stored in the TCB field during memory read cycles. **NOTE:** when the EDAC is enabled, the RMW bit in MCFG2 must be set.

6.10 SDRAM access

6.10.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. The controller supports 64M, 256M and 512M device with 8 - 12 column-address bits, up to 13 row-address bits, and 4 banks. The size of each of the two banks can be programmed in binary steps between 4 Mbyte and 512 Mbyte. The operation of the SDRAM controller is controlled through MCFG2 and MCFG3 (see below). Note that only 32-bit data bus width is supported for SDRAM banks.

6.10.2 Address mapping

The two SDRAM banks can be mapped starting at address 0x40000000 or 0x60000000. When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped at 0x60000000 as long as the SRAM disable bit is not set. If the SRAM disable bit is set, all access to SRAM is disabled and the SDRAM banks are mapped starting at 0x40000000.

6.10.3 Initialisation

After reset, the controller automatically performs the SDRAM initialisation sequence of PRECHARGE, 2x AUTO-REFRESH and LOAD-MODE-REG on both banks

simultaneously. The controller programs the SDRAM to use page burst on read and single location access on write. A CAS latency of 3 is programmed by default, but can be changed later by software issuing additional LOAD-MODE-REG commands.

6.10.4 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), some SDRAM parameters can be programmed through memory configuration register 2 (MCFG2). The programmable SDRAM parameters can be seen in table 11:

Function	Parameter	range	unit
CAS latency		2 - 3	clocks
Precharge to activate	t_{RP}	2 - 3	clocks
Auto-refresh command period	t_{RFC}	3 - 11	clocks
Auto-refresh interval		10 - 32768	clocks

Table 11: SDRAM programmable timing parameters

Remaining SDRAM timing parameters are according the PC100/PC133 specification.

6.10.5 Refresh

The SDRAM controller contains a refresh function that periodically issues an AUTO-REFRESH command to both SDRAM banks. The period between the commands (in clock periods) is programmed in the refresh counter reload field in the MCFG3 register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 μ s (corresponding to 780 or 1560 clocks at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by setting bit 31 in MCFG2.

6.10.6 SDRAM commands

The controller can issue three SDRAM commands by writing to the SDRAM command field in MCFG2: PRE-CHARGE, AUTO-REFRESH and LOAD-MODE-REG (LMR). If the LMR command is issued, the CAS delay as programmed in MCFG2 will be used, remaining fields are fixed: page read burst, single location write, sequential burst. The command field will be cleared after a command has been executed. Note that when changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time.

6.10.7 Read cycles

A read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command after the programmed CAS delay. A read burst is performed if a burst access has been requested on the AHB bus. The read cycle is terminated with a PRE-CHARGE command, no banks are left open between two accesses.

6.10.8 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. A write burst on the AHB bus will generate a burst of write commands without idle cycles in-between.

6.10.9 Address bus connection

The address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. Devices with less than 13 address pins should leave the MSB part of A[14:2] unconnected.

6.11 Memory configuration register 1 (MCFG1)

Memory configuration register 1 is used to program the timing of rom and local I/O accesses.

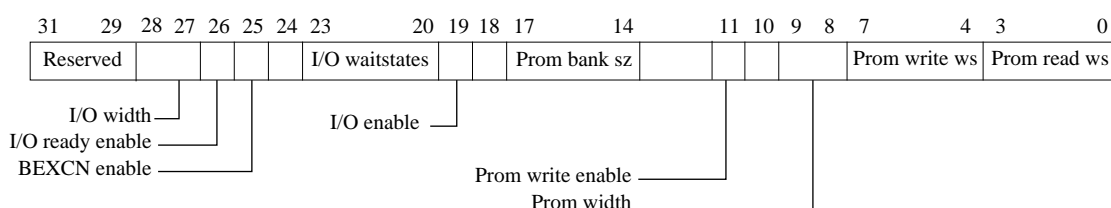


Figure 34: Memory configuration register 1

- [3:0]: Prom read waitstates. Defines the number of waitstates during prom read cycles (“0000”=0, “0001”=1,... “1111”=15).
- [7:4]: Prom write waitstates. Defines the number of waitstates during prom write cycles (“0000”=0, “0001”=1,... “1111”=15).
- [9:8]: Prom width. Defines the data width of the prom area (“00”=8, “10”=32, “11”=39).
- [10]: Reserved
- [11]: Prom write enable. If set, enables write cycles to the prom area.
- [13:12]: Reserved
- [17:14]: Rom bank size. Defines the size of each rom bank (“0000”=8 Kbyte, “0001”=16 Kbyte... “1111”=256 Mbyte).
- [19]: I/O enable. If set, the access to the memory bus I/O area are enabled.
- [23:20]: I/O waitstates. Defines the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
- [25]: Bus error (BEXCN) enable.
- [26]: Bus ready (BRDYN) enable.
- [28:27]: I/O bus width. Defines the data width of the I/O area (“00”=8, “10”=32).

During power-up, the prom width (bits [9:8]) are set with value on PIO[1:0] inputs. The prom waitstates fields are set to 15 (maximum). External bus error and bus ready are disabled. All other fields are undefined.

6.12 Memory configuration register 2 (MCFG2)

Memory configuration register 2 is used to control the timing of the SRAM and SDRAM.

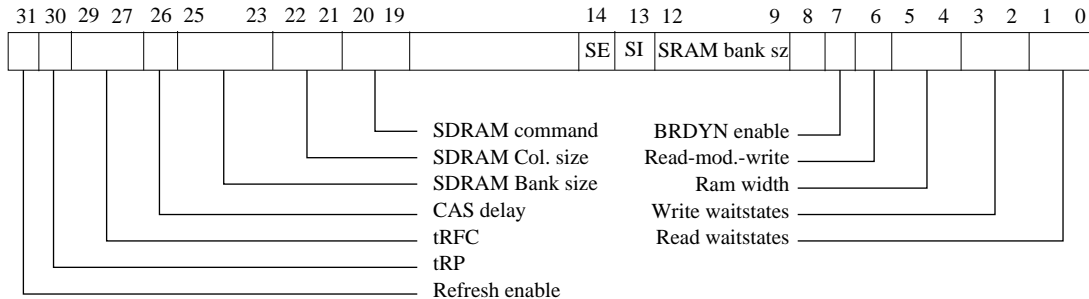


Figure 35: Memory configuration register 2

- [1:0]: Ram read waitstates. Defines the number of waitstates during ram read cycles (“00”=0, “01”=1, “10”=2, “11”=3).
- [3:2]: Ram write waitstates. Defines the number of waitstates during ram write cycles (“00”=0, “01”=1, “10”=2, “11”=3).
- [5:4]: Ram with. Defines the data with of the ram area (“00”=8, “01”=16, “1X”= 32).
- [6]: Read-modify-write. Enable read-modify-write cycles on sub-word writes to 16- and 32-bit areas with common write strobe (no byte write strobe).
- [7]: Bus ready enable. If set, will enable BRDYN for ram area
- [12:9]: Ram bank size. Defines the size of each ram bank (“0000”=8 Kbyte, “0001”=16 Kbyte... “1111”=256 Mbyte).
- [13]: SI - SRAM disable. If set together with bit 14 (SDRAM enable), the static ram access will be disabled.
- [14]: SE - SDRAM enable. If set, the SDRAM controller will be enabled.
- [20:19] SDRAM command. Writing a non-zero value will generate an SDRAM command: “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after command has been executed.
- [22:21]: SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=4096 when bit[25:23]= “111”, 2048 otherwise.
- [25:23]: SDRAM banks size. Defines the banks size for SDRAM chip selectes: “000”=4 Mbyte, “001”=8 Mbyte, “010”=16 Mbyte “111”=512 Mbyte.
- [26]: SDRAM CAS delay. Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time.
- [29:27]: SDRAM t_{RFC} timing. t_{RFC} will be equal to 3 + field-value system clocks.
- [30]: SDRAM t_{RP} timing. t_{RP} will be equal to 2 or 3 system clocks (0/1).
- [31]: SDRAM refresh. If set, the SDRAM refresh will be enabled.

6.13 Memory configuration register 3 (MCFG3)

MCFG3 contains the reload value for the SDRAM refresh counter. The register also controls the memory EDAC, and contains the configuration of the register file EDAC.



Figure 36: Memory configuration register 3

- [31:30]: Register file check bits (RFC) - Indicates how many checkbits are used for the register file (11=7 (EDAC))
- [27]: Memory EDAC (ME) - Indicates if a memory EDAC is present
- [26:12]: SDRAM refresh counter reload value.
- [11]: WB - EDAC diagnostic write bypass
- [10]: RB - EDAC diagnostic read bypass
- [9]: RAM EDAC enable (RE) - Enable EDAC checking of the RAM area
- [8]: PROM EDAC enable (PE) - Enable EDAC checking of the PROM area. At reset, this bit is initialised with the value of PIO[2]
- [7:0]: TCB - Test checkbits. This field replaces the normal checkbits during store cycles when WB is set. TCB is also loaded with the memory checkbits during load cycles when RB is set.

The period between each AUTO-REFRESH command is calculated as follows:

$$t_{\text{REFRESH}} = ((\text{reload value}) + 1) / \text{SYSCLK}$$

6.14 Write protection

Write protection is provided to protect the memory and I/O areas against accidental overwriting. It is implemented as two block protect units capable of disabling or enabling write access to a binary aligned memory block in the range of 32 Kbyte - 1 Mbyte. Each block protect unit is controlled through a control register (figure 37). The units operate as follows: on each write access to RAM, address bits (29:15) are xored with the tag field in the control register, and anded with the mask field. A write protection error is generated if the result is not equal to zero, the corresponding unit is enabled and the block protect bit (BP) is set, or if the BP bit is cleared and the result is equal to zero. If a write protection error is detected, the write cycle is aborted and a memory access error is generated.

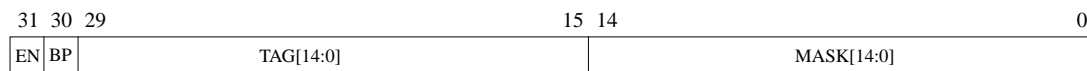


Figure 37: Write protection register 1 & 2

- [14:0] Address mask (MASK) - this field contains the address mask
- [29:15] Address tag (TAG) - this field is compared against address(29:15)
- [30] Block protect (BP) - if set, selects block protect mode
- [31] Enable (EN) - if set, enables the write protect unit

The ROM area can be write protected by clearing the write enable bit MCFG1.

6.15 Using BRDYN

The BRDYN signal can be used to stretch access cycles to the I/O area and the ram area decoded by RAMSN[4]. The accesses will always have at least the pre-programmed number of waitstates as defined in memory configuration registers 1 & 2, but will be further stretched until BRDYN is asserted. BRDYN should be asserted in the cycle preceeding the last one. The use of BRDYN can be enabled separately for the I/O and RAM areas.

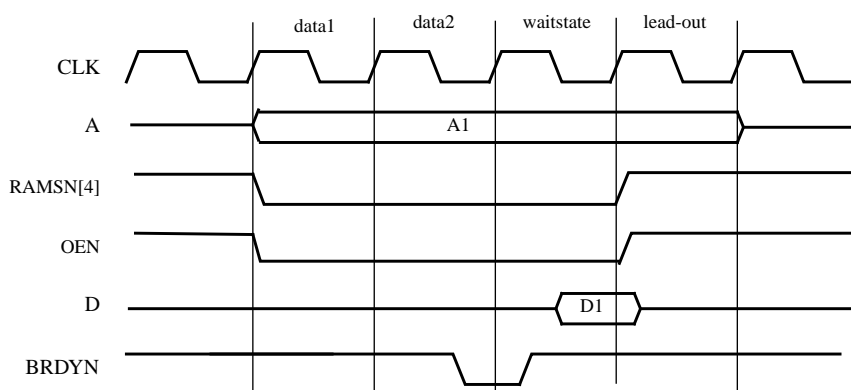


Figure 38: RAM read cycle with one BRDYN controlled waitstate

6.16 Access errors

An access error can be signalled by asserting the BEXCN signal, which is sampled together with the data. If the usage of BEXCN is enabled in memory configuration register 1, an error response will be generated on the internal AMBA bus. BEXCN can be enabled or disabled through memory configuration register 1, and is active for all areas (PROM, I/O and RAM).

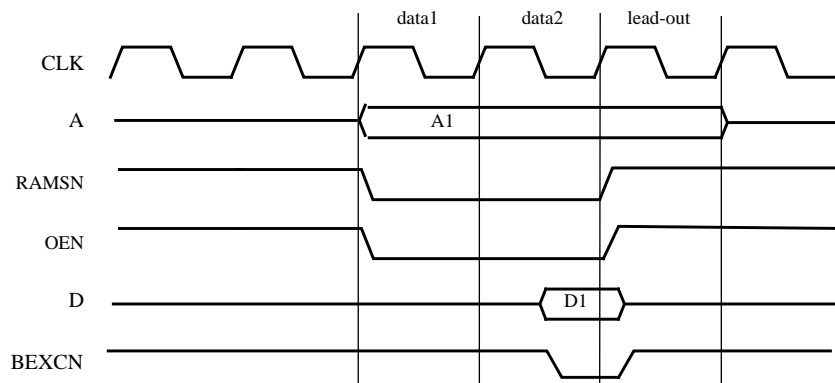


Figure 39: Read cycle with BEXCN

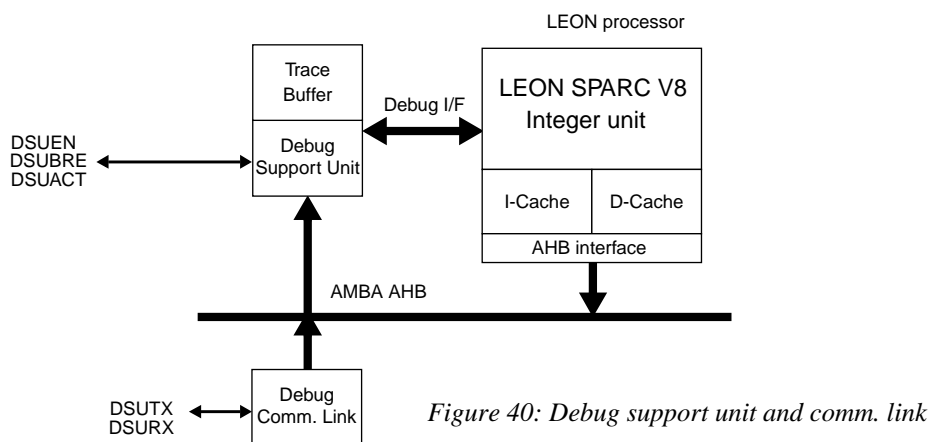
6.17 Attaching an external DRAM controller

To attach an external DRAM controller, RAMSN[4] should be used since it allows the cycle time to vary through the use of BRDYN. In this way, delays can be inserted as required for opening of banks and refresh.

7 Hardware debug support

7.1 Overview

The LEON processor includes hardware debug support to aid software debugging on target hardware. The support is provided through two modules: a debug support unit (DSU) and a debug communication link (DCL). The DSU can put the processor in debug mode, allowing read/write access to all processor registers and cache memories. The DSU also contains a trace buffer which stores executed instructions or data transfers on the AMBA AHB bus. The debug communications link implements a simple read/write protocol and uses standard asynchronous UART communications (RS232C).



7.2 Debug support unit

7.2.1 Overview

The debug support unit is used to control the trace buffer and the processor debug mode. The DSU is attached to the AHB bus as slave, occupying a 2 Mbyte address space. Through this address space, any AHB master can access the processor registers and the contents of the trace buffer. The DSU control registers can be accessed at any time, while the processor registers and caches can only be accessed when the processor has entered debug mode. The trace buffer can be accessed only when tracing is disabled/completed. In debug mode, the processor pipeline is held and the processor is controlled by the DSU. Entering the debug mode can occur on the following events:

- executing a breakpoint instruction (ta 1)
- integer unit hardware breakpoint/watchpoint hit (trap 0xb)
- rising edge of the external break signal (DSUBRE)
- setting the break-now (BN) bit in the DSU control register
- a trap that would cause the processor to enter error mode
- occurrence of any, or a selection of traps as defined in the DSU control register
- after a single-step operation
- DSU breakpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external pin (DSUEN). When the debug mode is entered, the following actions are taken:

- PC and nPC are saved in temporary registers (accessible by the debug unit)
- an output signal (DSUACT) is asserted to indicate the debug state
- the timer unit is (optionally) stopped to freeze the LEON timers and watchdog

The instruction that caused the processor to enter debug mode is not executed, and the processor state is kept unmodified. Execution is resumed by clearing the BN bit in the DSU control register or by de-asserting DSUEN. The timer unit will be re-enabled and execution will continue from the saved PC and nPC. Debug mode can also be entered after the processor has entered error mode, for instance when an application has terminated and halted the processor. The error mode can be reset and the processor restarted at any address.

7.2.2 Trace buffer

The trace buffer consists of a circular buffer that stores executed instructions or AHB data transfers. A 30-bit counter is also provided and stored in the trace as time tag. The trace buffer operation is controlled through the DSU control register and the Trace buffer control register (see below). When the processor enters debug mode, tracing is suspended. The size of the trace buffer is 512 words (= 8kbyte).

The trace buffer is 128 bits wide, the information stored is indicated in table 12 and table 13 below:

Bits	Name	Definition
127	AHB breakpoint hit	Set to '1' if a DSU AHB breakpoint hit occurred.
126	-	Unused
125:96	DSU counter	The value of the DSU counter
95:92	IRL	Processor interrupt request input
91:88	PIL	Processor interrupt level (psr.pil)
95:80	Trap type	Processor trap type (psr.tt)
79	Hwrite	AHB HWRITE
78:77	Htrans	AHB HTRANS
76:74	Hsize	AHB HSIZE
73:71	Hburst	AHB HBURST
70:67	Hmaster	AHB HMASTER
66	Hmastlock	AHB HMASTLOCK
65:64	Hresp	AHB HRESP
63:32	Load/Store data	AHB HRDATA or HWDATA
31:0	Load/Store address	AHB HADDR

Table 12: Trace buffer data allocation, AHB tracing mode

Bits	Name	Definition
127	Instruction breakpoint hit	Set to '1' if a DSU instruction breakpoint hit occurred.
126	Multi-cycle instruction	Set to '1' on the second and third instance of a multi-cycle instruction (LDD, ST or FPOP)
125:96	DSU counter	The value of the DSU counter
95:64	Load/Store parameters	Instruction result, Store address or Store data
63:34	Program counter	Program counter (2 lsb bits removed since they are always zero)
33	Instruction trap	Set to '1' if traced instruction trapped
32	Processor error mode	Set to '1' if the traced instruction caused processor error mode
31:0	Opcode	Instruction opcode

Table 13: Trace buffer data allocation, Instruction tracing mode

During instruction tracing, one instruction is stored per line in the trace buffer with the exception of multi-cycle instructions. Multi-cycle instructions are entered two or three times in the trace buffer. For store instructions, bits [63:32] correspond to the store address on the first entry and to the stored data on the second entry (and third in case of STD). Bit 126 is set on the second and third entry to indicate this. A double load (LDD) is entered twice in the trace buffer, with bits [63:32] containing the loaded data. Multiply and divide instructions are entered twice, but only the last entry contains the result. Bit 126 is set for the second entry. For FPU operation producing a double-precision result, the first entry puts the MSB 32 bits of the results in bit [63:32] while the second entry puts the LSB 32 bits in this field. When a trace is frozen, interrupt 11 is generated.

The trace buffer can executed instructions, transfers on AHB or both (mixed-mode). The trace buffer control register contains two counters that store the address of which location of the trace buffer will be written on next trace. Since the buffer is circular, it actually points to the oldest entry in the buffer. The indexes are automatically incremented after each stored trace entry.

In mixed mode, the buffer is divided on two halves, with instructions stored in the lower half and AHB transfers in the upper half. The MSB bit of the AHB index counter is then automatically kept high, while the MSB of the instruction index counter is kept low.

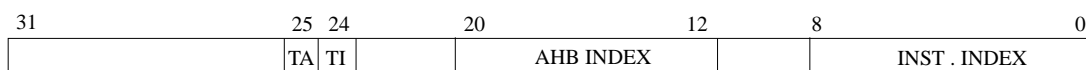


Figure 41: Trace buffer control register

- [8:0] : Instruction trace index counter
- [20:12] : AHB trace index counter
- [24] : Trace instruction enable
- [25] : Trace AHB enable

7.2.3 DSU memory map

DSU memory map can be seen in table 14 below.

Address	Register
0x800000c4	DSU UART status register
0x800000c8	DSU UART control register
0x800000cc	DSU UART scaler register
0x90000000	DSU control register
0x90000004	Trace buffer control register
0x90000008	Time tag counter
0x90000010	AHB break address 1
0x90000014	AHB mask 1
0x90000018	AHB break address 2
0x9000001C	AHB mask 2
0x90010000 - 0x90020000	Trace buffer
..0	Trace bits 127 - 96
...4	Trace bits 95 - 64
...8	Trace bits 63 - 32
...C	Trace bits 31 - 0
0x90020000 - 0x90040000	IU/FPU register file
0x90080000 - 0x90100000	IU special purpose registers
0x90080000	Y register
0x90080004	PSR register
0x90080008	WIM register
0x9008000C	TBR register
0x90080010	PC register
0x90080014	NPC register
0x90080018	FSR register
0x9008001C	DSU trap register
0x90080040 - 0x9008007C	ASR16 - ASR31 (when implemented)
0x90100000 - 0x90140000	Instruction cache tags
0x90140000 - 0x90180000	Instruction cache data
0x90180000 - 0x901C0000	Data cache tags
0x901C0000 - 0x90200000	Data cache data

Table 14: DSU address space

The addresses of the IU/FPU registers depends on how many register windows has been implemented and if and FPU is present. The registers can be accessed at the following addresses (NWINDOWS = number of SPARC register windows = 8):

- %on : $0x90020000 + (((psr.cwp * 64) + 32 + n) \bmod (NWINDOWS * 64))$
- %ln : $0x90020000 + (((psr.cwp * 64) + 64 + n) \bmod (NWINDOWS * 64))$
- %in : $0x90020000 + (((psr.cwp * 64) + 96 + n) \bmod (NWINDOWS * 64))$
- %gn : $0x90020000 + (NWINDOWS * 64) + 128$
- %fn : $0x90020000 + (NWINDOWS * 64)$

7.2.4 DSU control register

The DSU is controlled by the DSU control register:

31	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DCNT			RE	DR	LR	SS	PE	EE	EB	DM	DE	BZ	BX	BB	BN	BS	BW	BE	FT	BT	DM	TE

Figure 42: DSU control register

- 0: Trace enable (TE). Enables the trace buffer.
- 1: Delay counter mode (DM). In mixed tracing mode, setting this bit will cause the delay counter to decrement on AHB traces. If reset, the delay counter will decrement on instruction traces.
- 2: Break on trace (BT) - if set, will generate a DSU break condition on trace freeze.
- 3: Freeze timers (FT) - if set, the scaler in the LEON timer unit will be stopped during debug mode to preserve the time for the software application.
- 4: Break on error (BE) - if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).
- 5: Break on IU watchpoint - if set, debug mode will be forced on a IU watchpoint (trap 0xb).
- 6: Break on S/W breakpoint (BS) - if set, debug mode will be forced when an breakpoint instruction (ta 1) is executed.
- 7: Break now (BN) -Force processor into debug mode. If cleared, the processor will resume execution.
- 8: Break on DSU breakpoint (BD) - if set, will force the processor to debug mode when an DSU breakpoint is hit.
- 9: Break on trap (BX) - if set, will force the processor into debug mode when any trap occurs.
- 10: Break on error traps (BZ) - if set, will force the processor into debug mode on all *except* the following traps: privileged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.
- 11: Delay counter enable (DE) - if set, the trace buffer delay counter will decrement for each stored trace. This bit is set automatically when an DSU breakpoint is hit and the delay counter is not equal to zero.
- 12: Debug mode (DM). Indicates when the processor has entered debug mode (read-only).
- 13: EB - value of the external DSUBRE signal (read-only)
- 14: EE - value of the external DSUEN signal (read-only)
- 15: Processor error mode (PE) - returns '1' on read when processor is in error mode, else '0'.
- 16: Single step (SS) - if set, the processor will execute one instruction and the return to debug mode.
- 17: Link response (LR) - is set, the DSU communication link will send a response word after AHB transfer.
- 18: Debug mode response (DR) - if set, the DSU communication link will send a response word when the processor enters debug mode.
- 19: Reset error mode (RE) - if set, will clear the error mode in the processor.
- 28:20 Trace buffer delay counter (DCNT).

7.2.5 DSU breakpoint registers

The DSU contains two breakpoint registers for matching either AHB addresses or executed processor instructions. A breakpoint hit is typically used to freeze the trace buffer, but can also put the processor in debug mode. Freezing can be delayed by programming the TDELAY field in the DSU control register to a non-zero value. In this case, the TDELAY value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. If the BT bit in the DSU control register is set, the DSU will force the processor into debug mode when the trace buffer is frozen. Note that due to pipeline delays, up to 4 additional instruction can be executed before the processor is placed in debug mode. A mask register is associated with each breakpoint, allowing breaking on a block of addresses. Only address bits with the corresponding mask bit set to '1' are compared during breakpoint detection. To break on executed instructions, the EX bit should be set. To break on AHB load or store accesses, the LD and/or ST bits should be set.

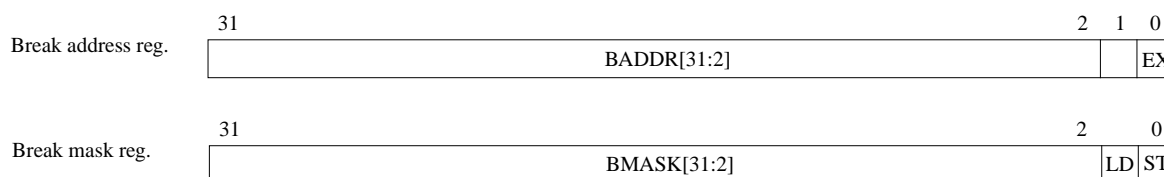


Figure 43: DSU breakpoint registers

7.2.6 DSU trap register

The DSU trap register is a read-only register that indicates which SPARC trap type that caused the processor to enter debug mode. When debug mode is force by setting the BN bit in the DSU control register, the trap type will be 0xb (hardware watchpoint trap).



Figure 44: DSU trap register

- [11:4] : 8-bit SPARC trap type
- 12 : Error mode (EM). Set if the trap would have cause the processor to enter error mode.

7.3 DSU communication link

7.3.1 Operation

The DSU communication link consists of a UART connected to the AHB bus as a master (figure 45). A simple communication protocol is supported to transmit access parameters and data. A link command consist of a control byte, followed by and a 32-bit address, followed by optional write data. If the LR bit in the DSU control register is set, a response byte will be sent after each AHB transfer. If the LR bit is not set, a write access does not return any response, while a read access only returns the read data. Data is sent on 8-bit basis as shown in figure 47. Through the communication link, a read or write transfer can be generated to any address on the AHB bus.

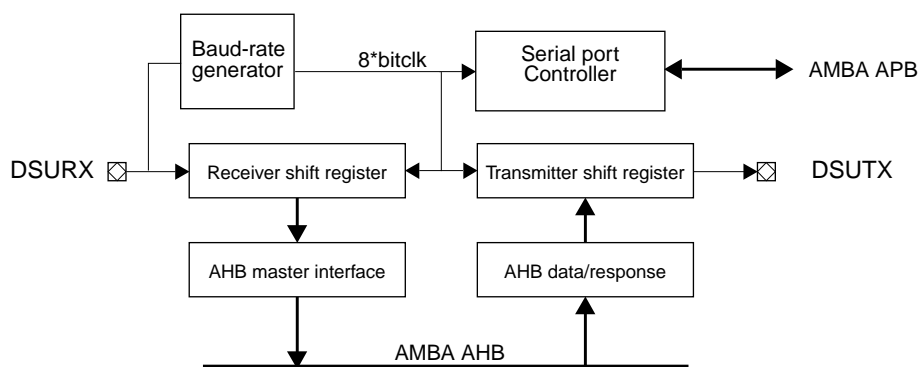


Figure 45: DSU communication link block diagram

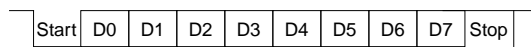


Figure 46: DSU UART data frame

DSU Write Command

Send 11 | Length -1 | Addr[31:24] | Addr[23:16] | Addr[15:8] | Addr[7:0] | Data[31:24] | Data[23:16] | Data[15:8] | Data[7:0]

Receive Resp. byte (optional)

DSU Read command

Send 10 | Length -1 | Addr[31:24] | Addr[23:16] | Addr[15:8] | Addr[7:0]

Receive Data[31:24] | Data[23:16] | Data[15:8] | Data[7:0] | Resp. byte (optional)

Response byte encoding

bit 7:3 = 000000
bit 2 = DMODE
bit 1:0 = HRESP

Figure 47: DSU Communication link commands

A response byte is can optionally be sent when the processor goes from execution mode to debug mode. Block transfers can be performed by setting the length field to $n-1$, where n denotes the number of transferred words. For write accesses, the control byte and address is sent once, followed by the number of data words to be written. The address is automatically incremented after each data word. For read accesses, the control byte and address is sent once and the corresponding number of data words is returned.

7.3.2 DSU UART control register



Figure 48: UART control register

- 0: Receiver enable (RE) - if set, enables both the transmitter and receiver.
- 1: Baud rate locked (BL) - is automatically set when the baud rate is locked.

7.3.3 DSU UART status register



Figure 49: UART status register

- 0: Data ready (DR) - indicates that new data is available in the receiver holding register.
- 1: Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty.
- 2: Transmitter hold register empty (TH) - indicates that the transmitter hold register is empty.
- 4: Overrun (OV) - indicates that one or more character have been lost due to overrun.
- 6: Framing error (FE) - indicates that a framing error was detected.

7.3.4 Baud rate generation

The UART contains a 14-bit down-counting scaler to generate the desired baud-rate. The scaler is clocked by the system clock and generates a UART tick each time it underflows. The scaler is reloaded with the value of the UART scaler reload register after each underflow. The resulting UART tick frequency should be 8 times the desired baud-rate.

If not programmed by software, the baud rate will be automatically be discovered. This is done by searching for the shortest period between two falling edges of the received data (corresponding to two bit periods). When three identical two-bit periods has been found, the corresponding scaler reload value is latched into the reload register, and the BL bit is set in the UART control register. If the BL bit is reset by software, the baud rate discovery process is restarted. The baud-rate discovery is also restarted when a 'break' is received by the receiver, allowing to change to baudrate from the external transmitter. For proper baudrate detection, the value 0x55 should be transmitted to the receiver after reset or after sending break.

The best scaler value for manually programming the baudrate can be calculated as follows:

$$\text{scaler} = (((\text{system_clk} * 10) / (\text{baudrate} * 8)) - 5) / 10$$



Figure 50: DSU UART scaler reload register

7.4 Common operations

7.4.1 Instruction breakpoints

To insert instruction breakpoints, the breakpoint instruction (ta 1) should be used. This will leave the four IU hardware breakpoints free to be used as data watchpoints. Since cache snooping is only done on the data cache, the instruction cache must be flushed after the insertion or removal of breakpoints. To minimize the influence on execution, it is enough to clear the corresponding instruction cache tag (which is accesible through the DSU).

The DSU hardware breakpoints should only be used to freeze the trace buffer, and not for software debugging since there is a 4-cycle delay from the breakpoint hit before the processor enters the debug mode.

7.4.2 Single stepping

By writing the SS bit and resetting the BN bit in the DSU control register, the processor will resume execution for one instruction and then automatically enter debug mode.

7.4.3 Alternative debug sources

It is possible to debug the processor through any available AHB master since the DSU is a regular AHB slave. For instance, if a PCI interface is available, all debugging features will be available from any other PCI master.

7.4.4 Booting from DSU

By asserting DSUEN and DSUBRE at reset time, the processor will directly enter debug mode without executing any instructions. The system can then be initialised from the communication link, and applications can be downloaded and debugged. Additionally, external (flash) prompts for standalone booting can be re-programmed.

7.5 DSU monitor

Gaisler Research provides a DSU monitor that allows both standalone debugging as well as an interface to gdb. See www.gaisler.com for details.

7.6 External DSU signals

The DSU uses five external signals: DSUACT, DSUBRE, DSUEN, DSURX and DSUTX.

8 Signals

8.1 Memory bus signals

Name	Type	Function	Active
A[30:0]	Output	Memory address	High
BEXCN	Input	Bus exception	Low
BRDYN	Input	Bus ready strobe	Low
D[31:0]	Bidir	Memory data	High
CB[7:0]	Bidir	Memory EDAC checkbits	High
IOSN	Output	Local I/O select	Low
OEN	Output	Output enable	Low
RAMOEN[3:0]	Output	SRAM output enable	Low
RAMSN[3:0]	Output	SRAM chip-select	Low
READ	Output	Read strobe	High
ROMSN[1:0]	Output	PROM chip-select	Low
RWEN[3:0]	Output	SRAM write enable	Low
SDCASN	Output	SDRAM column address strobe	Low
SDCLK	Output	SDRAM clock	-
SDCKE[1:0]	Output	SDRAM clock enable	High
SDCSN[1:0]	Output	SDRAM chip select	Low
SDDQM[3:0]	Output	SDRAM data mask	Low
SDRASN	Output	SDRAM row address strobe	Low
SDWEN	Output	SDRAM write enable	Low
WRITEN	Output	Write strobe	Low

Table 15: Memory bus signals

8.2 System interface signals

Name	Type	Function	Active
CLK	Input	System clock	High
ERRORN	Open-drain	System error	Low
PIO[15:0]	Bidir	Parallel I/O port	High
RESETN	Input	System reset	Low
WDOGN	Open-drain	Watchdog output	Low
DSUACT	Output	DSU active	High
DSUBRE	Input	DSU break	High
DSUEN	Input	DSU enable	High
DSURX	Input	DSU communication link transmission input	High
DSUTX	Output	DSU communication link transmission output	High

Table 16: System interface signals

8.3 Signal description

All signals are clocked on the rising edge of CLK.

A[30:0] - Address bus (output)

These active high outputs carry the address during accesses on the memory bus. When no access is performed, the address of the last access is driven (also internal cycles).

BEXCN - Bus exception (input)

This active low input is sampled simultaneously with the data during accesses on the memory bus. If asserted, a memory error will be generated.

BRDYN - Bus ready (input)

This active low input indicates that the access to a memory mapped I/O area can be terminated on the next rising clock edge.

CLK - Processor clock (input)

This active high input provides the main processor clock.

D[31:0] - Data bus (bi-directional)

D[31:0] carries the data during transfers on the memory bus. The processor only drives the bus during write cycles. During accesses to 8-bit areas, only D[31:24] are used.

DSUACT - DSU active (output)

This active high output is asserted when the processor is in debug mode and controlled by the DSU.

DSUBRE - DSU break enable

A low-to-high transition on this active high input will generate break condition and put the processor in debug mode.

DSUEN - DSU enable (input)

The active high input enables the DSU unit. If de-asserted, the DSU trace buffer will continue to operate but the processor will not enter debug mode.

DSURX - DSU receiver (input)

This active high input provides the data to the DSU communication link receiver

DSUTX - DSU transmitter (output)

This active high input provides the output from the DSU communication link transmitter.

ERROR - Processor error (open-drain output)

This active low output is asserted when the processor has entered error state and is halted. This happens when traps are disabled and an synchronous (un-maskable) trap occurs.

IOSN - I/O select (output)

This active low output is the chip-select signal for the memory mapped I/O area.

OEN - Output enable (output)

This active low output is asserted during read cycles on the memory bus.

PIO[15:0] - Parallel I/O port (bi-directional)

These bi-directional signals can be used as inputs or outputs to control external devices.

RAMOEN[3:0] - RAM output enable (output)

These active low signals provide an individual output enable for each RAM bank.

RAMSN[3:0] - RAM chip-select (output)

These active low outputs provide the chip-select signals for each RAM bank.

READ - Read cycle

This active high output is asserted during read cycles on the memory bus.

RESETN - Processor reset (input)

When asserted, this active low input will reset the processor and all on-chip peripherals.

ROMSN[1:0] - PROM chip-select (output)

These active low outputs provide the chip-select signal for the PROM area. ROMSN[0] is asserted when the lower half of the PROM area is accessed (0 - 0x10000000), while ROMSN[1] is asserted for the upper half.

RWEN [3:0] - RAM write enable (output)

These active low outputs provide individual write strobes for each byte lane. RWEN[0] controls D[31:24], RWEN[1] controls D[23:16], etc.

SDCLK - SDRAM clock

SDRAM clock, can be configured to be identical or inverted in relation to the system clock.

SDCKE[1:0] - SDRAM clock enable

Currently unused, driven permanently high.

SDCASN - SDRAM column address strobe

This active low signal provides a common CAS for all SDRAM devices.

SDCSN[1:0] - SDRAM chip select

These active low outputs provide the chip select signals for the two SDRAM banks.

SDDQM[3:0] - SDRAM data mask

These active low outputs provide the DQM signals for both SDRAM banks.

SDRASN - SDRAM row address strobe

This active low signal provides a common RAS for all SDRAM devices.

SDWEN - SDRAM write strobe

This active low signal provides a common write strobe for all SDRAM devices.

WDOGN - Watchdog time-out (open-drain output)

This active low output is asserted when the watchdog times-out.

WRITEN - Write enable (output)

This active low output provides a write strobe during write cycles on the memory bus.