GRLIB IP Core User’s Manual

Version 1.0.19, September 2008

Jiri Gaisler, Edvin Catovic, Marko Isomäki, Kristoffer Glembo, Sandi Habinc

Table of contents

1 Introduction

1.1 Scope

This document describes specific IP cores provided with the GRLIB IP library. When applicable, the cores use the GRLIP plug&play configuration method as described in the ‘GRLIB User’s Manual’.

1.2 IP core overview

The tables below lists the provided IP cores and their AMBA plug&play device ID. The license column indicates if a core is available under GNU GPL and/or under a commercial license. Cores marked with FT are only available in the FT version of GRLIB. Note: the open-source version of GRLIB includes only cores marked with GPL or LGPL.

Table 1. Processors and support functions

Name	Function	Vendor:Device	License
LEON3	SPARC V8 32-bit processor	0x01 : 0x003	COM/GPL
DSU3	Multi-processor Debug support unit	0x01 : 0x004	COM/GPL
IRQMP	Multi-processor Interrupt controller	0x01 : 0x00D	COM/GPL
GRFPU	High-performance IEEE-754 Floating-point unit	-	COM
GRFPU-Lite	Low-area IEEE-754 Floating-point unit	-	COM
LEON3FT	Fault-tolerant SPARC V8 32-bit Processor	0x01 : 0x053	FT

Table 2. Floating-point units

Name	Function	Vendor:Device	License
GRFPU	High-performance IEEE-754 Floating-point unit	-	COM
GRFPU-Lite	Low-area IEEE-754 Floating-point unit	-	COM

Table 3. Memory controllers

Name	Function	Vendor:Device	License
SRCTRL	8/32-bit PROM/SRAM controller	0x01 : 0x008	COM/GPL
SDCTRL	PC133 SDRAM controller	0x01 : 0x009	COM/GPL
FTSDCTRL	32/64-bit PC133 SDRAM Controller with EDAC	0x01 : 0x055	FT
FTSRCTRL	Fault Tolerant 32-bit PROM/SRAM/IO Controller	0x01 : 0x051	FT
MCTRL	8/16/32-bit PROM/SRAM/SDRAM controller	0x04 : 0x00F	LGPL
FTMCTRL	8//32-bit PROM/SRAM/SDRAM controller with EDAC	0x01 : 0x054	FT
AHBSTAT	AHB failing address register	0x01 : 0x052	COM/GPL
DDRCTRL	8/16/32/64-bit DDR controller with two AHB ports (Xilinx only)	0x01 : 0x023	COM/GPL
DDRSPA	Single-port 16/32/64 bit DDR266 controller (Xilinx and Altera)	0x01 : 0x025	COM/GPL
SSRCTRL	32-bit synchronous SRAM (SSRAM) controller	0x01 : 0x00A	COM
FTSRCTRL8	8-bit SRAM / 16-bit IO Memory Controller with EDAC	0x01 : 0x056	FT
SPIMCTRL	SPI Memory controller	0x01 : 0x045	COM/GPL

Table 4. AMBA Bus control

Name	Function	Vendor:Device	License
AHB2AHB	Uni-directional AHB/AHB Bridge	0x01:0x020	COM
AHBBRIDGE	Bi-directional AHB/AHB Bridge	0x01:0x020	COM
AHBCTRL	AMBA AHB bus controller with plug&play	-	COM/GPL
AHBCTRL_MB	AMBA AHB bus controller for multiple buses with plug&play		COM
APBCTRL	AMBA APB Bridge with plug&play	0x01 : 0x006	COM/GPL
AHBTRACE	AMBA AHB Trace buffer	0x01 : 0x017	COM/GPL

Table 5. PCI interface

Name	Function	Vendor:Device	License
PCITARGET	32-bit target-only PCI interface	0x01 : 0x012	COM/GPL
PCIMTF/GRPCI	32-bit PCI master/target interface with FIFO	0x01 : 0x014	COM/GPL
PCITRACE	32-bit PCI trace buffer	0x01 : 0x015	COM/GPL
PCIDMA	DMA controller for PCIMTF	0x01 : 0x016	COM/GPL
PCIARB	PCI Bus arbiter	0x04 : 0x010	LGPL
WILD2AHB	WildCard Debug Interface with DMA Master Interface	0x01 : 0x079	COM/GPL

Table 6. On-chip memory functions

Name	Function	Vendor:Device	License
AHBRAM	Single-port RAM with AHB interface	0x01 : 0x00E	COM/GPL
AHBROM	ROM generator with AHB interface	0x01 : 0x01B	COM/GPL
SYNCRAM	Parametrizable 1-port RAM	-	COM/GPL
SYNCRAM_2P	Parametrizable 2-port RAM	-	COM/GPL
SYNCRAM_DP	Parametrizable dual-port RAM	-	COM/GPL
REGFILE_3P	Parametrizable 3-port register file	-	COM/GPL
FTAHBRAM	RAM with AHB interface and EDAC protection	0x01 : 0x050	FT

Table 7. Serial communication

Name	Function	Vendor:Device	License
AHBUART	Serial/AHB debug interface	0x01 : 0x007	COM/GPL
AHBJTAG	JTAG/AHB debug interface	0x01 : 0x01C	COM/GPL
APBPS2	PS2 Keyboard interface with APB interface	0x01 : 0x060	COM/GPL
APBUART	Programmable UART with APB interface	0x01 : 0x00C	COM/GPL
CAN_OC	Opencores CAN 2.0 MAC with AHB interface	0x01 : 0x019	COM/GPL
GRCAN	CAN 2.0 Controller with DMA	0x01 : 0x03D	COM
GRSPW	SpaceWire link with RMAP and AHB interface	0x01 : 0x01F	FT
I2CMST	I2C Master with APB interface	0x01 : 0x028	COM/GPL
I2CSLV	I2C Slave with APB interface	0x01 : 0x03E	COM/GPL
SPICTRL	SPI Controller with APB interface	0x01 : 0x02D	COM/GPL

Table 8. Ethernet interface

Name	Function	Vendor:Device	License
GRETH	Gaisler Research 10/100 Mbit Ethernet MAC with AHB I/F	0x01 : 0x01D	COM/GPL
GRETH_GIGA	Gaisler Research 10/100/1000 Mbit Ethernet MAC with AHB	0x01 : 0x01D	COM

Table 9. USB interface

Name	Function	Vendor:Device	License
GRUSBHC	USB-2.0 Host controller (UHCI/EHCI) with AHB I/F	0x01 : 0x027	COM
USBDCL	USB-2.0 device controller / AHB debug communication link	0x01 : 0x022	COM

Table 10. MIL-STD-1553 Bus interface

Name	Function	Device ID	License
B1553BC	1553 Bus controller with AHB interface	0x01 : 0x070	COM
B1553RT	1553 Remote terminal with AHB interface	0x01 : 0x071	COM
B1553BRM	1553 BC/RT/Monitor with AHB interface	0x01 : 0x072	COM

Table 11. Encryption

Name	Function	Vendor:Device	License
GRAES	128-bit AES Encryption/Decryption Core	0x01 : 0x073	COM
GRECC	Elliptic Curve Cryptography Core	0x01 : 0x074	COM

Table 12. Simulation and debugging

Name	Function	Vendor:Device	License
SRAM	SRAM simulation model with srecord pre-load	-	COM/GPL
MT48LC16M16	Micron SDRAM model with srecord pre-load	-	Free
MT46V16M16	Micron DDR model	-	Free
CY7C1354B	Cypress ZBT SSRAM model with srecord pre-load	-	Free
AHBMSTEM	AHB master simulation model with scripting	0x01 : 0x040	COM/GPL
AHBSLVEM	AHB slave simulation model with scripting	0x01 : 0x041	COM/GPL
AMBAMON	AHB and APB protocol monitor	-	COM

Table 13. CCSDS Telecommand and telemetry functions

Name	Function	Vendor:Device	License
GRTM	CCSDS Telemetry Encoder	0x01 : 0x030	FT
GRTC	CCSDS Telecommand Decoder	0x01 : 0x031	FT
GRPW	Packetwire receiver with AHB interface	0x01 : 0x032	COM/GPL
GRCTM	CCSDS Time manager	0x01 : 0x033	COM/GPL
GRHCAN	CAN controller with DMA	0x01 : 0x034	FT
GRFIFO	External FIFO Interface with DMA	0x01 : 0x035	COM
GRADCDAC	Combined ADC / DAC Interface	0x01 : 0x036	COM
GRPULSE	General Purpose Input Output	0x01 : 0x037	FT
GRTIMER	General Purpose Timer Unit	0x01 : 0x038	FT
AHB2PP	Packet Parallel Interface	0x01 : 0x039	FT
GRVERSION	Version and Revision information register	0x01 : 0x03A	FT
APB2PW	PacketWire Transmitter Interface	0x01 : 0x03B	COM/GPL
PW2APB	PacketWire Receiver Interface	0x01 : 0x03C	COM/GPL
GRCE/GRCD	CCSDS/ECSS Convolutional Encoder and Quicklook Decoder	N/A	FT
GRTMRX	CCSDS Telemetry Receiver	0x01 : 0x082	{internal}
GRTCTX	CCSDS Telecommand Transmitter	0x01 : 0x083	{internal}

Note: The CCSDS functions are described in separate manuals.

Table 14. HAPS functions

Name	Function	Vendor:Device	License
HAPSTRAK	HapsTrak controller for HAPS boards	0x01 : 0x077	GPL
FLASH_1X1	32/16-bit PROM Controller for HAPS FLASH_1x1	0x01 : 0x00A	COM *
SRAM_1X1	32-bit SSRAM / PROM Controller for HAPS SRAM_1x1	0x01 : 0x00A	COM *
TEST_1X2	Controller for HAPS test daughter board TEST_1x2	0x01 : 0x078	COM/GPL
BIO1	Controller for HAPS I/O board BIO1	0x01 : 0x07A	COM/GPL
SDRAM_1X1	32-bit SDRAM Controller for HAPS SDRAM_1x1	0x01 : 0x009	COM/GPL
DDR_1X1	64-bit DDR266 Controller for HAPS DDR_1x1	0x01 : 0x025	COM/GPL
GEPHY_1X1	Ethernet Controller for HAPS GEPHY_1x1	0x01 : 0x00A	COM **

Note*: The underlying SSRAM controller used in the FLASH_1X1 and SRAM_1X1 cores is provided in VHDL netlist format in the GRLIB GPL distribution. The VHDL source code is only provided under commercial license.
Note**: The 10/100 Mbit Media Access Controller (MAC) is available in the GRLIB GPL distribution. The 1000 Mbit MAC is only provided under commercial license.

Note: The HAPS functions are described in separate manuals.

1.3 Implementation characteristics

The table below shows the approximate area for some of the GRLIP IP blocks mapped on Virtex2, Actel-AX and typical ASIC technologies. The area depends strongly on configuration options (generics), optimization constraints and used synthesis tools. The data in the table should therefore be seen as an indication only. The tools used to obtain the area was Synplify-8.1 for FPGA and Synopsys DC for ASIC. The LUT area for Altera Stratix devices is roughly the same as for Virtex2. Using XST instead of Synplify for Xilinx FPGAs gives typically 15% larger area.

Table 15. Approximate area consumption for some standard GRLIB IP cores

	Virtex2		**AX/RTAX**		**ASIC**
Block	**LUT**	**RAM16**	**Cells**	**RAM64**	**Gates**
AHBCTRL	200		500		1,000
AHBJTAG	120		350		1,000
AHBUART (DSU UART)	450		800		2,000
APBCTRL	150		200		800
APBPS2	450		800		2,000
APBUART	200		300		1,000
APBVGA	250	4	-		1,400
ATACTRL	400		600		2,000
CAN_OC (CAN-2.0 core with AHB I/F)	1,600	2	2,800	2	8,000
GRCAN (CAN 2.0 Controller with DMA)	2,300		4,800		20,000
DDRCTRL	1,600	2	-		10,000
DDRSPA (32-bit)	900	2	-		-

Table 15. Approximate area consumption for some standard GRLIB IP cores

	Virtex2		AX/RTAX		ASIC
Block	**LUT**	**RAM16**	**Cells**	**RAM64**	**Gates**
DIV32 (64/32-bit iterative divider)	400		500		2,000
GPTIMER (16-bit scaler + 2x32-bit timers)	250		400		1,300
GRETH 10/100 Mbit Ethernet MAC	1,500		2,500	2	8,000
GRETH 10/100 Mbit Ethernet MAC with EDCL	2,600	1	4,000	4	15,000
GRFPU-Lite including LEON3 controller	4,000	6	7,000	4	35,000
GRFPU IEEE-754 floating-point unit	8,500	2	-		100,000
GRFPC for LEON3	5,000	4	-		25,000
GRGPIO, 16-bit configuration	100		150		800
GRSWP Spacewire link	1,900	3	2,800	3	15,000
GRSWP Spacewire link with RMAP	3,000	4	4,500	4	25,000
GRTC CCSDS telecommad decoder front-end	2,000		3,000		15,000
I2CMST I2C Master	200		300		1,500
I2CSLV I2C Slave	150		250		1,000
IRQMP (1 processor)	300		350		1,500
LEON3, 8 + 8 Kbyte cache	4,300	12	6,500	40	20,000
LEON3, 8 + 8 Kbyte cache + DSU3	5,000	12	7,500	40	25,000
LOGAN, 32 channels, 1024 traces, 1 trigger	300	2	-		-
MCTRL	350		1,000		1,500
MCTRL including SDRAM support	600		1,400		2,000
MUL32 (32x32 multiplier, 4-cycle iterative)	200		1,400		5,500
PCI_TARGET, simple PCI target	150		500		800
PCI_MTF, master/target PCI with FIFO	1,100	4	2,000	4	6,000
PCIDMA, master/target PCI with FIFO/DMA	1,800	4	3,000	4	9,000
PCITRACE	300	2	600	4	1,400
SRCTRL	100		200		500
SDCTRL	300		600		1,200
SPICTRL	400		700		2,500
SPIMCTRL	300		600		1,200
SVGACTRL	1,200	2	1,600	2	8,000
USBDCL	2,000				12,000

Table 16. Approximate area consumption for some FT GRLIB IP cores

Block	**RTAX2000 (Cells)**	**ASIC (gates)**
GRFPU-Lite-FT including LEON3 controller	7,100 + 4 RAM64K36	36,000
GRFPCFT for LEON3	-	30,000 + RAM
LEON3FT, 8 + 4 Kbyte cache	7,500 + 40 RAM64K36	22,000 + RAM
LEON3FT, 8 + 4 Kbyte cache + DSU3	8,500 + 44 RAM64K36	27,000 + RAM
LEON3FT, 8 + 4 Kbyte cache with FPU + DSU3	16,000 + 48 RAM64K36	60,000 + RAM
FTSRCTRL	700	2,500
FTSRCTRL8	750	-
FTSDCTRL	1,000	3,500
FTAHBRAM (2 Kbyte with EDAC)	300 + 5 RAM64K36	2,000 + RAM

The table below show the area resources for some common FPGA devices. It can be used to quickly estimate if a certain GRLIB design will fit the target device.

Table 17. Area resources for some common FPGA devices

FPGA	**Logic**	**Memory**
Actel AX1000	18,144 Cells	32 RAM64K36
Actel AX2000	32,248 Cells	64 RAM64K36
Xilinx Spartan3-1500	33,248 LUT	64 RAMB16
Xilinx Virtex2-3000	28,672 LUT	96 RAMB16
Xilinx Virtex2-6000	67,584 LUT	144 RAMB16

2 AHB2AHB - Uni-directional AHB to AHB bridge

2.1 Overview

Uni-directional AHB to AHB bridge is used to connect two AHB buses clocked by synchronous clocks with any frequency ratio. The bridge is connected through a pair consisting of an AHB slave and an AHB master interface. AHB transfer forwarding is performed in one direction, AHB transfers to the slave interface are forwarded to the master interface. Application of the uni-directional bridge include system partitioning, clock domain partitioning and system expansion.

Features offered by the uni-directional AHB to AHB bridge are:

- single and burst AHB transfers
- data buffering in internal FIFOs
- efficient bus utilization through use of SPLIT response and data prefetching
- posted writes
- dead-lock detection logic enables use of two uni-directional bridges to build bi-directional bridge (see AHB/AHB bridge core)

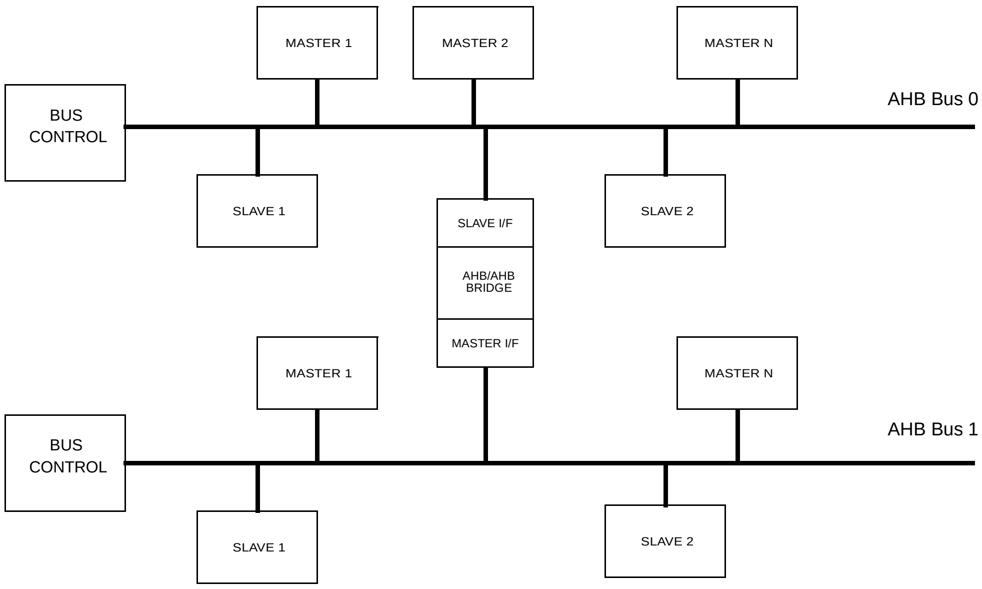


Figure 1. Two AHB buses connected with (uni-directional) AHB/AHB bridge

2.2 Operation

2.2.1 General

Address space occupied by the AHB/AHB bridge on the slave bus is configurable and determined by Bank Address Registers in the slave interface AHB Plug&Play configuration record.

The bridge is capable of handling single and burst transfers of all burst types. Supported transfer sizes (HSIZE) are byte, halfword and word.

For AHB write transfers write data is always buffered in internal FIFO implementing posted writes. For AHB read transfers the bridge uses GRLIB’s AMBA Plug&Play information to determine whether the read data will be prefetch and buffered in internal FIFO. If the target address for an AHB read burst transfer is a prefetchable location the read data will be prefetched and buffered.

AHB master initiating a read transfer to the bridge is always splitted on the first transfer attempt to allow other masters to use the slave bus while the bridge performs read transfer on the master bus.

If the interrupt forwarding is enabled the interrupts on the slave bus interrupt lines will be forwarded to the master bus and vice versa.

2.2.2 AHB read transfers

When a read transfer is registered on the slave interface the bridge gives SPLIT response. The master that initiated the transfer will be de-granted allowing other bus master to use the slave bus while the bridge performs read transfer on the master side. The master interface requests the bus and starts the read transfer on the master side. Single transfers on the slave side are translated to single transfers with the same AHB address and control signals on the master side. Translation of burst transfers from slave to master side depends on burst type, burst length and the AHB/AHB bridge configuration.

If the read FIFO is enabled and the transfer is a burst transfer to a prefetchable location the master interface will prefetch data in the internal read FIFO. Burst type as well as other AHB address and control signals are the same as for the splitted transfer on the slave side. Fixed length bursts have length according to the number of beats in the transfer. If the splitted burst on the slave side was an incremental burst of unspecified length (INCR), the length of the burst is unknown. In this case the master interface performs an incremental burst of configurable length (determined by VHDL generic *rburst)*. The bridge can be configured to recognize an INCR read burst marked as instruction fetch (indicated on HPROT signal). In this case the prefetching on the master side is completed at the end of cache line (cache line size is configurable through VHDL generic *iburst*). When the burst transfer is completed on the master side, the splitted master that initiated the transfer (on the slave side) is allowed in bus arbitration by asserting appropriate HSPLIT signal to the AHB controller. The splitted master re-attempts the transfer and the bridge will return data with zero wait states.

If the read FIFO is disabled or the burst is to non-prefetchable area the burst transfer on the master side is performed using sequence of NONSEQ, BUSY and SEQ transfers. First access in the burst on the master side is of NONSEQ type. Since the master interface can not decide whether the splitted burst will continue on the slave side or not, the master bus is held by performing BUSY transfers. On the slave side the splitted master that initiated the transfer is allowed in bus arbitration by asserting HSPLIT signal to the AHB controller. The first access in the transfer is completed by returning read data. Next access in the transfer on the slave side is extended by asserting HREADY low. On the master side the next access is started by performing SEQ transfer (and then holding the bus using BUSY transfers). This sequence is repeated until the transfer is ended on the slave side.

In case of an error response on the master side the error response will be given for the same access (address) on the slave side. SPLIT and RETRY responses on the master side are re-attempted until OKAY or ERROR response is received.

2.2.3 AHB write transfers

AHB/AHB bridge implements posted writes. During the AHB write transfer on the slave side the data is buffered in the internal write FIFO and the transfer is completed on the slave side by always giving OKAY response. The master interface requests the bus and performs the write transfer when the master bus is granted. If the burst transfer is longer than the size of the write FIFO the SPLIT response is given when the FIFO gets full. When the FIFO becomes empty the splitted master is allowed to re-attempt the remaining accesses of the write burst transfer.

2.2.4 Locked transfers

The AHB/AHB bridge supports locked transfers. The master bus will be locked when the bus is granted and remain locked until the transfer completes on the slave side.

Locked transfers can however *not* be made after another access which received SPLIT until it has received split complete. This is because the bridge will return split complete for the first access first which will cause deadlock since the arbiter is not allowed to change master until a locked transfer has been completed.

2.3 Registers

The core does not implement any registers.

2.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x020. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

2.5 Implementation

2.5.1 Technology mapping

Uni-directional AHB to AHB bridge has one technology mapping generic *memtech*. *memtech* selects which memory technology will be used to implement FIFO memories.

2.5.2 RAM usage

Uni-directional AHB to AHB bridge instantiates one or two *syncram_2p* blocks from the technology mapping library (TECHMAP). If prefetching is enabled one *syncram_2p* block with organization *rbufsz* x 32 is used to implement read FIFO (*rbufsz* is the size of the read FIFO). One *syncram_2p* block with organization *wbufsz* x 32, is always used to implement write FIFO (where *wbufsz* is the size of the write FIFO).

2.6 Configuration options

Table 18 shows the configuration options of the core (VHDL generics).

Table 18. Configuration options (VHDl generics)

Generic	Function	Allowed range	Default
memtech	Memory technology		0
hsindex	Slave I/F AHB index	0 to NAHBMAX-1	0
hmindex	Master I/F AHB index	0 to NAHBMAX-1	0
dir	0 - clock frequency on the master bus is lower then the frequency on the slave bus 1 - clock frequency on the master bus is higher or equal to the frequency on the slave bus	0 - 1	0
ffact	Frequency scaling factor between AHB clocks on master and slave buses.	1 - 15	2
slv	Slave bridge. Used in bi-directional bridge configuration where *slv* is set to 0 for master bridge and 1 for slave bridge. When a dead-lock condition is detected slave bridge (*slv*=1) will give RETRY response to current access, effectively resolving the dead-lock situation.	0 - 1	0
pfen	Prefetch enable. Enables read FIFO.	0 - 1	0
irqsync	Interrupt forwarding. Forward interrupts from slave interface to master interface and vice versa. 0 - no interrupt forwarding, 1 - forward interrupts 1 - 15, 2 - forward interrupts 0 - 31. Since interrupts are forwarded in both directions, interrupt forwarding should be enabled for one bridge only in a bi-directional AHB/AHB bridge.	0 - 2	0

Table 18. Configuration options (VHDl generics)

Generic	Function	Allowed range	Default
rbufsz	Read FIFO size. Determines the maximum length of prefetching read bursts on the master side. For systems where AHB masters perform fixed length burst (INCRx , WRAPx) *rbufsz* should not be less than the length of the longest fixed length burst.	2 - 32	8
wbufsz	Write FIFO size	2 - 32	2
iburst	Instruction fetch burst length	4 - 8	8
rburst	Incremental read burst length. Determines the length of incremental read burst of unspecified length (INCR) on the master interface.	4 - 32	8
bar0	Address area 0 decoded by the bridge's slave interface. Appears as memory address register (BAR0) on the slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
bar1	Address area 1 (BAR1)	0 - 1073741823	0
bar2	Address area 2 (BAR2)	0 - 1073741823	0
bar3	Address area 3 (BAR2)	0 - 1073741823	0
sbus	The number of the AHB bus to which the slave interface is connected. The value appears in bits [1:0] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
mbus	The number of the AHB bus to which the master interface is connected. The value appears in bits [3:2] of the user-defined register 0 in the slave interface configuration record and master configuration record.	0-3	0
ioarea	Address to configuration area for the master interface AHB bus. Appears in the bridge's slave interface user-defined register 1.	0 - 16#FFF#	0
ibrsten	Instruction fetch burst enable. If set, bridge will perform bursts of *iburst* length for opcode access (HPROT[0] = '0'), otherwise bursts of *rburst* length will be used both for data and opcode access.	0 - 1	0

2.7 Signal descriptions

Table 19 shows the interface signals of the core (VHDL ports).

Table 19. Signal descriptions (VHDL ports)

Signal name	Field	Type	Function	Active
RST		Input	Reset	Low
HCLKM		Input	AHB master bus clock	-
HCLKS		Input	AHB slave bus clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSO2	*	Input	AHB slave input vector signals (on master i/f side). Used to decode cachability and prefetch-ability Plug&Play information on master bus.	-
LCKI	slck blck	Input	Used in systems with multiple AHB/AHB bridges (e.g. bi-directional AHB/AHB bridge) to detect dead-lock condition. Tie to “00” in systems with only uni-directional AHB/AHB bus.	High
LCKO	slck blck	Output	Indicates possible dead-lock condition	High

* see GRLIB IP Library User’s Manual

2.8 Library dependencies

Table 20 shows the libraries used when instantiating the core (VHDL libraries).

Table 20. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

2.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity ahb2ahb_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of ahb2ahb_ex is
  constant NCPU : integer := 4;
  -- AMBA signals, maser bus
signal ahbmi1 : ahb_mst_in_type;
  signal ahbmo_m : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi_m : ahb_slv_in_type;
  signal ahbso_m : ahb_slv_out_type := (others => ahbs_none);
-- AMBA signals, slave bus
signal ahbmi_s : ahb_mst_in_type;
  signal ahbmo_s : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi_s : ahb_slv_in_type;
  signal ahbso_s : ahb_slv_out_type := (others => ahbs_none);
signal ahbsov_s : ahb_slv_out_vector;
signal nolock : ahb2ahb_ctrl_type;
begin

nolock <= (others => ‘0’);

  -- AHB/AHB uni-directional bridge
ahb2ahb0 : ahb2ahb generic map (
    hsindex  => 1,
    hmindex  => 3,
    dir      => 0,
    slv      => 0,
    ffact    => 2,
    memtech => memtech,
    pfen     => 1,
    irqsync  => 1,
    rbufsz   => 16,
    wbufsz   => 8,
    iburst   => 4,
    rburst   => 16,
    bar0     => ahb2ahb_membar(16#600#, ’1’, ’1’, 16#E00#),
    bar1     => ahb2ahb_iobar(16#800#, 16#800#),
    bar2     => 0,
    bar3     => 0,
    sbus     => 0,
    mbus     => 1,
    ioarea   => hsb_ioarea,
    ibrsten  => 0)
  port map (
    hclkm  => hclkm,
    hclks  => hclks,
    rstn   => rstn,
    ahbsi  => ahbsi_S,
    ahbso  => ahbso_s,
    ahbmi  => ahbmi_m,
    ahbmo  => ahbmo_m,
    ahbso2 => ahbsov_m,
    lcki   => nolock,
    lcko   => open);

end;
```

3 AHBBRIDGE - Bi-directional AHB/AHB bridge

3.1 Overview

A pair of uni-directional bridges (AHB2AHB) can be instantiated to form a bi-directional bridge. The bi-directional AHB/AHB bridge (AHBBRIDGE) instantiates two uni-directional bridges which are configured to suite bus architecture shown in figure 2. The bus architecture consists of two AHB buses: a high-speed AHB bus hosting LEON3 CPU(s) and external memory controller and a low-speed AHB bus hosting communication IP-cores. For other bus architectures, a more general bi-directional bridge can be created by instantiating two uni-directional AHB to AHB bridges (see AHB2AHB core).

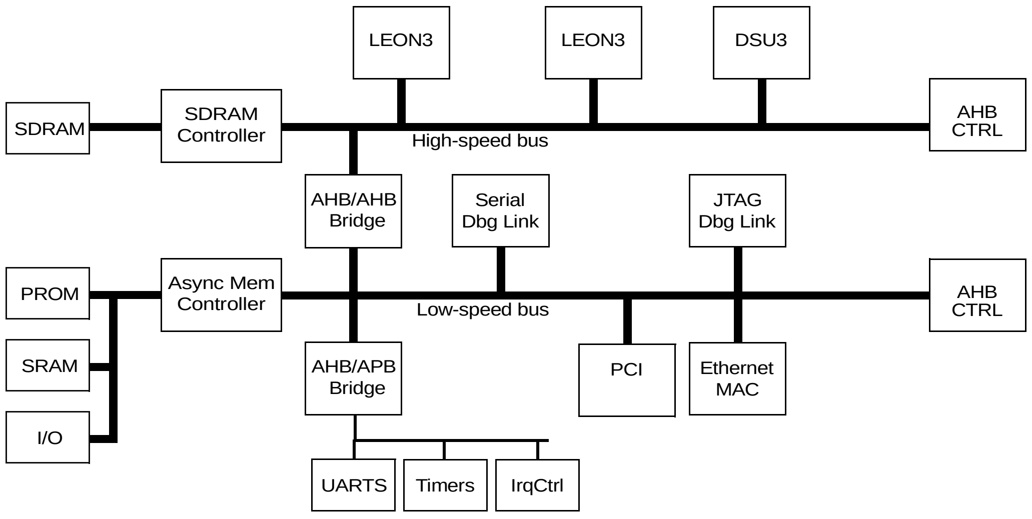


Figure 2. LEON3 system with a bi-directional AHB/AHB bridge

3.2 Operation

3.2.1 General

AHB/AHB bridge is connected to each AHB bus through a pair consisting of AHB Master and AHB Slave interface. Address space occupied by the AHB/AHB bridge on each bus is determined by Bank Address Registers which are configured through VHDL-generics. Bridge is capable of handling single and burst transfers in both direction. Internal FIFOs are used for data buffering. The bridge implements SPLIT response to improve AHB bus utilization. For more information on AHB transfers please refer to uni-directional AHB/AHB bridge (AHB2AHB) documentation.

The requirements on the two bus clocks are that they are synchronous and that the high-speed bus clock runs at higher frequency then the low-speed bus clock. Allowed frequency ratio between the two clocks is two or higher. (Although AHBBRIDGE does not support bus architectures where buses are running at the same clock frequency or where CPU bus runs at lower frequency than the bus hosting communication cores, a bridge supporting such bus architecture can be created using two uni-directional AHB/AHB bridges (see AHB2AHB core)).

The two uni-directional bridges forming the bi-directional AHB/AHB bridge are configured asymmetrically. Configuration of the bridge connecting high-speed bus with the low-speed bus (down bus) is optimized for the bus traffic generated by the CPU since the CPU is the only master on the high-speed bus (except for the bridge itself). Read transfers generated by the CPU are single read transfers generated by single load instructions (LD), read bursts of length two generated by double load instructions (LDD) or incremental read bursts of maximal length equal to cache line size (4 or 8 words) generated during instruction cache line fill. The size of the read FIFO for the down bridge is therefore configurable to 4 or 8 entries which is the maximal read burst length. If a read burst is instruction fetch (indicated on AHB HPROT signal) to a prefetchable area the bridge will prefetch data to the end of a instruction cache line. If a read burst to a prefetchable area is data access two words will be prefetched (this transfer is generated by LDD instruction). The write FIFO has two entries capable of buffering the longest write burst (generated by STD instruction). The down bridge also performs interrupt forwarding, interrupt lines 1-15 on both buses are monitored and an interrupt on one bus is forwarded to the another one.

Since the low-speed bus does not host a LEON3 CPU all AHB transfers forwarded by the uni-directional bridge connecting the low-speed bus and the high-speed bus (up bridge) are data transfers. Therefore the bridge does not make distinction between instruction and data transfers. Size of read and write FIFOs for this bridge is configurable and should be set by the user to suite burst transfers generated by the cores on the low-speed bus.

A dead-lock situation can occur if the bus is simultaneously accessed from both buses. Bridge contains dead-lock detection logic which will resolve dead-lock condition by giving RETRY response on the low-speed bus.

3.3 Registers

The core does not implement any registers.

3.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x020. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

3.5 Configuration options

Table 21 shows the configuration options of the core (VHDL generics).

Table 21. Configuration options

Generic	Function	Allowed range	Default
memtech	Memory technology	-	0
ffact	Frequency ratio	2 -	2
hsb_hsindex	AHB slave index on the high-speed bus	0 to NAHBMAX-1	0
hsb_hmindex	AHB master index on the high-speed bus	0 to NAHBMAX-1	0
hsb_iclsize	Cache line size (in number of words) for CPUs on the high-speed bus. Determines the number of the words that are prefetched by the bridge when CPU performs instruction bursts.	4, 8	8
hsb_bank0	Address area 0 mapped on the high-speed bus and decoded by the bridge's slave interface on the low-speed bus. Appears as memory address register (BAR0) on the bridge's low-speed bus slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
hsb_bank1	Address area 1 mapped on the high-speed bus	0 - 1073741823	0
hsb_bank2	Address area 2 mapped on the high-speed bus	0 - 1073741823	0
hsb_bank3	Address area 3 mapped on the high-speed bus	0 - 1073741823	0
hsb_ioarea	High-speed bus configuration area will appear in the bridge's slave interface user-defined register 1 on the low-speed bus.	0 - 16#FFF#	0
lsb_hsindex	AHB slave index on the low-speed bus	0 to NAHBMAX-1	0
lsb_hmindex	AHB master index on the low-speed bus	0 to NAHBMAX-1	0
lsb_rburst	Size of the prefetch buffer for read transfers initiated on the low-speed-bus and crossing the bridge.	16, 32	16
lsb_wburst	Size of the write buffer for write transfers initiated on the low-speed bus and crossing the bridge.	16, 32	16
lsb_bank0	Address area 0 mapped on the low-speed bus and decoded by the bridge's slave interface on the high-speed bus. Appears as memory address register (BAR0) on the bridge's high-speed bus slave interface. The generic has the same bit layout as bank address registers with bits [19:18] suppressed (use functions ahb2ahb_membar and ahb2ahb_iobar in gaisler.misc package to generate this generic).	0 - 1073741823	0
lsb_bank1	Address area 1 mapped on the low-speed bus	0 - 1073741823	0
lsb_bank2	Address area 2 mapped on the low-speed bus	0 - 1073741823	0
lsb_bank3	Address area 3 mapped on the low-speed bus	0 - 1073741823	0
lsb_ioarea	Address to low-speed bus configuration area will appear in the bridge's slave interface user-defined register 1 on the high-speed bus.	0 - 16#FFF#	0

3.6 Signal descriptions

Table 22 shows the interface signals of the core (VHDL ports).

Table 22. Signal descriptions

Signal name	Type	Function	Active
RST	Input	Reset	Low
HSB_HCLK	Input	High-speed AHB clock	-
LSB_HCLK	Input	Low-speed AHB clock	-
HSB_AHBSI	Input	High-speed bus AHB slave input signals	-
HSB_AHBSO	Output	High-speed bus AHB slave output signals	-
HSB_AHBSOV	Input	High-speed bus AHB slave input signals	-
HSB_AHBMI	Input	High-speed bus AHB master input signals	-
HSB_AHBMO	Output	High-speed bus AHB master output signals	-
LSB_AHBSI	Input	Low-speed bus AHB slave input signals	-
LSB_AHBSO	Output	Low-speed bus AHB slave output signals	-
LSB_AHBSOV	Input	Low-speed bus AHB slave input signals	-
LSB_AHBMI	Input	Low-speed bus AHB master input signals	-
LSB_AHBMO	Output	Low-speed bus AHB master output signals	-

3.7 Library dependencies

Table 23 shows the libraries used when instantiating the core (VHDL libraries).

Table 23. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

4 AHBCTRL - AMBA AHB controller with plug&play support

4.1 Overview

The AMBA AHB controller is a combined AHB arbiter, bus multiplexer and slave decoder according to the AMBA 2.0 standard.

The controller supports up to 16 AHB masters, and 16 AHB slaves. The maximum number of masters and slaves are defined in the GRLIB.AMBA package, in the VHDL constants NAHBSLV and NAHBMST. It can also be set with the *nahbm* and *nahbs* VHDL generics.

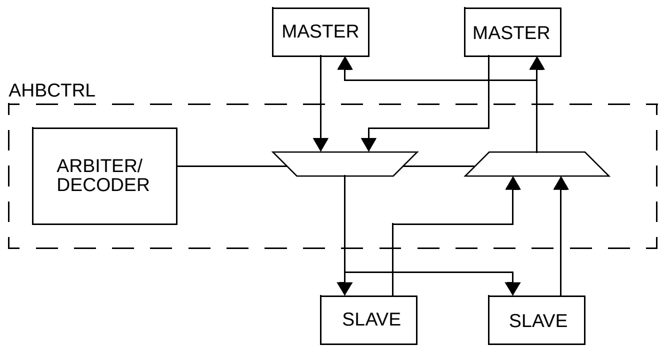


Figure 3. AHB controller block diagram

4.2 Operation

4.2.1 Arbitration

The AHB controller supports two arbitration algorithms: fixed-priority and round-robin. The selection is done by the VHDL generic *rrobin*. In fixed-priority mode (*rrobin* = 0), the bus request priority is equal to the master's bus index, with index 0 being the lowest priority. If no master requests the bus, the master with bus index 0 (set by the VHDL generic *defmast)* will be granted.

In round-robin mode, priority is rotated one step after each AHB transfer. If no master requests the bus, the last owner will be granted (bus parking). The VHDL generic *mprio* can be used to specify one or more masters that should be prioritized when the core is configured for round-robin mode.

During incremental bursts, the AHB master should keep the bus request asserted until the last access as recommended in the AMBA 2.0 specification, or it might loose bus ownership. For fixed-length burst, the AHB master will be granted the bus during the full burst, and can release the bus request immediately after the first access has started. For this to work however, the VHDL generic *fixbrst* should be set to 1.

4.2.2 Decoding

Decoding (generation of HSEL) of AHB slaves is done using the plug&play method explained in the GRLIB User's Manual. A slave can occupy any binary aligned address space with a size of 1 - 4096 Mbyte. A specific I/O area is also decoded, where slaves can occupy 256 byte - 1 Mbyte. The default address of the I/O area is 0xFFF00000, but can be changed with the *ioaddr* and *iomask* VHDL generics. Access to unused addresses will cause an AHB error response.

4.2.3 Plug&play information

GRLIB devices contain a number of plug&play information words which are included in the AHB records they drive on the bus (see the GRLIB user's manual for more information). These records are combined into an array which is connected to the AHB controller unit.

The plug&play information is mapped on a read-only address area, defined by the *cfgaddr* and *cfg-mask* VHDL generics, in combination with the *ioaddr* and *iomask* VHDL generics. By default, the area is mapped on address 0xFFFFF000 - 0xFFFFFFFF. The master information is placed on the first 2 kbyte of the block (0xFFFFF000 - 0xFFFFF800), while the slave information is placed on the second 2 kbyte block. Each unit occupies 32 bytes, which means that the area has place for 64 masters and 64 slaves. The address of the plug&play information for a certain unit is defined by its bus index. The address for masters is thus 0xFFFFF000 + n*32, and 0xFFFFF800 + n*32 for slaves.

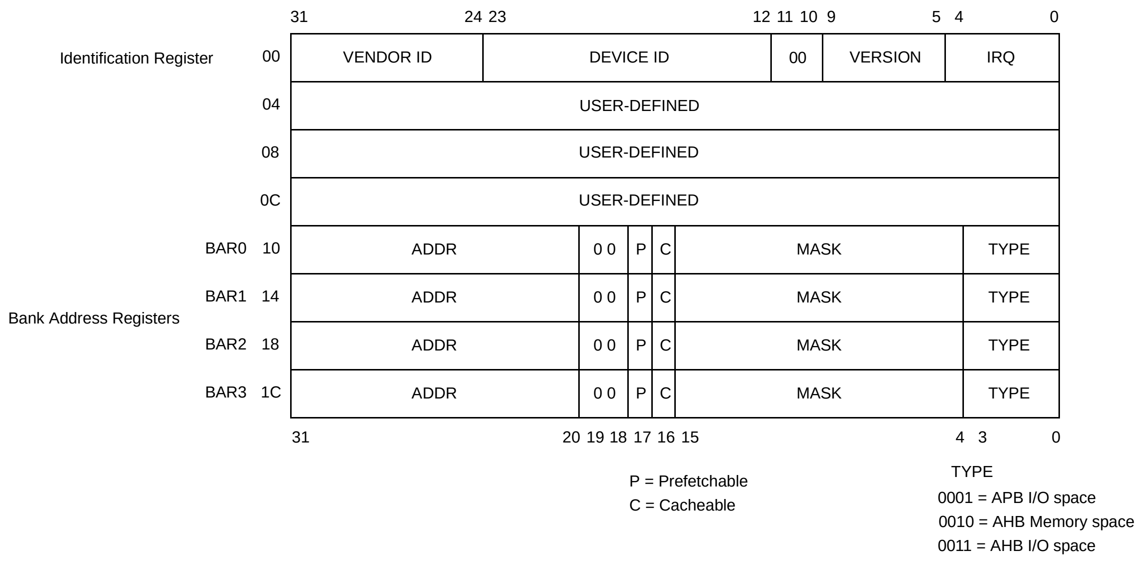


Figure 4. AHB plug&play information record

4.3 AHB split support

AHB SPLIT functionality is supported if the *split* VHDL generic is set to 1. In this case, all slaves must drive the AHB SPLIT signal.

It is important to implement the split functionality in slaves carefully since locked splits can otherwise easily lead to deadlocks. A locked access to a slave which is currently processing (it has returned a split response but not yet split complete) an access which it returned split for to another master must be handled first. This means that the slave must either be able to return an OKAY response to the locked access immediately or it has to split it but return split complete to the master performing the locked transfer before it has finished the first access which received split.

4.4 AHB bus monitor

An AHB bus monitor is integrated into the core. It is enabled with the *enbusmon* generic. It has the same functionality as the AHB and arbiter parts in the AMBA monitor core (AMBAMON). For more information on which rules are checked se the AMBAMON documentation.

4.5 AHB early burst termination

The core will perform early burst termination if the VHDL generic *enebterm* is set to 1. The core will agressively re-arbitrate all transfers which are not locked. This functionality is only intended to be used, and can only be enabled, in simulation.

4.6 Registers

The core does not implement any registers.

4.7 Configuration options

Table 24 shows the configuration options of the core (VHDL generics).

Table 24. Configuration options

Generic	Function	Allowed range	Default
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 32-bit AHB address (i.e. 31 downto 20)	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FFF#
cfgaddr	The MSB address of the configuration area. Sets 12 bits in the 32-bit AHB address (i.e. 19 downto 8).	0 - 16#FFF#	16#FF0#
cfgmask	The address mask of the configuration area. Sets the size of the configuration area and the start address together with cfgaddr. If set to 0, the configuration will be disabled.	0 - 16#FFF#	16#FF0#
rrobin	Selects between round-robin (1) or fixed-priority (0) bus arbitration algorithm.	0 - 1	0
split	Enable support for AHB SPLIT response	0 - 1	0
defmast	Default AHB master	0 - NAHBMST-1	0
ioen	AHB I/O area enable. Set to0 to disable the I/O area	0 - 1	1
nahbm	Number of AHB masters	1 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves	1 - NAHBSLV	NAHBSLV
timeout	Perform bus timeout checks (NOT IMPLEMENTED).	0 - 1	0
fixbrst	Enable support for fixed-length bursts	0 - 1	0
debug	Print configuration (0=none, 1=short, 2=all cores)	0 - 2	2
fpnpen	Enables full decoding of the PnP configuration records. When disabled the user-defined registers in the PnP configuration records are not mapped in the configuration area.	0 - 1	0
icheck	Check bus index	0 - 1	1
devid	Assign unique device identifier readable from plug and play area.	N/A	0
enbusmon	Enable AHB bus monitor	0 - 1	0
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	N/A	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	N/A	0
mprio	Master(s) with highest priority. This value is converted to a vector where each position corresponds to a master. To prioritize masters x and y set this generic to $2^x + 2^y$.	N/A	0
enebterm	Enable early burst termination, only in simulation.	0 - 1	0

4.8 Signal descriptions

Table 25 shows the interface signals of the core (VHDL ports).

Table 25. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
MSTI	*	Output	AMBA AHB master interface record array	-
MSTO	*	Input	AMBA AHB master interface record array	-
SLVI	*	Output	AMBA AHB slave interface record array	-
SLVO	*	Input	AMBA AHB slave interface record array	-

* see GRLIB IP Library User’s Manual

4.9 Library dependencies

Table 26 shows libraries used when instantiating the core (VHDL libraries).

Table 26. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

4.10 Component declaration

```
library grlib;
use grlib.amba.all;

component ahbctrl
  generic (
    defmast : integer := 0;-- default master
    split   : integer := 0;-- split support
    rrobin  : integer := 0;-- round-robin arbitration
    timeout : integer range 0 to 255 := 0;  -- HREADY timeout
    ioaddr  : ahb_addr_type := 16#fff#;  -- I/O area MSB address
    iomask  : ahb_addr_type := 16#fff#;   -- I/O area address mask
    cfgaddr : ahb_addr_type := 16#ff0#;  -- config area MSB address
    cfgmask : ahb_addr_type := 16#ff0#; -- config area address maskk
    nahbm   : integer range 1 to NAHBMST := NAHBMST; -- number of masters
    nahbs   : integer range 1 to NAHBSLV := NAHBSLV; -- number of slaves
    ioen    : integer range 0 to 15 := 1;   -- enable I/O area
    disirq  : integer range 0 to 1 := 0; -- disable interrupt routing
    fixbrst : integer range 0 to 1 := 0; -- support fix-length bursts
    debug : integer range 0 to 2 := 2; -- print configuration to consolee
    fpnpen : integer range 0 to 1 := 0;     -- full PnP configuration decoding
    icheck  : integer range 0 to 1 := 1
    devid       : integer := 0;     -- unique device ID
    enbusmon    : integer range 0 to 1 := 0; --enable bus monitor
    assertwarn  : integer range 0 to 1 := 0; --enable assertions for warnings
    asserterr   : integer range 0 to 1 := 0; --enable assertions for errors
    hmstdisable : integer := 0; --disable master checks
    hslvdisable : integer := 0; --disable slave checks
    arbdisable  : integer := 0; --disable arbiter checks
    mprio       : integer := 0; --master with highest priority
    enebterm    : integer range 0 to 1 := 0  --enable early burst termination
);
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    msti    : out ahb_mst_in_type;
    msto    : in  ahb_mst_out_vector;
    slvi    : out ahb_slv_in_type;
    slvo    : in  ahb_slv_out_vector;
    testen  : in  std_ulogic := ’0’;
    testrst : in  std_ulogic := ’1’;
    scanen  : in  std_ulogic := ’0’;
    testoen : in  std_ulogic := ’1’
  );
  end component;
```

4.11 Instantiation

This example shows the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;

.
.

  -- AMBA signals
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

-- ARBITER

ahb0 : ahbctrl -- AHB arbiter/multiplexer
  generic map (defmast => CFG_DEFMST, split => CFG_SPLIT,
rrobin => CFG_RROBIN, ioaddr => CFG_AHBIO, nahbm => 8, nahbs => 8)
  port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);


-- AHB slave

sr0 : srctrl generic map (hindex => 3)
port map (rstn, clkm, ahbsi, ahbso(3), memi, memo, sdo3);

-- AHB master

e1 : eth_oc
      generic map (mstndx => 2, slvndx => 5, ioaddr => CFG_ETHIO, irq => 12, memtech =>
memtech)
      port map (rstn, clkm, ahbsi, ahbso(5), ahbmi  => ahbmi,
ahbmo => ahbmo(2), ethi1, etho1);
...
end;
```

4.12 Debug print-out

If the debug generic is set to 2, the plug&play information of all attached AHB units are printed to the console during the start of simulation. Reporting starts by scanning the master interface array from 0 to NAHBMST - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The descriptions for these ids are obtained from the GRLIB.DEVICES package, and are then printed on standard out together with the master number. If the index check is enabled (done with a VHDL generic), the report module also checks if the hindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

This procedure is repeated for slave interfaces found in the slave interface array. It is scanned from 0 to NAHBSLV - 1 and the same information is printed and the same checks are done as for the master interfaces. In addition, the address range and memory type is checked and printed. The address information includes type, address, mask, cacheable and pre-fetchable fields. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is type, start address, size, cacheability and pre-fetchability. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. APB I/O ranges are ignored by this module.

```
# vsim -c -quiet leon3mp
VSIM 1> run
# LEON3 MP Demonstration design
# GRLIB Version 1.0.7
# Target technology: inferred,  memory library: inferred
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area disabled
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Gaisler Research        Leon3 SPARC V8 Processor
# ahbctrl: mst1: Gaisler Research        AHB Debug UART
# ahbctrl: slv0: European Space Agency   Leon2 Memory Controller
# ahbctrl:       memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl:       memory at 0x20000000, size 512 Mbyte
# ahbctrl:       memory at 0x40000000, size 1024 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Gaisler Research        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: European Space Agency   Leon2 Memory Controller
# apbctrl:       I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Gaisler Research        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Gaisler Research        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Gaisler Research        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv7: Gaisler Research        AHB Debug UART
# apbctrl:       I/O ports at 0x80000700, size 256 byte
# apbctrl: slv11: Gaisler Research        General Purpose I/O port
# apbctrl:       I/O ports at 0x80000b00, size 256 byte
# grgpio11: 8-bit GPIO Unit rev 0
# gptimer3: GR Timer Unit rev 0, 8-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1
# apbuart1: Generic UART rev 1, fifo 4, irq 2
# ahbuart7: AHB Debug UART rev 0
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 1*8 kbyte, dcache 1*8 kbyte
VSIM 2>
```

5 AHBCTRL_MB - AMBA AHB bus controller bus with support for multiple AHB buses

5.1 Overview

AMBA AHB bus controller with support for multiple AHB buses performs arbitration and bus control on one AHB bus identically to single-bus AHB controller (AHBCTRL). In addition AHBCTRL_MB can decode plug&play information for multiple AHB buses making it suitable for use in multiple AHB bus systems. The AMBA AHB bus controller supports systems with up to 4 AHB buses.

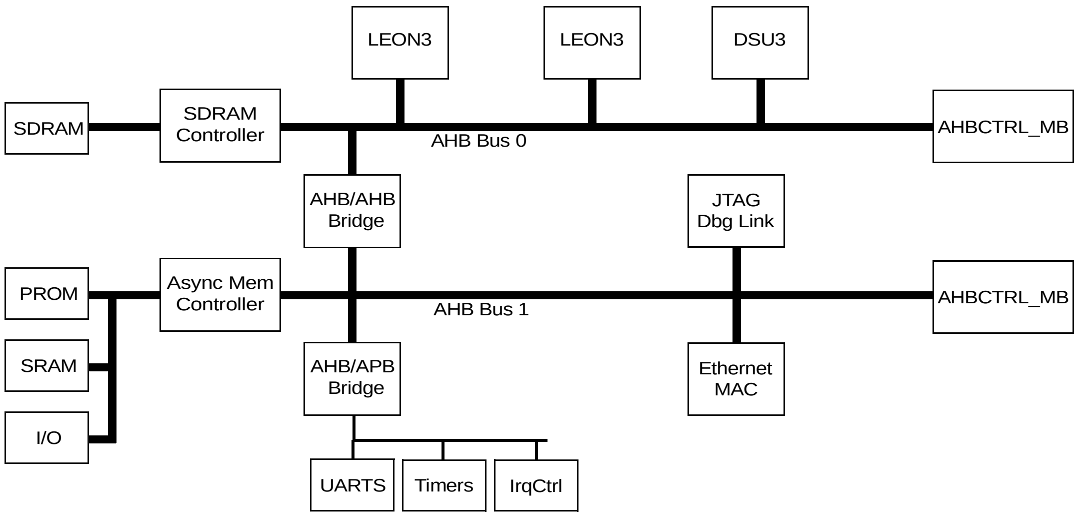


Figure 5. System with multiple AHB buses

The AMBA AHB controller with support for multiple AHB buses is a combined AHB arbiter, bus multiplexer and slave decoder according to the AMBA 2.0 standard.

The controller supports up to 16 AHB masters, and 16 AHB slaves per AHB bus. The maximum number of masters and slaves per bus are defined in the GRLIB.AMBA package, in the VHDL constants NAHBSLV and NAHBMST. It can also be set with the *nahbm* and *nahbs* VHDL generics.

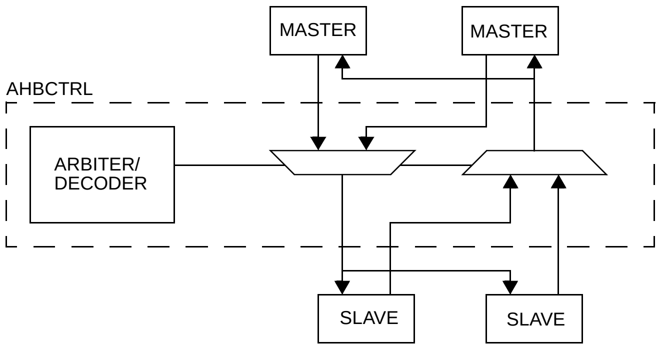


Figure 6. AHB controller block diagram

5.2 Operation

5.2.1 Arbitration

The AHB controller supports two arbitration algorithms: fixed-priority and round-robin. The selection is done by the VHDL generic *rrobin*. In fixed-priority mode (*rrobin* = 0), the bus request priority is equal to the master's bus index, with index 0 being the lowest priority. If no master requests the bus, the master with bus index 0 (set by the VHDL generic *defmast)* will be granted.

In round-robin mode, priority is rotated one step after each AHB transfer. If no master requests the bus, the last owner will be granted (bus parking).

During incremental bursts, the AHB master should keep the bus request asserted until the last access as recommended in the AMBA 2.0 specification, or it might loose bus ownership. For fixed-length burst, the AHB master will be granted the bus during the full burst, and can release the bus request immediately after the first access has started. For this to work however, the VHDL generic *fixbrst* should be set to 1.

5.2.2 Decoding

Decoding (generation of HSEL) of AHB slaves is done using the plug&play method explained in the GRLIB User's Manual. A slave can occupy any binary aligned address space with a size of 1 - 4096 Mbyte. A specific I/O area is also decoded, where slaves can occupy 256 byte - 1 Mbyte. The default address of the I/O area is 0xFFF00000, but can be changed with the *cfgaddr* and *cfgmask* VHDL generics. Access to unused addresses will cause an AHB error response.

5.2.3 Plug&play information

GRLIB devices contain a number of plug&play information words which are included in the AHB records they drive on the bus (see the GRLIB user's manual for more information). These records are combined into an array which is connected to the AHB controller unit.

The plug&play information is mapped on a read-only address area, defined by the *cfgaddr* and *cfg-mask* VHDL generics. By default, the area is mapped on address 0xFFFFF000 - 0xFFFFFFFF and contains information for all buses in the system. The master information is placed on the first 2 kbyte of the block (0xFFFFF000 - 0xFFFFF800), while the slave information is placed on the second 2 kbyte block. The master and slave information area contains plug&play information for all AHB buses in the system where each bus occupies 512 bytes (masters on bus 0 occupy 0xFFFFF000 - 0xFFFFF200, masters on bus 1 occupy 0xFFFFF200 - 0xFFFFF400, ...). Each unit occupies 32 bytes, which means that the area has place for 16 masters and 16 slaves per bus (64 master and 64 slaves totally). The address of the plug&play information for a certain unit is defined by the AHB bus and its master/slave index. The address for masters is thus 0xFFFFF000 + *bus**512 + *n**32, and 0xFFFFF800 + *bus**512 + *n**32 for slaves (where *bus* is bus number and *n* is master/slave index). In a multiple AHB bus system one AHB controller should decode plug&play information (e.g. AHBCTRL_MB on bus 0) while configuration area should be disabled for the AHB controllers on other buses.

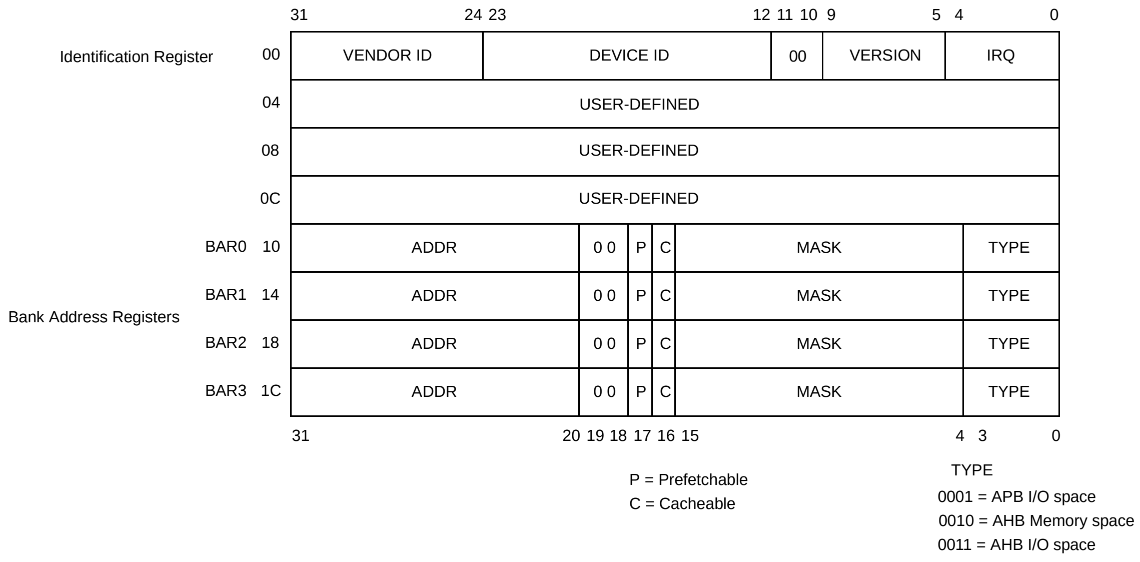


Figure 7. AHB plug&play information record

5.3 AHB split support

AHB SPLIT functionality is supported if the *split* VHDL generic is set to 1. In this case, all slaves must drive the AHB SPLIT signal.

It is important to implement the split functionality in slaves carefully since locked splits can otherwise easily lead to deadlocks. A locked access to a slave which is currently processing (it has returned a split response but not yet split complete) an access which it returned split for to another master must be handled first. This means that the slave must either be able to return an OKAY response to the locked access immediately or it has to split it but return split complete to the master performing the locked transfer before it has finished the first access which received split.

5.4 Registers

The core does not implement any registers.

5.5 Configuration options

Table 27 shows the configuration options of the core (VHDL generics).

Table 27. Configuration options

Generic	**Function**	**Allowed range**	**Default**
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FFF#
cfgaddr	The MSB address of the configuration area.	0 - 16#FFF#	16#FF0#
cfgmask	The address mask of the configuration area. Sets the size of the configuration area and the start address together with cfgaddr. If set to 0, the configuration will be disabled.	0 - 16#FFF#	16#FF0#
rrobin	Selects between round-robin (1) or fixed-priority (0) bus arbitration algorithm.	0 - 1	0
split	Enable support for AHB SPLIT response	0 - 1	0
defmast	Default AHB master	0 - NAHBMST-1	0
ioen	AHB I/O area enable. Set ot 0 to disable the I/O area	0 - 1	1
nahbm	Number of AHB masters	1 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves	1 - NAHBSLV	NAHBSLV
timeout	Perform bus timeout checks (NOT IMPLEMENTED).	0 - 1	0
fixbrst	Enable support for fixed-length bursts	0 - 1	0
debug	Print configuration (0=none, 1=short, 2=all cores)	0 - 2	2
fpnpen	Enables full decoding of the PnP configuration records. When disabled the user-defined registers in the PnP configuration records are not mapped in the configuration area.	0 - 1	0
icheck	Check bus index	0 - 1	1
busndx	AHB bus number	0 - 3	0
devid	Assign unique device identifier readable from plug and play area.	N/A	0
enbusmon	Enable AHB bus monitor	0 - 1	0
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	N/A	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	N/A	0
mprio	Master(s) with highest priority. This value is converted to a vector where each position corresponds to a master. To prioritize masters x and y set this generic to $2^x + 2^y$.	N/A	0
enebterm	Enable early burst termination, only in simulation.	0 - 1	0

5.6 Signal descriptions

Table 28 shows the interface signals of the core (VHDL ports).

Table 28. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
MSTI	*	Output	AMBA AHB master interface record array	-
MSTO	*	Input	AMBA AHB master interface record array (vector)	-
SLVI	*	Output	AMBA AHB slave interface record array	-
SLVO	*	Input	AMBA AHB slave interface record array (vector)	-

* see GRLIB IP Library User’s Manual

5.7 Library dependencies

Table 29 shows libraries used when instantiating the core (VHDL libraries).

Table 29. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

5.8 Component declaration

```
library grlib;
use grlib.amba.all;

component ahbctrl
  generic (
    defmast : integer := 0;-- default master
    split   : integer := 0;-- split support
    rrobin  : integer := 0;-- round-robin arbitration
    timeout : integer range 0 to 255 := 0;  -- HREADY timeout
    ioaddr  : ahb_addr_type := 16#fff#;  -- I/O area MSB address
    iomask  : ahb_addr_type := 16#fff#;   -- I/O area address mask
    cfgaddr : ahb_addr_type := 16#ff0#;  -- config area MSB address
    cfgmask : ahb_addr_type := 16#ff0#; -- config area address maskk
    nahbm   : integer range 1 to NAHBMST := NAHBMST; -- number of masters
    nahbs   : integer range 1 to NAHBSLV := NAHBSLV; -- number of slaves
    ioen    : integer range 0 to 15 := 1;   -- enable I/O area
    disirq  : integer range 0 to 1 := 0; -- disable interrupt routing
fixbrst : integer range 0 to 1 := 0; -- support fix-length bursts
debug : integer range 0 to 2 := 2; -- print configuration to consolee
    fpnpen : integer range 0 to 1 := 0;     -- full PnP configuration decoding
busndx  : integer range 0 to 3 := 0;
    icheck  : integer range 0 to 1 := 1
);
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    msti    : out ahb_mst_in_type;
    msto    : in  ahb_mst_out_bus_vector;
    slvi    : out ahb_slv_in_type;
    slvo    : in  ahb_slv_out_bus_vector
  );
  end component;
```

5.9 Instantiation

This example shows how the core can be instantiated. Note that master and slave outputs from all AHB buses in the systems are combined into vectors *ahbmo* and *ahbso* where each element holds master or slave outputs on one AHB bus.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;

.
.

  -- AMBA signals
  signal ahbsi0, ahbsi1 : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_bus_vector := (others => (others => ahbs_none));
  signal ahbmi1, ahbmi0: ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_bus_vector := (others => (others => ahbm_none));

begin

-- ARBITERS/MULTIPLEXERS
hsahb0 : ahbctrl_mb -- AHB arbiter/multiplexer
  generic map (defmast => CFG_NCPU+1, split => 1, rrobin => CFG_RROBIN, ioaddr => CFG_AHBIO,
               ioen => 1, nahbm => CFG_NCPU+2, nahbs => 3, fpnpen => 1)
  port map (rstn, clkf, ahbmi0, ahbmo, ahbsi0, ahbso);

  lsahb0 :   ahbctrl_mb-- AHB arbiter/multiplexer
  generic map (defmast => maxahbm-1, split => 1, rrobin => CFG_RROBIN, ioaddr => 16#FFF#,
              iomask => 16#FFF#, ioen => 1, nahbm => maxahbm, nahbs => 8, busndx => 1,
              cfgmask => 0, fpnpen => 1)
  port map (rstn, clks, ahbmi1, ahbmo, ahbsi1, ahbso);

-- AHB slave
sr0 : srctrl generic map (hindex => 3)
port map (rstn, clkm, ahbsi0, ahbso(0)(3), memi, memo, sdo3);

-- AHB master
e1 : eth_oc
      generic map (mstndx => 2, slvndx => 5, ioaddr => CFG_ETHIO, irq => 12,
                   memtech => memtech)
      port map (rstn, clkm, ahbsi0, ahbso(0)(5), ahbmi  => ahbmi0,
                ahbmo => ahbmo(0)(2), ethi1, etho1);
...
end;
```

5.10 Debug print-out

If the debug generic is set to 2, the plug&play information of all attached AHB units on the current AHB bus are printed to the console during the start of simulation. Reporting starts by scanning the master interface array from 0 to NAHBMST - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The descriptions for these ids are obtained from the GRLIB.DEVICES package, and are then printed on standard out together with the master number. If the index check is enabled (done with a VHDL generic), the report module also checks if the hindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

This procedure is repeated for slave interfaces found in the slave interface array. It is scanned from 0 to NAHBSLV - 1 and the same information is printed and the same checks are done as for the master interfaces. In addition, the address range and memory type is checked and printed. The address information includes type, address, mask, cacheable and pre-fetchable fields. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is type, start address, size, cacheability and pre-fetchability. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. APB I/O ranges are ignored by this module.

```
# vsim -c -quiet leon3mp
VSIM 1> run
# LEON3 MP Demonstration design
# GRLIB Version 1.0.7
# Target technology: inferred,  memory library: inferred
# ahbctrl: AHB arbiter/multiplexer rev 1
# ahbctrl: Common I/O area disabled
# ahbctrl: Configuration area at 0xfffff000, 4 kbyte
# ahbctrl: mst0: Gaisler Research        Leon3 SPARC V8 Processor
# ahbctrl: mst1: Gaisler Research        AHB Debug UART
# ahbctrl: slv0: European Space Agency   Leon2 Memory Controller
# ahbctrl:       memory at 0x00000000, size 512 Mbyte, cacheable, prefetch
# ahbctrl:       memory at 0x20000000, size 512 Mbyte
# ahbctrl:       memory at 0x40000000, size 1024 Mbyte, cacheable, prefetch
# ahbctrl: slv1: Gaisler Research        AHB/APB Bridge
# ahbctrl:       memory at 0x80000000, size 1 Mbyte
# apbctrl: APB Bridge at 0x80000000 rev 1
# apbctrl: slv0: European Space Agency   Leon2 Memory Controller
# apbctrl:       I/O ports at 0x80000000, size 256 byte
# apbctrl: slv1: Gaisler Research        Generic UART
# apbctrl:       I/O ports at 0x80000100, size 256 byte
# apbctrl: slv2: Gaisler Research        Multi-processor Interrupt Ctrl.
# apbctrl:       I/O ports at 0x80000200, size 256 byte
# apbctrl: slv3: Gaisler Research        Modular Timer Unit
# apbctrl:       I/O ports at 0x80000300, size 256 byte
# apbctrl: slv7: Gaisler Research        AHB Debug UART
# apbctrl:       I/O ports at 0x80000700, size 256 byte
# apbctrl: slv11: Gaisler Research        General Purpose I/O port
# apbctrl:       I/O ports at 0x80000b00, size 256 byte
# grgpio11: 8-bit GPIO Unit rev 0
# gptimer3: GR Timer Unit rev 0, 8-bit scaler, 2 32-bit timers, irq 8
# irqmp: Multi-processor Interrupt Controller rev 3, #cpu 1
# apbuart1: Generic UART rev 1, fifo 4, irq 2
# ahbuart7: AHB Debug UART rev 0
# leon3_0: LEON3 SPARC V8 processor rev 0
# leon3_0: icache 1*8 kbyte, dcache 1*8 kbyte
VSIM 2>
```

6 AHBJTAG - JTAG Debug Link with AHB Master Interface

6.1 Overview

The JTAG debug interface provides access to on-chip AMBA AHB bus through JTAG. The JTAG debug interface implements a simple protocol which translates JTAG instructions to AHB transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus.

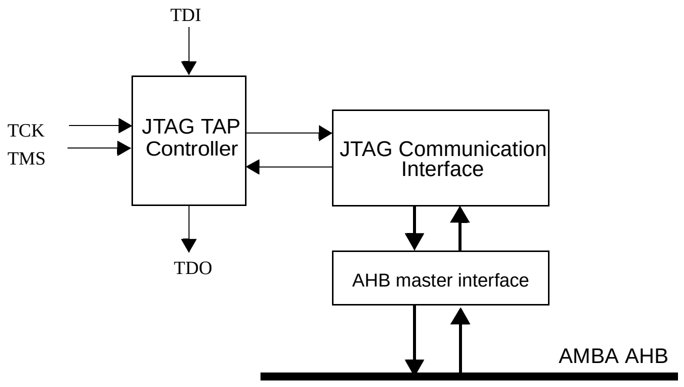


Figure 8. JTAG Debug link block diagram

6.2 Operation

6.2.1 Transmission protocol

The JTAG Debug link decodes two JTAG instructions and implements two JTAG data registers: the command/address register and data register. A read access is initiated by shifting in a command consisting of read/write bit, AHB access size and AHB address into the command/address register. The AHB read access is performed and data is ready to be shifted out of the data register. Write access is performed by shifting in command, AHB size and AHB address into the command/data register followed by shifting in write data into the data register. Sequential transfers can be performed by shifting in command and address for the transfer start address and shifting in SEQ bit in data register for following accesses. The SEQ bit will increment the AHB address for the subsequent access. Sequential transfers should not cross a 1 kB boundary. Sequential transfers are always word based.

Table 30. JTAG debug link Command/Address register

34	33 32	31 … 0
W	SIZE	AHB ADDRESS

34	Write (W) - ‘0’ - read transfer, ‘1’ - write transfer
33 32	AHB transfer size - “00” - byte, “01” - half-word, “10” - word, “11”- reserved
31 30	AHB address

Table 31. JTAG debug link Data register

32	31 … 0
SEQ	AHB DATA

32	Sequential transfer (SEQ) - If ‘1’ is shifted in this bit position when read data is shifted out or write data shifted in, the subsequent transfer will be to next word address.
31 30	AHB Data - AHB write/read data. For byte and half-word transfers data is aligned according to big-endian order where data with address offset 0 data is placed in MSB bits.

6.3 Registers

The core does not implement any registers mapped in the AMBA AHB or APB address space.

6.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x01C. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

6.5 Configuration options

Table 32 shows the configuration options of the core (VHDL generics).

Table 32. Configuration options

Generic	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
hindex	AHB master index	0 - NAHBMST-1	0
nsync	Number of synchronization registers between clock regions	1 - 2	1
idcode	JTAG IDCODE instruction code (generic tech only)	0 - 255	9
manf	Manufacturer id. Appears as bits 11-1 in TAP controllers device identification register. Used only for generic technology. Default is Gaisler Research manufacturer id.	0 - 2047	804
part	Part number (generic tech only). Bits 27-12 in device id. reg.	0 - 65535	0
ver	Version number (generic tech only). Bits 31-28 in device id. reg.	0 - 15	0
ainst	Code of the JTAG instruction used to access JTAG Debug link command/address register	0 - 255	2
dinst	Code of the JTAG instruction used to access JTAG Debug link data register	0 - 255	3
scantest	Enable scan test support	0 - 1	0

6.6 Signal descriptions

Table 33 shows the interface signals of the core (VHDL ports).

Table 33. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock (AHB clock domain)	-
TCK	N/A	Input	JTAG clock*	-
TCKN	N/A	Input	Inverted JTAG clock*	-
TMS	N/A	Input	JTAG TMS signal*	High
TDI	N/A	Input	JTAG TDI signal*	High
TDO	N/A	Output	JTAG TDO signal*	High
AHBI	***	Input	AHB Master interface input	-
AHBO	***	Output	AHB Master interface output	-
TAPO_TCK	N/A	Output	TAP Controller User interface TCK signal**	High
TAPO_TDI	N/A	Output	TAP Controller User interface TDI signal**	High
TAPO_INST[7:0]	N/A	Output	TAP Controller User interface INSTsignal**	High
TAPO_RST	N/A	Output	TAP Controller User interface RST signal**	High
TAPO_CAPT	N/A	Output	TAP Controller User interface CAPT signal**	High
TAPO_SHFT	N/A	Output	TAP Controller User interface SHFT signal**	High
TAPO_UPD	N/A	Output	TAP Controller User interface UPD signal**	High
TAPI_TDO	N/A	Input	TAP Controller User interface TDO signal**	High

*) If the target technology is Xilinx or Altera the cores JTAG signals TCK, TCKN, TMS, TDI and TDO are not used. Instead the dedicated FPGA JTAG pins are used. These pins are implicitly made visible to the core through TAP controller instantiation.

**) User interface signals from the JTAG TAP controller. These signals are used to interface additional user defined JTAG data registers such as boundary-scan register. For more information on the JTAG TAP controller user interface see JTAG TAP Controller IP-core documentation. If not used tie TAPI_TDO to ground and leave TAPO_* outputs unconnected.

***) see GRLIB IP Library User’s Manual

6.7 Library dependencies

Table 34 shows libraries used when instantiating the core (VHDL libraries).

Table 34. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	JTAG	Signals, component	Signals and component declaration

6.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.jtag.all;
```

```
entity ahbjtag_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- JTAG signals
    tck  : in std_ulogic;
    tms  : in std_ulogic;
    tdi  : in std_ulogic;
    tdo  : out std_ulogic
);
end;

architecture rtl of ahbjtag_ex is

  -- AMBA signals
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
signal gnd : std_ulogic;


constant clkperiod : integer := 100;

begin

gnd <= ‘0’;

  -- AMBA Components are instantiated here
  ...

-- AHB JTAG
  ahbjtag0 : ahbjtag generic map(tech => 0, hindex => 1)
 port map(rstn, clkm, tck, tckn, tms, tdi, tdo, ahbmi, ahbmo(1),
              open, open, open, open, open, open, open, gnd);


jtagproc : process
  begin
wait;
    jtagcom(tdo, tck, tms, tdi, 100, 20, 16#40000000#, true);
    wait;
   end process;

end;
```

6.9 Simulation

DSU communication over the JTAG debug link can be simulated using *jtagcom* procedure. The *jtagcom* procedure sends JTAG commands to the AHBJTAG on JTAG signals TCK, TMS, TDI and TDO. The commands read out and report the device identification code, optionally put the CPU(s) in debug mode, perform three write operations to the memory and read out the data from the memory. The JTAG test works if the generic JTAG tap controller is used and will not work with built-in TAP macros (such as Altera and Xilinx JTAG macros) since these macros don’t have visible JTAG pins. The jtagcom procedure is part of *jtagtst* package in *gaisler* library and has following declaration:

```
procedure jtagcom(signal tdo           : in std_ulogic;
                    signal tck, tms, tdi : out std_ulogic;
                    cp, start, addr      : in integer;
                          -- cp - TCK clock period in ns
                          -- start - time in us when JTAG test is started
                          -- addr - read/write operation destination address
                    haltcpu          : in boolean);
```

7 AHBRAM - Single-port RAM with AHB interface

7.1 Overview

AHBRAM implements a 32-bit wide on-chip RAM with an AHB slave interface. Memory size is configurable in binary steps through a VHDL generic. Minimum size is 1kB and maximum size is dependent on target technology and physical resources. Read accesses are zero-waitstate, write access have one waitstate. The RAM supports byte- and half-word accesses, as well as all types of AHB burst accesses. Internally, the AHBRAM instantiates four 8-bit wide SYNCRAM blocks.

7.2 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x00E. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

7.3 Configuration options

Table 35 shows the configuration options of the core (VHDL generics).

Table 35. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr.*	0 - 16#FFF#	16#FF0#
tech	Technology to implement on-chip RAM	0 - NTECH	0
kbytes	RAM size in Kbytes	target-dependent	1

7.4 Signal descriptions

Table 36 shows the interface signals of the core (VHDL ports).

Table 36. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

7.5 Library dependencies

Table 37 shows libraries used when instantiating the core (VHDL libraries).

Table 37. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

7.6 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbram
generic ( hindex : integer := 0; haddr : integer := 0; hmask : integer := 16#fff#;
tech : integer := 0; kbytes : integer := 1);
port (
rst : in std_ulogic;
clk : in std_ulogic;
ahbsi : in ahb_slv_in_type;
ahbso : out ahb_slv_out_type
);
end component;
```

7.7 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

.
.

ahbram0 : ahbram generic map (hindex => 7, haddr => CFG_AHBRADDR,
tech => CFG_MEMTECH, kbytes => 8)
    port map ( rstn, clkm, ahbsi, ahbso(7));
```

8 AHBROM - Single-port ROM with AHB interface

8.1 Overview

The AHBROM core implements a 32-bit wide on-chip ROM with an AHB slave interface. Read accesses take zero waitstates, or one waitstate if the pipeline option is enabled. The ROM supports byte- and half-word accesses, as well as all types of AHB burst accesses.

8.2 PROM generation

The AHBPROM is automatically generated by the make utility in GRLIB. The input format is a sparc-elf binary file, produced by the BCC cross-compiler (sparc-elf-gcc). To create a PROM, first compile a suitable binary and the run the make utility:

```
bash$ sparc-elf-gcc prom.S -o prom.exe
bash$ make ahbrom.vhd

Creating ahbrom.vhd : file size 272 bytes, address bits 9
```

The default binary file for creating a PROM is prom.exe. To use a different file, run make with the FILE parameter set to the input file:

```
bash$ make ahbrom.vhd FILE=myfile.exe
```

The created PROM is realized in synthesizable VHDL code, using a CASE statement. For FPGA targets, most synthesis tools will map the CASE statement on a block RAM/ROM if available. For ASIC implementations, the ROM will be synthesized as gates. It is then recommended to use the *pipe* option to improve the timing.

8.3 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x01B. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

8.4 Configuration options

Table 38 shows the configuration options of the core (VHDL generics).

Table 38. Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB slave bus index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with *haddr.*	0 - 16#FFF#	16#FF0#
tech	Not used		
pipe	Add a pipeline stage on read data	0	0
kbytes	Not used		

8.5 Signal descriptions

Table 39 shows the interface signals of the core (VHDL ports).

Table 39. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AMB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User's Manual

8.6 Library dependencies

Table 40 shows libraries used when instantiating the core (VHDL libraries).

Table 40. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

8.7 Component declaration

```
component ahbrom
generic ( hindex : integer := 0; haddr : integer := 0; hmask : integer := 16#fff#;
pipe : integer := 0; tech : integer := 0);
port (
rst : in std_ulogic;
clk : in std_ulogic;
ahbsi : in ahb_slv_in_type;
ahbso : out ahb_slv_out_type
);
end component;
```

8.8 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
.
.

brom : entity work.ahbrom
      generic map (hindex => 8, haddr => CFG_AHBRODDR, pipe => CFG_AHBROPIP)
      port map ( rstn, clkm, ahbsi, ahbso(8));
```

9 AHBSTAT - AHB Status Registers

9.1 Overview

The status registers store information about AMBA AHB accesses triggering an error response. There is a status register and a failing address register capturing the control and address signal values of a failing AMBA bus transaction, or the occurence of a correctable error being signaled from a fault tolerant core.

The status register and the failing address register are accessed from the AMBA APB bus.

9.2 Operation

The registers monitor AMBA AHB bus transactions and store the current HADDR, HWRITE, HMASTER and HSIZE internally. The monitoring are always active after startup and reset until an error response (HRESP = “01”) is detected. When the error is detected, the status and address register contents are frozen and the New Error (NE) bit is set to one. At the same time an interrupt is generated.

The interrupt is generated on the line selected by the *pirq* VHDL generic.

The interrupt is usually connected to the interrupt controller to inform the processor of the error condition. The normal procedure is that an interrupt routine handles the error with the aid of the information in the status registers. When it is finished it resets the NE bit and the monitoring becomes active again.

Not only error responses on the AHB bus can be detected. Many of the fault tolerant units containing EDAC have a correctable error signal which is asserted each time a single error is detected. When such an error is detected, the effect will be the same as for an AHB error response, The only difference is that the Correctable Error (CE) bit in the status register is set to one when a single error is detected. When the CE bit is set the interrupt routine can acquire the address containing the single error from the failing address register and correct it. When it is finished it resets the CE bit and the monitoring becomes active again.

The correctable error signals from the fault tolerant units should be connected to the *stati.cerror* input signal vector of the AHB status register core, which is or-ed internally and if the resulting signal is asserted, it will have the same effect as an AHB error response.

9.3 Registers

The core is programmed through registers mapped into APB address space.

Table 41. AHB Status registers

APB address offset	Registers
0x0	AHB Status register
0x4	AHB Failing address register

Table 42. AHB Status register

31 10	9	8	7	6 3	2 0
RESERVED	CE	NE	HWRITE	HMASTER	HSIZE

31: 10	RESERVED
9	CE: Correctable Error. Set if the detected error was caused by a single error and zero otherwise.
8	NE: New Error. Deasserted at start-up and after reset. Asserted when an error is detected. Reset by writing a zero to it.

Table 42. AHB Status register

7	The HWRITE signal of the AHB transaction that caused the error.
6: 3	The HMASTER signal of the AHB transaction that caused the error.
2: 0	The HSIZE signal of the AHB transaction that caused the error

Table 43. AHB Failing address register

31 … 0
AHB FAILING ADDRESS

31: 0	The HADDR signal of the AHB transaction that caused the error.

9.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x052. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

9.5 Configuration options

Table 44 shows the configuration options of the core (VHDL generics).

Table 44. Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAHBSLV-1	0
paddr	APB address	0 - 16#FFF#	0
pmask	APB address mask	0 - 16#FFF#	16#FFF#
pirq	Interrupt line driven by the core	0 - 16#FFF#	0
nftslv	Number of FT slaves connected to the cerror vector	1 - NAHBSLV-1	3

9.6 Signal descriptions

Table 45 shows the interface signals of the core (VHDL ports).

Table 45. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB slave input signals	-
AHBSI	*	Input	AHB slave output signals	-
STATI	CERROR	Input	Correctable Error Signals	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-

* see GRLIB IP Library User’s Manual

9.7 Library dependencies

Table 46 shows libraries used when instantiating the core (VHDL libraries).

Table 46. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MISC	Component	Component declaration

9.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the status register. There are three Fault Tolerant units with EDAC connected to the status register *cerror* vector. The connection of the different memory controllers to external memory is not shown.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;
  --other signals
  ....
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo, sdo2: sdctrl_out_type;

  signal sdi : sdctrl_in_type;

-- correctable error vector
  signal stati : ahbstat_in_type;
  signal aramo : ahbram_out_type;

begin

  -- AMBA Components are defined here ...

-- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(13));
    stati.cerror(3 to NAHBSLV-1) <= (others => ‘0’);

--FT AHB RAM
  a0 : ftahbram generic map(hindex => 1, haddr => 1, tech => inferred,
    kbytes => 64, pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
    errcnt => 1, cntbits => 4)
    port map(rst, clk, ahbsi, ahbso, apbi, apbo(4), aramo);
    stati.cerror(0) <= aramo.ce;
-- SDRAM controller
  sdc : ftsdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
    cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(3), sdi, sdo);
    stati.cerror(1) <= sdo.ce;

-- Memory controller
  mctrl0 : ftsrctrl generic map (rmw => 1, pindex => 10, paddr => 10,
    edacen => 1, errcnt => 1, cntbits => 4)
    port map (rstn, clk, ahbsi, ahbso(0), apbi, apbo(10), memi, memo, sdo2);
    stati.cerror(2) <= memo.ce;
end;
```

10 AHBTRACE - AHB Trace buffer

10.1 Overview

The trace buffer consists of a circular buffer that stores AMBA AHB data transfers. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis.

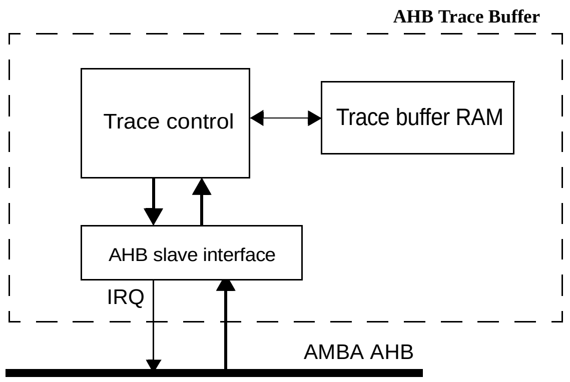


Figure 9. Block diagram

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

Table 47. AHB Trace buffer data allocation

Bits	**Name**	**Definition**
127:96	Time tag	The value of the time tag counter
95	AHB breakpoint hit	Set to '1' if a DSU AHB breakpoint hit occurred.
94:80	Hirq	AHB HIRQ[15:1]
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

In addition to the AHB signals, a 32-bit counter is also stored in the trace as time tag.

10.2 Operation

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AMBA AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is written is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. An interrupt is generated when a breakpoint is hit.

Note: the LEON3 Debug Support Unit (DSU3) also includes an AHB trace buffer. The trace buffer is intended to be used in system without the LEON3 processor or when the DSU3 is not present.

The size of the trace buffer is configured by means of the *kbytes* VHDL generic, defining the size of the complete buffer in kbytes.

The size of the trace buffer is TBD kbyte, with the resulting line depth of TBD/16 kbyte.

10.3 Registers

10.3.1 Register address map

The trace buffer occupies 128 kbyte address space in the AHB I/O area. The following register address are decoded:.

Table 48. Trace buffer address space

Address	**Register**
0x000000	Trace buffer control register
0x000004	Trace buffer index register
0x000008	Time tag counter
0x000010	AHB break address 1
0x000014	AHB mask 1
0x000018	AHB break address 2
0x00001C	AHB mask 2
0x010000 - 0x020000	Trace buffer
..0	Trace bits 127 - 96
...4	Trace bits 95 - 64
...8	Trace bits 63 - 32
...C	Trace bits 31 - 0

10.3.2 Trace buffer control register

The trace buffer is controlled by the trace buffer control register:

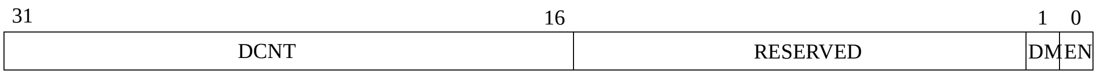


Figure 10. Trace buffer control register

0:	Trace enable (EN). Enables the trace buffer.
1:	Delay counter mode (DM). Indicates that the trace buffer is in delay counter mode.
31:16	Trace buffer delay counter (DCNT). Note that the number of bits actually implemented depends on the size of the trace buffer.

10.3.3 Trace buffer index register

The trace buffer index register indicates the address of the next 128-bit line to be written.

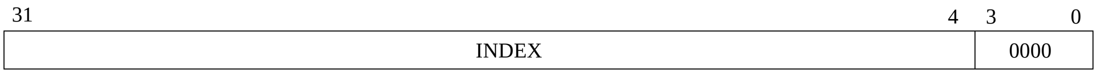


Figure 11. Trace buffer index register

31:4	Trace buffer index counter (INDEX). Note that the number of bits actually implemented depends on the size of the trace buffer.

10.3.4 Trace buffer time tag register

The time tag register contains a 32-bit counter that increments each clock when the trace buffer is enabled. The value of the counter is stored in the trace to provide a time tag.

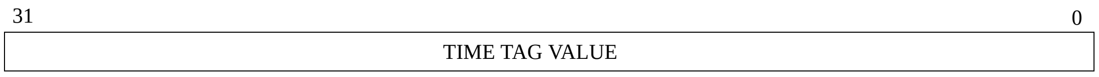


Figure 12. Trace buffer time tag counter

10.3.5 Trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. 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 AHB load or store accesses, the LD and/or ST bits should be set.

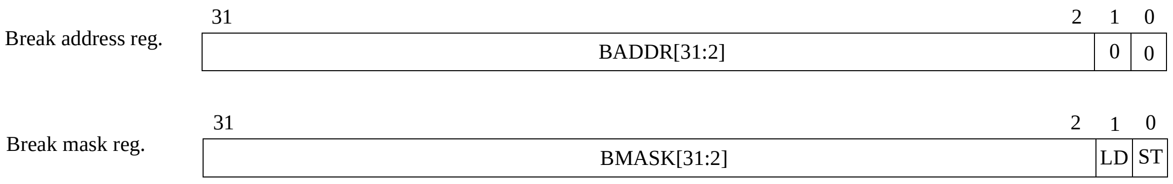


Figure 13. Trace buffer breakpoint registers

BADDR : breakpoint address (bits 31:2)
BMASK : Breakpoint mask (see text)
LD : break on data load address
ST : beak on data store address

10.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x017. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

10.5 Configuration options

Table 49 shows the configuration options of the core (VHDL generics).

Table 49. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave bus index	0 - NAHBSLV-1	0
ioaddr	The MSB address of the I/O area. Sets the 12 most significant bits in the 20-bit I/O address.	0 - 16#FFF#	16#000#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#E00#
irq	Interrupt number	0 - NAHBIRQ-1	0
tech	Technology to implement on-chip RAM	0 - NTECH	0
kbytes	Trace buffer size in kbytes	1 - 64	1

10.6 Signal descriptions

Table 50 shows the interface signals of the core (VHDL ports).

Table 50. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User’s Manual

10.7 Library dependencies

Table 51 shows libraries used when instantiating the core (VHDL libraries).

Table 51. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	MISC	Component	Component declaration

10.8 Component declaration

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

component ahbtrace is
  generic (
    hindex  : integer := 0;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#E00#;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 1);
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type);
end component;
```

11 AHBUART- AMBA AHB Serial Debug Interface

11.1 Overview

The interface consists of a UART connected to the AMBA AHB bus as a master. A simple communication protocol is supported to transmit access parameters and data. Through the communication link, a read or write transfer can be generated to any address on the AMBA AHB bus.

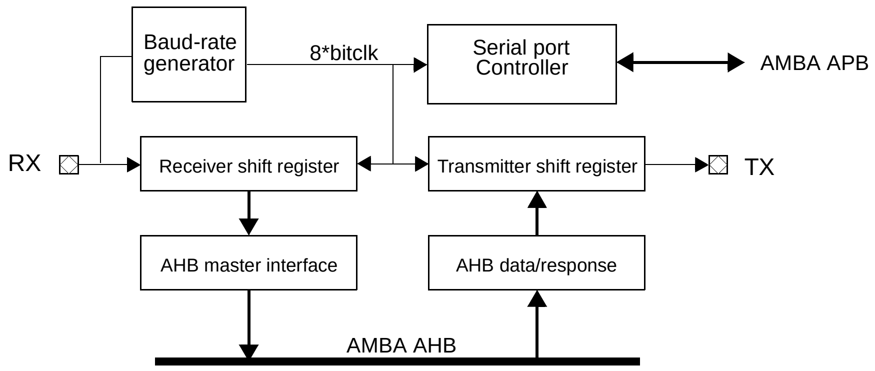


Figure 14. Block diagram

11.2 Operation

11.2.1 Transmission protocol

The interface supports a simple protocol where commands consist of a control byte, followed by a 32-bit address, followed by optional write data. 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 below.

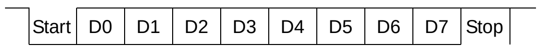


Figure 15. Data frame

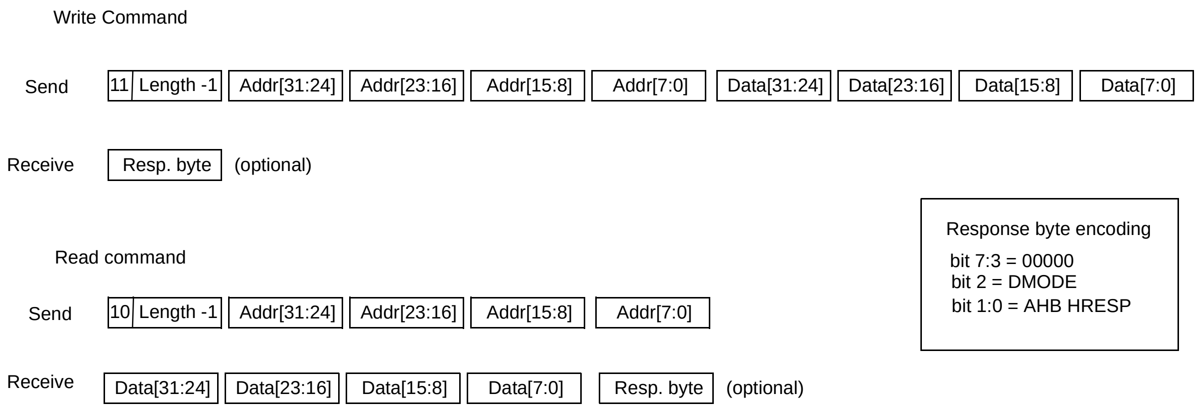


Figure 16. Commands

Block transfers can be performed be 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.

11.2.2 Baud rate generation

The UART contains a 18-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 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’ or framing error is detected 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:

```
scaler = (((system_clk*10)/(baudrate*8))-5)/10
```

11.3 Registers

The core is programmed through registers mapped into APB address space.

Table 52. AHB UART registers

APB address offset	Register
0x4	AHB UART status register
0x8	AHB UART control register
0xC	AHB UART scaler register

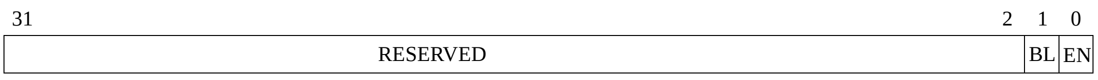


Figure 17. AHB UART control register

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.

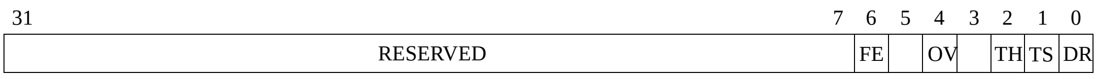


Figure 18. AHB UART status register

0: Data ready (DR) - indicates that new data has been received by the AMBA AHB master interface.
1: Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty.

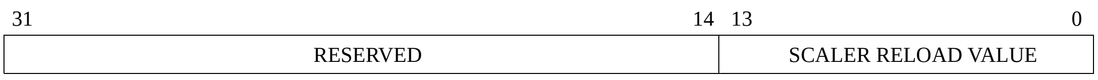


Figure 19. AHB UART scaler reload register

11.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x007. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

11.5 Configuration options

Table 53 shows the configuration options of the core (VHDL generics).

Table 53. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

11.6 Signal descriptions

Table 54 shows the interface signals of the core (VHDL ports)..

Table 54. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
UARTI	RXD	Input	UART receiver data	High
	CTSN	Input	UART clear-to-send	High
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	High
	TXD	Output	UART transmit data	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

11.7 Library dependencies

Table 55 shows libraries used when instantiating the core (VHDL libraries).

Table 55. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	UART	Signals, component	Signals and component declaration

11.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.uart.all;

entity ahbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    ahbrxd   : in  std_ulogic;
    ahbtxd   : out std_ulogic
    );
end;

architecture rtl of ahbuart_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- UART signals
  signal ahbuarti : uart_in_type;
  signal ahbuarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
  ...


  -- AHB UART
  ahbuart0 : ahbuart
  generic map (hindex => 5, pindex => 7, paddr => 7)
  port map (rstn, clk, ahbuarti, ahbuarto, apbi, apbo(7), ahbmi, ahbmo(5));

  -- AHB UART input data
  ahbuarti.rxd <= ahbrxd;

  -- connect AHB UART output to entity output signal
  ahbtxd <= ahbuarto.txd;

end;
```

12 AMBAMON - AMBA Bus Monitor

12.1 Overview

The AMBA bus monitor checks the AHB and APB buses for violations against a set of rules. When an error is detected a signal is asserted and error message is (optionally) printed.

12.2 Rules

This section lists all rules checked by the AMBA monitor. The rules are divided into four different tables depending on which type of device they apply to.

Some requirements of the AMBA specification are not adopted by the GRLIB implementation (on a system level). These requirements are listed in the table below.

Table 56. Requirements not checked in GRLIB

Rule Number	Description	References
1	A slave which issues RETRY must only be accessed by one master at a time.	AMBA Spec. Rev 2.0 3-38.

Table 57. AHB master rules.

Rule Number	Description	References
1	Busy can only occur in the middle of bursts. That is only after a NON-SEQ, SEQ or BUSY.	AMBA Spec. Rev 2.0 3-9. http://www.arm.com/support/faqip/492.html
2	Busy can only occur in the middle of bursts. It can be the last access of a burst but only for INCR bursts.	AMBA Spec. Rev 2.0 3-9. http://www.arm.com/support/faqip/492.html
3	The address and control signals must reflect the next transfer in the burst during busy cycles.	AMBA Spec. Rev 2.0 3-9.
4	The first transfer of a single access or a burst must be NONSEQ (this is ensured together with rule 1).	AMBA Spec. Rev 2.0 3-9.
5	HSIZE must never be larger than the bus width.	AMBA Spec. Rev 2.0 3-43.
6	HADDR must be aligned to the transfer size.	AMBA Spec. Rev 2.0 3-12, 3-25. http://www.arm.com/support/faqip/582.html
7	Address and controls signals can only change when hready is low if the previous HTRANS value was IDLE, BUSY or if an ERROR, SPLIT or RETRY response is given.	http://www.arm.com/support/faqip/487.html http://www.arm.com/support/faqip/579.html
8	Address and control signals cannot change between consecutive BUSY cycles.	AMBA Spec. Rev 2.0 3-9.
9	Address must be related to the previous access according to HBURST and HSIZE and control signals must be identical for SEQUENTIAL accesses.	AMBA Spec. Rev 2.0 3-9.
10	Master must cancel the following transfer when receiving an RETRY response.	AMBA Spec. Rev 2.0 3-22.
11	Master must cancel the following transfer when receiving an SPLIT response.	AMBA Spec. Rev 2.0 3-22.

Table 57. AHB master rules.

Rule Number	Description	References
12	Master must reattempt the transfer which received a RETRY response.	AMBA Spec. Rev 2.0 3-21. http://www.arm.com/support/faqip/603.html.
13	Master must reattempt the transfer which received a SPLIT response.	AMBA Spec. Rev 2.0 3-21. http://www.arm.com/support/faqip/603.html.
14	Master can optionally cancel the following transfer when receiving an ERROR response. Only a warning is given if assertions are enabled if it does not cancel the following transfer.	AMBA Spec. Rev 2.0 3-23.
15	Master must hold HWDATA stable for the whole data phase when wait states are inserted. Only the appropriate byte lanes need to be driven for subword transfers.	AMBA Spec. Rev 2.0 3-7. AMBA Spec. Rev 2.0 3-25.
16	Bursts must not cross a 1 kB address boundary.	AMBA Spec. Rev 2.0 3-11.
17	HMASTLOCK indicates that the current transfer is part of a locked sequence. It must have the same timing as address/control.	AMBA Spec. Rev 2.0 3-28.
18	HLOCK must be asserted at least one clock cycle before the address phase to which it refers.	AMBA Spec. Rev 2.0 3-28.
19	HLOCK must be asserted for the duration of a burst and can only be deasserted so that HMASTLOCK is deasserted after the final address phase.	http://www.arm.com/support/faqip/597.html
20	HLOCK must be deasserted in the last address phase of a burst.	http://www.arm.com/support/faqip/588.html
21	HTRANS must be driven to IDLE during reset.	http://www.arm.com/support/faqip/495.html
22	HTRANS can only change from IDLE to NONSEQ or stay IDLE when HREADY is deasserted.	http://www.arm.com/support/faqip/579.html

Table 58. AHB slave rules.

Rule Number	Description	References
1	AHB slave must respond with a zero wait state OKAY response to BUSY cycles in the same way as for IDLE.	AMBA Spec. Rev 2.0 3-9.
2	AHB slave must respond with a zero wait state OKAY response to IDLE.	AMBA Spec. Rev 2.0 3-9.
3	HRESP should be set to ERROR, SPLIT or RETRY only one cycle before HREADY is driven high.	AMBA Spec. Rev 2.0 3-22.
4	Two-cycle ERROR response must be given.	AMBA Spec. Rev 2.0 3-22.
5	Two-cycle SPLIT response must be given.	AMBA Spec. Rev 2.0 3-22.
6	Two-cycle RETRY response must be given.	AMBA Spec. Rev 2.0 3-22.
7	SPLIT complete signalled to master which did not have pending access.	AMBA Spec. Rev 2.0 3-36.
8	Split complete must not be signalled during same cycle as SPLIT.	http://www.arm.com/support/faqip/616.html
9	It is recommended that slaves drive HREADY high and HRESP to OKAY when not selected. A warning will be given if this is not followed.	http://www.arm.com/support/faqip/476.html
10	It is recommended that slaves do not insert more than 16 wait states. If this is violated a warning will be given if assertions are enabled.	AMBA Spec. Rev 2.0 3-20.

Table 59. APB slave rules.

Rule Number	Description	References
1	The bus must move to the SETUP state or remain in the IDLE state when in the IDLE state.	AMBA Spec. Rev 2.0 5-4.
2	The bus must move from SETUP to ENABLE in one cycle.	AMBA Spec. Rev 2.0 5-4.
3	The bus must move from ENABLE to SETUP or IDLE in one cycle.	AMBA Spec. Rev 2.0 5-5.
4	The bus must never be in another state than IDLE, SETUP, ENABLE.	AMBA Spec. Rev 2.0 5-4.
5	PADDR must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
6	PWRITE must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
7	PWDATA must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.
8	Only one PSEL must be enabled at a time.	AMBA Spec. Rev 2.0 5-4.
9	PSEL must be stable during transition from SETUP to ENABLE.	AMBA Spec. Rev 2.0 5-5.

Table 60. Arbiter rules

Rule Number	Description	References
1	HreadyIn to slaves and master must be driven by the currently selected device.	http://www.arm.com/support/faqip/482.html
2	A master which received a SPLIT response must not be granted the bus until the slave has set the corresponding HSPLIT line.	AMBA Spec. Rev 2.0 3-35.
3	The dummy master must be selected when a SPLIT response is received for a locked transfer.	http://www.arm.com/support/faqip/14307.html

12.3 Configuration options

Table 61 shows the configuration options of the core (VHDL generics).

***Table 61.* Configuration options**

Generic	Function	Allowed range	Default
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	1
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	1
hmstdisable	Disable AHB master rule check. To disable a master rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	-	0
hslvdisable	Disable AHB slave tests. Values are assigned as for hmstdisable.	-	0
pslvdisable	Disable APB slave tests. Values are assigned as for hmstdisable.	-	0
arbdisable	Disable Arbiter tests. Values are assigned as for hmstdisable.	-	0
nahbm	Number of AHB masters in the system.	0 - NAHBMST	NAHBMST
nahbs	Number of AHB slaves in the system.	0 - NAHBSLV	NAHBSLV
napb	Number of APB slaves in the system.	0 - NAPBSLV	NAPBSLV

12.4 Signal descriptions

Table 62 shows the interface signals of the core (VHDL ports).

Table 62. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBMI	*	Input	AHB master interface input record	-
AHBMO	*	Input	AHB master interface output record array	-
AHBSI	*	Input	AHB slave interface input record	-
AHBSO	*	Input	AHB slave interface output record array	-
APBI	*	Input	APB slave interface input record	
APBO	*	Input	APB slave interface output record array	
ERR	N/A	Output	Error signal (error detected)	High

* see GRLIB IP Library User’s Manual

12.5 Library dependencies

Table 63 shows libraries used when instantiating the core (VHDL libraries).

Table 63. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	SIM	Component	Component declaration

12.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.sim.all;

entity ambamon_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic
end;

architecture rtl of ambamon_ex is
-- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);



  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


begin
  -- AMBA Components are instantiated here
   ...
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.sim.all;

entity ambamon_ex is
  port (
    clk : in  std_ulogic;
    rst : in  std_ulogic;
    err : out std_ulogic
end;

architecture rtl of ambamon_ex is
  -- AHB signals
  signal ahbmi  : ahb_mst_in_type;
  signal ahbmo  : ahb_mst_out_vector := (others => apb_none);

  -- AHB signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => apb_none);

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);


begin

  mon0 : ambamon
  generic map(
    assert_err => 1,
    assert_war => 0,
    nahbm      => 2,
    nahbs      => 2,
    napb       => 1
  )
  port map(
    rst        => rst,
    clk        => clk,
    ahbmi      => ahbmi,
    ahbmo      => ahbmo,
    ahbsi      => ahbsi,
    ahbso      => ahbso,
    apbi       => apbi,
    apbo       => apbo,
    err        => err);

end;
```

13 APBCTRL - AMBA AHB/APB bridge with plug&play support

13.1 Overview

The AMBA AHB/APB bridge is a APB bus master according the AMBA 2.0 standard.

The controller supports up to 16 slaves. The actual maximum number of slaves is defined in the GRLIB.AMBA package, in the VHDL constant NAPBSLV. The number of slaves can also be set using the *nslaves* VHDL generic.

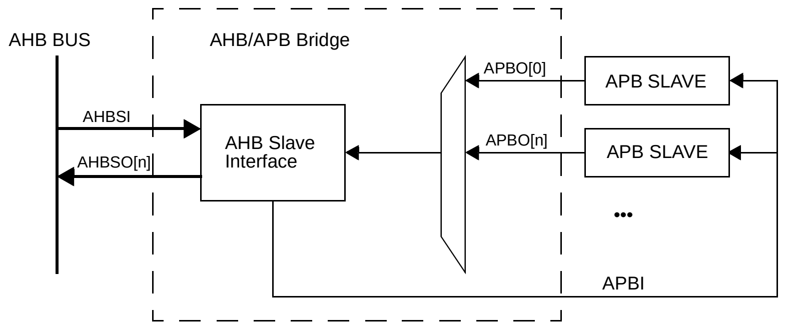


Figure 20. AHB/APB bridge block diagram

13.2 Operation

13.2.1 Decoding

Decoding (generation of PSEL) of APB slaves is done using the plug&play method explained in the GRLIB IP Library User's Manual. A slave can occupy any binary aligned address space with a size of 256 bytes - 1 Mbyte. Write to unassingned areas will be ignored, while reads from unassigned areas will return an arbitrary value. AHB error response will never be generated.

13.2.2 Plug&play information

GRLIB APB slaves contain two plug&play information words which are included in the APB records they drive on the bus (see the GRLIB IP Library User's Manual for more information). These records are combined into an array which is connected to the APB bridge.

The plug&play information is mapped on a read-only address area at the top 4 kbytes of the bridge address space. Each plug&play block occupies 8 bytes. The address of the plug&play information for a certain unit is defined by its bus index. If the bridge is mapped on AHB address 0x80000000, the address for the plug&play records is thus 0x800FF000 + n*8.

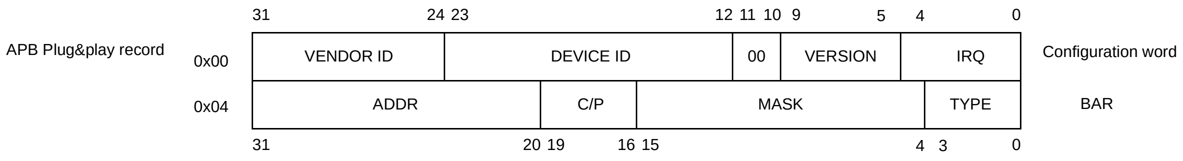


Figure 21. APB plug&play information

13.3 APB bus monitor

An APB bus monitor is integrated into the core. It is enabled with the enbusmon generic. It has the same functionality as the APB parts in the AMBA monitor core (AMBAMON). For more information on which rules are checked se the AMBAMON documentation.

13.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x006. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

13.5 Configuration options

Table 64 shows the configuration options of the core (VHDL generics).

Table 64. Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	The MSB address of the AHB area. Sets the 12 most significant bits in the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
hmask	The AHB area address mask. Sets the size of the AHB area and the start address together with haddr.	0 - 16#FFF#	16#FFF#
nslaves	The maximum number of slaves	1 - NAPBSLV	NAPBSLV
debug	Print debug information during simulation	0 - 2	2
icheck	Enable bus index checking (PINDEX)	0 - 1	1
enbusmon	Enable APB bus monitor	0 - 1	0
asserterr	Enable assertions for AMBA requirements. Violations are asserted with severity error.	0 - 1	0
assertwarn	Enable assertions for AMBA recommendations. Violations are asserted with severity warning.	0 - 1	0
pslvdisable	Disable APB slave rule check. To disable a slave rule check a value is assigned so that the binary representation contains a one at the position corresponding to the rule number, e.g 0x80 disables rule 7.	N/A	0

13.6 Signal descriptions

Table 65 shows the interface signals of the core (VHDL ports).

Table 65. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	AHB reset	Low
CLK	N/A	Input	AHB clock	-
AHBI	*	Input	AHB slave input	-
AHBO	*	Output	AHB slave output	-
APBI	*	Output	APB slave inputs	-
APBO	*	Input	APB slave outputs	-

* see GRLIB IP Library User's Manual

13.7 Library dependencies

Table 66 shows libraries used when instantiating the core (VHDL libraries).

Table 66. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions

13.8 Component declaration

```
library grlib;
use grlib.amba.all;

component apbctrl
  generic (
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#fff#;
    nslaves : integer range 1 to NAPBSLV := NAPBSLV;
    debug   : integer range 0 to 2 := 2;   -- print config to console
    icheck  : integer range 0 to 1 := 1
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbi    : in  ahb_slv_in_type;
    ahbo    : out ahb_slv_out_type;
    apbi    : out apb_slv_in_type;
    apbo    : in  apb_slv_out_vector
  );
  end component;
```

13.9 Instantiation

This example shows how an APB bridge can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use work.debug.all;

.
.

  -- AMBA signals

  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);


begin


-- APB bridge

apb0 : apbctrl-- AHB/APB bridge
  generic map (hindex => 1, haddr => CFG_APBADDR)
  port map (rstn, clk, ahbsi, ahbso(1), apbi, apbo );
```

```
-- APB slaves

uart1 : apbuart
    generic map (pindex => 1, paddr => 1,  pirq => 2)
    port map (rstn, clk, apbi, apbo(1), u1i, u1o);

irqctrl0 : irqmp
    generic map (pindex => 2, paddr => 2)
    port map (rstn, clk, apbi, apbo(2), irqo, irqi);


...
end;
```

13.10 Debug print-out

The APB bridge can print-out the plug-play information from the attached during simulation. This is enabled by setting the debug VHDL generic to 2. Reporting starts by scanning the array from 0 to NAPBSLV - 1 (defined in the grlib.amba package). It checks each entry in the array for a valid vendor-id (all nonzero ids are considered valid) and if one is found, it also retrieves the device-id. The description for these ids are obtained from the GRLIB.DEVICES package, and is printed on standard out together with the slave number. If the index check is enabled (done with a VHDL generic), the report module also checks if the pindex number returned in the record matches the array number of the record currently checked (the array index). If they do not match, the simulation is aborted and an error message is printed.

The address range and memory type is also checked and printed. The address information includes type, address and mask. The address ranges currently defined are AHB memory, AHB I/O and APB I/O. All APB devices are in the APB I/O range so the type does not have to be checked. From this information, the report module calculates the start address of the device and the size of the range. The information finally printed is start address and size.

14 APBPS2 - PS/2 keyboard with APB interface

14.1 Introduction

The PS/2 interface is a bidirectional synchronous serial bus primarily used for keyboard and mouse communications. The APBPS2 core implements the PS2 protocol with a APB back-end. Figure 22 shows a model of APBPS2 and the electrical interface.

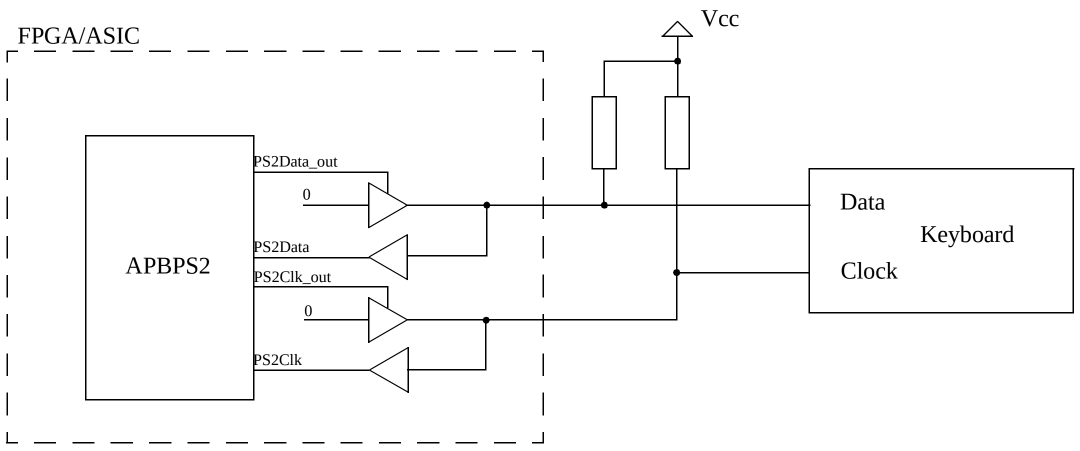


Figure 22. APBPS2 electrical interface

PS/2 data is sent in 11 bits frames. The first bit is a start bit followed by eight data bits, one odd parity bit and finally one stop bit. Figure 23 shows a typical PS/2 data frame.

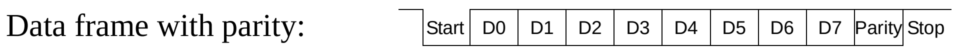


Figure 23. PS/2 data frame

14.2 Receiver operation

The receiver of APBPS2 receives the data from the keyboard or mouse, and converts it to 8-bit data frames to be read out via the APB bus. It is enabled through the receiver enable (RE) bit in the PS/2 control register. If a parity error or framing error occurs, the data frame will be discarded. Correctly received data will be transferred to a 16 byte FIFO. The data ready (DR) bit in the PS/2 status register will be set, and retained as long as the FIFO contains at least one data frame. When the FIFO is full, the receiver buffer full (RF) bit in the status register is set. The keyboard will be inhibited and buffer data until the FIFO gets read again. Interrupt is sent when a correct stop bit is received then it’s up to the software to handle any resend operations if the parity bit is wrong. Figure 24 shows a flow chart for the operations of the receiver state machine.

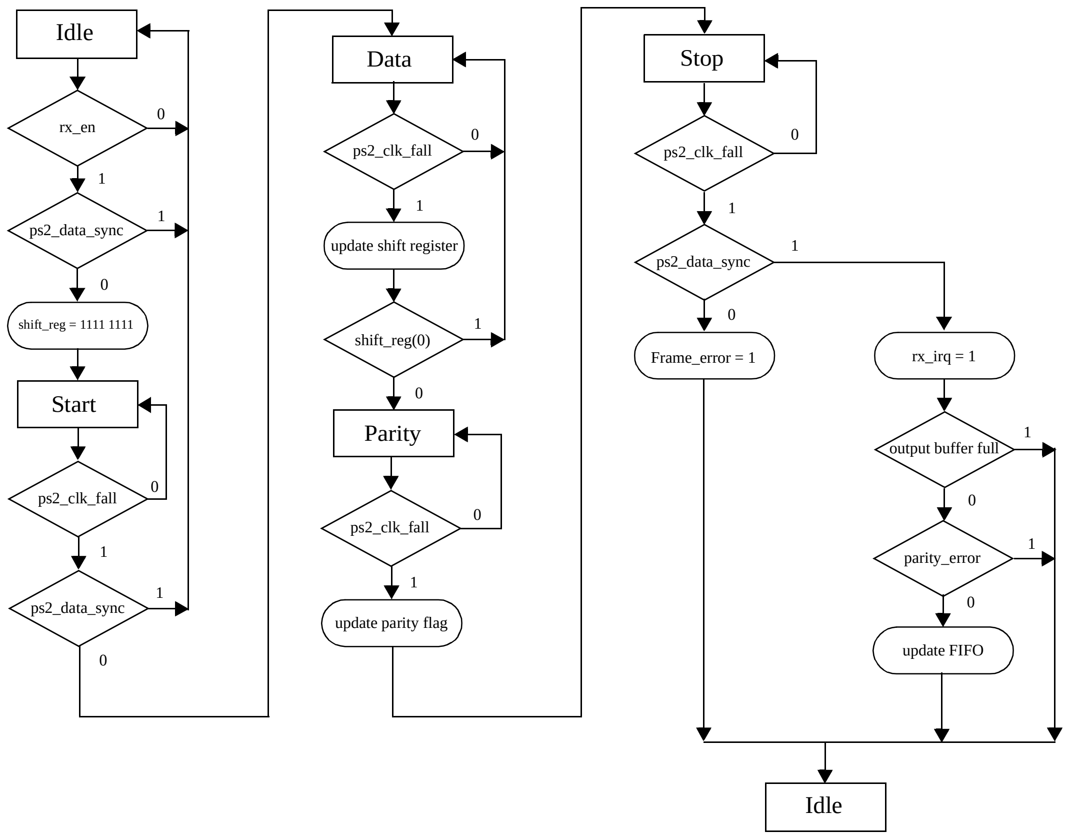


Figure 24. Flow chart for the receiver state machine

14.3 Transmitter operations

The transmitter part of APBPS2 is enabled for through the transmitter enable (TE) bit in the PS/2 control register. The PS/2 interface has a 16 byte transmission FIFO that stores commands sent by the CPU. Commands are used to set the LEDs on the keyboard, and the typematic rate and delay. Typematic rate is the repeat rate of a key that is held down, while the delay controls for how long a key has to be held down before it begins automatically repeating. Typematic repeat rates, delays and possible other commands are listed in table 74.

If the TE bit is set and the transmission FIFO is not empty a transmission of the command will start. The host will pull the clock line low for at least 100 us and then transmit a start bit, the eight bit command, an odd parity bit, a stop bit and wait for an acknowledgement bit by the device. When this happens an interrupt is generated. Figure 25 shows the flow chart for the transmission state machine.

14.4 Clock generation

A PS/2 interface should generate a clock of 10.0 - 16.7 KHz. To generate the PS/2 clock, APBPS2 divides the APB clock with either a fixed or programmable division factor. The divider consist of a 14-bit down-counter and can divide the APB clock with a factor of 1 - 16383. If the *fixed* generic is set to 1, the division rate is set to the *fKHz* generic divided by 10 in order to generate a 10 KHz clock. If *fixed* is 0, the division rate can be programmed through the timer reload register.

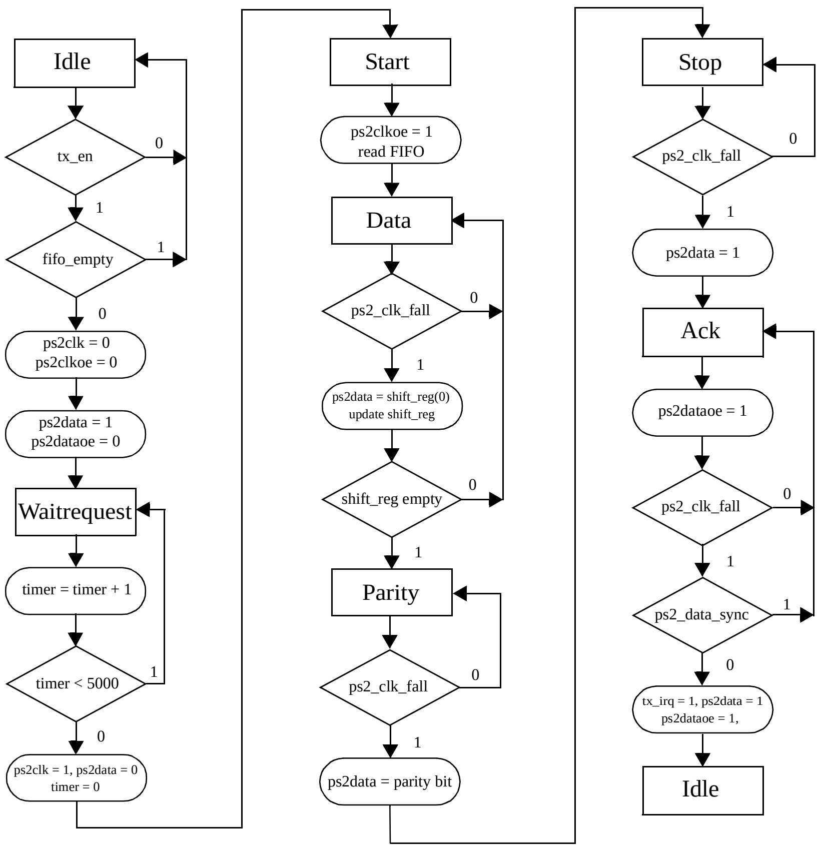


Figure 25. Flow chart for the transmitter state machine

14.5 Registers

The core is controlled through registers mapped into APB address space.

Table 67. APB PS/2 registers

APB address offset	Register
0x00	PS/2 Data register
0x04	PS/2 Status register
0x08	PS/2 Control register
0x0C	PS/2 Timer reload register

14.5.1 PS/2 Data Register

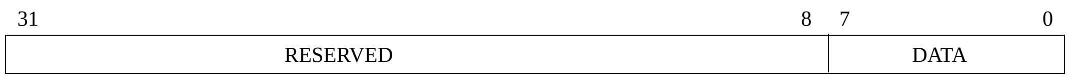


Figure 26. PS/2 data register

[7:0]: Receiver holding FIFO (read access) and Transmitter holding FIFO (write access). If the receiver FIFO is not empty, read accesses retrieve the next byte from the FIFO. Bytes written to this field are stored in the transmitter holding FIFO if it is not full.

14.5.2 PS/2 Status Register

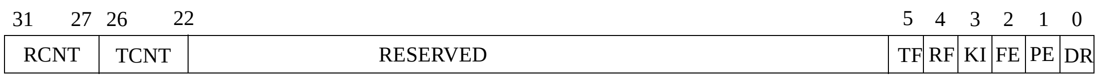


Figure 27. PS/2 status register

0:	Data ready (DR) - indicates that new data is available in the receiver holding register (read only).
1:	Parity error (PE) - indicates that a parity error was detected.
2:	Framing error (FE) - indicates that a framing error was detected.
3:	Keyboard inhibit (KI) - indicates that the keyboard is inhibited.
4:	Receiver buffer full (RF) - indicates that the output buffer (FIFO) is full (read only).
5:	Transmitter buffer full (TF) - indicates that the input buffer (FIFO) is full (read only).
[26:22]:	Transmit FIFO count (TCNT) - shows the number of data frames in the transmit FIFO (read only).
[31:27]:	Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO (read only).

14.5.3 PS/2 Control Register

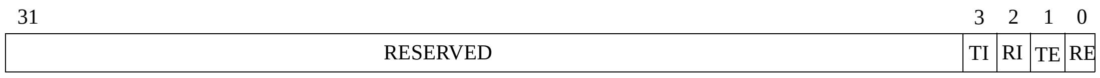


Figure 28. PS/2 control register

0:	Receiver enable (RE) - if set, enables the receiver.
1:	Transmitter enable (TE) - if set, enables the transmitter.
2:	Keyboard interrupt enable (RI) - if set, interrupts are generated when a frame is received
3:	Host interrupt enable (TI) - if set, interrupts are generated when a frame is transmitted

14.5.4 PS/2 Timer Reload Register

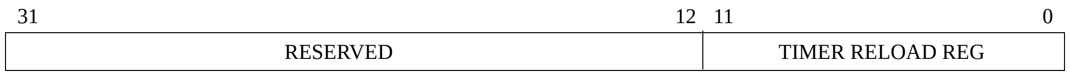


Figure 29. PS/2 timer register

[11:0]:	PS/2 timer reload register

14.6 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x061. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

14.7 Configuration options

Table 68 shows the configuration options of the core (VHDL generics).

Table 68. Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
fKHz	Frequency of APB clock in KHz.	1 - 163830	50000
fixed	Used fixed clock divider to generate PS/2 clock	0 - 1	1

14.8 Signal descriptions

Table 69 shows the interface signals of the core (VHDL ports).

Table 69. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
PS2I	PS2_CLK_I	Input	PS/2 clock input	-
	PS2_DATA_I	Input	PS/2 data input	-
PS2O	PS2_CLK_O	Output	PS/2 clock output	-
	PS2_CLK_OE	Output	PS/2 clock output enable	Low
	PS2_DATA_O	Output	PS/2 data output	-
	PS2_DATA_OE	Output	PS/2 data output enable	Low

* see GRLIB IP Library User’s Manual

14.9 Library dependencies

Table 70 shows libraries used when instantiating the core (VHDL libraries).

Table 70. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	MISC	Signals, component	PS/2 signal and component declaration

14.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.gencomp.all;
```

```
library gaisler;
use gaisler.misc.all;

entity apbps2_ex is
 port (
    rstn : in std_ulogic;
    clk : in std_ulogic;

    -- PS/2 signals
    ps2clk : inout std_ulogic;
    ps2data : inout std_ulogic
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- PS/2 signals
  signal kbdi : ps2_in_type;
  signal kbdo : ps2_out_type;

begin

ps20 : apbps2 generic map(pindex => 5, paddr => 5, pirq => 4)
      port map(rstn, clkm, apbi, apbo(5), kbdi, kbdo);

kbdclk_pad : iopad generic map (tech => padtech)
      port map (ps2clk,kbdo.ps2_clk_o, kbdo.ps2_clk_oe, kbdi.ps2_clk_i);

kbdata_pad : iopad generic map (tech => padtech)
        port map (ps2data, kbdo.ps2_data_o, kbdo.ps2_data_oe, kbdi.ps2_data_i);

end;
```

14.11 Keboard scan codes

Table 71. Scan code set 2, 104-key keyboard

KEY	MAKE	BREAK	- -	KEY	MAKE	BREAK	- -	KEY	MAKE	BREAK
A	1C	F0,1C		9	46	F0,46		[	54	FO,54
B	32	F0,32			`0E	F0,0E		INSERT	E0,70	E0,F0,70
C	21	F0,21		-	4E	F0,4E		HOME	E0,6C	E0,F0,6C
D	23	F0,23		=	55	FO,55		PG UP	E0,7D	E0,F0,7D
E	24	F0,24		\	5D	F0,5D		DELETE	E0,71	E0,F0,71
F	2B	F0,2B		BKSP	66	F0,66		END	E0,69	E0,F0,69
G	34	F0,34		SPACE	29	F0,29		PG DN	E0,7A	E0,F0,7A
H	33	F0,33		TAB	0D	F0,0D		U ARROW	E0,75	E0,F0,75
I	43	F0,43		CAPS	58	F0,58		L ARROW	E0,6B	E0,F0,6B
J	3B	F0,3B		L SHFT	12	FO,12		D ARROW	E0,72	E0,F0,72
K	42	F0,42		L CTRL	14	FO,14		R ARROW	E0,74	E0,F0,74
L	4B	F0,4B		L GUI	E0,1F	E0,F0,1F		NUM	77	F0,77
M	3A	F0,3A		L ALT	11	F0,11		KP /	E0,4A	E0,F0,4A
N	31	F0,31		R SHFT	59	F0,59		KP *	7C	F0,7C
O	44	F0,44		R CTRL	E0,14	E0,F0,14		KP -	7B	F0,7B
P	4D	F0,4D		R GUI	E0,27	E0,F0,27		KP +	79	F0,79
Q	15	F0,15		R ALT	E0,11	E0,F0,11		KP EN	E0,5A	E0,F0,5A
R	2D	F0,2D		APPS	E0,2F	E0,F0,2F		KP .	71	F0,71
S	1B	F0,1B		ENTER	5A	F0,5A		KP 0	70	F0,70
T	2C	F0,2C		ESC	76	F0,76		KP 1	69	F0,69
U	3C	F0,3C		F1	5	F0,05		KP 2	72	F0,72
V	2A	F0,2A		F2	6	F0,06		KP 3	7A	F0,7A
W	1D	F0,1D		F3	4	F0,04		KP 4	6B	F0,6B
X	22	F0,22		F4	0C	F0,0C		KP 5	73	F0,73
Y	35	F0,35		F5	3	F0,03		KP 6	74	F0,74
Z	1A	F0,1A		F6	0B	F0,0B		KP 7	6C	F0,6C
0	45	F0,45		F7	83	F0,83		KP 8	75	F0,75
1	16	F0,16		F8	0A	F0,0A		KP 9	7D	F0,7D
2	1E	F0,1E		F9	1	F0,01		]	5B	F0,5B
3	26	F0,26		F10	9	F0,09		;	4C	F0,4C
4	25	F0,25		F11	78	F0,78			52	F0,52
5	2E	F0,2E		F12	7	F0,07		,	41	F0,41
6	36	F0,36		PRNT SCRN	E0,12, E0,7C	E0,F0, 7C,E0, F0,12		.	49	F0,49
7	3D	F0,3D		SCROLL	7E	F0,7E		/	4A	F0,4A
8	3E	F0,3E		PAUSE	E1,14,77, E1,F0,14, F0,77	-NONE-				

Table 72. Windows multimedia scan codes

KEY	MAKE	BREAK
Next Track	E0, 4D	E0, F0, 4D
Previous Track	E0, 15	E0, F0, 15
Stop	E0, 3B	E0, F0, 3B
Play/Pause	E0, 34	E0, F0, 34
Mute	E0, 23	E0, F0, 23
Volume Up	E0, 32	E0, F0, 32
Volume Down	E0, 21	E0, F0, 21
Media Select	E0, 50	E0, F0, 50
E-Mail	E0, 48	E0, F0, 48
Calculator	E0, 2B	E0, F0, 2B
My Computer	E0, 40	E0, F0, 40
WWW Search	E0, 10	E0, F0, 10
WWW Home	E0, 3A	E0, F0, 3A
WWW Back	E0, 38	E0, F0, 38
WWW Forward	E0, 30	E0, F0, 30
WWW Stop	E0, 28	E0, F0, 28
WWW Refresh	E0, 20	E0, F0, 20
WWW Favorites	E0, 18	E0, F0, 18

Table 73. ACPI scan codes (Advanced Configuration and Power Interface)

KEY	MAKE	BREAK
Power	E0, 37	E0, F0, 37
Sleep	E0, 3F	E0, F0, 3F
Wake	E0, 5E	E0, F0, 5E

14.12 Keyboard commands

Table 74. Transmit commands:

Command	Description
0xED	Set status LED's - keyboard will reply with ACK (0xFA). The host follows this command with an argument byte*
0xEE	Echo command - expects an echo response
0xF0	Set scan code set - keyboard will reply with ACK (0xFA) and wait for another byte. 0x01-0x03 which determines the scan code set to use. 0x00 returns the current set.
0xF2	Read ID - the keyboard responds by sending a two byte device ID of 0xAB 0x83
0xF3	Set typematic repeat rate - keyboard will reply with ACK (0xFA) and wait for another byte which determines the typematic rate.
0xF4	Keyboard enable - clears the keyboards output buffer, enables keyboard scanning and returns an acknowledgement.
0xF5	Keyboard disable - resets the keyboard, disables keyboard scanning and returns an acknowledgement.
0xF6	Set default - load default typematic rate/delay (10.9cps/500ms) and scan code set 2
0xFE	Resend - upon receipt of the resend command the keyboard will retransmit the last byte
0xFF	Reset - resets the keyboard

* bit 0 controls the scroll lock, bit 1 the num lock, bit 2 the caps lock, bit 3-7 are ignored

Table 75. Receive commands:

Command	Description
0xFA	Acknowledge
0xAA	Power on self test passed (BAT completed)
0xEE	Echo respond
0xFE	Resend - upon receipt of the resend command the host should retransmit the last byte
0x00	Error or buffer overflow
0xFF	Error of buffer overflow

Table 76. The typematic rate/delay argument byte

MSB							LSB
0	DELAY	DELAY	RATE	RATE	RATE	RATE	RATE

Table 77. Typematic repeat rates

Bits 0-4	Rate (cps)		Bits 0-4	Rate (cps)		Bits 0-4	Rate (cps)		Bits 0-4	Rate (cps)
00h	30		08h	15		10h	7.5		18h	3.7
01h	26.7		09h	13.3		11h	6.7		19h	3.3
02h	24		0Ah	12		12h	6		1Ah	3
03h	21.8		0Bh	10.9		13h	5.5		1Bh	2.7
04h	20.7		0Ch	10		14h	5		1Ch	2.5
05h	18.5		0Dh	9.2		15h	4.6		1Dh	2.3
06h	17.1		0Eh	8.6		16h	4.3		1Eh	2.1
07h	16		0Fh	8		17h	4		1Fh	2

Table 78. Typematic delays

Bits 5-6	Delay (seconds)
00b	0.25
01b	0.5
10b	0.75
11b	1

15 APBUART - AMBA APB UART Serial Interface

15.1 Overview

The interface is provided for serial communications. The UART supports 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-bit clock divider. Two FIFOs are used for data transfer between the APB bus and UART, when *fifosize* VHDL generic > 1. Two holding registers are used data transfer between the APB bus and UART, when *fifosize* VHDL generic = 1. Hardware flow-control is supported through the RTSN/CTSN hand-shake signals, when *flow* VHDL generic is set. Parity is supported, when *parity* VHDL generic is set.

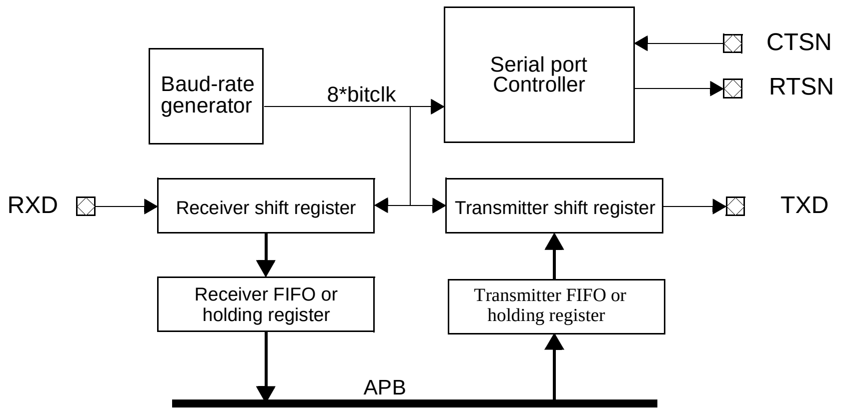

Serial port Controller
CTSN
RTSN
Baud-rate generator
8*bitclk
RXD
Receiver shift register
Transmitter shift register
TXD
Receiver FIFO or holding register
Transmitter FIFO or holding register
APB

Figure 30. Block diagram

15.2 Operation

15.2.1 Transmitter operation

The transmitter is enabled through the TE bit in the UART control register. Data that is to be transferred is stored in the FIFO/holding register by writing to the data register. This FIFO is configurable to different sizes via the *fifosize* VHDL generic. When the size is 1, only a single holding register is used but in the following discussion both will be referred to as FIFOs. When ready to transmit, data is transferred from the transmitter FIFO/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 bit (figure 31). The least significant bit of the data is sent first.

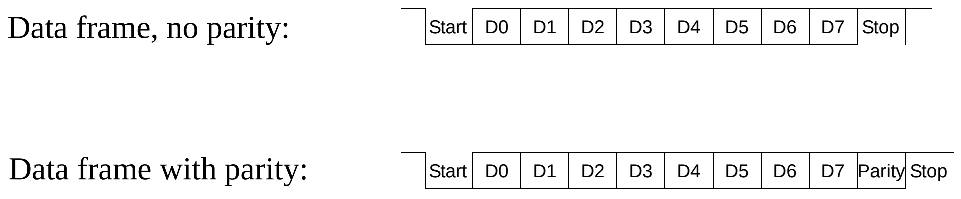


Figure 31. UART data frames

Following the transmission of the stop bit, if a new character is not available in the transmitter FIFO, the transmitter serial data output remains high and the transmitter shift register empty bit (TS) will be set in the UART status register. Transmission resumes and the TS is cleared when a new character is loaded into the transmitter FIFO. When the FIFO is empty the TE bit is set in the status register. If the transmitter is disabled, it will immediately stop any active transmissions including the character currently being shifted out from the transmitter shift register. The transmitter holding register may not be loaded when the transmitter is disabled or when the FIFO (or holding register) is full. If this is done, data might be overwritten and one or more frames are lost.

The discussion above applies to any FIFO configurations including the special case with a holding register (VHDL generic *fifosize* = 1). If FIFOs are used (VHDL generic *fifosize* > 1) some additional status and control bits are available. The TF status bit (not to be confused with the TF control bit) is set if the transmitter FIFO is currently full and the TH bit is set as long as the FIFO is *less* than half-full (less than half of entries in the FIFO contain data). The TF control bit enables FIFO interrupts when set. The status register also contains a counter (TCNT) showing the current number of data entries in the FIFO.

When 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 receivers RTSN, overrun can effectively be prevented.

15.2.2 Receiver operation

The receiver is enabled for data reception through the receiver enable (RE) bit in the UART 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 shifted through an 8-bit shift register where all bits have to have the same value before the new value is taken into account, effectively forming a low-pass filter with a cut-off frequency of 1/8 system clock.

The receiver also has a configurable FIFO which is identical to the one in the transmitter. As mentioned in the transmitter part, both the holding register and FIFO will be referred to as FIFO.

During reception, the least significant bit is received first. The data is then transferred to the receiver FIFO and the data ready (DR) bit is set in the UART status register as soon as the FIFO contains at least one data frame. 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. The data frame is not stored in the FIFO if an error is detected. Also, the new error status bits are or:ed with the old values before they are stored into the status register. Thus, they are not cleared until written to with zeros from the AMBA APB bus. If both the receiver FIFO and shift registers are full 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 FIFO is full. When the holding register is read, the RTSN will automatically be reasserted again.

When the VHDL generic *fifosize* > 1, which means that holding registers are not considered here, some additional status and control bits are available. The RF status bit (not to be confused with the RF control bit) is set when the receiver FIFO is full. The RH status bit is set when the receiver FIFO is half-full (at least half of the entries in the FIFO contain data frames). The RF control bit enables receiver FIFO interrupts when set. A RCNT field is also available showing the current number of data frames in the FIFO.

15.3 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. It 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 external clock input rather than the system clock. In this case, the frequency of external clock must be less than half the frequency of the system clock.

15.4 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.

15.5 FIFO debug mode

FIFO debug mode is entered by setting the debug mode bit in the control register. In this mode it is possible to read the transmitter FIFO and write the receiver FIFO through the FIFO debug register. The transmitter output is held inactive when in debug mode. A write to the receiver FIFO generates an interrupt if receiver interrupts are enabled.

15.6 Interrupt generation

Interrupts are generated differently when a holding register is used (VHDL generic *fifosize* = 1) and when FIFOs are used (VHDL generic *fifosize* > 1). When holding registers are used, 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.

For FIFOs, two different kinds of interrupts are available: normal interrupts and FIFO interrupts. For the transmitter, normal interrupts are generated when transmitter interrupts are enabled (TI), the transmitter is enabled and the transmitter FIFO goes from containing data to being empty. FIFO interrupts are generated when the FIFO interrupts are enabled (TF), transmissions are enabled (TE) and the UART is less than half-full (that is, whenever the TH status bit is set). This is a level interrupt and the interrupt signal is continuously driven high as long as the condition prevails. The receiver interrupts work in the same way. Normal interrupts are generated in the same manner as for the holding register. FIFO interrupts are generated when receiver FIFO interrupts are enabled, the receiver is enabled and the FIFO is half-full. The interrupt signal is continuously driven high as long as the receiver FIFO is half-full (at least half of the entries contain data frames).

15.7 Registers

The core is controlled through registers mapped into APB address space.

Table 79. UART registers

APB address offset	Register
0x0	UART Data register
0x4	UART Status register
0x8	UART Control register
0xC	UART Scaler register
0x10	UART FIFO debug register

15.7.1 UART Data Register

Table 80. UART data register

31 8	7 0
RESERVED	DATA

7: 0	Receiver holding register or FIFO (read access)
7: 0	Transmitter holding register or FIFO (write access)

15.7.2 UART Status Register

Table 81. UART status register

31 26	25 20	19 11	10	9	8	7	6	5	4	3	2	1	0
RCNT	TCNT	RESERVED	RF	TF	RH	TH	FE	PE	OV	BR	TE	TS	DR

31: 26	Receiver FIFO count (RCNT) - shows the number of data frames in the receiver FIFO.
25: 20	Transmitter FIFO count (TCNT) - shows the number of data frames in the transmitter FIFO.
10	Receiver FIFO full (RF) - indicates that the Receiver FIFO is full.
9	Transmitter FIFO full (TF) - indicates that the Transmitter FIFO is full.
8	Receiver FIFO half-full (RH) -indicates that at least half of the FIFO is holding data.
7	Transmitter FIFO half-full (TH) - indicates that the FIFO is less than half-full.
6	Framing error (FE) - indicates that a framing error was detected.
5	Parity error (PE) - indicates that a parity error was detected.
4	Overrun (OV) - indicates that one or more character have been lost due to overrun.
3	Break received (BR) - indicates that a BREAK has been received.
2	Transmitter FIFO empty (TE) - indicates that the transmitter FIFO is empty.
1	Transmitter shift register empty (TS) - indicates that the transmitter shift register is empty.
0	Data ready (DR) - indicates that new data is available in the receiver holding register

15.7.3 UART Control Register

Table 82. UART control register

31	30 13	12	11	10	9	8	7	6	5	4	3	2	1	0
FA	RESERVED	OE	DB	RF	TF	EC	LB	FL	PE	PS	TI	RI	TE	RE

31	FIFOs available (FA) - Set to 1 when receiver and transmitter FIFOs are available. When 0, only holding register are available.
30: 13	RESERVED
12	Output enable (OE) - Transmitter output enable. Driven directly on the txen output.
11	FIFO debug mode enable (DB) - when set, it is possible to read and write the FIFO debug register
10	Receiver FIFO interrupt enable (RF) - when set, Receiver FIFO level interrupts are enabled
9	Transmitter FIFO interrupt enable (TF) - when set, Transmitter FIFO level interrupts are enabled.
8	External Clock (EC) - if set, the UART scaler will be clocked by UARTI.EXTCLK
7	Loop back (LB) - if set, loop back mode will be enabled
6	Flow control (FL) - if set, enables flow control using CTS/RTS (when implemented)
5	Parity enable (PE) - if set, enables parity generation and checking (when implemented)
4	Parity select (PS) - selects parity polarity (0 = even parity, 1 = odd parity) (when implemented)
3	Transmitter interrupt enable (TI) - if set, interrupts are generated when a frame is transmitted
2	Receiver interrupt enable (RI) - if set, interrupts are generated when a frame is received
1	Transmitter enable (TE) - if set, enables the transmitter.
0	Receiver enable (RE) - if set, enables the receiver.

15.7.4 UART Scaler Register

Table 83. UART scaler reload register

31 12	11 0
RESERVED	SCALER RELOAD VALUE

11: 0	Scaler reload value

15.7.5 UART FIFO Debug Register

Table 84. UART FIFO debug register

31 8	7 0
RESERVED	DATA

7: 0	Transmitter holding register or FIFO (read access)
7: 0	Receiver holding register or FIFO (write access)

15.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x00C. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

15.9 Configuration options

Table 85 shows the configuration options of the core (VHDL generics).

Table 85. Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
console	Prints output from the UART on console during VHDL simulation and speeds up simulation by always returning '1' for Data Ready bit of UART Status register. Does not effect synthesis.	0 - 1	0
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
parity	Enables parity	0 - 1	1
flow	Enables flow control	0 - 1	1
fifosize	Selects the size of the Receiver and Transmitter FIFOs	1, 2, 4, 8, 16, 32	1
abits	Select the number of APB address bits used to decode the register addresses	3 - 8	8

15.10 Signal descriptions

Table 86 shows the interface signals of the core (VHDL ports).

Table 86. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
UARTI	RXD	Input	UART receiver data	-
	CTSN	Input	UART clear-to-send	Low
	EXTCLK	Input	Use as alternative UART clock	-
UARTO	RTSN	Output	UART request-to-send	Low
	TXD	Output	UART transmit data	-
	SCALER	Output	UART scaler value	-
	TXEN	Output	Output enable for transmitter	High
	FLOW	Output	Unused	-
	RXEN	Output	Receiver enable	High

* see GRLIB IP Library User's Manual

15.11 Library dependencies

Table 87 shows libraries that should be used when instantiating the core.

Table 87. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	UART	Signals, component	Signal and component declaration

15.12 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.uart.all;

entity apbuart_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- UART signals
    rxd   : in  std_ulogic;
    txd   : out std_ulogic
    );
end;

architecture rtl of apbuart_ex is

  -- APB signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- UART signals
  signal uarti : uart_in_type;
  signal uarto : uart_out_type;

begin

  -- AMBA Components are instantiated here
   ...


  -- APB UART
  uart0 : apbuart
  generic map (pindex => 1, paddr => 1,  pirq => 2,
console => 1, fifosize => 1)
  port map (rstn, clk, apbi, apbo(1), uarti, uarto);

  -- UART input data
  uarti.rxd <= rxd;

  -- APB UART inputs not used in this configuration
  uarti.ctsn <= ’0’; uarti.extclk <= ’0’;

  -- connect APB UART output to entity output signal
  txd <= uarto.txd;

end;
```

16 APBVGA - VGA controller with APB interface

16.1 Introduction

The APBVGA core is a text-only video controller with a resolution of 640x480 pixels, creating a display of 80x37 characters. The controller consists of a video signal generator, a 4 Kbyte text buffer, and a ROM for character pixel information. The video controller is controlled through an APB interface.

A block diagram for the data path is shown in figure 32.

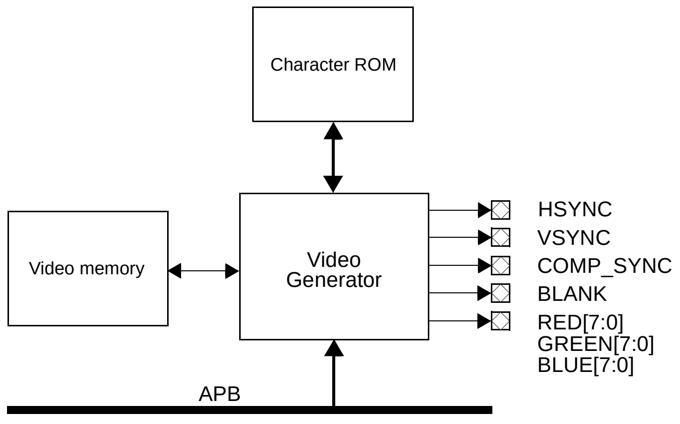


Figure 32. APBVGA block diagram

16.2 Operation

The video timing of APBVGA is fixed to generate a 640x480 display with 60 Hz refresh rate. The text font is encoded using 8x13 pixels. The display is created by scanning a segment of 2960 characters of the 4 Kbyte text buffer, rasterizing the characters using the character ROM, and sending the pixel data to an external video DAC using three 8-bit color channels. The required pixel clock is 25.175 MHz, which should be provided on the VGACLK input.

Writing to the video memory is made through the VGA data register. Bits [7:0] contains the character to be written, while bits [19:8] defines the text buffer address. Foreground and background colours are set through the background and foreground registers. These 24 bits corresponds to the three pixel colors, RED, GREEN and BLUE. The eight most significant bits defines the red intensity, the next eight bits defines the green intensity and the eight least significant bits defines the blue intensity. Maximum intensity for a color is received when all eight bits are set and minimum intensity when none of the bits are set. Changing the foreground color results in that all characters change their color, it is not possible to just change the color of one character. In addition to the color channels, the video controller generates HSYNC, VSYNC, CSYNC and BLANK. Togetherm the signals are suitable to drive an external video DAC such as ADV7125 or similar.

APBVGA implements hardware scrolling to minimize processor overhead. The controller monitors maintains a reference pointer containing the buffer address of the first character on the top-most line. When the text buffer is written with an address larger than the reference pointer + 2960, the pointer is incremented with 80. The 4 Kbyte text buffer is sufficient to buffer 51 lines of 80 characters. To simplify hardware design, the last 16 bytes (4080 - 4095) should not be written. When address 4079 has been written, the software driver should wrap to address 0. Sofware scrolling can be implemented by only using the first 2960 address in the text buffer, thereby never activating the hardware scolling mechanism.

16.3 Registers

The APB VGA is controlled through three registers mapped into APB address space.

Table 88. APB VGA registers

APB address offset	**Register**
0x0	VGA Data register
0x4	VGA Background color
0x8	VGA Foreground color

16.3.1 VGA Data Register

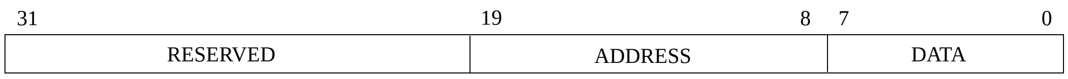


Figure 33. VGA data register

[19:8]: Video memory address (write access)
[7:0]: Video memory data (write access)

16.3.2 VGA Background Color

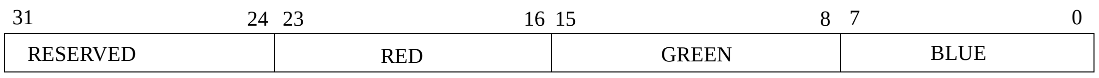


Figure 34. VGA Background color

[23:16]: Video background color red.
[15:8]: Video background color green.
[7:0]: Video background color blue.

16.3.3 VGA Foreground Color

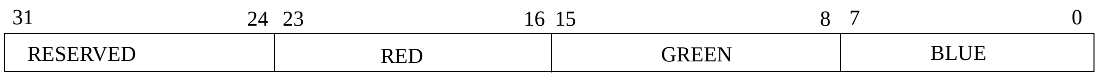


Figure 35. VGA Foreground color

[23:16]: Video foreground color red.
[15:8]: Video foreground color green.
[7:0]: Video foreground color blue.

16.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x060. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

16.5 Configuration options

Table 89 shows the configuration options of the core (VHDL generics).

Table 89. Configuration options

Generic	Function	Allowed range	Default
memtech	Technology to implement on-chip RAM	0 - NTECH	2
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#

16.6 Signal descriptions

Table 90 shows the interface signals of the core (VHDL ports).

Table 90. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
VGACLK	N/A	Input	VGA Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
VGAO	HSYNC	Output	Horizontal synchronization	High
	VSYNC		Vertical synchronization	High
	COMP_SYNC		Composite synchronization	Low
	BLANK		Blanking	Low
	VIDEO_OUT_R[7:0]		Video out, color red	-
	VIDEO_OUT_G[7:0]		Video out, color green	-
	VIDEO_OUT_B[7:0]		Video out, color blue	-

* see GRLIB IP Library User’s Manual

16.7 Library dependencies

Table 91 shows libraries used when instantiating the core (VHDL libraries).

Table 91. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	APB signal definitions
GAISLER	MISC	Signals, component	VGA signal and component declaration

16.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;
```

```
.
.


architecture rtl of apbuart_ex is

signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
signal vgao  : apbvga_out_type;

begin
  -- AMBA Components are instantiated here
   ...

  -- APB VGA
  vga0 : apbvga
  generic map (memtech => 2, pindex => 6, paddr => 6)
  port map (rstn, clk, vgaclk, apbi, apbo(6), vgao);
end;
```

17 ATACTRL - ATA Controller

17.1 Overview

The ATACTRL core is an ATA/ATAPI-5 host interface based on the OCIDEC-2 and OCIDEC-3 cores from OpenCores, with an additional AMBA AHB interface. This IP core provides an interface to IDE (Integrated Drive Electronics) devices, compatible to the ATA/ATAPI-5 standard. The ATACTRL supports PIO and MWDMA transfers. The MWDMA support is complying with SFF-8038i “Bus Master Programming Interface for IDE ATA Controllers Rev 1.0”, with the exception that the ATACTRL only have one IDE channel.

Figure 36 shows a block diagram of the ATACTRL core. Register access and PIO data transfer is done through the AHB slave interface. A write access to the registers in the core is done with no wait states, while a read access has one wait state. When the ATA data device is accessed, waitstates are inserted on the AHB bus until the transfer is completed. MWDMA data transfers are done through the AHB master interface. The DMA insterface has a configurable FIFO size, with a default of 8 32-bit words.

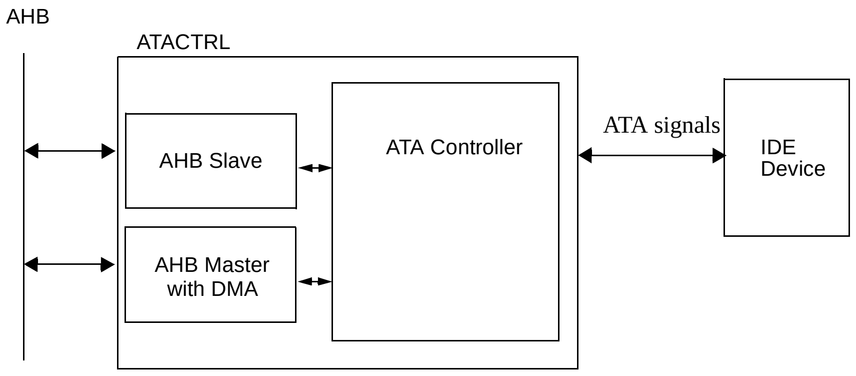


Figure 36. Block diagram of the internal structure of the ATACTRL.

17.2 Operation

17.2.1 Device hardware reset

Bit 0 in the core control register controls the reset signal to the ATA device. When this bit is set to ‘1’ the reset signal is asserted. After system reset, this bit is set to ‘1’. The hardware reset procedure should follow this protocol:

1: Set the ATA reset bit to ‘1’ and wait for at least 25 us.

2: Set the ATA reset bit to ‘0’ and wait for at least 2 ms.

3: Read the ATA status register until the busy bit is cleared.

4: Execute an Identify device or an Identify packet device command for all connected devices.

17.3 Registers

The core has 32-bit wide registers mapped into the AHB address space, with an offset shown in table 92. The registers in the ATA device are also mapped into the AHB address space starting with an offset of 0x40, see table 96. Data bits 7:0 are used for accessing the 8-bit registers in the ATA device and bits 15:0 are used for accessing the 16-bit data register in the ATA device.

Table 92. Core Registers.

Name	Offset	
CTRL	0x00	Control register
STAT	0x04	Status register
PCTR	0x08	PIO compatible timing register
PFTR0	0x0c	PIO fast timing register device 0
PFTR1	0x10	PIO fast timing register device 1
DTR0	0x14	DMA timing register device 0
DTR1	0x18	DMA timing register device 1
BMCMD	0x1C	Bus master IDE command register
BMVD0	0x20	Reserved
BMSTA	0x24	Bus master IDE status register
BMVD1	0x28	Reserved
PRDTB	0x2C	Bus master PRD table address

All the core registers can be read from and written to by any master on the AHB bus.

The Control register has the following bit layout, see table 93.

Table 93. Control Register

Bit #	Description
31	CompactFlash power on switch
30:16	Reserved
15	DMA enable
14	Reserved
13	DMA direction
12:10	Reserved
9	Reserved (Big Endian Little Endian conversion device 1)
8	Reserved (Big Endian Little Endian conversion device 0)
7	IDE enable
6	Fast timing device 1 enable
5	Fast timing device 0 enable
4	Reserved (PIO write ping-pong enable)
3	Fast timing device 1 IORDY enable
2	Fast timing device 0 IORDY enable
1	Compatible timing IORDY enable
0	ATA reset

All bits except bit 0 are set to zero after reset. Bit 0 is set to ‘1’. The timing registers should be loaded with an appropriate value before any of bit 1, 2, 3, 5, 6, 7 in the control register is set to ‘1’. Bit 31 controls the power signal for CompactFlash cards. When set to ‘1’ the power signal is connected to Vcc. This signal is used when the power to the CompactFlash card is controlled via a transistor in the design.

The Status register has the following bit layout, see table 94.

Table 94. Status Register

Bit #	**Description**
31:28	Device ID (= 0x02)
27:24	Revision Number (= 0)
23:16	Reserved
15	DMA transfer in progress
14:11	Reserved
10	DMA receive buffer empty
9	DMA transmit buffer full
8	DMARQ line status
7	PIO transfer in progress
6	Reserved (PIO write ping-pong full)
5:1	Reserved
0	IDE interrupt status (when set to 1, indicates that a device asserted its interrupt line)

The PIO timing registers have the following bit layout, see table 95.

Table 95. PIO Timing Register (PCTR, PFTR0, PFTR1)

Bit #	**Description**
31:24	End of Cycle Time (Teoc)
23:16	DIOW - data hold (T4)
15:8	DIOR/DIOW pulse width (T2)
7:0	Address valid to DIOR/DIOW (T1)

All timing values are in ns per clock cycle minus 2, rounded up to the nearest integer value.

Teoc = (T0 - T1 - T2) or T9 or T2i whichever is greater. For more timing information, read the ATA/ATAPI-5 standard document and see figure 37.

The PCTR timing register determines the timing for all PIO access, except for access to the data register. For data register access, the timing is determined by the PFTR0 or the PFTR1 register.

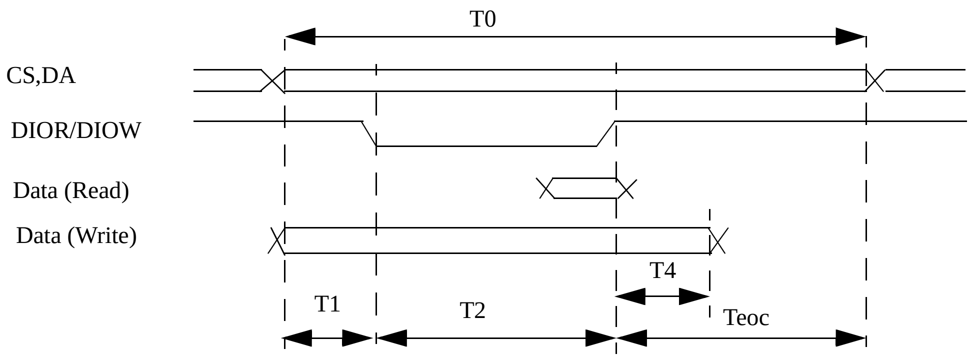


Figure 37. PIO timing diagram.

Table 96. ATA Device Registers

Name	Offset	Width
Data Register	0x40	16
Features/Error Register	0x44	8
Sector Number Register	0x48	8
Sector Count Register	0x4c	8
Cylinder Low Register	0x50	8
Cylinder High Register	0x54	8
Device/Head Register	0x58	8
Command/Status Register	0x5c	8
Alternate Status/Device Control Register	0x78	8

17.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x024. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

17.5 Configuration options

Table 97 shows the configuration options of the core (VHDL generics).

Table 97. Configuration options

Generic	Function	Allowed range	Default
mhindex	AHB master interafce bus index.	0 - NAHBMST-1	0
shindex	AHB slave interafce bus index.	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining the address space.	0 - 0xFFF	0x000
hmask	MASK field of the AHB BAR0 defining the address space.	0 - 0xFFF	0xFF0
pirq	Index of the interrupt line.	0 - NAHBIRQ-1	0
fdepth	DMA FIFO depth in words	2, 4, 8, 16, 32 or 64	8
mwdma	Enable MWDMA support	0 - 1	0
TWIDTH	Timing counter width		8
PIO_mode0_T1	Reset value for T1		6
PIO_mode0_T2	Reset value for T2		28
PIO_mode0_T4	Reset value for T4		2
PIO_mode0_Teoc	Reset value for Teoc		23

The default values for PIO_mode0_* are calculated for a 100 MHz clock.

17.6 Signal descriptions

Table 98 shows the interface signals of the core (VHDL ports).

Table 98. Signal descriptions

Signal name	Field	Type	Function	Active
rst		Input	Synchronous reset signal	Low
arst		Input	Asynchronous reset signal	Low
clk		Input	Clock signal	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
ATAI	ddi[15:0]		Data Input	
	iordy		IO channel ready	
	intrq		interrupt	
	dmarq		DMA REQ signal	
ATAO	rstn	Output	Reset signal to ATA device	Low
	ddo[15:0]	Output	Data Output	-
	oen	Output	Data output enable	High
	da[2:0]	Output	Device address	-
	cs0	Output	Chip Select 0	Low
	cs1	Output	Chip Select 1	Low
	dior	Output	Device IO read	Low
	diow	Output	Device IO write	Low
	dmack	Output	DMA ACK signal	-
CFO	atasel	Output	Select “True-IDE“ mode for CompactFlash	-
	csel	Output	Device Master select signal	-
	da[10:3]	Output	Grounded address signals	-
	power	Output	Power switch	-
	we	Output	Connected to Vcc for True-IDE mode	-

* see GRLIB IP Library User’s Manual

17.7 Library dependencies

Table 99 shows libraries used when instantiating the core (VHDL libraries).

Table 99. Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	ATA	Signals, component	ATACTRL component declarations, ATA signals

17.8 Software support

The ATA Controller is supported by the linux-2.6 kernel, in the snapgear-p35 version and later. The snapgear embedded linux distribution can be downloaded from www.gaisler.com. Since the register set of the ATA controller follows the IDE standard, compatible software drivers from other operating systems should easily be adaptable.

17.9 Instantiation

This example shows how the can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
library gaisler;
use gaisler.ata.all;
use work.config.all;


signal idei : ata_in_type;
signal ideo : ata_out_type;

atac0 : atactrl
    generic map(tech => 0, fdepth => 8, mhindex => 2, mwdma => 1,
     shindex => 3, haddr => 16#A00#, hmask => 16#fff#, pirq  => 6,
      TWIDTH   => 8,         -- counter width
      -- PIO mode 0 settings (@100MHz clock)
      PIO_mode0_T1   => 6,   -- 70ns
      PIO_mode0_T2   => 28,  -- 290ns
      PIO_mode0_T4   => 2,   -- 30ns
      PIO_mode0_Teoc => 23   -- 240ns ==> T0 - T1 - T2 = 600 - 70 - 290 = 240
    )
    port map( rst => rstn, arst => vcc(0), clk => clkm, ahbmi => ahbmi,
        ahbmo => ahbmo(2), ahbsi => ahbsi, ahbso => ahbso(3),
      atai => idei, atao => ideo);

    ata_rstn_pad : outpad generic map (tech => padtech)
      port map (ata_rstn, ideo.rst);
    ata_data_pad : iopadv generic map (tech => padtech, width => 16, oepol => 1)
        port map (ata_data, ideo.ddo, ideo.oen, idei.ddi);
    ata_da_pad : outpadv generic map (tech => padtech, width => 3)
        port map (ata_da, ideo.da);
    ata_cs0_pad : outpad generic map (tech => padtech)
        port map (ata_cs0, ideo.cs0);
    ata_cs1_pad : outpad generic map (tech => padtech)
        port map (ata_cs1, ideo.cs1);
    ata_dior_pad : outpad generic map (tech => padtech)
        port map (ata_dior, ideo.dior);
    ata_diow_pad : outpad generic map (tech => padtech)
        port map (ata_diow, ideo.diow);
    iordy_pad : inpad generic map (tech => padtech)
        port map (ata_iordy, idei.iordy);
    intrq_pad : inpad generic map (tech => padtech)
        port map (ata_intrq, idei.intrq);
    dmarq_pad : inpad generic map (tech => padtech)
        port map (ata_dmarq, idei.dmarq);
    dmack_pad : outpad generic map (tech => padtech)
        port map (ata_dmack, ideo.dmack);
```

18 B1553BC - AMBA plug&play interface for Actel Core1553BBC

18.1 Overview

The interface provides a complete Mil-Std-1553B Bus Controller (BC). The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core1553BBC core.

The interface provides a complete, MIL-STD-1553B Bus Controller (BC). The interface reads message descriptor blocks from the memory and generates messages that are transmitted on and transmitted on the 1553B bus. Data received is written to the memory.

The interface consists of five main blocks: the 1553B encoder, the 1553B decoder, a protocol controller block, a CPU interface, and a backend interface.

A single 1553B encoder takes each word to be transmitted and serializes it using Manchester encoding. The encoder includes independent logic to prevent the BC from transmitting for greater than the allowed period and to provide loopback fail logic. The loopback logic monitors the received data and verifies that the interface has correctly received every word that is transmitted. The encoder output is gated with the bus enable signals to select which buses the encoder should be transmitting. Since the BC knows which bus is in use at any time, only a single decoder is required.

The decoder takes the serial Manchester received data from the bus and extracts the received data words The decoder contains a digital phased lock loop (PLL) that generates a recovery clock used to sample the incoming serial data. The data is then deserialized and the 16-bit word decoded. The decoder detects whether a command, status or data word has been received and checks that no Manchester encoding or parity errors occurred in the word.

The protocol controller block handles all the message sequencing and error recovery. This is a complex state machine that reads the 1553B message frames from memory and transmits them on the 1553B bus. The AMBA interface allows a system processor to access the control registers. It also allows the processor to directly access the memory connected to the backend interface, this simplifies the system design.

The B1553BC core provides an AMBA interface with GRLIB plug&play for the Actel Core1553BBC core (MIL-STD-1553B Bus Controller). B1553BC implements two AMBA interfaces: one AHB master interface for the memory interface, and one APB slave interface for the CPU interface and control registers.

The Actel Core1553BBC core, entity named BC1553B, is configured to use the shared memory interface, and only internal register access is allowed through the APB slave interface. Data is read and stored via DMA using the AHB master interface.

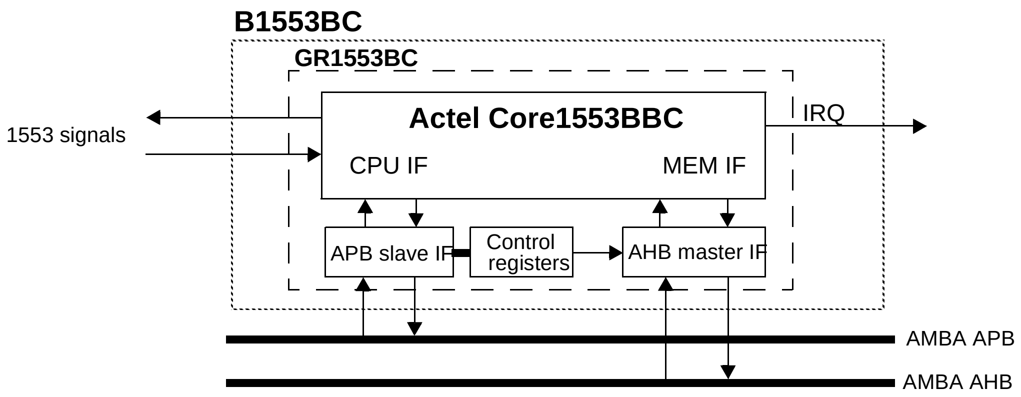


Figure 38. Block diagram

18.2 AHB interface

The Core1553BBC operates on a 65536 x 16 bit memory buffer, and therefore a 128 kilobyte aligned memory area should be allocated. The memory is accessed via the AMBA AHB bus. The Core1553BBC uses only 16 address bits, and the top 15 address bits of the 32-bit AHB address can be programmed in the AHB page address register. The 16-bit address provided by the Core1553BBC is left-shifted one bit, and forms the AHB address together with the AHB page address register. Note that all pointers given to the Core1553BBC core need to be right-shifted one bit because of this. All AHB accesses are done as half word single transfers.

The endianness of the interface depends on the endian VHDL generic.

The AMBA AHB protection control signal HPROT is driven permanently with “0011”, i.e a not cacheable, not bufferable, privileged data access. The AMBA AHB lock signal HLOCK is driven with ‘0’.

18.3 Operation

To transmit data on the 1553 bus, an instruction list and 1553 messages should be set up in the memory by the processor. After the bus interface has been activated, it will start to process the instruction list and read/write data words from/to the specified memory locations. Interrupts are generated when interrupt instructions are executed, on errors or when the interface has completed the list.

18.4 Registers

The core is programmed through registers mapped into APB address space. The internal registers of Core1553BBC are mapped on the eight lowest APB addresses. These addresses are 32-bit word aligned although only the lowest 16 bits are used. Refer to the *Actel Core1553BBC MIL-STD-1553B Bus Controller* data sheet for detailed information.

*Table 100.*B1553BC registers

APB address offset	**Register**
0x00	Control/Status
0x04	Setup
0x08	List pointer
0x0C	Message pointer
0x10	Clock value
0x14	Asynchronous list pointer
0x18	Stack pointer
0x1C	Interrupt register
0x20	GR1553 status/control
0x24	AHB page address register

Table 101. GR1553 status register (read)

31 ... 3	2	1	0
RESERVED	extflag	memfail	busy

31: 3	RESERVED
2	External flag bit. Drives the extflag input of the Core1553BBC. Resets to zero.
1	Memory failure. Shows the value of the memfail output from Core1553BBC.
0	Busy. Shows the value of the busy output from Core1553BBC.

Table 102. GR1553 status register (write)

31 1	0
RESERVED	extflag

31: 2	RESERVED
0	External flag bit. Drives the extflag input of the Core1553BBC. Resets to zero.

Table 103. GR1553 status register (write)

31 17	16 0
ahbaddr	RESERVED

31: 17	Holds the 15 top most bits of the AHB address of the allocated memory area
16: 0	RESERVED

18.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x070. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

18.6 Configuration options

Table 104 shows the configuration options of the core (VHDL generics).

*Table 104.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt number	0 - NAHBIRQ -1	0

18.7 Signal descriptions

Table 105 shows the interface signals of the core (VHDL ports).

*Table 105.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
B1553I	-	Input	1553 bus input signals	-
	busainp		Positive data input from the A receiver	High
	busainn		Negative data input from the A receiver	Low
	busbinp		Positive data to the B receiver	High
	busbinn		Negative data to the B receiver	Low
B1553O	-	Output	1553 bus output signals	-
	busainen		Enable for the A receiver	High
	busaoutin		Inhibit for the A transmitter	High
	busaoutp		Positive data to the A transmitter	High
	busaoutn		Negative data to the A transmitter	Low
	busbinen		Enable for the B receiver	High
	busboutin		Inhibit for the B transmitter	High
	busboutp		Positive output to the B transmitter	High
	busboutn		Negative output to the B transmitter	Low
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

18.8 Library dependencies

Table 106 shows libraries used when instantiating the core (VHDL libraries).

*Table 106.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal definitions
GAISLER	B1553	Signals, component	Signal and component declaration

The B1553BC depends on GRLIB, GAISLER, GR1553 and Core1553BBC.

18.9 Component declaration

The core has the following component declaration.

```
component b1553bc is
    generic (
      hindex  : integer := 0;
      pindex  : integer := 0;
      paddr   : integer := 0;
      pmask   : integer := 16#fff#;
      pirq    : integer := 0
      );
    port (
      rstn    : in  std_ulogic;
      clk     : in  std_ulogic;
      b1553i  : in  b1553_in_type;
      b1553o  : out b1553_out_type;
      apbi    : in  apb_slv_in_type;
      apbo    : out apb_slv_out_type;
      ahbi    : in  ahb_mst_in_type;
      ahbo    : out ahb_mst_out_type
      );
  end component;
```

18.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.b1553.all;
...
signal bin : b1553_in_type;
signal bout : b1553_out_type;
...
bc1553_0 : b1553bc
    generic map (hindex => 2, pindex => 12, paddr => 12, pirq => 2)
    port map (rstn, clkm, bin, bout, apbi, apbo(12), ahbmi, ahbmo(2));
```

19 B1553BRM - AMBA plug&play interface for Actel Core1553BRM

19.1 Overview

The interface provides a complete Mil-Std-1553B Bus Controller (BC), Remote Terminal (RT) or Monitor Terminal (MT). The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core1553BRM core.

The interface consists of six main blocks: 1553 encoder, 1553B decoders, a protocol controller block, AMBA bus interface, command word legality interface, and a backend interface.

The interface can be configured to provide all three functions BC, RT and MT or any combination of the three. All variations use all six blocks except for the command legalization interface, which is only required on RT functions that implement RT legalization function externally.

A single 1553 encoder takes each word to be transmitted and serializes it using Manchester encoding. The encoder also includes independent logic to prevent the interface from transmitting for greater than the allowed period as well as loopback fail logic. The loopback logic monitors the received data and verifies that the interface has correctly received every word that it transmits. The output of the encoder is gated with the bus enable signals to select which buses the interface should be transmitting on. Two decoders take the serial Manchester received data from each bus and extract the received data words.

The decoder contains a digital phased lock loop (PLL) that generates a recovery clock used to sample the incoming serial data. The data is then de-serialized and the 16-bit word decoded. The decoder detects whether a command, status, or data word has been received, and checks that no Manchester encoding or parity errors occurred in the word.

The protocol controller block handles all the message sequencing and error recovery for all three operating modes, Bus Controller, Remote Terminal, and Bus Monitor. This is complex state machine that processes messages based on the message tables setup in memory, or reacts to incoming command words. The protocol controller implementation varies depending on which functions are implemented. The AMBA interface allows a system processor to access the control registers. It also allows the processor to directly access the memory connected to the backend interface, this simplifies the system design.

The interface comprises 33 16-bit registers. Of the 33 registers, 17 are used for control function and 16 for RT command legalization.

The B1553BRM core provides an AMBA interface for the Actel Core1553BRM core (MIL-STD-1553B Bus Controller/Remote Terminal/Bus Monitor). The B1553BRM core implements two AMBA interfaces: one AHB master interface for the memory interface, and one APB slave interface for the CPU interface and control registers.

The Actel Core1553BRM core, entity named BRM, is configured to use the shared memory interface, and only internal register access is allowed through the APB slave interface. Data is read and stored via DMA using the AHB master interface.

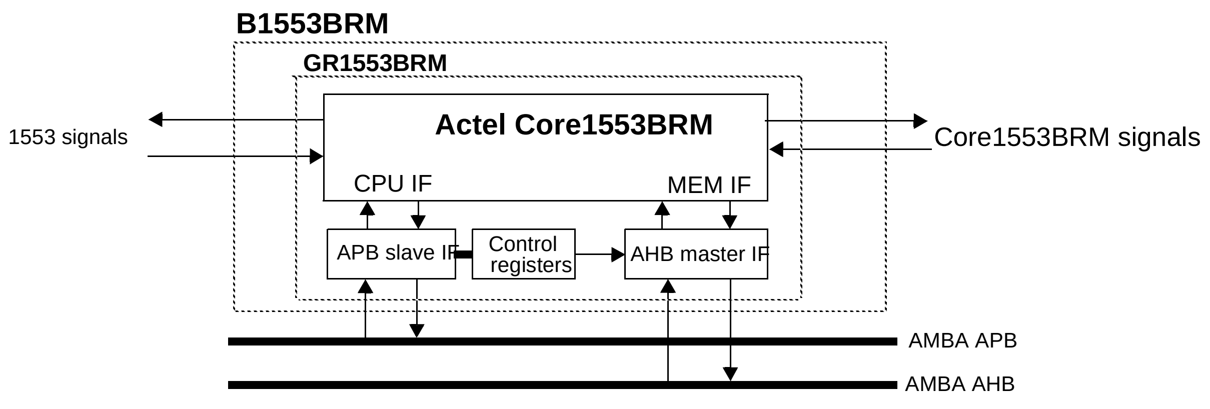


Figure 39. Block diagram

19.2 AHB interface

The amount of memory that the Mil-Std-1553B interface can address is 128 (2***abit* VHDL generic, i.e. *abit* => 128) kbytes. The base address of this memory area must be aligned to a boundary of its own size and written into the AHB page address register.

The 16 bit address provided by the Core1553BRM core is shifted left one bit, and forms the AHB address together with the AHB page address register. Note that all pointers given to the Core1553BRM core needs to be right shifted one bit because of this.

The amount of memory needed for the Core1553BRM core is operation and implementation specific. Any configuration between 1 to 128 kilobytes is possible although a typical system needs at least 4 kbyte of memory. The allocated memory area needs to be aligned to a boundary of its own size and the number of bits needed to address this area must be specificed with the *abits* VHDL generic.

The address bus of the Core1553BRM is 16 bits wide but the amount of bits actually used depends on the setup of the data structures. The AHB page address register should be programmed with the 32-*abits* top bits of the 32-bit AHB address, *abit* being a VHDL generic. The address provided by the Core1553BRM core is shifted left one bit, and forms the AHB address together with the AHB page address register. Note that all pointers given to the Core1553BRM core needs to be right shifted one bit because of this.

When the Core1553BRM core has been granted access to the bus it expects to be able to do a series of uninterrupted accesses. To handle this requirement the AHB master locks the bus during these transfers. In the worst case, the Core1553BRM can do up to 7 writes in one such access and each write takes 2 plus the number of waitstate cycles with 4 idle cycles between each write strobe. This means care has to be taken if using two simultaneous active Core1553BRM cores on the same AHB bus.All AHB accesses are done as half word single transfers.

The endianness of the interface depends on the endian VHDL generic.

The AMBA AHB protection control signal HPROT is driven permanently with "0011" i.e a not cacheable, not bufferable, privileged data access. During all AHB accesses the AMBA AHB lock signal HLOCK is driven with `1' and `0' otherwise.

19.3 Operation

The mode of operation can be selected with the mselin VHDL generic or later changed by writing to the “operation and status” register of the Core1553BRM core. For information about how the core functions during the different modes of operation see the *Actel Core1553BRM MIL-STD-1553 BC, RT, and MT* data sheet.

19.4 Registers

The core is programmed through registers mapped into APB address space. The internal registers of Core1553BRM are mapped on the 33 lowest APB addresses. These addresses are 32-bit word aligned although only the lowest 16 bits are used. Refer to the *Actel Core1553BRM MIL-STD-1553 BC, RT, and MT* data sheet for detailed information.

*Table 107.*B1553BRM registers

APB address offset	**Register**
0x00 - 0x84	Core1553BRM registers
0x100	B1553BRM status/control
0x104	B1553BRM interrupt settings
0x108	AHB page address register

B1553BRM status/control register

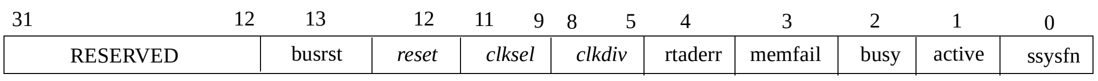


Figure 40. B1553BRM status/control register

13	Bus reset. If set a bus reset mode code has been received. Generates an irq when set.
12:5	*Bit 12-5 are only available in the b1553brm_async toplevel which allows separate AHB and BRM frequencies.*
12	Reset. For asynchronous toplevel only. Software reset. Self clearing.
11:9	Clock select. For asynchronous toplevel only. 0 - AHB clock, 1 - divided AHB clock, 2 - brm_clk1, 3 - brm_clk2.
8:5	Clock divisor. For asynchronous toplevel only. BRM core clocked by AHB clock divided by 2*(clkdiv+1).
4:	Address error. Shows the value of the rtaderr output from Core1553BRM.
3:	Memory failure. Shows the value of the memfail output from Core1553BRM.
2:	Busy. Shows the value of the busy output from Core1553BRM.
1:	Active. Show the value of the active output from Core1553BRM.
0:	Ssyfn. Connects directly to the ssyfn input of the Core1553BRM core. Resets to 1.

B1553BRM interrupt register

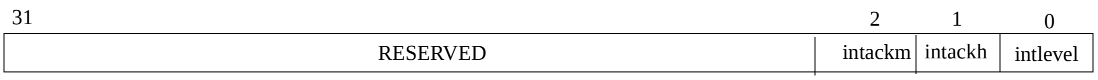


Figure 41. B1553RM interrupt register

2:	Message interrupt acknowledge. Controls the intackm input signal of the Core1553BRM core.
1:	Hardware interrupt acknowledge. Controls the intackh input signal of the Core1553BRM core.
0:	Interrupt level. Controls the intlevel input signal of the Core1553BRM core.

AHB page address register

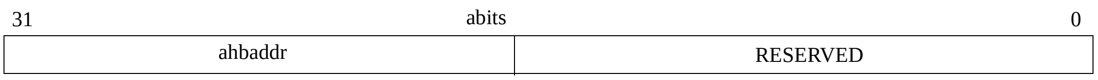


Figure 42. AHB page address register

[31:17]:	Holds the top most bits of the AHB address of the allocated memory area.

19.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x072. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

19.6 Configuration options

Table 108 shows the configuration options of the core (VHDL generics).

*Table 108.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0-NAHBMST-1	0
pindex	APB slave index	0-NAPBSLV-1	0
paddr	ADDR field of the APB BAR	0-16#FFF#	0
pmask	MASK field of the APB BAR	0-16#FF0#	16#FF0#
pirq	Index of the interrupt line	0-NAHBIRQ-1	0
endian	Data endianness of the AHB bus (Big = 0, Little = 1)	0 -1	0
ahbaddr	Reset value for address register	16#00000#-16#FFFFF#	16#00000#
abits	Number of bits needed to address the memory area	12-17	17
rtaddr	RT address	0 - 31	0
rtaddrp	RT address parity bit. Set to achieve odd parity.	0 - 1	1
lockn	Lock rtaddrin, rtaddrp, mselin and abstdin	0-1	0
mselin	Mode select	0-3	0
abstdin	Bus standard A/B	0-1	0
bcenable	Enable bus controller	0-1	1
rtenable	Enable remote terminal	0-1	1
mtenable	Enable bus monitor	0-1	1
legregs	Enable legalization registers	0-1	1
enhanced	Enable enhanced register	0-1	1
initfreq	Initial operation frequency	12,16,20,24	20
betiming	Backend timing	0-1	1
nsync	*Number of sync registers*	*1-2*	*2*

The nsync generic is only available for the b1553_async core.

Except for endian, ahbaddr and abits these generics drive the corresponding signal or generic of the Core1553BRM core. Bcenable, rtenable, mtenable, legregs, enhanced, initfreq and betiming connects to generics on the Core1553BRM core and therefore are ignored unless the RTL version is used.

19.7 Signal descriptions

Table 109 shows the interface signals of the core (VHDL ports).

*Table 109.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
RSTOUTN	N/A	Output	Reset from BRM core	Low
CLK	N/A	Input	System clock (AHB)	-
BRM_CLK1	N/A	Input	BRM clock 1	-
BRM_CLK2	N/A	Input	BRM clock 1	-
TCLK	N/A	Input	External time base	-
B1553I	-	Input	1553 bus input signals	-
	busainp		Positive data input from the A receiver	High
	busainn		Negative data input from the A receiver	Low
	busbinp		Positive data to the B receiver	High
	busbinn		Negative data to the B receiver	Low
B1553O	-	Output	1553 bus output signals	-
	busainen		Enable for the A receiver	High
	busaoutin		Inhibit for the A transmitter	High
	busaoutp		Positive data to the A transmitter	High
	busaoutn		Negative data to the A transmitter	Low
	busbinen		Enable for the B receiver	High
	busboutin		Inhibit for the B transmitter	High
	busboutp		Positive output to the B transmitter	High
	busboutn		Negative output to the B transmitter	Low
BRMI	-	Input	BRM input signals	-
	cmdok		Command word validation alright	High
BRMO	-	Output	BRM output signals	-
	msgstart		Message process started	High
	cmdsync		Start of command word on bus	High
	syncnow		Synchronize received	High
	busreset		Reset command received	High
	opmode		Operating mode	-
	cmdval		Active command	-
	cmdokout		Command word validated	High
	cmdstb		Active command value changed	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

19.8 Library dependencies

Table 110 shows libraries used when instantiating the core (VHDL libraries).

*Table 110.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal definitions
GAISLER	B1553	Signals, component	Signal and component declaration

The B1553BRM depends on VHDL libraries GRLIB, GAISLER, GR1553 and Core1553BRM.

19.9 Component declaration

The core has the following component declaration.

```
component b1553brm is
     generic (
       hindex       : integer := 0;
       pindex       : integer := 0;
       paddr        : integer := 0;
       pmask        : integer := 16#ff0#;
       pirq         : integer := 0;
       ahbaddr      : integer range 0 to 16#FFFFF# := 0;
       abits        : integer range 12 to 17 := 16;
       rtaddr       : integer range 0 to 31 := 0;
       rtaddrp      : integer range 0 to 1  := 1;
       lockn        : integer range 0 to 1  := 1;
       mselin       : integer range 0 to 3  := 1;
       abstdin      : integer range 0 to 1  := 0;

       bcenable     : integer range 0 to 1  := 1;
       rtenable     : integer range 0 to 1  := 1;
       mtenable     : integer range 0 to 1  := 1;
       legregs      : integer range 0 to 4  := 1;
       enhanced     : integer range 0 to 1  := 1;
       initfreq     : integer range 12 to 24:= 20;
       betiming     : integer range 0 to 1  := 1
       );
     port (
       rstn      : in    std_ulogic;
       rstoutn   : out   std_ulogic;
       clk       : in    std_ulogic;
       tclk      : in    std_ulogic;
       brmi      : in    brm1553_in_type;
       brmo      : out   brm1553_out_type;
       b1553i    : in    b1553_in_type;
       b1553o    : out   b1553_out_type;
       apbi      : in    apb_slv_in_type;
       apbo      : out   apb_slv_out_type;
       ahbi      : in    ahb_mst_in_type;
       ahbo      : out   ahb_mst_out_type
       );
  end component;
```

19.10 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.b1553.all;
```

```
...
signal bin : b1553_in_type;
signal bout : b1553_out_type;
signal brmi : brm1553_in_type;
signal brmo : brm1553_out_type;
...

bc1553_0 : b1553brm
    generic map (hindex => 2, pindex => 12, paddr => 16#10#, pirq => 2,
abits => 17, mselin => 0)
    port map (rstn, open, clkm, gnd(0), brmi, brmo, bin, bout, apbi, apbo(12), ahbmi,
ahbmo(2));
```

20 B1553RT - AMBA plug&play interface for Actel Core1553BRT

20.1 Overview

The interface provides a complete Mil-Std-1553B Remote Terminal (RT). The interface connects to the MIL-STD-1553B bus through external transceivers and transformers. The interface is based on the Actel Core1553BRT core.

The interface provides a complete, dual-redundant MIL-STD-1553B remote terminal (RT) apart from the transceivers required to interface to the bus. At a high level, the interface simply provides a set of memory mapped sub-addresses that 'receive data written to' or 'transmit data read from.' The interface requires 2,048 words of memory, which can be shared with a local processor. The interface supports all 1553B mode codes and allows the user to designate as illegal any mode code or any particular sub-address for both transmit and receive operations. The command legalization can be done internally or via a command legalization interface.

The interface consists of six main blocks: 1553B encoders, 1553B decoders, backend interface, command decoder, RT controller blocks and a command legalization block.

A single 1553B encoder is used for the interface. This takes each word to be transmitted and serializes it, after which the signal is Manchester encoded. The encoder also includes both logic to prevent the RT from transmitting for greater than the allowed period and loopback fail logic. The loopback logic monitors the received data and verifies that the interface has correctly received every word that it transmits. The output of the encoder is gated with the bus enable signals to select which buses the RT should use to transmit.

The interface includes two 1553B decoders. The decoder takes the serial Manchester data received from the bus and extracts the received data words. The decoder contains a digital phased lock loop (PLL) that generates a recovery clock used to sample the incoming serial data. The data is then deserialized and the 16-bit word decoded. The decoder detects whether a command or data word is received, and also performs Manchester encoding and parity error checking.

The command decoder and RT controller blocks decode the incoming command words, verifying the legality. Then the protocol state machine responds to the command, transmitting or receiving data or processing a mode code.

The B1553RT core provides an AMBA interface with GRLIB plug&play for the Actel Core1553BRT (MIL-STD-1553B Remote Terminal). B1553RT implements two AMBA interfaces: one AHB master interface for the memory interface, and one APB slave interface for the control registers.

The Actel Core1553BRT core, entity named RT1553B, is configured to use the shared memory interface. Data is read and stored via DMA using the AHB master interface.

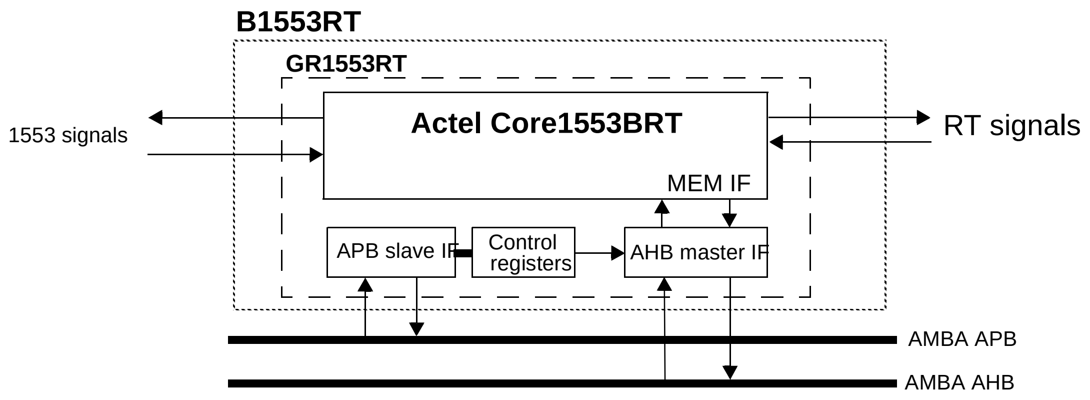


Figure 43. B1553RT block diagram

20.2 Operation

The Core1553BRT core operates on a 2048*16 bit memory buffer, and therefore a 4 kilobyte memory area should be allocated. The memory is accessed via the AMBA AHB bus. The Core1553BRT uses only 11 address bits, and the top 20 address bits of the 32-bit AHB address can be programmed in the AHB page address register. The 11-bit address provided by the Core1553BRT core is left-shifted one bit, and forms the AHB address together with the AHB page address register. All AHB accesses are done as half word single transfers.

The used memory area has the following address map. Note that all 1553 data is 16 bit wide and will occupy two bytes. Every sub-address needs memory to hold up to 32 16 bit words.

*Table 111.*Memory map for 1553 data

Address	**Content**
0x000-0x03F	RX transfer status/command words
0x040-0x07F	Receive sub-address 1 ...
0x780-0x7BF	Receive sub-address 30
0x7C0-0x7FF	TX transfer status/command words
0x800-0x83F	Not used
0x840-0x87F	Transfer sub-address 1 ...
0xF80-0xFBF	Transfer sub-address 30
0xFC0-0xFFF	Not used

At the start of a bus transfer the core writes the 1553B command word (if the wrtcmd bit is set in the control register) to the address subaddress*2 for receive commands and 0x7C0 + subaddress*2 for transmit commands. After a bus transfer has completed a transfer status word is written to the same location as the command word (if wrttsw bit is set in the control register). The command word of the last transfer can always be read out through the interrupt vector and command value register.

The transfer status word written to memory has the following layout:

*Table 112.*Transfer Status Word layout

Bit	**Name**	**Description**
15	USED	Always set to 1 at the end of bus transfer
14	OKAY	Set to 1 if no errors were detected
13	BUSN	Set to 0 if transfer was on bus A, to 1 if bus B
12	BROADCAST	Transfer was a broadcast command
11	LPBKERRB	The loopback logic detected error on bus B
10	LPBKERRA	The loopback logic detected error on bus A
9	ILLCMD	Illegal command
8	MEMIFERR	DMA access error did not omplete in time
7	MANERR	Manchester coding error detected
6	PARERR	Parity error detected
5	WCNTERR	Wrong number of words was received
4:0	COUNT	For sub address 1-30: number of words received/transmitted, 0 means 32
		For sub address 0 and 31: received/transmitted mode code

All mode codes are legal except the following: dynamic bus control (0), selected transmitter shutdown (20) and override selected transmitter shutdown (21).

The transfer BIT word mode code transfers a word as specified in the table below:

*Table 113.*Built In Test word

Bit	Name	Description
15	BUSINUSE	Set to 0 if transfer was on bus A, to 1 if bus B
14	LPBKERRB	The loopback logic detected error on bus B. Cleared by CLRERR.
13	LPBKERRA	The loopback logic detected error on bus A. Cleared by CLRERR.
12	SHUTDOWNB	Indicates that bus B has been shutdown
11	SHUTDOWNA	Indicates that bus A has been shutdown
10	TFLAGINH	Terminal flag inhibit setting
9	WCNTERR	Word count error has occured. Cleared by CLRERR.
8	MANERR	Manchester coding error detected. Cleared by CLRERR.
7	PARERR	Parity error detected. Cleared by CLRERR.
6	RTRTTO	RT to RT transfer timeout. Cleared by CLRERR.
5	MEMFAIL	DMA transfer not completed in time. Cleared by CLRERR.
4:0	VERSION	Core1553RT version

The AMBA AHB protection control signal HPROT is driven permanently with “0011”, i.e a not cacheable, not bufferable, privileged data access. The AMBA AHB lock signal HLOCK is driven with ‘0’.

20.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 114.*B1553RT registers

APB Address offset	Register
0x00	Status
0x04	Control
0x08	Vector word
0x0C	Interrupt vector and command value
0x10	AHB page address register

Status register (read only)

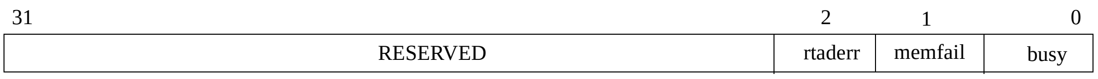


Figure 44. Status register

2:	RT address error. Incorrect RT address parity bit..
1:	Memory failure. DMA transfer did not complete in time. Cleared using CLRERR bit in control register.
0:	Busy. Indicates that the RT is busy with a transfer.

Control register

31	19	18	17	16	15	14	13	12 8	7 6	5	4	3	2	1	0
RESERVED	sa30loop	bcasten	intenbbr	extmdata	wrtcmd	wrttsw	rtaddrp	rtaddr	clkspd	clrerr	intack	tflag	ssflag	rtbusy	sreq

Figure 45. Control register

19:	Set to ‘1’ to enable internal loopback of subaddress 30. Transmits from sa 30 reads from the receive buffer for sa 30.
18:	Set to ‘1’ to enable broadcasts messages. If ‘0’ address 31 is treated as normal RT address.
17:	‘1’ enables interrupts for bad messages. If ‘0’ only good messages generates interrupts.
16:	If ‘1’ mode code data is written to / read from memory. If ‘0’ the vword register is used for transmit vector word mode code and the data for synchronize with data is discarded.
15:	If ‘1’ the command word is written to memory at the start of a bus transfer.
14:	If ‘1’ the transfer status word is written to memory at the end of a bus transfer.
13:	RT address parity bit. Odd parity over rtaddr and rtaddrp must be achieved.
12:8	RT address.
7:6	Clock speed. Should be set to indicate the clock frequency of the core. 0 - 12, 1 - 16, 2 - 20, 3 - 24 MHz
5:	Set to ‘1’ and then to ‘0’ to clear internal errors.
4:	Clear the interrupt. Should be set to ‘1’ to give a interrupt pulse on each message.
3:	Controls the terminal flag bit in the 1553B status word. This can be masked by the "inhibit terminal flag bit" mode code.
2:	Controls the subsystem flag bit in the 1553B status word.
1:	Controls the busy bit in the 1553B status word.
0:	Controls the service request bit in the 1553B status word.

Vector word register

31 16	15 0
RESERVED	vword

Figure 46. Vector word register

[15:0] Used for transmit vector word mode code if extmdata bit is ‘0’ in control register.

Interrupt vector and command value register

31	18 7	6 0
RESERVED	cmdval	intvect

Figure 47. Interrupt vector register

[18:7] For each message the CMDVAL output of Core1553BRT is latched into this register.
18 - broadcast
17 - 1 for transmit, 0 for receive
16:12 - subaddress
11:7 - word count / mode code

[6:0] Shows the value of the interrupt vector output of the Core1553BRT.

AHB page address register

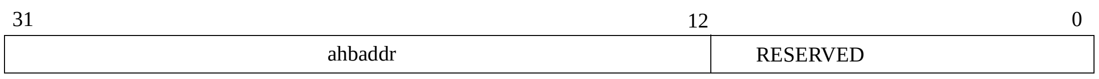


Figure 48. Address register

[31:12]: Holds the 20 top most bits of the AHB address of the allocated memory area. Resets to the value specified with the ahbaddr VHDL generic.

20.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x071. For a description of vendor and device identifiers see GRLIB IP Library User's Manual.

20.5 Configuration options

Table 115 shows the configuration options of the core (VHDL generics).

*Table 115.*Configuration options

Generic	Function	Allowed range	Default
endian	Endianness of the AHB bus (Big = 0)	0 - 1	0
ahbaddr	Reset value for address register	16#00000#-16#FFFFF#	16#00000#
clkspd	Clock speed	0 - 3	1
rtaddr	RT address	0 - 31	0
rtaddrp	RT address parity bit. Set to achieve odd parity.	0 - 1	1
wrtcmd	Write command word to memory	0 - 1	1
wrttsw	Write status word to memory	0 - 1	1
extmdata	Read/write mode code data from/to memory	0 - 1	0
intenbbr	Generate interrupts for bad messages	0 - 1	0
bcasten	Broadcast enable	0 - 1	1
sa30loop	Use sub-address 30 as loopback	0 - 1	0

All VHDL generics except endian are reset values for the corresponding bits in the wrapper control register.

20.6 Signal descriptions

Table 116 shows the interface signals of the core (VHDL ports).

*Table 116.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
B1553I	-	Input	1553 bus input signals	-
	busainp		Positive data input from the A receiver	High
	busainn		Negative data input from the A receiver	Low
	busbinp		Positive data to the B receiver	High
	busbinn		Negative data to the B receiver	Low
B1553O	-	Output	1553 bus output signals	-
	busainen		Enable for the A receiver	High
	busaoutin		Inhibit for the A transmitter	High
	busaoutp		Positive data to the A transmitter	High
	busaoutn		Negative data to the A transmitter	Low
	busbinen		Enable for the B receiver	High
	busboutin		Inhibit for the B transmitter	High
	busboutp		Positive output to the B transmitter	High
	busboutn		Negative output to the B transmitter	Low
RTI	-	Input	RT input signals	-
	cmdok		Command word validation alright	High
	useextok		Enable external command word validation	High
RTO	-	Output	RT output signals	-
	msgstart		Message process started	High
	cmdsync		Start of command word on bus	High
	syncnow		Synchronize received	High
	busreset		Reset command received	High
	cmdval		Active command	-
	cmdokout		Command word validated	High
	cmdstb		Active command value changed	High
	addrlat		Address latch enable	High
	intlat		Interrupt latch enable	High
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

20.7 Library dependencies

Table 117 shows libraries that should be used when instantiating the core (VHDL libraries).

*Table 117.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	Signal definitions
GAISLER	B1553	Signals, component	Signal and component declaration

The B1553RT depends on GRLIB, GAISLER, GR1553 and Core1553BRT.

20.8 Component declaration

The core has the following component declaration.

```
component b1553rt is
    generic (
        hindex    : integer := 0;
        pindex    : integer := 0;
        paddr     : integer := 0;
        pmask     : integer := 16#fff#;
        pirq      : integer := 0;
        ahbaddr   : integer range 0 to 16#FFFFF# := 0;
        clkspd    : integer range 0 to 3 := 1;
        rtaddr    : integer range 0 to 31 := 0;
        rtaddrp   : integer range 0 to 1 := 1;
        wrtcmd    : integer range 0 to 1 := 1;
        wrttsw    : integer range 0 to 1 := 1;
        extmdata  : integer range 0 to 1 := 0;
        intenbbr  : integer range 0 to 1 := 0;
        bcasten   : integer range 0 to 1 := 1;
        sa30loop  : integer range 0 to 1 := 0);
    port (
      rstn      : in    std_ulogic;
      clk       : in    std_ulogic;
      b1553i    : in    b1553_in_type;
      b1553o    : out   b1553_out_type;
      rti       : in    rt1553_in_type;
      rto       : out   rt1553_out_type;
      apbi      : in    apb_slv_in_type;
      apbo      : out   apb_slv_out_type;
      ahbi      : in    ahb_mst_in_type;
      ahbo      : out   ahb_mst_out_type);
  end component;
```

20.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.b1553.all;
...
signal bin : b1553_in_type;
signal bout : b1553_out_type;
signal rti : rt1553_in_type;
signal rto : rt1553_out_type;
...
rt : b1553rt
    generic map (hindex => 3, pindex => 13, paddr => 13, pmask => 16#fff#,
pirq => 3, rtaddr => 1, rtaddrp => 0, sa30loop => 1)
    port map (rstn, clkm, bin, bout, rti, rto, apbi, apbo(13), ahbmi, ahbmo(3));
```

```
        rti.useextok <= ’0’;
```

21 CAN_OC - GRLIB wrapper for OpenCores CAN Interface core

21.1 Overview

CAN_OC is GRLIB wrapper for the CAN core from Opencores. It provides a bridge between AMBA AHB and the CAN Core registers. The AHB slave interface is mapped in the AHB I/O space using the GRLIB plug&play functionality. The CAN core interrupt is routed to the AHB interrupt bus, and the interrupt number is selected through the *irq* generic. The FIFO RAM in the CAN core is implemented using the GRLIB parametrizable SYNCRAM_2P memories, assuring portability to all supported technologies.

This CAN interface implements the CAN 20.A and 2.0B protocols. It is based on the Philips SJA1000 and has a compatible register map with a few exceptions.

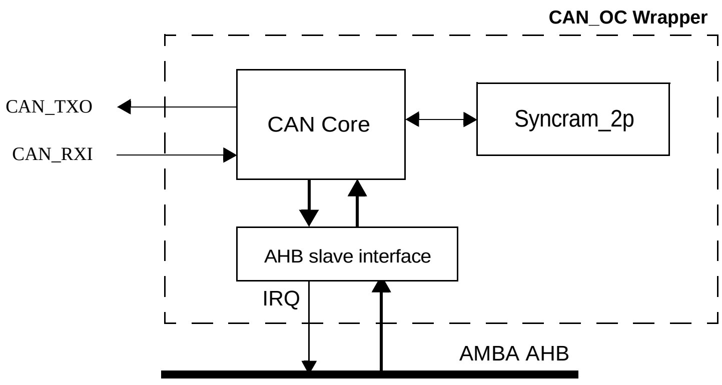


Figure 49. Block diagram

21.2 Opencores CAN controller overview

This CAN controller is based on the Philips SJA1000 and has a compatible register map with a few exceptions. It also supports both BasicCAN (PCA82C200 like) and PeliCAN mode. In PeliCAN mode the extended features of CAN 2.0B is supported. The mode of operation is chosen through the Clock Divider register.

This document will list the registers and their functionality. The Philips SJA1000 data sheet can be used as a reference if something needs clarification. See also the Design considerations chapter for differences between this core and the SJA1000.

The register map and functionality is different between the two modes of operation. First the Basic-CAN mode will be described followed by PeliCAN. Common registers (clock divisor and bus timing) are described in a separate chapter. The register map also differs depending on whether the core is in operating mode or in reset mode. When reset the core starts in reset mode awaiting configuration. Operating mode is entered by clearing the reset request bit in the command register. To re-enter reset mode set this bit high again.

21.3 AHB interface

All registers are one byte wide and the addresses specified in this document are byte addresses. Byte reads and writes should be used when interfacing with this core. The read byte is duplicated on all byte lanes of the AHB bus. The wrapper is big endian so the core expects the MSB at the lowest address.

The bit numbering in this document uses bit 7 as MSB and bit 0 as LSB.

21.4 BasicCAN mode

21.4.1 BasicCAN register map

*Table 118.*BasicCAN address allocation

Address	Operating mode		Reset mode	
	Read	**Write**	**Read**	**Write**
0	Control	Control	Control	Control
1	(0xFF)	Command	(0xFF)	Command
2	Status	-	Status	-
3	Interrupt	-	Interrupt	-
4	(0xFF)	-	Acceptance code	Acceptance code
5	(0xFF)	-	Acceptance mask	Acceptance mask
6	(0xFF)	-	Bus timing 0	Bus timing 0
7	(0xFF)	-	Bus timing 1	Bus timing 1
8	(0x00)	-	(0x00)	-
9	(0x00)	-	(0x00)	-
10	TX id1	TX id1	(0xFF)	-
11	TX id2, rtr, dlc	TX id2, rtr, dlc	(0xFF)	-
12	TX data byte 1	TX data byte 1	(0xFF)	-
13	TX data byte 2	TX data byte 2	(0xFF)	-
14	TX data byte 3	TX data byte 3	(0xFF)	-
15	TX data byte 4	TX data byte 4	(0xFF)	-
16	TX data byte 5	TX data byte 5	(0xFF)	-
17	TX data byte 6	TX data byte 6	(0xFF)	-
18	TX data byte 7	TX data byte 7	(0xFF)	-
19	TX data byte 8	TX data byte 8	(0xFF)	-
20	RX id1	-	RX id1	-
21	RX id2, rtr, dlc	-	RX id2, rtr, dlc	-
22	RX data byte 1	-	RX data byte 1	-
23	RX data byte 2	-	RX data byte 2	-
24	RX data byte 3	-	RX data byte 3	-
25	RX data byte 4	-	RX data byte 4	-
26	RX data byte 5	-	RX data byte 5	-
27	RX data byte 6	-	RX data byte 6	-
28	RX data byte 7	-	RX data byte 7	-
29	RX data byte 8	-	RX data byte 8	-
30	(0x00)	-	(0x00)	-
31	Clock divider	Clock divider	Clock divider	Clock divider

21.4.2 Control register

The control register contains interrupt enable bits as well as the reset request bit.

*Table 119.*Bit interpretation of control register (CR) (address 0)

Bit	Name	Description
CR.7	-	reserved
CR.6	-	reserved
CR.5	-	reserved
CR.4	Overrun Interrupt Enable	1 - enabled, 0 - disabled
CR.3	Error Interrupt Enable	1 - enabled, 0 - disabled
CR.2	Transmit Interrupt Enable	1 - enabled, 0 - disabled
CR.1	Receive Interrupt Enable	1 - enabled, 0 - disabled
CR.0	Reset request	Writing 1 to this bit aborts any ongoing transfer and enters reset mode. Writing 0 returns to operating mode.

21.4.3 Command register

Writing a one to the corresponding bit in this register initiates an action supported by the core.

*Table 120.*Bit interpretation of command register (CMR) (address 1)

Bit	Name	Description
CMR.7	-	reserved
CMR.6	-	reserved
CMR.5	-	reserved
CMR.4	-	not used (go to sleep in SJA1000 core)
CMR.3	Clear data overrun	Clear the data overrun status bit
CMR.2	Release receive buffer	Free the current receive buffer for new reception
CMR.1	Abort transmission	Aborts a not yet started transmission.
CMR.0	Transmission request	Starts the transfer of the message in the TX buffer

A transmission is started by writing 1 to CMR.0. It can only be aborted by writing 1 to CMR.1 and only if the transfer has not yet started. If the transmission has started it will not be aborted when setting CMR.1 but it will not be retransmitted if an error occurs.

Giving the Release receive buffer command should be done after reading the contents of the receive buffer in order to release this memory. If there is another message waiting in the FIFO a new receive interrupt will be generated (if enabled) and the receive buffer status bit will be set again.

To clear the Data overrun status bit CMR.3 must be written with 1.

21.4.4 Status register

The status register is read only and reflects the current status of the core.

*Table 121.*Bit interpretation of status register (SR) (address 2)

Bit	Name	Description
SR.7	Bus status	1 when the core is in bus-off and not involved in bus activities
SR.6	Error status	At least one of the error counters have reached or exceeded the CPU warning limit (96).
SR.5	Transmit status	1 when transmitting a message
SR.4	Receive status	1 when receiving a message
SR.3	Transmission complete	1 indicates the last message was successfully transferred.
SR.2	Transmit buffer status	1 means CPU can write into the transmit buffer
SR.1	Data overrun status	1 if a message was lost because no space in fifo.
SR.0	Receive buffer status	1 if messages available in the receive fifo.

Receive buffer status is cleared when the Release receive buffer command is given and set high if there are more messages available in the fifo.

The data overrun status signals that a message which was accepted could not be placed in the fifo because not enough space left. NOTE: This bit differs from the SJA1000 behavior and is set first when the fifo has been read out.

When the transmit buffer status is high the transmit buffer is available to be written into by the CPU. During an on-going transmission the buffer is locked and this bit is 0.

The transmission complete bit is set to 0 when a transmission request has been issued and will not be set to 1 again until a message has successfully been transmitted.

21.4.5 Interrupt register

The interrupt register signals to CPU what caused the interrupt. The interrupt bits are only set if the corresponding interrupt enable bit is set in the control register.

*Table 122.*Bit interpretation of interrupt register (IR) (address 3)

Bit	Name	Description
IR.7	-	reserved
IR.6	-	reserved
IR.5	-	reserved
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	Set when SR.1 goes from 0 to 1.
IR.2	Error interrupt	Set when the error status or bus status are changed.
IR.1	Transmit interrupt	Set when the transmit buffer is released (status bit 0->1)
IR.0	Receive interrupt	This bit is set while there are more messages in the fifo.

This register is reset on read with the exception of IR.0. Note that this differs from the SJA1000 behavior where all bits are reset on read in BasicCAN mode. This core resets the receive interrupt bit when the release receive buffer command is given (like in PeliCAN mode).

Also note that bit IR.5 through IR.7 reads as 1 but IR.4 is 0.

21.4.6 Transmit buffer

The table below shows the layout of the transmit buffer. In BasicCAN only standard frame messages can be transmitted and received (EFF messages on the bus are ignored).

*Table 123.*Transmit buffer layout

Addr	**Name**	**Bits**							
		7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
10	ID byte 1	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
11	ID byte 2	ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0
12	TX data 1	TX byte 1							
13	TX data 2	TX byte 2							
14	TX data 3	TX byte 3							
15	TX data 4	TX byte 4							
16	TX data 5	TX byte 5							
17	TX data 6	TX byte 6							
18	TX data 7	TX byte 7							
19	TX data 8	TX byte 8							

If the RTR bit is set no data bytes will be sent but DLC is still part of the frame and must be specified according to the requested frame. Note that it is possible to specify a DLC larger than 8 bytes but should not be done for compatibility reasons. If DLC > 8 still only 8 bytes can be sent.

21.4.7 Receive buffer

The receive buffer on address 20 through 29 is the visible part of the 64 byte RX FIFO. Its layout is identical to that of the transmit buffer.

21.4.8 Acceptance filter

Messages can be filtered based on their identifiers using the acceptance code and acceptance mask registers. The top 8 bits of the 11 bit identifier are compared with the acceptance code register only comparing the bits set to zero in the acceptance mask register. If a match is detected the message is stored to the fifo.

21.5 PeliCAN mode

21.5.1 PeliCAN register map

*Table 124.*PeliCAN address allocation

	Operating mode				**Reset mode**	
#	**Read**		**Write**		**Read**	**Write**
0	Mode		Mode		Mode	Mode
1	(0x00)		Command		(0x00)	Command
2	Status		-		Status	-
3	Interrupt		-		Interrupt	-
4	Interrupt enable		Interrupt enable		Interrupt enable	Interrupt enable
5	reserved (0x00)		-		reserved (0x00)	-
6	Bus timing 0		-		Bus timing 0	Bus timing 0
7	Bus timing 1		-		Bus timing 1	Bus timing 1
8	(0x00)		-		(0x00)	-
9	(0x00)		-		(0x00)	-
10	reserved (0x00)		-		reserved (0x00)	-
11	Arbitration lost capture		-		Arbitration lost capture	-
12	Error code capture		-		Error code capture	-
13	Error warning limit		-		Error warning limit	Error warning limit
14	RX error counter		-		RX error counter	RX error counter
15	TX error counter		-		TX error counter	TX error counter
16	RX FI SFF	RX FI EFF	TX FI SFF	TX FI EFF	Acceptance code 0	Acceptance code 0
17	RX ID 1	RX ID 1	TX ID 1	TX ID 1	Acceptance code 1	Acceptance code 1
18	RX ID 2	RX ID 2	TX ID 2	TX ID 2	Acceptance code 2	Acceptance code 2
19	RX data 1	RX ID 3	TX data 1	TX ID 3	Acceptance code 3	Acceptance code 3
20	RX data 2	RX ID 4	TX data 2	TX ID 4	Acceptance mask 0	Acceptance mask 0
21	RX data 3	RX data 1	TX data 3	TX data 1	Acceptance mask 1	Acceptance mask 1
22	RX data 4	RX data 2	TX data 4	TX data 2	Acceptance mask 2	Acceptance mask 2
23	RX data 5	RX data 3	TX data 5	TX data 3	Acceptance mask 3	Acceptance mask 3
24	RX data 6	RX data 4	TX data 6	TX data 4	reserved (0x00)	-
25	RX data 7	RX data 5	TX data 7	TX data 5	reserved (0x00)	-
26	RX data 8	RX data 6	TX data 8	TX data 6	reserved (0x00)	-
27	FIFO	RX data 7	-	TX data 7	reserved (0x00)	-
28	FIFO	RX data 8	-	TX data 8	reserved (0x00)	-
29	RX message counter		-		RX msg counter	-
30	(0x00)		-		(0x00)	-
31	Clock divider		Clock divider		Clock divider	Clock divider

The transmit and receive buffers have different layout depending on if standard frame format (SFF) or extended frame format (EFF) is to be transmitted/received. See the specific section below.

21.5.2 Mode register

*Table 125.*Bit interpretation of mode register (MOD) (address 0)

Bit	Name	Description
MOD.7	-	reserved
MOD.6	-	reserved
MOD.5	-	reserved
MOD.4	-	not used (sleep mode in SJA1000)
MOD.3	Acceptance filter mode	1 - single filter mode, 0 - dual filter mode
MOD.2	Self test mode	If set the controller is in self test mode
MOD.1	Listen only mode	If set the controller is in listen only mode
MOD.0	Reset mode	Writing 1 to this bit aborts any ongoing transfer and enters reset mode. Writing 0 returns to operating mode

Writing to MOD.1-3 can only be done when reset mode has been entered previously.

In Listen only mode the core will not send any acknowledgements. Note that unlike the SJA1000 the Opencores core does not become error passive and active error frames are still sent!

When in Self test mode the core can complete a successful transmission without getting an acknowledgement if given the Self reception request command. Note that the core must still be connected to a real bus, it does not do an internal loopback.

21.5.3 Command register

Writing a one to the corresponding bit in this register initiates an action supported by the core.

*Table 126.*Bit interpretation of command register (CMR) (address 1)

Bit	Name	Description
CMR.7	-	reserved
CMR.6	-	reserved
CMR.5	-	reserved
CMR.4	Self reception request	Transmits and simultaneously receives a message
CMR.3	Clear data overrun	Clears the data overrun status bit
CMR.2	Release receive buffer	Free the current receive buffer for new reception
CMR.1	Abort transmission	Aborts a not yet started transmission.
CMR.0	Transmission request	Starts the transfer of the message in the TX buffer

A transmission is started by writing 1 to CMR.0. It can only be aborted by writing 1 to CMR.1 and only if the transfer has not yet started. Setting CMR.0 and CMR.1 simultaneously will result in a so called single shot transfer, i.e. the core will not try to retransmit the message if not successful the first time.

Giving the Release receive buffer command should be done after reading the contents of the receive buffer in order to release this memory. If there is another message waiting in the FIFO a new receive interrupt will be generated (if enabled) and the receive buffer status bit will be set again.

The Self reception request bit together with the self test mode makes it possible to do a self test of the core without any other cores on the bus. A message will simultaneously be transmitted and received and both receive and transmit interrupt will be generated.

21.5.4 Status register

The status register is read only and reflects the current status of the core.

*Table 127.*Bit interpretation of command register (SR) (address 2)

Bit	Name	Description
SR.7	Bus status	1 when the core is in bus-off and not involved in bus activities
SR.6	Error status	At least one of the error counters have reached or exceeded the error warning limit.
SR.5	Transmit status	1 when transmitting a message
SR.4	Receive status	1 when receiving a message
SR.3	Transmission complete	1 indicates the last message was successfully transferred.
SR.2	Transmit buffer status	1 means CPU can write into the transmit buffer
SR.1	Data overrun status	1 if a message was lost because no space in fifo.
SR.0	Receive buffer status	1 if messages available in the receive fifo.

Receive buffer status is cleared when there are no more messages in the fifo. The data overrun status signals that a message which was accepted could not be placed in the fifo because not enough space left. NOTE: This bit differs from the SJA1000 behavior and is set first when the fifo has been read out.

When the transmit buffer status is high the transmit buffer is available to be written into by the CPU. During an on-going transmission the buffer is locked and this bit is 0.

The transmission complete bit is set to 0 when a transmission request or self reception request has been issued and will not be set to 1 again until a message has successfully been transmitted.

21.5.5 Interrupt register

The interrupt register signals to CPU what caused the interrupt. The interrupt bits are only set if the corresponding interrupt enable bit is set in the interrupt enable register.

*Table 128.*Bit interpretation of interrupt register (IR) (address 3)

Bit	Name	Description
IR.7	Bus error interrupt	Set if an error on the bus has been detected
IR.6	Arbitration lost interrupt	Set when the core has lost arbitration
IR.5	Error passive interrupt	Set when the core goes between error active and error passive
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	Set when data overrun status bit is set
IR.2	Error warning interrupt	Set on every change of the error status or bus status
IR.1	Transmit interrupt	Set when the transmit buffer is released
IR.0	Receive interrupt	Set while the fifo is not empty.

This register is reset on read with the exception of IR.0 which is reset when the fifo has been emptied.

21.5.6 Interrupt enable register

In the interrupt enable register the separate interrupt sources can be enabled/disabled. If enabled the corresponding bit in the interrupt register can be set and an interrupt generated.

*Table 129.*Bit interpretation of interrupt enable register (IER) (address 4)

Bit	Name	Description
IR.7	Bus error interrupt	1 - enabled, 0 - disabled
IR.6	Arbitration lost interrupt	1 - enabled, 0 - disabled
IR.5	Error passive interrupt	1 - enabled, 0 - disabled
IR.4	-	not used (wake-up interrupt of SJA1000)
IR.3	Data overrun interrupt	1 - enabled, 0 - disabled
IR.2	Error warning interrupt	1 - enabled, 0 - disabled.
IR.1	Transmit interrupt	1 - enabled, 0 - disabled
IR.0	Receive interrupt	1 - enabled, 0 - disabled

21.5.7 Arbitration lost capture register

*Table 130.*Bit interpretation of arbitration lost capture register (ALC) (address 11)

Bit	Name	Description
ALC.7-5	-	reserved
ALC.4-0	Bit number	Bit where arbitration is lost

When the core loses arbitration the bit position of the bit stream processor is captured into arbitration lost capture register. The register will not change content again until read out.

21.5.8 Error code capture register

*Table 131.*Bit interpretation of error code capture register (ECC) (address 12)

Bit	Name	Description
ECC.7-6	Error code	Error code number
ECC.5	Direction	1 - Reception, 0 - transmission error
ECC.4-0	Segment	Where in the frame the error occurred

When a bus error occurs the error code capture register is set according to what kind of error occurred, if it was while transmitting or receiving and where in the frame it happened. As with the ALC register the ECC register will not change value until it has been read out. The table below shows how to interpret bit 7-6 of ECC.

*Table 132.*Error code interpretation

ECC.7-6	Description
0	Bit error
1	Form error
2	Stuff error
3	Other

Bit 4 downto 0 of the ECC register is interpreted as below

*Table 133.*Bit interpretation of ECC.4-0

ECC.4-0	**Description**
0x03	Start of frame
0x02	ID.28 - ID.21
0x06	ID.20 - ID.18
0x04	Bit SRTR
0x05	Bit IDE
0x07	ID.17 - ID.13
0x0F	ID.12 - ID.5
0x0E	ID.4 - ID.0
0x0C	Bit RTR
0x0D	Reserved bit 1
0x09	Reserved bit 0
0x0B	Data length code
0x0A	Data field
0x08	CRC sequence
0x18	CRC delimiter
0x19	Acknowledge slot
0x1B	Acknowledge delimiter
0x1A	End of frame
0x12	Intermission
0x11	Active error flag
0x16	Passive error flag
0x13	Tolerate dominant bits
0x17	Error delimiter
0x1C	Overload flag

21.5.9 Error warning limit register

This registers allows for setting the CPU error warning limit. It defaults to 96. Note that this register is only writable in reset mode.

21.5.10 RX error counter register (address 14)

This register shows the value of the rx error counter. It is writable in reset mode. A bus-off event resets this counter to 0.

21.5.11 TX error counter register (address 15)

This register shows the value of the tx error counter. It is writable in reset mode. If a bus-off event occurs this register is initialized as to count down the protocol defined 128 occurrences of the bus-free signal and the status of the bus-off recovery can be read out from this register. The CPU can force a bus-off by writing 255 to this register. Note that unlike the SJA1000 this core will signal bus-off immediately and not first when entering operating mode. The bus-off recovery sequence starts when entering operating mode after writing 255 to this register in reset mode.

21.5.12 Transmit buffer

The transmit buffer is write-only and mapped on address 16 to 28. Reading of this area is mapped to the receive buffer described in the next section. The layout of the transmit buffer depends on whether a standard frame (SFF) or an extended frame (EFF) is to be sent as seen below.

Table 134.

#	**Write (SFF)**	**Write(EFF)**
16	TX frame information	TX frame information
17	TX ID 1	TX ID 1
18	TX ID 2	TX ID 2
19	TX data 1	TX ID 3
20	TX data 2	TX ID 4
21	TX data 3	TX data 1
22	TX data 4	TX data 2
23	TX data 5	TX data 3
24	TX data 6	TX data 4
25	TX data 7	TX data 5
26	TX data 8	TX data 6
27	-	TX data 7
28	-	TX data 8

TX frame information (this field has the same layout for both SFF and EFF frames)

*Table 135.*TX frame information address 16

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
FF	RTR	-	-	DLC.3	DLC.2	DLC.1	DLC.0

Bit 7 - FF selects the frame format, i.e. whether this is to be interpreted as an extended or standard frame. 1 = EFF, 0 = SFF.
Bit 6 - RTR should be set to 1 for an remote transmission request frame.
Bit 5:4 - are don’t care.
Bit 3:0 - DLC specifies the Data Length Code and should be a value between 0 and 8. If a value greater than 8 is used 8 bytes will be transmitted.

TX identifier 1 (this field is the same for both SFF and EFF frames)

*Table 136.*TX identifier 1 address 17

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Bit 7:0 - The top eight bits of the identifier.

TX identifier 2, SFF frame

*Table 137.*TX identifier 2 address 18

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.20	ID.19	ID.18	-	-	-	-	-

Bit 7:5 - Bottom three bits of an SFF identifier.
Bit 4:0 - Don’t care.

TX identifier 2, EFF frame

*Table 138.*TX identifier 2 address 18

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Bit 7:0 - Bit 20 downto 13 of 29 bit EFF identifier.

TX identifier 3, EFF frame

*Table 139.*TX identifier 3 address 19

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Bit 7:0 - Bit 12 downto 5 of 29 bit EFF identifier.

TX identifier 4, EFF frame

*Table 140.*TX identifier 4 address 20

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
ID.4	ID.3	ID.2	ID.1	ID.0	-	-	-

Bit 7:3 - Bit 4 downto 0 of 29 bit EFF identifier
Bit 2:0 - Don’t care

Data field

For SFF frames the data field is located at address 19 to 26 and for EFF frames at 21 to 28. The data is transmitted starting from the MSB at the lowest address.

21.5.13 Receive buffer

Table 141.

#	Read (SFF)	Read (EFF)
16	RX frame information	RX frame information
17	RX ID 1	RX ID 1
18	RX ID 2	RX ID 2
19	RX data 1	RX ID 3
20	RX data 2	RX ID 4
21	RX data 3	RX data 1
22	RX data 4	RX data 2
23	RX data 5	RX data 3
24	RX data 6	RX data 4
25	RX data 7	RX data 5
26	RX data 8	RX data 6
27	RX FI of next message in fifo	RX data 7
28	RX ID1 of next message in fifo	RX data 8

RX frame information (this field has the same layout for both SFF and EFF frames)

*Table 142.*RX frame information address 16

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0

Bit 7 - Frame format of received message. 1 = EFF, 0 = SFF.
Bit 6 - 1 if RTR frame.
Bit 5:4 - Always 0.
Bit 3:0 - DLC specifies the Data Length Code.

RX identifier 1(this field is the same for both SFF and EFF frames)

*Table 143.*RX identifier 1 address 17

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

Bit 7:0 - The top eight bits of the identifier.

RX identifier 2, SFF frame

*Table 144.*RX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	RTR	0	0	0	0

Bit 7:5 - Bottom three bits of an SFF identifier.
Bit 4 - 1 if RTR frame.
Bit 3:0 - Always 0.

RX identifier 2, EFF frame

*Table 145.*RX identifier 2 address 18

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

Bit 7:0 - Bit 20 downto 13 of 29 bit EFF identifier.

RX identifier 3, EFF frame

*Table 146.*RX identifier 3 address 19

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

Bit 7:0 - Bit 12 downto 5 of 29 bit EFF identifier.

RX identifier 4, EFF frame

*Table 147.*RX identifier 4 address 20

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

Bit 7:3 -	Bit 4 downto 0 of 29 bit EFF identifier
Bit 2-	1 if RTR frame
Bit 1:0 -	Don’t care

Data field

For received SFF frames the data field is located at address 19 to 26 and for EFF frames at 21 to 28.

21.5.14 Acceptance filter

The acceptance filter can be used to filter out messages not meeting certain demands. If a message is filtered out it will not be put into the receive fifo and the CPU will not have to deal with it.

There are two different filtering modes, single and dual filter. Which one is used is controlled by bit 3 in the mode register. In single filter mode only one 4 byte filter is used. In dual filter two smaller filters are used and if either of these signals a match the message is accepted. Each filter consists of two parts the acceptance code and the acceptance mask. The code registers are used for specifying the pattern to match and the mask registers specify don’t care bits. In total eight registers are used for the acceptance filter as shown in the table below. Note that they are only read/writable in reset mode.

*Table 148.*Acceptance filter registers

Address	**Description**
16	Acceptance code 0 (ACR0)
17	Acceptance code 1 (ACR1)
18	Acceptance code 2 (ACR2)
19	Acceptance code 3 (ACR3)
20	Acceptance mask 0 (AMR0)
21	Acceptance mask 1 (AMR1)
22	Acceptance mask 2 (AMR2)
23	Acceptance mask 3 (AMR3)

Single filter mode, standard frame

When receiving a standard frame in single filter mode the registers ACR0-3 are compared against the incoming message in the following way:

ACR0.7-0 & ACR1.7-5 are compared to ID.28-18
ACR1.4 is compared to the RTR bit.
ACR1.3-0 are unused.
ACR2 & ACR3 are compared to data byte 1 & 2.

The corresponding bits in the AMR registers selects if the results of the comparison doesn’t matter. A set bit in the mask register means don’t care.

Single filter mode, extended frame

When receiving an extended frame in single filter mode the registers ACR0-3 are compared against the incoming message in the following way:

ACR0.7-0 & ACR1.7-0 are compared to ID.28-13
ACR2.7-0 & ACR3.7-3 are compared to ID.12-0
ACR3.2 are compared to the RTR bit
ACR3.1-0 are unused.

The corresponding bits in the AMR registers selects if the results of the comparison doesn’t matter. A set bit in the mask register means don’t care.

Dual filter mode, standard frame

When receiving a standard frame in dual filter mode the registers ACR0-3 are compared against the incoming message in the following way:

Filter 1

ACR0.7-0 & ACR1.7-5 are compared to ID.28-18
ACR1.4 is compared to the RTR bit.
ACR1.3-0 are compared against upper nibble of data byte 1
ACR3.3-0 are compared against lower nibble of data byte 1

Filter 2

ACR2.7-0 & ACR3.7-5 are compared to ID.28-18
ACR3.4 is compared to the RTR bit.

The corresponding bits in the AMR registers selects if the results of the comparison doesn’t matter. A set bit in the mask register means don’t care.

Dual filter mode, extended frame

When receiving a standard frame in dual filter mode the registers ACR0-3 are compared against the incoming message in the following way:

Filter 1

ACR0.7-0 & ACR1.7-0 are compared to ID.28-13

Filter 2

ACR2.7-0 & ACR3.7-0 are compared to ID.28-13

The corresponding bits in the AMR registers selects if the results of the comparison doesn’t matter. A set bit in the mask register means don’t care.

21.5.15 RX message counter

The RX message counter register at address 29 holds the number of messages currently stored in the receive fifo. The top three bits are always 0.

21.6 Common registers

There are three common registers with the same addresses and the same functionality in both BasiCAN and PeliCAN mode. These are the clock divider register and bus timing register 0 and 1.

21.6.1 Clock divider register

The only real function of this register in the GRLIB version of the Opencores CAN is to choose between PeliCAN and BasiCAN. The clkout output of the Opencore CAN core is not connected and it is its frequency that can be controlled with this register.

*Table 149.*Bit interpretation of clock divider register (CDR) (address 31)

Bit	Name	Description
CDR.7	CAN mode	1 - PeliCAN, 0 - BasiCAN
CDR.6	-	unused (cbp bit of SJA1000)
CDR.5	-	unused (rxinten bit of SJA1000)
CDR.4	-	reserved
CDR.3	Clock off	Disable the clkout output
CDR.2-0	Clock divisor	Frequency selector

21.6.2 Bus timing 0

*Table 150.*Bit interpretation of bus timing 0 register (BTR0) (address 6)

Bit	Name	Description
BTR0.7-6	SJW	Synchronization jump width
BTR0.5-0	BRP	Baud rate prescaler

The CAN core system clock is calculated as:

$t_{scl} = 2*t_{clk}*(BRP+1)$

where t_{clk} is the system clock.

The sync jump width defines how many clock cycles (t_{scl}) a bit period may be adjusted with by one re-synchronization.

21.6.3 Bus timing 1

*Table 151.*Bit interpretation of bus timing 1 register (BTR1) (address 7)

Bit	Name	Description
BTR1.7	SAM	1 - The bus is sampled three times, 0 - single sample point
BTR1.6-4	TSEG2	Time segment 2
BTR1.3-0	TSEG1	Time segment 1

The CAN bus bit period is determined by the CAN system clock and time segment 1 and 2 as shown in the equations below:

$$t_{tseg1} = t_{scl} * (TSEG1+1)$$
$$t_{tseg2} = t_{scl} * (TSEG2+1)$$
$$t_{bit} = t_{tseg1} + t_{tseg2} + t_{scl}$$

The additional t_{scl} term comes from the initial sync segment. Sampling is done between TSEG1 and TSEG2 in the bit period.

21.7 Design considerations

This section lists known differences between this CAN controller and SJA1000 on which is it based:

- All bits related to sleep mode are unavailable
- Output control and test registers do not exist (reads 0x00)
- Clock divisor register bit 6 (CBP) and 5 (RXINTEN) are not implemented
- Overrun irq and status not set until fifo is read out

BasicCAN specific differences:

- The receive irq bit is not reset on read, works like in PeliCAN mode
- Bit CR.6 always reads 0 and is not a flip flop with no effect as in SJA1000

PeliCAN specific differences:

- Writing 256 to tx error counter gives immediate bus-off when still in reset mode
- Read Buffer Start Address register does not exist
- Addresses above 31 are not implemented (i.e. the internal RAM/FIFO access)
- The core transmits active error frames in Listen only mode

21.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x019. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

21.9 Configuration options

Table 152 shows the configuration options of the core (VHDL generics).

*Table 152.*Configuration options

Generic	Function	Allowed range	Default
slvndx	AHB slave bus index	0 - NAHBSLV-1	0
ioaddr	The AHB I/O area base address. Compared with bit 19-8 of the 32-bit AHB address.	0 - 16#FFF#	16#FFF#
iomask	The I/O area address mask. Sets the size of the I/O area and the start address together with ioaddr.	0 - 16#FFF#	16#FF0#
irq	Interrupt number	0 - NAHBIRQ-1	0
memtech	Technology to implement on-chip RAM	0	0 - NTECH

21.10 Signal descriptions

Table 153 shows the interface signals of the core (VHDL ports).

*Table 153.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK		Input	AHB clock	
RESETN		Input	Reset	Low
AHBSI	*	Input	AMBA AHB slave inputs	-
AHBSO	*	Input	AMBA AHB slave outputs	
CAN_RXI		Input	CAN receiver input	High
CAN_TXO		Output	CAN transmitter output	High

*1) see AMBA specification

21.11 Library dependencies

Table 154 shows libraries that should be used when instantiating the core.

*Table 154.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Types	AMBA signal type definitions
GAISLER	CAN	Component	Component declaration

21.12 Component declaration

```
library grlib;
use grlib.amba.all;
use gaisler.can.all;

component can_oc
   generic (
    slvndx    : integer := 0;
    ioaddr    : integer := 16#000#;
    iomask    : integer := 16#FF0#;
    irq       : integer := 0;
    memtech   : integer := 0);
   port (
      resetn  : in  std_logic;
      clk     : in  std_logic;
      ahbsi   : in  ahb_slv_in_type;
      ahbso   : out ahb_slv_out_type;
      can_rxi : in  std_logic;
      can_txo : out std_logic
);
   end component;
```

22 CLKGEN - Clock generation

22.1 Overview

The CLKGEN clock generator implements internal clock generation and buffering.

22.2 Technology specific clock generators

22.2.1 Overview

The core is a wrapper that instantiates technology specific primitives depending on the value of the *tech* VHDL generic. Each supported technology has its own subsection below. Table 155 lists the subsection applicable for each technology setting. The table is arranged after the technology's numerical value in GRLIB. The subsections are ordered in alphabetical order after technology vendor.

*Table 155.*Overview of technology specific clock generator sections

Technology	Numerical value	Comment	Section
inferred	0	Default when no technology specific generator is available.	22.2.2
virtex	1		22.2.11
virtex2	2		22.2.12
memvirage	3	No technology specific clock generator available.	22.2.2
axcel	4		22.2.3
proasic	5		22.2.3
atc18s	6	No technology specific clock generator available.	22.2.2
altera	7		22.2.6
umc	8	No technology specific clock generator available.	22.2.2
rhumc	9		22.2.9
apa3	10		22.2.4
spartan3	11		22.2.10
ihp25	12	No technology specific clock generator available.	22.2.2
rhlib18t	13		22.2.8
virtex4	14		22.2.12
lattice	15	No technology specific clock generator available.	22.2.2
ut25	16	No technology specific clock generator available.	22.2.2
spartan3e	17		22.2.10
peregrine	18	No technology specific clock generator available.	22.2.2
memartisan	19	No technology specific clock generator available.	22.2.2
virtex5	20		22.2.13
custom1	21	No technology specific clock generator available.	22.2.2
ihp25rh	22	No technology specific clock generator available.	22.2.2
stratix1	23		22.2.6
stratix2	24		22.2.6
eclipse	25	No technology specific clock generator available.	22.2.2
stratix3	26		22.2.7
cyclone3	27		22.2.5
memvirage90	28	No technology specific clock generator available.	22.2.2
tsmc90	29	No technology specific clock generator available.	22.2.2
easic90	30	No technology specific clock generator available.	22.2.2

22.2.2 Generic technology

This implementation is used when the clock generator does not support instantiation of technology specific primitives or when the inferred technology has been selected.

This implementation connects the input clock, CLKIN or PCICLKIN depending on the *pcien* and *pci-sysclk* VHDL generic, to the SDCLK, CLK1XU, and CLK outputs. The CLKN output is driven by the inverted input clock. The PCICLK output is directly driven by PCICLKIN. Both clock lock signals are always driven to '1' and the CLK2X output is always driven to '0'.

In simulation, CLK, CLKN and CLK1XU transitions are skewed 1 ns relative to the SDRAM clock output.

22.2.3 Actel Axcellerator/ProASIC

Generics used in this technology:	pcisysclk
Instantiated technology primitives:	None
Signals not driven in this technology:	clk4x, clk1xu, clk2xu

This technology selection does not instantiate any technology specific primitives. The core's clock output, CLK, is driven by the CLKIN or PCICLKIN input depending on the value of VHDL generics *pcien* and *pcisysclk.*

The PCICLK is always directly connected to PCICLKIN. Outputs SDCLK, CLKN and CLK2X, are driven to ground. Both clock lock signals, CGO.CLKLOCK and CGO.PCILOCK, are always driven high.

22.2.4 Actel ProASIC3

Generics used in this technology:	clk_mul, clk_div, clk_odiv, pcisysclk, pcien, freq
Instantiated technology primitives:	PLLINT, PLL
Signals not driven in this technology:	sdclk, clk2x, clk4x, clk1xu, clk2xu

This technology instantiates a PLL and a PLLINT to generate the main clock. The instantiation of a PLLINT macro allows the PLL reference clock to be driven from an I/O that is routed through the regular FPGA routing fabric. Figure 50 shows the instantiated primitives, the PLL EXTFB input is not shown and the EXTFB port on the instantiated component is always tied to ground. The figure shows which of the core's output ports that are driven by the PLL. The PCICLOCK will directly connected to PCICLKIN if VHDL generic *pcien* is non-zero, while CGO.PCILOCK is always driven high. The VHDL generics *pcien* and *pcisysclk* are used to select the reference clock. The values driven on the PLL inputs are listed in tables 156 and 157.

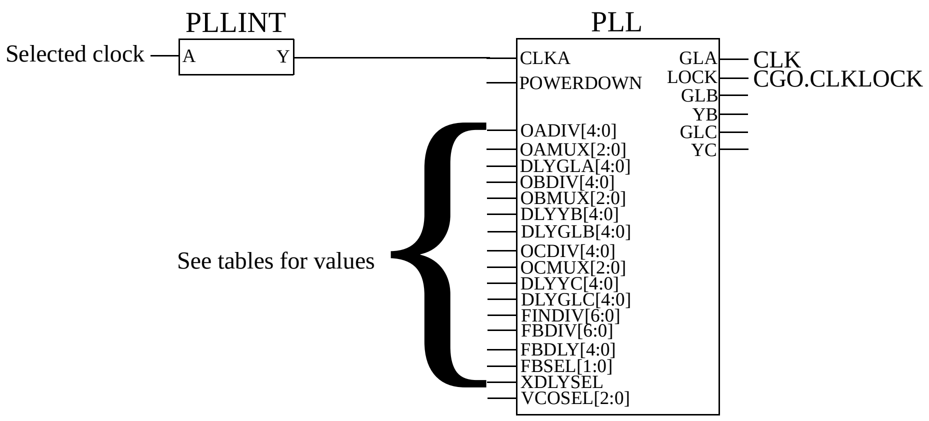


Figure 50. Actel ProASIC3 clock generation

*Table 156.*Constant input signals on Actel ProASIC3 PLL

Signal name	Value	Comment
OADIV[4:0]	VHDL generic *clk_odiv* - 1	Output divider
OAMUX[2:0]	0b100	Post-PLL MUXA
DLYGLA[4:0]	0	Delay on Global A
OBDIV[4:0]	0	Output divider
OBMUX[2:0]	0	Post-PLL MUXB
DLYYB[4:0]	0	Delay on YB
DLYGLB[4:0]	0	Delay on Global B
OCDIV[4:0]	0	Output divider
OCMUX[2:0]	0	Post-PLL MUXC
DLYYC[4:0]	0	Delay on YC
DLYGLC[4:0]	0	Delay on Global C
FINDIV[6:0]	VHDL generic *clk_div* - 1	Input divider
FBDIV[6:0]	VHDL generic *clk_mul* - 1	Feedback divider
FBDLY[4:0]	0	Feedback delay
FBSEL[1:0]	0b01	2-bit PLL feedback MUX
XDLYSEL	0	1-bit PLL feedback MUX
VCOSEL[2:0]	*See table 157 below*	VCO gear control. Selects one of four frequency ranges.

The PLL primitive has one parameter, VCOFREQUENCY, which is calculated with:

$$VCOFREQUENCY = \frac{freq \cdot clkmul}{clkdiv} / 1000$$

The calculations are performed with integer precision. This value is also used to determine the value driven on PLL input VCOSEL[2:0]. Table 157 lists the signal value depending on the value of VCOFREQUENCY.

*Table 157.*VCOSEL[2:0] on Actel ProASIC3 PLL

Value of VCOFREQUENCY	Value driven on VCOSEL[2:0]
< 44	0b000
< 88	0b010
< 175	0b100
>= 175	0b110

22.2.5 Altera Cyclone III

Generics used in this technology:	clk_mul, clk_div, sdramen, pcien, pcisysclk, freq, clk2xen
Instantiated technology primitives:	ALTPLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu

This technology instantiates an ALTPLL primitive to generate the required clocks, see figure 51. The ALTPLL attributes are listed in table 158. As can be seen in this table the attributes OPERATION_MODE and COMPENSATE_CLOCK depend on the VHDL generic *sdramen*.

*Table 158.*Altera Cyclone III ALTPLL attributes

Attribute name*	**Value with *sdramen* = 1**	**Value with *sdramen* = 0**
INTENDED_DEVICE_FAMILY	“Cyclone III”	“Cyclone III”
OPERATION_MODE	“ZERO_DELAY_BUFFER”	“NORMAL”
COMPENSATE_CLOCK	“CLK1”	“clock0”
INCLK0_INPUT_FREQUENCY	1000000000 / (VHDL generic *freq*)	1000000000 / (VHDL generic *freq*)
WIDTH_CLOCK	5	5
CLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK1_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK1_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK2_MULTIPLY_BY	VHDL generic *clk_mul * 2*	VHDL generic *clk_mul * 2*
CLK2_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*

*Any attributes not listed are assumed to have their default value

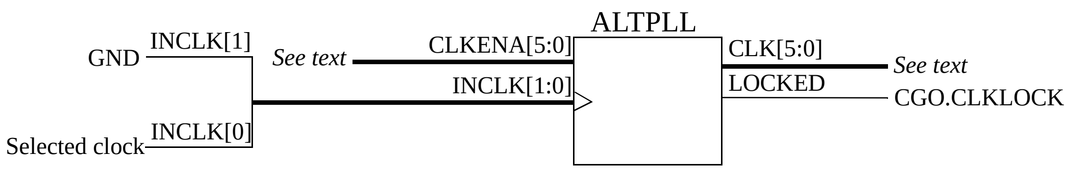


Figure 51. Altera Cyclone III ALTPLL

The value driven on the ALTPLL clock enable signal is dependent on the VHDL generics *clk2xen* and *sdramen*, table 159 lists the effect of these generics.

*Table 159.*Effect of VHDL generics *clk2xen* and *sdramen* on ALTPLL clock enable input

Value of sdramen	**Value of clk2xen**	**Value of CLKENA[5:0]**
0	0	0b000001
0	1	0b000101
1	0	0b000011
1	1	0b000111

Table 160 lists the connections of the core's input and outputs to the ALTPLL ports.

*Table 160.*Connections between core ports and ALTPLL ports

Core signal	Core direction	ALTPLL signal
CLKIN/PCICLKIN*	Input	INCLK[0]
CLK	Output	CLK[0]
CLKN	Output	$\overline{\text{CLK[0]}}$ (CLK[0] through an inverter)
CLK2X	Output	CLK[2]
SDCLK	Output	CLK[1]
CGO.CLKLOCK	Output	LOCKED

* Depending on VHDL generics PCIEN and PCISYSCLK, as described below.

The clocks can be generated using either the CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. If *pcien* is 0 or *pcisysclk* is 0 the input clock to the ALTPLL will be CLKIN. If *pcien* is non-zero and *pcisysclk* is 1 the input to the ALTPLL will be PCICLKIN.

The PCICLK output will connected to the PCICLKIN input if VHDL generic *pcien* is non-zero. Otherwise the PCICLK output will be driven to ground. The CGO.PCILOCK signal is always driven high.

22.2.6 Altera Stratix 1/2

Generics used in this technology:	clk_mul, clk_div, sdramen, pcien, pcisysclk, freq, clk2xen
Instantiated technology primitives:	ALTPLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu

This technology instantiates an ALTPLL primitive to generate the required clocks, see figure 52. The ALTPLL attributes are listed in table 161. As can be seen in this table the OPERATION_MODE attribute depends on the VHDL generic *sdramen*.

*Table 161.*Altera Stratix 1/2 ALTPLL attributes

Attribute name*	Value with *sdramen* = 1	Value with *sdramen* = 0
OPERATION_MODE	"ZERO_DELAY_BUFFER"	"NORMAL"
INCLK0_INPUT_FREQUENCY	1000000000 / (VHDL generic *freq*)	1000000000 / (VHDL generic *freq*)
WIDTH_CLOCK	6	6
CLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
CLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
CLK1_MULTIPLY_BY	VHDL generic *clk_mul* * 2	VHDL generic *clk_mul* * 2
CLK1_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*
EXTCLK0_MULTIPLY_BY	VHDL generic *clk_mul*	VHDL generic *clk_mul*
EXTCLK0_DIVIDE_BY	VHDL generic *clk_div*	VHDL generic *clk_div*

*Any attributes not listed are assumed to have their default value

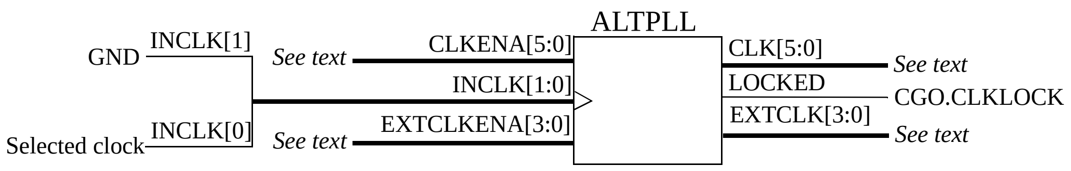


Figure 52. Altera Stratix 1/2 ALTPLL

The values driven on the ALTPLL clock enable signals are dependent on the VHDL generic *clk2xen*, table 162 lists the effect of *clk2xen*.

*Table 162.*Effect of VHDL generic *clk2xen* on ALTPLL clock enable inputs

Signal	**Value with *clk2xen* = 0**	**Value with *clk2xen* /= 0**
CLKENA[5:0]	0b000001	0b000011
EXTCLKENA[3:0]	0b0001	0b0011

Table 163 lists the connections of the core's input and outputs to the ALTPLL ports.

*Table 163.*Connections between core ports and ALTPLL ports

Core signal	**Core direction**	**ALTPLL signal**
CLKIN/PCICLKIN*	Input	INCLK[0]
CLK	Output	CLK[0]
CLKN	Output	$\overline{\text{CLK[0]}}$ (CLK[0] through an inverter)
CLK2X	Output	CLK[1]
SDCLK	Output	EXTCLK[0]
CGO.CLKLOCK	Output	LOCKED

* Depending on VHDL generics PCIEN and PCISYSCLK, as described below.

The clocks can be generated using either the CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. If *pcien* is 0 or *pcisysclk* is 0 the input clock to the ALTPLL will be CLKIN. If *pcien* is non-zero and *pcisysclk* is 1 the input to the ALTPLL will be PCICLKIN.

The PCICLK output will connected to the PCICLKIN input if VHDL generic *pcien* is non-zero. Otherwise the PCICLK output will be driven to ground. The CGO.PCILOCK signal is always driven high.

22.2.7 Altera Stratix 3

This technology is not fully supported at this time.

22.2.8 RHLIB18t

Generics used in this technology:	clk_mul, clk_div
Instantiated technology primitives:	lfdll_top
Signals not driven in this technology:	-

Please contact Gaisler Research for information concerning the use of this clock generator.

22.2.9 RHUMC

Generics used in this technology:	None
Instantiated technology primitives:	pll_ip
Signals not driven in this technology:	-

Please contact Gaisler Research for information concerning the use of this clock generator.

22.2.10 Xilinx Spartan3/e

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x

The main clock is generated with a DCM which is instantiated with the attributes listed in table 164. The input clock source connected to the CLKIN input is either the core's CLKIN input or the PCI-CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 53.

*Table 164.*Spartan 3/e DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_mul*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_div*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"2X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

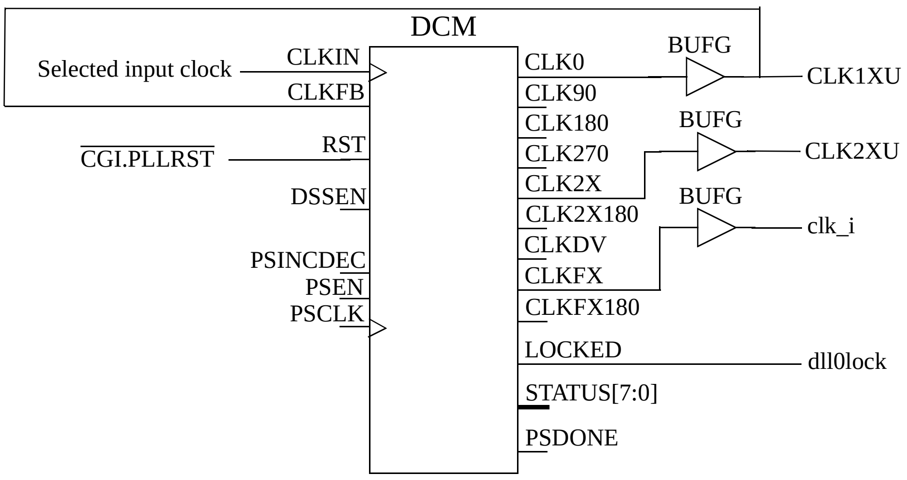


Figure 53. Spartan 3/e generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 54 is instantiated. The attributes of this DCM are the same as in table 164, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2 and the CLK_FEEDBACK attribute is set to "1X". The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 55 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

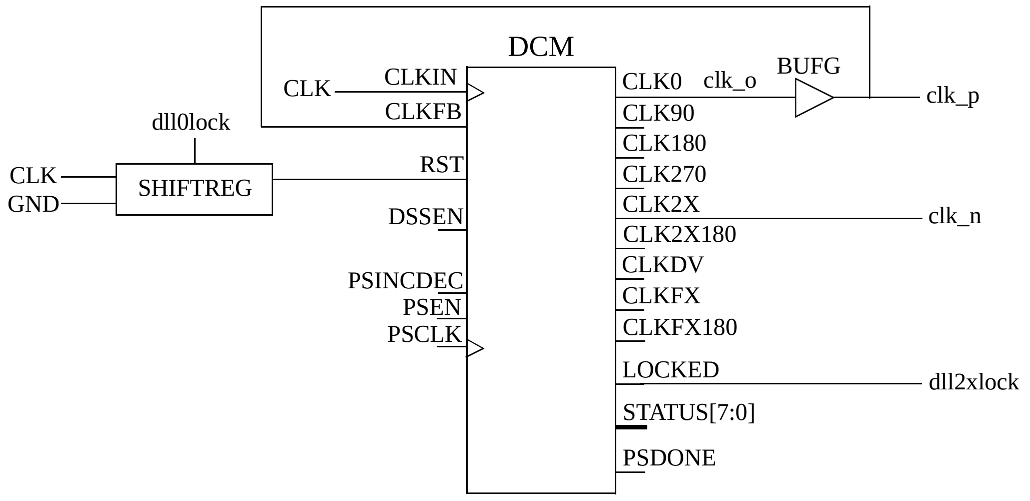


Figure 54. Spartan 3/e generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

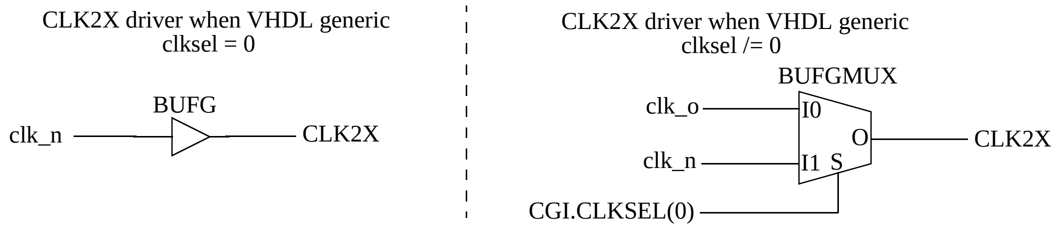

CLK2X driver when VHDL generic clksel = 0
BUFG
clk_n
CLK2X
CLK2X driver when VHDL generic clksel /= 0
BUFGMUX
clk_o
I0
O
CLK2X
clk_n
I1 S
CGI.CLKSEL(0)

Figure 55. Spartan 3/e selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. The core's CLKN output is driven by the selected signal through an inverter. Figure 56 illustrates the connections.

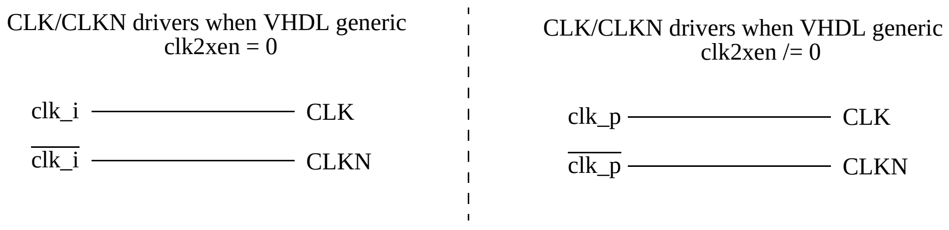

CLK/CLKN drivers when VHDL generic clk2xen = 0
clk_i
CLK
$\overline{\text{clk\_i}}$
CLKN
CLK/CLKN drivers when VHDL generic clk2xen /= 0
clk_p
CLK
$\overline{\text{clk\_p}}$
CLKN

Figure 56. Spartan 3/e clock generator outputs CLK and CLKN

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven by the main clock DCM's CLK2X output via a BUFG, please see figure 57.

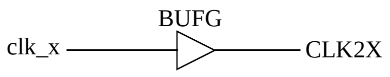


Figure 57. Spartan 3/e generation of CLK2X clock when VHDL generic *clk2xen* is zero

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 58 is instantiated. This DCM has the same attributes as the CLK2X DCM. The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 55. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 57.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

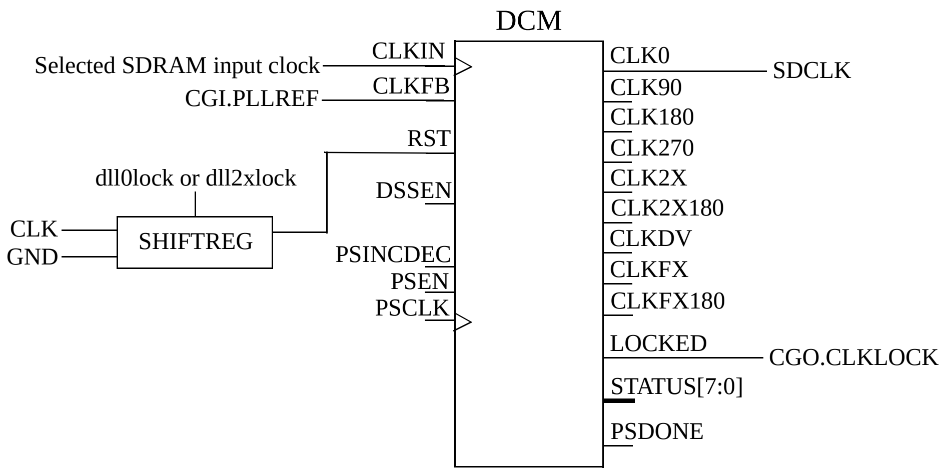


Figure 58. Spartan 3/e generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core’s SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the *clk2xen* VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 55, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core’s CLK output.

When the *sdramen* VHDL generic is set to 0 the core’s CGO.CLKLOCK output is connected to the CLK2X DCM’s LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM’s LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 59 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

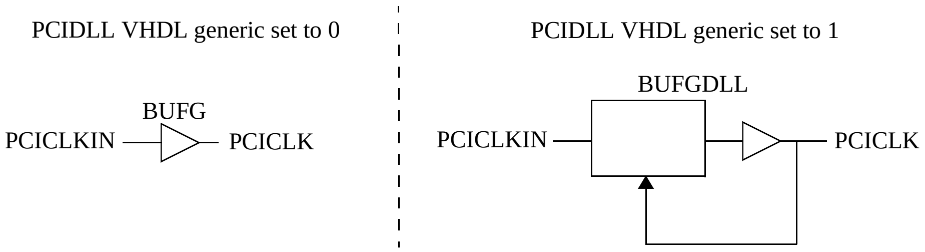


Figure 59. Spartan 3/e PCI clock generation

22.2.11 Xilinx Virtex

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk
Instantiated technology primitives:	BUFG, BUFGDLL, CLKDLL
Signals not driven in this technology:	clk4x, clk1xu, clk2xu

The main clock is generated with the help of a CLKDLL. Figure 60 below shows how the CLKDLL primitive is connected. The input clock source is either the core's CLKIN input or the PCICLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The figure shows three potential drivers of the BUFG driving the output clock CLK, the driver is selected via the VHDL generics *clk_mul* and *clk_div*. If *clk_mul/clk_div* is equal to 2 the CLK2X output is selected, if *clk_div/clk_mul* equals 2 the CLKDV output is selected, otherwise the CLK0 output drives the BUFG. The inverted main clock output, CLKN, is the BUFG output connected via an inverter.

The figure shows a dashed line connecting the CLKDLL's LOCKED output to the core output CGO.CLKLOCK. The driver of the CGO.CLKLOCK output depends on the instantiation of a CLKDLL for the SDRAM clock. See description of the SDRAM clock below.

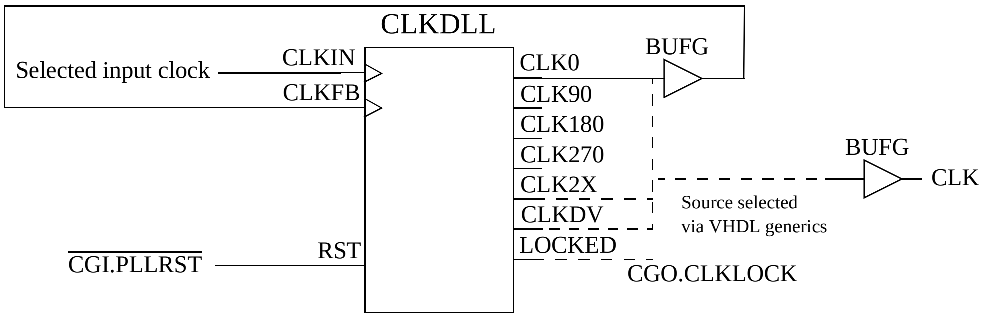


Figure 60. Virtex generation of main clock

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback, VHDL generic *noclkfb* set to 0, a CLKDLL is instantiated as depicted in figure 61. Note how the CLKDLL's RST input is connected via a shift register clocked by the main clock. The shift register is loaded with all '1' when the LOCKED signal of the main clock CLKDLL is low. When the LOCKED signal from the main clock CLKDLL is asserted the SDRAM CLKDLL's RST input will be deasserted after four main clock cycles.

For all other configurations the SDRAM clock is driven by the main clock and the CGO.CLKLOCK signal is driven by the main clock CLKDLL's LOCKED output. The SDRAM CLKDLL must be present if the core's CLK2X output shall be driven.

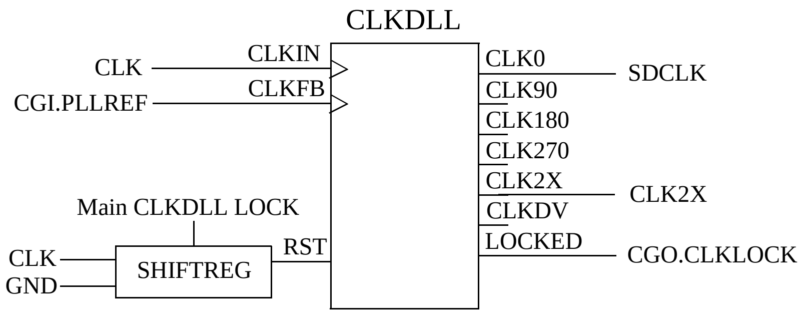


Figure 61. Virtex generation of SDRAM clock with feedback clock enabled

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 62 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

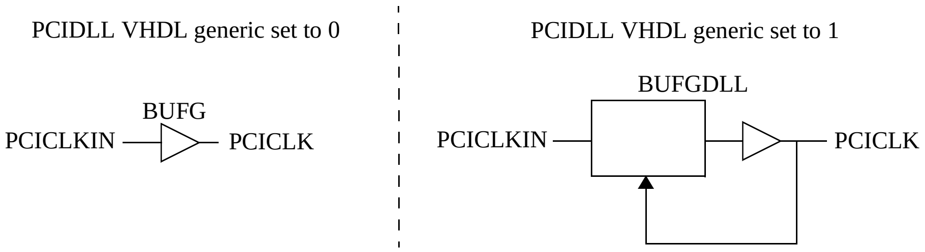


Figure 62. Virtex PCI clock generation

22.2.12 Xilinx Virtex 2/4

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x

The main clock is generated with a DCM which is instantiated with the attributes listed in table 165. The input clock source connected to the CLKIN input is either the core’s CLKIN input or the PCI-CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM’s connections is shown in figure 63.

*Table 165.*Virtex 2/4 DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_mul*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_div*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"1X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

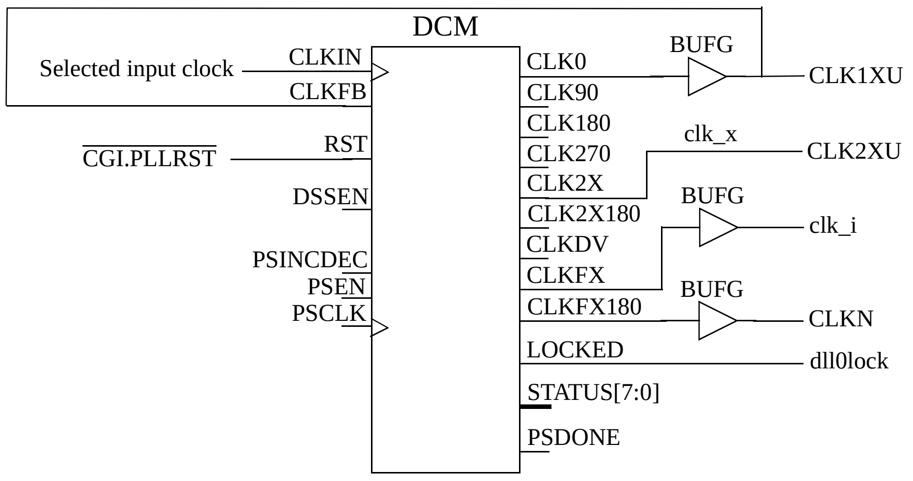


Figure 63. Virtex 2/4 generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 64 is instantiated. The attributes of this DCM are the same as in table 165, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2. The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 65 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

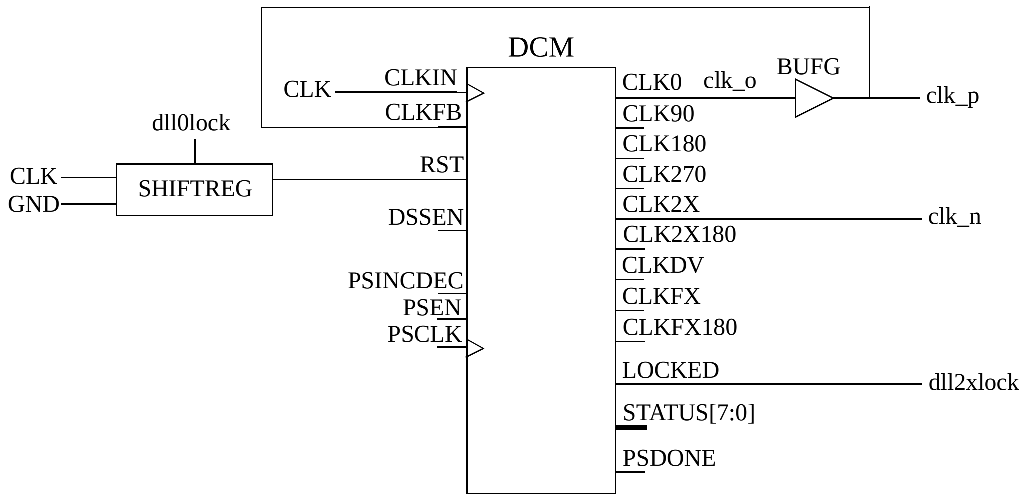


Figure 64. Virtex 2/4 generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. Note that the CLKN output always originates from the main clock DCM, as shown in figure 63.

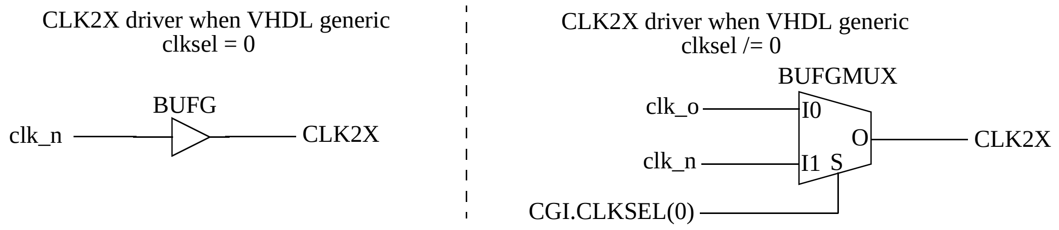

CLK2X driver when VHDL generic clksel = 0
BUFG
clk_n
CLK2X
CLK2X driver when VHDL generic clksel /= 0
BUFGMUX
clk_o
I0
O
clk_n
I1 S
CLK2X
CGI.CLKSEL(0)

Figure 65. Virtex 2/4 selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven by the main clock DCM's CLK2X output via a BUFG, please see figure 66.

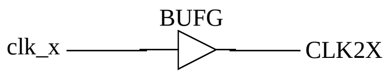


Figure 66. Virtex 2/4 generation of CLK2X clock when VHDL generic *clk2xen* is zero

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 67. The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 64. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 65.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

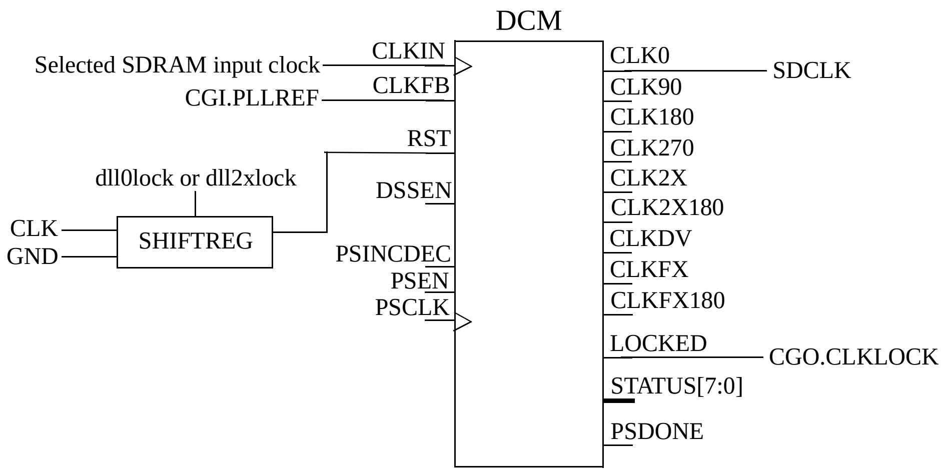


Figure 67. Virtex 2/4 generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core’s SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the clk2xen VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 65, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core’s CLK output.

When the *sdramen* VHDL generic is set to 0 the core’s CGO.CLKLOCK output is connected to the CLK2X DCM’s LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM’s LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 68 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

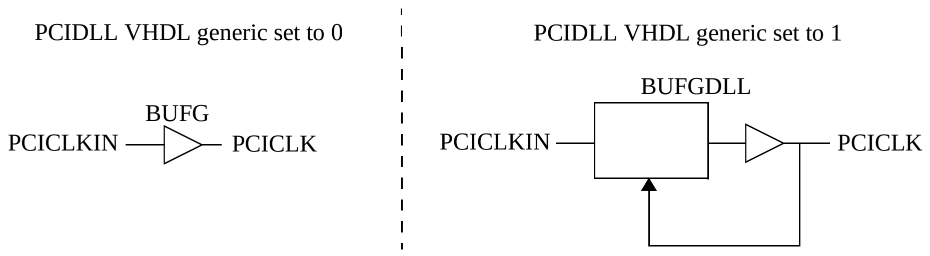


Figure 68. Virtex 2/4 PCI clock generation

22.2.13 Xilinx Virtex 5

Generics used in this technology:	clk_mul, clk_div, sdramen, noclkfb, pcien, pcidll, pcisysclk, freq, clk2xen, clksel
Instantiated technology primitives:	BUFG, BUFMUX, DCM, BUFGDLL
Signals not driven in this technology:	clk4x

The main clock is generated with a DCM which is instantiated with the attributes listed in table 166. The input clock source connected to the CLKIN input is either the core’s CLKIN input or the PCI-

CLKIN input. This is selected with the VHDL generics *pcien* and *pcisysclk*. The main DCM's connections is shown in figure 69.

*Table 166.*Virtex 5 DCM attributes

Attribute name*	**Value**
CLKDV_DIVIDE	2.0
CLKFX_DIVIDE	Determined by core's VHDL generic *clk_mul*
CLKFX_MULTIPLY	Determined by core's VHDL generic *clk_div*
CLKIN_DIVIDE_BY_2	false
CLKIN_PERIOD	10.0
CLKOUT_PHASE_SHIFT	"NONE"
CLK_FEEDBACK	"1X"
DESKEW_ADJUST	"SYSTEM_SYNCHRONOUS"
DFS_FREQUENCY_MODE	"LOW"
DLL_FREQUENCY_MODE	"LOW"
DSS_MODE	"NONE"
DUTY_CYCLE_CORRECTION	true
FACTORY_JF	X"C080"
PHASE_SHIFT	0
STARTUP_WAIT	false

*Any attributes not listed are assumed to have their default value

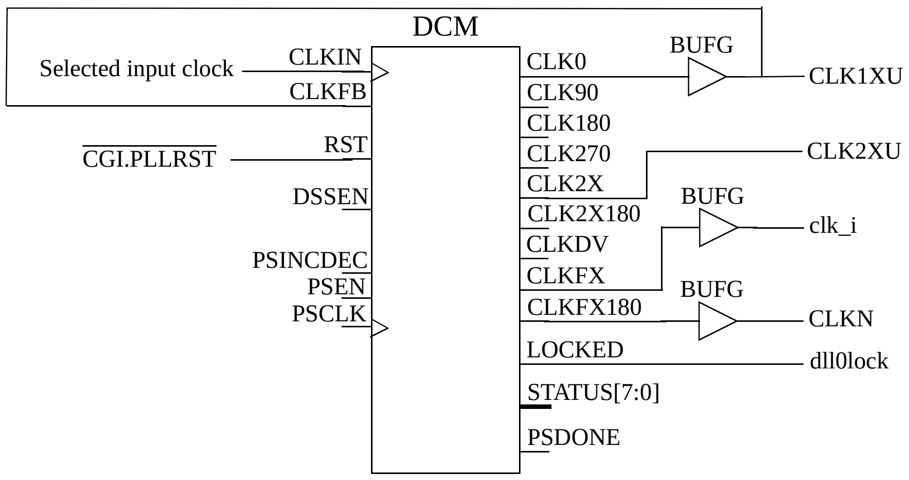


Figure 69. Virtex 5 generation of main clock

If the VHDL generic *clk2xen* is non-zero the DCM shown in figure 70 is instantiated. The attributes of this DCM are the same as in table 166, except that the CLKFX_MULTIPLY and CLKFX_DIVIDE attributes are both set to 2. The dll0lock signal is connected to the LOCKED output of the main clock DCM. When this signal is low all the bits in the shift register connected to the CLK2X DCM's RST input are set to '1'. When the dll0lock signal is asserted it will take four main clock cycles until the RST input is deasserted. Depending on the value of the *clksel* VHDL generic the core's CLK2X output is either driven by a BUFG or a BUFGMUX. Figure 71 shows the two alternatives and how the CGI.CLKSEL(0) input is used to selected between the CLK0 and CLK2X output of the CLK2X DCM.

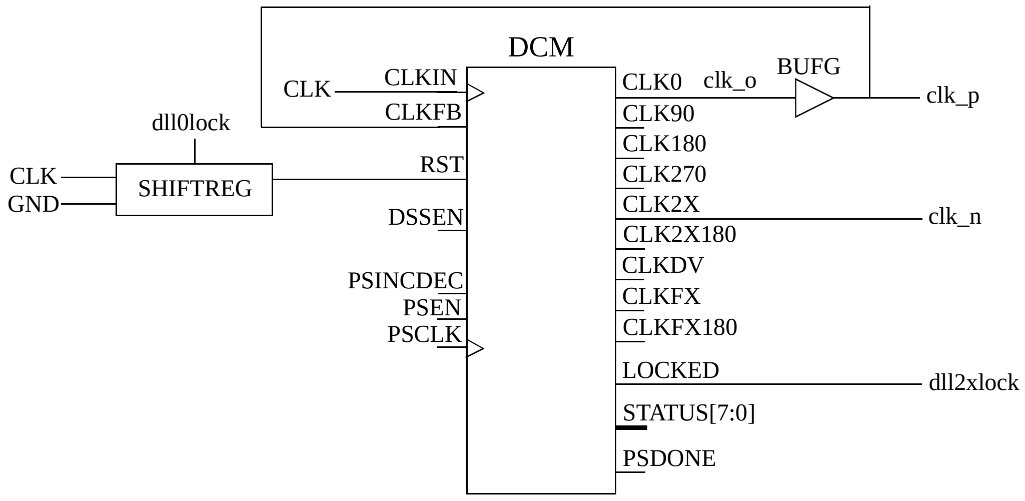


Figure 70. Virtex 5 generation of CLK2X clock when VHDL generic *clk2xen* is non-zero

The value of the *clk2xen* VHDL generic also decides which output that drives the core's CLK output. If the VHDL generic is non-zero the CLK output is driven by the clk_p signal originating from the CLK2X DCM. Otherwise the CLK output is connected to the clk_i signal originating from the main clock DCM. Note that the CLKN output always originates from the main clock DCM, as shown in figure 69.

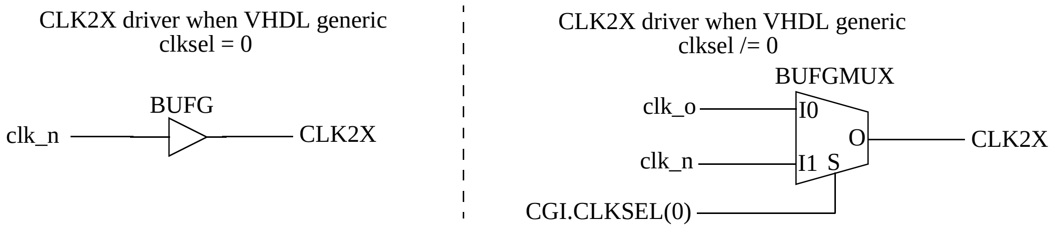

CLK2X driver when VHDL generic clksel = 0
BUFG
clk_n
CLK2X
CLK2X driver when VHDL generic clksel /= 0
BUFGMUX
clk_o
I0
O
CLK2X
clk_n
I1
S
CGI.CLKSEL(0)

Figure 71. Virtex 5 selection of CLK2X clock when VHDL generic *clk2xen* is non-zero

If the VHDL generic *clk2xen* is zero the dll0lock signal from the main clock DCM is either connected to the SDRAM DCM, described below, or if the SDRAM DCM is non-existent, to the core's CGO.CLKLOCK output. This setting also leads to the core's CLK2X output being driven directly by the main clock DCM's CLK2X output.

If the SDRAM clock is enabled, via the *sdramen* VHDL generic, and the clock generator is configured to use clock feedback the DCM shown in figure 72. This DCM has the same attributes as the main clock DCM described in table 166, with the exceptions that CLKFX_MULTIPLY and CLKFX_DIVIDE are both set to 2 and DESKEW_ADJUST is set to "SOURCE_SYNCHRONOUS".

The input to the SDRAM DCM input clock is determined via the *clk2xen* VHDL generic. If the VHDL generic is set to 0 the input is the main CLK, if the generic is set to 1 the input is the clk_p out of the CLK2X DCM shown in figure 64. If the *clk2xen* VHDL generic is set to 2 the clock input to the SDRAM DCM depends on the *clksel* VHDL generic. The input in this last case is the CLK2X output shown in figure 71.

If the CLK2X DCM has been instantiated the SDRAM DCM RST input depends on the LOCKED output of the CLK2X DCM. If the CLK2X DCM has not been instantiated the SDRAM DCM RST input depends on the LOCKED output from the main clock DCM. The applicable LOCKED signal is utilized to keep the SDRAM DCM in reset until its input clock has been stabilized. This is done with a shift register with the same method used for the CLK2X DCM RST.

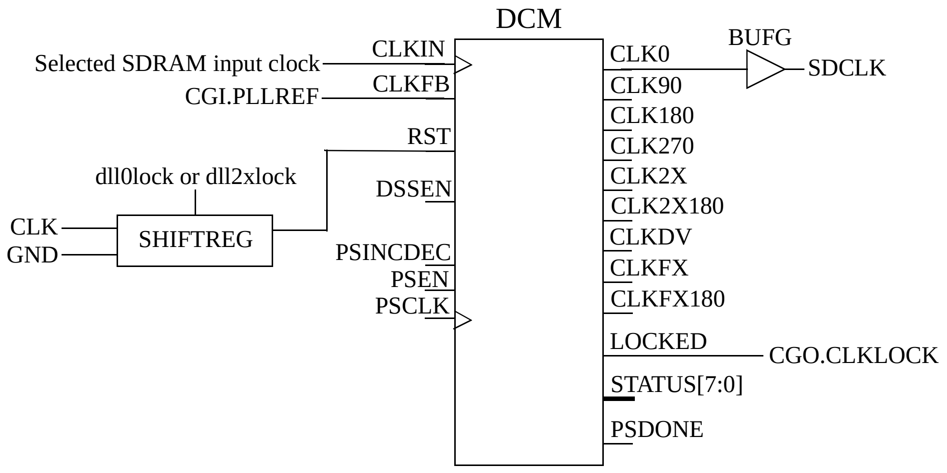


Figure 72. Virtex 5 generation of SDRAM clock

If the SDRAM clock is disabled (*sdramen* VHDL generic set to 0) or the core has been configured not to use clock feedback (*noclockfb* VHDL generic set to 1) the driver of the core's SDCLK output is determined by the value of the *clk2xen* VHDL generic. If the clk2xen VHDL generic is set to 2, the SDRAM clock output is the same as the CLK2X output shown in figure 71, in other words it also depends on the *clksel* VHDL generic. If the *clk2xen* VHDL generic has any other value the SDCLK output is the same as the core's CLK output.

When the *sdramen* VHDL generic is set to 0 the core's CGO.CLKLOCK output is connected to the CLK2X DCM's LOCKED output, if the DCM exists, otherwise the CGO.CLKLOCK output is connected to the main clock DCM's LOCKED output.

If PCI clock generation is enabled via the *pcien* VHDL generic the core instantiates either a BUFG or a BUFGDLL as depicted in figure 73 below. Note that the PCI clock must be enabled if the main clock is to be driven by the PCICLKIN input. If the PCI clock is disabled the PCICLK output is driven to zero. The CGO.PCILOCK output is always driven high in all configurations.

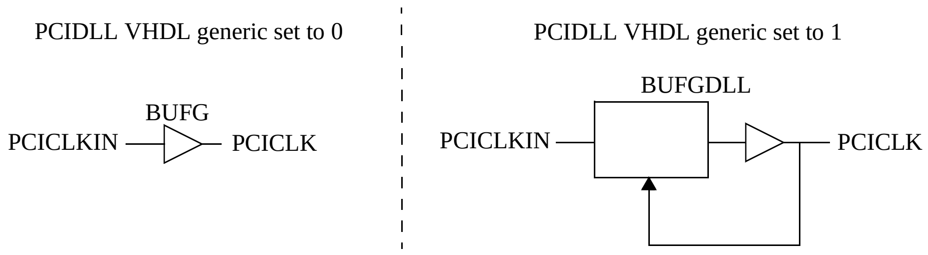


Figure 73. Virtex 5 PCI clock generation

22.3 Configuration options

Table 167 shows the configuration options of the core (VHDL generics).

*Table 167.*Configuration options

Generic name	Function	Allowed range	Default
tech	Target technology	0 - NTECH	inferred
clk_mul	Clock multiplier, used in clock scaling. Not all techbologies support clock scaling.		1
clk_div	Clock divisor, used in clock scaling. Not all technologies support clock scaling.		1
sdramen	When this generic is set to 1 the core will generate a clock on the SDCLK. Not supported by all technologies. See technology specific description.		0
noclkfb	When this generic is set to 0 the core will use the CGI.PLLREF input as feedback clock for some technologies. See technology specific description.		1
pcien	When this generic is set to 1 the PCI clock is activated. Otherwise the PCICLKIN input is typically unused. See technology specific descriptions.		0
pcidll	When this generic is set to 1, a DLL will be instantiated for the PCI input clock for some technologies. See the technology specific descriptions.		0
pcisysclk	When this generic is set to 1 the clock generator will use the pciclkin input as the main clock reference. This also requires generic pcien to be set to 1.		0
freq	Clock frequency in kHz		25000
clk2xen	Enables 2x clock output. Not available in all technolgies and may have additional options. See technology specific description.		0
clksel	Enable clock select. Not available in all technologies.		0
clk_odiv	ProASIC3 output divider. Only used in ProASIC3 technology.		0

22.4 Signal descriptions

Table 168 shows the interface signals of the core (VHDL ports).

*Table 168.*Signal descriptions

Signal name	Field	Type	Function	Active
CLKIN	N/A	Input	Reference clock input	-
PCICLKIN	N/A	Input	PCI clock input	
CLK	N/A	Output	Main clock	-
CLKN	N/A	Output	Inverted main clock	-
CLK2X	N/A	Output	2x clock	-
SDCLK	N/A	Output	SDRAM clock	-
PCICLK	N/A	Output	PCI clock	-
CGI	PLLREF	Input	Optional reference for PLL	-
	PLLRST	Input	Optional reset for PLL	
	PLLCTRL	Input	Optional control for PLL	
	CLKSEL	Input	Optional clock select	
CGO	CLKLOCK	Output	Lock signal for main clock	
	PCILOCK	Output	Lock signal for PCI clock	
CLK4X	N/A	Output	4x clock	
CLK1XU	N/A	Output	Unscaled 1x clock	
CLK2XU	N/A	Output	Unscaled 2x clock	

22.5 Library dependencies

Table 169 shows the libraries used when instantiating the core (VHDL libraries).

*Table 169.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Component, signals	Core signal definitions
TECHMAP	ALLCLKGEN	Component	Technology specific CLKGEN components

22.6 Instantiation

This example shows how the core can be instantiated together with the GRLIB reset generator.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity clkgen_ex is
  port (
    resetn  : in  std_ulogic;
    clk  : in  std_ulogic; -- 50 MHz main clock
    pllref   : in  std_ulogic
  );
end;

architecture example of clkgen_ex is

signal lclk, clkm, rstn, rstraw, sdclkl, clk50: std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;
```

```
begin
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw;

  pllref_pad : clkpad generic map (tech => padtech) port map (pllref, cgi.pllref);

  clk_pad : clkpad generic map (tech => padtech) port map (clk, lclk);

  clkgen0 : clkgen  -- clock generator
    generic map (clktech, CFG_CLKMUL, CFG_CLKDIV, CFG_MCTRL_SDEN,
                 CFG_CLK_NOFB, 0, 0, 0, BOARD_FREQ)
    port map (lclk, lclk, clkm, open, open, sdclkl, open, cgi, cgo, open, clk50);

  sdclk_pad : outpad generic map (tech => padtech, slew => 1, strength => 24)
    port map (sdclk, sdclkl);

  resetn_pad : inpad generic map (tech => padtech) port map (resetn, rst);

  rst0 : rstgen  -- reset generator
    port map (rst, clkm, cgo.clklock, rstn, rstraw);

end;
```

23 CMP7GRLIB - Actel CoreMP7/GRLIB bridge

23.1 Overview

The core instantiates an Actel CoreMP7 processor together with Actel’s CoreMP7Bridge and the GRLIB Core MP7 wrapper CMP7WRAP.

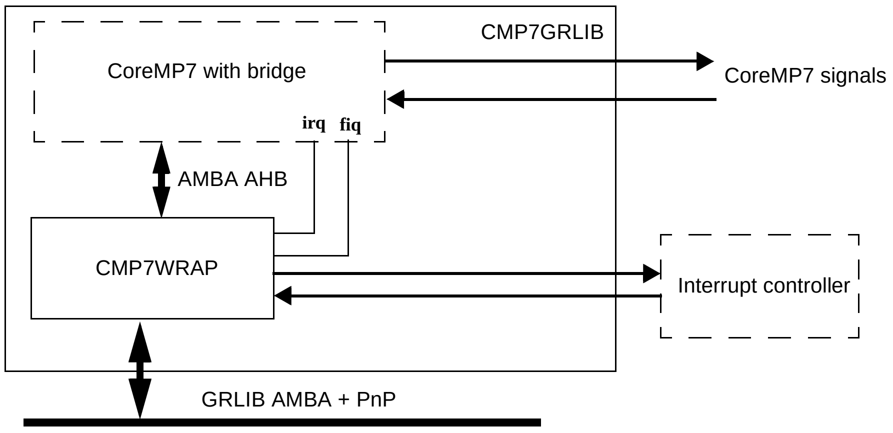


Figure 74. Block diagram

23.2 Operation

23.2.1 Overview

The core instantiates an Actel CoreMP7 processor together with a CoreMP7Bridge. The CoreMP7Bridge is connected to the CoreMP7/GRLIB wrapper (CMP7WRAP) from Gaisler Research. The CoreMP7Bridge and the CoreMP7/GRLIB wrapper allows the CoreMP7 processor to be used with any GRLIB cores.

For detailed information on VHDL generics and signal descriptions please see the documentation for the instantiated cores.

23.2.2 Dependencies

The core instantiates components that are available in the CoreConsole IP library from Actel. Development has been done with CoreConsole version 1.4. GRLIB includes functionality for copying the required files from a CoreConsole repository into the GRLIB tree. Please see the CoreMP7 template design documentation for further information.

23.3 Registers

The core instantiates the CoreMP7 GRLIB wrapper (CMP7WRAP) which has registers that can be accessed via AMBA APB.

23.4 Vendor and device identifier

See documentation for the CoreMP7 GRLIB wrapper (CMP7WRAP).

23.5 Implementation

23.5.1 Technology mapping

The core instantiates an Actel CoreMP7 processor block as a black box.

23.5.2 RAM usage

The core does not use any RAM components.

23.6 Configuration options

Table 170 shows the configuration options of the core (VHDL generics).

*Table 170.*Configuration options

Generic name	Function	Allowed range	Default
debug	CoreMP7Bridge generic, debug options	N/A	2
syncfiq	CoreMP7Bridge generic, synchronize nFIQ	N/A	0
syncirq	CoreMP7Bridge generic, synchronize nIRQ	N/A	0
hindex	CMP7WRAP GRLIB wrapper generic	N/A	0
pindex	CMP7WRAP GRLIB wrapper generic	N/A	0
pmask	CMP7WRAP GRLIB wrapper generic	N/A	16#FFF#

23.7 Signal descriptions

Table 171 shows the interface signals of the core (VHDL ports).

*Table 171.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RST	Input	Unused	-
	RUN	Input	Unused	-
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
	PWD	Output	Unused	-
WDOGRES	N/A	Input	Watchdog signal to MP7Bridge	-
WDOGRESN	N/A	Output	Watchdog signal from MP7Bridge	-
ICE_NTRST	N/A	Input	RealView ICE JTAG Reset	-
ICE_TCK	N/A	Input	RealView ICE JTAG Clock Enable	-
ICE_TDI	N/A	Input	RealView ICE JTAG Data In	-
ICE_TMS	N/A	Inpuit	RealView ICE JTAG Mode Select	-
ICE_VTREF	N/A	Output	RealView ICE Target Reference Voltage	-
ICE_TDO	N/A	Output	RealView ICE JTAG Data Out	-
ICE_RTCK	N/A	Output	RealView ICE JTAG RTSCK (Used for adaptive clocking)	-
ICE_NSRST	N/A	Input/ Output	RealView ICE JTAG System Reset	-
ICE_DBGACK	N/A	Output	RealView ICE JTAG Debug Acknowledge	-
ICE_DBGRQ	N/A	Output	RealView ICE JTAG Debug Request	-

*Table 171.*Signal descriptions

Signal name	Field	Type	Function	Active
ICE_TDOUT	N/A	Output	RealView ICE JTAG	-
ICE_NTDOEN	N/A	Output	RealView ICE JTAG	-
UJTAG_TCK	N/A	Input	JTAG interface to MP7Bridge	-
UJTAG_TDI	N/A	Input	JTAG interface to MP7Bridge	-
UJTAG_TMS	N/A	Input	JTAG interface to MP7Bridge	-
UJTAG_TRSTB	N/A	Input	JTAG interface to MP7Bridge	-
UJTAG_TDO	N/A	Output	JTAG interface to MP7Bridge	-
CPA	N/A	Input	Co-processor interface signal	-
CPB	N/A	Input	Co-processor interface signal	-
CPSEQ	N/A	Output	Co-processor interface signal	-
CPTBIT	N/A	Output	Co-processor interface signal	-
CPNI	N/A	Output	Co-processor interface signal	-
CPNMREQ	N/A	Output	Co-processor interface signal	-
CPNOPC	N/A	Output	Co-processor interface signal	-
CPNTRANS	N/A	Output	Co-processor interface signal	-
DBGBREAK	N/A	Input	ETM interface signal	-
DBGEXT[1:0]	N/A	Input	ETM interface signal	-
DBGRQ	N/A	Input	ETM interface signal	-
DBGACK	N/A	Output	ETM interface signal	-
DBGCOMMRX	N/A	Output	ETM interface signal	-
DBGCOMMTX	N/A	Output	ETM interface signal	-
DBGINSTR-VALID	N/A	Output	ETM interface signal	-
DBGRNG[1:0]	N/A	Output	ETM interface signal	-
DBGNEXEC	N/A	Output	ETM interface signal	-
CFGBIGEND	N/A	Input	Controls the core’s endianess setting	High
DMORE	N/A	Output	Asserted during LDM and STM instructions.	High
DBGEN	N/A	Input	Should always be asserted.	High

* see GRLIB IP Library User’s Manual

23.8 Library dependencies

Table 172 shows the libraries used when instantiating the core (VHDL libraries).

*Table 172.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	LEON3	Signals	Signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

23.9 Component declaration

The core’s component declaration is shown below.

```
component cmp7grlib
  generic(
    DEBUG   : integer := 2;
    SYNCFIQ : integer := 0;
```

```
      SYNCIRQ : integer := 0;
      hindex  : integer := 0;
      pindex  : integer := 0;
      paddr   : integer := 16#000#;
      pmask   : integer := 16#fff#
      );
    port (
      rst             : in  std_ulogic;
      clk             : in  std_ulogic;
      ahbmi           : in  ahb_mst_in_type;
      ahbmo           : out ahb_mst_out_type;
      apbi            : in apb_slv_in_type;
      apbo            : out apb_slv_out_type;
      irqi            : in  l3_irq_in_type;
      irqo            : out l3_irq_out_type;
      WDOGRES         : in std_logic;
      WDOGRESn        : out std_logic;
      ICE_nTRST       : in  std_logic;
      ICE_TCK         : in  std_logic;
      ICE_TDI         : in  std_logic;
      ICE_TMS         : in  std_logic;
      ICE_VTref       : out std_logic;
      ICE_TDO         : out std_logic;
      ICE_RTCK        : out std_logic;
      ICE_nSRST       : inout std_logic;
      ICE_DBGACK      : out std_logic;
      ICE_DBGRQ       : out std_logic;
      ICE_TDOUT       : out std_logic;
      ICE_nTDOEN      : out std_logic;
      UJTAG_TCK       : in std_logic;
      UJTAG_TDI       : in std_logic;
      UJTAG_TMS       : in std_logic;
      UJTAG_TRSTB     : in std_logic;
      UJTAG_TDO       : out std_logic;
      CPA             : in std_logic;
      CPB             : in std_logic;
      CPSEQ           : out std_logic;
      CPTBIT          : out std_logic;
      CPnI            : out std_logic;
      CPnMREQ         : out std_logic;
      CPnOPC          : out std_logic;
      CPnTRANS        : out std_logic;
      DBGBREAK        : in std_logic;
      DBGEXT          : in std_logic_vector(1 downto 0);
      DBGRQ           : in std_logic;
      DBGACK          : out std_logic;
      DBGCOMMRX       : out std_logic;
      DBGCOMMTX       : out std_logic;
      DBGINSTRVALID   : out std_logic;
      DBGRNG          : out std_logic_vector(1 downto 0);
      DBGnEXEC        : out std_logic;)
      CFGBIGEND       : in std_logic;
      DMORE           : out std_logic;
      DBGEN           : in std_logic
      );
  end component;
```

24 CMP7WRAP - Actel CoreMP7 GRLIB wrapper

24.1 Overview

The core allows using Actel's CoreMP7 processor within GRLIB. The CoreMP7, paired with the CoreMP7Bridge, already provides an AMBA interface and the core provides GRLIB plug'n'play information, handles interrupt steering and performs endianness adaption.

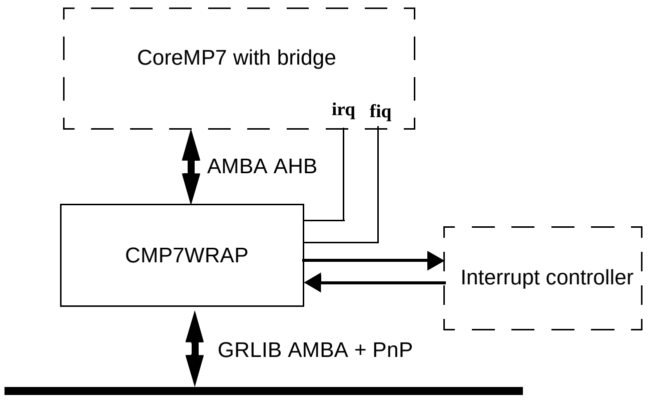


Figure 75. Block diagram

24.2 Operation

24.2.1 Overview

The core is typically instantiated via the CMP7GRLIB CoreMP7/GRLIB bridge.

24.2.2 Bus interface

Since the CoreMP7 paired with the CoreMP7Bridge (supplied by Actel) already has an AMBA interface, the core does not provide any rapping of the actual bus interface. The core provides the GRLIB plug'n'play side band signals. On the CoreMP7 side the core has inputs and outputs for all AMBA related signals going to and from the CoreMP7Bridge. On the GRLIB side the wrapper has the same inputs and outputs with the addition of the plug'n'play signals.

24.2.3 Interrupt wrapping

The CoreMP7 and the Leon3 handle interrupts differently. To make the CoreMP7 compatible with the GRLIB interrupt controller (IRQMP), the core provides an interface between the two. When an interrupt occurs the wrapper will assert either the IRQ or FIQ signal connected to the CoreMP7Bridge. Which signal that is asserted is determined by a configurable mask register in the core. Using this register the designer can decide which of the 15 available interrupts that should be forwarded as regular interrupts and which that should be forwarded as fast interrupts.

When an interrupt has been handled by the processor, software must write to an acknowledge register in the core. When this is done the wrapper asserts the acknowledge signal to the interrupt controller to signal that the interrupt has been handled. The core also automatically provides the interrupt level of the handled interrupt. This is required by IRQMP to clear the correct interrupt. The core's acknowledge register automatically resets itself after one cycle.

The interrupt level register that is used when an interrupt is acknowledged can also be accessed by the processor. This feature can be used to establish the source of an interrupt.

The block diagram in figure 75 provides a simplified view of the core in a system, leaving out the APB connection used for accessing the core's internal registers.

24.2.4 Endianness

The CoreMP7 is a bi-endian processor (little- or big-endian), while the LEON3 and GRLIB is big-endian. This poses a problem when operating in little endian mode and making byte and half-word accesses. While big-endian mode works correctly without any changes, Actel’s software tools are intended for use with the processor in little-endian mode. The core provides means of making the CoreMP7 compatible with GRLIB in this mode as well.

The core has an input named BIGEND which should be connected to the same source as the CFG-BIGEND input signal on the CoreMP7. Any changes made to this signal will then be propagated both to the processor and the core.

24.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 173.*GRSLINK registers

APB address offset	Register
0x00	Fast interrupt request mask register
0x04	Interrupt level register
0x08	Interrupt acknowledge register

Table 174. CMP7WRAP Fast interrupt request mask register

31 16	15 1	0
RESERVED	FIQ_MASK	R

31 :16	RESERVED
15:1	Read instruction (FIQ_MASK) - Fast interrupt request mask, read/write.
0	RESERVED

Table 175. CMP7WRAP Interrupt level register

31 4	3 0
RESERVED	IRL

31 :4	RESERVED
3:0	Interrupt Level (IRL) - Interrupt level, read only.

Table 176. CM7WRAP Interrupt acknowledge register

31 1	0
RESERVED	ACK

31:1	RESERVED
0	Acknowledge (ACK) - Interrupt acknowledge, write only.

24.4 Vendor and device identifier

The core's AHB master interface has vendor identifier 0xAC (Actel Corporation) and device identifier 0x001. The core's APB slave interface has vendor identifier 0x01 (Gaisler Research) and device identifier 0x065. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

24.5 Implementation

24.5.1 Technology mapping

The core does not instantiate any technology specific primitives.

24.5.2 RAM usage

The core does not use any RAM components.

24.6 Configuration options

Table 177 shows the configuration options of the core (VHDL generics).

*Table 177.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - (NAHBMST-1)	0
pindex	APB slave index	0 - (NAPBMAX-1)	0
paddr	12-bit MSB APB address	0 - 16#FFF#	16#000#
pmask	APB address address mask.	0 - 16#FFF#	16#FFF#

24.7 Signal descriptions

Table 178 shows the interface signals of the core (VHDL ports).

*Table 178.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RST	Input	Unused	-
	RUN	Input	Unused	-
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
	PWD	Output	Unused	-
BIGEND	N/A	Input	Controls the core's endianess setting	-
NIRQ	N/A	Output	CoreMP7Bridge interrupt input	Low
NFIQ	N/A	Output	CoreMP7Bridge fast interrupt input	Low
HGRANT	N/A	Output	CoreMP7Bridge AHB HGRANT input	High
HRDATA[31:0]	N/A	Output	CoreMP7Bridge AHB HRDATA input	-

*Table 178.*Signal descriptions

Signal name	Field	Type	Function	Active
HREADY	N/A	Output	CoreMP7Bridge AHB HREADY input	High
HRESP[1:0]	N/A	Output	CoreMP7Bridge AHB HRESP input	-
HADDR[31:0]	N/A	Input	CoreMP7Bridge AHB HADDR output	-
HBURST	N/A	Input	CoreMP7Bridge AHB HBURST output	-
HBUSREQ	N/A	Input	CoreMP7Bridge AHB HBUSREQ output	-
HLOCK	N/A	Input	CoreMP7Bridge AHB HLOCK output	-
HPROT	N/A	Input	CoreMP7Bridge AHB HPROT output	-
HRESETN	N/A	Input	CoreMP7Bridge AHB HRESETn output	Low
HSIZE	N/A	Input	CoreMP7Bridge AHB HSIZE output	-
HTRANS	N/A	Input	CoreMP7Bridge AHB HTRANS output	-
HWDATA[31:0]	N/A	Input	CoreMP7Bridge AHB HWDATA output	-
HWRITE	N/A	Input	CoreMP7Bridge AHBHWRITE output	High

* see GRLIB IP Library User's Manual

24.8 Library dependencies

Table 179 shows the libraries used when instantiating the core (VHDL libraries).

*Table 179.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	LEON3	Signals	Signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

24.9 Component declaration

The core's component declaration is shown below.

```
component cmp7wrap
    generic (
      hindex  : integer := 0;
      pindex  : integer := 0;
      paddr   : integer := 16#000#;
      pmask   : integer := 16#fff#
      );
    port (
      rst     : in  std_ulogic;
      clk     : in  std_ulogic;
      ahbmi   : in  ahb_mst_in_type;
      ahbmo   : out ahb_mst_out_type;
      apbi    : in  apb_slv_in_type;
      apbo    : out apb_slv_out_type;
      irqi    : in  l3_irq_in_type;
      irqo    : out l3_irq_out_type;
      nirq    : out std_ulogic;
      nfiq    : out std_ulogic;
      bigend  : in  std_ulogic;
      -- From wrapper to MP7Bridge
      HGRANT  : out std_logic;
      HRDATA  : out std_logic_vector(31 downto 0);
      HREADY  : out std_logic;
      HRESP   : out std_logic_vector(1 downto 0);
      -- To wrapper from MP7Bridge
      HADDR   : in std_logic_vector(31 downto 0);
      HBURST  : in std_logic_vector(2 downto 0);
      HBUSREQ : in std_logic;
      HLOCK   : in std_logic;
      HPROT   : in std_logic_vector(3 downto 0);
      HRESETn : in std_logic;
      HSIZE   : in std_logic_vector(2 downto 0);
      HTRANS  : in std_logic_vector(1 downto 0);
      HWDATA  : in std_logic_vector(31 downto 0);
      HWRITE  : in std_logic
      );
  end component;
```

25 DDRSPA - 16-, 32- and 64-bit DDR266 Controller

25.1 Overview

DDRSPA is a DDR266 SDRAM controller with AMBA AHB back-end. The controller can interface two 16-, 32- or 64-bit DDR266 memory banks to a 32-bit AHB bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for DDR SDRAM access. The DDR controller is programmed by writing to a configuration register mapped located in AHB I/O address space. Internally, DDRSPA consists of a ABH/DDR controller and a technology specific DDR PHY. Currently supported technologies for the PHY is Xilinx Virtex2/Virtex4 and Altera Startix-II. The modular design of DDRSPA allows to add support for other target technologies in a simple manner. The DDRSPA is used in the following GRLIB template designs: *leon3-avnet-ml401*, *leon3-avnet-eval-xc4v*, *leon3-digilent-xup*, *leon3-digilent-xc3s1600e*, *leon3-altera-ep2s60-ddr* and *leon3-altera-ep3c25*.

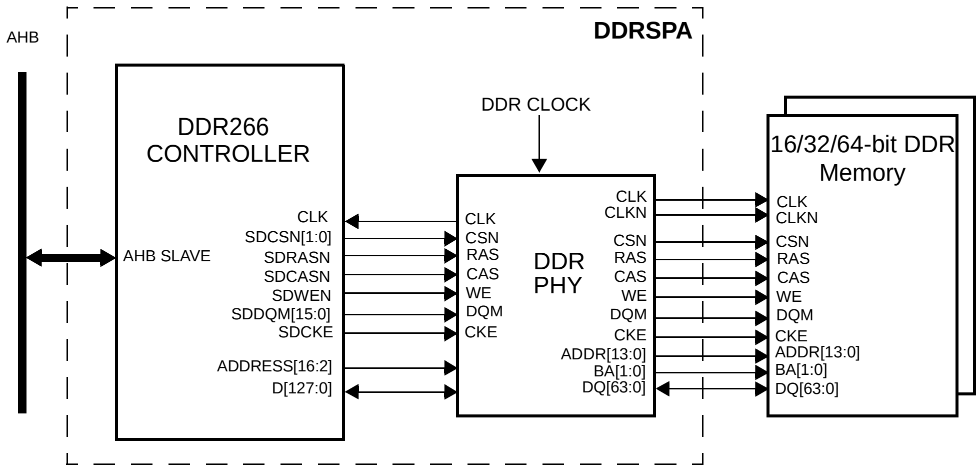


Figure 76. DDRSPA Memory controller conected to AMBA bus and DDR SDRAM

25.2 Operation

25.2.1 General

Double data-rate SDRAM (DDR RAM) access is supported to two banks of 16-, 32- or 64-bit DDR266 compatible memory devices. The controller supports 64M, 128M, 256M, 512M and 1G devices with 9- 12 column-address bits, up to 14 row-address bits, and 4 internal banks. The size of each of each chip select can be programmed in binary steps between 8 Mbyte and 1024 Mbyte. The DDR data width is set by the **ddrbits** VHDL generic, and will affect the width of DM, DQS and DQ signals. The DDR data width does not change the behavior of the AHB interface, except for data latency. When the VHDL generic **mobile** is set to a value not equal to 0, the controller supports mobile DDR SDRAM (LPDDR).

25.2.2 Read cycles

An AHB read access to the controller will cause a corresponding access to the external DDR RAM. The read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command. CAS latency of 2 (CL=2) or 3 (CL=3) can be used. Byte, half-word (16-bit) and word (32-bit) AHB accesses are supported. Incremental AHB burst access are supported for 32-bit words only. The read cycle(s) are always terminated with a PRE-CHARGE command, no banks are left open between two accesses. DDR read cycles are always performed in (aligned) 8-word bursts, which are stored in a FIFO. After an initial latency, the data is then read out on the AHB bus with zero waitstates.

25.2.3 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. An AHB write burst will store up to 8 words in a FIFO, before writing the data to the DDR memory. As in the read case, only word bursts are supported

25.2.4 Initialization

If the `pwron` VHDL generic is 1, then the DDR controller will automatically perform the DDR initialization sequence as described in the JEDEC DDR266 standard: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG, PRE-CHARGE, 2xREFRESH and LOAD-MODE-REG; or as described in the JEDEC LPDDR standard when mobile DDR is enabled: PRE-CHARGE, 2xREFRESH, LOAD-MODE-REG and LOAD-EXTMODE-REG. The VHDL generics **col** and **Mbyte** can be used to also set the correct address decoding after reset. In this case, no further software initialization is needed. The DDR initialization can be performed at a later stage by setting bit 15 in the DDR control register.

25.2.5 Configurable DDR SDRAM timing parameters

To provide optimum access cycles for different DDR devices (and at different frequencies), three timing parameters can be programmed through the memory configuration register (SDCFG): TRCD, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 180.

*Table 180.*DDR SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to read/write (t_{RCD})	TRCD + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCD, TRP and TRFC are programmed such that the DDR200/266 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 181.*DDR SDRAM example programming

DDR SDRAM settings	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz: CL=2, TRP=0, TRFC=4, TRCD=0	20	80	20	70	50
133 MHz: CL=2, TRP=1, TRFC=6, TRCD=1	20	82	22	67	52

When the DDRSPA controller uses CAS latency (CL) of two cycles a DDR SDRAM speed grade of -75Z or better is needed to meet 133 MHz timing.

When mobile DDR support is enabled, two additional timing parameters can be programmed though the Power-Saving configuration register.

*Table 182.*Mobile DDR SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Exit Power-down mode to first valid command (t_{XP})	tXP + 1
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

25.2.6 Refresh

The DDRSPA 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 SDCFG register. Depending on SDRAM type, the required period is typically 7.8 us (corresponding to 780 at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by bit 31 in SDCTRL register.

25.2.7 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile DDR functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode and mobile DDR is disabled, the controller introduce a delay of 200 clock cycles and a AUTO REFRESH command before any other memory access is allowed. When mobile DDR is enabled the delay before the AUTO REFRESH command is defined by tXSR in the Power-Saving configuration register. The minimum duration of this mode is defined by tRFC. This mode is only available then the VHDL generic **mobile** >= 1.

25.2.8 Clock Stop

In the clock stop mode, the external clock to the SDRAM is stop at a low level (DDR_CLK is low and DDR_CLKB is high). This reduce the power consumption of the SDRAM while retaining the data. To enable the clock stop mode, set the PMODE bits in the Power-Saving configuration register to “100” (Clock Stop). The controller will enter clock stop mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. The REFRESH command will still be issued by the controller in this mode. This mode is only available then the VHDL generic **mobile** >= 1 and mobile DDR functionality is enabled.

25.2.9 Power-Down

When entering the power-down mode all input and output buffers, including DDR_CLK and DDR_CLKB and excluding DDR_CKE, are deactivated. This is a more efficient power saving mode then clock stop mode, with a grater reduction of the SDRAM’s power consumption. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one or two (when tXP in the Power-Saving configuration register is ‘1’) clock cycles are added before issue any command to the memory. This mode is only available then the VHDL generic **mobile** >= 1.

25.2.10 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared followed by the mobile SDRAM initialization sequence. The mobile SDRAM initialization sequence can be performed by setting bit 15 in the DDR control register. This mode is only available then the VHDL generic **mobile** >= 1 and mobile DDR functionality is enabled.

25.2.11 Status Read Register

The status read register (SRR) is used to read the manufacturer ID, revision ID, refresh multiplier, width type, and density of the SDRAM. To Read the SSR a LOAD MODE REGISTER command with BA0 = 1 and BA1 = 0 must be issued followed by a READ command with the address set to 0. This command sequence is executed then the Status Read Register is read. This register is only available then the VHDL generic **mobile** >= 1 and mobile DDR functionality is enabled. Only DDR_CSB[0] is enabled during this operation.

25.2.12 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available then the VHDL generic **mobile** >= 1 and mobile DDR functionality is enabled.

25.2.13 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available then the VHDL generic **mobile** >= 1 and mobile DDR functionality is enabled.

25.2.14 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG and REFRESH. If the LEMR command is issued, the PLL Reset bit as programmed in SDCFG will be used, when mobile DDR support is enabled the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. If the LMR command is issued, the CAS latency as programmed in the Power-Saving configuration register will be used and remaining fields are fixed: 8 word sequential burst. The command field will be cleared after a command has been executed.

25.2.15 Clocking

The DDR controller is designed to operate with two clock domains, one for the DDR memory clock and one for the AHB clock. The two clock domains do not have to be the same or be phase-aligned. The DDR input clock (CLK_DDR) can be multiplied and divided by the DDR PHY to form the final DDR clock frequency. The final DDR clock is driven on one output (CLKDDRO), which should always be connected to the CLKDDRI input. If the AHB clock and DDR clock area generated from the same clock source, a timing-ignore constraint should be placed between the CLK_AHB and CLKDDRI to avoid optimization of false-paths during synthesis and place&route.

The Xilinx version of the PHY generates the internal DDR read clock using an external clock feed-back. The feed-back should have the same delay as DDR signals to and from the DDR memories. The feed-back should be driven by DDR_CLK_FB_OUT, and returned on DDR_CLK_FB. Most Xilinx FPGA boards with DDR provides clock feed-backs of this sort. The supported frequencies for the Xilinx PHY depends on the clock-to-output delay of the DDR output registers, and the internal delay from the DDR input registers to the read data FIFO. Virtex2 and Virtex4 can typically run at 120 MHz, while Spartan3e can run at 100 MHz.

The read data clock in the Xilinx version of the PHY is generated using a DCM to offset internal delay of the DDR clock feed back. If the automatic DCM phase adjustment does not work due to unsuitable pin selection, extra delay can be added through the RSKEW VHDL generic. The VHDL generic can be between -255 and 255, and is passed directly to the PHASE_SHIFT generic of the DCM.

The Altera version of the PHY use the DQS signals and an internal PLL to generate the DDR read clock. No external clock feed-back is needed and the DDR_CLK_FB_OUT/DDR_CLK_FB signals are not used. The supported frequencies for the Altera PHY are 100, 110, 120 and 130 MHz. For Altera CycloneIII, the read data clock is generated by the PLL. The phase shift of the read data clock is set be the VHDL generic RSKEW in ps (e.g. a value of 2500 equals 90’ phase for a 100MHz system).

25.2.16 Pads

The DDRSPA core has technology-specific pads inside the core. The external DDR signals should therefore be connected directly the top-level ports, without any logic in between.

25.2.17 Registers

The DDRSPA core implements two control registers. The registers are mapped into AHB I/O address space defined by the AHB BAR1 of the core.

*Table 183.*DDR controller registers

Address offset - AHB I/O - BAR1	**Register**
0x00	SDRAM control register
0x04	SDRAM configuration register (read-only)
0x08	SDRAM Power-Saving configuration register
0x0C	Reserved
0x10	Status Read Register (Only available when mobile DDR support is enabled)
0x14	PHY configuration register 0 (Only available when VHDL generic confapi = 1, TCI RTL_PHY)
0x18	PHY configuration register 1 (Only available when VHDL generic confapi = 1, TCI TRL_PHY)

Table 184. SDRAM control register (SDCTRL)

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tCD	SDRAM bank size	SRAM col. size	SDRAM commnad	PR	IN	CE	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled. This register bit is read only when Power-Saving mode is other then none.
30	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile DDR support is enabled, this bit also represent the MSB in the tRFC timing.
29: 27	SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile DDR support is enabled, this field is extended with the bit 30.
26	SDRAM tRCD delay. Sets tRCD to 2 + field value clocks.
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 8 Mbyte, “001”= 16 Mbyte, “010”= 32 Mbyte .... “111”= 1024 Mbyte.
22: 21	SDRAM column size. “00”=512, “01”=1024, “10”=2048, “11”=4096
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRECHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	PLL Reset. This bit is used to set the PLL RESET bit during LOAD-CONFIG-REG commands.

Table 184. SDRAM control register (SDCTRL)

16	Initialize (IN). Set to ‘1’ to perform power-on DDR RAM initialisation. Is automatically cleared when initialisation is completed. This register bit is read only when Power-Saving mode is other then none.
15	Clock enable (CE). This value is driven on the CKE inputs of the DDR RAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

Table 185. SDRAM configuration register (SDCFG)

31 20	19 16	15	14 12	11 0
Reserved	CONFAPI	MD	Data width	DDR Clock frequency

31: 20	Reserved
19: 16	Register API configuration. 0 = Standard register API. 1 = TCI TSMC90 PHY register API.
15	Mobile DDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
14: 12	DDR data width: “001” = 16 bits, “010” = 32 bits, “011” = 64 bits (read-only)
11: 0	Frequency of the (external) DDR clock (read-only)

Table 186. SDRAM Power-Saving configuration register

31	30	29 25	24	23 20	19	18 16	15 8	7 5	4 3	2 0
ME	CL	Reserved	tC	tXSR	tXP	PMODE	Reserved	DS	TCSR	PASR

31	Mobile DDR functionality enabled. ‘1’ = Enabled (support for Mobile DDR SDRAM), ‘0’ = disabled (support for standard DDR SDRAM)
30	CAS latency; ‘0’ => CL = 2, ‘1’ => CL = 3
29: 25	Reserved
24	SDRAM tCKE timing, tCKE will be equal to 1 or 2 clocks (0/1). (Read only when Mobile DDR support is disabled).
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile DDR support is disabled).
19	SDRAM tXP timing. tXP will be equal to 2 or 3 system clocks (0/1). (Read only when Mobile DDR support is disabled).
18: 16	Power-Saving mode (Read only when Mobile DDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “100”: Clock-Stop (CKS) “101”: Deep Power-Down (DPD)
15: 8	Reserved
7: 5	Selectable output drive strength (Read only when Mobile DDR support is disabled). “000”: Full “001”: One-half “010”: One-quarter “011”: Three-quarter

Table 186. SDRAM Power-Saving configuration register

4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile DDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile DDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

Table 187. Status Read Register

31 16	15 0
SRR_16	SRR

31: 16	Status Read Register when 16-bit DDR memory is used (read only)
15: 0	Status Read Register when 32/64-bit DDR memory is used (read only)

Table 188. PHY configuration register 0 (TCI RTL_PHY only)

31	30	29	28	27 22	21 16	15 8	7 0
R1	R0	P1	P0	TSTCTRL1	TSTCTRL0	MDAJ_DLL1	MDAJ_DLL0

31	Reset DLL 1 (active high)
30	Reset DLL 1 (active high)
29	Power Down DLL 1 (active high)
28	Power Down DLL 1 (active high)
27: 22	Test control DLL 1 tstclkin(1) is connected to SIGI_1 on DDL 1 when bit 26:25 is NOT equal to “00“. tstclkin(0) is connected to SIGI_0 on DDL 1 when bit 23:22 is NOT equal to “00“.
21: 16	Test control DLL 0
15: 8	Master delay adjustment input DLL 1
7: 0	Master delay adjustment input DLL 0

Table 189. PHY configuration register 1 (TCI RTL_PHY only)

31 24	23 16	15 8	7 0
ADJ_RSYNC	ADJ_90	ADJ_DQS1	ADJ_DQS0

31: 24	Slave delay adjustment input for resync clock (Slave 1 DLL 1)
23: 16	Slave delay adjustment input for 90’ clock (Slave 0 DLL 1)
15: 8	Slave delay adjustment input for DQS 1 (Slave 1 DLL 0)
7: 0	Slave delay adjustment input for DQS 0 (Slave 0 DLL 0)

25.3 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x025. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

25.4 Configuration options

Table 190 shows the configuration options of the core (VHDL generics).

*Table 190.*Configuration options

Generic	Function	Allowed range	Default
fabtech	PHY technology selection	virtex2, virtex4, spartan3e, altera	virtex2
memtech	Technology selection for DDR FIFOs	infered, virtex2, virtex4, spartan3e, altera	infered
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where DDR control register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space	0 - 16#FFF#	16#FFF#
ddrbits	Data bus width of external DDR memory	16, 32, 64	16
MHz	DDR clock input frequency in MHz.	10 - 200	100
clkmul, clkdiv	The DDR input clock is multiplied with the clkmul generic and divided with clkdiv to create the final DDR clock	2 - 32	2
rstdel	Clock reset delay in micro-seconds.	1 - 1023	200
col	Default number of column address bits	9 - 12	9
Mbyte	Default memory chip select bank size in Mbyte	8 - 1024	16
pwron	Enable SDRAM at power-on initialization	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
ahbfreq	Frequency in MHz of the AHB clock domain	1 - 1023	50
rskew	Additional read data clock skew Read data clock phase for Altera CycloneIII	-255 - 255. 0 - 9999	0
mobile	Enable Mobile DDR support 0: Mobile DDR support disabled 1: Mobile DDR support enabled but not default 2: Mobile DDR support enabled by default 3: Mobile DDR support only (no regular DDR support)	0 - 3	0
confapi	Set the PHY configuration register API: 0 = standard register API (conf0 and conf1 disabled). 1 = TCI RTL_PHY register API.		
conf0	Reset value for PHY register 0, conf[31:0]	0 - 16#FFFFFFFF#	0
conf1	Reset value for PHY register1, conf[63:32]	0 - 16#FFFFFFFF#	0
regoutput	Enables registers on signal going from controller to PHY	0 - 1	0

25.5 Signal descriptions

Table 191 shows the interface signals of the core (VHDL ports).

*Table 191.*Signal descriptions

Signal name	**Type**	**Function**	**Active**
RST_DDR	Input	Reset input for DDR clock domain	Low
RST_AHB	Input	Reset input for AHB clock domain	Low
CLK_DDR	Input	DDR input Clock	-
CLK_AHB	Input	AHB clock	-
LOCK	Output	DDR clock generator locked	High
CLKDDRO		Internal DDR clock output after clock multiplication	
CLKDDRI		Clock input for the internal DDR clock domain. Must be connected to CLKDDRO.	
AHBSI	Input	AHB slave input signals	-
AHBSO	Output	AHB slave output signals	-
DDR_CLK[2:0]	Output	DDR memory clocks (positive)	High
DDR_CLKB[2:0]	Output	DDR memory clocks (negative)	Low
DDR_CLK_FB_OUT	Output	Same a DDR_CLK, but used to drive an external clock feedback.	-
DDR_CLK_FB	Input	Clock input for the DDR clock feed-back	-
DDR_CKE[1:0]	Output	DDR memory clock enable	High
DDR_CSB[1:0]	Output	DDR memory chip select	Low
DDR_WEB	Output	DDR memory write enable	Low
DDR_RASB	Output	DDR memory row address strobe	Low
DDR_CASB	Output	DDR memory column address strobe	Low
DDR_DM[DDRBITS/8-1:0]	Output	DDR memory data mask	Low
DDR_DQS[DDRBITS/8-1:0]	Bidir	DDR memory data strobe	Low
DDR_AD[13:0]	Output	DDR memory address bus	Low
DDR_BA[1:0]	Output	DDR memory bank address	Low
DDR_DQ[DDRBITS-1:0]	BiDir	DDR memory data bus	-

1) see GRLIB IP Library User’s Manual 2) Polarity selected with the oepol generic

25.6 Library dependencies

Table 192 shows libraries used when instantiating the core (VHDL libraries).

*Table 192.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

25.7 Component declaration

```
component ddrspa
  generic (
    fabtech : integer := 0;
    memtech : integer := 0;
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#f00#;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#fff#;
    MHz     : integer := 100;
    clkmul  : integer := 2;
    clkdiv  : integer := 2;
    col     : integer := 9;
    Mbyte   : integer := 16;
    rstdel  : integer := 200;
    pwron   : integer := 0;
    oepol   : integer := 0;
    ddrbits : integer := 16;
    ahbfreq : integer := 50
  );
  port (
    rst_ddr : in  std_ulogic;
    rst_ahb : in  std_ulogic;
    clk_ddr : in  std_ulogic;
    clk_ahb : in  std_ulogic;
    lock    : out std_ulogic;-- DCM locked
    clkddro : out std_ulogic;-- DCM locked
    clkddri : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    ddr_clk : out std_logic_vector(2 downto 0);
    ddr_clkb: out std_logic_vector(2 downto 0);
    ddr_clk_fb_out  : out std_logic;
    ddr_clk_fb  : in std_logic;
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                       -- ddr write enable
    ddr_rasb  : out std_ulogic;                       -- ddr ras
    ddr_casb  : out std_ulogic;                       -- ddr cas
    ddr_dm   : out std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_ad      : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba      : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq    : inout  std_logic_vector (ddrbits-1 downto 0) -- ddr data

  );
  end component;
```

25.8 Instantiation

This examples shows how the core can be instantiated.

The DDR SDRAM controller decodes SDRAM area at 0x40000000 - 0x7FFFFFFF. The SDRAM registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;

entity ddr_Interface is
  port ( ddr_clk  : out std_logic_vector(2 downto 0);
    ddr_clkb  : out std_logic_vector(2 downto 0);
    ddr_clk_fb  : in std_logic;
    ddr_clk_fb_out  : out std_logic;
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                       -- ddr write enable
    ddr_rasb  : out std_ulogic;                       -- ddr ras
    ddr_casb  : out std_ulogic;                       -- ddr cas
    ddr_dm   : out std_logic_vector (7 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (7 downto 0);    -- ddr dqs
    ddr_ad      : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba      : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq  : inout std_logic_vector (63 downto 0); -- ddr data

);
end;

architecture rtl of mctrl_ex is

  -- AMBA bus
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal clkml, lock : std_ulogic;


begin

-- DDR controller


ddrc : ddrspa generic map ( fabtech => virtex4, ddrbits => 64, memtech => memtech,
hindex => 4, haddr => 16#400#, hmask => 16#F00#, ioaddr => 1,
pwron => 1, MHz => 100, col => 9, Mbyte => 32, ahbfreq => 50, ddrbits => 64)
port map (
rstneg, rstn, lclk, clkm, lock, clkml, clkml,  ahbsi, ahbso(4),
ddr_clk, ddr_clkb, ddr_clk_fb_out, ddr_clk_fb,
ddr_cke, ddr_csb, ddr_web, ddr_rasb, ddr_casb,
ddr_dm, ddr_dqs, ddr_adl, ddr_ba, ddr_dq);
```

26 DDR2SPA - 16-, 32- and 64-bit DDR2 Controller

26.1 Overview

DDR2SPA is a DDR2 SDRAM controller with AMBA AHB back-end. The controller can interface two 16-, 32- or 64-bit DDR2 memory banks to a 32-bit AHB bus. The controller acts as a slave on the AHB bus where it occupies a configurable amount of address space for DDR2 SDRAM access. The DDR2 controller is programmed by writing to two configuration register mapped located in AHB I/O address space. Internally, DDR2SPA consists of a ABH/DDR2 controller and a technology specific DDR2 PHY. Currently supported technologies for the PHY is Xilinx Virtex4 and Virtex5 and Altera StratixIII. The modular design of DDR2SPA allows to add support for other target technologies in a simple manner. The DDR2SPA is used in the following GRLIB template design: *leon3-xilinx-ml505, leon3-altera-ep3sl150*.

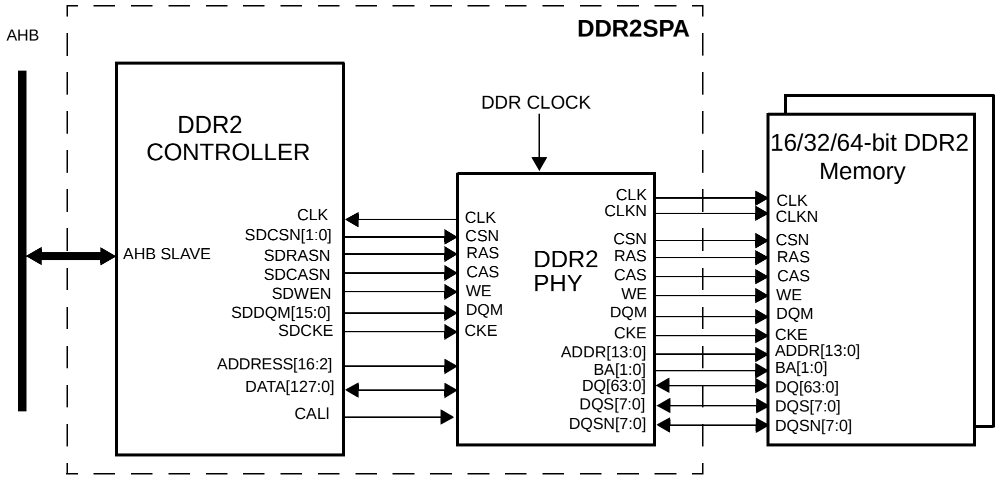


Figure 77. DDR2SPA Memory controller connected to AMBA bus and DDR2 SDRAM

26.2 Operation

26.2.1 General

Double data-rate SDRAM (DDR2 RAM) access is supported to two banks of 16-, 32- or 64-bit DDR2-400 compatible memory devices. The controller supports 64M, 128M, 256M, 512M and 1G devices with 9- 12 column-address bits, up to 14 row-address bits, and 4 internal banks. The size of each chip select can be programmed in binary steps between 8 Mbyte and 1024 Mbyte. The DDR data width is set by the DDRBITS generic, and will affect the width of DM, DQS and DQ signals. The DDR data width does not change the behavior of the AHB interface, except for data latency.

26.2.2 Read cycles

An AHB read access to the controller will cause a corresponding access to the external DDR2 RAM. The read cycle is started by performing an ACTIVATE command to the desired bank and row, followed by a READ command. CAS latency of 3(CL=3) is always used. Byte, half-word (16-bit) and word (32-bit) AHB accesses are supported. Incremental AHB burst access are supported for 32-bit words only. The read cycle(s) are always terminated with a PRE-CHARGE command, no banks are left open between two accesses. DDR2 read cycles are always performed in (aligned) 8-word bursts, which are stored in a FIFO. After an initial latency, the data is then read out on the AHB bus with zero waitstates.

26.2.3 Write cycles

Write cycles are performed similarly to read cycles, with the difference that WRITE commands are issued after activation. An AHB write burst will store up to 8 words in a FIFO, before writing the data to the DDR2 memory. As in the read case, only word bursts are supported

26.2.4 Initialization

If the `pwron` VHDL generic is 1, then the DDR2 controller will automatically perform the DDR2 initialization sequence as described in the JEDEC DDR2 standard: PRE-CHARGE, LOAD-EXT-MODE-REG2, LOAD-EXTMODE-REG3, LOAD-EXTMODE-REG1, LOAD-MODE-REG, PRE-CHARGE, 2xREFRESH, LOAD-MODE-REG, LOAD-EXTMODE-REG1(ODC default), and LOAD-EXTMODE-REG1(ODC exit). The VHDL generics **col** and **Mbyte** can be used to also set the correct address decoding after reset. In this case, no further software initialization is needed. The DDR2 initialization can be performed at a later stage by setting bit 16 in the DDR2 control register DDR2CFG1.

26.2.5 Configurable DDR2 SDRAM timing parameters

To provide optimum access cycles for different DDR2 devices (and at different frequencies), four timing parameters can be programmed through the memory configuration registers (DDR2CFG1, and DDR2CFG3): TRCD, TRP, TRFCD and TWR. The value of these field affects the DDR2RAM timing as described in table 193.

*Table 193.*DDR2 SDRAM programmable minimum timing parameters

DDR2 SDRAM timing parameter	**Minimum timing (clocks)**
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to read/write (t_{RCD})	TRCD + 2
Write recovery time (t_{WR})	TWR-3

If the TCD, TRP, TRFC and TWR are programmed such that the DDR2 specifications are full filled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 130 and 200 MHz operation and the resulting DDR2 SDRAM timing (in ns):

*Table 194.*DDR SDRAM example programming

DDR2 SDRAM settings	t_{RCD}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
130 MHz: TRP=0, TRFC=7, TRCD=0	15	76	15	76	61
200 MHz: TRP=1, TRFC=13, TRCD=1	15	65	15	80	50

The timing values shown in table 194 is the minimum value and is valid for 64-bit DDR2 interface. The DDR2SPA controller always uses CAS latency (CL) of three cycles.

26.2.6 Refresh

The DDR2SPA 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 DDR2CFG1 register. Depending on SDRAM type, the required period is typically 7.8 us (corresponding to 780 at 100 MHz). The generated refresh period is calculated as (reload value+1)/sysclk. The refresh function is enabled by bit 31 in DDR2CFG1 register.

26.2.7 DDR2 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG1: PRE-CHARGE, LOAD-EXTMODE-REG, LOAD-MODE-REG and REFRESH. If the LMR command is issued, the PLL Reset bit as programmed in SDCFG1 will be used, remaining fields are fixed: 4 word sequential burst, CL=3. If the LEMR command is issued, the OCD bits will be used as programmed in the DDR2CFG1 register. All other bits are set to zero. The command field will be cleared after a command has been executed.

26.2.8 Clocking

The DDR2 controller is designed to operate with two clock domains, one for the DDR2 memory clock and one for the AHB clock. The two clock domains do not have to be the same or be phase-aligned. The DDR2 input clock (CLK_DDR) can be multiplied and divided by the DDR PHY to form the final DDR2 clock frequency. The final DDR2 clock is driven on one output (CLKDDRO), which should always be connected to the CLKDDRI input. If the AHB clock and DDR clock area generated from the same clock source, a timing-ignore constraint should be placed between the CLK_AHB and CLKDDRI to avoid optimization of false-paths during synthesis and place&route.

The PHY generates the internal DDR read clock from the DDR2 clock (CLKDDRO). This means that the controller do not use a feed-back clock from the DDR memory to generate the read clock. To compensate for the board delay, the input pad-delay can be calibrated using the DDR2CFG2 control register. The supported frequencies for the Xilinx PHY depends on the clock-to-output delay of the DDR output registers, and the internal delay from the DDR input registers to the read data FIFO. Virtex5 can typically run at 200 MHz.

26.2.9 Read calibration

The input pad-delay for the data signals (DDR_DQ) can be calibrated to compensate for the board delay between the memory and the FPGA. This can be done in two ways: by setting the VHDL generics ddelayb[7:0] to the desired delay value (only in Xilinx), or by writing to the DDR2CFG3 register to increase/decrease the delay value. The input delay can be set to between 0 and 63 tap delays for Virtex4/5 and between 0 and 15 for StratixIII, each with an delay of 78 ps (see FPGA data sheet for more information). The delay for each byte can be calibrated independently. There is two bits in the control register for each byte. One bit determine if the delay should be increased or decreased. The other bit determine if the delay for this byte should be updated. Bit 31 in the DDR2CFG3 register is a reset bit for the delays. If set to ‘1’, the delay will be set to the default value (set by the VHDL generic ddelay[7:0]). To increase the calibration range, the controller can add additional read latency cycles. The number of additional read latency cycles is set by the RD bits in the DDR2CFG3 register. For Altera StratixIII the data sampling clock can be skewed to increase the calibration range. This is done writing the PLL_SKEW bits in the DDR2CFG3 register. For Altera StratixIII, additional input delay can be added by setting the D2 and D3 delay for the DDR_DQ signals in the .qsf constrains file, see Altera StratixIII documentation on input delays for more information. For Altera StratixIII the phase of the data sampling clock can be set by the VHDL generic rskew.

For Xilinx Sparatan-3 a clock loop is utilized for sampling of incoming data. The DDR_CLK_FB_OUT port should therefore be connected to a signal path of equal length as the DDR_CLK + DDR_DQS signal path. The other end of the signal path is to be connected to the DDR_CLK_FB port. The fed back clock can then be skewed for alignment with incoming data using the rskew generic. The rskew generic can be set between +/-255 resulting in a linear +/-360 degree change of the clock skew. Bits 29 and 30 in the DDR2CFG3 register can be used for altering the skew at runtime.

26.2.10 Pads

The DDR2SPA core has technology-specific pads inside the core. The external DDR2 signals should therefore be connected directly the top-level ports, without any logic in between.

26.2.11 Registers

The DDR2SPA core implements three control registers. The registers are mapped into AHB I/O address space defined by the AHB BAR1 of the core.

*Table 195.*DDR2 controller registers

Address offset - AHB I/O - BAR1	**Register**
0x00	DDR2 SDRAM control register (DDR2CFG1)
0x04	DDR2 SDRAM control register (read-only) (DDR2CFG2)
0x08	DDR2 SDRAM control register (DDR2CFG3)

.

Table 196. DDR2 SRAM control register 1 (DDR2CFG1)

31	30	29 28	27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	OCD	EMR	RES	tCD	SDRAM bank size	SRAM col. size	SDRAM command	PR	IN	CE	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled.
30	OCD operation
29: 28	Selects Extended mode register (1,2,3)
27	RESERVED
26	SDRAM tRCD delay. Sets tRCD to 2 + field value clocks.
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 8 Mbyte, “001”= 16 Mbyte, “010”= 32 Mbyte .... “111”= 1024 Mbyte.
22: 21	SDRAM column size. “00”=512, “01”=1024, “10”=2048, “11”=4096
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	PLL Reset. This bit is used to set the PLL RESET bit during LOAD-CONFIG-REG commands.
16	Initialize (IN). Set to ‘1’ to perform power-on DDR RAM initialisation. Is automatically cleared when initialisation is completed.
15	Clock enable (CE). This value is driven on the CKE inputs of the DDR RAM. Should be set to ‘1’ for correct operation.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

Table 197. DDR2 SDRAM configuration register 2 (DDR2CFG2)

31 15	14 12	11 0
reserved	Data width	DDR Clock frequency

14: 12	DDR data width: “001” = 16 bits, “010” = 32 bits, “011” = 64 bits (read-only)
11: 0	Frequency of the (external) DDR clock (read-only)

Table 198. DDR2 SDRAM configuration register 3 (DDR2CFG3)

31	30 29	28	27 23	22 18	17 16	15 8	7 0
	PLL	tRP	tWR	tRFC	RD	inc/dec delay	Update delay

31	Reset byte delay

Table 198. DDR2 SDRAM configuration register 3 (DDR2CFG3)

30: 29	PLL_SKEW Bit 29: Update clock phase Bit 30: 1 = Inc / 0 = Dec clock phase
28	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1).
27 23	SDRAM write recovery time. tWR will be equal to field value - 3 system clock
22: 18	SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks.
17: 16	Number of added read delay cycles, default = 1
15: 8	Set to ‘1’ to increment byte delay, set to ‘0’ to decrement delay
7: 0	Set to ‘1’ to update byte delay

26.3 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x02E. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

26.4 Configuration options

Table 199 shows the configuration options of the core (VHDL generics).

*Table 199.*Configuration options

Generic	Function	Allowed range	Default
fabtech	PHY technology selection	virtex4, virtex5, stratix3	virtex4
memtech	Technology selection for DDR FIFOs	inferred, virtex2, virtex4, spartan3e, altera	inferred
hindex	AHB slave index	0 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where DDR control register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space	0 - 16#FFF#	16#FFF#
ddrbits	Data bus width of external DDR memory	16, 32, 64	16
MHz	DDR clock input frequency in MHz.	10 - 200	100
clkmul, clkdiv	The DDR input clock is multiplied with the clkmul generic and divided with clkdiv to create the final DDR clock	2 - 32	2
rstdel	Clock reset delay in micro-seconds.	1 - 1023	200
col	Default number of column address bits	9 - 12	9
Mbyte	Default memory chip select bank size in Mbyte	8 - 1024	16
pwron	Enable SDRAM at power-on initialization	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
ahbfreq	Frequency in MHz of the AHB clock domain	1 - 1023	50
readdly	Additional read latency cycles (used to increase calibra-tion range)	0-3	1
TRFC	Reset value for the tRFC timing parameter in ns.	75-155	130
ddelayb0*	Input data delay for bit[7:0]	0-63	0
ddelayb1*	Input data delay for bit[15:8]	0-63	0
ddelayb2*	Input data delay for bit[23:16]	0-63	0
ddelayb3*	Input data delay for bit[31:24]	0-63	0
ddelayb4*	Input data delay for bit[39:32]	0-63	0
ddelayb5*	Input data delay for bit[47:40]	0-63	0
ddelayb6*	Input data delay for bit[55:48]	0-63	0
ddelayb7*	Input data delay for bit[63:56]	0-63	0
numidelctrl*	Number of IDELAYCTRL the core will instantiate	-	4
norefclk*	Set to 1 if no 200 MHz reference clock is connected to clkref200 input.	0-1	0
odten	Enable odt: 0 = Disabled, 1 = 75Ohm, 2 =150Ohm, 3 = 50Ohm	0-3	0
rskew**	Set the phase relationship between the DDR controller clock and the input data sampling clock. Sets the phase in ps.	0 - 9999	0

* only available in Virtex4/5 implementation. ** only available in StratixIII implementation.

26.5 Implementation

26.5.1 Technology mapping

The core has two technology mapping VHDL generics: *memtech* and *padtech*. The VHDL generic *memtech* controls the technology used for memory cell implementation. The VHDL generic *padtech* controls the technology used in the PHY implementation. See the GRLIB Users's Manual for available settings.

26.5.2 RAM usage

The FIFOs in the core are implemented with the *syncram_2p* (with separate clock for each port) component found in the technology mapping library (TECHMAP). The number of RAMs used for the FIFO implementation depends om the DDR data width, set by the *ddrbits* VHDL generic.

*Table 200.*RAM usage

RAM dimension (depth x width)	**Number of RAMs (DDR data width 64)**	**Number of RAMs (DDR data width 32)**	**Number of RAMs (DDR data width 16)**
4 x 128	1		
4 x 32	4		
5 x 64		1	
5 x 32		2	
6 x 32			2

26.6 Signal descriptions

Table 201 shows the interface signals of the core (VHDL ports).

*Table 201.*Signal descriptions

Signal name	Type	Function	Active
RST_DDR	Input	Reset input for the DDR PHY	Low
RST_AHB	Input	Reset input for AHB clock domain	Low
CLK_DDR	Input	DDR input Clock	-
CLK_AHB	Input	AHB clock	-
CLKREF200	Input	200 MHz reference clock	-
LOCK	Output	DDR clock generator locked	High
CLKDDRO		Internal DDR clock output after clock multiplication	
CLKDDRI		Clock input for the internal DDR clock domain. Must be connected to CLKDDRO.	
AHBSI	Input	AHB slave input signals	-
AHBSO	Output	AHB slave output signals	-
DDR_CLK[2:0]	Output	DDR memory clocks (positive)	High
DDR_CLKB[2:0]	Output	DDR memory clocks (negative)	Low
DDR_CLK_FB_OUT	Output	DDR data synchronization clock, connect this to a signal path with equal length of the DDR_CLK trace + DDR_DQS trace	-
DDR_CLK_FB	Input	DDR data synchronization clock, connect this to the other end of the signal path connected to DDR_CLK_FB_OUT	-
DDR_CKE[1:0]	Output	DDR memory clock enable	High
DDR_CSB[1:0]	Output	DDR memory chip select	Low
DDR_WEB	Output	DDR memory write enable	Low
DDR_RASB	Output	DDR memory row address strobe	Low
DDR_CASB	Output	DDR memory column address strobe	Low
DDR_DM[DDRBITS/8-1:0]	Output	DDR memory data mask	Low
DDR_DQS[DDRBITS/8-1:0]	Bidir	DDR memory data strobe	Low
DDR_DQSN[DDRBITS/8-1:0]	Bidir	DDR memory data strobe (inverted)	High
DDR_AD[13:0]	Output	DDR memory address bus	Low
DDR_BA[1:0]	Output	DDR memory bank address	Low
DDR_DQ[DDRBITS-1:0]	BiDir	DDR memory data bus	-
DDR_ODT[1:0]	Output	DDR memory odt	Low

1) see GRLIB IP Library User's Manual 2) Polarity selected with the oepol generic

26.7 Library dependencies

Table 202 shows libraries used when instantiating the core (VHDL libraries).

*Table 202.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

26.8 Component declaration

```
component ddr2spa
  generic (
    fabtech : integer := 0;
    memtech : integer := 0;
    hindex  : integer := 0;
    haddr   : integer := 0;
    hmask   : integer := 16#f00#;
    ioaddr  : integer := 16#000#;
    iomask  : integer := 16#fff#;
    MHz     : integer := 100;
    clkmul  : integer := 2;
    clkdiv  : integer := 2;
    col     : integer := 9;
    Mbyte   : integer := 16;
    rstdel  : integer := 200;
    pwron   : integer := 0;
    oepol   : integer := 0;
    ddrbits : integer := 16;
    ahbfreq : integer := 50;
    readdly : integer := 1;
    ddelayb0: integer := 0;
    ddelayb1: integer := 0;
    ddelayb2: integer := 0;
    ddelayb3: integer := 0;
    ddelayb4: integer := 0;
    ddelayb5: integer := 0;
    ddelayb6: integer := 0;
    ddelayb7: integer := 0
  );
  port (
    rst_ddr    : in  std_ulogic;
    rst_ahb    : in  std_ulogic;
    clk_ddr    : in  std_ulogic;
    clk_ahb    : in  std_ulogic;
    clkref200 : in std_ulogic;
    lock       : out std_ulogic;-- DCM locked
    clkddro    : out std_ulogic;-- DCM locked
    clkddri    : in  std_ulogic;
    ahbsi      : in  ahb_slv_in_type;
    ahbso      : out ahb_slv_out_type;
    ddr_clk    : out std_logic_vector(2 downto 0);
    ddr_clkb   : out std_logic_vector(2 downto 0);
    ddr_cke    : out std_logic_vector(1 downto 0);
    ddr_csb    : out std_logic_vector(1 downto 0);
    ddr_web    : out std_ulogic;                       -- ddr write enable
    ddr_rasb   : out std_ulogic;                       -- ddr ras
    ddr_casb   : out std_ulogic;                       -- ddr cas
    ddr_dm     : out std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dm
    ddr_dqs    : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_dqsn   : inout std_logic_vector (ddrbits/8-1 downto 0);    -- ddr dqs
    ddr_ad     : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba     : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq     : inout  std_logic_vector (ddrbits-1 downto 0); -- ddr data
    ddr_odt    : out std_logic_vector(1 downto 0) -- odt
  );
  end component;
```

26.9 Instantiation

This example shows how the core can be instantiated.

The DDR SDRAM controller decodes SDRAM area at 0x40000000 - 0x7FFFFFFF. The DDR2 SDRAM registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;

entity ddr_Interface is
  port (
    ddr_clk  : out std_logic_vector(2 downto 0);
    ddr_clkb : out std_logic_vector(2 downto 0);
    ddr_cke  : out std_logic_vector(1 downto 0);
    ddr_csb  : out std_logic_vector(1 downto 0);
    ddr_web  : out std_ulogic;                       -- ddr write enable
    ddr_rasb : out std_ulogic;                       -- ddr ras
    ddr_casb : out std_ulogic;                       -- ddr cas
    ddr_dm   : out std_logic_vector (7 downto 0);    -- ddr dm
    ddr_dqs  : inout std_logic_vector (7 downto 0);    -- ddr dqs
    ddr_dqsn : inout std_logic_vector (7 downto 0);    -- ddr dqsn
    ddr_ad   : out std_logic_vector (13 downto 0);   -- ddr address
    ddr_ba   : out std_logic_vector (1 downto 0);    -- ddr bank address
    ddr_dq   : inout std_logic_vector (63 downto 0); -- ddr data
    ddr_odt  : out std_logic_vector (1 downto 0) -- ddr odt
);
end;

architecture rtl of mctrl_ex is

  -- AMBA bus
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal clkml, lock, clk_200,
signal clk_200 : std_ulogic; -- 200 MHz reference clock
signal ddrclkin, ahbclk : std_ulogic; -- DDR input clock and AMBA sys clock
signal rstn  : std_ulogic; -- Synchronous reset signal
signal reset : std_ulogic; -- Asynchronous reset signal


begin

-- DDR controller


ddrc : ddr2spa generic map ( fabtech => virtex4, ddrbits => 64, memtech => memtech,
hindex => 4, haddr => 16#400#, hmask => 16#F00#, ioaddr => 1,
pwron => 1, MHz => 100, col => 9, Mbyte => 32, ahbfreq => 50, ddrbits => 64,
readdly => 1, ddelayb0 => 0, ddelayb1 => 0, ddelayb2 => 0, ddelayb3 => 0,
ddelayb4 => 0, ddelayb5 => 0, ddelayb6 => 0, ddelayb7 => 0)
port map (
reset, rstn, ddrclkin, ahbclk, clk_200, lock, clkml, clkml,  ahbsi, ahbso(4),
ddr_clk, ddr_clkb,
ddr_cke, ddr_csb, ddr_web, ddr_rasb, ddr_casb,
ddr_dm, ddr_dqs, ddr_adl, ddr_ba, ddr_dq, ddr_odt);
```

27 DIV32 - Signed/unsigned 64/32 divider module

27.1 Overview

The divider module performs signed/unsigned 64-bit by 32-bit division. It implements the radix-2 non-restoring iterative division algorithm. The division operation takes 36 clock cycles. The divider leaves no remainder. The result is rounded towards zero. Negative result, zero result and overflow (according to the overflow detection method B of SPARC V8 Architecture manual) are detected.

27.2 Operation

The division is started when ‘1’ is samples on DIVI.START on positive clock edge. Operands are latched externally and provided on inputs DIVI.Y, DIVI.OP1 and DIVI.OP2 during the whole operation. The result appears on the outputs during the clock cycle following the clock cycle after the DIVO.READY was asserted. Asserting the HOLD input at any time will freeze the operation, until HOLDN is de-asserted.

27.3 Signal descriptions

Table 203 shows the interface signals of the core (VHDL ports).

*Table 203.*Signal declarations

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
HOLDN	N/A	Input	Hold	Low
DIVI	Y[32:0]	Input	Dividend - MSB part Y[32] - Sign bit Y[31:0] - Dividend MSB part in 2’s complement format	High
	OP1[32:0]		Dividend - LSB part OP1[32] - Sign bit OP1[31:0] - Dividend LSB part in 2’s complement format	High
	FLUSH		Flush current operation	High
	SIGNED		Signed division	High
	START		Start division	High
DIVO	READY	Output	The result is available one clock after the ready signal is asserted.	High
	NREADY		Not used	-
	ICC[3:0]		Condition codes ICC[3] - Negative result ICC[2] - Zero result ICC[1] - Overflow ICC[0] - Not used. Always ‘0’.	High
	RESULT[31:0]		Result	High

27.4 Library dependencies

Table 204 shows libraries used when instantiating the core (VHDL libraries).

*Table 204.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	ARITH	Signals, component	Divider module signals, component declaration

27.5 Component declaration

The core has the following component declaration.

```
component div32
port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    holdn   : in  std_ulogic;
    divi    : in  div32_in_type;
    divo    : out div32_out_type
);
end component;
```

27.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use gaisler.arith.all;

.
.
.

signal divi  : div32_in_type;
signal divo  : div32_out_type;

begin

div0 : div32 port map (rst, clk, holdn, divi, divo);

end;
```

28 DSU3 - LEON3 Hardware Debug Support Unit

28.1 Overview

To simplify debugging on target hardware, the LEON3 processor implements a debug mode during which the pipeline is idle and the processor is controlled through a special debug interface. The LEON3 Debug Support Unit (DSU) is used to control the processor during debug mode. The DSU acts as an AHB slave and can be accessed by any AHB master. An external debug host can therefore access the DSU through several different interfaces. Such an interface can be a serial UART (RS232), JTAG, PCI, USB or ethernet. The DSU supports multi-processor systems and can handle up to 16 processors.

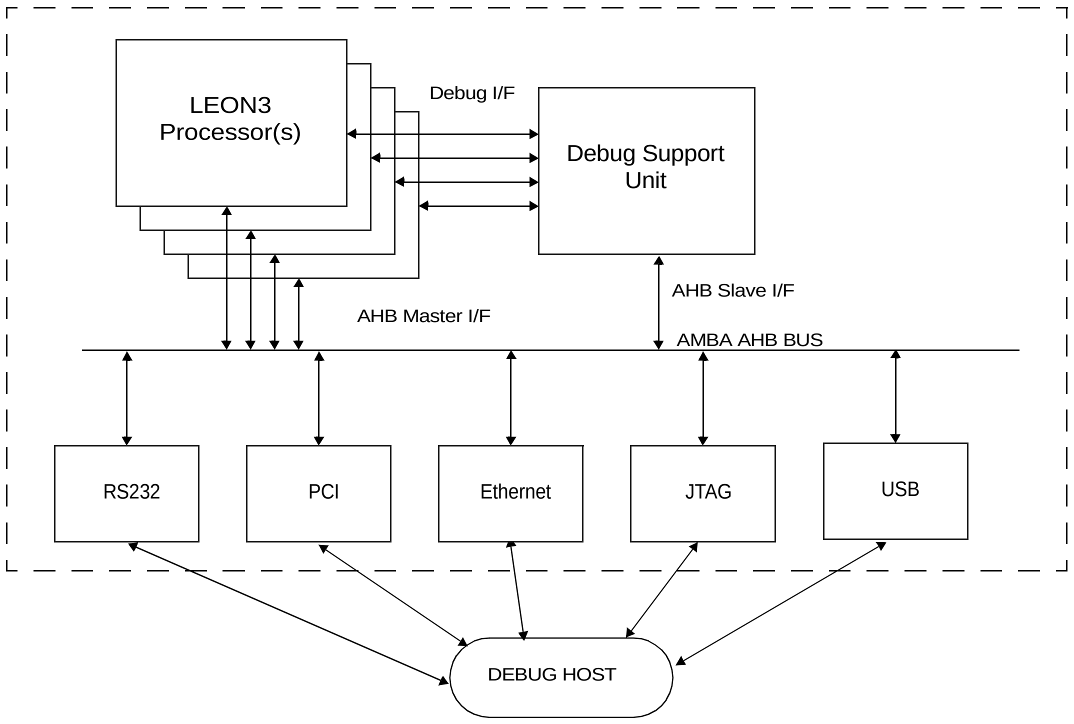


Figure 78. LEON3/DSU Connection

28.2 Operation

Through the DSU AHB slave interface, any AHB master can access the processor registers and the contents of the instruction trace buffer. The DSU control registers can be accessed at any time, while the processor registers, caches and trace buffer can only be accessed when the processor has entered debug mode. In debug mode, the processor pipeline is held and the processor state can be accessed 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
- one of the processors in a multiprocessor system has entered the debug mode

DSU AHB breakpoint hit

The debug mode can only be entered when the debug support unit is enabled through an external signal (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.

When a processor is in the debug mode, an access to ASI diagnostic area is forwarded to the IU which performs access with ASI equal to value in the DSU ASI register and address consisting of 20 LSB bits of the original address.

28.3 AHB Trace Buffer

The AHB trace buffer consists of a circular buffer that stores AHB data transfers. The address, data and various control signals of the AHB bus are stored and can be read out for later analysis. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 205.*AHB Trace buffer data allocation

Bits	Name	Definition
127	AHB breakpoint hit	Set to ‘1’ if a DSU AHB breakpoint hit occurred.
126	-	Not used
125:96	Time tag	DSU time tag counter
95	-	Not used
94:80	Hirq	AHB HIRQ[15:1]
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

In addition to the AHB signals, the DSU time tag counter is also stored in the trace.

The trace buffer is enabled by setting the enable bit (EN) in the trace control register. Each AHB transfer is then stored in the buffer in a circular manner. The address to which the next transfer is written is held in the trace buffer index register, and is automatically incremented after each transfer. Tracing is stopped when the EN bit is reset, or when a AHB breakpoint is hit. Tracing is temporarily suspended when the processor enters debug mode. Note that neither the trace buffer memory nor the breakpoint registers (see below) can be read/written by software when the trace buffer is enabled.

28.4 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The instruction trace buffer is located in the processor, and read out via the DSU. The trace buffer is 128 bits wide, the information stored is indicated in the table below:

*Table 206.*Instruction trace buffer data allocation

Bits	Name	Definition
127	-	Unused
126	Multi-cycle instruction	Set to ‘1’ on the second and third instance of a multi-cycle instruction (LDD, ST or FPOP)
125:96	Time tag	The value of the DSU time tag 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

During 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 the processor enters debug mode, tracing is suspended. The trace buffer and the trace buffer control register can be read and written while the processor is in the debug mode. During the instruction tracing (processor in normal mode) the trace buffer and the trace buffer control register can not be accessed.

28.5 DSU memory map

The DSU memory map can be seen in table 207 below. In a multiprocessor systems, the register map is duplicated and address bits 27 - 24 are used to index the processor.

*Table 207.*DSU memory map

Address offset	**Register**
0x000000	DSU control register
0x000008	Time tag counter
0x000020	Break and Single Step register
0x000024	Debug Mode Mask register
0x000040	AHB trace buffer control register
0x000044	AHB trace buffer index register
0x000050	AHB breakpoint address 1
0x000054	AHB mask register 1
0x000058	AHB breakpoint address 2
0x00005c	AHB mask register 2
0x100000 - 0x10FFFF	Instruction trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x110000	Instruction Trace buffer control register
0x200000 - 0x210000	AHB trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x300000 - 0x3007FC	IU register file
0x300800 - 0x300FFC	IU register file check bits (LEON3FT only)
0x301000 - 0x30107C	FPU register file
0x400000 - 0x4FFFFC	IU special purpose registers
0x400000	Y register
0x400004	PSR register
0x400008	WIM register
0x40000C	TBR register
0x400010	PC register
0x400014	NPC register
0x400018	FSR register
0x40001C	CPSR register
0x400020	DSU trap register
0x400024	DSU ASI register
0x400040 - 0x40007C	ASR16 - ASR31 (when implemented)

0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0])
	ASI = 0x9 : Local instruction RAM
	ASI = 0xB : Local data RAM
	ASI = 0xC : Instruction cache tags
	ASI = 0xD : Instruction cache data
	ASI = 0xE : Data cache tags
	ASI = 0xF : Data cache data
	ASI = 0x1E : Separate snoop tags

The addresses of the IU registers depends on how many register windows has been implemented:

- %o*n* : 0x300000 + (((psr.cwp * 64) + 32 + *n**4) mod (NWINDOWS*64))
- %l*n* : 0x300000 + (((psr.cwp * 64) + 64 + *n**4) mod (NWINDOWS*64))
- %i*n* : 0x300000 + (((psr.cwp * 64) + 96 + *n**4) mod (NWINDOWS*64))
- %g*n* : 0x300000 + (NWINDOWS*64)
- %f*n* : 0x301000 + *n**4

28.6 DSU registers

28.6.1 DSU control register

The DSU is controlled by the DSU control register:

.

31	11	10	9	8	7	6	5	4	3	2	1	0
	PW	HL	PE	EB	EE	DM	BZ	BX	BS	BW	BE	TE

Figure 79. DSU control register

[0]: Trace enable (TE). Enables instruction tracing. If set the instructions will be stored in the trace buffer. Remains set when then processor enters debug or error mode.

[1]: Break on error (BE) - if set, will force the processor to debug mode when the processor would have entered error condition (trap in trap).

[2]: Break on IU watchpoint (BW)- if set, debug mode will be forced on a IU watchpoint (trap 0xb).

[3]: Break on S/W breakpoint (BS) - if set, debug mode will be forced when an breakpoint instruction (ta 1) is executed.

[4]: Break on trap (BX) - if set, will force the processor into debug mode when any trap occurs.

[5]: Break on error traps (BZ) - if set, will force the processor into debug mode on all *except* the following traps: priviledged_instruction, fpu_disabled, window_overflow, window_underflow, asynchronous_interrupt, ticc_trap.

[6]: Debug mode (DM). Indicates when the processor has entered debug mode (read-only).

[7]: EE - value of the external DSUEN signal (read-only)

[8]: EB - value of the external DSUBRE signal (read-only)

[9]: Processor error mode (PE) - returns ‘1’ on read when processor is in error mode, else ‘0’. If written with ‘1’, it will clear the error and halt mode.

[10]: Processor halt (HL). Returns ‘1’ on read when processor is halted. If the processor is in debug mode, setting this bit will put the processor in halt mode.

[11]: Power down (PW). Returns ‘1’ when processor in in power-down mode.

28.6.2 DSU Break and Single Step register

This register is used to break or single step the processor(s). This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

31		18	17	16	15		2	1	0
SS15	. . .	SS2	SS1	SS0	BN15	. . .	BN2	BN1	BN0

Figure 80. DSU Break and Single Step register

[15:0] : Break now (BNx) -Force processor x into debug mode if the Break on watchpoint (BW) bit in the processors DSU control register is set. If cleared, the processor x will resume execution.

[31:16] : Single step (SSx) - if set, the processor x will execute one instruction and return to debug mode. The bit remains set after the processor goes into the debug mode.

28.6.3 DSU Debug Mode Mask Register

When one of the processors in a multiprocessor LEON3 system enters the debug mode the value of the DSU Debug Mode Mask register determines if the other processors are forced in the debug mode. This register controls all processors in a multi-processor system, and is only accessible in the DSU memory map of processor 0.

31		18	17	16	15		2	1	0
DM15	. . .	DM2	DM1	DM0	ED15	. . .	ED2	ED1	ED0

Figure 81. DSU Debug Mode Mask register

[15:0] : Enter debug mode (EDx) - Force processor x into debug mode if any of processors in a multiprocessor system enters the debug mode. If 0, the processor x will not enter the debug mode.

[31:16]: Debug mode mask. If set, the corresponding processor will not be able to force running processors into debug mode even if it enters debug mode.

28.6.4 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).

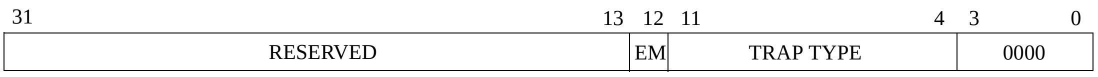


Figure 82. DSU trap register

[11:4]: 8-bit SPARC trap type

[12]: Error mode (EM). Set if the trap would have cause the processor to enter error mode.

28.6.5 Trace buffer time tag counter

The trace buffer time tag counter is incremented each clock as long as the processor is running. The counter is stopped when the processor enters debug mode, and restarted when execution is resumed.

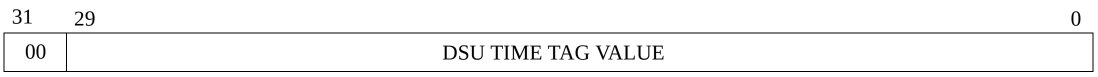


Figure 83. Trace buffer time tag counter

The value is used as time tag in the instruction and AHB trace buffer.

The width of the timer (up to 30 bits) is configurable through the DSU generic port.

28.6.6 DSU ASI register

The DSU can perform diagnostic accesses to different ASI areas. The value in the ASI diagnostic access register is used as ASI while the address is supplied from the DSU.

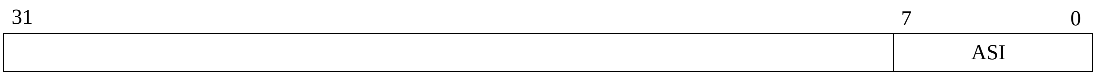


Figure 84. ASI diagnostic access register

[7:0]: ASI to be used on diagnostic ASI access

28.6.7 AHB Trace buffer control register

The AHB trace buffer is controlled by the AHB trace buffer control register:

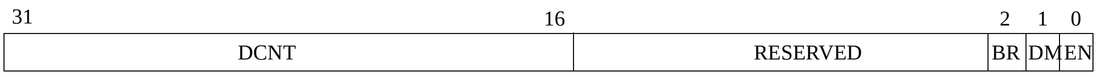


Figure 85. AHB trace buffer control register

[0]: Trace enable (EN). Enables the trace buffer.

[1]: Delay counter mode (DM). Indicates that the trace buffer is in delay counter mode.

[2]: Break (BR). If set, the processor will be put in debug mode when AHB trace buffer stops due to AHB breakpoint hit.

[31:16] Trace buffer delay counter (DCNT). Note that the number of bits actually implemented depends on the size of the trace buffer.

28.6.8 AHB trace buffer index register

The AHB trace buffer index register contains the address of the next trace line to be written.

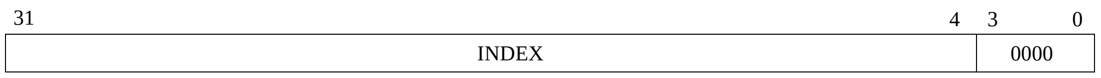


Figure 86. AHB trace buffer index register

31:4	Trace buffer index counter (INDEX). Note that the number of bits actually implemented depends on the size of the trace buffer.

28.6.9 AHB trace buffer breakpoint registers

The DSU contains two breakpoint registers for matching AHB addresses. A breakpoint hit is used to freeze the trace buffer by automatically clearing the enable bit. Freezing can be delayed by programming the DCNT field in the trace buffer control register to a non-zero value. In this case, the DCNT value will be decremented for each additional trace until it reaches zero, after which the trace buffer is frozen. 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 AHB load or store accesses, the LD and/or ST bits should be set.

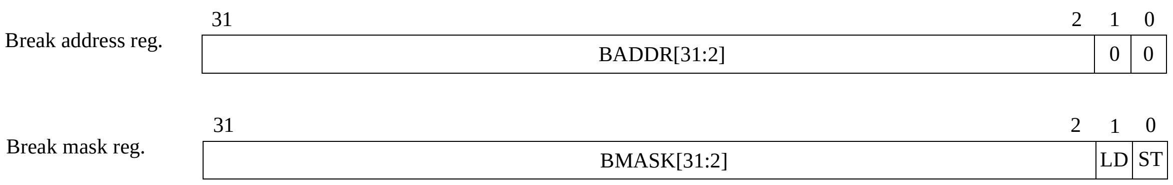


Figure 87. Trace buffer breakpoint registers

[31:2]:	Breakpoint address (bits 31:2)
[31:2]:	Breakpoint mask (see text)
[1]:	LD - break on data load address
[0]:	ST - beak on data store address

28.6.10 Instruction trace control register

The instruction trace control register contains a pointer that indicates the next line of the instruction trace buffer to be written.

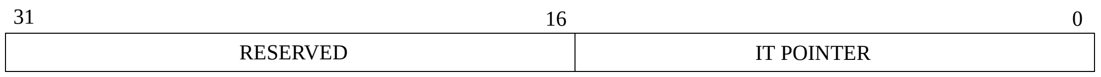


Figure 88. Instruction trace control register

[15:0]	Instruction trace pointer. Note that the number of bits actually implemented depends on the size of the trace buffer.

28.7 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x017. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

28.8 Technology mapping

DSU3 has one technology mapping generic, *tech*. This generic controls the implementation of which tehcnology will be used to implement the trace buffer memories. The AHB trace buffer will use four identical *syncram* block to implement the buffer memory. The all syncrams will be 32-bit wide. The depth will depend on the KBYTES generic, which indicates the total size of trace buffer in Kbytes. If KBYTES = 1 (1 Kbyte), then four RAM blocks of 64x32 will be used. If KBYTES = 2, then the RAM blocks will be 128x32 and so on.

28.9 Configuration options

Table 208 shows the configuration options of the core (VHDL generics).

*Table 208.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - AHBSLVMAX-1	0
haddr	AHB slave address (AHB[31:20])	0 - 16#FFF#	16#900#
hmask	AHB slave address mask	0 - 16#FFF#	16#F00#
ncpu	Number of attached processors	1 - 16	1
tbits	Number of bits in the time tag counter	2 - 30	30
tech	Memory technology for trace buffer RAM	0 - TECHMAX-1	0 (inferred)
kbytes	Size of trace buffer memory in Kbytes. A value of 0 will disable the trace buffer function.	0 - 64	0 (disabled)

28.10 Signal descriptions

Table 209 shows the interface signals of the core (VHDL ports).

*Table 209.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
DBGI	-	Input	Debug signals from LEON3	-
DBGO	-	Output	Debug signals to LEON3	-
DSUI	ENABLE	Input	DSU enable	High
	BREAK	Input	DSU break	High
DSUO	ACTIVE	Output	Debug mode	High
	PWD[n-1 : 0]	Output	Clock gating enable for processor [n]	High

* see GRLIB IP Library User’s Manual

28.11 Library dependencies

Table 210 shows libraries used when instantiating the core (VHDL libraries).

*Table 210.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3	Component, signals	Component declaration, signals declaration

28.12 Component declaration

The core has the following component declaration.

```
component dsu3
  generic (
    hindex : integer := 0;
    haddr : integer := 16#900#;
    hmask : integer := 16#f00#;
    ncpu    : integer := 1;
    tbits   : integer := 30;
    tech    : integer := 0;
    irq     : integer := 0;
    kbytes  : integer := 0
  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    ahbmi  : in  ahb_mst_in_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : out ahb_slv_out_type;
    dbgi   : in l3_debug_out_vector(0 to NCPU-1);
    dbgo   : out l3_debug_in_vector(0 to NCPU-1);
    dsui   : in dsu_in_type;
    dsuo   : out dsu_out_type
  );
  end component;
```

28.13 Instantiation

This example shows how the core can be instantiated.

The DSU is always instantiated with at least one LEON3 processor. It is suitable to use a generate loop for the instantiation of the processors and DSU and showed below.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

constant NCPU : integer := 1; -- select number of processors

signal leon3i : l3_in_vector(0 to NCPU-1);
signal leon3o : l3_out_vector(0 to NCPU-1);
signal irqi   : irq_in_vector(0 to NCPU-1);
signal irqo   : irq_out_vector(0 to NCPU-1);

signal dbgi : l3_debug_in_vector(0 to NCPU-1);
signal dbgo : l3_debug_out_vector(0 to NCPU-1);

signal dsui   : dsu_in_type;
signal dsuo   : dsu_out_type;

.
begin

cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s-- LEON3 processor
    generic map (ahbndx => i, fabtech => FABTECH, memtech => MEMTECH)
    port map (clkm, rstn, ahbmi, ahbmo(i), ahbsi, ahbsi, ahbso,
 irqi(i), irqo(i), dbgi(i), dbgo(i));
    irqi(i) <= leon3o(i).irq; leon3i(i).irq <= irqo(i);
end generate;

dsu0 : dsu3-- LEON3 Debug Support Unit
    generic map (ahbndx => 2, ncpu => NCPU, tech => memtech, kbytes => 2)
    port map (rstn, clkm, ahbmi, ahbsi, ahbso(2), dbgo, dbgi, dsui, dsuo);
dsui.enable <= dsuen; dsui.break <= dsubre; dsuact <= dsuo.active;
```

29 FTAHBRAM - On-chip SRAM with EDAC and AHB interface

29.1 Overview

The FTAHBRAM core is a version of the AHBRAM core with added Error Detection And Correction (EDAC). The on-chip memory is accessed via an AMBA AHB slave interface. The memory implements 2 kbytes of data (configured via the *kbytes* VHDL generics). Registers are accessed via an AMB APB interface.

The on-chip memory implements volatile memory that is protected by means of Error Detection And Correction (EDAC). One error can be corrected and two errors can be detected, which is performed by using a (32, 7) BCH code. Some of the optional features available are single error counter, diagnostic reads and writes and autoscrubbing (automatic correction of single errors during reads). Configuration is performed via a configuration register.

Figure 89 shows a block diagram of the internals of the memory.

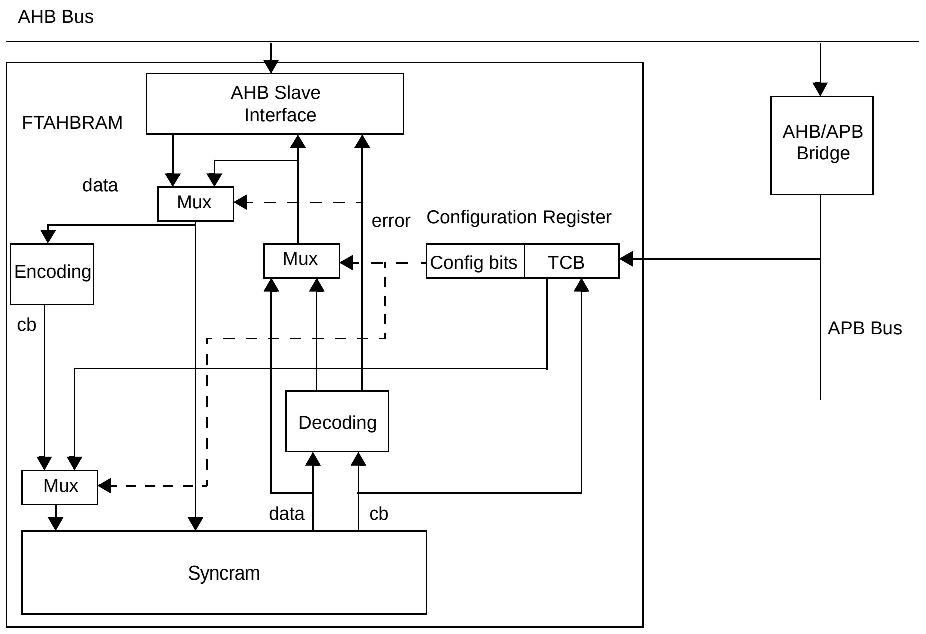


Figure 89. Block diagram

29.2 Operation

The on-chip fault tolerant memory is accessed through an AMBA AHB slave interface.

The memory address range is configurable with VHDL generics. As for the standard AHB RAM, the memory technology and size is configurable through the tech and kbytes VHDL generics. The minimum size is 1 kb and the maximum is technology dependent but the values can only be increased in binary steps.

Run-time configuration is done by writing to a configuration register accessed through an AMBA APB interface.

The address of the interface and the available options are configured with VHDL generics. The EDAC functionality can be completely removed by setting the edacen VHDL generic to zero during synthesis. The APB interface is also removed since it is redundant without EDAC.

The following can be configured during run-time: EDAC can be enabled and disabled. When it is disabled, reads and writes will behave as the standard memory. Read and write diagnostics can be controlled through separate bits. The single error counter can be reset.

If EDAC is disabled (EN bit in configuration register set to 0) write data is passed directly to the memory area and read data will appear on the AHB bus immediately after it arrives from memory. If EDAC is enabled write data is passed to an encoder which outputs a 7-bit checksum. The checksum is stored together with the data in memory and the whole operation is performed without any added waitstates. This applies to word stores (32-bit). If a byte or halfword store is performed, the whole word to which the byte or halfword belongs must first be read from memory (read - modify - write). A new checksum is calculated when the new data is placed in the word and both data and checksum are stored in memory. This is done with 1 - 2 additional waitstates compared to the non EDAC case.

Reads with EDAC disabled are performed with 0 or 1 waitstates while there could also be 2 waitstates when EDAC is enabled. There is no difference between word and subword reads. Table 211 shows a summary of the number of waitstates for the different operations with and without EDAC.

*Table 211.*Summary of the number of waitstates for the different operations for the memory.

Operation	**Waitstates with EDAC Disabled**	**Waitstates with EDAC Enabled**
Read	0 - 1	0 - 2
Word write	0	0
Subword write	0	1 - 2

If the ahbpipe VHDL generic is set to 1, pipeline registers are enabled for the AHB input signals. If the pipeline registers are enabled, one extra waitstate should be added to the read and subword write cases in Table 211.

When EDAC is used, the data is decoded the first cycle after it arrives from the memory and appears on the bus the next cycle if no uncorrectable error is detected. The decoding is done by comparing the stored checksum with a new one which is calculated from the stored data. This decoding is also done during the read phase for a subword write. A so-called syndrome is generated from the comparison between the checksum and it determines the number of errors that occured. One error is automatically corrected and this situation is not visible on the bus. Two or more detected errors cannot be corrected so the operation is aborted and the required two cycle error response is given on the AHB bus (see the AMBA manual for more details). If no errors are detected data is passed through the decoder unaltered.

As mentioned earlier the memory provides read and write diagnostics when EDAC is enabled. When write diagnostics are enabled, the calculated checksum is not stored in memory during the write phase. Instead, the TCB field from the configuration register is used. In the same manner, if read diagnostics are enabled, the stored checksum from memory is stored in the TCB field during a read (and also during a subword write). This way, the EDAC functionality can be tested during run-time. Note that checkbits are stored in TCB during reads and subword writes even if a multiple error is detected.

An additional feature is the single error counter which can be enabled with the *errcnten* VHDL generic. A single error counter (SEC) field is present in the configuration register, and is incremented each time a single databit error is encountered (reads or subword writes). The number of bits of this counter is 8, set with the *cntbits* VHDL generic. It is accessed through the configuration register. Each counter bit can be reset to zero by writing a one to it. The counter saturates at the value 2^8 - 1 ($2^{cntbits}$ - 1). Each time a single error is detected the aramo.ce signal will be driven high for one cycle. This signal should be connected to an AHB status register which stores information and generates interrupts (see the AHB Status register documentation for more information).

Autoscrubbing is an error handling feature which is enabled with the *autoscrub* VHDL generic and cannot be controlled through the configuration register. If enabled, every single error encountered during a read results in the word being written back with the error corrected and new checkbits generated. It is not visible externally except for that it can generate an extra waitstate. This happens if the read is followed by an odd numbered read in a burst sequence of reads or if a subword write follows. These situations are very rare during normal operation so the total timing impact is negligible. The aramo.ce signal is normally used to generate interrupts which starts an interrupt routine that corrects errors.

Since this is not necessary when autoscrubbing is enabled, aramo.ce should not be connected to an AHB status register or the interrupt should be disabled in the interrupt controller.

29.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 212.*FTAHBRAM registers

APB Address offset	**Register**
0x0	Configuration Register

Table 213. Configuration Register

31 13+8	12+8 13	12 10	9	8	7	6 0
	SEC	MEMSIZE	WB	RB	EN	TCB

12+8: 13	Single error counter (SEC): Incremented each time a single error is corrected (includes errors on checkbits). Each bit can be set to zero by writing a one to it. This feature is only available if the errcnten VHDL generic is set.
12: 10	Log2 of the current memory size
9	Write Bypass (WB): When set, the TCB field is stored as check bits when a write is performed to the memory.
8	Read Bypass (RB) : When set during a read or subword write, the check bits loaded from memory are stored in the TCB field.
7	EDAC Enable (EB): When set, the EDAC is used otherwise it is bypassed during read and write operations.
6: 0	Test Check Bits (TCB) : Used as checkbits when the WB bit is set during writes and loaded with the check bits during a read operation when the RB bit is set.

Any unused most significant bits are reserved. Always read as ‘000...0’.

All fields except TCB are initialised at reset. The EDAC is initally disabled (EN = 0), which also applies to diagnostics fiels (RB and WB are zero).

When available, the single error counter (SEC) field is cleared to zero.

29.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x050. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

29.5 Configuration options

Table 214 shows the configuration options of the core (VHDL generics).

*Table 214.*Configuration options

Generic	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the memory.	0 to NAHBMAX-1	0
haddr	ADDR field of the AHB BAR	0 to 16#FFF#	0
hmask	MASK field of the AHB BAR	0 to 16#FFF#	16#FFF#
tech	Memory technology	0 to NTECH	0
kbytes	SRAM size in kbytes	1 to targetdep.	1
pindex	Selects which APB select signal (PSEL) will be used to access the memory configuration registers	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
edacen	Enable (1)/Disable (0) on-chip EDAC	0 to 1	0
autoscrub	Automatically store back corrected data with new check-bits during a read when a single error is detected. Is ignored when edacen is deasserted.	0 to 1	0
errcnten	Enables a single error counter	0 to 1	0
cntbits	number of bits in the single error counter	1 to 8	1
ahbpipe	Enable pipeline register on AHB input signals	0 to 1	0

29.6 Signal descriptions

Table 215 shows the interface signals of the core (VHDL ports).

*Table 215.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ARAMO	CE	Output	Single error detected	High

* see GRLIB IP Library User’s Manual

29.7 Library dependencies

Tabel 216 shows libraries used when instantiating the core (VHDL libraries).

*Table 216.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Signals and component declaration

29.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
library gaisler;

use grlib.amba.all;
use gaisler.misc.all;

entity ftram_ex is
  port(
    rst : std_ulogic;
    clk : std_ulogic;

    .... --others signals
  );
end;

architecture rtl of ftram_ex is

--AMBA signals
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_type;
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector;



--other needed signals here
signal stati   : ahbstat_in_type;
signal aramo   : ahbram_out_type;

begin

--other component instantiations here
...

-- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbso, stati, apbi, apbo(13));
    stati.cerror(1 to NAHBSLV-1) <= (others => ‘0’);

--FT AHB RAM
a0 : ftahbram generic map(hindex => 1, haddr => 1, tech => inferred,
  kbytes => 64, pindex => 4, paddr => 4, edacen => 1, autoscrub => 0,
  errcnt => 1, cntbits => 4)
  port map(rst, clk, ahbsi, ahbso(1), apbi, apbo(4), aramo);
  stati.cerror(0) <= aramo.ce;

end architecture;
```

30 FTMCTRL - 8/16/32-bit Memory Controller with EDAC

30.1 Overview

The FTMCTRL combined 8/16/32-bit memory controller provides a bridge between external memory and the AHB bus. The memory controller can handle four types of devices: PROM, asynchronous static ram (SRAM), synchronous dynamic ram (SDRAM) and memory mapped I/O devices (IO). The PROM, SRAM and SDRAM areas can be EDAC-protected using a (39,7) BCH code. The EDAC provides single-error correction and double-error detection for each 32-bit memory word.

The memory controller is configured through three configuration registers accessible via an APB bus interface. The external data bus can be configured in 8-, 16-, or 32-bit mode, depending on application requirements. The controller decodes three address spaces on the AHB bus (PROM, IO, and SRAM/ SDRAM). The addresses are determined through VHDL generics.

External chip-selects are provided for up to four PROM banks, one IO bank, five SRAM banks and two SDRAM banks. Figure 90 below shows how the connection to the different device types is made.

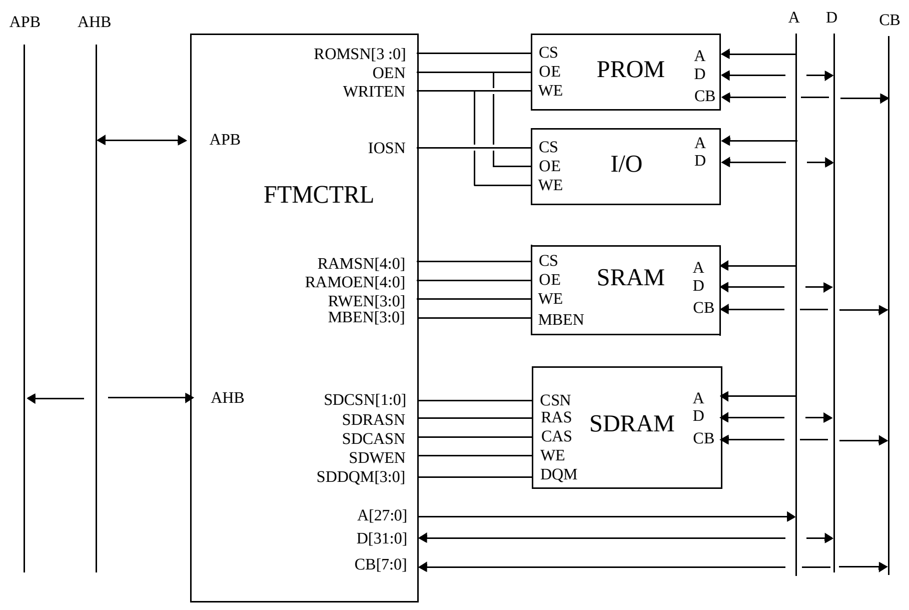


Figure 90. FTMCTRL connected to different types of memory devices

30.2 PROM access

Up to four PROM chip-select signals are provided for the PROM area, ROMSN[3:0]. There are two modes: one with two chip-select signals and one with four. The size of the banks can be set in binary steps from 16 kB to 256 MB.

A read access to PROM consists of two data cycles and between 0 and 30 waitstates. The read data (and optional EDAC check-bits) are latched on the rising edge of the clock on the last data cycle. 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 PROM devices. Figure 91 shows the basic read cycle waveform (zero waitstate) for non-consecutive PROM reads. Note that the address is undefined in the lead-out cycle. Figure 92 shows the timing for consecutive cycles (zero waitstate). Waitstates are added by extending the data2 phase. This is shown in figure 93 and applies to both consecutive and non-consecutive cycles. Only an even number of waitstates can be assigned to the PROM area.

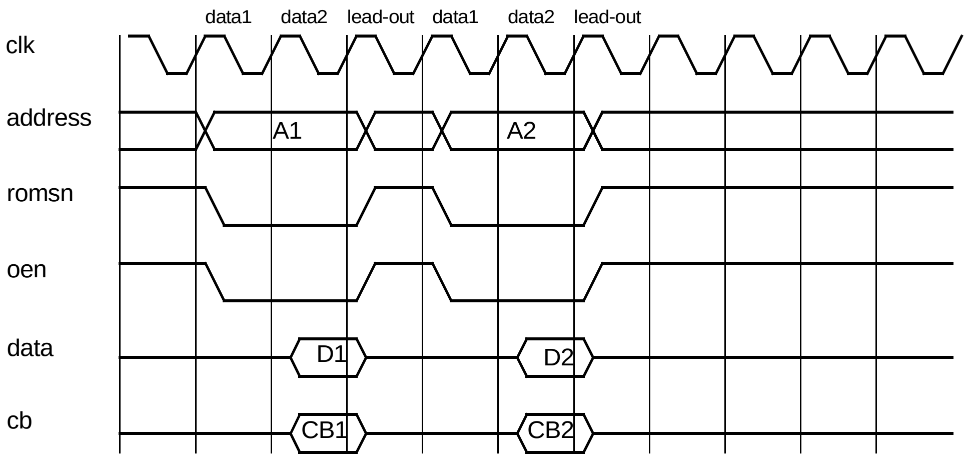


Figure 91. Prom non-consecutive read cyclecs.

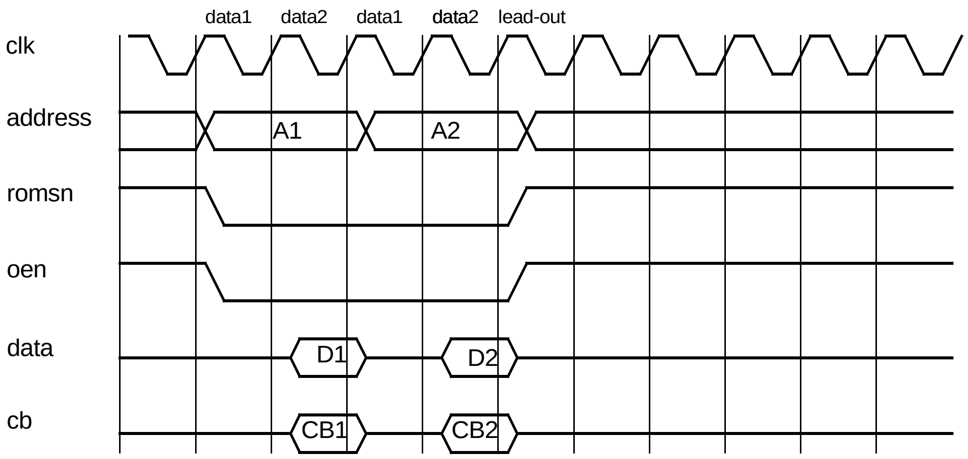


Figure 92. Prom consecutive read cyclecs.

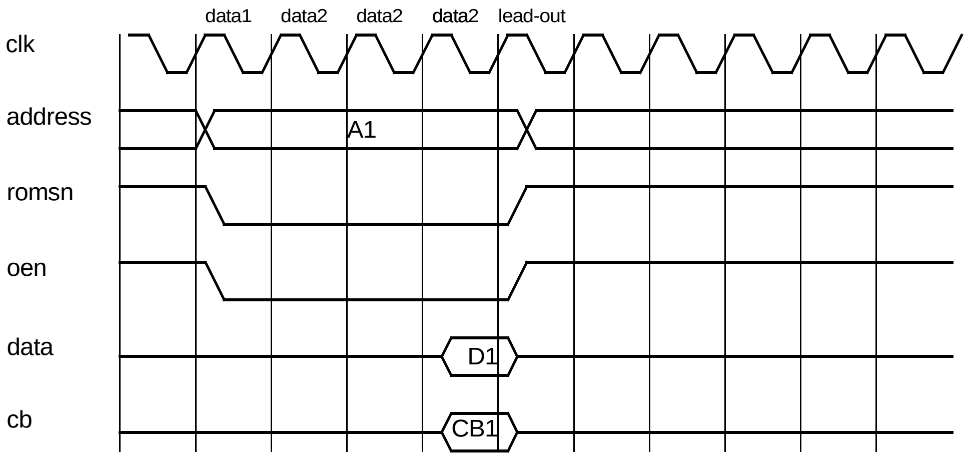


Figure 93. Prom read access with two waitstates.

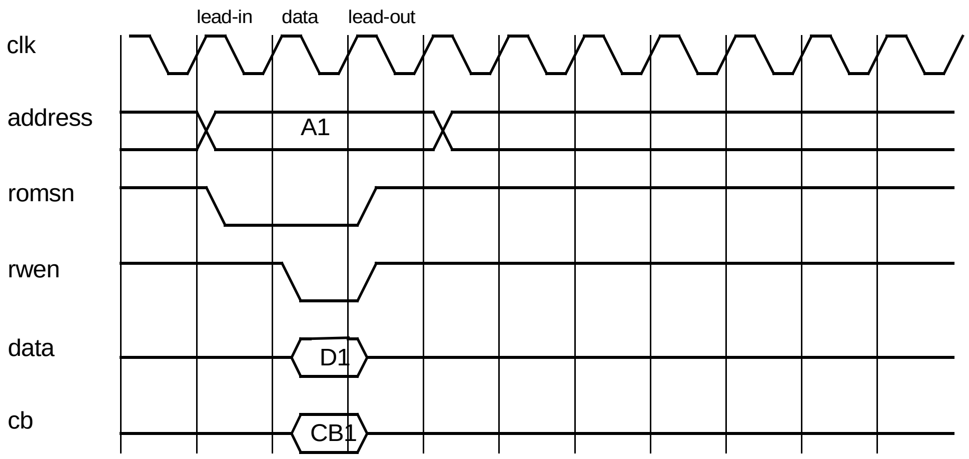


Figure 94. Prom write cycle (0-waitstates)

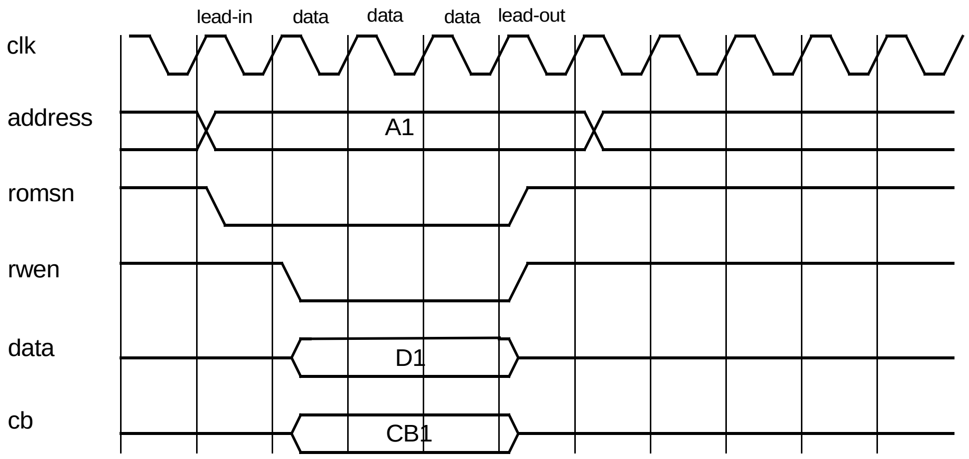


Figure 95. Prom write cycle (2-waitstates)

30.3 Memory mapped IO

Accesses to IO have similar timing as PROM accesses. The IO select (IOSN) and output enable (OEN) signals are delayed one clock to provide stable address before IOSN is asserted. All accesses are performed as non-consecutive accesses as shown in figure 96. The data2 phase is extended when waitstates are added.

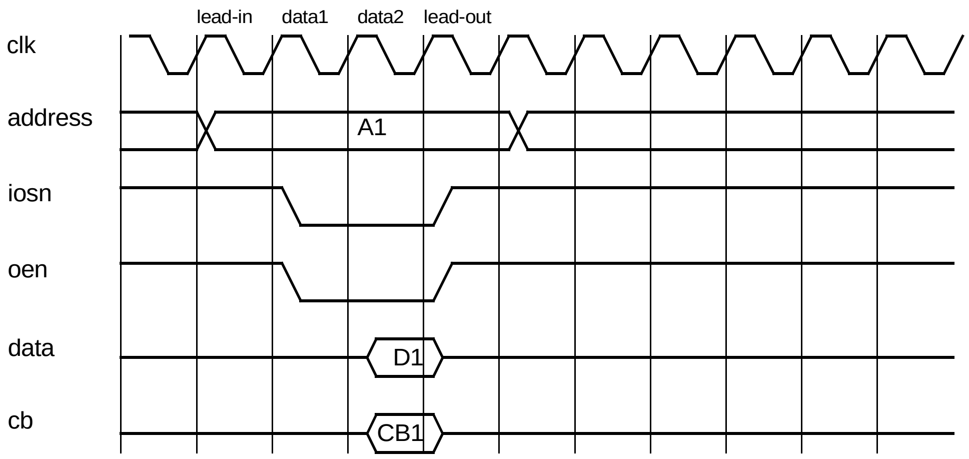


Figure 96. I/O read cycle (0-waitstates)

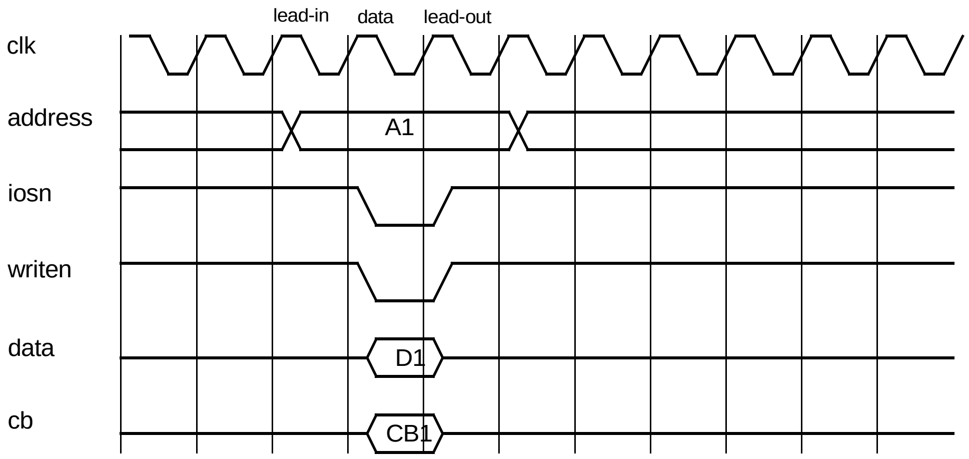


Figure 97. I/O write cycle (0-waitstates)

30.4 SRAM access

The SRAM area is 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 and cannot be used simultaneously with SDRAM memory. A read access to SRAM consists of two data cycles and between zero and three waitstates. The read data (and optional EDAC check-bits) are latched on the rising edge of the clock on the last data cycle. 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. Figure 98 shows the basic read cycle waveform (zero waitstate). Waitstates are added in the same way as for PROM in figure 93.

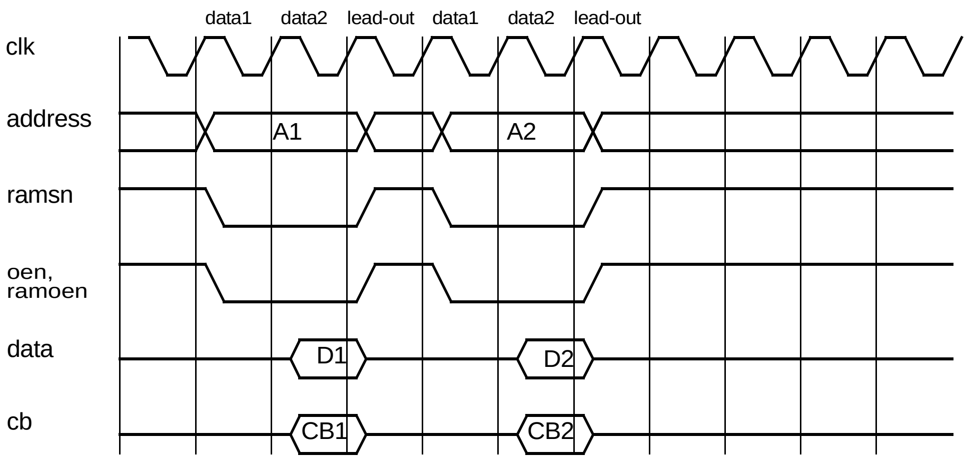


Figure 98. Sram non-consecutive read cyclecs.

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 takes a minimum of three cycles. Waitstates are added in the same way as for PROM.

Each byte lane has an individual write strobe to allow efficient byte and half-word writes. If the memory uses a common write strobe for the full 16- or 32-bit data, the read-modify-write bit MCFG2 should be set to enable read-modify-write cycles for sub-word writes.

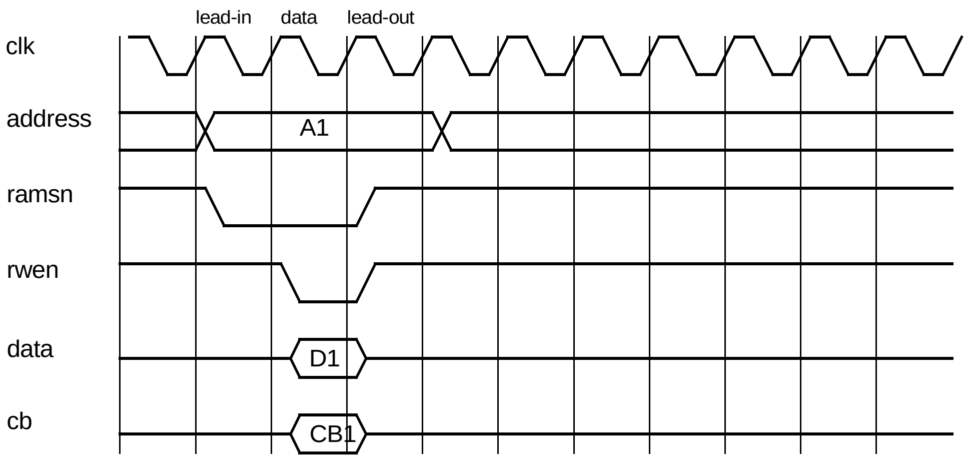


Figure 99. Sram write cycle (0-waitstates)

30.5 8-bit and 16-bit PROM and SRAM access

To support applications with low memory and performance requirements efficiently, the SRAM and PROM areas can be individually configured for 8- or 16-bit operation by programming the ROM and RAM width fields in the memory configuration registers. Since reads to memory are always done on 32-bit word basis, read access to 8-bit memory will be transformed in a burst of four read cycles while access to 16-bit memory will generate a burst of two 16-bit reads. During writes, only the necessary bytes will be written. Figure 100 shows an interface example with 8-bit PROM and 8-bit SRAM. Figure 101 shows an example of a 16-bit memory interface.

EDAC is not supported for 16-bit wide memories and therefore the EDAC enable bit corresponding to a 16-bit wide area must not be set.

It is not allowed to set the ROM or RAM width fields to 8-bit width if ram8 is not set and also not to 16-bit width if ram16 is not set.

The RMW bit must not be set if RAM EDAC is not enabled when RAM width is set to 8-bit.

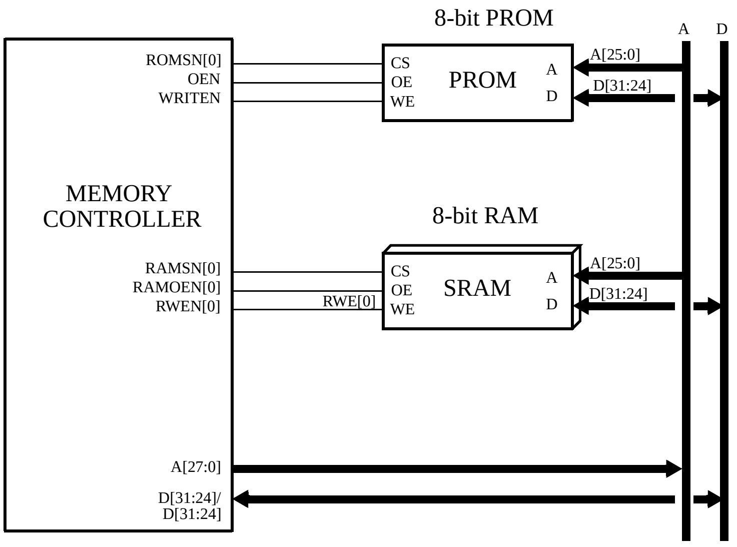


Figure 100. 8-bit memory interface example

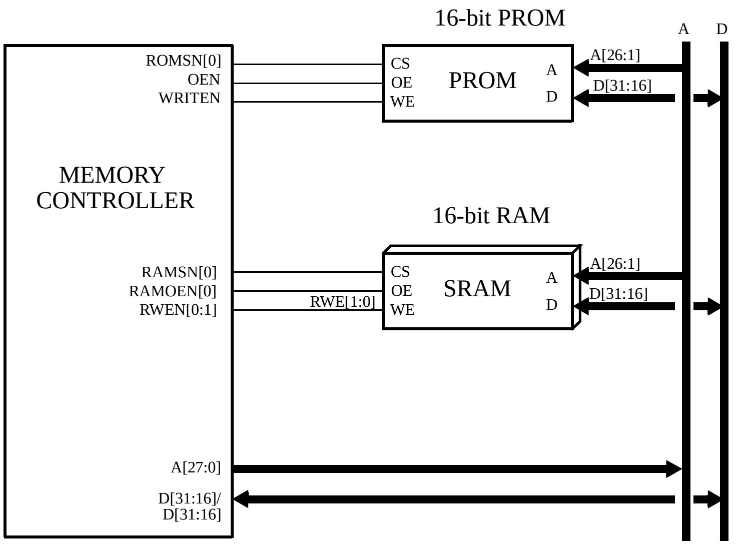


Figure 101. 16-bit memory interface example

In 8-bit mode, the PROM/SRAM devices should be connected to the MSB byte of the data bus (D[31:24]). The LSB address bus should be used for addressing (A[25:0]). In 16-bit mode, D[31:16] should be used as data bus, and A[26:1] as address bus. EDAC protection is not available in 16-bit mode.

30.6 8- and 16-bit I/O access

Similar to the PROM/SRAM areas, the IO area can also be configured to 8- or 16-bits mode. However, the I/O device will NOT be accessed by multiple 8/16 bits accesses as the memory areas, but only with one single access just as in 32-bit mode. To access an IO device on an 8-bit bus, only byte accesses should be used (LDUB/STB instructions for the CPU). To accesses an IO device on a 16-bit bus, only halfword accesses should be used (LDUH/STH instructions for the CPU). To access the I/O-area in 8- or 16-bit mode, ram8 or ram16 must be set respectively.

30.7 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. Burst cycles will not be generated to the IO area.

Only word (HSIZE = “010”) bursts of incremental type (HBURST=INCR, INCR4, INCR8 or INCR16) are supported.

30.8 SDRAM access

30.8.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. This is implemented by a special version of the SDCTRL SDRAM controller core from Gaisler Research, which is optionally instantiated as a sub-block. The SDRAM controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, and up to 13 row-address bits. The size 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). Both 32- and 64-bit data bus width is supported, allowing the interface of 64-bit DIMM modules. The memory controller can be configured to use either a shared or separate bus connecting the controller and SDRAM devices.

30.8.2 Address mapping

The two SDRAM chip-select signals are decoded. SDRAM area is mapped into the upper half of the RAM area defined by BAR2 register, and cannot be used simultaneously with fifth SRAM bank (RAMSN[4]). When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped into upper half of the RAM area 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 into the lower half of the RAM area.

30.8.3 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. The controller programs the SDRAM to use single location access on write. The controller programs the SDRAM to use line burst of length 8 when *pageburst* VHDL generic is 0. The controller programs the SDRAM to use page burst when *pageburst* VHDL generic is 1. The controller programs the SDRAM to use page burst or line burst of length 8, selectable via the MCFG2 register, when *pageburst* VHDL generic is 2.

30.8.4 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 217.

*Table 217.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 218.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

30.9 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 µs (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.

30.9.1 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. Line burst of length 8 will be set for read when *pageburst* VHDL generic is 0. Page burst will be set for read when *pageburst* VHDL generic is 1. Page burst or line burst of length 8, selectable via the MCFG2 register will be set, when *pageburst* VHDL generic is 2. Remaining fields are fixed: single location write, sequential burst. The command field will be cleared after a command has been executed. When changing the value of the CAS delay, a LOAD-MODE-REGISTER command should be generated at the same time. NOTE: when issuing SDRAM commands, the SDRAM refresh must be disabled.

30.9.2 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.

30.9.3 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.

30.9.4 Address bus

The memory controller can be configured to either share the address and data buses with the SRAM, or to use separate address and data buses. When the buses are shared, the address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. The MSB part of A[14:2] can be left unconnected if not used. When separate buses are used, the SDRAM address bus should be connected to SA[12:0] and the bank address to SA[14:13].

30.9.5 Data bus

SDRAM can be connected to the memory controller through the common or separate data bus. If the separate bus is used the width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used. There is a drive signal vector and separate data vector available for SDRAM. The drive vector has one drive signal for each data bit. These signals can be used to remove timing problems with the output delay when a separate SDRAM bus is used.

30.9.6 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera device, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed. For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

30.9.7 Initialisation

Each time the SDRAM is enabled (bit 14 in MCFG2), an SDRAM initialisation sequence will be sent to both SDRAM banks. The sequence consists of one PRECHARGE, eight AUTO-REFRESH and one LOAD-COMMAND-REGISTER command.

30.10 Memory EDAC

The FTMCTRL is provided with an EDAC that 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. A correctable error will be handled transparently by the memory controller, but adding one waitstate to the access. If an un-correctable error (double-error) is detected, the current AHB cycle will end with an error response. The EDAC can be used during access to PROM, SRAM and SDRAM areas by setting the corresponding EDAC enable bits in the MCFG3 register. The equations below show how the EDAC checkbits are generated:

CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31

CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28

$\overline{\text{CB2}}$ = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31

$\overline{\text{CB3}}$ = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29

CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31

CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31

CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31

If the SRAM 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. Data is always accessed as words (4 bytes at a time) and the corresponding checkbits are located at the address acquired by inverting the word address (bits 2 to 27) and using it as a byte address. The same chip-select is kept active. A word written as four bytes to addresses 0, 1, 2, 3 will have its checkbits at address 0xFFFFFFF, addresses 4, 5, 6, 7 at 0xFFFFFFE and so on. All the bits up to the maximum bank size will be inverted while the same chip-select is always asserted. This way all the bank sizes can be supported and no memory will be unused (except for a maximum of 4 byte in the gap between the data and checkbit area). A read access will automatically read the four data bytes individually from the nominal addresses and the EDAC checkbit byte from the top part of the bank. A write cycle is performed the same way. Byte or half-word write accesses will result in an automatic read-modify-write access where 4 data bytes and the checkbit byte are firstly read, and then 4 data bytes and the newly calculated checkbit byte are writen back to the memory. This 8-bit mode applies to SRAM while SDRAM always uses 32-bit accesses. The size of the memory bank is determined from the settings in MCFG2. The EDAC cannot be used on memory areas configured in 16-bit mode.

If the ROM is configured in 8-bit mode, EDAC protection is provided in a similar way as for the SRAM memory described above. The difference is that write accesses are not being handled automatically. Instead, write accesses must only be performed as individual byte accesses by the software, writing one byte at a time, and the corresponding checkbit byte must be calculated and be written to the correct location by the software.

The operation of the EDAC can be tested trough the MCFG3 register. 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 memory configuration register 2 must be set.

EDAC is not supported for 64-bit wide SDRAM data busses.

30.11 Bus Ready signalling

The BRDYN signal can be used to stretch all types of access cycles to the PROM, I/O area and the SRAM area decoded by RAMSN[4]. This covers read and write accesses in general, and additionally read-modify-write accesses to the SRAM area. 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 preceding the last one. If bit 29 in MCFG1 is set, BRDYN can be asserted asynchronously with the system clock. In this case, the read data must be kept stable until the de-assertion of OEN/RAMOEN and BRDYN must be asserted for at least 1.5 clock cycle. The use of BRDYN can be enabled separately for the PROM, I/O and RAMSN[4] areas. It is recommended that BRDYN is asserted until the corresponding chip select signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall.

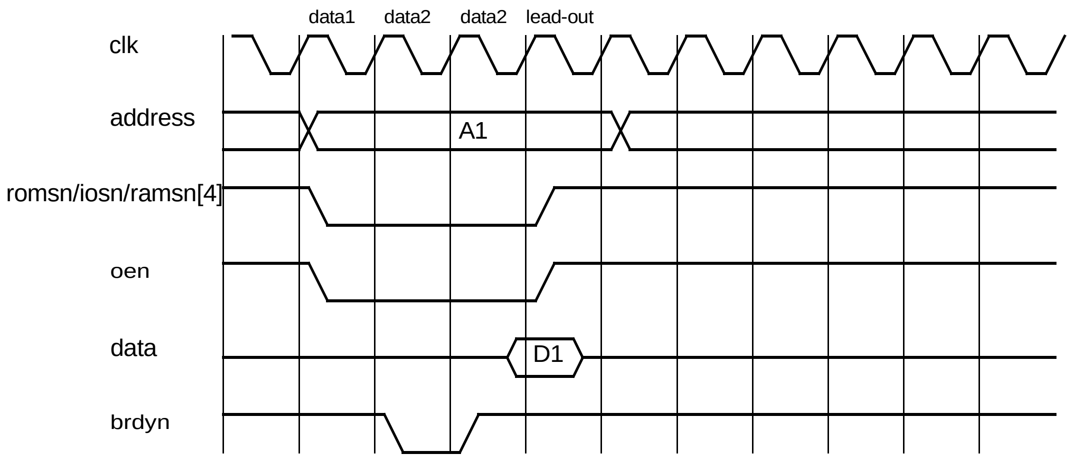


Figure 102. READ cycle with one extra data2 cycle added with BRDYN (synchronous sampling). Lead-out cycle is only applicable for I/O accesses.

Figure 103 shows the use of BRDYN with asynchronous sampling. BRDYN is kept asserted for more than 1.5 clock-cycle. Two synchronization registers are used so it will take at least one additional cycle from when BRDYN is first asserted until it is visible internally. In figure 103 one cycle is added to the data2 phase.

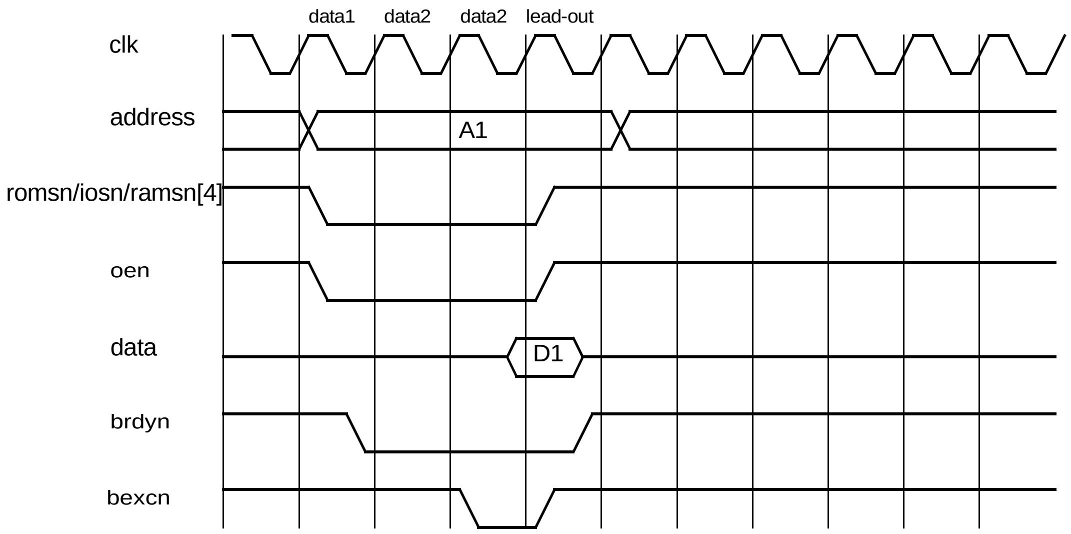


Figure 103. BRDYN (asynchronous) sampling and BEXCN timing. Lead-out cycle is only applicable for I/O-accesses.

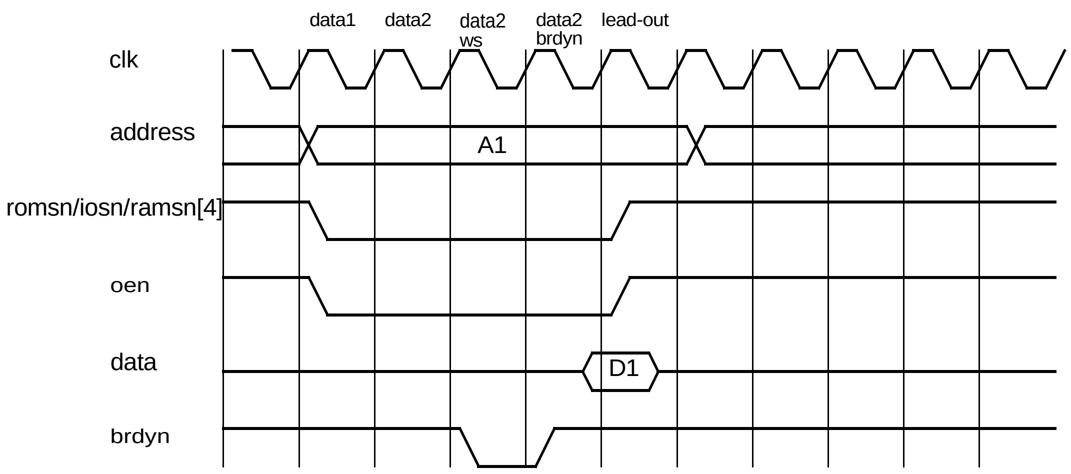


Figure 104. Read cycle with one waitstate (configured) and one BRDYN generated waitstate (synchronous sampling).

30.12 Access errors

An access error can be signalled by asserting the BEXCN signal for read and write accesses. For reads it is sampled together with the read data. For writes it is sampled on the last rising edge before chip select is de-asserted, which is controlled by means of waitstates or bus ready signalling. If the usage of BEXCN is enabled in memory configuration register 1, an error response will be generated on the internal AHB bus. BEXCN can be enabled or disabled through memory configuration register 1, and is active for all areas (PROM, IO and RAM). BEXCN is only sampled in the last access for 8- and 16-bit mode for RAM and PROM. That is, when four bytes are written for a word access to 8-bit wide memory BEXCN is only sampled in the last access with the same timing as a single access in 32-bit mode.

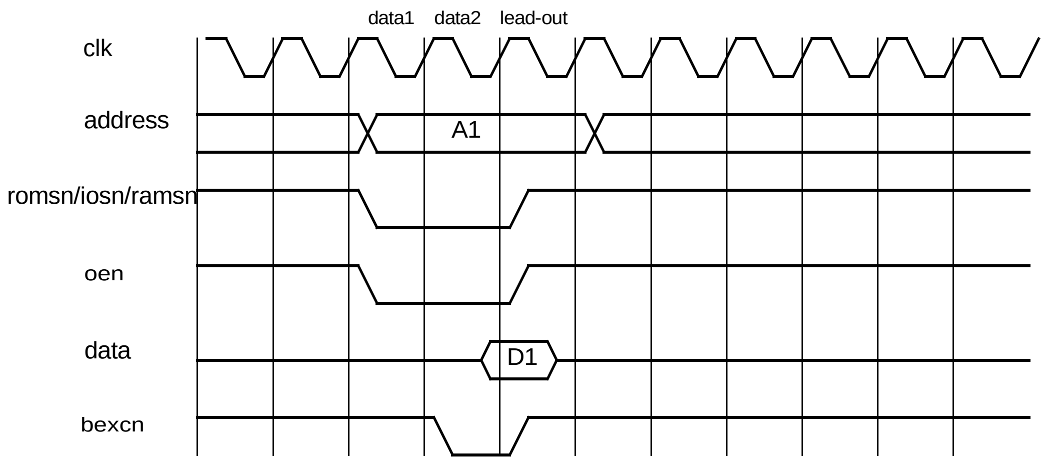


Figure 105. Read cycle with BEXCN.

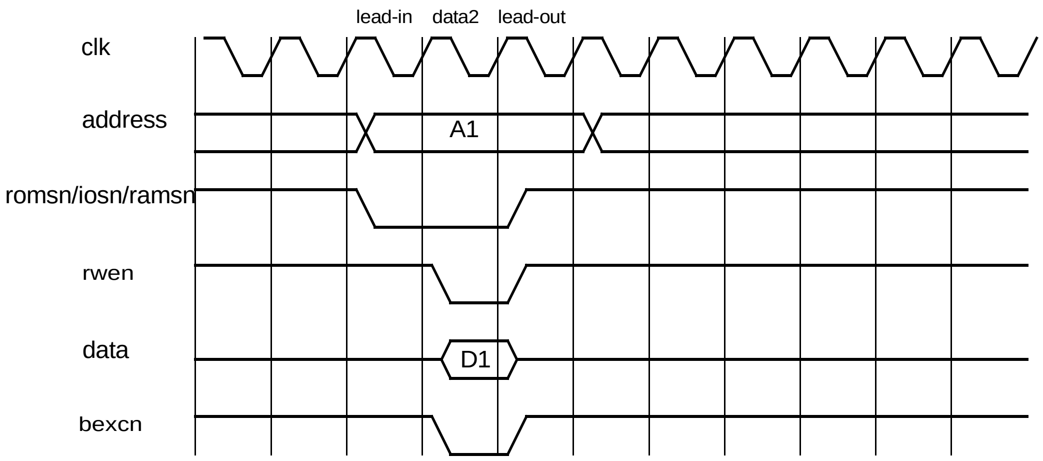


Figure 106. Write cycle with BEXCN. Chip-select (iosn) is not asserted in lead-in cycle for io-accesses.

30.13 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.

30.14 Output enable timing

A drive signal vector for the data I/O-pads is provided which has one drive signal for each data bit. It can be used if the synthesis tool does not generate separate registers automatically for the current technology. This can remove timing problems with output delay. An additional vector is used for the separate SDRAM bus.

30.15 Registers

The core is programmed through registers mapped into APB address space.

*Table 219.*FTMCTRL memory controller registers

APB Address offset	Register
0x0	Memory configuration register 1 (MCFG1)
0x4	Memory configuration register 2 (MCFG2)
0x8	Memory configuration register 3 (MCFG3)

30.15.1 Memory configuration register 1 (MCFG1)

Memory configuration register 1 is used to program the timing of rom and IO accesses.

Table 220. Memory configuration register 1.

31	30	29	28	27	26	25	24	23	20	19	18	17	
	PBRDY	ABRDY	IOBUSW		IBRDY	BEXCN		IO WAITSTATES		IOEN		ROMBANKSZ	
	14	13	12	11	10	9	8	7	4	3			0
		RESERVED		PWEN		PROM WIDTH		PROM WRITE WS		PROM READ WS			

31	RESERVED
30	PROM area bus ready enable (PBRDY) - Enables bus ready (BRDYN) signalling for the PROM area. Reset to ‘0’.
29	Asynchronous bus ready (ABRDY) - Enables asynchronous bus ready.

Table 220. Memory configuration register 1.

28 : 27	I/O bus width (IOBUSW) - Sets the data width of the I/O area (“00”=8, “01”=16, “10” =32).
26	I/O bus ready enable (IBRDY) - Enables bus ready (BRDYN) signalling for the I/O area. Reset to ‘0’.
25	Bus error enable (BEXCN) - Enables bus error signalling for all areas. Reset to ‘0’.
24	RESERVED
23 : 20	I/O waitstates (IO WAITSTATES) - Sets the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
19	I/O enable (IOEN) - Enables accesses to the memory bus I/O area.
18	RESERVED
17: 14	PROM bank size (ROMBANKSZ) - Returns current PROM bank size when read. “0000” is a special case and corresponds to a bank size of 256 MB. All other values give the bank size in binary steps: “0001”=16kB, “0010”=32kB, ... , “1111”=256 MB. For value “0000” or “1111” only two chip selects are available. For other values, two chip select signals are available for fixed bank sizes. For other values, four chip select signals are available for programmable bank sizes. Programmable bank sizes can be changed by writing to this register field. The written values correspond to the bank sizes and number of chip-selects as above. Reset to “0000” when programmable. Programmable ROMBANKSZ is only available when romasel VHDL generic is 0. For other values this is a read-only register field containing the fixed bank size value.
13:12	RESERVED
11	PROM write enable (PWEN) - Enables write cycles to the PROM area.
10	RESERVED
9 : 8	PROM width (PROM WIDTH) - Sets the data width of the PROM area (“00”=8, “01”=16, “10”=32).
7 : 4	PROM write waitstates (PROM WRITE WS) - Sets the number of wait states for PROM write cycles (“0000”=0, “0001”=2, “0010”=4,..., “1111”=30).
3 : 0	PROM read waitstates (PROM READ WS) - Sets the number of wait states for PROM read cycles (“0000”=0, “0001”=2, “0010”=4,...,”1111”=30). Reset to “1111”.

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

30.15.2 Memory configuration register 2 (MCFG2)

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

Table 221. Memory configuration register 2.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SDRF	TRP	SDRAM TRFC			TCAS	SDRAM BANKSZ			SDRAM COLSZ		SDRAM CMD		D64	SDPB	
15	14	13	12			9	8	7	6	5	4	3	2	1	0
	SE	SI	RAM BANK SIZE					RBRDY	RMW	RAM WIDTH		RAM WRITE WS		RAM READ WS	

31	SDRAM refresh (SDRF) - Enables SDRAM refresh.
30	SRAM TRP parameter (TRP) - t_{RP} will be equal to 2 or 3 system clocks (0/1).
29 : 27	SDRAM TRFC parameter (SDRAM TRFC) - t_{RFC} will be equal to 3+field-value system clocks.
26	SDRAM TCAS parameter (TCAS) - Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (t_{RCD}).
25 : 23	SDRAM bank size (SDRAM BANKSZ) - Sets the bank size for SDRAM chip selects (“000”=4 Mbyte, “001”=8 Mbyte, “010”=16 Mbyte.... “111”=512 Mbyte).
22 : 21	SDRAM column size (SDRAM COLSZ) - “00”=256, “01”=512, “10”=1024, “11”=4096 when bit 25:23=”111” 2048 otherwise.

Table 221. Memory configuration register 2.

20 : 19	SDRAM command (SDRAM CMD) - Writing a non-zero value will generate a SDRAM command. “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after the command has been executed.
18	64-bit SDRAM data bus (D64) - Reads ‘1’ if the memory controller is configured for 64-bit SDRAM data bus width, ‘0’ otherwise. Read-only.
17	SDRAM Page Burst (SDPB) - SDRAM programmed for page bursts on read when set, else programmed for line burst lengths of 8 on read. Programmable when pageburst VHDL generic is 2, else read-only.
16 : 15	RESERVED
14	SDRAM enable (SE) - Enables the SDRAM controller and disables fifth SRAM bank (RAMSN[4]).
13	SRAM disable (SI) - Disables accesses to SRAM bank if bit 14 (SE) is set to ‘1’.
12 : 9	RAM bank size (RAM BANK SIZE) - Sets the size of each RAM bank (“0000”=8 kbyte, “0001”=16 kbyte, ..., “1111”=256 Mbyte).
8	RESERVED
7	RAM bus ready enable (RBRDY) - Enables bus ready signalling for the RAM area.
6	Read-modify-write enable (RMW) - Enables read-modify-write cycles for sub-word writes to 16- bit 32-bit areas with common write strobe (no byte write strobe).
5 : 4	RAM width (RAM WIDTH) - Sets the data width of the RAM area (“00”=8, “01”=16, “1X”=32).
3 : 2	RAM write waitstates (RAM WRITE WS) - Sets the number of wait states for RAM write cycles (“00”=0, “01”=1, “10”=2, “11”=3).
1 : 0	RAM read waitstates (RAM READ WS) - Sets the number of wait states for RAM read cycles (“00”=0, “01”=1, “10”=2, “11”=3).

30.15.3 Memory configuration register 3 (MCFG3)

MCFG3 contains the reload value for the SDRAM refresh counter and to control and monitor the memory EDAC. It also contains the configuration of the register file EDAC.

Table 222. Memory configuration register 3.

31 28	27	26
RESERVED	ME	SDRAM REFRESH COUNTER

12	11	10	9	8	7 0
	WB	RB	RE	PE	TCB

31 : 28	RESERVED
27	Memory EDAC (ME) - Indicates if memory EDAC is present.
26 : 12	SDRAM refresh counter reload value (SDRAM REFRESH COUNTER)
11	EDAC diagnostic write bypass (WB) - Enables EDAC write bypass.
10	EDAC diagnostic read bypass (RB) - Enables EDAC read bypass.
9	RAM EDAC enable (RE) - Enable EDAC checking of the RAM area (including SDRAM).
8	PROM EDAC enable (PE) - Enable EDAC checking of the PROM area. Ar reset, this bit is initialized with the value of MEMI.EDAC.
7 : 0	Test checkbits (TCB) - This field replaces the normal checkbits during write cycles when WB is set. It is also loaded with the memory checkbits during read cycles when RB is set.

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

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

30.16 Vendor and device identifiers

The core has vendor identifier 0x01 (GAISLER) and device identifier 0x05F. For description of vendor and device identifiers, see GRLIB IP Library User’s Manual.

30.17 Configuration options

Table 223 shows the configuration options of the core (VHDL generics).

*Table 223.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0x1FFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#E00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space. Default I/O area is 0x20000000 - 0x2FFFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#E00#
ramaddr	ADDR field of the AHB BAR2 defining RAM address space. Default RAM area is 0x40000000-0x7FFFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR2 defining RAM address space.	0 -16#FFF#	16#C00#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
wprot	RAM write protection.	0 - 1	0
invclk	unused	N/A	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
romasel	Sets the PROM bank size. 0 selects a programmable mode where the rombanksz field in the MCFG1 register sets the bank size. See the description of the MCFG1 register for more details. Values 1 - 14 sets the size in binary steps (1 = 16 kB, 2 = 32 kB, ..., 14=128 MB). Four chip-selects are available for these values. 15 sets the bank size to 256 MB with two chip-selects. Values 16 - 28 sets the bank size in binary steps (16 = 64 kB, 17 = 128 kB, 28 = 256 MB). Two chip-selects are available for this range. The selected bank size is readable from the rombanksz field in the MCFG1 register for the non-programmable modes.	0 - 28	28
sdrasel	log2(RAM address space size) - 1. E.g if size of the RAM address space is 0x40000000 sdrasel is log2(2^30)-1= 29.	0 - 31	29
srbanks	Number of SRAM banks.	0 - 5	4
ram8	Enable 8-bit PROM, SRAM and I/O access.	0 - 1	0
ram16	Enable 16-bit PROM, SRAM and I/O access.	0 - 1	0
sden	Enable SDRAM controller.	0 - 1	0
sepbus	SDRAM is located on separate bus.	0 - 1	1
sdbits	32 or 64 -bit SDRAM data bus.	32, 64	32
oepol	Select polarity of drive signals for data pads. 0 = active low, 1 = active high.	0 - 1	0
edac	Enable EDAC	0 - 1	0
sdlsb	Select least significant bit of the address bus that is connected to SDRAM.	-	2
syncrst	Choose between synchronous and asynchronous reset for chip-select, oen and drive signals.	0 - 1	0
pageburst	Line burst read of length 8 when 0, page burst read when 1, programmable read burst type when 2.	0-2	0
scantest	Enable scan test support	0 - 1	0

30.18 Scan support

Scan support is enabled by setting the SCANTEST generic to 1. When enabled, the asynchronous reset of any flip-flop will be connected to AHBI.testrst during when AHBI.testen = ‘1’.

30.19 Signal descriptions

Table 224 shows the interface signals of the core (VHDL ports).

*Table 224.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
MEMI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	CBI[7:0]	Input	EDAC checkbits	High
	WRN[3:0]	Input	SRAM write enable feedback signal	Low
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	High
	EDAC	Input	The reset value for the PROM EDAC enable bit	High
	SD[31:0]	Input	SDRAM separate data bus	High

*Table 224.*Signal descriptions

Signal name	Field	Type	Function	Active
MEMO	ADDRESS[27:0]	Output	Memory address	High
	CBO[7:0]	Output	EDAC Checkbit	
	DATA[31:0]	Output	Memory data	-
	SDDATA[63:0]	Output	Sdram memory data	-
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Local I/O select	Low
	ROMSN[3:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0]. Any WRN[ ] signal can be used for CB[ ].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0]. Any MBEN[ ] signal can be used for CB[ ].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low/High
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signals.	Low/High
	SVBDRIVE[63:0]	Output	Vectored I/O-pad drive signals for separate sdram bus.	Low/High
	READ	Output	Read strobe	High
	SA[14:0]	Output	SDRAM separate address bus	High
	CE	Output	Single error detected	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
WPROT	WPROTHIT	Input	Unused	-

*Table 224.*Signal descriptions

Signal name	Field	Type	Function	Active
SDO	SDCASN	Output	SDRAM column address strobe	Low
	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDDQM[7:0]	Output	SDRAM data mask: SDDQM[7] corresponds to SD[63:56], SDDQM[6] corresponds to SD[55:48], SDDQM[5] corresponds to SD[47:40], SDDQM[4] corresponds to SD[39:32], SDDQM[3] corresponds to SD[31:24], SDDQM[2] corresponds to SD[23:16], SDDQM[1] corresponds to SD[15:8], SDDQM[0] corresponds to SD[7:0]. Any SDDQM[ ] signal can be used for CB[ ].	Low
	SDRASN	Output	SDRAM row address strobe	Low
	SDWEN	Output	SDRAM write enable	Low

* see GRLIB IP Library User’s Manual

30.20 Library dependencies

Table 225 shows libraries used when instantiating the core (VHDL libraries).

*Table 225.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals	Memory bus signals definitions
		Components	FTMCTRL component

30.21 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0x1FFFFFFF, I/O-area is 0x20000000-0x3FFFFFFF and RAM area is 0x40000000 - 0x7FFFFFFF. SDRAM controller is enabled. SDRAM clock is synchronized with system clock by clock generator.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(3 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdram_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  -- Memory controller
  ftmctrl0 : ftmctrl generic map (srbanks => 1, sden => 1, edac => 1)
    port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wprot, sdo);

  -- memory controller inputs not used in this configuration
  memi.brdyn <= ’1’; memi.bexcn <= ’1’; memi.wrn <= "1111";
  memi.sd <= sd;
  -- prom width at reset
  memi.bwidth <= "10";

  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => memi.data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= "1111" & memo.ramoen(0);
  sa <= memo.sa;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;
end;
```

31 FTSDCTRL - 32/64-bit PC133 SDRAM Controller with EDAC

31.1 Overview

The fault tolerant SDRAM memory interface handles PC133 SDRAM compatible memory devices attached to a 32- or 64-bit wide data bus. The interface acts as a slave on the AHB bus where it occupies configurable amount of address space for SDRAM access. An optional Error Detection And Correction Unit (EDAC) logic (only for the 32 - bit bus) corrects one bit error and detects two bit errors.

The SDRAM controller function is programmed by means of register(s) mapped into AHB I/O address space. Chip-select decoding is done for two SDRAM banks.

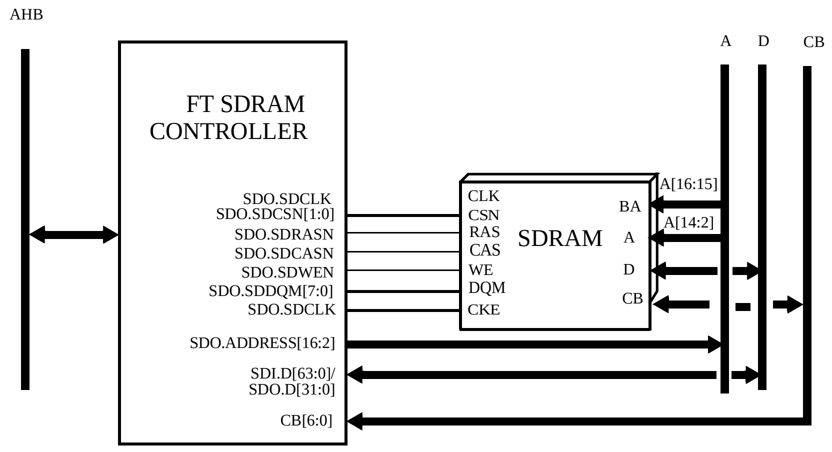


Figure 107. FT SDRAM memory controller connected to AMBA bus and SDRAM

31.2 Operation

31.2.1 General

Synchronous Dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. The controller supports 64, 256 and 512 Mbyte devices 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 the configuration register SDCFG. A second register, ECFG, is available for configuring the EDAC functions. SDRAM banks data bus width is configurable between 32 and 64 bits.

31.2.2 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 8x 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.

31.2.3 Configurable SDRAM timing parameters

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

*Table 226.*SDRAM programmable timing parameters

Function	Parameter	range	unit
CAS latency, RAS/CAS delay	t_{CAS}, t_{RCD}	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

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

31.2.4 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 SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (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 SDCFG register.

31.2.5 SDRAM commands

The controller can issue three SDRAM commands by writing to the SDRAM command field in SDCFG: PRE-CHARGE, AUTO-REFRESH and LOAD-MODE-REG (LMR). If the LMR command is issued, the CAS delay as programmed in SDCFG 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.

31.2.6 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. Note that only word bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and halfwords but they cannot be used.

31.2.7 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. As in the read case, only word bursts are supported.

31.2.8 Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13], and the data bus to SD[31:0] or SD[63:0] if 64-bit data bus is used.

31.2.9 Data bus

Data bus width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used.

31.2.10 Clocking

The SDRAM clock typically requires special synchronisation at layout level. For Virtex targets, GR Clock Generator can be configured to produce a properly synchronised SDRAM clock. For other FPGA targets, the GR Clock Generator can produce an inverted clock.

31.2.11 EDAC

The controller optionally contains Error Detection And Correction (EDAC) logic, using a BCH(32, 7) code. It is capable of correcting one bit error and detecting two bit errors. The EDAC logic does not add any additional waitstates during normal operation. Detected errors will cause additional waitstates for correction (single errors) or error reporting (multiple errors). Single errors are automatically corrected and generally not visible externally unless explicitly checked.

This checking is done by monitoring the ce signal and single error counter. This counter holds the number of detected single errors. The ce signal is asserted one clock cycle when a single error is detected and should be connected to the AHB status register. This module stores the AHB status of the instruction causing the single error and generates interrupts (see the AHB status register documentation for more information).

The EDAC functionality can be enabled/disabled during run-time from the ECFG register (and the logic can also be completely removed during synthesis with VHDL generics. The ECFG register also contains control bits and checkbit fields for diagnostic reads. These diagnostic functions are used for testing the EDAC functions on-chip and allows one to store arbitrary checkbits with each written word. Checkbits read from memory can also be controlled.

64-bit bus support is not provided when EDAC is enabled. Thus, the sd64 and edacen VHDL generics should never be set to one at the same time.

The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
C̅B̅2̅ = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
C̅B̅3̅ = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

31.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1.

If EDAC is enabled through the use of the edacen VHDL generic, an EDAC configuration register will be available.

*Table 227.*FT SDRAM controller registers

AHB address offset	Register
0x0	SDRAM Configuration register
0x4	EDAC Configuration register

31.3.1 SDRAM configuration register (SDCFG)

SDRAM configuration register is used to control the timing of the SDRAM.

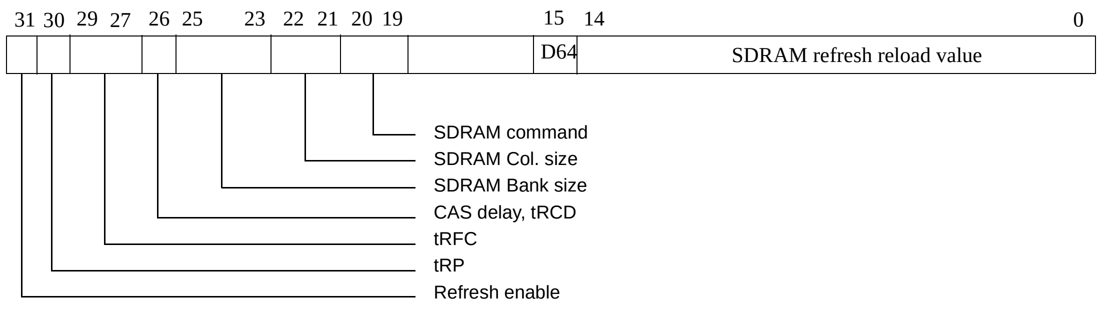


Figure 108. SDRAM configuration register

[14:0]: The period between each AUTO-REFRESH command - Calculated as follows:$t_{REFRESH}$ = ((reload value) + 1) / SYSCLK

[15]: 64-bit data bus (D64) - Reads ‘1’ if memory controller is configured for 64-bit data bus, otherwise ‘0’. Read-only.

[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 selects: “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. Also sets RAS/CAS delay (tRCD).

[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.

31.3.2 EDAC Configuration register (ECFG)

The EDAC configuration register controls the EDAC functions of the SDRAM controller during run time.

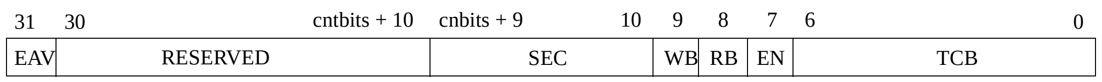


Figure 109. EDAC configuration register

[6:0] TCB : Test checkbits. These bits are written as checkbits into memory during a write operation when the WB bit in the ECFG register is set. Checkbits read from memory during a read operation are written to this field when the RB bit is set.

[7] EN : EDAC enable. Run time enable/disable of the EDAC functions. If EDAC is disabled no error detection will be done during reads and subword writes. Checkbits will still be written to memory during write operations.

[8] RB : Read bypass. Store the checkbits read from memory during a read operation into the TCB field.

[9] WB : Write bypass. Write the TCB field as checkbits into memory for all write operations.

[cntbits + 9:10] SEC : Single error counter. This field is available when the errcnt VHDL generic is set to one during synthesis. It increments each time a single error is detected. It saturates when the maximum value is reached. The maximum value is the largest number representable in the number of bits used, which in turn is determined by the cntbits VHDL generic. Each bit in the counter can be reset by writing a one to it.

[30:cntbits + 10] Reserved.

[31] EAV : EDAC available. This bit is always one if the SDRAM controller contains EDAC.

31.4 Vendor and device identifiers

The module has vendor identifier 0x01 (Gaisler Research) and device identifier 0x055. For a description of vendor and device identifiers see GRLIB IP Library User’s Manual.

31.5 Configuration options

Table 228 shows the configuration options of the core (VHDL generics).

*Table 228.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
pwron	Enable SDRAM at power-on.	0 - 1	0
sdbits	32 or 64 -bit data bus width.	32, 64	32
edacen	EDAC enable. If set to one, EDAC logic will be included in the synthesized design. An EDAC configuration register will also be available.	0 - 1	0
errcnt	Include an single error counter which is accessible from the EDAC configuration register.	0 - 1	0
cntbits	Number of bits used in the single error counter	1 - 8	1

31.6 Signal descriptions

Table 229 shows the interface signals of the core (VHDL ports).

*Table 229.*Signals declarations

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	-
	CB[7:0]	Input	Checkbits	-
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0]. Any DQM[ ] signal can be used for CB[ ].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low
	ADDRESS[16:2]	Output	SDRAM address	-
	DATA[31:0]	Output	SDRAM data	-
	CB[7:0]	Output	Checkbits	-
CE	N/A	Output	Correctable Error	High

* see GRLIB IP Library User’s Manual

31.7 Library dependencies

Table 5 shows libraries used when instantiating the core (VHDL libraries).

*Table 230.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

31.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the FT SDRAM controller. The external SDRAM bus is defined in the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator. It is also shown how the correctable error (CE) signal is connected to the ahb status register. It is not mandatory to connect this signal. In this example, 3 units can be connected to the status register.

The SDRAM controller decodes SDRAM area: 0x60000000 - 0x6FFFFFFF. SDRAM Configuration and EDAC configuration registers are mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
    ... -- other signals

-- sdram memory bus
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0); -- optional sdram data
    cb       : inout std_logic_vector(7 downto 0) --EDAC checkbits
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect SDRAM controller and SDRAM memory bus
  signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic; -- system clock and reset
signal ce : std_logic_vector(0 to 2); --correctable error signal vector

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- AMBA Components are defined here ...
  ...

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 3)
    port map(rstn, clkm, ahbmi, ahbsi, ce, apbi, apbo(13));

  -- SDRAM controller
  sdc : ftsdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, fast => 0, pwron => 1, invclk => 0, edacen => 1, errcnt => 1,
    cntbits => 4)
    port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo, ce(0));

  -- input signals
  sdi.data(31 downto 0) <= sd(31 downto 0);

  -- connect SDRAM controller outputs to entity output signals
  sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
  sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
  sddqm <= sdo.dqm;

  -- I/O pads driving data bus signals
  sd_pad : iopadv generic map (width => 32)
      port map (sd(31 downto 0), sdo.data, sdo.bdrive, sdi.data(31 downto 0));

  -- I/O pads driving checkbit signals
  cb_pad : iopadv generic map (width => 8)
      port map (cb, sdo.cb, sdo.bdrive, sdi.cb);


end;
```

32 FTSRCTRL - Fault Tolerant 32-bit PROM/SRAM/IO Controller

32.1 Overview

The fault tolerant 32-bit PROM/SRAM memory interface uses a common 32-bit memory bus to interface PROM, SRAM and I/O devices. Support for 8-bit PROM banks can also be separately enabled. In addition it also provides an Error Detection And Correction Unit (EDAC), correcting one and detecting two errors. Configuration of the memory controller functions is performed through the APB bus interface.

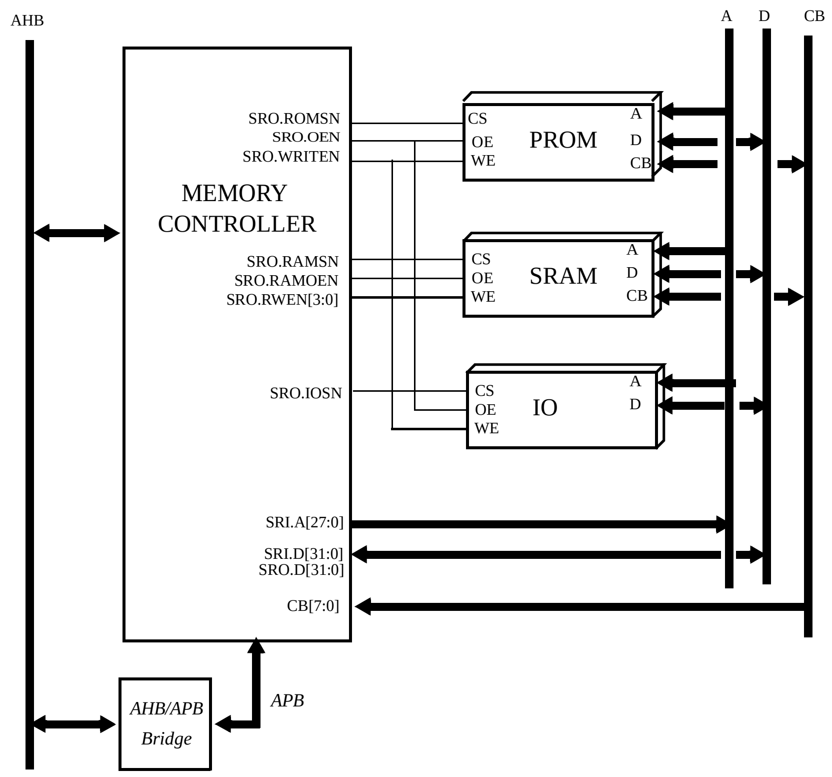


Figure 110. 32-bit FT PROM/SRAM/IO controller

32.2 Operation

The controller is configured through VHDL generics to decode three address ranges: PROM, SRAM and I/O area. By default the PROM area is mapped into address range 0x0 - 0x00FFFFFF, the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM and PROM can have up to 8 chip select signals. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WREN). If the SRAM uses a common write enable signal the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses. Number of waitstates is separately configurable for the three address ranges.

The EDAC function is optional, and can be enabled with the *edacen* VHDL generic. The configuration of the EDAC is done through a configuration register accessed from the APB bus. During nominal operation, the EDAC checksum is generated and checked automatically. Single errors are corrected without generating any indication of this condition in the bus response. If a multiple error is detected, a two cycle error response is given on the AHB bus.

Single errors can be monitored in two ways:

- by monitoring the CE signal which is asserted for one cycle each time a single error is detected.
- by checking the single error counter which is accessed from the MCFG3 configuration register.

The CE signal can be connected to the AHB status register which stores information of the AHB instruction causing the error and also generates interrupts. See the AHB status register documentation for more information. When EDAC is enabled, one extra latency cycle is generated during reads and subword writes.

The EDAC function can be enabled for SRAM and PROM area accesses, but not for I/O area accesses. For the SRAM area, the EDAC functionality is only supported for accessing 32-bit wide SRAM banks. For the PROM area, the EDAC functionality is supported for accessing 32-bit wide PROM banks, as well as for read accesses to 8-bit wide PROM banks.

The equations below show how the EDAC checkbits are generated:

```
CB0 = D0 ^ D4 ^ D6 ^ D7 ^ D8 ^ D9 ^ D11 ^ D14 ^ D17 ^ D18 ^ D19 ^ D21 ^ D26 ^ D28 ^ D29 ^ D31
CB1 = D0 ^ D1 ^ D2 ^ D4 ^ D6 ^ D8 ^ D10 ^ D12 ^ D16 ^ D17 ^ D18 ^ D20 ^ D22 ^ D24 ^ D26 ^ D28
C̅B̅2̅ = D0 ^ D3 ^ D4 ^ D7 ^ D9 ^ D10 ^ D13 ^ D15 ^ D16 ^ D19 ^ D20 ^ D23 ^ D25 ^ D26 ^ D29 ^ D31
C̅B̅3̅ = D0 ^ D1 ^ D5 ^ D6 ^ D7 ^ D11 ^ D12 ^ D13 ^ D16 ^ D17 ^ D21 ^ D22 ^ D23 ^ D27 ^ D28 ^ D29
CB4 = D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D14 ^ D15 ^ D18 ^ D19 ^ D20 ^ D21 ^ D22 ^ D23 ^ D30 ^ D31
CB5 = D8 ^ D9 ^ D10 ^ D11 ^ D12 ^ D13 ^ D14 ^ D15 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
CB6 = D0 ^ D1 ^ D2 ^ D3 ^ D4 ^ D5 ^ D6 ^ D7 ^ D24 ^ D25 ^ D26 ^ D27 ^ D28 ^ D29 ^ D30 ^ D31
```

32.2.1 8-bit PROM access

The FTSRCTRL controller can be configured to access an 8-bit wide PROM. The data bus of the external PROM should be connected to the upper byte of the 32-bit data bus, i.e. D[31:24]. The 8-bit mode is enabled with the prom8en VHDL generic. When enabled, read accesses to the PROM area will be done in four-byte bursts for all 32-, 16- and 8-bit AMBA AHB accesses. The whole 32-bit word is then output on the AHB data bus, allowing the master to chose the bytes needed (big-endian).

Writes should be done one byte at a time. For correct word aligned 32-bit word write accesses, the byte should always be driven on bits 31 to 24 on the AHB data bus. For non-aligned 32-bit word write accesses, the byte should be driven on the bits of the AHB data bus that correspond to the byte address (big-endian). For correct half-word aligned 16-bit half-word write accesses, the byte should always be driven on bits 31 to 24, or 15 to 8, on the AHB data bus. For non-aligned 16-bit half-word write accesses, the byte should be driven on the bits of the AHB data bus that correspond to the byte address (big-endian). For 8-bit word write accesses the byte should always be driven on the AHB data bus bits that corresponds to the byte address (big-endian). To summarize, all legal AMBA AHB write accesses are supported according to the AMBA standard, additional illegal accesses are supported as described above, and it is always the addressed byte that is output.

It is possible to dynamically switch between 8- and 32-bit PROM mode by writing to the RBW field of the MCFG1 register. The BWIDTH[1:0] input signal determines the reset value of this RBW register field. When RBW is “00” then 8-bit mode is selected. If RBW is “10” then 32-bit mode is selected. Other RBW values are reserved for future use. SRAM access is not affected by the 8-bit PROM mode.

It is also possible to use the EDAC in the 8-bit PROM mode, configured by the edacen VHDL generic, and enabled via the MCFG3 register. Read accesses to the 8-bit PROM area will be done in five-byte bursts for all 32-, 16- and 8-bit AMBA AHB accesses. After a potential correction, the whole 32-bit word is output on the AHB data bus, allowing the master to chose the bytes needed (big-endian). EDAC support is not provided for write accesses, they are instead performed in the same way as without the EDAC enabled. The checksum byte must be written by the user into the correct byte address location.

The fifth byte corresponds to the EDAC checksum and is located in the upper part of the effective memory area, as explained in detail in the definition of the MCFG1 memory configuration register. The EDAC checksums are located in the upper quarter of what is defined as available EDAC area by means of the EBSZ field and the ROMBSZ field or rombanksz VHDL generic. When set to 0, the size of the available EDAC area is defined as the PROM bank size. When set to 1, as twice the PROM bank size. When set to 2, as four times the PROM bank size. And when set to 3, as eight times the PROM bank size. For any other value than 0, the use of multiple PROM banks is required.

Example, if ROMBSZ=10 and EBSZ=1, the EDAC area is 8kB*2^ROMBSZ*2^EBSZ= 16MB=0x01000000. The checksum byte for the first word located at address 0x00000000 to 0x00000003 is located at 0x00C00000. The checksum byte for the second word located at address 0x00000004 to 0x00000007 is located at 0x00C00001, and so on. Since EBSZ=1, two PROM banks are required for implementing the EDAC area, each bank with size 8MB=0x00800000.

32.2.2 Access errors

The active low Bus Exception signal (BEXCN) can be used to signal access errors. It is enabled by setting the BEXCEN bit in MCFG1 and is active for all types of accesses to all areas (PROM, SRAM and I/O). The BEXCN signal is sampled on the same cycle as read data is sampled. For writes it is sampled on the last rising edge before writen/rwen is de-asserted (writen and rwen are clocked on the falling edge). When a bus exception is detected an error response will be generated for the access.

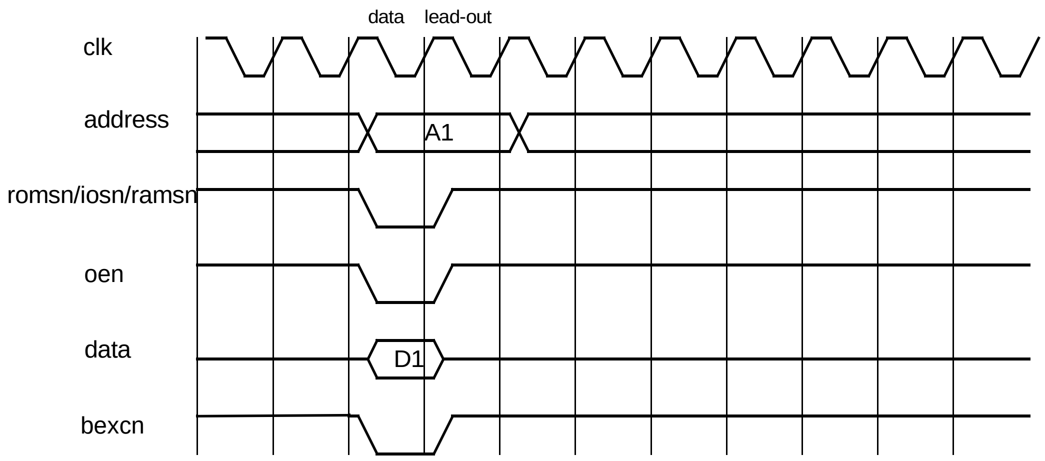


Figure 111. Read cycle with BEXCN.

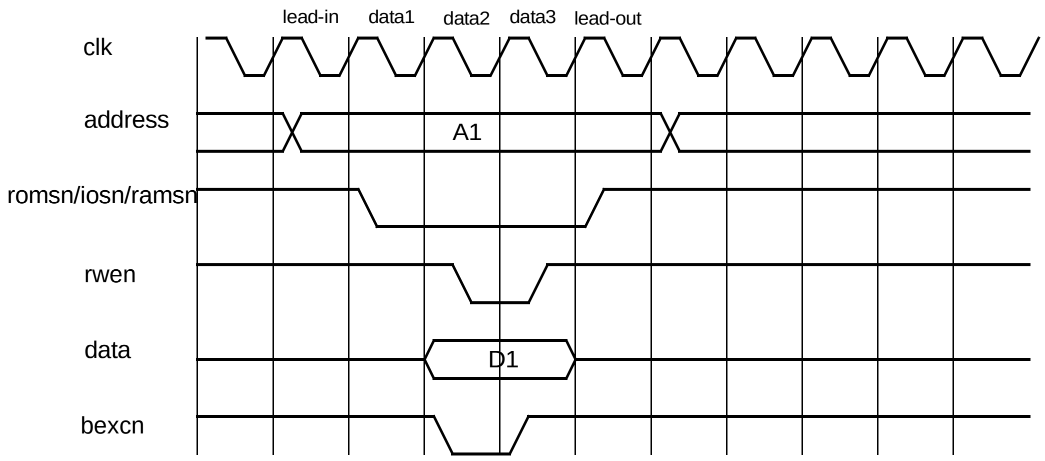


Figure 112. Write cycle with BEXCN.

32.2.3 Using bus ready signalling

The Bus Ready (BRDYN) signal can be used to add waitstates to I/O-area accesses, covering the complete memory area and both read and write accesses. It is enabled by setting the Bus Ready

Enable (BRDYEN) bit in the MCFG1 register. An access will have at least the amount of waitstates set with the VHDL generic or through the register, but will be further stretched until BRDYN is asserted. Additional waitstates can thus be inserted after the pre-set number of waitstates by de-asserting the BRDYN signal. BRDYN should be asserted in the cycle preceding the last one. It is recommended that BRDYN remains asserted until the IOSN signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall. Read accesses will have the same timing as when EDAC is enabled while write accesses will have the timing as for single accesses even if bursts are performed.

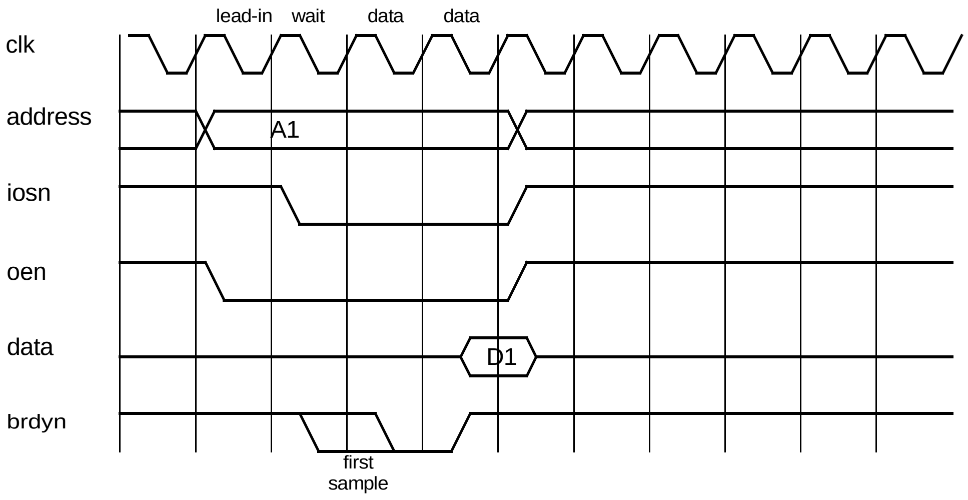


Figure 113. I/O READ cycle, programmed with 1 wait state, and with an extra data cycle added with BRDYN.

32.3 PROM/SRAM/IO waveforms

The internal and external waveforms of the interface are presented in the figures hereafter.

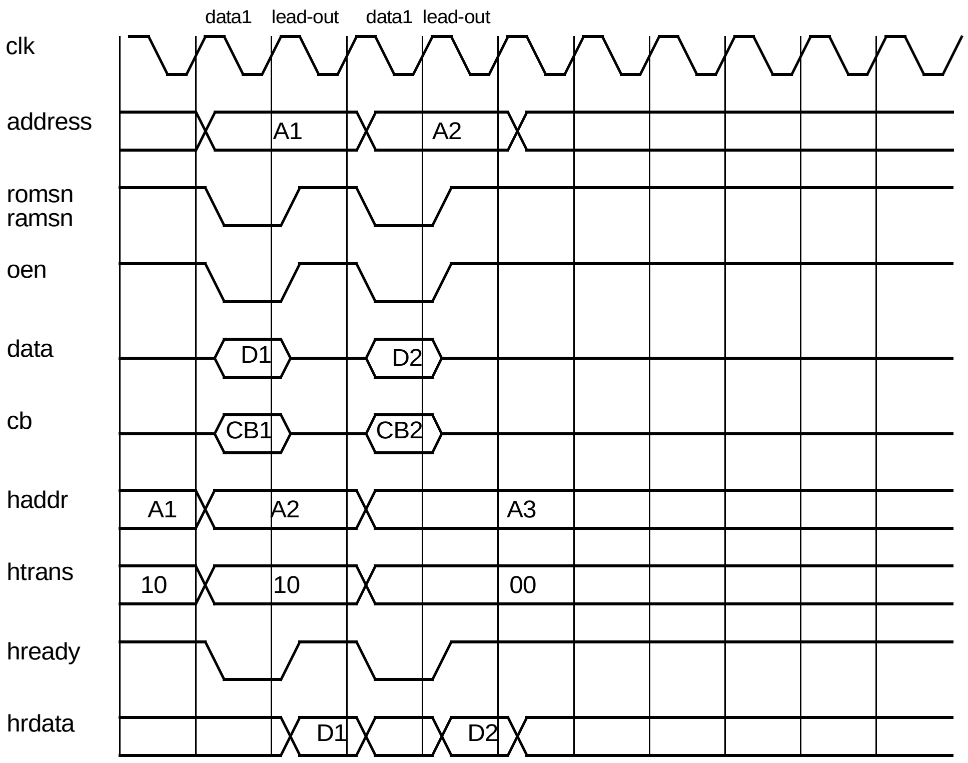


Figure 114. PROM/SRAM non-consecutive read cyclecs.

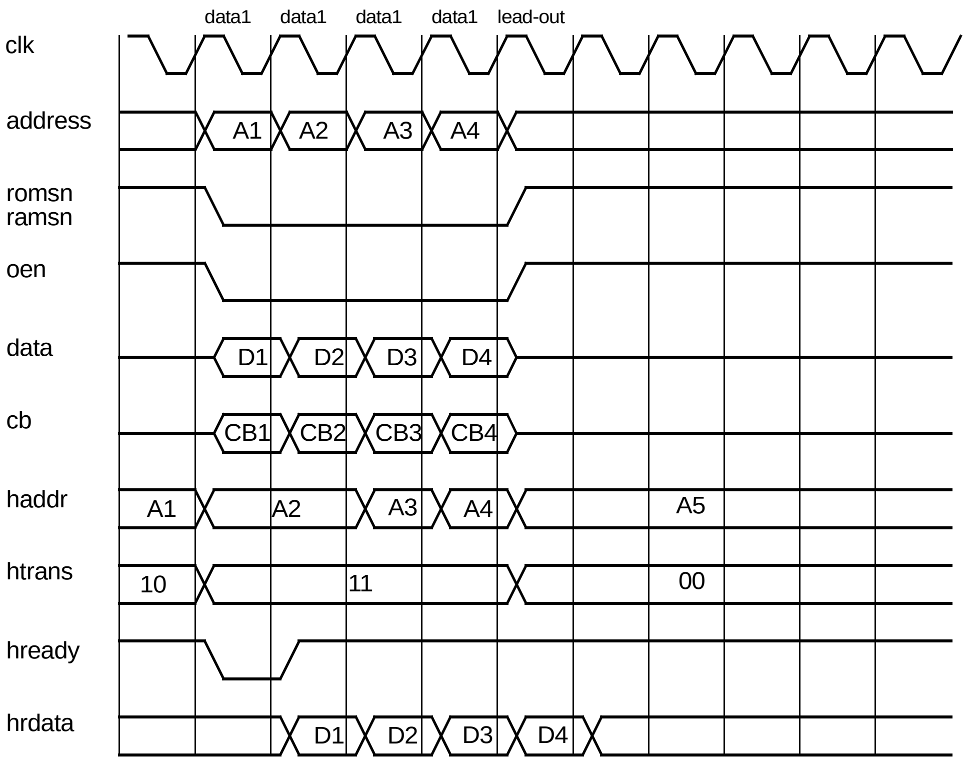


Figure 115. 32-bit PROM/SRAM sequential read access with 0 wait-states and EDAC disabled.

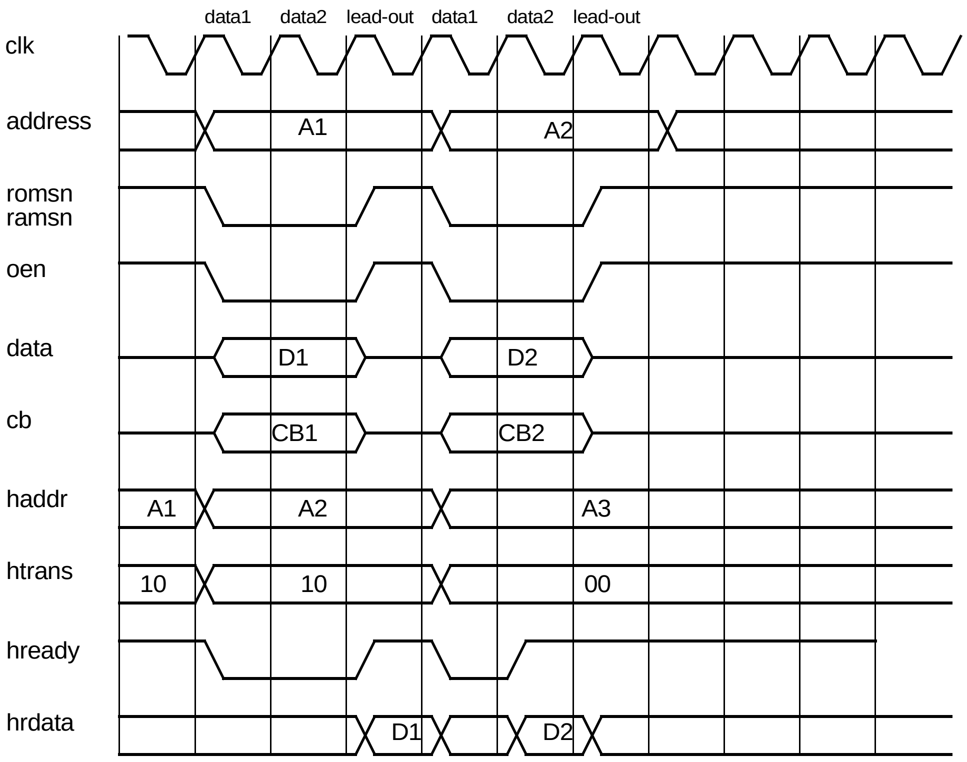


Figure 116. 32-bit PROM/SRAM non-sequential read access with 0 wait-states and EDAC enabled.

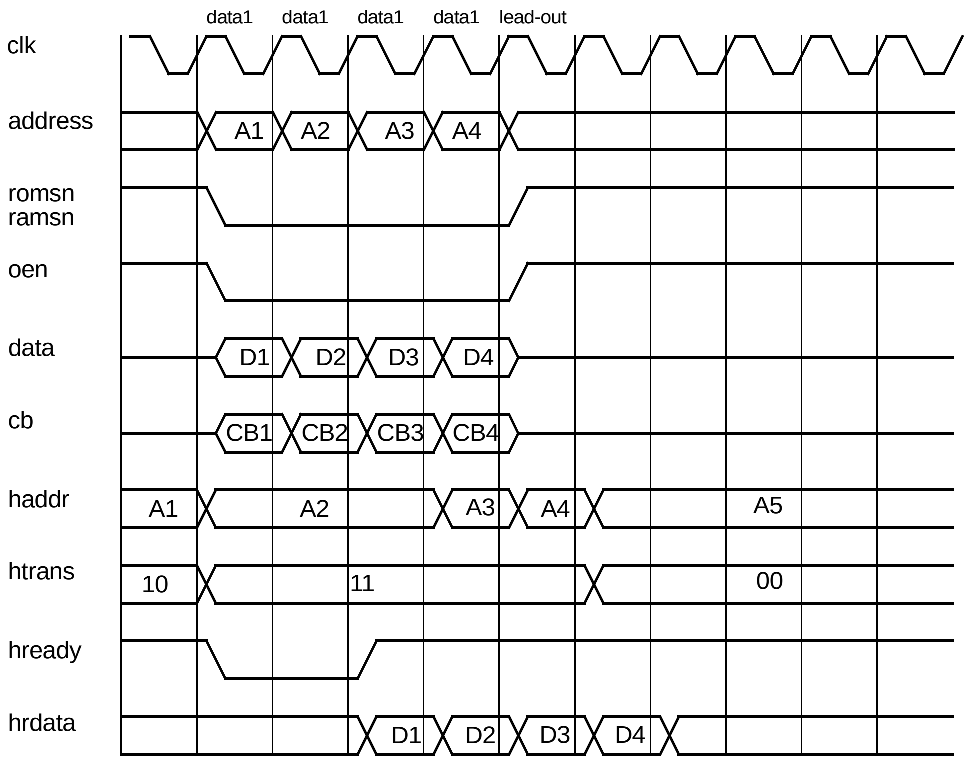


Figure 117. 32-bit PROM/SRAM sequential read access with 0 wait-states and EDAC enabled..

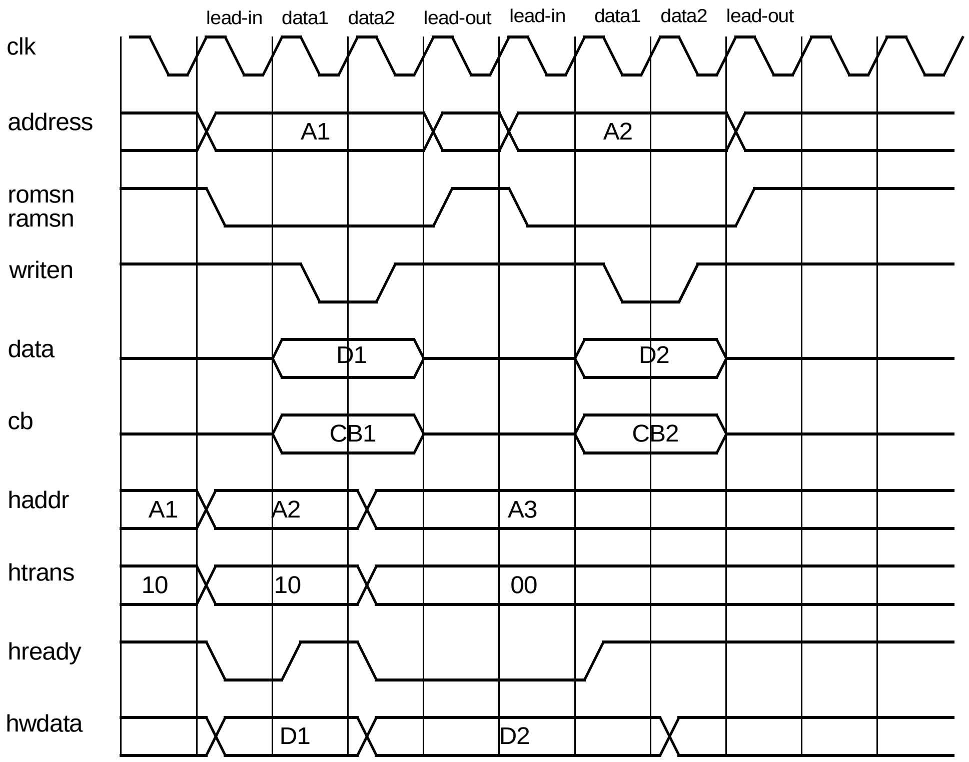


Figure 118. 32-bit PROM/SRAM non-sequential write access with 0 wait-states and EDAC disabled.

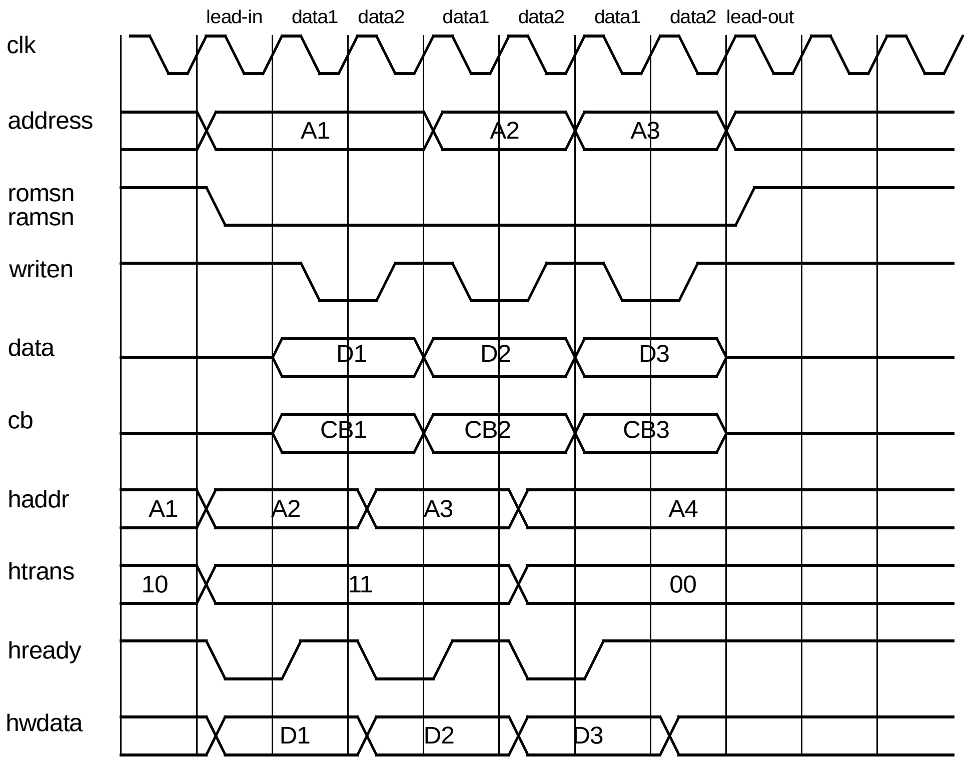


Figure 119. 32-bit PROM/SRAM sequential write access with 0 wait-states and EDAC disabled.

If waitstates are configured through the VHDL generics or registers, one extra data cycle will be inserted for each waitstate in both read and write cycles. The timing for write accesses is not affected when EDAC is enabled while one extra latency cycle is introduced for single access reads and at the beginning of read bursts.

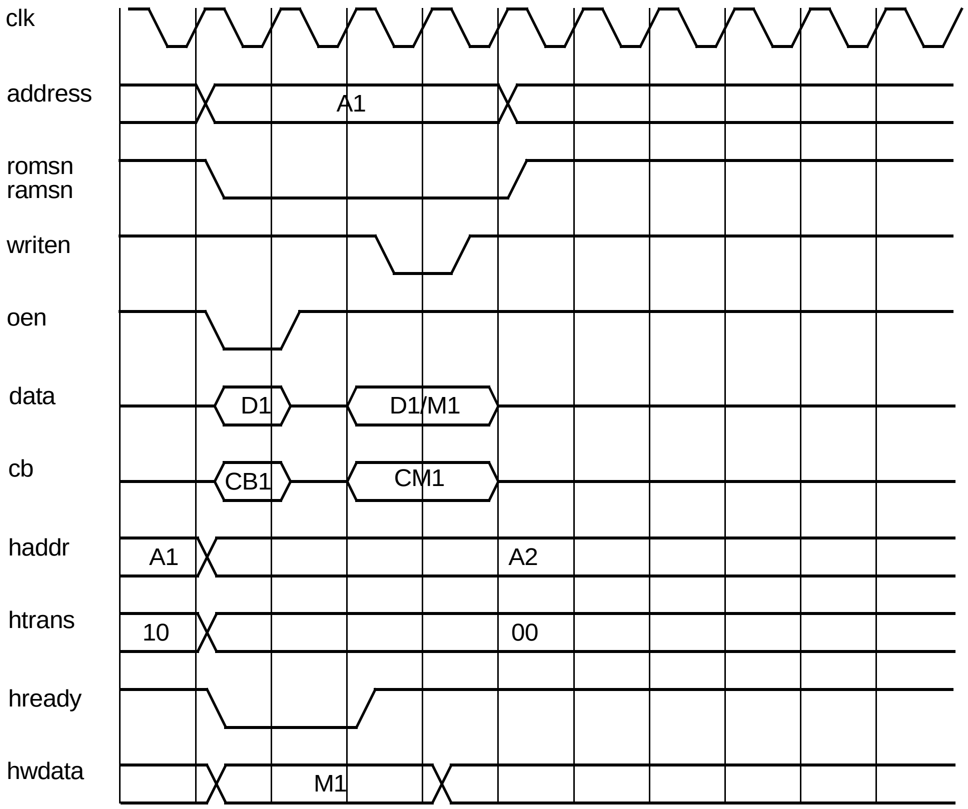


Figure 120. 32-bit PROM/SRAM rmw access with 0 wait-states and EDAC disabled.

Read-Modify-Write (RMW) accesses will have an additional waitstate inserted to accommodate decoding when EDAC is enabled.

I/O accesses are similar to PROM and SRAM accesses but a lead-in and lead-out cycle is always present.

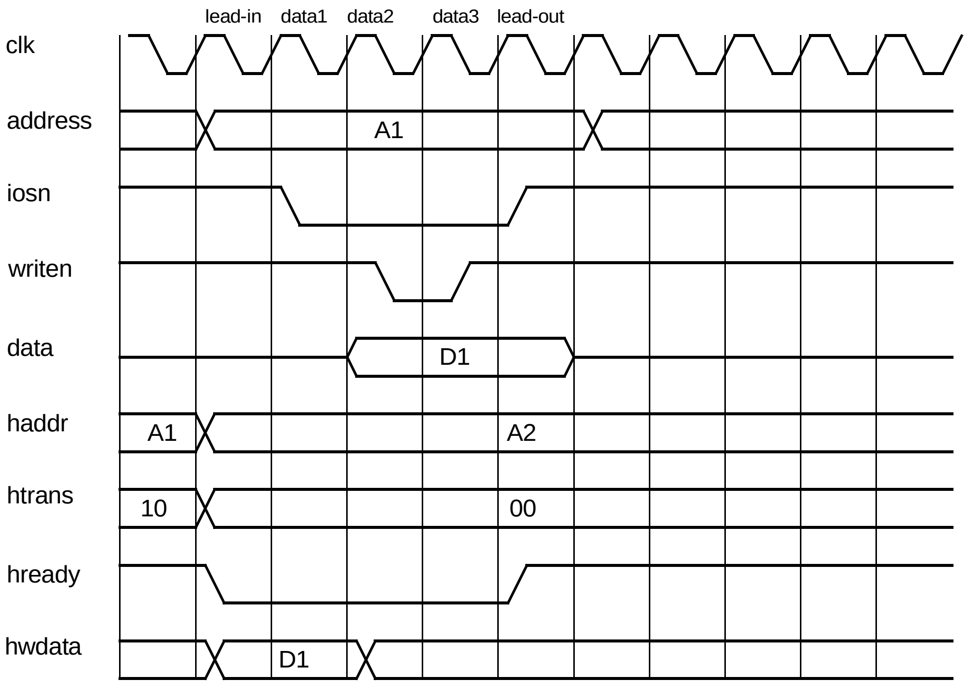


Figure 121. I/O write access with 0 wait-states.

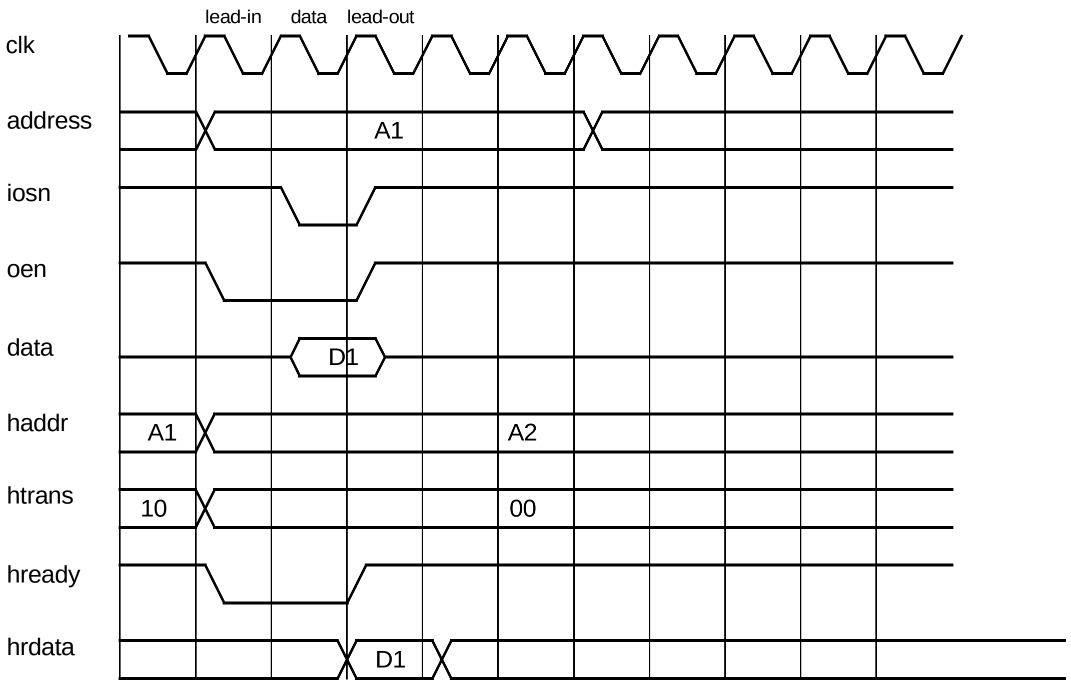


Figure 122. I/O read access with 0 wait-states

32.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 231.*FT PROM/SRAM/IO controller registers

APB Address offset	Register
0x0	Memory configuration register 1
0x4	Memory configuration register 2
0x8	Memory configuration register 3

Table 232. Memory configuration register 1.

31 27	26	25	24	23 20	19 18	17 14	13 12	11	10	9 8	7 4	3 0
RESERVED	BR	BE		IOWS		ROMBSZ	EBSZ	RW		RBW	RESERVED	ROMWS

31: 27	RESERVED
26	Bus ready enable (BR) - Enables the bus ready signal (BRDYN) for I/O-area.
25	Bus exception enable (BE) - Enables the bus exception signal (BEXCEN) for PROM, SRAM and I/O areas
24	RESERVED
23: 20	I/O wait states (IOWS) - Sets the number of waitstates for accesses to the I/O-area. Only available if the wsreg VHDL generic is set to one.
19: 18	RESERVED
17: 14	ROM bank size (ROMBSZ) - Sets the PROM bank size. Only available if the rombanksz VHDL generic is set to zero. Otherwise, the rombanksz VHDL generic sets the bank size and the value can be read from this field. 0 = 8 kB, 1 = 16 kB, ..., 15=256 MB
13: 12	EDAC bank size (EBSZ) - Sets the EDAC bank size for 8-bit PROM support. Only available if the rombanksz VHDL generic is zero, and edacen and prom8en VHDL generics are one. Otherwise, the value is fixed to 0. The resulting EDAC bank size is 2^EBSZ * 2^ROMBSZ * 8kB. Note that only the three lower quarters of the bank can be used for user data. The EDAC checksums are placed in the upper quarter of the bank.

Table 232. Memory configuration register 1.

11	ROM write enable (RW) - Enables writes to the PROM memory area. When disabled, writes to the PROM area will generate an ERROR response on the AHB bus.
10	RESERVED
9: 8	ROM data bus width (RBW) - Sets the PROM data bus width. “00” = 8-bit, “10” = 32-bit, others reserved.
7: 4	RESERVED
3: 0	ROM waitstates (ROMWS) - Sets the number of waitstates for accesses to the PROM area. Only available if the wsreg generic is set to one.

Table 233. Memory configuration register 2.

31 13	12 9	8 7	6	5 4 3 2	1 0
RESERVED	RAMBSZ		RW	RESERVED	RAMW

31: 13	RESERVED
12: 9	RAM bank size (RAMBSZ) - Sets the number of waitstates for accesses to the RAM area. Only available if the banksz VHDL generic is set to zero. Otherwise, the banksz VHDL generic sets the bank size and the value can be read from this field. 0 = 8 kB, 1 = 16 kB, ..., 15=256 MB
8: 7	RESERVED
6	Read-modify-write enable (RW) - Enables read-modify-write cycles for write accesses. Only available if the rmw VHDL generic is set to one.
5: 2	RESERVED
1: 0	RAM waitstates (RAMW) - Sets the number of waitstates for accesses to the RAM area. Only available if the wsreg VHDL generic is set to one.

Table 234. Memory configuration register 3.

31 20	19 12	11	10	9	8	7 6 5 4 3 2 1 0
RESERVED	SEC	WB	RB	SE	PE	TCB

31: 20	RESERVED
19: 12	Single error counter.(SEC) - This field increments each time a single error is detected until the maximum value that can be stored in the field is reached. Each bit can be reset by writing a one to it.
11	Write bypass (WB) - Enables EDAC write bypass. When enabled the TCB field will be used as checkbits in all write operations.
10	Read bypass (RB) - Enables EDAC read bypass. When enabled checkbits read from memory in all read operations will be stored in the TCB field.
9	SRAM EDAC enable (SE) - Enables EDAC for the SRAM area.
8	PROM EDAC enable (PE) - Enables EDAC for the PROM area. Reset value is taken from the input signal sri.edac.
7: 0	Test checkbits (TCB) - Used as checkbits in write operations when WB is activated and checkbits from read operations are stored here when RB is activated.

All the fields in MCFG3 register are available if the edacen VHDL generic is set to one except SEC field which also requires that the errcnt VHDL generic is set to one. The exact breakpoint between the SEC and RESERVED field depends on the cntbits generic. The breakpoint is 11+cntbits. The values shown in the table is for maximum cntbits value 8.

32.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x051. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

32.6 Configuration options

Table 231 shows the configuration options of the core (VHDL generics).

Table 235. Controller configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index.	1 - NAHBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default RAM area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 - 16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area.	0 - 15	0
romws	Number of waitstates during access to PROM area.	0 - 15	2
iows	Number of waitstates during access to IO area.	0 - 15	2
rmw	Enable read-modify-write cycles.	0 - 1	0
srbanks	Set the number of RAM banks.	1 - 8	1
banksz	Set the size of bank 1 - 4. 1 = 16 kB, ... , 15 = 256 MB. If set to zero, the bank size is set with the rambsz field in the MCFG2 register.	0 - 15	15
rombanks	Sets the number of PROM banks available.	1 - 8	1
rombanksz	Sets the size of one PROM bank. 1 = 16 kB, 2 = 32 kB, ..., 15 = 256 MB. If set to zero, the bank size is set with the rombsz field in the MCFG1 register.	0 - 15	15
rombankszdef	Sets the reset value of the rombsz register field in MCFG1 if available.	0 - 15	15
pindex	APB slave index.	1 - NAPBSLV-1	0
paddr	APB address.	1 - 16#FFF#	0
pmask	APB address mask.	1 - 16#FFF#	16#FFF#
edacen	EDAC enable. If set to one, EDAC logic is synthesized.	0 - 1	0
errcnt	If one, a single error counter is added.	0 - 1	0
cntbits	Number of bits in the single error counter.	1 - 8	1
wsreg	Enable programmable waitstate generation.	0 - 1	0
prom8en	Enable 8-bit PROM mode.	0 - 1	0
oepol	Select polarity of output enable signals. 0 = active low, 1 = active high.	0 - 1	0

32.7 Signal descriptions

Table 236 shows the interface signals of the core (VHDL ports).

*Table 236.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low

*Table 236.*Signal descriptions

Signal name	Field	Type	Function	Active
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	-
	SD[31:0]	Input	Not used	-
	CB[7:0]	Input	Checkbits	-
	PROMDATA[31:0]	Input	Not used	-
	EDAC	Input	The reset value for the PROM EDAC enable bit	High

*Table 236.*Signal descriptions

Signal name	Field	Type	Function	Active
SRO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	RAMSN[7:0]	Output	SRAM chip-select	Low
	RAMOEN[7:0]	Output	SRAM output enable	Low
	IOSN	Output	IO area chip select	Low
	ROMSN[7:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0]. Any WRN[ ] signal can be used for CB[ ].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0]. Any MBEN[ ] signal can be used for CB[ ].	
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus.Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low
	READ	Output	Read strobe	High
	RAMN	Output	Common SRAM Chip Select. Always asserted when one of the 8 RAMSN signals is asserted.	Low
	ROMN	Output	Common PROM Chip Select. Always asserted when one of the 8 ROMSN signals is asserted.	Low
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Checkbits	-
	PSEL	Output	Not used	-
	CE	Output	Single error detected.	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SDO	SDCASN	Output	Not used. All signals are drive to inactive state.	Low

* see GRLIB IP Library User’s Manual

32.8 Library dependencies

Table 237 shows libraries used when instantiating the core (VHDL libraries).

*Table 237.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

32.9 Component declaration

The core has the following component declaration.

```
component ftsrctrl is
  generic (
    hindex       : integer := 0;
    romaddr      : integer := 0;
    rommask      : integer := 16#ff0#;
    ramaddr      : integer := 16#400#;
    rammask      : integer := 16#ff0#;
    ioaddr       : integer := 16#200#;
    iomask       : integer := 16#ff0#;
    ramws        : integer := 0;
    romws        : integer := 2;
    iows         : integer := 2;
    rmw          : integer := 0;
    srbanks      : integer range 1 to 8  := 1;
    banksz       : integer range 0 to 15 := 15;
    rombanks     : integer range 1 to 8  := 1;
    rombanksz    : integer range 0 to 15 := 15;
    rombankszdef : integer range 0 to 15 := 15;
    pindex       : integer := 0;
    paddr        : integer := 0;
    pmask        : integer := 16#fff#;
    edacen       : integer range 0 to 1 := 1;
    errcnt       : integer range 0 to 1 := 0;
    cntbits      : integer range 1 to 8 := 1;
    wsreg        : integer := 0;
    oepol        : integer := 0;
    prom8en      : integer := 0
  );
  port (
    rst          : in  std_ulogic;
    clk          : in  std_ulogic;
    ahbsi        : in  ahb_slv_in_type;
    ahbso        : out ahb_slv_out_type;
    apbi         : in  apb_slv_in_type;
    apbo         : out apb_slv_out_type;
    sri          : in  memory_in_type;
    sro          : out memory_out_type;
    sdo          : out sdctrl_out_type
  );
end component;
```

32.10 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined in the example design's port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator. The CE signal of the memory controller is also connected to the AHB status register.

Memory controller decodes default memory areas: PROM area is 0x0 - 0xFFFFFF and RAM area is 0x40000000 - 0x40FFFFF.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0); -- optional sdram data
    cb       : inout std_logic_vector(7 downto 0); --checkbits
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdctrl_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
  signal gnd : std_ulogic;

  signal stati : ahbstat_in_type; --correctable error vector

begin

  -- AMBA Components are defined here ...


  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- AHB Status Register
  astat0 : ahbstat generic map(pindex => 13, paddr => 13, pirq => 11,
    nftslv => 1)
    port map(rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(13));

  stati.cerror(0) <= memo.ce;

  -- Memory controller
  mctrl0 : ftsrctrl generic map (rmw => 1, pindex => 10, paddr => 10,
    edacen => 1, errcnt => 1, cntbits => 4)
    port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(10), memi, memo,
  sdo);


  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  --I/O pads driving checkbit signals
  cb_pad : iopadv generic map (width => 8)
      port map (pad => cb,
                o => memi.cb,
                en => memo.bdrive(0),
                i => memo.cb;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= memo.ramoen;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;

end;
```

33 FTSRCTRL8 - 8-bit SRAM/16-bit IO Memory Controller with EDAC

33.1 Overview

The fault tolerant 8-bit SRAM/16-bit I/O memory interface uses a common 16-bit data bus to interface 8-bit SRAM and 16-bit I/O devices. It provides an Error Detection And Correction unit (EDAC), correcting up to two errors and detecting up to four errors in a data byte. The EDAC eight checkbits are stored in parallel with the 8-bit data in SRAM memory. Configuration of the memory controller functions is performed through the APB bus interface.

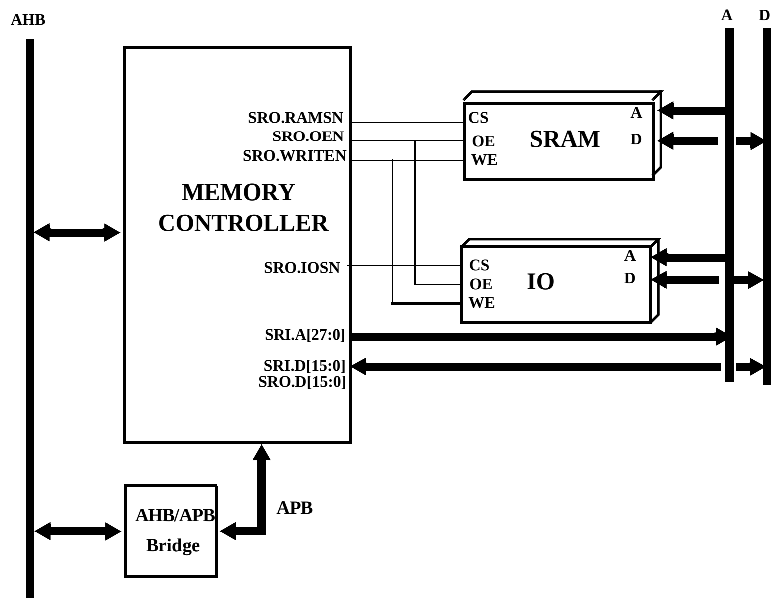


Figure 123. Block diagram

33.2 Operation

The controller is configured through VHDL generics to decode two address ranges: SRAM and I/O area. By default the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM can have up to 8 chip select signals. The controller generates a common write-enable signal (WRITEN) for both SRAM and I/O. The number of waitstates may be separately configured for the two address ranges.

The EDAC function is optional, and can be enabled with the edacen VHDL generic. The configuration of the EDAC is done through a configuration register accessed from the APB bus. During nominal operation, the EDAC checksum is generated and checked automatically. The 8-bit input to the EDAC function is split into two 4-bit nibbles. A modified hamming(8,4,4) coding featuring a single error correction and double error detection is applied to each 4-bit nibble. This makes the EDAC capable of correcting up to two errors and detecting up to four errors per 8-bit data. Single errors (correctable errors) are corrected without generating any indication of this condition in the bus response. If a multiple error (uncorrectable errors) is detected, a two cycle error response is given on the AHB bus.

Single errors may be monitored in two ways:

- by monitoring the CE signal which is asserted for one cycle each time a correctable error is detected.
- by checking the single error counter which is accessed from the MCFG3 configuration register.

The CE signal can be connected to the AHB status register which stores information of the AHB instruction causing the error and also generates interrupts. See the AHB status register documentation for more information.

The EDAC function can only be enabled for SRAM area accesses. If a 16-bit or 32-bit bus access is performed, the memory controller calculates the EDAC checksum for each byte read from the memory but the indication of single error is only signaled when the access is done. (I.e. if more than one byte in a 32-bit access has a single error, only one error is indicated for the hole 32-bit access.)

The equations below show how the EDAC checkbits are generated:

```
CB7 = Data[15] ^ Data[14] ^ Data[13]     // i.e. Data[7]
CB6 = Data[15] ^ Data[14] ^ Data[12]     // i.e. Data[6]
CB5 = Data[15] ^ Data[13] ^ Data[12]     // i.e. Data[5]
CB4 = Data[14] ^ Data[13] ^ Data[12]     // i.e. Data[4]
CB3 = Data[11] ^ Data[10] ^ Data[ 9]     // i.e. Data[3]
CB2 = Data[11] ^ Data[10] ^ Data[ 8]     // i.e. Data[2]
CB1 = Data[11] ^ Data[ 9] ^ Data[ 8]     // i.e. Data[1]
CB0 = Data[10] ^ Data[ 9] ^ Data[ 8]     // i.e. Data[0]
```

33.2.1 Memory access

The memory controller supports 32/16/8-bit single accesses and 32-bit burst accesses to the SRAM. A 32-bit or a 16-bit access is performed as multiple 8-bit accesses on the 16-bit memory bus, where data is transferred on data lines 8 to 15 (Data[15:8]). The eight checkbits generated/used by the EDAC are transferred on the eight first data lines (Data[7:0]). For 32-bit and 16-bit accesses, the bytes read from the memory are arranged according to the big-endian order (i.e. for a 32-bit read access, the bytes read from memory address A, A+1, A+2, and A+3 correspond to the bit[31:24], bit[23:16], bit[15:8], and bit[7:0] in the 32-bit word transferred to the AMBA bus. The table 245 shows the expected latency from the memory controller.

*Table 238.*FTSCTRL8 access latency

Accesses	**Single data**	**First data (burst)**	**Middle data (burst)**	**Last data (burst)**
32-bit write	10	8	8	10
32-bit read	6	6	4	4
16-bit write	4 (+1)	-	-	-
16-bit read	4	-	-	-
8-bit write	4	-	-	-
8-bit read	3	-	-	-

One extra cycle is added for 16-bit burst accesses when Bus Exception is enabled.

33.2.2 I/O access

The memory controller accepts 32/16/8-bit single accesses to the I/O area, but the access generated towards the I/O device is always 16-bit. The two least significant bits of the AMBA address (byte address) determine which half word that should be transferred to the I/O device. (i.e. If the byte address is 0 and it is a 32-bit access, bits 16 to 31 on the AHB bus is transferred on the 16-bit memory bus. If the byte address is 2 and it is a 16-bit access, bit 0 to 15 on the AHB bus is transferred on the 16-bit memory bus.) If the access is an 8-bit access, the data is transferred on data lines 8 to 15 (Data[15:8]) on the memory bus. In case of a write, data lines 0 to 7 is also written to the I/O device but these data lines do not transfer any valid data.

33.2.3 Using Bus Exception

The active low Bus Exception signal (BEXCN) can be used to signal access errors. It is enabled by setting the BEXCEN bit in MCFG1 and is only active for the I/O area. The BEXCN signal is sampled on the same cycle as data is written to memory or read data is sampled. When a bus exception is detected an error response will be generated for the access. One additional latency cycle is added to the AMBA access when the Bus Exception is enable.

33.2.4 Using Bus Ready

The Bus Ready (BRDYN) signal can be used to add waitstates to I/O-area accesses. It is enabled by setting the Bus Ready Enable (BRDYEN) bit in the MCFG1 register. An access will have at least the amount of waitstates set with the VHDL generic or through the register, but will be further stretched until BRDYN is asserted. Additional waitstates can thus be inserted after the pre-set number of wait-states by deasserting the BRDYN signal. BRDYN should be asserted in the cycle preceding the last one. It is recommended that BRDY remains asserted until the IOSN signal is de-asserted, to ensure that the access has been properly completed and avoiding the system to stall.

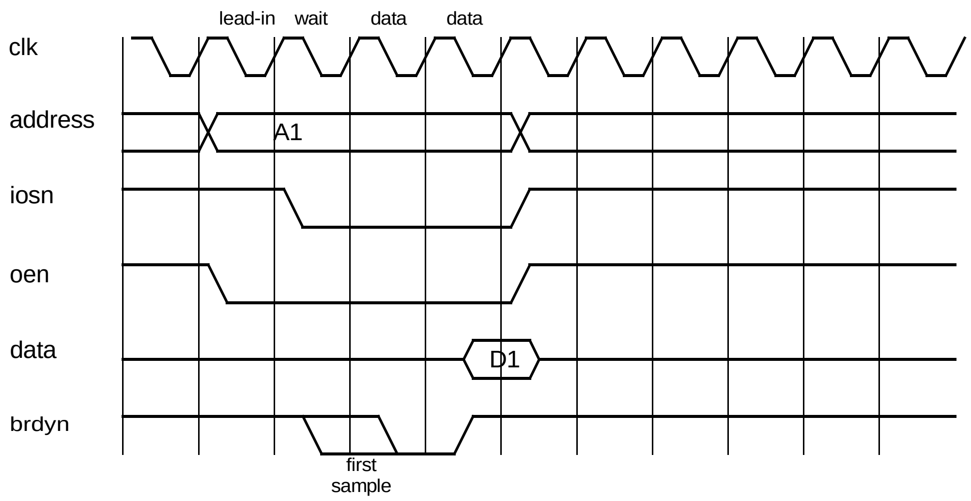


Figure 124. I/O READ cycle, programmed with 1 wait state, and with an extra data cycle added with BRDYN.

33.3 SRAM/IO waveforms

The internal and external waveforms of the interface are presented in the figures below.

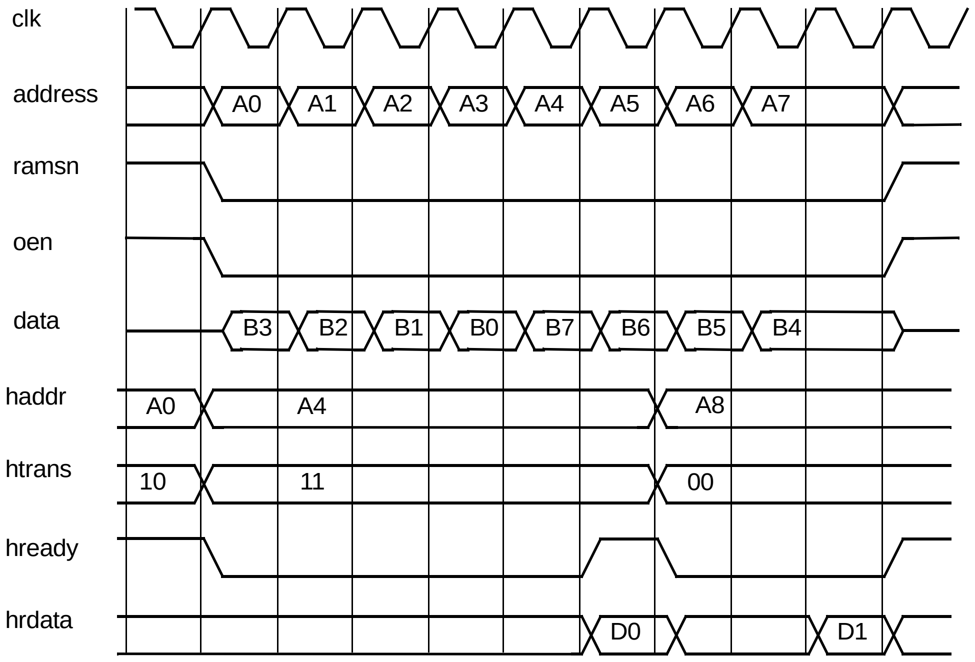


Figure 125. 32-bit SRAM sequential read accesses with 0 wait-states and EDAC enabled.

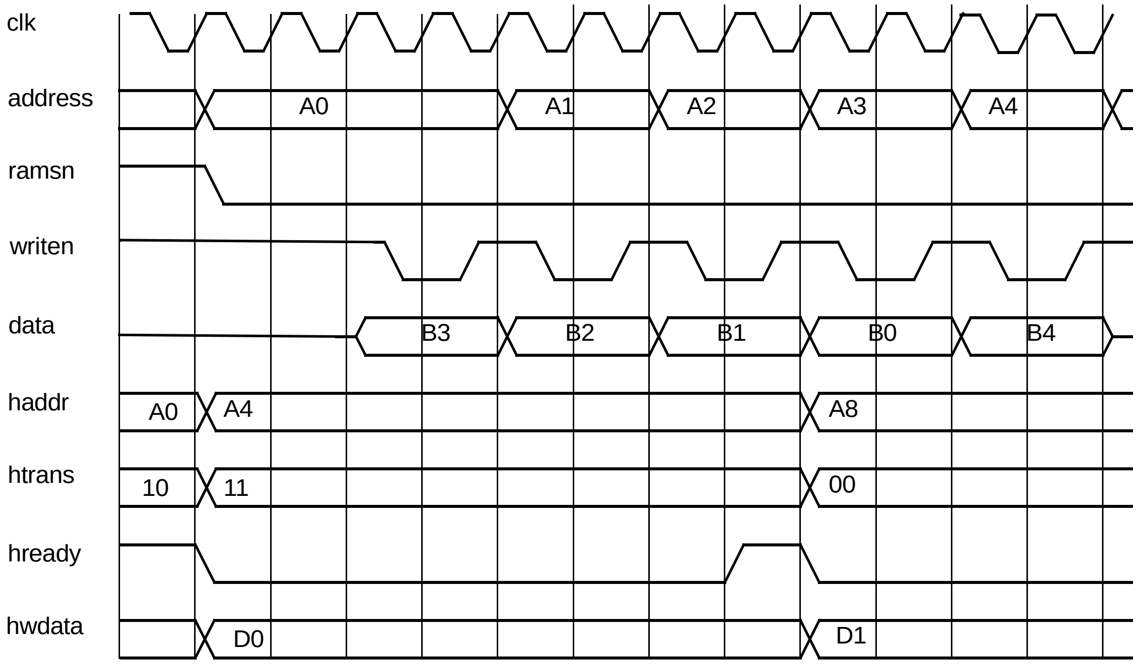


Figure 126. 32-bit SRAM sequential writeaccess with 0 wait-states and EDAC enabled.

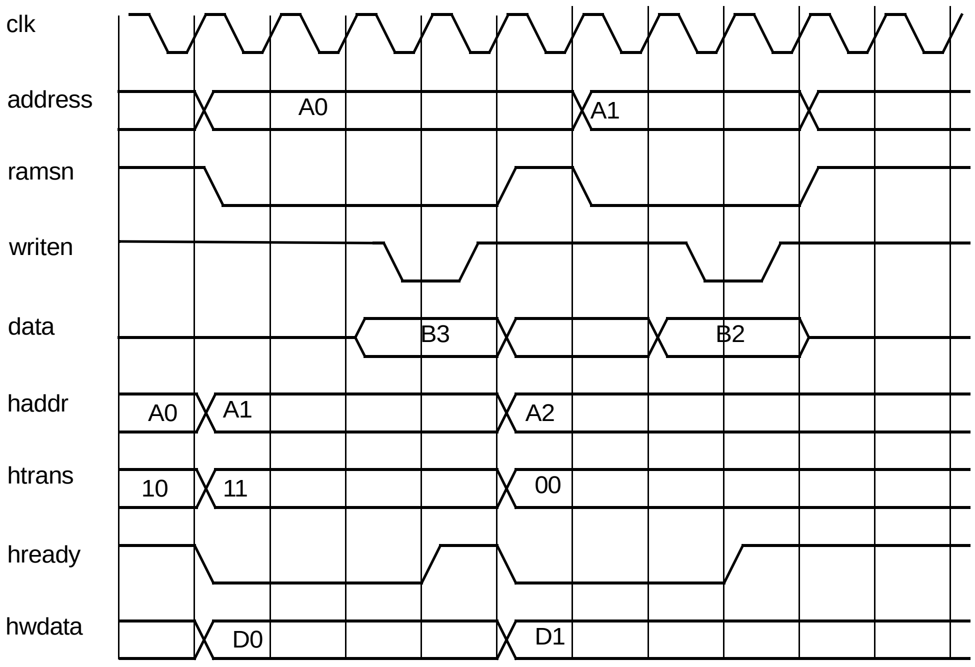


Figure 127. 8-bit SRAM non-sequential write access with 0 wait-states and EDAC enabled.

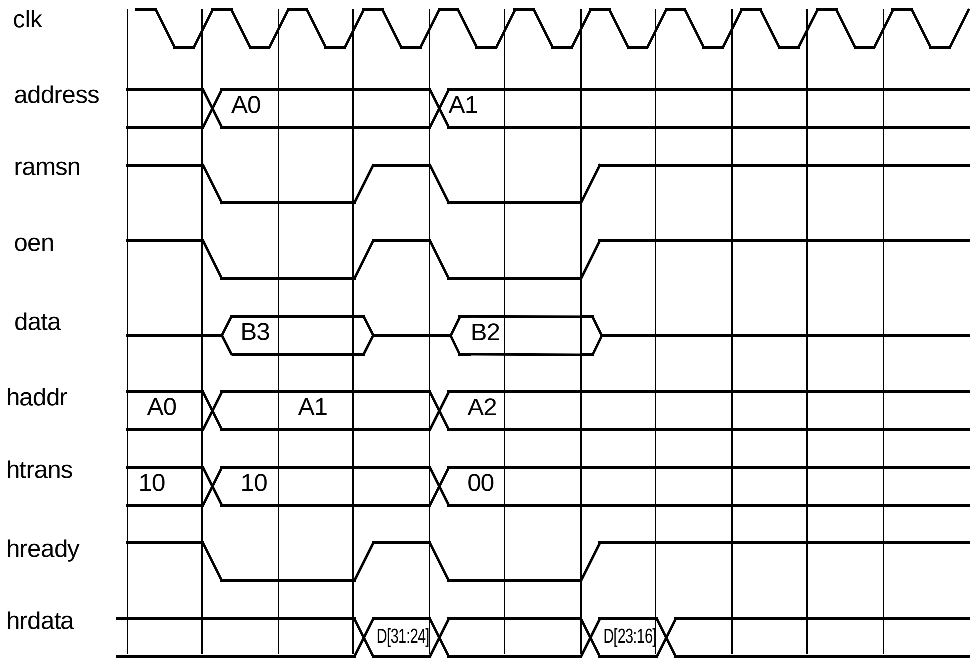


Figure 128. 8-bit SRAM non-sequential read access with 0 wait-states and EDAC enabled.

On a read access, data is sampled one clock cycle before HREADY is asserted.

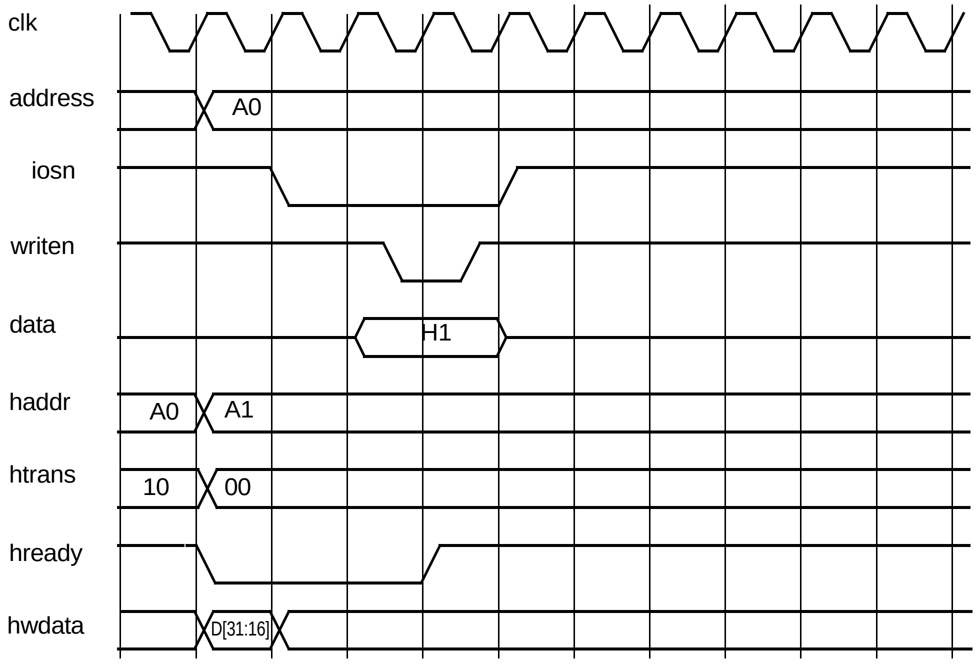


Figure 129. 16-bit I/O non-sequential write access with 0 wait-states.

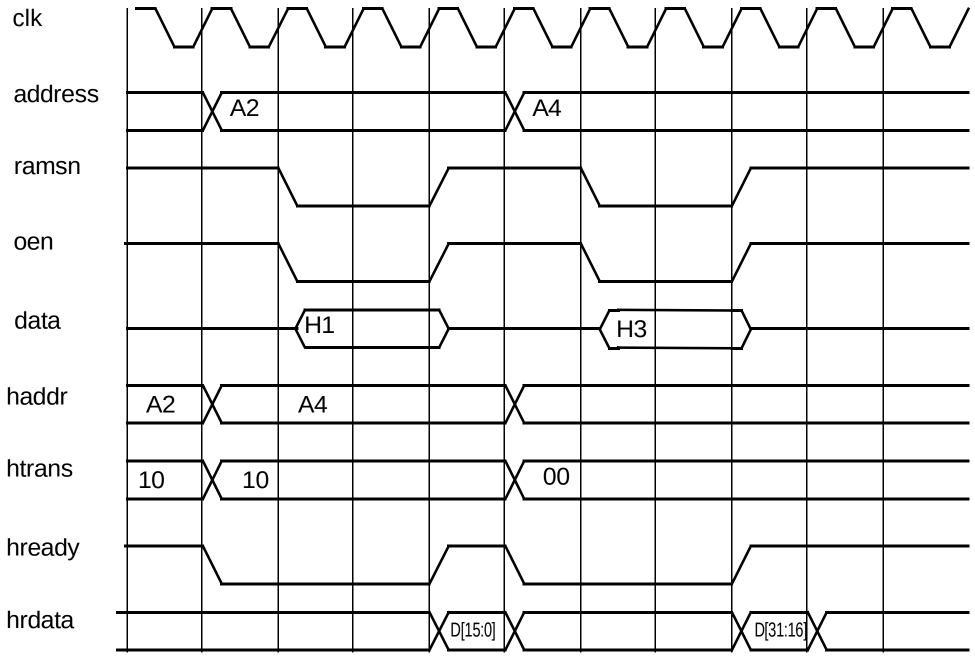


Figure 130. 16-bit I/O non-sequential read access with 0 wait-states.

I/O write accesses are extended with one extra latency cycle if the bus exception is enabled.

If waitstates are configured through the VHDL generics or registers, one extra data cycle will be inserted for each waitstate in both read and write cycles.

33.4 Registers

The core is programmed through registers mapped into APB address space.

*Table 239.*FT SRAM/IO controller registers

APB Address offset	Register
0x0	Memory configuration register 1
0x4	Memory configuration register 2
0x8	Memory configuration register 3

Table 240. MCFG1 register

31 : 27	26	25	24	23 : 20	19 : 0
RESERVED	BRDY	BEXC		IOWS	RESERVED

31 : 27	RESERVED
26	BRDYEN: Enables the BRDYN signal.
25	BEXCEN: Enables the BEXCN signal.
24	RESERVED
23 : 20	IOWS: Sets the number of waitstates for accesses to the IO area. Only available if the wsreg VHDL generic is set to one.
19 : 0	RESERVED

Table 241. MCFG2 register

31 : 13	12 : 9	8 : 2	1 : 0
RESERVED	RAMBSZ	RESERVED	RAMWS

31 : 12	RESERVED
12 : 9	RAMBSZ: Sets the SRAM bank size. Only available if the banksz VHDL generic is set to zero. Otherwise the banksz VHDL generic sets the bank size. 0 = 8 kB, 15 = 256 MB.
8 : 2	RESERVED
1 : 0	RAMWS: Sets the number of waitstates for accesses to the RAM area. Only available if the wsreg VHDL generic is set to one.

Table 242. MCFG3 register

31 : cnt + 13	cnt + 12 : 12	11	10	9	8	7 : 0
RESERVED	SEC	WB	RB	SEN		TCB

31 : cnt+13	RESERVED
cnt+12 : 12	SEC. Single error counter. This field increments each time a single error is detected. It saturates at the maximum value that can be stored in this field. Each bit can be reset by writing a one to it. cnt = the number of counter bits.
11	WB: Write bypass. If set, the TCB field will be used as checkbits in all write operations.

Table 242. MCFG3 register

10	RB: Read bypass. If set, checkbits read from memory in all read operations will be stored in the TCB field.
9	SEN: SRAM EDAC enable. If set, EDAC will be active for the SRAM area.
8	RESERVED
7 : 0	TCB: Used as checkbits in write operations when WB is one and checkbits from read operations are stored here when RB is one.

All the fields in the MCFG3 register are available if the edacen VHDL generic is set to one except for the SEC field which also requires that the errcnt VHDL generic is set to one.

33.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x056. For description of vendor and device identifiers see the GRLIB IP Library User’s Manual.

33.6 Configuration options

Table 239 shows the configuration options of the core (VHDL generics).

Table 243. Controller configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index.	1 - NAHBSLV-1	0
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default RAM area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 - 16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area.	0 - 15	0
iows	Number of waitstates during access to IO area.	0 - 15	2
srbanks	Set the number of RAM banks.	1 - 8	1
banksz	Set the size of bank 1 - 4. 1 = 16 kB, ... , 15 = 256 MB. If set to zero, the bank size is set with the rambsz field in the MCFG2 register.	0 - 15	15
pindex	APB slave index.	1 - NAPBSLV-1	0
paddr	APB address.	1 - 16#FFF#	0
pmask	APB address mask.	1 - 16#FFF#	16#FFF#
edacen	EDAC enable. If set to one, EDAC logic is synthesized.	0 - 1	0
errcnt	If one, a single error counter is added.	0 - 1	0
cntbits	Number of bits in the single error counter.	1 - 8	1
wsreg	Enable programmable waitstate generation.	0 - 1	0

33.7 Signal descriptions

Table 244 shows the interface signals of the core (VHDL ports).

*Table 244.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low

*Table 244.*Signal descriptions

Signal name	Field	Type	Function	Active
SRI	DATA[31:0]	Input	Memory data: [15:0] used for IO accesses [7:0] used for checkbits for SRAM accesses [15:8] use for data for SRAM accesses	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	Not used	-
	SD[31:0]	Input	Not used	-
	CB[7:0]	Input	Not used	-
	PROMDATA[31:0]	Input	Not used	-
	EDAC	Input	Not used	-

*Table 244.*Signal descriptions

Signal name	Field	Type	Function	Active
SRO	ADDRESS[31:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data: [15:0] used for IO accesses [7:0] used for checkbits for SRAM accesses [15:8] use for data for SRAM accesses	High
	RAMSN[7:0]	Output	SRAM chip-select	Low
	RAMOEN[7:0]	Output	SRAM output enable	Low
	IOSN	Output	IO area chip select	Low
	ROMSN[7:0]	Output	Not used	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[15:8], WRN[1] corresponds to DATA[7:0], WRN[3:2] Not used	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[15:8], BDRIVE[1] corresponds to DATA[7:0], BDRIVE[3:2] Not used	Low
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signal.	Low
	READ	Output	Read strobe	High
	RAMN	Output	Common SRAM Chip Select. Always asserted when one of the 8 RAMSN signals is asserted.	Low
	ROMN	Output	Not used	-
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Not used	-
	PSEL	Output	Not used	-
	CE	Output	Single error detected.	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-

* see GRLIB IP Library User’s Manual

33.8 Library dependencies

Table 245 shows libraries used when instantiating the core (VHDL libraries).

*Table 245.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

33.9 Component declaration

The core has the following component declaration.

```
component ftsrctrl8 is
  generic (
    hindex       : integer := 0;
    ramaddr      : integer := 16#400#;
    rammask      : integer := 16#ff0#;
    ioaddr       : integer := 16#200#;
    iomask       : integer := 16#ff0#;
    ramws        : integer := 0;
    iows         : integer := 2;
    srbanks      : integer range 1 to 8  := 1;
    banksz       : integer range 0 to 15 := 15;
    pindex       : integer := 0;
    paddr        : integer := 0;
    pmask        : integer := 16#fff#;
    edacen       : integer range 0 to 1 := 1;
    errcnt       : integer range 0 to 1 := 0;
    cntbits      : integer range 1 to 8 := 1;
    wsreg        : integer := 0;
    oepol        : integer := 0
  );
  port (
    rst          : in  std_ulogic;
    clk          : in  std_ulogic;
    ahbsi        : in  ahb_slv_in_type;
    ahbso        : out ahb_slv_out_type;
    apbi         : in  apb_slv_in_type;
    apbo         : out apb_slv_out_type;
    sri          : in  memory_in_type;
    sro          : out memory_out_type
  );
end component;
```

33.10 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined in the example design’s port map and connected to the memory controller. The system clock and reset are generated by GR Clock Generator and Reset Generator. The CE signal of the memory controller is also connected to the AHB status register.

The memory controller decodes default memory areas: I/O area is 0x20000000 - 0x20FFFFFF and RAM area is 0x40000000 - 0x40FFFFF.

```
library ieee;
```

```
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity ftsrctrl8_ex is
  port (
    resetn     : in  std_ulogic;
    clk        : in  std_ulogic;

    address    : out std_logic_vector(27 downto 0);
    data       : inout std_logic_vector(31 downto 0);
    ramsn      : out std_logic_vector (3 downto 0);
    ramoen     : out std_logic_vector (3 downto 0);
    rwen       : out std_logic_vector (3 downto 0);
    oen        : out std_ulogic;
    writen     : out std_ulogic;
    read       : out std_ulogic;
    iosn       : out std_ulogic;
    brdyn      : in  std_ulogic; -- Bus ready
    bexcn      : in  std_ulogic  -- Bus exception
   );
end;

architecture rtl of ftsrctrl8_ex is
signal memi  : memory_in_type;
signal memo  : memory_out_type;

signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal clkm, rstn, rstraw : std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

signal stati : ahbstat_in_type;

begin

  -- clock and reset
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw; cgi.pllref  <= ’0’;
  clk_pad : clkpad port map (clk, clkm);
  rst0 : rstgen               -- reset generator
  port map (resetn, clkm, ’1’, rstn, rstraw);

  -- AHB controller
  ahb0 : ahbctrl              -- AHB arbiter/multiplexer
  generic map (rrobin => 1, ioaddr => 16#fff#, devid => 16#201#)
  port map (rstn, clkm, ahbmi, ahbmo, ahbsi, ahbso);

  -- Memory controller
  sr0 : ftsrctrl8 generic map (hindex => 0, pindex => 0, edacen => 1)
    port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(0), memi, memo);

  brdyn_pad : inpad port map (brdyn, memi.brdyn);
  bexcn_pad : inpad port map (bexcn, memi.bexcn);
  addr_pad : outpadv generic map (width => 28 )
    port map (address, memo.address(27 downto 0));
  rams_pad : outpadv generic map (width => 4)
    port map (ramsn, memo.ramsn(3 downto 0));
  oen_pad  : outpad
    port map (oen, memo.oen);
  rwen_pad : outpadv generic map (width => 4)
    port map (rwen, memo.wrn);
  roen_pad : outpadv generic map (width => 4)
    port map (ramoen, memo.ramoen(3 downto 0));
  wri_pad  : outpad
    port map (writen, memo.writen);
  read_pad : outpad
    port map (read, memo.read);
  iosn_pad : outpad
    port map (iosn, memo.iosn);
  data_pad : iopadvv generic map (width => 8) -- SRAM and I/O Data
    port map (data(15 downto 8), memo.data(15 downto 8),
       memo.vbdrive(15 downto 8), memi.data(15 downto 8));
  cbdata_pad : iopadvv generic map (width => 8) -- SRAM checkbits and I/O Data
    port map (data(7 downto 0), memo.data(7 downto 0),
       memo.vbdrive(7 downto 0), memi.data(7 downto 0));

  -- APB bridge and AHB stat
  apb0 : apbctrl              -- AHB/APB bridge
  generic map (hindex => 1, haddr => 16#800#)
    port map (rstn, clkm, ahbsi, ahbso(1), apbi, apbo );

  stati.cerror(0) <= memo.ce;
  ahbstat0 : ahbstat generic map (pindex => 15, paddr => 15, pirq => 1)
    port map (rstn, clkm, ahbmi, ahbsi, stati, apbi, apbo(15));
end;
```

34 GPTIMER - General Purpose Timer Unit

34.1 Overview

The General Purpose Timer Unit provides a common prescaler and decrementing timer(s). Number of timers is configurable through the *ntimers* VHDL generic in the range 1 to 7. Prescaler width is configured through the sbits VHDL generic. Timer width is configured through the *tbits* VHDL generic. The timer unit acts a slave on AMBA APB bus. The unit implements one 16 bit prescaler and 3 decrementing 32 bit timer(s). The unit is capable of asserting interrupt on timer(s) underflow. Interrupt is configurable to be common for the whole unit or separate for each timer.

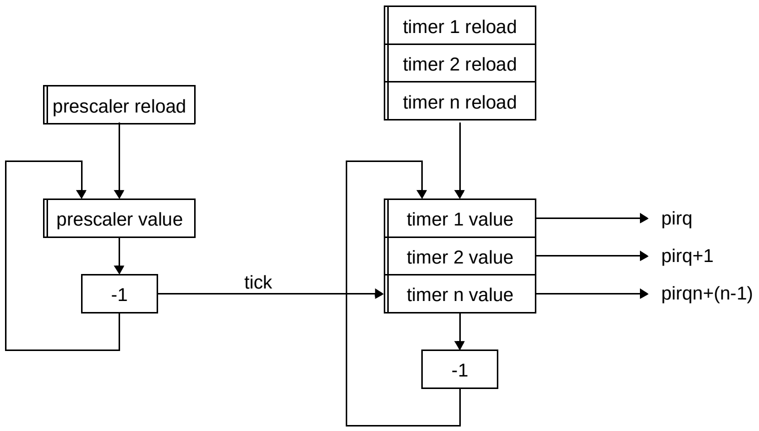


Figure 131. General Purpose Timer Unit block diagram

34.2 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.

The operation of each timers is controlled through its control register. A timer is enabled by setting the enable bit in the control register. The timer value is then decremented on each prescaler tick. When a timer underflows, it will automatically be reloaded with the value of the corresponding timer reload register if the restart bit in the control register is set, otherwise it will stop at -1 and reset the enable bit.

The timer unit can be configured to generate common interrupt through a VHDL-generic. The shared interrupt will be signalled when any of the timers with interrupt enable bit underflows. The timer unit will signal an interrupt on appropriate line when a timer underflows (if the interrupt enable bit for the current timer is set), when configured to signal interrupt for each timer. The interrupt pending bit in the control register of the underflown timer will be set and remain set until cleared by writing ‘0’.

To minimize complexity, timers share the same decrementer. This means that the minimum allowed prescaler division factor is *ntimers*+1 (reload register = *ntimers*) where *ntimers* is the number of implemented timers.

By setting the chain bit in the control register timer *n* can be chained with preceding timer *n*-1. Decrementing timer *n* will start when timer *n*-1 underflows.

Each timer can be reloaded with the value in its reload register at any time by writing a ‘one’ to the load bit in the control register. The last timer acts as a watchdog, driving a watchdog output signal when expired, when the *wdog* VHDL generic is set to a time-out value larger than 0. At reset, the scaler is set to all ones and the watchdog timer is set to the *wdog* VHDL generic, 0xFFF.

34.3 Registers

The core is programmed through registers mapped into APB address space. The number of implemented registers depend on number of implemented timers.

*Table 246.*General Purpose Timer Unit registers

APB address offset	**Register**
0x00	Scaler value
0x04	Scaler reload value
0x08	Configuration register
0x0C	Unused
0x10	Timer 1 counter value register
0x14	Timer 1 reload value register
0x18	Timer 1 control register
0x1C	Unused
0x*n*0	Timer *n* counter value register
0x*n*4	Timer *n* reload value register
0x*n*8	Timer *n* control register

Table 247. Scaler value

31 – 16	16-1 – 0
“000..0”	SCALER VALUE

16-1: 0	Scaler value Any unused most significant bits are reserved. Always reads as ‘000...0’.

Table 248. Scaler reload value

31 – 16	16-1 – 0
“000..0”	SCALER RELOAD VALUE

16-1: 0	Scaler reload value Any unused most significant bits are reserved. Always reada as ‘000...0’.

Table 249. General Purpose Timer Unit Configuration Register

31 – 10	9	8	7 – 3	2 – 0
“000..0”	DF	SI	IRQ	TIMERS

31: 10	Reserved. Always reads as ‘000...0’.
9	Disable timer freeze (DF). If set the timer unit can not be freezed, otherwise signal GPTI.DHALT freezes the timer unit.
8	Separate interrupts (SI). Reads ‘1’ if the timer unit generates separate interrupts for each timer, otherwise ‘0’. Read-only.
7: 3	APB Interrupt: If configured to use common interrupt all timers will drive APB interrupt nr. IRQ, otherwise timer *n*will drive APB Interrupt IRQ+*n* (has to be less the MAXIRQ). Read-only.
2: 0	Number of implemented timers. Read-only.

Table 250. Timer counter value register

32-1 – 0
TIMER COUNTER VALUE

Table 250. Timer counter value register

32-1: 0	Timer Counter value. Decremented by 1 for each prescaler tick. Any unused most significant bits are reserved. Always reads as ‘000...0’.

Table 251. Timer reload value register

32-1 … 0
TIMER RELOAD VALUE

32-1: 0	Timer Reload value. This value is loaded into the timer counter value register when ‘1’ is written to load bit in the timers control register or when the RS bit is set in the control register and the timer underflows. Any unused most significant bits are reserved. Always reads as ‘000...0’.

Table 252. General Purpose Timer Unit Configuration Register

31 … 7	6	5	4	3	2	1	0
“000..0”	DH	CH	IP	IE	LD	RS	EN

31: 7	Reserved. Always reads as ‘000...0’.
6	Debug Halt (DH): Value of GPTI.DHALT signal which is used to freeze counters (e.g. when a system is in debug mode). Read-only.
5	Chain (CH): Chain with preceding timer. If set for timer *n*, decrementing timer *n* begins when timer (*n*-1) underflows.
4	Interrupt Pending (IP): Sets when an interrupt is signalled. Remains ‘1’ until cleared by writing ‘0’ to this bit.
3	Interrupt Enable (IE): If set the timer signals interrupt when it underflows.
2	Load (LD): Load value from the timer reload register to the timer counter value register.
1	Restart (RS): If set, the timer counter value register is reloaded with the value of the reload register when the timer underflows
0	Enable (EN): Enable the timer.

34.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x011. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

34.5 Configuration options

Table 253 shows the configuration options of the core (VHDL generics).

*Table 253.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the timer unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 4095	0
pmask	The APB address mask	0 to 4095	4095
nbits	Defines the number of bits in the timers	1 to 32	32
ntimers	Defines the number of timers in the unit	1 to 7	1
pirq	Defines which APB interrupt the timers will generate	0 to MAXIRQ-1	0
sepirq	If set to 1, each timer will drive an individual interrupt line, starting with interrupt *irq*. If set to 0, all timers will drive the same interrupt line (*irq*).	0 to MAXIRQ-1 (note: *ntimers* + *irq* must be less than MAXIRQ)	0
sbits	Defines the number of bits in the scaler	1 to 32	16
wdog	Watchdog reset value. When set to a non-zero value, the last timer will be enabled and pre-loaded with this value at reset. When the timer value reaches 0, the WDOG output is driven active.	0 to 2^{nbits} - 1	0

34.6 Signal descriptions

Table 254 shows the interface signals of the core (VHDL ports).

*Table 254.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPTI	DHALT	Input	Freeze timers	High
	EXTCLK	Input	Use as alternative clock	-
GPTO	TICK[0:7]	Output	Timer ticks. TICK[0] is high for one clock each time the scaler underflows. TICK[1-n] are high for one clock each time the corrspondning timer underflows.	High
	WDOG	Output	Watchdog output. Equivalent to interrupt pending bit of last timer.	High
	WDOGN	Output	Watchdog output Equivalent to interrupt pending bit of last timer.	Low

* see GRLIB IP Library User’s Manual

34.7 Library dependencies

Table 255 shows libraries used when instantiating the core (VHDL libraries).

*Table 255.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, component	Component declaration

34.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity gptimer_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ... -- other signals
    );
end;

architecture rtl of gptimer_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- GP Timer Unit input signals
  signal gpti : gptimer_in_type;

begin

  -- AMBA Components are instantiated here
  ...


  -- General Purpose Timer Unit
  timer0 : gptimer
  generic map (pindex => 3, paddr => 3, pirq => 8, sepirq => 1)
  port map (rstn, clk, apbi, apbo(3), gpti, open);

  gpti.dhalt <= ’0’; gpti.extclk <= ’0’; -- unused inputs

end;
```

35 GRAES - Advanced Encryption Standard

35.1 Overview

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm for high throughput application (like audio or video streams). The GRAES core implements the AES-128 algorithm, supporting the Electronic Codebook (ECB) method. The AES-128 algorithm is specified in the "Advanced Encryption Standard (AES)" document, Federal Information Processing Standards (FIPS) Publication 197. The document is established by the National Institute of Standards and Technology (NIST).

The core provides the following internal AMBA AHB slave interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Advanced Encryption Standard (AES) core
- AMBA AHB slave
- GRLIB plug&play wrapper

Note that the core can also be used without the GRLIB plug&play information.

35.2 Operation

The input and output for the AES algorithm each consist of sequences of 128 bits (digits with values of 0 or 1). These sequences will sometimes be referred to as blocks and the number of bits they contain will be referred to as their length. The cipher key for the AES-128 algorithm is a sequence of 128 bits (can also be 192 or 256 bits for other algorithms).

To transfer a 128 bit key or data block four write operations are necessary since the bus interface is 32 bit wide. After supplying a "key will be input" command to the control register, the key is input via four registers. After supplying a "data will be input" command to the control register, the input data is written via four registers. After the last input data register is written, the encryption or decryption is started. The progress can be observed via the debug register. When the operation is completed, an interrupt is generated. The output data is then read out via four registers. Note that the above sequence must be respected. It is not required to write a new key between each data input. There is no command needed for reading out the result.

The implementation requires around 89 clock cycles for a 128 bit data block in encryption direction and around 90 clock cycles for decryption direction. For decryption an initial key calculation is required. This takes around 10 additional clock cycles per every new key. Typically large amounts of data are decrypted (and also encrypted) with the same key. The key initialization for the decryption round does not influence the throughput.

35.3 Background

The Federal Information Processing Standards (FIPS) Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of the Information Technology Management Reform Act.

The Advanced Encryption Standard (AES) standard specifies the Rijndael algorithm, a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. Rijndael was designed to handle additional block sizes and key lengths, however they are not adopted in this standard.

35.4 AES-128 parameters

The GRAES core implements AES-128. An AES algorithm is defined by the following parameters according to FIPS-197:

- *Nk* number of 32-bit words comprising the cipher key
- *Nr* number of rounds

The AES-128 algorithm is specified as *Nk*=4 and *Nr*=10.

The GRAES core has been verified against the complete set of Known Answer Test vectors included in the AES Algorithm Validation Suite (AESAVS) from National Institute of Standards and Technology (NIST), Information Technology Laboratory, Computer Security Division.

35.5 Throughput

The data throughput for the GRAES core is around 128/90 bits per clock cycle, i.e. approximately 1.4 Mbits per MHz.

The underlaying AES core has been implemented in a dual crypto chip on 250 nm technology as depicted in the figure below. The throughput at 33 MHz operating frequency was 42 Mbit/s, the power consumption was 9,6 mW, and the size was 14,5 kgates.

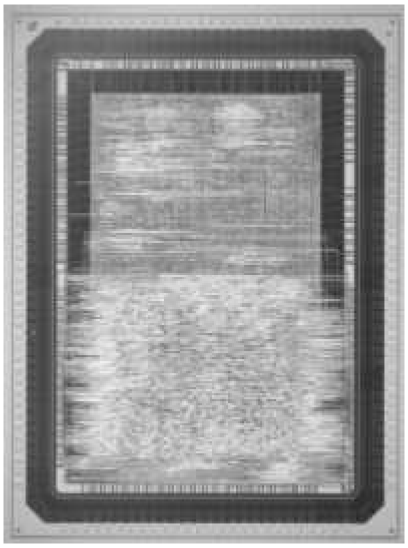


Figure 132. Dual Crypto Chip

35.6 Characteristics

The GRAES core has been synthesized for a Xilinx Virtex-2 XC2V6000-4 devices with the following results:

- LUTs: 5040 (7%)
- 256x1 ROMs (ROM256X1): 128

- Frequency:125 MHz

35.7 Registers

The core is programmed through registers mapped into AHB I/O address space.

*Table 256.*GRAES registers

AHB I/O address offset	Register
16#000#	Control Register
16#010#	Data Input 0 Register
16#014#	Data Input 1 Register
16#018#	Data Input 2 Register
16#01C#	Data Input 3 Register
16#020#	Data Output 0 Register
16#024#	Data Output 1 Register
16#028#	Data Output 2 Register
16#02C#	Data Output 3 Register
16#03C#	Debug Register

35.7.1 Control Register (W)

*Table 257.*Control Register

31 ... 2	1	0
-	DEC	KEY

31-2:	-	Unused
1:	DEC	0 = “encrypt”, 1 = “decrypt” (only relevant when KEY=1)
0:	KEY	0 = “data will be input”, 1 = “key will be input”

Note that the Data Input Registers cannot be written before a command is given to the Control Register. Note that the Data Input Registers must then be written in sequence, and all four registers must be written else the core ends up in an undefined state.

The KEY bit determines whether a key will be input (KEY=1), or data will be input (KEY=0). When a “key will be input” command is written, the DEC bit determines whether decryption (DEC=1) or encryption (DEC=0) should be applied to the subsequent data input.

Note that the register cannot be written after a command has been given, until the specific operation completes. A write access will be terminated with an AMBA AHB error response till the Data Input Register 3 has been written, and the with an AMBA AHB retry response till the operation completes. Any read access to this register results in an AMBA AHB error response.

35.7.2 Debug Register (R)

*Table 258.*Debug Register

31 ... 0
FSM

31-0:	FSM	Finite State Machine

Any write access to this register results in an AMBA AHB error response.

35.7.3 Data Input Registers (W)

*Table 259.*Data Input 0 Register

31	0
Data/Key(127 downto 96)	

*Table 260.*Data Input 1 Register

31	0
Data/Key(95 downto 64)	

*Table 261.*Data Input 2 Register

31	0
Data/Key(63 downto 32)	

*Table 262.*Data Input 3 Register

31	0
Data/Key(31 downto 0)	

Note that these registers can only be written with a key after a “key will be input” command has been written to the control register. Note that the registers must then be written in sequence, and all four registers must be written else the core ends up in an undefined state.

Note that these registers can only be written with data after a “data will be input” command has been written to the control register, else an AMBA AHB error response is given. Note that the registers must then be written in sequence and all four registers must be written else the core ends up in an undefined state. The encryption or decryption operation is started when the Data Input 3 Register is written to with data.

35.7.4 Data Output Registers (R)

*Table 263.*Data Output 0 Register

31	0
Data(127 downto 96)	

*Table 264.*Data Output 1 Register

31	0
Data(95 downto 64)	

*Table 265.*Data Output 2 Register

31	0
Data(63 downto 32)	

*Table 266.*Data Output 3 Register

31	0
Data(31 downto 0)	

Note that these registers can only be read after encryption or decryption has been completed. An AMBA AHB response is given to read accesses that occur while the encryption or decryption is in progress. If a read access is attempted before an encryption or decryption has even been initiated, then an AMBA AHB erro response is given. Write accesses to these registers result in an AMBA AHB error response.

35.8 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x073. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

35.9 Configuration options

Table 267 shows the configuration options of the core (VHDL generics).

*Table 267.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	0 - NAHBSLV-1	0
ioaddr	Addr field of the AHB I/O BAR	0 - 16#FFF#	0
iomask	Mask field of the AHB I/O BAR	0 - 16#FFF#	16#FFC#
hirq	Interrupt line used by the GRAES	0 - NAHBIRQ-1	0

35.10 Signal descriptions

Table 268 shows the interface signals of the core (VHDL ports).

*Table 268.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBI	*	Input	AHB slave input signals	-
AHBO	*	Output	AHB slave output signals	-
DEBUG[0:4]	N/A	Output	Debug information	-

* see GRLIB IP Library User’s Manual

Note that the AES core can also be used without the GRLIB plug&play information. The AMBA AHB signals are then provided as IEEE Std_Logic_1164 compatible scalars and vectors.

35.11 Library dependencies

Table 269 shows libraries used when instantiating the core (VHDL libraries).

*Table 269.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRAES component declarations

35.12 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;
```

```
library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.crypto.all;
...
...
   signal debug: std_logic_vector(0 to 4);
..
..
   GRAES0: graes
      generic map (
         hindex         => hindex,
         ioaddr         => ioaddr,
         iomask         => iomask,
         hirq           => hirq)
      port map (
         rstn           => rstn,
         clk            => clk,
         ahbi           => ahbsi,
         ahbo           => ahbso(hindex),
         debug          => debug);
```

36 GRCAN - CAN 2.0 Controller with DMA

36.1 Overview

The CAN controller is assumed to operate in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling. The AMBA AHB bus is used for retrieving and storing CAN messages in memory external to the CAN controller. This memory can be located on-chip, as shown in the block diagram, or external to the chip.

The CAN controller supports transmission and reception of sets of messages by use of circular buffers located in memory external to the core. Separate transmit and receive buffers are assumed. Reception and transmission of sets of messages can be ongoing simultaneously.

After a set of message transfers has been set up via the AMBA APB interface the DMA controller initiates a burst of read accesses on the AMBA AHB bus to fetch messages from memory, which are performed by the AHB master. The messages are then transmitted by the CAN core. When a programmable number of messages have been transmitted, the DMA controller issues an interrupt.

After the reception has been set up via the AMBA APB interface, messages are received by the CAN core. To store messages to memory, the DMA controller initiates a burst of write accesses on the AMBA AHB bus, which are performed by the AHB master. When a programmable number of messages have been received, the DMA controller issues an interrupt.

The CAN controller can detect a SYNC message and generate an interrupt, which is also available as an output signal from the core. The SYNC message identifier is programmable via the AMBA APB interface. Separate synchronisation message interrupts are provided.

The CAN controller can transmit and receive messages on either of two CAN busses, but only on one at a time. The selection is programmable via the AMBA APB interface.

Note that it is not possible to receive a CAN message while transmitting one.

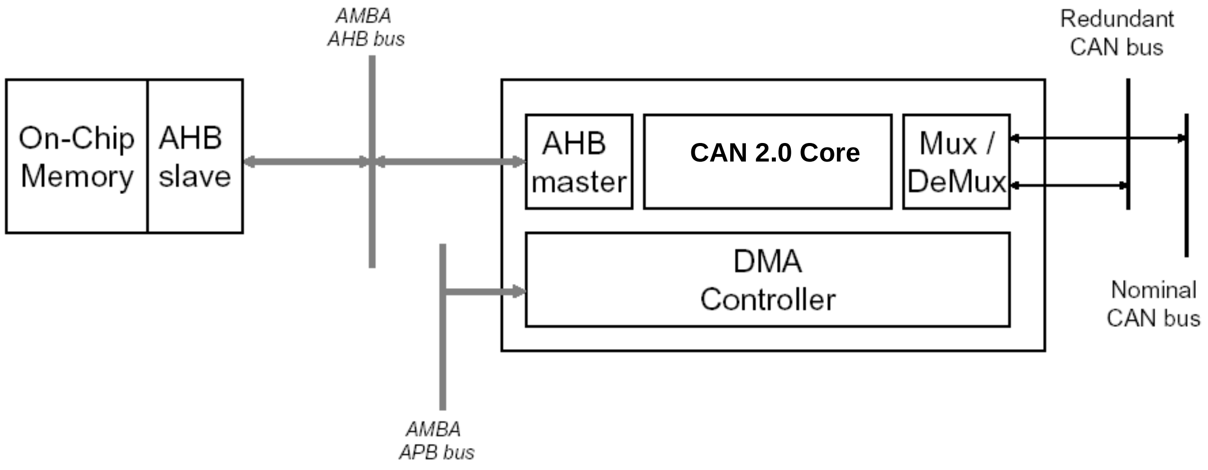


Figure 133. Block diagram of the internal structure of the GRCAN.

36.1.1 Function

The core implements the following functions:

- CAN protocol
- Message transmission
- Message filtering and reception

- SYNC message reception
- Status and monitoring
- Interrupt generation
- Redundancy selection

36.1.2 Interfaces

The core provides the following external and internal interfaces:

- CAN interface
- AMBA AHB master interface, with sideband signals as per [GRLIB] including:
    - cacheability information
    - interrupt bus
    - configuration information
    - diagnostic information
- AMBA APB slave interface, with sideband signals as per [GRLIB] including:
    - interrupt bus
    - configuration information
    - diagnostic information

36.1.3 Hierarchy

The CAN controller core can be partitioned in the following hierarchical elements:

- CAN 2.0 Core
- Redundancy Multiplexer / De-multiplexer
- Direct Memory Access controller
- AMBA APB slave
- AMBA AHB master

36.2 Interface

The external interface towards the CAN bus features two redundant pairs of transmit output and receive input (i.e. 0 and 1).

The active pair (i.e. 0 or 1) is bselectable by means of a configuration register bit. Note that all reception and transmission is made over the active pair.

For each pair, there is one enable output (i.e. 0 and 1), each being individually programmable. Note that the enable outputs can be used for enabling an external physical driver. Note that both pairs can be enabled simultaneously. Note that the polarity for the enable/inhibit inputs on physical interface drivers differs, thus the meaning of the enable output is undefined.

Redundancy is implemented by means of Selective Bus Access, as specified in [CANWG]. Note that the active pair selection above provides means to meet this requirement.

36.3 Protocol

The CAN protocol is based on a CAN 2.0 controller VHDL core. The CAN controller complies with [CANSTD], except for the overload frame generation.

Note that there are three different CAN types generally defined:

- 2.0A, which considers 29 bit ID messages as an error

- 2.0B Passive, which ignores 29 bit ID messages
- 2.0B Active, which handles 11 and 29 bit ID messages

Only 2.0B Active is implemented.

36.4 Status and monitoring

The CAN interface incorporates status and monitoring functionalities. This includes:

- Transmitter active indicator
- Bus-Off condition indicator
- Error-Passive condition indicator
- Over-run indicator
- 8-bit Transmission error counter
- 8-bit Reception error counter

The status is available via a register and is also stored in a circular buffer for each received message.

36.5 Transmission

The transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

36.5.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to the CAN controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

Each CAN message occupies 4 consecutive 32-bit words in memory. Each CAN message is aligned to 4 words address boundaries (i.e. the 4 least significant byte address bits are zero for the first word in a CAN message).

The size of the buffer is defined by the CanTxSIZE.SIZE field, specifying the number of CAN messages * 4 that fit in the buffer.

E.g. CanTxSIZE.SIZE =2 means 8 CAN messages fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one message position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanTxSIZE.SIZE =2 means that 7 CAN messages fit in the buffer at any given time.

36.5.2 Write and read pointers

The write pointer (CanTxWR.WRITE) indicates the position+1 of the last CAN message written to the buffer. The write pointer operates on number of CAN messages, not on absolute or relative addresses.

The read pointer (CanTxRD.READ) indicates the position+1 of the last CAN message read from the buffer. The read pointer operates on number of CAN messages, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN messages available in the buffer for transmission. The difference is calculated using the buffer size, specified by the CanTx-SIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE=2 and CanTxRD.READ=0.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =0 and CanTxRD.READ =6.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =1 and CanTxRD.READ =7.
- There are 2 CAN messages available for transmit when CanTxSIZE.SIZE=2, CanTxWR.WRITE =5 and CanTxRD.READ =3.

When a CAN message has been successfully transmitted, the read pointer (CanTxRD.READ) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the write pointer CanTxWR.WRITE and read pointer CanTxRD.READ are equal, there are no CAN messages available for transmission.

36.5.3 Location

The location of the circular buffer is defined by a base address (CanTxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

36.5.4 Transmission procedure

When the channel is enabled (CanTxCTRL.ENABLE=1), as soon as there is a difference between the write and read pointer, a message transmission will be started. Note that the channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the message transmission to start prematurely.

A message transmission will begin with a fetch of the complete CAN message from the circular buffer to a local fetch-buffer in the CAN controller. After a successful data fetch, a transmission request will be forwarded to the CAN core. If there is at least an additional CAN message available in the circular buffer, a prefetch of this CAN message from the circular buffer to a local prefetch-buffer in the CAN controller will be performed. The CAN controller can thus hold two CAN messages for transmission: one in the fetch buffer, which is fed to the CAN core, and one in the prefetch buffer.

After a message has been successfully transmitted, the prefetch-buffer contents are moved to the fetch buffer (provided that there is message ready). The read pointer (CanTxRD.READ) is automatically incremented after a successful transmission, i.e. after the fetch-buffer contents have been transmitted, taking wrap around effects of the circular buffer into account. If there is at least an additional CAN message available in the circular buffer, a new prefetch will be performed.

If the write and read pointers are equal, no more prefetches and fetches will be performed, and transmission will stop.

If the single shot mode is enabled for the transmit channel (CanTxCTRL.SINGLE=1), any message for which the arbitration is lost, or failed for some other reason, will lead to the disabling of the channel (CanTxCTRL.ENABLE=0), and the message will not be put up for re-arbitration.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the Tx, TxEmpty and TxIrq which are issued on the successful transmission of a message, when all messages have been transmitted successfully and when a predefined number of messages have been transmitted successfully. The TxLoss interrupt is issued whenever transmission arbitration has been lost, could also be caused by a communications error. The TxSync interrupt is issued when a message matching the SYNC Code Filter Register.SYNC and SYNC Mask Filter Register.MASK registers is successfully transmitted. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

36.5.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanTxADDR.ADDR) field.

While the channel is disabled, the read pointer (CanTxRD.READ) can be changed to an arbitrary value pointing to the first message to be transmitted, and the write pointer (CanTxWR.WRITE) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

36.5.6 AMBA AHB error

Definition:

- a message fetch occurs when no other messages is being transmitted
- a message prefetch occurs when a previously fetched message is being transmitted
- the local fetch buffer holds the message being fetched
- the local prefetch buffer holds the message being prefetched
- the local fetch buffer holds the message being transmitted by the CAN core
- a successfully prefetched message is copied from the local prefetch buffer to the local fetch buffer when that buffer is freed after a successful transmission.

An AHB error response occurring on the AMBA AHB bus while a CAN message is being fetched will result in a TxAHBErr interrupt.

If the CanCONF.ABORT bit is set to 0b, the channel causing the AHB error will skip the message being fetched from memory and will increment the read pointer. No message will be transmitted.

If the CanCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (CanTx-CTRL.ENABLE is cleared automatically to 0 b). The read pointer can be used to determine which message caused the AHB error. Note that it could be any of the four word accesses required to read a message that caused the AHB error.

If the CanCONF.ABORT bit is set to 1b, all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs, as indicated by the CanSTAT.AHBErr bit being 1b. The accesses will be disabled until the CanSTAT register is read, and automatically clearing bit CanSTAT.AHBErr.

An AHB error response occurring on the AMBA AHB bus while a CAN message is being prefetched will not cause an interrupt, but will stop the ongoing prefetch and further prefetch will be prevented temporarily. The ongoing transmission of a CAN message from the fetch buffer will not be affected. When the fetch buffer is freed after a successful transmission, a new fetch will be initiated, and if this fetch results in an AHB error response occurring on the AMBA AHB bus, this will be handled as for the case above. If no AHB error occurs, prefetch will be allowed again.

36.5.7 Enable and disable

When an enabled transmit channel is disabled (CanTxCTRL.ENABLE=0b), any ongoing CAN message transfer request will not be aborted until a CAN bus arbitration is lost or the message has been sent successfully. If the message is sent successfully, the read pointer (CanTxRD.READ) is automatically incremented. Any associated interrupts will be generated.

The progress of the any ongoing access can be observed via the CanTxCTRL.ONGOING bit. The CanTxCTRL.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers). It is also possible to wait for the Tx and TxLoss interrupts described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

Priority inversion is handled by disabling the transmitting channel, i.e. setting CanTxCTRL.ENABLE=0b as described above, and observing the progress, i.e. reading via the CanTxCTRL.ONGOING bit as described above. When the transmit channel is disabled, it can be re-configured and a higher priority message can be transmitted. Note that the single shot mode does not require the channel to be disabled, but the progress should still be observed as above.

No message transmission is started while the channel is not enabled.

36.5.8 Interrupts

During transmission several interrupts can be generated:

•	TxLoss:	Message arbitration lost for transmit (could be caused by communcations error, as indicated by other interrupts as well)
•	TxErrCntr:	Error counter incremented for transmit
•	TxSync:	Synchronization message transmitted
•	Tx:	Successful transmission of one message
•	TxEmpty:	Successful transmission of all messages in buffer
•	TxIrq:	Successful transmission of a predefined number of messages
•	TxAHBErr:	AHB access error during transmission
•	Off:	Bus-off condition
•	Pass:	Error-passive condition

The Tx, TxEmpty and TxIrq interrupts are only generated as the result of a successful message transmission, after the CanTxRD.READ pointer has been incremented.

36.6 Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

36.6.1 Circular buffer

The receive channel operates on a circular buffer located in memory external to the CAN controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

Each CAN message occupies 4 consecutive 32-bit words in memory. Each CAN message is aligned to 4 words address boundaries (i.e. the 4 least significant byte address bits are zero for the first word in a CAN message).

The size of the buffer is defined by the CanRxSIZE.SIZE field, specifying the number of CAN messages * 4 that fit in the buffer.

E.g. CanRxSIZE.SIZE=2 means 8 CAN messages fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one message position in the buffer empty. This is to simplify wrap-around condition checking.

E.g. CanRxSIZE.SIZE=2 means that 7 CAN messages fit in the buffer at any given time.

36.6.2 Write and read pointers

The write pointer (CanRxWR.WRITE) indicates the position+1 of the last CAN message written to the buffer. The write pointer operates on number of CAN messages, not on absolute or relative addresses.

The read pointer (CanRxRD.READ) indicates the position+1 of the last CAN message read from the buffer. The read pointer operates on number of CAN messages, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of CAN message positions available in the buffer for reception. The difference is calculated using the buffer size, specified by the CanRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE=2 and CanRxRD.READ=0.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =0 and CanRxRD.READ=6.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =1 and CanRxRD.READ=7.
- There are 2 CAN messages available for read-out when CanRxSIZE.SIZE=2, CanRxWR.WRITE =5 and CanRxRD.READ=3.

When a CAN message has been successfully received and stored, the write pointer (CanRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the read pointer CanRxRD.READ equals (CanRxWR.WRITE+1) modulo (CanRxSIZE.SIZE*4), there is no space available for receiving another CAN message.

The error behavior of the CAN core is according to the CAN standard, which applies to the error counter, buss-off condition and error-passive condition.

36.6.3 Location

The location of the circular buffer is defined by a base address (CanRxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

36.6.4 Reception procedure

When the channel is enabled (CanRxCTRL.ENABLE=1), and there is space available for a message in the circular buffer (as defined by the write and read pointer), as soon as a message is received by the CAN core, an AMBA AHB store access will be started. The received message will be temporarily stored in a local store-buffer in the CAN controller. Note that the channel should not be enabled until the write and read pointers are configured, to avoid the message reception to start prematurely

After a message has been successfully stored the CAN controller is ready to receive a new message. The write pointer (CanRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are the Rx, RxFull and RxIrq which are issued on the successful reception of a message, when the message buffer has been successfully filled and when a predefined number of messages have been received successfully. The RxMiss interrupt is issued whenever a message has been received but does not match a message filtering setting, i.e. neither for the receive channel nor for the SYNC message described hereafter.

The RxSync interrupt is issued when a message matching the SYNC Code Filter Register.SYNC and SYNC Mask Filter Register.MASK registers has been successfully received. Additional interrupts are provided to signal error conditions on the CAN bus and AMBA bus.

36.6.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (CanRxADDR.ADDR) field.

While the channel is disabled, the write pointer (CanRxWR.WRITE) can be changed to an arbitrary value pointing to the first message to be received, and the read pointer (CanRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

36.6.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while a CAN message is being stored will result in an RxAHBErr interrupt.

If the CanCONF.ABORT bit is set to 0b, the channel causing the AHB error will skip the received message, not storing it to memory. The write pointer will be incremented.

If the CanCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (CanRxCTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which message caused the AHB error. Note that it could be any of the four word accesses required to writ a message that caused the AHB error.

If the CanCONF.ABORT bit is set to 1b, all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs, as indicated by the CanSTAT.AHBErr bit being 1b. The accesses will be disabled until the CanSTAT register is read, and automatically clearing bit CanSTAT.AHBErr.

36.6.7 Enable and disable

When an enabled receive channel is disabled (CanRxCTRL.ENABLE=0b), any ongoing CAN message storage on the AHB bus will not be aborted, and no new message storage will be started. Note that only complete messages can be received from the CAN core. If the message is stored successfully, the write pointer (CanRxWR.WRITE) is automatically incremented. Any associated interrupts will be generated.

The progress of the any ongoing access can be observed via the CanRxCTRL.ONGOING bit. The CanRxCTRL.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers). It is also possible to wait for the Rx and RxMiss interrupts described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers.

No message reception is performed while the channel is not enabled

36.6.8 Interrupts

During reception several interrupts can be generated:

• RxMiss:	Message filtered away for receive
• RxErrCntr:	Error counter incremented for receive
• RxSync:	Synchronization message received
• Rx:	Successful reception of one message
• RxFull:	Successful reception of all messages possible to store in buffer

• RxIrq:	Successful reception of a predefined number of messages
• RxAHBErr:	AHB access error during reception
• OR:	Over-run during reception
• OFF:	Bus-off condition
• PASS:	Error-passive condition

The Rx, RxFull and RxIrq interrupts are only generated as the result of a successful message reception, after the CanRxWR.WRITE pointer has been incremented.

The OR interrupt is generated when a message is received while a previously received message is still being stored. A full circular buffer will lead to OR interrupts for any subsequently received messages. Note that the last message stored which fills the circular buffer will not generate an OR interrupt. The overrun is also reported with the CanSTAT.OR bit, which is cleared when reading the register.

The error behavior of the CAN core is according to the CAN standard, which applies to the error counter, buss-off condition and error-passive condition.

36.7 Global reset and enable

When the CanCTRL.RESET bit is set to 1b, a reset of the core is performed. The reset clears all the register fields to their default values. Any ongoing CAN message transfer request will be aborted, potentially violating the CAN protocol.

When the CanCTRL.ENABLE bit is cleared to 0b, the CAN core is reset and the configuration bits CanCONF.SCALER, CanCONF.PS1, CanCONF.PS2, CanCONF.RSJ and CanCONF.BPR may be modified. When disabled, the CAN controller will be in sleep mode not affecting the CAN bus by only sending recessive bits. Note that the CAN core requires that 10 recessive bits are received before any reception or transmission can be initiated. This can be caused either by no unit sending on the CAN bus, or by random bits in message transfers.

36.8 Interrupt

Three interrupts are implemented by the CAN interface:

Index:	Name:	Description:
0	IRQ	Common output from interrupt handler
1	TxSYNC	Synchronization message transmitted (optional)
2	RxSYNC	Synchronization message received (optional)

The interrupts are configured by means of the *pirq* VHDL generic and the *singleirq* VHDL generic.

36.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 270.*GRCAN registers

APB address offset	Register
16#000#	Configuration Register
16#004#	Status Register
16#008#	Control Register
16#018#	SYNC Mask Filter Register
16#01C#	SYNC Code Filter Register
16#100#	Pending Interrupt Masked Status Register
16#104#	Pending Interrupt Masked Register
16#108#	Pending Interrupt Status Register
16#10C#	Pending Interrupt Register
16#110#	Interrupt Mask Register
16#114#	Pending Interrupt Clear Register
16#200#	Transmit Channel Control Register
16#204#	Transmit Channel Address Register
16#208#	Transmit Channel Size Register
16#20C#	Transmit Channel Write Register
16#210#	Transmit Channel Read Register
16#214#	Transmit Channel Interrupt Register
16#300#	Receive Channel Control Register
16#304#	Receive Channel Address Register
16#308#	Receive Channel Size Register
16#30C#	Receive Channel Write Register
16#310#	Receive Channel Read Register
16#314#	Receive Channel Interrupt Register
16#318#	Receive Channel Mask Register
16#31C#	Receive Channel Code Register

36.9.1 Configuration Register [CanCONF] R/W

*Table 271.*Configuration Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SCALER								PS1				PS2			
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RSJ					BPR					Silent	Selection	Enable 1	Enable 0	Abort

31-24:	SCALER	Prescaler setting, 8-bit: system clock / (SCALER +1)
23-20:	PS1	Phase Segment 1, 4-bit: (valid range 1 to 15)
19-16:	PS2	Phase Segment 2, 4-bit: (valid range 2 to 8)
14-12:	RSJ	ReSynchronization Jumps, 3-bit: (valid range 1 to 4)
9:8:	BPR	Baud rate, 2-bit: 00b = system clock / (SCALER +1) / 1 01b = system clock / (SCALER +1) / 2 10b = system clock / (SCALER +1) / 4

		11b = system clock / (SCALER +1) / 8
4:	SILENT	Listen only to the CAN bus, send recessive bits.
3:	SELECTION	Selection receiver input and transmitter output: Select receive input 0 as active when 0b, Select receive input 1 as active when 1b Select transmit output 0 as active when 0b, Select transmit output 1 as active when 1b
2:	ENABLE1	Set value of output 1 enable
1:	ENABLE0	Set value of output 0 enable
0:	ABORT	Abort transfer on AHB ERROR

All bits are cleared to 0 at reset.

Note that constraints on PS1, PS2 and RSJ are defined as:

- PS1 +1 >= PS2
- PS1 > PS2
- PS2 >= RSJ

Note that CAN standard TSEG1 is defined by PS1+1.

Note that CAN standard TSEG2 is defined by PS2.

Note that the SCALER setting defines the CAN time quantum, together with the BPR setting:

system clock / (SCALER+1) / BPR

where SCALER is in range 0 to 255, and the resulting division factor due to BPR is 1, 2, 4 or 8.

Note that the resulting bit rate is:

system clock / (SCALER+1) / BPR * (1+ PS1+1 + PS2)

where PS1 is in the range 1 to 15, and PS2 is in the range 2 to 8.

Note that RSJ defines the number of allowed re-synchronization jumps according to the CAN standard, being in the range 1 to 4.

Note that the transmit or receive channel active during the AMBA AHB error is disabled if the ABORT bit is set to 1b. Note that all accesses to the AMBA AHB bus will be disabled after an AMBA AHB error occurs while the ABORT bit is set to 1b. The accesses will be disabled until the CanSTAT register is read.

36.9.2 Status Register [CanSTAT] R

*Table 272.*Status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TxChannels				RxChannels				TxErrCntr							
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RxErrCntr											Active	AHB Err	OR	Off	Pass

31-28:	TxChannels	Number of TxChannels -1, 4-bit
27-24:	RxChannels	Number of RxChannels -1, 4-bit
23-16:	TxErrCntr	Transmission error counter, 8-bit
15-8:	RxErrCntr	Reception error counter, 8-bit
4:	ACTIVE	Transmission ongoing
3:	AHBErr	AMBA AHB master interface blocked due to previous AHB error
2:	OR	Overrun during reception
1:	OFF	Bus-off condition
0:	PASS	Error-passive condition

All bits are cleared to 0 at reset.

The OR bit is set if a message with a matching ID is received and cannot be stored via the AMBA AHB bus, this can be caused by bandwidth limitations or when the circular buffer for reception is already full.

The OR and AHBErr status bits are cleared when the register has been read.

Note that TxErrCntr and RxErrCntr are defined according to CAN protocol.

Note that the AHBErr bit is only set to 1b if an AMBA AHB error occurs while the CanCONF.ABORT bit is set to 1b.

36.9.3 Control Register [CanCTRL] R/W

*Table 273.*Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
														Reset	Enable

1:	RESET	Reset complete core when 1
0:	ENABLE	Enable CAN controller, when 1. Reset CAN controller, when 0

All bits are cleared to 0 at reset.

Note that RESET is read back as 0b.

Note that ENABLE should be cleared to 0b to while other settings are modified, ensuring that the CAN core is properly synchronized.

Note that when ENABLE is cleared to 0b, the CAN interface is in sleep mode, only outputting recessive bits.

Note that the CAN core requires that 10 recessive bits be received before receive and transmit operations can begin.

36.9.4 SYNC Code Filter Register [CanCODE] R/W

*Table 274.*SYNC Code Filter Register

31	30	29	28 … 0
			SYNC

28-0:	SYNC	Message Identifier

All bits are cleared to 0 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

36.9.5 SYNC Mask Filter Register [CanMASK] R/W

*Table 275.*SYNC Mask Filter Register

31	30	29	28 … 0
			MASK

28-0:	MASK	Message Identifier

All bits are set to 1 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

A RxSYNC message ID is matched when:

((Received-ID XOR CanCODE.SYNC) AND CanMASK.MASK) = 0

A TxSYNC message ID is matched when:

((Transmitted-ID XOR CanCODE.SYNC) AND CanMASK.MASK) = 0

36.9.6 Transmit Channel Control Register [CanTxCTRL] R/W

*Table 276.*Transmit Channel Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													Sin-gle	Ongoing	Enable

2:	SINGLE	Single shot mode
1:	ONGOING	Transmission ongoing
0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that if the SINGLE bit is 1b, the channel is disabled (i.e. the ENABLE bit is cleared to 0b) if the arbitration on the CAN bus is lost.

Note that in the case an AHB bus error occurs during an access while fetching transmit data, and the CanCONF.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing message transmission is not aborted, unless the CAN arbitration is lost or communication has failed.

Note that the ONGOING bit being 1b indicates that message transmission is ongoing and that configuration of the channel is not safe.

36.9.7 Transmit Channel Address Register [CanTxADDR] R/W

*Table 277.*Transmit Channel Address Register

31 — 10	9 — 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

36.9.8 Transmit Channel Size Register [CanTxSIZE] R/W

*Table 278.*Transmit Channel Size Register

31 — 21	20 — 6	5 — 0
	SIZE	

20-6:	SIZE	The size of the circular buffer is SIZE*4 messages

All bits are cleared to 0 at reset.

Valid SIZE values are between 0 and 16384.

Note that each message occupies four 32-bit words.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 messages can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field may be made configurable by means of a VHDL generic. In this case it should be set to 16-1 bits width.

36.9.9 Transmit Channel Write Register [CanTxWR] R/W

*Table 279.*Transmit Channel Write Register

31 — 20	19 — 4	3 — 0
	WRITE	

19-4:	WRITE	Pointer to last written message +1

All bits are cleared to 0 at reset.

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last message to transmit.

Note that it is not possible to fill the buffer. There is always one message position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

36.9.10 Transmit Channel Read Register [CanTxRD] R/W

*Table 280.*Transmit Channel Read Register

31 — 20	19 — 4	3 — 0
	READ	

19-4:	READ	Pointer to last read message +1

All bits are cleared to 0 at reset.

The READ field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last message transmitted.

Note that the READ field can be use to read out the progress of a transfer.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the READ field can be automatically incremented even if the transmit channel has been disabled, since the last requested transfer is not aborted until CAN bus arbitration is lost.

When the Transmit Channel Read Pointer catches up with the Transmit Channel Write Register, an interrupt is generated (TxEmpty). Note that this indicates that all messages in the buffer have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

36.9.11 Transmit Channel Interrupt Register [CanTxIRQ] R/W

*Table 281.*Transmit Channel Interrupt Register

31 20	19 4	3 0
	IRQ	

19-4:	IRQ	Interrupt is generated when CanTxRD.READ=IRQ, as a consequence of a message transmission

All bits are cleared to 0 at reset.

Note that this indicates that a programmed number of messages have been transmitted.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

36.9.12 Receive Channel Control Register [CanRxCTRL] R/W

*Table 282.*Receive Channel Control Register

31 2	1	0
	OnGoing	Enable

1:	ONGOING	Reception ongoing (read-only)
0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that in the case an AHB bus error occurs during an access while fetching transmit data, and the CanCONF.ABORT bit is 1b, then the ENALBE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing message reception is not aborted

Note that the ONGOING bit being 1b indicates that message reception is ongoing and that configuration of the channel is not safe.

36.9.13 Receive Channel Address Register [CanRxADDR] R/W

*Table 283.*Receive Channel Address Register

31 10	9 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

36.9.14 Receive Channel Size Register [CanRxSIZE] R/W

*Table 284.*Receive Channel Size Register

31 21	20 6	5 0
	SIZE	

20-6:	SIZE	The size of the circular buffer is SIZE*4 messages

All bits are cleared to 0 at reset.

Valid SIZE values are between 0 and 16384.

Note that each message occupies four 32-bit words.

Note that the resulting behavior of invalid SIZE values is undefined.

Note that only (SIZE*4)-1 messages can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field may be made configurable by means of a VHDL generic. In this case it should be set to 16-1 bits width.

36.9.15 Receive Channel Write Register [CanRxWR] R/W

*Table 285.*Receive Channel Write Register

31 20	19 4	3 0
	WRITE	

19-4:	WRITE	Pointer to last written message +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last message received.

Note that the WRITE field can be use to read out the progress of a transfer.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the receive channel is not enabled.

36.9.16 Receive Channel Read Register [CanRxRD] R/W

*Table 286.*Receive Channel Read Register

31 20	19 4	3 0
	READ	

19-4:	READ	Pointer to last read message +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last message that has been read out.

Note that it is not possible to fill the buffer. There is always one message position in buffer unused. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

36.9.17 Receive Channel Interrupt Register [CanRxIRQ] R/W

*Table 287.*Receive Channel Interrupt Register

31 20	19 4	3 0
	IRQ	

19-4:	IRQ	Interrupt is generated when CanRxWR.WRITE=IRQ, as a consequence of a message reception

All bits are cleared to 0 at reset.

Note that this indicates that a programmed number of messages have been received.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

36.9.18 Receive Channel Mask Register [CanRxMASK] R/W

*Table 288.*Receive Channel Mask Register

31	30	29	28	0
			AM	

28-0:	AM	Acceptance Mask, bits set to 1b are taken into account in the comparison between the received message ID and the CanRxCODE.AC field

All bits are set to 1 at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

36.9.19 Receive Channel Code Register [CanRxCODE] R/W

*Table 289.*Receive Channel Code Register

31	30	29	28	0
			AC	

28-0:	AC	Acceptance Code, used in comparison with the received message

All bits are cleared to 0at reset.

Note that Base ID is bits 28 to 18 and Extended ID is bits 17 to 0.

A message ID is matched when:

((Received-ID XOR CanRxCODE.AC) AND CanRxMASS.AM) = 0

36.9.20 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialise the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

• Pending Interrupt Masked Status Register	[CanPIMSR]	R
• Pending Interrupt Masked Register	[CanPIMR]	R
• Pending Interrupt Status Register	[CanPISR]	R
• Pending Interrupt Register	[CanPIR]	R/W
• Interrupt Mask Register	[CanIMR]	R/W
• Pending Interrupt Clear Register	[CanPICR]	W

*Table 290.*Interrupt registers

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
															Tx Loss

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rx Miss	Tx Err Cntr	Rx Err Cntr	Tx Syn c	Rx Syn c	Tx	Rx	Tx Emp ty	Rx Full	Tx IRQ	Rx IRQ	Tx AH B Err	Rx AH B Err	OR	Off	Pass

16:	TxLoss	Message arbitration lost during transmission (could be caused by communcations error, as indicated by other interrupts as well)
15:	RxMiss	Message filtered away during reception
14:	TxErrCntr	Transmission error counter incremented
13:	RxErrCntr	Reception error counter incremented
12:	TxSync	Synchronization message transmitted
11:	RxSync	Synchronization message received
10:	Tx	Successful transmission of message
9:	Rx	Successful reception of message
8:	TxEmpty	Successful transmission of all messages in buffer
7:	RxFull	Successful reception of all messages possible to store in buffer
6:	TxIRQ	Successful transmission of a predefined number of messages
5:	RxIRQ	Successful reception of a predefined number of messages
4:	TxAHBErr	AHB error during transmission
3:	RxAHBErr	AHB error during reception
2:	OR	Over-run during reception
1:	OFF	Bus-off condition
0:	PASS	Error-passive condition

All bits in all interrupt registers are reset to 0b after reset.

Note that the TxAHBErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis. The interrupt generation is independent of the Can-CONF.ABORT field setting.

Note that the RxAHBErr interrupt is generated in such way that the corresponding read and write pointers are valid for failure analysis. The interrupt generation is independent of the Can-CONF.ABORT field setting.

36.10 Memory mapping

The CAN message is represented in memory as shown in table 291.

*Table 291.*CAN message representation in memory.

AHB addr																
0x0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	IDE	RTR	-	bID											eID	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	eID															
0x4	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	DLC				-	-	-	-	TxErrCntr							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RxErrCntr								-	-	-	-	Ahb Err	OR	Off	Pass
0x8	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 0 (first transmitted)								Byte 1							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 2								Byte 3							
0xC	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	Byte 4								Byte 5							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Byte 6								Byte 7 (last transmitted)							

Values:	Levels according to CAN standard:	1b is recessive, 0b is dominant	
Legend:	Naming and number in according to CAN standard		
IDE	Identifier Extension:	1b for Extended Format, 0b for Standard Format	
RTR	Remote Transmission Request:	1b for Remote Frame, 0b for Data Frame	
bID	Base Identifier		
eID	Extended Identifier		
DLC	Data Length Code, according to CAN standard:		
		0000b	0 bytes
		0001b	1 byte
		0010b	2 bytes
		0011b	3 bytes
		0100b	4 bytes
		0101b	5 bytes
		0110b	6 bytes
		0111b	7 bytes

		1000b	8 bytes
		OTHERS	illegal
TxErrCntr	Transmission Error Counter		
RxErrCntr	Reception Error Counter		
AHBErr	AHB interface blocked due to AHB Error when 1b		
OR	Reception Over run when 1b		
OFF	Bus Off mode when 1b		
PASS	Error Passive mode when 1b		
Byte 00 to 07	Transmit/Receive data, Byte 00 first Byte 07 last		

36.11 Vendor and device identifiers

The module has vendor identfier 0x01 (Gaisler Research) and device identfier 0x03D. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

36.12 Configuration options

Table 292 shows the configuration options of the core (VHDL generics).

*Table 292.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRCAN.	0 - NAHBIRQ-1	0
singleirq	Implement only one common interrupt	0 - 1	0
txchannels	Number of transmit channels	1 - 1	1
rxchannels	Number of receive channels	1 - 1	1
ptrwidth	Width of message pointers	4 - 16	16

36.13 Signal descriptions

Table 293 shows the interface signals of the core (VHDL ports).

*Table 293.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
CANI	Rx[1:0]	Input	Receive lines	-
CANO	Tx[1:0]	Output	Transmit lines	-
	En[1:0]		Transmit enables	-

* see GRLIB IP Library User’s Manual

36.14 Library dependencies

Table 294 shows the libraries used when instantiating the core (VHDL libraries).

*Table 294.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CAN	Signals, component	GRCAN component and signal declarations.

36.15 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  gaisler;
use      gaisler.can.all;

entity example is
   generic (
      padtech:       in    integer := 0);
   port (
      -- CAN interface
      cantx:         out   std_logic_vector(1 downto 0);
      canrx:         in    std_logic_vector(1 downto 0);
      canen:         out   std_logic_vector(1 downto 0);

...

   -- Signal declarations
   signal   rstn:          std_ulogic;
   signal   clk:           std_ulogic;

   signal   ahbmo:         ahb_mst_out_vector := (others => ahbm_none);
   signal   ahbmi:         ahb_mst_in_type;

   signal   apbi:          apb_slv_in_type;
   signal   apbo:          apb_slv_out_vector := (others => apb_none);

   signal   cani0:         can_in_type;
   signal   cano0:         can_out_type;

...

   -- Component instantiation
   grcan0: grcan
      generic map (
         hindex          => 1,
         pindex          => 1,
         paddr           => 16#00C",
         pmask           => 16#FFC",
         pirq            => 1,
         txchannels      => 1,
         rxchannels      => 1,
         ptrwidth        => 16)
      port map (
         rstn            => rstn,
         clk             => clk,
         apbi            => apbi,
         apbo            => apbo(1),
         ahbi            => ahbmi,
         ahbo            => ahbmo(1),
         cani            => cani0,
         cano            => cano0);
```

```
cantx0_pad : outpad
   generic map (tech => padtech) port map (cantx(0), cani0.tx(0));

canrx0_pad : inpad
   generic map (tech => padtech) port map (canrx(0), cani0.rx(0));

canen0_pad : outpad
   generic map (tech => padtech) port map (canen(0), cani0.en(0));

cantx1_pad : outpad
   generic map (tech => padtech) port map (cantx(1), cani0.tx(1));

canrx1_pad : inpad
   generic map (tech => padtech) port map (canrx(1), cani0.rx(1));

canen1_pad : outpad
   generic map (tech => padtech) port map (canen(1), cani0.en(1));
```

37 GRECC - Elliptic Curve Cryptography

37.1 Overview

Elliptic Curve Cryptography (ECC) is used as a public key mechanism. The computational burden that is inhibited by ECC is less than the one of RSA. ECC provides the same level of security as RSA but with a significantly shorter key length. ECC is well suited for application in mobile communication.

The GRECC core implements encryption and decryption for an elliptic curve based on 233-bit key and point lengths. The implemented curve is denoted as *sect233r1* or *B-233*.

The *sect233r1* elliptic curve domain parameters are specified in the “Standards for Efficient Cryptography (SEC) - SEC2: Recommended Elliptic Curve Domain Parameters” document. The document is established by the Standards for Efficient Cryptography Group (SECG).

The *B-233* elliptic curve domain parameters are specified in the “Digital Signature Standard (DSS)” document, Federal Information Processing Standards (FIPS) Publication 186-2. The document is established by the National Institute of Standards and Technology (NIST).

The GRECC can be used with algorithms such as:

- Elliptic Curve Digital Signature Algorithm DSA (ECDSA), which appears in FIPS 186-2, IEEE 1363-2000 and ISO/IEC 15946-2
- Elliptic Curve El Gamal Method (key exchange protocol)
- Elliptic Curve Diffie-Hellman (ECDH) (key agreement protocol)

The core provides the following internal AMBA APB slave interface, with sideband signals as per [GRLIB] including:

- interrupt bus
- configuration information
- diagnostic information

The core can be partition in the following hierarchical elements:

- Elliptic Curve Cryptography (ECC) core
- AMBA APB slave
- GRLIB plug&play wrapper

Note that the core can also be used without the GRLIB plug&play information.

37.2 Operation

Elliptic Curve Cryptography (ECC) is an asymmetric cryptographic approach (also known as public key cryptography) that applies different keys for encryption and decryption. The most expensive operation during both encryption and decryption is the elliptic curve point multiplication. Hereby, a point on the elliptic curve is multiplied with a long integer ($k*P$ multiplication). The bit sizes of the coordinates of the point $P=(x, y)$ and the factor k have a length of hundreds of bits.

In this implementation the key and the point lengths are 233 bit, so that for every key there are 8 write cycles necessary and for every point (consisting of x and y) there are 16 write cycles necessary. After at least 16700 clock cycles the result can be read out.

The key is input via eight registers. The input point P_{in}=(x, y) is written via eight registers for x and eight registers for y. After the last y input register is written, the encryption or decryption is started. The progress can be observed via the status register. When the operation is completed, an interrupt is generated. The output point P_{out}=(x, y) is then read out via eight registers for x and eight registers for y.

37.3 Advantages

The main operation in ECC is the k*P multiplication. One k*P multiplication requires about 1500 field multiplications in the base field, which is the most expensive base operation. The complexity of a field multiplication can be reduced by applying the Karatsuba method. Normally the Karatsuba approach is applied recursively. The GRECC core includes an iterative implementation of the Karatsuba method which allows to realize area efficient hardware accelerators for the *k*P* multiplication. Hardware accelerators which are realized applying an iterative approach need up to 60 per cent less area and about 30 per cent less energy per multiplication than the recursive variants.

37.4 Background

The Standards for Efficient Cryptography Group (SECG) was initiated by Certicom Corporation to address the difficulty vendors and users face when building and deploying interoperable security solutions. The SECG is a broad international coalition comprised of leading technology companies and key industry players in the information security industry. One of the goals is to enable the effective incorporation of Elliptic Curve Cryptographic (ECC) technology into these various cryptographic solutions.

The Standards for Efficient Cryptography Group (SECG) has develop two sets of documents. The first set, under the name SEC, specifies interoperable cryptographic technologies and solutions. The second set, Guidelines for Efficient Cryptography (GEC), provides background information on elliptic curve cryptography and recommendations for ECC parameter and curve selection.

The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted under the provisions of the Information Technology Management Reform Act.

This Digital Signature Standard (DSS) specifies a suite of algorithms which can be used to generate a digital signature. Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory. In addition, the recipient of signed data can use a digital signature in proving to a third party that the signature was in fact generated by the signatory. This is known as nonrepudiation since the signatory cannot, at a later time, repudiate the signature.

37.5 233-bit elliptic curve domain parameters

The core implements the 233-bit elliptic curve domain parameters *sect233r1*, or the equivalent *B-233*, which are verifiably random parameters. The following specification is established in “Standards for Efficient Cryptography (SEC) - SEC 2: Recommended Elliptic Curve Domain Parameters”. The verifiably random elliptic curve domain parameters over F_{2^m} are specified by the septuple T = (m; f (x); a; b; G; n; h) where m = 233 and the representation of $F_{2^{233}}$ is defined by:

$f(x) = x^{233}+x^{74}+1$

The curve E: $y^2+xy = x^3+ax^2+b$ over F_{2^m} is defined by:

a = 0000 00000000 00000000 00000000 00000000 00000000 00000000 00000001

b = 0066 647EDE6C 332C7F8C 0923BB58 213B333B 20E9CE42 81FE115F 7D8F90AD

The base point G in compressed form is:

G = 0300FA C9DFCBAC 8313BB21 39F1BB75 5FEF65BC 391F8B36 F8F8EB73 71FD558B

and in uncompressed form is:

G = 04 00FAC9DF CBAC8313 BB2139F1 BB755FEF 65BC391F 8B36F8F8

EB7371FD 558B0100 6A08A419 03350678 E58528BE BF8A0BEF F867A7CA

36716F7E 01F81052

Finally the order n of G and the cofactor are:

n = 0100 00000000 00000000 00000000 0013E974 E72F8A69 22031D26 03CFE0D7

h = 02

37.6 Throughput

The data throughput for the GRECC core is around 233/16700 bits per clock cycle, i.e. approximately 13.9 kbits per MHz.

The underlaying EEC core has been implemented in a dual crypto chip on 250 nm technology as depicted in the figure below. The throughput at 33 MHz operating frequency was 850 kbit/s, the power consumption was 56,8 mW, and the size was 27,6 kgates.

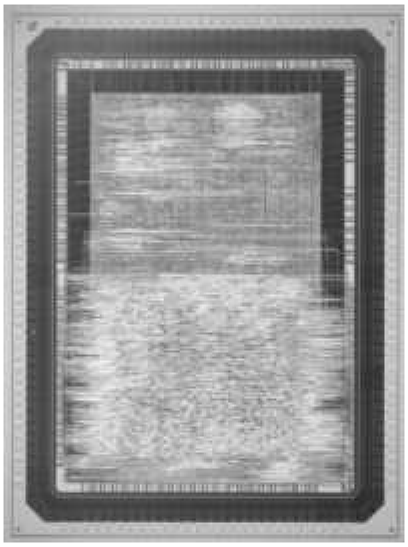


Figure 134. Dual Crypto Chip

37.7 Characteristics

The GRECC core has been synthesized for a Xilinx Virtex-2 XC2V6000-4 devices with the following results:

- LUTs: 12850 (19%)
- Frequency:93 MHz

37.8 Registers

The core is programmed through registers mapped into APB address space.

*Table 295.*GRECC registers

APB address offset	**Register**
16#020#	Key 0 Register
16#024#	Key 1 Register
16#028#	Key 2 Register
16#02C#	Key 3 Register
16#030#	Key 4 Register
16#034#	Key 5 Register
16#038#	Key 6 Register
16#03C#	Key 7 Register
16#040#	Point X Input 0 Register
16#044#	Point X Input 1 Register
16#048#	Point X Input 2 Register
16#04C#	Point X Input 3 Register
16#050#	Point X Input 4 Register
16#054#	Point X Input 5 Register
16#058#	Point X Input 6 Register
16#05C#	Point X Input 7 Register
16#060#	Point Y Input 0 Register
16#064#	Point Y Input 1 Register
16#068#	Point Y Input 2 Register
16#06C#	Point Y Input 3 Register
16#070#	Point Y Input 4 Register
16#074#	Point Y Input 5 Register
16#078#	Point Y Input 6 Register
16#07C#	Point Y Input 7 Register
16#0A0#	Point X Output 0 Register
16#0A4#	Point X Output 1 Register
16#0A8#	Point X Output 2 Register
16#0AC#	Point X Output 3 Register
16#0B0#	Point X Output 4 Register
16#0B4#	Point X Output 5 Register
16#0B8#	Point X Output 6 Register
16#0BC#	Point X Output 7 Register
16#0C0#	Point Y Output 0 Register
16#0C4#	Point Y Output 1 Register
16#0C8#	Point Y Output 2 Register
16#0CC#	Point Y Output 3 Register
16#0D0#	Point Y Output 4 Register
16#0D4#	Point Y Output 5 Register
16#0D8#	Point Y Output 6 Register
16#0DC#	Point Y Output 7 Register
16#0FC#	Status Register

37.8.1 Key 0 to 7 Registers (W)

*Table 296.*Key 0 Register (least significant)

31 0
KEY(31 downto 0)

*Table 297.*Key 1 Register

31 0
KEY(63 downto32)

*Table 298.*Key 2 Register

31 0
KEY(95 downto 64)

*Table 299.*Key 3 Register

31 0
KEY(127 downto 96)

*Table 300.*Key 4 Register

31 0
KEY(159 downto 128)

*Table 301.*Key 5 Register

31 0
KEY(191 downto 160)

*Table 302.*Key 6 Register

31 0
KEY(223 downto 192)

*Table 303.*Key 7 Register (most significant)

31 9	8 0
-	KEY(232 downto 224)

37.8.2 Point X Input 0 to 7 Registers (W)

*Table 304.*Point X Input 0 Register (least significant)

31 0
X(31 downto 0)

*Table 305.*Point X Input 1 Register

31 0
X(63 downto32)

*Table 306.*Point X Input 2 Register

31 0
X(95 downto 64)

*Table 307.*Point X Input 3 Register

31 … 0
X(127 downto 96)

*Table 308.*Point X Input 4 Register

31 … 0
X(159 downto 128)

*Table 309.*Point X Input 5 Register

31 … 0
X(191 downto 160)

*Table 310.*Point X Input 6 Register

31 … 0
X(223 downto 192)

*Table 311.*Point X Input 7 Register (most significant)

31 … 9	8 … 0
-	X(232 downto 224)

37.8.3 Point Y Input 0 to 7 Registers (W)

*Table 312.*Point Y Input 0 Register (least significant)

31	0
Y(31 downto 0)	

*Table 313.*Point Y Input 1 Register

31	0
Y(63 downto32)	

*Table 314.*Point Y Input 2 Register

31	0
Y(95 downto 64)	

*Table 315.*Point Y Input 3 Register

31	0
Y(127 downto 96)	

*Table 316.*Point Y Input 4 Register

31	0
Y(159 downto 128)	

*Table 317.*Point Y Input 5 Register

31	0
Y(191 downto 160)	

*Table 318.*Point Y Input 6 Register

31	0
Y(223 downto 192)	

*Table 319.*Point Y Input 7 Register (most significant)

31	9	8	0
-		Y(232 downto 224)	

The encryption or decryption operation is started when the Point Y Input 7 Register is written.

37.8.4 Point X Output 0 to 7 Registers (R)

*Table 320.*Point X Output 0 Register (least significant)

31	0
X(31 downto 0)	

*Table 321.*Point X Output 1 Register

31	0
X(63 downto32)	

*Table 322.*Point X Output 2 Register

31	0
X(95 downto 64)	

*Table 323.*Point X Output 3 Register

31	0
X(127 downto 96)	

*Table 324.*Point X Output 4 Register

31	0
X(159 downto 128)	

*Table 325.*Point X Output 5 Register

31	0
X(191 downto 160)	

*Table 326.*Point X Output 6 Register

31	0
X(223 downto 192)	

*Table 327.*Point X Output 7 Register (most significant)

31 9	8 0
-	X(232 downto 224)

37.8.5 Point Y Output 0 to 7 Registers (R)

*Table 328.*Point Y Output 0 Register (least significant)

31	0
Y(31 downto 0)	

*Table 329.*Point Y Output 1 Register

31	0
Y(63 downto32)	

*Table 330.*Point Y Output 2 Register

31	0
Y(95 downto 64)	

*Table 331.*Point Y Output 3 Register

31	0
Y(127 downto 96)	

*Table 332.*Point Y Output 4 Register

31	0
Y(159 downto 128)	

*Table 333.*Point Y Output 5 Register

31	0
Y(191 downto 160)	

*Table 334.*Point Y Output 6 Register

31	0
Y(223 downto 192)	

*Table 335.*Point Y Output 7 Register (most significant)

31	9	8	0
-		Y(232 downto 224)	

37.8.6 Status Register (R)

*Table 336.*Status Register

31	1	0
.		FSM

31-1:	-	Unused
0:	FSM	0 when ongoing, 1 when idle or ready

37.9 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x074. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

37.10 Configuration options

Table 337 shows the configuration options of the core (VHDL generics).

*Table 337.*Configuration options

Generic	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB BAR	0 - 16#FFF#	0
pmask	Mask field of the APB BAR	0 - 16#FFF#	16#FFC#
pirq	Interrupt line used by the GRECC	0 - NAHBIRQ-1	0

37.11 Signal descriptions

Table 338 shows the interface signals of the core (VHDL ports).

*Table 338.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
DEBUG[10:0]	N/A	Output	Debug information	-

* see GRLIB IP Library User's Manual

Note that the ECC core can also be used without the GRLIB plug&play information. The AMBA APB signals are then provided as IEEE Std_Logic_1164 compatible scalars and vectors.

37.12 Library dependencies

Table 339 shows libraries used when instantiating the core (VHDL libraries).

*Table 339.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	CRYPTO	Component	GRECC component declarations

37.13 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.crypto.all;
...
...
   signal debug: std_logic_vector(10 downto 0);
..
```

```
..
   grecc0: grecc
      generic map (
         pindex          => pindex,
         paddr           => paddr,
         pmask           => pmask,
         pirq            => pirq)
      port map (
         rstn            => rstn,
         clk             => clk,
         apbi            => apbi,
         apbo            => apbo(pindex),
         debug           => debug);
```

38 GRETH - Ethernet Media Access Controller (MAC) with EDCL support

38.1 Overview

Gaisler Research's Ethernet Media Access Controller (GRETH) provides an interface between an AMBA-AHB bus and an Ethernet network. It supports 10/100 Mbit speed in both full- and half-duplex. The AMBA interface consists of an APB interface for configuration and control and an AHB master interface which handles the dataflow. The dataflow is handled through DMA channels. There is one DMA engine for the transmitter and one for the receiver. Both share the same AHB master interface. The ethernet interface supports both the MII and RMII interfaces which should be connected to an external PHY. The GRETH also provides access to the MII Management interface which is used to configure the PHY.

Optional hardware support for the Ethernet Debug Communication Link (EDCL) protocol is also provided. This is an UDP/IP based protocol used for remote debugging.

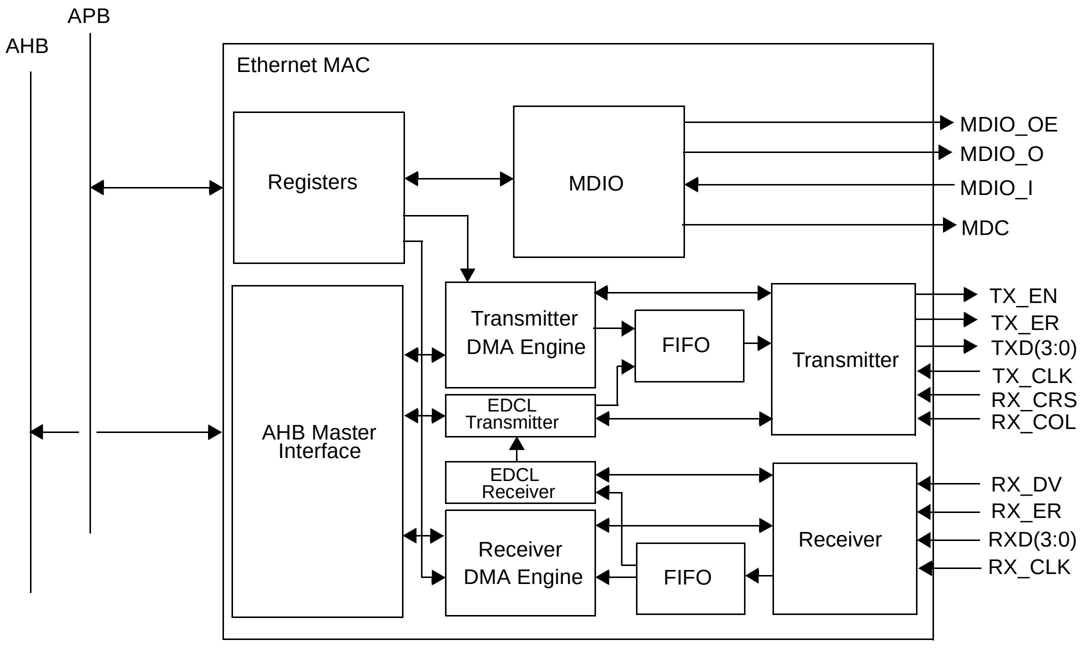


Figure 135. Block diagram of the internal structure of the GRETH.

38.2 Operation

38.2.1 System overview

The GRETH consists of 3 functional units: The DMA channels, MDIO interface and the optional Ethernet Debug Communication Link (EDCL).

The main functionality consists of the DMA channels which are used to transfer data between an AHB bus and an Ethernet network. There is one transmitter DMA channel and one Receiver DMA channel. The operation of the DMA channels is controlled through registers accessible through the APB interface.

The MDIO interface is used for accessing configuration and status registers in one or more PHYs connected to the MAC. The operation of this interface is also controlled through the APB interface.

The optional EDCL provides read and write access to an AHB bus through Ethernet. It uses the UDP, IP, ARP protocols together with a custom application layer protocol to accomplish this. The EDCL contains no user accessible registers and always runs in parallel with the DMA channels.

The Media Independent Interface (MII) is used for communicating with the PHY. There is an Ethernet transmitter which sends all data from the AHB domain on the Ethernet using the MII interface. Correspondingly, there is an Ethernet receiver which stores all data from the Ethernet on the AHB bus. Both of these interfaces use FIFOs when transferring the data streams. The GRETH also supports the RMII which uses a subset of the MII signals.

The EDCL and the DMA channels share the Ethernet receiver and transmitter.

38.2.2 Protocol support

The GRETH is implemented according to IEEE standard 802.3-2002. There is no support for the optional control sublayer and no multicast addresses can be assigned to the MAC. This means that packets with type 0x8808 (the only currently defined ctrl packets) are discarded.

38.2.3 Clocking

GRETH has three clock domains: The AHB clock, Ethernet receiver clock and the Ethernet transmitter clock. The ethernet transmitter and receiver clocks are generated by the external ethernet PHY, and are inputs to the core through the MII interface. The three clock domains are unrelated to each other and all signals crossing the clock regions are fully synchronized inside the core.

Both full-duplex and half-duplex operating modes are supported and both can be run in either 10 or 100 Mbit. The minimum AHB clock for 10 Mbit operation is 2.5 MHz, while 18 MHz is needed for 100 Mbit. Using a lower AHB clock than specified will lead to excessive packet loss.

38.3 Tx DMA interface

The transmitter DMA interface is used for transmitting data on an Ethernet network. The transmission is done using descriptors located in memory.

38.3.1 Setting up a descriptor.

A single descriptor is shown in table 340 and 341. The number of bytes to be sent should be set in the length field and the address field should point to the data. The address must be word-aligned. If the interrupt enable (IE) bit is set, an interrupt will be generated when the packet has been sent (this requires that the transmitter interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was transmitted successfully or not. The Wrap (WR) bit is also a control bit that should be set before transmission and it will be explained later in this section.

Table 340. GRETH transmit descriptor word 0 (address offset 0x0)

31 16	15	14	13	12	11	10 0
RESERVED	AL	UE	IE	WR	EN	LENGTH

31: 16	RESERVED
15	Attempt Limit Error (AL) - The packet was not transmitted because the maximum number of attempts was reached.
14	Underrun Error (UE) - The packet was incorrectly transmitted due to a FIFO underrun error.

Table 340. GRETH transmit descriptor word 0 (address offset 0x0)

13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes to be transmitted.

Table 341. GRETH transmit descriptor word 1 (address offset 0x4)

31	2	1 0
ADDRESS		RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the GRETH.

38.3.2 Starting transmissions

Enabling a descriptor is not enough to start a transmission. A pointer to the memory area holding the descriptors must first be set in the GRETH. This is done in the transmitter descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor.The first descriptor should be located at the base address and when it has been used by the GRETH the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the control register. This tells the GRETH that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

38.3.3 Descriptor handling after transmission

When a transmission of a packet has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the FIFO became empty before the packet was completely transmitted while the Alignment Error bit is set if more collisions occurred than allowed. The packet was successfully transmitted only if both of these bits are zero. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched.

The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the GRETH. There are three bits in the GRETH status register that hold transmission status. The Transmitter Error (TE) bit is set each time an transmission ended with an error (when at least one of the two status bits in the transmit descriptor has been set). The Transmitter Interrupt (TI) is set each time a transmission ended successfully.

The transmitter AHB error (TA) bit is set when an AHB error was encountered either when reading a descriptor or when reading packet data. Any active transmissions were aborted and the transmitter was disabled. The transmitter can be activated again by setting the transmit enable register.

38.3.4 Setting up the data for transmission

The data to be transmitted should be placed beginning at the address pointed by the descriptor address field. The GRETH does not add the Ethernet address and type fields so they must also be stored in the data buffer. The 4 B Ethernet CRC is automatically appended at the end of each packet. Each descriptor will be sent as a single Ethernet packet. If the size field in a descriptor is greater than 1514 B, the packet will not be sent.

38.4 Rx DMA interface

The receiver DMA interface is used for receiving data from an Ethernet network. The reception is done using descriptors located in memory.

38.4.1 Setting up descriptors

A single descriptor is shown in table 342 and 343. The address field should point to a word-aligned buffer where the received data should be stored. The GRETH will never store more than 1514 B to the buffer. If the interrupt enable (IE) bit is set, an interrupt will be generated when a packet has been received to this buffer (this requires that the receiver interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was received successfully or not. The Wrap (WR) bit is also a control bit that should be set before the descriptor is enabled and it will be explained later in this section.

Table 342. GRETH receive descriptor word 0 (address offset 0x0)

31 19	18	17	16	15	14	13	12	11	10 0
RESERVED	LE	OE	CE	FT	AE	IE	WR	EN	LENGTH

31: 19	RESERVED
18	Length error (LE) - The length/type field of the packet did not match the actual number of received bytes.
17	Overrun error (OE) - The frame was incorrectly received due to a FIFO overrun.
16	CRC error (CE) - A CRC error was detected in this frame.
15	Frame too long (FT) - A frame larger than the maximum size was received. The excessive part was truncated.
14	Alignment error (AE) - An odd number of nibbles were received.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when a packet has been received to this descriptor provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes received to this descriptor.

Table 343. GRETH receive descriptor word 1 (address offset 0x4)

31 2	1 0
ADDRESS	RES

31: 2	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.
1: 0	RESERVED

38.4.2 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH. This is done in the receiver descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate reception is to set the receiver enable bit in the control register. This will make the GRETH read the first descriptor and wait for an incoming packet.

38.4.3 Descriptor handling after reception

The GRETH indicates a completed reception by clearing the descriptor enable bit. The other control bits (WR, IE) are also cleared. The number of received bytes is shown in the length field. The parts of the Ethernet frame stored are the destination address, source address, type and data fields. Bits 17-14 in the first descriptor word are status bits indicating different receive errors. All four bits are zero after a reception without errors. The status bits are described in table 342.

Packets arriving that are smaller than the minimum Ethernet size of 64 B are not considered as a reception and are discarded. The current receive descriptor will be left untouched an used for the first packet arriving with an accepted size. The TS bit in the status register is set each time this event occurs.

If a packet is received with an address not accepted by the MAC, the IA status register bit will be set.

Packets larger than maximum size cause the FT bit in the receive descriptor to be set. The length field is not guaranteed to hold the correct value of received bytes. The counting stops after the word containing the last byte up to the maximum size limit has been written to memory.

The address word of the descriptor is never touched by the GRETH.

38.4.4 Reception with AHB errors

If an AHB error occurs during a descriptor read or data store, the Receiver AHB Error (RA) bit in the status register will be set and the receiver is disabled. The current reception is aborted. The receiver can be enabled again by setting the Receive Enable bit in the control register.

38.5 MDIO Interface

The MDIO interface provides access to PHY configuration and status registers through a two-wire interface which is included in the MII interface. The GRETH provided full support for the MDIO interface. If it is not needed in a design it can be removed with a VHDL generic.

The MDIO interface can be used to access from 1 to 32 PHY containing 1 to 32 16-bit registers. A read transfer i set up by writing the PHY and register addresses to the MDIO Control register and setting the read bit. This caused the Busy bit to be set and the operation is finished when the Busy bit is cleared. If the operation was successful the Linkfail bit is zero and the data field contains the read data. An unsuccessful operation is indicated by the Linkfail bit being set. The data field is undefined in this case.

A write operation is started by writing the 16-bit data, PHY address and register address to the MDIO Control register and setting the write bit. The operation is finished when the busy bit is cleared and it was successful if the Linkfail bit is zero.

38.6 Ethernet Debug Communication Link (EDCL)

The EDCL provides access to an on-chip AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol. The application layer protocol uses an ARQ algorithm to provide reliable AHB instruction transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus. The EDCL is optional and must be enabled with a generic.

38.6.1 Operation

The EDCL receives packets in parallel with the MAC receive DMA channel. It uses a separate MAC address which is used for distinguishing EDCL packets from packets destined to the MAC DMA channel. The EDCL also has an IP address which is set through generics. Since ARP packets use the Ethernet broadcast address, the IP-address must be used in this case to distinguish between EDCL ARP packets and those that should go to the DMA-channel. Packets that are determined to be EDCL packets are not processed by the receive DMA channel.

When the packets are checked to be correct, the AHB operation is performed. The operation is performed with the same AHB master interface that the DMA-engines use. The replies are automatically sent by the EDCL transmitter when the operation is finished. It shares the Ethernet transmitter with the transmitter DMA-engine but has higher priority.

38.6.2 EDCL protocols

The EDCL accepts Ethernet frames containing IP or ARP data. ARP is handled according to the protocol specification with no exceptions.

IP packets carry the actual AHB commands. The EDCL expects an Ethernet frame containing IP, UDP and the EDCL specific application layer parts. Table 344 shows the IP packet required by the EDCL. The contents of the different protocol headers can be found in TCP/IP literature.

*Table 344.*The IP packet expected by the EDCL.

Ethernet Header	IP Header	UDP Header	2 B Offset	4 B Control word	4 B Address	Data 0 - 242 4B Words	Ethernet CRC

The following is required for successful communication with the EDCL: A correct destination MAC address as set by the generics, an Ethernet type field containing 0x0806 (ARP) or 0x0800 (IP). The IP-address is then compared with the value determined by the generics for a match. The IP-header checksum and identification fields are not checked. There are a few restrictions on the IP-header fields. The version must be four and the header size must be 5 B (no options). The protocol field must always be 0x11 indicating a UDP packet. The length and checksum are the only IP fields changed for the reply.

The EDCL only provides one service at the moment and it is therefore not required to check the UDP port number. The reply will have the original source port number in both the source and destination fields. UDP checksum are not used and the checksum field is set to zero in the replies.

The UDP data field contains the EDCL application protocol fields. Table 345 shows the application protocol fields (data field excluded) in packets received by the EDCL. The 16-bit offset is used to align the rest of the application layer data to word boundaries in memory and can thus be set to any value. The R/W field determines whether a read (0) or a write(1) should be performed. The length

*Table 345.*The EDCL application layer fields in received frames.

16-bit Offset	14-bit Sequence number	1-bit R/W	10-bit Length	7-bit Unused

field contains the number of bytes to be read or written. If R/W is one the data field shown in table 344 contains the data to be written. If R/W is zero the data field is empty in the received packets. Table 346 shows the application layer fields of the replies from the EDCL. The length field is always zero for replies to write requests. For read requests it contains the number of bytes of data contained in the data field.

*Table 346.*The EDCL application layer fields in transmitted frames.

16-bit Offset	14-bit sequence number	1-bit ACK/NAK	10-bit Length	7-bit Unused

The EDCL implements a Go-Back-N algorithm providing reliable transfers. The 14-bit sequence number in received packets are checked against an internal counter for a match. If they do not match, no operation is performed and the ACK/NAK field is set to 1 in the reply frame. The reply frame contains the internal counter value in the sequence number field. If the sequence number matches, the operation is performed, the internal counter is incremented, the internal counter value is stored in the sequence number field and the ACK/NAK field is set to 0 in the reply. The length field is always set to 0 for ACK/NAK=1 frames. The unused field is not checked and is copied to the reply. It can thus be set to hold for example some extra identifier bits if needed.

38.6.3 EDCL IP and Ethernet address settings

The default value of the EDCL IP and MAC addresses are set by `ipaddrh, ipaddrl, macaddrh and macaddrl` generics. The IP address can later be changed by software, but the MAC address is fixed. To allow several EDCL enabled GRETH controllers on the same sub-net, the 4 LSB bits of the IP and MAC address can optionally be set by an input signal. This is enabled by setting the `edcl` generic = 2, and driving the 4-bit LSB value on ethi.edcladdr.

38.6.4 EDCL limitations

The EDCL is designed to work without software intervention thus requiring automatic configuration in hardware. To simplify the hardware the EDCL is assumed to be able to handle all operating modes of the PHY and only reads the auto-negotiated mode and configures the MAC. This prevents it from working with gigabit capable PHYs since they might be in the gigabit mode and this cannot be handled by the MAC.

38.7 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH supports two of them: The Media Independent Interface (MII) and the Reduced Media Independent Interface (RMII).

The MII was defined in the 802.3 standard and is most commonly supported. The ethernet interface have been implemented according to this specification. It uses 16 signals.

The RMII was developed to meet the need for an interface allowing Ethernet controllers with smaller pin counts. It uses 6 (7) signals which are a subset of the MII signals. Table 347 shows the mapping between the RMII signals and the GRLIB MII interface.

*Table 347.*Signal mappings between RMII and the GRLIB MII interface.

RMII	MII
txd[1:0]	txd[1:0]
tx_en	tx_en
crs_dv	rx_crs
rxd[1:0]	rxd[1:0]
ref_clk	rmii_clk
rx_er	not used

38.8 Software drivers

Drivers for the GRETH MAC is provided for the following operating systems: RTEMS, eCos, uClinux and Linux-2.6. The drivers are freely available in full source code under the GPL license from Gaisler Research’s web site (http://gaisler.com/).

38.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 348.*GRETH registers

APB address offset	Register
0x0	Control register
0x4	Status/Interrupt-source register
0x8	MAC Address MSB
0xC	MAC Address LSB
0x10	MDIO Control/Status
0x14	Transmit descriptor pointer
0x18	Receiver descriptor pointer
0x1C	EDCL IP

Table 349. GRETH control register

31	30 28	27 8	7	6	5	4	3	2	1	0
ED	BS	RESERVED	SP	RS	PM	FD	RI	TI	RE	TE

31	EDCL available (ED) - Set to one if the EDCL is available.
30: 28	EDCL buffer size (BS) - Shows the amount of memory used for EDCL buffers. 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB.
27: 8	RESERVED
7	Speed (SP) - Sets the current speed mode. 0 = 10 Mbit, 1 = 100 Mbit. Only used in RMII mode (rmii = 1). A default value is automatically read from the PHY after reset. Reset value: ‘1’.
6	Reset (RS) - A one written to this bit resets the GRETH core. Self clearing.
5	Promiscuous mode (PM) - If set, the GRETH operates in promiscuous mode which means it will receive all packets regardless of the destination address. Reset value: ‘0’.
4	Full duplex (FD) - If set, the GRETH operates in full-duplex mode otherwise it operates in half-duplex. Reset value: ‘0’.
3	Receiver interrupt (RI) - Enable Receiver Interrupts. An interrupt will be generated each time a packet is received when this bit is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error. Reset value: ‘0’.
2	Transmitter interrupt (TI) - Enable Transmitter Interrupts. An interrupt will be generated each time a packet is transmitted when this bit is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error. Reset value: ‘0’.
1	Receive enable (RE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.
0	Transmit enable (TE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.

Table 350. GRETH status register

31 8	7	6	5	4	3	2	1	0
RESERVED	IA	TS	TA	RA	TI	RI	TE	RE

7	Invalid address (IA) - A packet with an address not accepted by the MAC was received. Cleared when written with a one. Reset value: ‘0’.
6	Too small (TS) - A packet smaller than the minimum size was received. Cleared when written with a one. Reset value: ‘0’.
5	Transmitter AHB error (TA) - An AHB error was encountered in transmitter DMA engine. Cleared when written with a one. Not Reset.
4	Receiver AHB error (RA) - An AHB error was encountered in receiver DMA engine. Cleared when written with a one. Not Reset.
3	Transmitter interrupt (TI) - A packet was transmitted without errors. Cleared when written with a one. Not Reset.
2	Receiver interrupt (RI) - A packet was received without errors. Cleared when written with a one. Not Reset.
1	Transmitter error (TE) - A packet was transmitted which terminated with an error. Cleared when written with a one. Not Reset.
0	Receiver error (RE) - A packet has been received which terminated with an error. Cleared when written with a one. Not Reset.

Table 351. GRETH MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the MAC address

31: 16	RESERVED
15: 0	The two most significant bytes of the MAC Address. Not Reset.

Table 352. GRETH MAC address LSB.

31 0
Bit 31 downto 0 of the MAC address

31: 0	The four least significant bytes of the MAC Address. Not Reset.

Table 353. GRETH MDIO ctrl/status register.

31 16	15 11	10 6	5	4	3	2	1	0
DATA	PHYADDR	REGADDR		NV	BU	LF	RD	WR

Table 353. GRETH MDIO ctrl/status register.

31: 16	Data (DATA) - Contains data read during a read operation and data that is transmitted is taken from this field. Reset value: 0x0000.
15: 11	PHY address (PHYADDR) - This field contains the address of the PHY that should be accessed during a write or read operation. Reset value: “00000”.
10: 6	Register address (REGADDR) - This field contains the address of the register that should be accessed during a write or read operation. Reset value: “00000”.
5	RESERVED
4	Not valid (NV) - When an operation is finished (BUSY = 0) this bit indicates whether valid data has been received that is, the data field contains correct data. Reset value: ‘0’.
3	Busy (BU) - When an operation is performed this bit is set to one. As soon as the operation is finished and the management link is idle this bit is cleared. Reset value: ‘0’.
2	Linkfail (LF) - When an operation completes (BUSY = 0) this bit is set if a functional management link was not detected. Reset value: ‘1’.
1	Read (RD) - Start a read operation on the management interface. Data is stored in the data field. Reset value: ‘0’.
0	Write (WR) - Start a write operation on the management interface. Data is taken from the Data field. Reset value: ‘0’.

Table 354. GRETH transmitter descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES

31: 10	Transmitter descriptor table base address (BASEADDR) - Base address to the transmitter descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 355. GRETH receiver descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES

31: 10	Receiver descriptor table base address (BASEADDR) - Base address to the receiver descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 356. GRETH EDCL IP register

31 0
EDCL IP ADDRESS

31: 0	EDCL IP address. Reset value is set with the ipaddrh and ipaddrl generics.

38.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x1D. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

38.11 Configuration options

Table 357 shows the configuration options of the core (VHDL generics).

*Table 357.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRETH.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the FIFOs.	0 - NTECH	inferred
ifg_gap	Number of ethernet clock cycles used for one interframe gap. Default value as required by the standard. Do not change unless you know what your doing.	1 - 255	24
attempt_limit	Maximum number of transmission attempts for one packet. Default value as required by the standard.	1 - 255	16
backoff_limit	Limit on the backoff size of the backoff time. Default value as required by the standard. Sets the number of bits used for the random value. Do not change unless you know what your doing.	1 - 10	10
slot_time	Number of ethernet clock cycles used for one slot- time. Default value as required by the ethernet standard. Do not change unless you know what you are doing.	1 - 255	128
mdcscaler	Sets the divisor value use to generate the mdio clock (mdc). The mdc frequency will be clk/(2*(mdcscaler+1)).	0 - 255	25
enable_mdio	Enable the Management interface,	0 - 1	0
fifosize	Sets the size in 32-bit words of the receiver and transmitter FIFOs.	4 - 32	8
nsync	Number of synchronization registers used.	1 - 2	2
edcl	Enable EDCL. 0 = disabled. 1 = enabled. 2 = enabled and 4-bit LSB of IP and ethernet MAC address programmed by ethi.edcladdr.	0 - 2	0
edclbufsz	Select the size of the EDCL buffer in kB.	1 - 64	1
macaddrh	Sets the upper 24 bits of the EDCL MAC address.*)	0 - 16#FFFFFF#	16#00005E#
macaddrl	Sets the lower 24 bits of the EDCL MAC address. *)	0 - 16#FFFFFF#	16#000000#
ipaddrh	Sets the upper 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#C0A8#
ipaddrl	Sets the lower 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#0035#
phyrstadr	Sets the reset value of the PHY address field in the MDIO register.	0 - 31	0
rmii	Selects the desired PHY interface. 0 = MII, 1 = RMII.	0 - 1	0
oepol	Selects polarity on output enable (ETHO.MDIO_OE). 0 = active low, 1 = active high	0 - 1	0

*) Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.

38.12 Signal descriptions

Table 358 shows the interface signals of the core (VHDL ports).

*Table 358.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ETHI	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rmii_clk	Input	Ethernet RMII clock.	-
	tx_clk	Input	Ethernet transmit clock.	-
	rx_clk	Input	Ethernet receive clock.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	rx_er	Input	Ethernet receive error.	High
	rx_col	Input	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Input	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	mdio_i	Input	Ethernet management data input	-
	phyrstaddr	Input	Reset address for GRETH PHY address field.	-
	edcladdr	Input	Sets the four least significant bits of the EDCL MAC address when the edcl generic is set to 2.	-
ETHO	reset	Output	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit enable.	High
	tx_er	Output	Ethernet transmit error.	High
	mdc	Output	Ethernet management data clock.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.

* see GRLIB IP Library User’s Manual

38.13 Library dependencies

Table 359 shows libraries used when instantiating the core (VHDL libraries).

*Table 359.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	NET	Signals, components	GRETH component declaration

38.14 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :: in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     enable_mdio  => 1,
     fifosize     => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#)
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

39 GRETH_GBIT - Gigabit Ethernet Media Access Controller (MAC) w. EDCL

39.1 Overview

Gaisler Research’s Gigabit Ethernet Media Access Controller (GRETH_GBIT) provides an interface between an AMBA-AHB bus and an Ethernet network. It supports 10/100/1000 Mbit speed in both full- and half-duplex. The AMBA interface consists of an APB interface for configuration and control and an AHB master interface which handles the dataflow. The dataflow is handled through DMA channels. There is one DMA engine for the transmitter and one for the receiver. Both share the same AHB master interface.

The ethernet interface supports the MII and GMII interfaces which should be connected to an external PHY. The GRETH also provides access to the MII Management interface which is used to configure the PHY. Optional hardware support for the Ethernet Debug Communication Link (EDCL) protocol is also provided. This is an UDP/IP based protocol used for remote debugging.

Some of the supported features for the DMA channels are Scatter Gather I/O and TCP/UDP over IPv4 checksum offloading for both receiver and transmitter. Software Drivers are provided for RTEMS, eCos, uClinux and Linux 2.6.

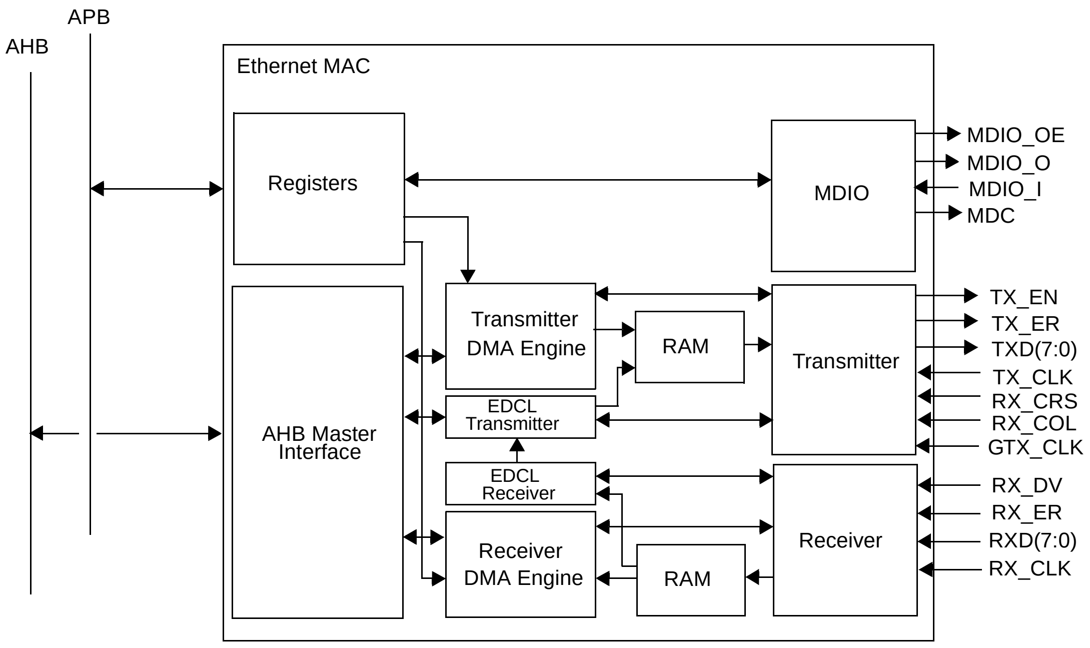


Figure 136. Block diagram of the internal structure of the GRETH_GBIT.

39.2 Operation

39.2.1 System overview

The GRETH_GBIT consists of 3 functional units: The DMA channels, MDIO interface and the optional Ethernet Debug Communication Link (EDCL).

The main functionality consists of the DMA channels which are used for transferring data between an AHB bus and an Ethernet network. There is one transmitter DMA channel and one Receiver DMA channel. The operation of the DMA channels is controlled through registers accessible through the APB interface.

The MDIO interface is used for accessing configuration and status registers in one or more PHYs connected to the MAC. The operation of this interface is also controlled through the APB interface.

The optional EDCL provides read and write access to an AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol to accomplish this. The EDCL contains no user accessible registers and always runs in parallel with the DMA channels.

The Media Independent Interface (MII) and Gigabit Media Independent Interface (GMII) are used for communicating with the PHY. More information can be found in section 39.7.

The EDCL and the DMA channels share the Ethernet receiver and transmitter. More information on these functional units is provided in sections 39.3 - 39.6.

39.2.2 Protocol support

The GRETH_GBIT is implemented according to IEEE standard 802.3-2002. There is no support for the optional control sublayer and no multicast addresses can be assigned to the MAC. This means that packets with type 0x8808 (the only currently defined ctrl packets) are discarded.

39.2.3 Hardware requirements

The GRETH_GBIT is synthesisable with most Synthesis tools. There are three or four clock domains depending on if the gigabit mode is used. The three domains always present are the AHB clock, Ethernet Receiver clock (RX_CLK) and the 10/100 Ethernet transmitter clock (TX_CLK). If the gigabit mode is also used the fourth clock domain is the gigabit transmitter clock (GTX_CLK). Both full-duplex and half-duplex operating modes are supported and both can be run in either 10/100 or 1000 Mbit. The system frequency requirement (AHB clock) for 10 Mbit operation is 2.5 MHz, 18 MHz for 100 Mbit and 40 MHz for 1000 Mbit mode. The 18 MHz limit was tested on a Xilinx board with a DCM that did not support lower frequencies so it might be possible to run it on lower frequencies. It might also be possible to run the 10 Mbit mode on lower frequencies.

RX_CLK and TX_CLK are sourced by the PHY while GTX_CLK is sourced by the MAC according to the 802.3-2002 standard. The GRETH_GBIT does not contain an internal clock generator so GTX_CLK should either be generated in the FPGA (with a PLL/DLL) or with an external oscillator.

39.3 Tx DMA interface

The transmitter DMA interface is used for transmitting data on an Ethernet network. The transmission is done using descriptors located in memory.

39.3.1 Setting up a descriptor.

A single descriptor is shown in table 360 and 361. The number of bytes to be sent should be set in the length field and the address field should point to the data. There are no alignment restrictions on the address field. If the interrupt enable (IE) bit is set, an interrupt will be generated when the packet has been sent (this requires that the transmitter interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was transmitted successfully or not.

Table 360. GRETH_GBIT transmit descriptor word 0 (address offset 0x0)

31 21	20	19	18	17	16	15	14	13	12	11	10 0
RESERVED	UC	TC	IC	MO	LC	AL	UE	IE	WR	EN	LENGTH

31: 21	RESERVED
20	UDP checksum (UC) - Calculate and insert the UDP checksum for this packet. The checksum is only inserted if an UDP packet is detected.
19	TCP checksum (TC) - Calculate and insert the TCP checksum for this packet. The checksum is only inserted if an TCP packet is detected.

Table 360. GRETH_GBIT transmit descriptor word 0 (address offset 0x0)

18	IP checksum (IC) - Calculate and insert the IP header checksum for this packet. The checksum is only inserted if an IP packet is detected.
17	More (MO) - More descriptors should be fetched for this packet (Scatter Gather I/O).
16	Late collision (LC) - A late collision occurred during the transmission (1000 Mbit mode only).
15	Attempt limit error (AL) - The packet was not transmitted because the maximum number of attempts was reached.
14	Underrun error (UE) - The packet was incorrectly transmitted due to a FIFO underrun error.
13	Interrupt enable (IE) - Enable Interrupts. An interrupt will be generated when the packet from this descriptor has been sent provided that the transmitter interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes to be transmitted.

Table 361. GRETH_GBIT transmit descriptor word 1 (address offset 0x4)

31 ... 0
ADDRESS

31: 0	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

To enable a descriptor the enable (EN) bit should be set and after this is done, the descriptor should not be touched until the enable bit has been cleared by the GRETH_GBIT. The rest of the fields in the descriptor are explained later in this section.

39.3.2 Starting transmissions

Enabling a descriptor is not enough to start a transmission. A pointer to the memory area holding the descriptors must first be set in the GRETH_GBIT. This is done in the transmitter descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH_GBIT the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when a transmission is active.

The final step to activate the transmission is to set the transmit enable bit in the control register. This tells the GRETH_GBIT that there are more active descriptors in the descriptor table. This bit should always be set when new descriptors are enabled, even if transmissions are already active. The descriptors must always be enabled before the transmit enable bit is set.

39.3.3 Descriptor handling after transmission

When a transmission of a packet has finished, status is written to the first word in the corresponding descriptor. The Underrun Error bit is set if the transmitter RAM was not able to provide data at a sufficient rate. This indicates a synchronization problem most probably caused by a low clock rate on the AHB clock. The whole packet is buffered in the transmitter RAM before transmission so underruns cannot be caused by bus congestion. The Alignment Error bit is set if more collisions occurred than allowed. When running in 1000 Mbit mode the Late Collision bit indicates that a collision occurred after the slottime boundary was passed.

The packet was successfully transmitted only if these three bits are zero. The other bits in the first descriptor word are set to zero after transmission while the second word is left untouched.

The enable bit should be used as the indicator when a descriptor can be used again, which is when it has been cleared by the GRETH_GBIT. There are three bits in the GRETH_GBIT status register that hold transmission status. The Transmit Error (TE) bit is set each time an transmission ended with an error (when at least one of the three status bits in the transmit descriptor has been set). The Transmit Successful (TI) is set each time a transmission ended successfully.

The Transmit AHB Error (TA) bit is set when an AHB error was encountered either when reading a descriptor, reading packet data or writing status to the descriptor. Any active transmissions are aborted and the transmitter is disabled. The transmitter can be activated again by setting the transmit enable register.

39.3.4 Setting up the data for transmission

The data to be transmitted should be placed beginning at the address pointed by the descriptor address field. The GRETH_GBIT does not add the Ethernet address and type fields so they must also be stored in the data buffer. The 4 B Ethernet CRC is automatically appended at the end of each packet. Each descriptor will be sent as a single Ethernet packet. If the size field in a descriptor is greater than 1514 B, the packet will not be sent.

39.3.5 Scatter Gather I/O

A packet can be generated from data fetched from several descriptors. This is called Scatter Gather I/O. The More (MO) bit should be set to 1 to indicate that more descriptors should be used to generate the current packet. When data from the current descriptor has been read to the RAM the next descriptor is fetched and the new data is appended to the previous data. This continues until a descriptor with the MO bit set to 0 is encountered. The packet will then be transmitted.

Status is written immediately when data has been read to RAM for descriptors with MO set to 1. The status bits are always set to 0 since no transmission has occurred. The status bits will be written will be written to the last descriptor for the packet (which had MO set to 0) when the transmission has finished.

No interrupts are generated for descriptors with MO set to 1 so the IE bit is don’t care in this case.

The checksum offload control bits (explained in section 39.3.6) must be set to the same values for all descriptors used for a single packet.

39.3.6 Checksum offloading

Support is provided for checksum calculations in hardware for TCP and UDP over IPv4. The checksum calculations are enabled in each descriptor and applies only to that packet (when the MO bit is set all descriptors used for a single packet must have the checksum control bits set in the same way).

The IP Checksum bit (IC) enables IP header checksum calculations. If an IPv4 packet is detected when transmitting the packet associated with the descriptor the header checksum is calculated and inserted. If TCP Checksum (TC) is set the TCP checksum is calculated and inserted if an TCP/IPv4 packet is detected. Finally, if the UDP Checksum bit is set the UDP checksum is calculated and inserted if a UDP/IPv4 packet is detected.

39.4 Rx DMA interface

The receiver DMA interface is used for receiving data from an Ethernet network. The reception is done using descriptors located in memory.

39.4.1 Setting up descriptors

A single descriptor is shown in table 362 and 363. The address field points at the location where the received data should be stored. There are no restrictions on alignment. The GRETH_GBIT will never store more than 1518 B to the buffer (the tagged maximum frame size excluding CRC). The CRC field (4 B) is never stored to memory so it is not included in this number. If the interrupt enable (IE) bit is set, an interrupt will be generated when a packet has been received to this buffer (this requires that the receiver interrupt bit in the control register is also set). The interrupt will be generated regardless of whether the packet was received successfully or not.

The enable bit is set to indicate that the descriptor is valid which means it can be used by the to store a packet. After it is set the descriptor should not be touched until the EN bit has been cleared by the GRETH_GBIT.

The rest of the fields in the descriptor are explained later in this section..

Table 362. GRETH_GBIT receive descriptor word 0 (address offset 0x0)

31 25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10 0
RESERVED	TR	TD	UR	UD	IR	ID	LE	OE	CE	FT	AE	IE	WR	EN	LENGTH

31: 25	RESERVED
24	TCP error (TR) - TCP checksum error detected.
23	TCP detected (TD) - TCP packet detected.
22	UDP error (UR) - UDP checksum error detected.
21	UDP detected (UD) - UDP packet detected.
20	IP error (IR) - IP checksum error detected.
19	IP detected (ID) - IP packet detected.
18	Length error (LE) - The length/type field of the packet did not match the actual number of received bytes.
17	Overrun error (OE) - The frame was incorrectly received due to a FIFO overrun.
16	CRC error (CE) - A CRC error was detected in this frame.
15	Frame too long (FT) - A frame larger than the maximum size was received. The excessive part was truncated.
14	Alignment error (AE) - An odd number of nibbles were received.
13	Interrupt Enable (IE) - Enable Interrupts. An interrupt will be generated when a packet has been received to this descriptor provided that the receiver interrupt enable bit in the control register is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error.
12	Wrap (WR) - Set to one to make the descriptor pointer wrap to zero after this descriptor has been used. If this bit is not set the pointer will increment by 8. The pointer automatically wraps to zero when the 1 kB boundary of the descriptor table is reached.
11	Enable (EN) - Set to one to enable the descriptor. Should always be set last of all the descriptor fields.
10: 0	LENGTH - The number of bytes received to this descriptor.

Table 363. GRETH_GBIT receive descriptor word 1 (address offset 0x4)

31 0
ADDRESS

31: 0	Address (ADDRESS) - Pointer to the buffer area from where the packet data will be loaded.

39.4.2 Starting reception

Enabling a descriptor is not enough to start reception. A pointer to the memory area holding the descriptors must first be set in the GRETH_GBIT. This is done in the receiver descriptor pointer register. The address must be aligned to a 1 kB boundary. Bits 31 to 10 hold the base address of descriptor area while bits 9 to 3 form a pointer to an individual descriptor. The first descriptor should be located at the base address and when it has been used by the GRETH_GBIT the pointer field is incremented by 8 to point at the next descriptor. The pointer will automatically wrap back to zero when the next 1 kB boundary has been reached (the descriptor at address offset 0x3F8 has been used). The WR bit in the descriptors can be set to make the pointer wrap back to zero before the 1 kB boundary.

The pointer field has also been made writable for maximum flexibility but care should be taken when writing to the descriptor pointer register. It should never be touched when reception is active.

The final step to activate reception is to set the receiver enable bit in the control register. This will make the GRETH_GBIT read the first descriptor and wait for an incoming packet.

39.4.3 Descriptor handling after reception

The GRETH indicates a completed reception by clearing the descriptor enable bit. The other control bits (WR, IE) are also cleared. The number of received bytes is shown in the length field. The parts of the Ethernet frame stored are the destination address, source address, type and data fields. Bits 24-14 in the first descriptor word are status bits indicating different receive errors. Bits 18 - 14 are zero after a reception without link layer errors. The status bits are described in table 362 (except the checksum offload bits which are also described in section 39.4.5).

Packets arriving that are smaller than the minimum Ethernet size of 64 B are not considered as a reception and are discarded. The current receive descriptor will be left untouched an used for the first packet arriving with an accepted size. The TS bit in the status register is set each time this event occurs.

If a packet is received with an address not accepted by the MAC, the IA status register bit will be set.

Packets larger than maximum size cause the FT bit in the receive descriptor to be set. The length field is not guaranteed to hold the correct value of received bytes. The counting stops after the word containing the last byte up to the maximum size limit has been written to memory.

The address word of the descriptor is never touched by the GRETH.

39.4.4 Reception with AHB errors

If an AHB error occurs during a descriptor read or data store, the Receiver AHB Error (RA) bit in the status register will be set and the receiver is disabled. The current reception is aborted. The receiver can be enabled again by setting the Receive Enable bit in the control register.

39.4.5 Checksum offload

Support is provided for checksum calculations in hardware for TCP/UDP over IPv4. The checksum logic is always active and detects IPv4 packets with TCP or UDP payloads. If IPv4 is detected the ID bit is set, UD is set if an UDP payload is detected in the IP packet and TD is set if a TCP payload is detected in the IP packet (TD and UD are never set if an IPv4 packet is not detected). When one or more of these packet types is detected its corresponding checksum is calculated and if an error is detected the checksum error bit for that packet type is set. The error bits are never set if the corresponding packet type is not detected.

39.5 MDIO Interface

The MDIO interface provides access to PHY configuration and status registers through a two-wire interface which is included in the MII interface. The GRETH_GBIT provides full support for the MDIO interface.

The MDIO interface can be used to access from 1 to 32 PHY containing 1 to 32 16-bit registers. A read transfer i set up by writing the PHY and register addresses to the MDIO Control register and setting the read bit. This caused the Busy bit to be set and the operation is finished when the Busy bit is cleared. If the operation was successful the Linkfail bit is zero and the data field contains the read data. An unsuccessful operation is indicated by the Linkfail bit being set. The data field is undefined in this case.

A write operation is started by writing the 16-bit data, PHY address and register address to the MDIO Control register and setting the write bit. The operation is finished when the busy bit is cleared and it was successful if the Linkfail bit is zero.

39.6 Ethernet Debug Communication Link (EDCL)

The EDCL provides access to an on-chip AHB bus through Ethernet. It uses the UDP, IP and ARP protocols together with a custom application layer protocol. The application layer protocol uses an ARQ algorithm to provide reliable AHB instruction transfers. Through this link, a read or write transfer can be generated to any address on the AHB bus. The EDCL is optional and must be enabled with a generic.

39.6.1 Operation

The EDCL receives packets in parallel with the MAC receive DMA channel. It uses a separate MAC address which is used for distinguishing EDCL packets from packets destined to the MAC DMA channel. The EDCL also has an IP address which is set through generics. Since ARP packets use the Ethernet broadcast address, the IP-address must be used in this case to distinguish between EDCL ARP packets and those that should go to the DMA-channel. Packets that are determined to be EDCL packets are not processed by the receive DMA channel.

When the packets are checked to be correct, the AHB operation is performed. The operation is performed with the same AHB master interface that the DMA-engines use. The replies are automatically sent by the EDCL transmitter when the operation is finished. It shares the Ethernet transmitter with the transmitter DMA-engine but has higher priority.

39.6.2 EDCL protocols

The EDCL accepts Ethernet frames containing IP or ARP data. ARP is handled according to the protocol specification with no exceptions.

IP packets carry the actual AHB commands. The EDCL expects an Ethernet frame containing IP, UDP and the EDCL specific application layer parts. Table 364 shows the IP packet required by the EDCL. The contents of the different protocol headers can be found in TCP/IP literature.

*Table 364.*The IP packet expected by the EDCL.

Ethernet Header	IP Header	UDP Header	2 B Offset	4 B Control word	4 B Address	Data 0 - 242 4B Words	Ethernet CRC

The following is required for successful communication with the EDCL: A correct destination MAC address as set by the generics, an Ethernet type field containing 0x0806 (ARP) or 0x0800 (IP). The IP-address is then compared with the value determined by the generics for a match. The IP-header checksum and identification fields are not checked. There are a few restrictions on the IP-header fields. The version must be four and the header size must be 5 B (no options). The protocol field must always be 0x11 indicating a UDP packet. The length and checksum are the only IP fields changed for the reply.

The EDCL only provides one service at the moment and it is therefore not required to check the UDP port number. The reply will have the original source port number in both the source and destination fields. UDP checksum are not used and the checksum field is set to zero in the replies.

The UDP data field contains the EDCL application protocol fields. Table 365 shows the application protocol fields (data field excluded) in packets received by the EDCL. The 16-bit offset is used to align the rest of the application layer data to word boundaries in memory and can thus be set to any value. The R/W field determines whether a read (0) or a write(1) should be performed. The length

*Table 365.*The EDCL application layer fields in received frames.

16-bit Offset	14-bit Sequence number	1-bit R/W	10-bit Length	7-bit Unused

field contains the number of bytes to be read or written. If R/W is one the data field shown in Table 364 contains the data to be written. If R/W is zero the data field is empty in the received packets. Table 366 shows the application layer fields of the replies from the EDCL. The length field is always zero for replies to write requests. For read requests it contains the number of bytes of data contained in the data field.

*Table 366.*The EDCL application layer fields in transmitted frames.

16-bit Offset	14-bit sequence number	1-bit ACK/NAK	10-bit Length	7-bit Unused

The EDCL implements a Go-Back-N algorithm providing reliable transfers. The 14-bit sequence number in received packets are checked against an internal counter for a match. If they do not match, no operation is performed and the ACK/NAK field is set to 1 in the reply frame. The reply frame contains the internal counter value in the sequence number field. If the sequence number matches, the operation is performed, the internal counter is incremented, the internal counter value is stored in the sequence number field and the ACK/NAK field is set to 0 in the reply. The length field is always set to 0 for ACK/NAK=1 frames. The unused field is not checked and is copied to the reply. It can thus be set to hold for example some extra id bits if needed.

39.7 Media Independent Interfaces

There are several interfaces defined between the MAC sublayer and the Physical layer. The GRETH_GBIT supports the Media Independent Interface (MII) and the Gigabit Media Independent Interface (GMII).

The GMII is used in 1000 Mbit mode and the MII in 10 and 100 Mbit. These interfaces are defined separately in the 802.3-2002 standard but in practice they share most of the signals. The GMII has 9 additional signals compared to the MII. Four data signals are added to the receiver and transmitter data interfaces respectively and a new transmit clock for the gigabit mode is also introduced.

*Table 367.*Signals in GMII and MII.

MII and GMII	**GMII Only**
txd[3:0]	txd[7:4]
tx_en	rxd[7:4]
tx_er	gtx_clk
rx_col	
rx_crs	
rxd[3:0]	
rx_clk	
rx_er	
rx_dv	

39.8 Registers

The core is programmed through registers mapped into APB address space.

*Table 368.*GRETH_GBIT registers

APB address offset	**Register**
0x0	Control register
0x4	Status/Interrupt-source register
0x8	MAC Address MSB
0xC	MAC Address LSB
0x10	MDIO Control/Status
0x14	Transmit descriptor pointer
0x18	Receiver descriptor pointer
0x1C	EDCL IP

Table 369. GRETH control register

31	30 28	27	26 10	9	8	7	6	5	4	3	2	1	0
ED	BS	GA	RESERVED	BM	GB	SP	RS	PR	FD	RI	TI	RE	TE

31	EDCL available (ED) - Set to one if the EDCL is available.
30: 28	EDCL buffer size (BS) - Shows the amount of memory used for EDCL buffers. 0 = 1 kB, 1 = 2 kB, ..., 6 = 64 kB.
27	Gigabit MAC available (GA) - This bit always reads as a 1 and indicates that the MAC has 1000 Mbit capability.
26: 10	RESERVED
9	Burstmode (BM) - When set to 1, transmissions use burstmode in 1000 Mbit Half-duplex mode (GB=1, FD = 0). When 0 in this speed mode normal transmissions are always used with extension inserted. Operation is undefined when set to 1 in other speed modes. Reset value: ‘0’.
8	Gigabit (GB) - 1 sets the current speed mode to 1000 Mbit and when set to 0, the speed mode is selected with bit 7 (SP). Reset value: ‘0’.
7	Speed (SP) - Sets the current speed mode. 0 = 10 Mbit, 1 = 100 Mbit. Must not be set to 1 at the same time as bit 8 (GB). Reset valuie: ‘0’.
6	Reset (RS) - A one written to this bit resets the GRETH_GBIT core. Self clearing.
5	Promiscuous mode (PM) - If set, the GRETH_GBIT operates in promiscuous mode which means it will receive all packets regardless of the destination address. Reset value: ‘0’.
4	Full duplex (FD) - If set, the GRETH_GBIT operates in full-duplex mode otherwise it operates in half-duplex. Reset value: ‘0’.
3	Receiver interrupt (RI) - Enable Receiver Interrupts. An interrupt will be generated each time a packet is received when this bit is set. The interrupt is generated regardless if the packet was received successfully or if it terminated with an error. Reset value: ‘0’.
2	Transmitter interrupt (TI) - Enable Transmitter Interrupts. An interrupt will be generated each time a packet is transmitted when this bit is set. The interrupt is generated regardless if the packet was transmitted successfully or if it terminated with an error. Reset value: ‘0’.
1	Receive enable (RE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH_GBIT will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.
0	Transmit enable (TE) - Should be written with a one each time new descriptors are enabled. As long as this bit is one the GRETH_GBIT will read new descriptors and as soon as it encounters a disabled descriptor it will stop until TE is set again. This bit should be written with a one after the new descriptors have been enabled. Reset value: ‘0’.

Table 370. GRETH_GBIT status register.

31 8	7	6	5	4	3	2	1	0
RESERVED	IA	TS	TA	RA	TI	RI	TE	RE

31: 8	RESERVED
7	Invalid address (IA) - A packet with an address not accepted by the MAC was received. Cleared when written with a one. Reset value: ‘0’.
6	Too small (TS) - A packet smaller than the minimum size was received. Cleared when written with a one. Reset value: ‘0’.
5	Transmitter AHB error (TA) - An AHB error was encountered in transmitter DMA engine. Cleared when written with a one. Not Reset.
4	Receiver AHB error (RA) - An AHB error was encountered in receiver DMA engine. Cleared when written with a one. Not Reset.
3	Transmit successful (TI) - A packet was transmitted without errors. Cleared when written with a one. Not Reset.
2	Receive successful (RI) - A packet was received without errors. Cleared when written with a one. Not Reset.
1	Transmitter error (TE) - A packet was transmitted which terminated with an error. Cleared when written with a one. Not Reset.
0	Receiver error (RE) - A packet has been received which terminated with an error. Cleared when written with a one. Not Reset.

Table 371. GRETH_GBIT MAC address MSB.

31 16	15 0
RESERVED	Bit 47 downto 32 of the MAC Address

31: 16	RESERVED
15: 0	The two most significant bytes of the MAC Address. Not Reset.

Table 372. GRETH_GBIT MAC address LSB.

31 0
Bit 31 downto 0 of the MAC Address

31: 0	The 4 least significant bytes of the MAC Address. Not Reset.

Table 373. GRETH_GBIT MDIO control/status register.

31 16	15 11	10 6	5	4	3	2	1	0
DATA	PHYADDR	REGADDR		NV	BU	LF	RD	WR

31: 16	Data (DATA) - Contains data read during a read operation and data that is transmitted is taken from this field. Reset value: 0x0000.
15: 11	PHY address (PHYADDR) - This field contains the address of the PHY that should be accessed during a write or read operation. Reset value: “00000”.
10: 6	Register address (REGADDR) - This field contains the address of the register that should be accessed during a write or read operation. Reset value: ‘”00000”.
5	RESERVED
4	Not valid (NV) - When an operation is finished (BUSY = 0) this bit indicates whether valid data has been received that is, the data field contains correct data. Reset value: ‘0’.

Table 373. GRETH_GBIT MDIO control/status register.

3	Busy (BU) - When an operation is performed this bit is set to one. As soon as the operation is finished and the management link is idle this bit is cleared. Reset value: ‘0’.
2	Linkfail (LF) - When an operation completes (BUSY = 0) this bit is set if a functional management link was not detected. Reset value: ‘1’.
1	Read (RD) - Start a read operation on the management interface. Data is stored in the data field. Reset value: ‘0’.
0	Write (WR) - Start a write operation on the management interface. Data is taken from the Data field. Reset value: ‘0’.

Table 374. GRETH_GBIT transmitter descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES

31: 10	Transmitter descriptor table base address (BASEADDR) - Base address to the transmitter descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 375. GRETH_GBIT receiver descriptor table base address register.

31 10	9 3	2 0
BASEADDR	DESCPNT	RES

31: 10	Receiver descriptor table base address (BASEADDR) - Base address to the receiver descriptor table.Not Reset.
9: 3	Descriptor pointer (DESCPNT) - Pointer to individual descriptors. Automatically incremented by the Ethernet MAC.
2: 0	RESERVED

Table 376. GRETH_GBIT EDCL IP register

31 0
EDCL IP ADDRESS

31: 0	EDCL IP address. Reset value is set with the ipaddrh and ipaddrl generics.

39.9 Software drivers

Drivers for the GRETH_GBIT MAC is provided for the following operating systems: RTEMS, eCos, uClinux and Linux-2.6. The drivers are freely available in full source code under the GPL license from Gaisler Research’s web site (http://www.gaisler.com/).

39.10 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x01D. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

39.11 Configuration options

Table 377 shows the configuration options of the core (VHDL generics).

*Table 377.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRETH.	0 - NAHBIRQ-1	0
memtech	Memory technology used for the FIFOs.	0 - NTECH	inferred
ifg_gap	Number of ethernet clock cycles used for one interframe gap. Default value as required by the standard. Do not change unless you know what your doing.	1 - 255	24
attempt_limit	Maximum number of transmission attempts for one packet. Default value as required by the standard.	1 - 255	16
backoff_limit	Limit on the backoff size of the backoff time. Default value as required by the standard. Sets the number of bits used for the random value. Do not change unless you know what your doing.	1 - 10	10
slot_time	Number of ethernet clock cycles used for one slot- time. Default value as required by the ethernet standard. Do not change unless you know what you are doing.	1 - 255	128
mdcscaler	Sets the divisor value use to generate the mdio clock (mdc). The mdc frequency will be clk/(2*(mdcscaler+1)).	0 - 255	25
nsync	Number of synchronization registers used.	1 - 2	2
edcl	Enable EDCL.	0 - 1	0
edclbufsz	Select the size of the EDCL buffer in kB.	1 - 64	1
burstlength	Sets the maximum burstlength used during DMA	4 - 128	32
macaddrh	Sets the upper 24 bits of the EDCL MAC address.*)	0 - 16#FFFFFF#	16#00005E#
macaddrl	Sets the lower 24 bits of the EDCL MAC address. *)	0 - 16#FFFFFF#	16#000000#
ipaddrh	Sets the upper 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#C0A8#
ipaddrl	Sets the lower 16 bits of the EDCL IP address reset value.	0 - 16#FFFF#	16#0035#
phyrstadr	Sets the reset value of the PHY address field in the MDIO register. When set to 32, the address is taken from the ethi.phyrstaddr signal.	0 - 32	0
sim	Set to 1 for simulations and 0 for synthesis. 1 selects a faster mdc clock to speed up simulations.	0 - 1	0

*) Not all addresses are allowed and most NICs and protocol implementations will discard frames with illegal addresses silently. Consult network literature if unsure about the addresses.

39.12 Signal descriptions

Table 378 shows the interface signals of the core (VHDL ports).

*Table 378.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
ETHI	gtx_clk	Input	Ethernet gigabit transmit clock.	-
	rmii_clk	Input	Ethernet RMII clock.	-
	tx_clk	Input	Ethernet transmit clock.	-
	rx_clk	Input	Ethernet receive clock.	-
	rxd	Input	Ethernet receive data.	-
	rx_dv	Input	Ethernet receive data valid.	High
	rx_er	Input	Ethernet receive error.	High
	rx_col	Input	Ethernet collision detected. (Asynchronous, sampled with tx_clk)	High
	rx_crs	Input	Ethernet carrier sense. (Asynchronous, sampled with tx_clk)	High
	mdio_i	Input	Ethernet management data input	-
	phyrstaddr	Input	Reset address for GRETH PHY address field.	-
	edcladdr	Input	Sets the four least significant bits of the EDCL MAC address when the edcl generic is set to 2.	-
ETHO	reset	Output	Ethernet reset (asserted when the MAC is reset).	Low
	txd	Output	Ethernet transmit data.	-
	tx_en	Output	Ethernet transmit enable.	High
	tx_er	Output	Ethernet transmit error.	High
	mdc	Output	Ethernet management data clock.	-
	mdio_o	Output	Ethernet management data output.	-
	mdio_oe	Output	Ethernet management data output enable.	Set by the oepol generic.

* see GRLIB IP Library User’s Manual

39.13 Library dependencies

Table 379 shows libraries used when instantiating the core (VHDL libraries).

*Table 379.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	ETHERNET_MAC	Signals, component	GRETH_GBIT component declarations, GRETH_GBIT signals.
GAISLER	NET	Signals	Ethernet signals

39.14 Instantiation

This example shows how the core an be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.ethernet_mac.all;

entity greth_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- ethernet signals
    ethi :  in  eth_in_type;
    etho :  in  eth_out_type
    );
end;

architecture rtl of greth_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- GRETH
  e1 : greth_gbit
    generic map(
     hindex       => 0,
     pindex       => 12,
     paddr        => 12,
     pirq         => 12,
     memtech      => inferred,
     mdcscaler    => 50,
     burstlength  => 32,
     nsync        => 1,
     edcl         => 1,
     edclbufsz    => 8,
     macaddrh     => 16#00005E#,
     macaddrl     => 16#00005D#,
     ipaddrh      => 16#c0a8#,
     ipaddrl      => 16#0035#)
 port map(
   rst          => rstn,
   clk          => clk,
   ahbmi        => ahbmi,
   ahbmo        => ahbmo(0),
   apbi         => apbi,
   apbo         => apbo(12),
   ethi         => ethi,
   etho         => etho
   );
end;
```

40 GRFIFO - FIFO Interface

40.1 Overview

The FIFO interface is assumed to operate in an AMBA bus system where both the AMBA AHB bus and the APB bus are present. The AMBA APB bus is used for configuration, control and status handling. The AMBA AHB bus is used for retrieving and storing FIFO data in memory external to the FIFO interface. This memory can be located on-chip or external to the chip.

The FIFO interface supports transmission and reception of blocks of data by use of circular buffers located in memory external to the core. Separate transmit and receive buffers are assumed. Reception and transmission of data can be ongoing simultaneously.

After a data transfer has been set up via the AMBA APB interface, the DMA controller initiates a burst of read accesses on the AMBA AHB bus to fetch data from memory that are performed by the AHB master. The data are then written to the external FIFO. When a programmable amount of data has been transmitted, the DMA controller issues an interrupt.

After reception has been set up via the AMBA APB interface, data are read from the external FIFO. To store data to memory, the DMA controller initiates a burst of write accesses on the AMBA AHB bus that are performed by the AHB master. When a programmable amount of data has been received, the DMA controller issues an interrupt.

The block diagram shows a possible usage of the FIFO interface.

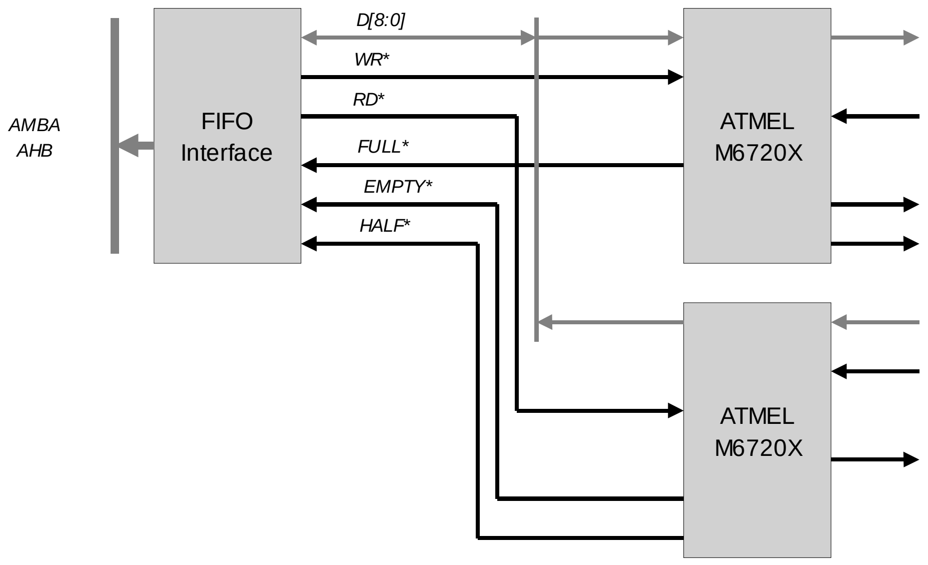


Figure 137. Block diagram of the GRFIFO environment.

40.1.1 Function

The core implements the following functions:

- data transmission to external FIFO
- circular transmit buffer
- direct memory access for transmitter
- data reception from external FIFO
- circular receive buffer for receiver
- direct memory access

- automatic 8- and 16-bit data width conversion
- general purpose input output

40.1.2 Transmission

Data to be transferred via the FIFO interface are fetched via the AMBA AHB master interface from on-chip or off-chip memory. This is performed by means of direct memory access (DMA), implementing a circular transmit buffer in the memory. The transmit channel is programmable via the AMBA APB slave interface, which is also used for the monitoring of the FIFO and DMA status.

The transmit channel is programmed in terms of a base address and size of the circular transmit buffer. The outgoing data are stored in the circular transmit buffer by the system. A write address pointer register is then set by the system to indicate the last byte written to the circular transmit buffer. An interrupt address pointer register is used by the system to specify a location in the circular transmit buffer from which a data read should cause an interrupt to be generated.

The FIFO interface automatically indicates with a read address pointer register the location of the last fetched byte from the circular transmit buffer. Read accesses are performed as incremental bursts, except when close to the location specified by the interrupt pointer register at which point the last bytes might be fetched by means of single accesses.

Data transferred via the FIFO interface can be either 8- or 16-bit wide. The handling of the transmit channel is however the same. All transfers performed by the AMBA AHB master are 32-bit word based. No byte or half-word transfers are performed.

To handle the 8- and 16-bit FIFO data width, a 32-bit read access might carry less than four valid bytes. In such a case, the remaining bytes are ignored. When additional data are available in the circular transmit buffer, the previously fetched bytes will be re-read together with the newly written bytes to form the 32-bit data. Only the new bytes will be transmitted to the FIFO, not to transmit the same byte more than once. The aforementioned write address pointer indicates what bytes are valid.

An interrupt is generated when the circular transmit buffer is empty. The status of the external FIFO is observed via the AMBA APB slave interface, indicating Full Flag and Half-Full Flag.

40.1.3 Reception

Data received via the FIFO interface are stored via the AMBA AHB master interface to on-chip or off-chip memory. This is performed by means of direct memory access (DMA), implementing a circular receive buffer in the memory. The receive channel is programmable via the AMBA APB slave interface, which is also used for the monitoring of the FIFO and DMA status.

The receive channel is programmed in terms of a base address and size of the circular receive buffer. The incoming data are stored in the circular receive buffer. The interface automatically indicates with a write address pointer register the location of the last stored byte. A read address pointer register is used by the system to indicate the last byte read from the circular receive buffer. An interrupt address pointer register is used by the system to specify a location in the circular receive buffer to which a data write should cause an interrupt to be generated.

Write accesses are performed as incremental bursts, except when close to the location specified by the interrupt pointer register at which point the last bytes might be stored by means of single accesses.

Data transferred via the FIFO interface can be either 8- or 16-bit wide. The handling of the receive channel is however the same. All transfers performed by the AMBA AHB master are 32-bit word based. No byte or half-word transfers are performed.

To handle the 8- and 16-bit FIFO data width, a 32-bit write access might carry less than four valid bytes. In such a case, the remaining bytes will all be zero. When additional data are received from the FIFO interface, the previously stored bytes will be re-written together with the newly received bytes to form the 32-bit data. In this way, the previously written bytes are never overwritten. The aforementioned write address pointer indicates what bytes are valid.

An interrupt is generated when the circular receive buffer is full. If more FIFO data are available, they will not be moved to the circular receive buffer. The status of the external FIFO is observed via the AMBA APB slave interface, indicating Empty Flag and Half-Full Flag.

40.1.4 General purpose input output

Data input and output signals unused by the FIFO interface can be used as general purpose input output, providing 0, 8 or 16 individually programmable channels.

40.1.5 Interfaces

The core provides the following external and internal interfaces:

- FIFO interface
- AMBA AHB master interface, with sideband signals as per [GLRIB] including:
    - cachability information
    - interrupt bus
    - configuration information
    - diagnostic information
- AMBA APB slave interface, with sideband signals as per [GLRIB] including:
    - interrupt bus
    - configuration information
    - diagnostic information

The interface is intended to be used with the following FIFO devices from ATMEL:

Name:	Type:	
M67204H	4K x 9 FIFO	ESA/SCC 9301/049, SMD/5962-89568
M67206H	16K x 9 FIFO	ESA/SCC 9301/048, SMD/5962-93177
M672061H	16K x 9 FIFO	ESA/SCC 9301/048, SMD/5962-93177

40.2 Interface

The external interface supports one or more FIFO devices for data output (transmission) and/or one or more FIFO devices for data input (reception). The external interface supports FIFO devices with 8- and 16-bit data width. Note that one device is used when 8-bit and two devices are used when 16-bit data width is needed. The data width is programmable. Note that this is performed commonly for both directions.

The external interface supports one parity bit over every 8 data bits. Note that there can be up to two parity bits in either direction. The parity is programmable in terms of odd or even parity. Note that odd parity is defined as an odd number of logical ones in the data bits and parity bit. Note that even parity is defined as an even number of logical ones in the data bits and parity bit. Parity is generated for write accesses to the external FIFO devices. Parity is checked for read accesses from the external FIFO devices and a parity failure results in an internal interrupt.

The external interface provides a Write Enable output signal. The external interface provides a Read Enable output signal. The timing of the access towards the FIFO devices is programmable in terms of wait states based on system clock periods.

The external interface provides an Empty Flag input signal, which is used for flow-control during the reading of data from the external FIFO, not reading any data while the external FIFO is empty. Note that the Empty Flag is sampled at the end of the read access to determine if the FIFO is empty. To determine when the FIFO is not empty, the Empty Flag is re-synchronized with Clk.

The external interface provides a Full Flag input signal, which is used for flow-control during the writing of data to the external FIFO, not writing any data while the external FIFO is full. Note that the Full Flag is sampled at the end of the write access to determine if the FIFO is full. To determine when the FIFO is not full, the Full Flag is re-synchronized with Clk.

The external interface provides a Half-Full Flag input signal, which is used as status information only.

The data input and output signals are possible to use as general purpose input output channels. This need is only realized when the data signals are not used by the FIFO interface. Each general purpose input output channel is individually programmed as input or output. The default reset configuration for each general purpose input output channel is as input. The default reset value each general purpose input output channel is logical zero. Note that protection toward spurious pulse commands during power up shall be mitigated as far as possible by means of I/O cell selection from the target technology.

40.3 Waveforms

The following figures show read and write accesses to the FIFO with 0 and 4 wait states, respectively.

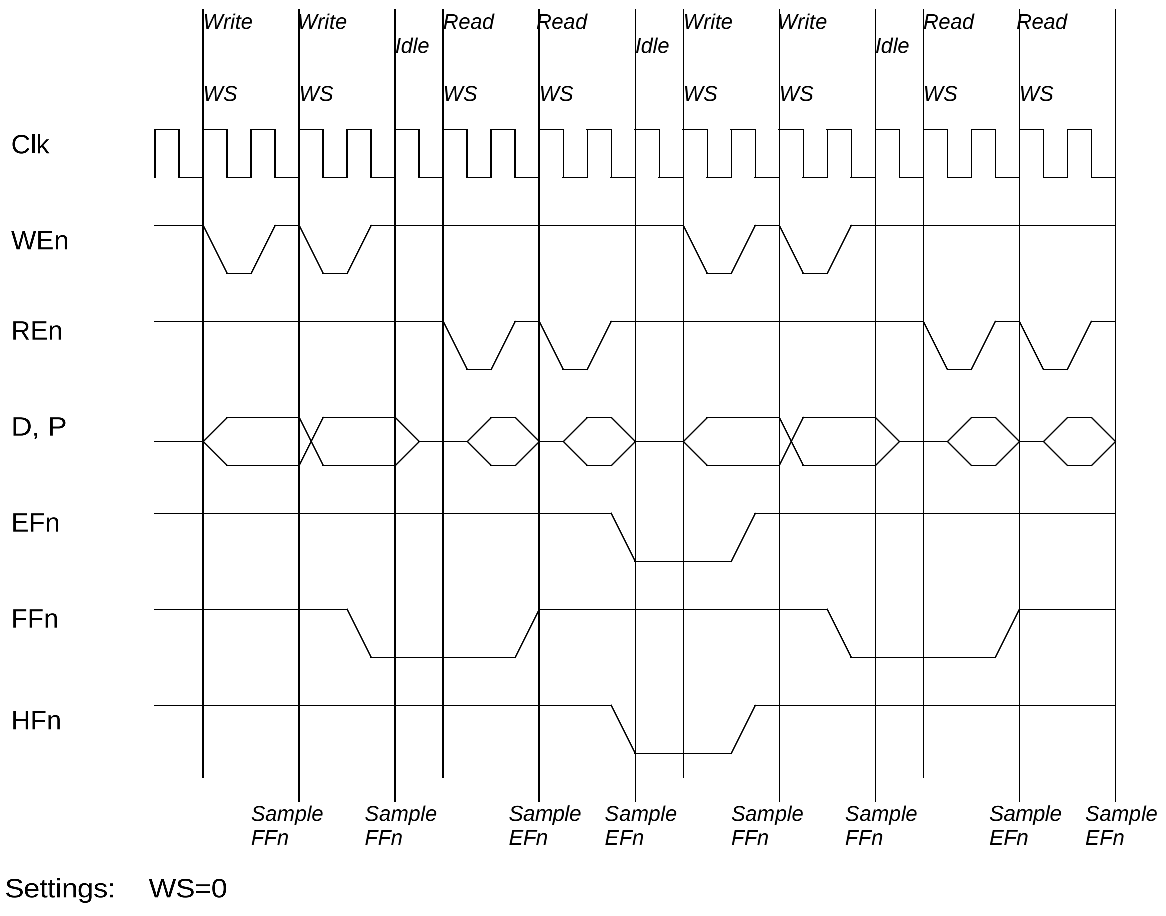


Figure 138. FIFO read and write access waveform, 0 wait states (WS)

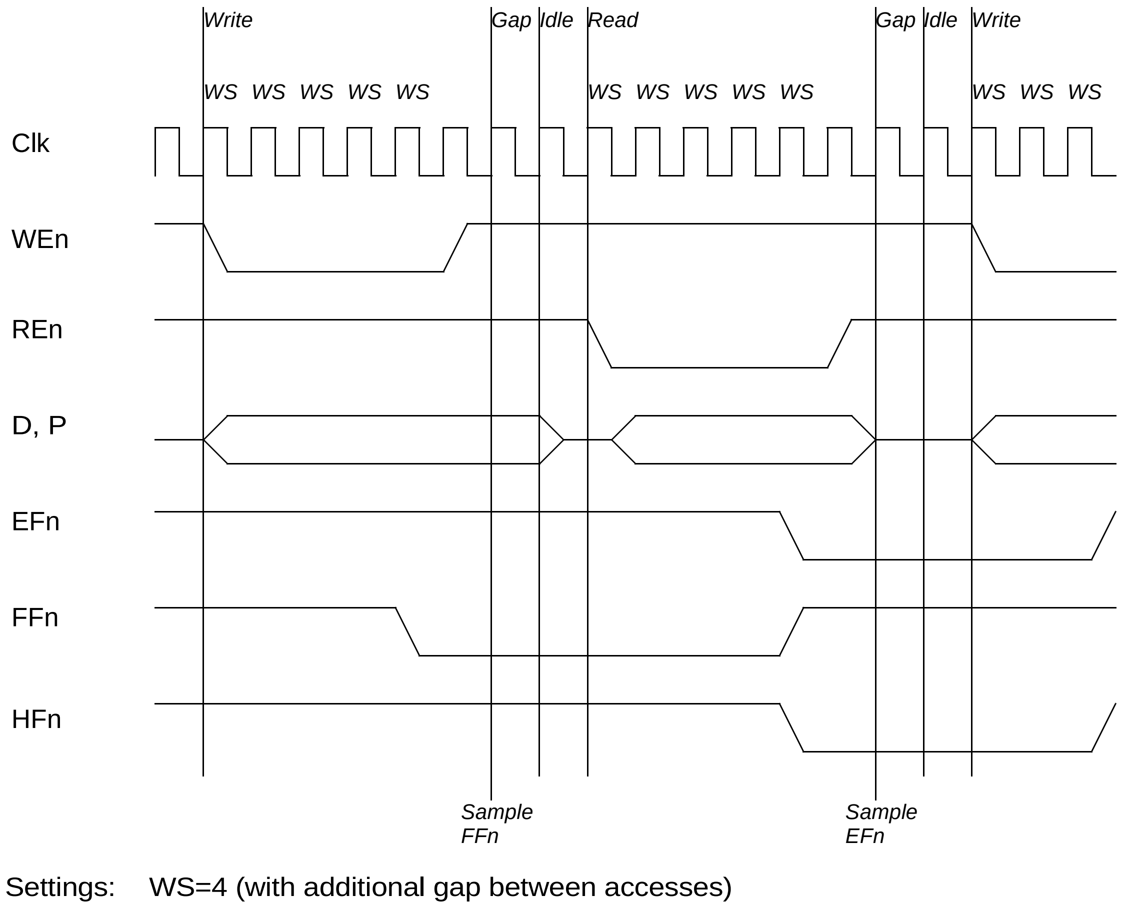


Figure 139. FIFO read and write access waveform, 4 wait states (WS)

40.4 Transmission

The transmit channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The transmit channel can be enabled or disabled.

40.4.1 Circular buffer

The transmit channel operates on a circular buffer located in memory external to the FIFO controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

The size of the buffer is defined by the FifoTxSIZE.SIZE field, specifying the number of 64 byte blocks that fit in the buffer.

E.g. FifoTxSIZE.SIZE = 1 means 64 bytes fit in the buffer.

Note however that it is not possible to fill the buffer completely, leaving at least one word in the buffer empty. This is to simplify wrap-around condition checking.

E.g. FifoTxSIZE.SIZE = 1 means that 60 bytes fit in the buffer at any given time.

40.4.2 Write and read pointers

The write pointer (FifoTxWR.WRITE) indicates the position+1 of the last byte written to the buffer. The write pointer operates on number of bytes, not on absolute or relative addresses.

The read pointer (FifoTxRD.READ) indicates the position+1 of the last byte read from the buffer. The read pointer operates on number of bytes, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of bytes available in the buffer for transmission. The difference is calculated using the buffer size, specified by the FifoTxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE=2 and FifoTxRD.READ=0.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =0 and FifoTxRD.READ =62.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =1 and FifoTxRD.READ =63.
- There are 2 bytes available for transmit when FifoTxSIZE.SIZE=1, FifoTxWR.WRITE =5 and FifoTxRD.READ =3.

When a byte has been successfully written to the FIFO, the read pointer (FifoTxRD.READ) is automatically incremented, taking wrap around effects of the circular buffer into account. Whenever the write pointer FifoTxWR.WRITE and read pointer FifoTxRD.READ are equal, there are no bytes available for transmission.

40.4.3 Location

The location of the circular buffer is defined by a base address (FifoTxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

40.4.4 Transmission procedure

When the channel is enabled (FifoTxCTRL.ENABLE=1), as soon as there is a difference between the write and read pointer, a transmission will be started. Note that the channel should not be enabled if a potential difference between the write and read pointers could be created, to avoid the data transmission to start prematurely.

A data transmission will begin with a fetch of the data from the circular buffer to a local buffer in the FIFO controller. After a successful fetch, a write access will be performed to the FIFO.

The read pointer (FifoTxRD.READ) is automatically incremented after a successful transmission, taking wrap around effects of the circular buffer into account. If there is at least one byte available in the circular buffer, a new fetch will be performed.

If the write and read pointers are equal, no more prefetches and fetches will be performed, and transmission will stop.

Interrupts are provided to aid the user during transmission, as described in detail later in this section. The main interrupts are the TxError, TxEmpty and TxIrq which are issued on the unsuccessful transmission of a byte due to an error condition on the AMBA bus, when all bytes have been transmitted successfully and when a predefined number of bytes have been transmitted successfully.

Note that 32-bit wide read accesses past the address of the last byte or halfword available for transmission can be performed as part of a burst operation, although no read accesses are made beyond the circular buffer size.

All accesses to the AMBA AHB bus are performed as two consecutive 32-bit accesses in a burst, or as a single 32-bit access in case of an AMBA AHB bus error.

40.4.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (FifoTxADDR.ADDR) field.

While the channel is disabled, the read pointer (FifoTxRD.READ) can be changed to an arbitrary value pointing to the first byte to be transmitted, and the write pointer (FifoTxWR.WRITE) can be changed to an arbitrary value.

When the channel is enabled, the transmission will start from the read pointer and continue to the write pointer.

40.4.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while data is being fetched will result in a TxError interrupt.

If the FifoCONF.ABORT bit is set to 0b, the channel causing the AHB error will re-try to read the data being fetched from memory till successful.

If the FifoCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (FifoTxCTRL.ENABLE is cleared automatically to 0 b). The read pointer can be used to determine which data caused the AHB error. The interface will not start any new write accesses to the FIFO. Any ongoing FIFO access will be completed and the FifoTxSTAT.TxOnGoing bit will be cleared. When the channel is re-enabled, the fetch and transmission of data will resume at the position where it was disabled, without losing any data.

40.4.7 Enable and disable

When an enabled transmit channel is disabled (FifoTxCTRL.ENABLE=0b), the interface will not start any new read accesses to the circular buffer by means of DMA over the AMBA AHB bus. No new write accesses to the FIFO will be started. Any ongoing FIFO access will be completed. If the data is written successfully, the read pointer (FifoTxRD.READ) is automatically incremented and the FifoTxSTAT.TxOnGoing bit will be cleared. Any associated interrupts will be generated.

Any other fetched or pre-fetched data from the circular buffer which is temporarily stored in the local buffer will be discarded, and will be fetched again when the transmit channel is re-enabled.

The progress of the any ongoing access can be observed via the FifoTxSTAT.TxOnGoing bit. The FifoTxSTAT.TxOnGoing must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers). It is also possible to wait for the TxEmpty interrupt described hereafter.

The channel can be re-enabled again without the need to re-configure the address, size and pointers. No data transmission is started while the channel is not enabled.

40.4.8 Interrupts

During transmission several interrupts can be generated:

- TxEmpty: Successful transmission of all data in buffer
- TxIrq: Successful transmission of a predefined number of data
- TxError: AHB access error during transmission

The TxEmpty and TxIrq interrupts are only generated as the result of a successful data transmission, after the FifoTxRD.READ pointer has been incremented.

40.5 Reception

The receive channel is defined by the following parameters:

- base address
- buffer size
- write pointer
- read pointer

The receive channel can be enabled or disabled.

40.5.1 Circular buffer

The receive channel operates on a circular buffer located in memory external to the FIFO controller. The circular buffer can also be used as a straight buffer. The buffer memory is accessed via the AMBA AHB master interface.

The size of the buffer is defined by the FifoRxSIZE.SIZE field, specifying the number 64 byte blocks that fit in the buffer.

E.g. FifoRxSIZE.SIZE=1 means 64 bytes fit in the buffer.

Note however that it is not possible for the hardware to fill the buffer completely, leaving at least two words in the buffer empty. This is to simplify wrap-around condition checking.

E.g. FifoRxSIZE.SIZE=1 means that 56 bytes fit in the buffer at any given time.

40.5.2 Write and read pointers

The write pointer (FifoRxWR.WRITE) indicates the position+1 of the last byte written to the buffer. The write pointer operates on number of bytes, not on absolute or relative addresses.

The read pointer (FifoRxRD.READ) indicates the position+1 of the last byte read from the buffer. The read pointer operates on number of bytes, not on absolute or relative addresses.

The difference between the write and the read pointers is the number of bytes available in the buffer for reception. The difference is calculated using the buffer size, specified by the FifoRxSIZE.SIZE field, taking wrap around effects of the circular buffer into account.

Examples:

- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =2 and FifoRxRD.READ=0.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =0 and FifoRxRD.READ=62.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =1 and FifoRxRD.READ=63.
- There are 2 bytes available for read-out when FifoRxSIZE.SIZE=1, FifoRxWR.WRITE =5 and FifoRxRD.READ=3.

When a byte has been successfully received and stored, the write pointer (FifoRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

40.5.3 Location

The location of the circular buffer is defined by a base address (FifoRxADDR.ADDR), which is an absolute address. The location of a circular buffer is aligned on a 1kbyte address boundary.

40.5.4 Reception procedure

When the channel is enabled (FifoRxCTRL.ENABLE=1), and there is space available for data in the circular buffer (as defined by the write and read pointer), a read access will be started towards the FIFO, and then an AMBA AHB store access will be started. The received data will be temporarily stored in a local store-buffer in the FIFO controller. Note that the channel should not be enabled until the write and read pointers are configured, to avoid the data reception to start prematurely

After a datum has been successfully stored the FIFO controller is ready to receive new data. The write pointer (FifoRxWR.WRITE) is automatically incremented, taking wrap around effects of the circular buffer into account.

Interrupts are provided to aid the user during reception, as described in detail later in this section. The main interrupts are the RxError, RxParity, RxFull and RxIrq which are issued on the unsuccessful reception of data due to an AMBA AHB error or parity error, when the buffer has been successfully filled and when a predefined number of data have been received successfully.

All accesses to the AMBA AHB bus are performed as two consecutive 32-bit accesses in a burst, or as a single 32-bit access in case of an AMBA AHB bus error.

40.5.5 Straight buffer

It is possible to use the circular buffer as a straight buffer, with a higher granularity than the 1kbyte address boundary limited by the base address (FifoRxADDR.ADDR) field.

While the channel is disabled, the write pointer (FifoRxWR.WRITE) can be changed to an arbitrary value pointing to the first data to be received, and the read pointer (FifoRxRD.READ) can be changed to an arbitrary value.

When the channel is enabled, the reception will start from the write pointer and continue to the read pointer.

40.5.6 AMBA AHB error

An AHB error response occurring on the AMBA AHB bus while data is being stored will result in an RxError interrupt.

If the FifoCONF.ABORT bit is set to 0b, the channel causing the AHB error will retry to store the received data till successful

If the FifoCONF.ABORT bit is set to 1b, the channel causing the AHB error will be disabled (FifoRx-CTRL.ENABLE is cleared automatically to 0b). The write pointer can be used to determine which address caused the AHB error. The interface will not start any new read accesses to the FIFO. Any ongoing FIFO access will be completed and the data will be stored in the local receive buffer. The FifoRxSTAT.ONGOING bit will be cleared. When the receive channel is re-enabled, the reception and storage of data will resume at the position where it was disabled, without losing any data.

40.5.7 Enable and disable

When an enabled receive channel is disabled (FifoRxCTRL.ENABLE=0b), any ongoing data storage on the AHB bus will not be aborted, and no new storage will be started. If the data is stored successfully, the write pointer (FifoRxWR.WRITE) is automatically incremented. Any associated interrupts will be generated. The interface will not start any new read accesses to the FIFO. Any ongoing FIFO access will be completed.

The channel can be re-enabled again without the need to re-configure the address, size and pointers. No data reception is performed while the channel is not enabled.

The progress of the any ongoing access can be observed via the FifoRxSTAT.ONGOING bit. Note that the there might be data left in the local store-buffer in the FIFO controller. This can be observed via the FifoRxSTAT.RxByteCntr field. The data will not be lost if the channel is not reconfigured before re-enabled.

To empty this data from the local store-buffer to the external memory, the channel needs to be ren-abled. By setting the FifoRxIRQ.IRQ field to match the value of the FifoRxWR.WRITE field plus the value of the FifoRxSTAT.RxByteCntr field, an emptying to the external memory is forced of any data temporarily stored in the local store-buffer. Note however that additional data could be received in the local store-buffer when the channel is re-enabled.

The FifoRxSTAT.ONGOING must be 0b before the channel can be re-configured safely (i.e. changing address, size or read/write pointers).

40.5.8 Interrupts

During reception several interrupts can be generated:

- RxFull: Successful reception of all data possible to store in buffer
- RxIrq: Successful reception of a predefined number of data
- RxError: AHB access error during reception
- RxParity: Parity error during reception

The RxFull and RxIrq interrupts are only generated as the result of a successful data reception, after the FifoRxWR.WRITE pointer has been incremented.

40.6 Operation

40.6.1 Global reset and enable

When the FifoCTRL.RESET bit is set to 1b, a reset of the core is performed. The reset clears all the register fields to their default values. Any ongoing data transfers will be aborted.

40.6.2 Interrupt

Seven interrupts are implemented by the FIFO interface:

Index:	Name:	Description:
0	TxIrq	Successful transmission of block of data
1	TxEmpty	Circular transmission buffer empty
2	TxError	AMBA AHB access error during transmission
3	RxIrq	Successful reception of block of data
4	RxFull	Circular reception buffer full
5	RxError	AMBA AHB access error during reception
6	RxParity	Parity error during reception

The interrupts are configured by means of the *pirq* VHDL generic. The setting of the *singleirq* VHDL generic results in a single interrupt output, instead of multiple, configured by the means of the *pirq* VHDL generic, and enables the read and write of the interrupt registers. When multiple interrupts are implemented, each interrupt is generated as a one system clock period long active high output pulse. When a single interrupt is implemented, it is generated as an active high level output.

40.6.3 Reset

After a reset the values of the output signals are as follows:

Signal:	Value after reset:
FIFOO.WEn	de-asserted
FIFOO.REn	de-asserted

40.6.4 Asynchronous interfaces

The following input signals are synchronized to Clk:

- FIFOI.EFn
- FIFOI.FFn
- FIFOI.HFn

40.7 Registers

The core is programmed through registers mapped into APB address space.

*Table 380.*GRFIFO registers

APB address offset	Register
0x000	Configuration Register
0x004	Status Register
0x008	Control Register
0x020	Transmit Channel Control Register
0x024	Transmit Channel Status Register
0x028	Transmit Channel Address Register
0x02C	Transmit Channel Size Register
0x030	Transmit Channel Write Register
0x034	Transmit Channel Read Register
0x038	Transmit Channel Interrupt Register
0x040	Receive Channel Control Register
0x044	Receive Channel Status Register
0x048	Receive Channel Address Register
0x04C	Receive Channel Size Register
0x050	Receive Channel Write Register
0x054	Receive Channel Read Register
0x058	Receive Channel Interrupt Register
0x060	Data Input Register
0x064	Data Output Register
0x068	Data Direction Register
0x100	Pending Interrupt Masked Status Register
0x104	Pending Interrupt Masked Register
0x108	Pending Interrupt Status Register
0x10C	Pending Interrupt Register
0x110	Interrupt Mask Register
0x114	Pending Interrupt Clear Register

40.7.1 Configuration Register [FifoCONF] R/W

*Table 381.*Configuration Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
									Abort	DW		Par-ity	WS		

Field:	Description:	
6:	ABORT	Abort transfer on AHB ERROR
5-4:	DW	Data width: 00b = none 01b = 8 bitFIFOO.Dout[7:0], FIFOI.Din[7:0] 10b = 16 bitFIFOO.Dout[15:0] FIFOI.Din[15:0]

		11b = spare/none
3:	PARITY	Parity type:
		0b = even
		1b = odd
2-0:	WS	Number of wait states, 0 to 7

All bits are cleared to 0 at reset.

Note that the transmit or receive channel active during the AMBA AHB error is disabled if the ABORT bit is set to 1b. Note that all accesses on the affected channel will be disabled after an AMBA AHB error occurs while the ABORT bit is set to 1b. The accesses will be disabled until the affected channel is re-enabled setting the FifoTxCTRL.ENABLE or FifoRxCTRL.ENABLE bit, respectively.

Note that a wait states corresponds to an additional clock cycle added to the period when the read or write strobe is asserted. The default asserted width is one clock period for the read or write strobe when WS=0. Note that an idle gap of one clock cycle is always inserted between read and write accesses, with neither the read nor the write strobe being asserted.

Note that an additional gap of one clock cycle with the read or write strobe de-asserted is inserted between two accesses when WS is equal to or larger than 100b.

40.7.2 Status Register [FifoSTAT] R

*Table 382.*Status register

31 28	27 24	23 16
TxChannels	RxChannels	-

15 6	5	4 0
-	SingleIrq	-

31-28:	TxChannels	Number of TxChannels -1, 4-bit
27-24:	RxChannels	Number of RxChannels -1, 4-bit
5:	SingleIrq	Single interrupt output and interrupt registers when set to 1

40.7.3 Control Register [FifoCTRL] R/W

*Table 383.*Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														Rese t	

1:	RESET	Reset complete FIFO interface, all registers

All bits are cleared to 0 at reset.

Note that RESET is read back as 0b.

40.7.4 Transmit Channel Control Register [FifoTxCTRL] R/W

*Table 384.*Transmit Channel Control Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
															Enable

0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that in the case of an AHB bus error during an access while fetching transmit data, and the Fifo-Conf.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing data writes to the FIFO are not aborted.

40.7.5 Transmit Channel Status Register [FifoTxSTAT] R

*Table 385.*Transmit Channel Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									TxOnGoing		TxIrq	TxEmpty	TxError	FF	HF

6:	TxOnGoing	Access ongoing
4:	TxIrq	Successful transmission of block of data
3:	TxEmpty	Transmission buffer has been emptied
2:	TxError	AMB AHB access error during transmission
1:	FF	FIFO Full Flag
0:	HF	FIFO Half-Full Flag

All bits are cleared to 0 at reset.

The following sticky status bits are cleared when the register has been read:

- TxIrq, TxEmpty and TxError.

40.7.6 Transmit Channel Address Register [FifoTxADDR] R/W

*Table 386.*Transmit Channel Address Register

31 — 10	9 — 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

40.7.7 Transmit Channel Size Register [FifoTxSIZE] R/W

*Table 387.*Transmit Channel Size Register

31 17	16 6	5 0
	SIZE	

16-6:	SIZE	Size of circular buffer, in number of 64 bytes block

All bits are cleared to 0 at reset.

Valid SIZE values are 0, and between 1 and 1024. Note that the resulting behavior of invalid SIZE values is undefined.

Note that only SIZE*64-4 bytes can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field is configurable indirectly by means of the VHDL generic (ptrwidth) which sets the width of the read and write data pointers. In the above example VHDL generic ptrwidth=16, making the SIZE field 11 bits wide.

40.7.8 Transmit Channel Write Register [FifoTxWR] R/W

*Table 388.*Transmit Channel Write Register

31 16	15 0
	WRITE

15-0:	WRITE	Pointer to last written byte + 1

All bits are cleared to 0 at reset.

The WRITE field is written to in order to initiate a transfer, indicating the position +1 of the last byte to transmit.

Note that it is not possible to fill the buffer. There is always one word position in buffer unused. Software is responsible for not over-writing the buffer on wrap around (i.e. setting WRITE=READ).

Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

40.7.9 Transmit Channel Read Register [FifoTxRD] R/W

*Table 389.*Transmit Channel Read Register

31 16	15 0
	READ

15-0:	READ	Pointer to last read byte + 1

All bits are cleared to 0 at reset.

The READ field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last byte transmitted.

Note that the READ field can be used to read out the progress of a transfer.

Note that the READ field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the READ field can be automatically incremented even if the transmit channel has been disabled, since the last requested transfer is not aborted until completed.

Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

40.7.10 Transmit Channel Interrupt Register [FifoTxIRQ] R/W

*Table 390.*Transmit Channel Interrupt Register

31 16	15 0
	IRQ

15-0:	IRQ	Pointer+1 to a byte address from which the read of transmitted data shall result in an interrupt

All bits are cleared to 0 at reset.

Note that this indicates that a programmed amount of data has been sent. Note that the LSB may be ignored for 16-bit wide FIFO devices.

The field is implemented as relative to the buffer base address (scaled with the SIZE field).

40.7.11 Receive Channel Control Register [FifoRxCTRL] R/W

*Table 391.*Receive Channel Control Register

31 2	1	0
		Enable

0:	ENABLE	Enable channel

All bits are cleared to 0 at reset.

Note that in the case of an AHB bus error during an access while storing receive data, and the Fifo-Conf.ABORT bit is 1b, then the ENABLE bit will be reset automatically.

At the time the ENABLE is cleared to 0b, any ongoing data reads from the FIFO are not aborted.

40.7.12 Receive Channel Status Register [FifoRxSTAT] R

*Table 392.*Receive Channel Status Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					RxByteCntr				RxOnGoing	RxParity	RxIrq	RxFull	RxError	EF	HF

10-8:	RxByteCntr	Number of bytes in local buffer
6:	RxOnGoing	Access ongoing
5:	RxParity	Parity error during reception
4:	RxIrq	Successful reception of block of data
3:	RxFull	Reception buffer has been filled
2:	RxError	AMB AHB access error during reception
1:	EF	FIFO Empty Flag
0:	HF	FIFO Half-Full Flag

All bits are cleared to 0 at reset.

The following sticky status bits are cleared when the register has been read:

- RxParity, RxIrq, RxFull and RxError.

The circular buffer is considered as full when there are two words or less left in the buffer.

40.7.13 Receive Channel Address Register [FifoRxADDR] R/W

*Table 393.*Receive Channel Address Register

31 10	9 0
ADDR	

31-10:	ADDR	Base address for circular buffer

All bits are cleared to 0 at reset.

40.7.14 Receive Channel Size Register [FifoRxSIZE] R/W

*Table 394.*Receive Channel Size Register

31 17	16 6	5 0
	SIZE	

16-6:	SIZE	Size of circular buffer, in number of 64 byte blocks

All bits are cleared to 0 at reset.

Valid SIZE values are 0, and between 1 and 1024. Note that the resulting behavior of invalid SIZE values is undefined.

Note that only SIZE*64-8 bytes can be stored simultaneously in the buffer. This is to simplify wrap-around condition checking.

The width of the SIZE field is configurable indirectly by means of the VHDL generic (ptrwidth) which sets the width of the read and write data pointers. In the above example VHDL generic ptrwidth=16, making the SIZE field 11 bits wide.

40.7.15 Receive Channel Write Register [FifoRxWR] R/W

*Table 395.*Receive Channel Write Register

31 16	15 0
	WRITE

15-0:	WRITE	Pointer to last written byte +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

The WRITE field is written to automatically when a transfer has been completed successfully, indicating the position +1 of the last byte received.

Note that the WRITE field can be used to read out the progress of a transfer.

Note that the WRITE field can be written to in order to set up the starting point of a transfer. This should only be done while the transmit channel is not enabled.

Note that the LSB may be ignored for 16-bit wide FIFO devices.

40.7.16 Receive Channel Read Register [FifoRxRD] R/W

*Table 396.*Receive Channel Read Register

31 16	15 0
	READ

15-0: READ Pointer to last read byte +1

All bits are cleared to 0 at reset.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

The READ field is written to in order to release the receive buffer, indicating the position +1 of the last byte that has been read out.

Note that it is not possible to fill the buffer. There is always one word position unused in the buffer. Software is responsible for not over-reading the buffer on wrap around (i.e. setting WRITE=READ).

Note that the LSB may be ignored for 16-bit wide FIFO devices

40.7.17 Receive Channel Interrupt Register [FifoRxIRQ] R/W

*Table 397.*Receive Channel Interrupt Register

31 16	15 0
	IRQ

15-0: IRQ Pointer+1 to a byte address to which the write of received data shall result in an interrupt

All bits are cleared to 0 at reset.

Note that this indicates that a programmed amount of data has been received.

The field is implemented as relative to the buffer base address (scaled with SIZE field).

Note that the LSB may be ignored for 16-bit wide FIFO devices.

Note that by setting the IRQ field to match the value of the Receive Channel Write Register.WRITE field plus the value of the Receive Channel Status Register.RxByteCntr field, an emptying to the external memory is forced of any data temporarily stored in the local buffer.

40.7.18 Data Input Register [FifoDIN] R

*Table 398.*Data Input Register

31 16	15 0
	DIN

15-0: DIN Input data *FIFOI.Din[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of FIFOI.Din[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

40.7.19 Data Output Register [FifoDOUT] R/W

*Table 399.*Data Output Register

31 16	15 0
	DOUT

15-0:	DOUT	Output data	*FIFOO.Dout[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of FIFOO.Dout[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

40.7.20 Data Register [FifoDDIR] R/W

*Table 400.*Data Direction Register

31 16	15 0
	DDIR

15-0:	DDIR	Direction: 0b = input = high impedance, 1b = output = driven	*FIFOO.Dout[15:0]*

All bits are cleared to 0 at reset.

Note that only the part of FIFOO.Dout[15:0] not used by the FIFO can be used as general purpose input output, see FifoCONF.DW.

Note that only bits dwidth-1 to 0 are implemented.

40.7.21 Interrupt registers

The interrupt registers give complete freedom to the software, by providing means to mask interrupts, clear interrupts, force interrupts and read interrupt status.

When an interrupt occurs the corresponding bit in the Pending Interrupt Register is set. The normal sequence to initialize and handle a module interrupt is:

- Set up the software interrupt-handler to accept an interrupt from the module.
- Read the Pending Interrupt Register to clear any spurious interrupts.
- Initialize the Interrupt Mask Register, unmasking each bit that should generate the module interrupt.
- When an interrupt occurs, read the Pending Interrupt Status Register in the software interrupt-handler to determine the causes of the interrupt.
- Handle the interrupt, taking into account all causes of the interrupt.
- Clear the handled interrupt using Pending Interrupt Clear Register.

Masking interrupts: After reset, all interrupt bits are masked, since the Interrupt Mask Register is zero. To enable generation of a module interrupt for an interrupt bit, set the corresponding bit in the Interrupt Mask Register.

Clearing interrupts: All bits of the Pending Interrupt Register are cleared when it is read or when the Pending Interrupt Masked Register is read. Reading the Pending Interrupt Masked Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register. Selected bits can be cleared by writing ones to the bits that shall be cleared to the Pending Interrupt Clear Register.

Forcing interrupts: When the Pending Interrupt Register is written, the resulting value is the original contents of the register logically OR-ed with the write data. This means that writing the register can force (set) an interrupt bit, but never clear it.

Reading interrupt status: Reading the Pending Interrupt Status Register yields the same data as a read of the Pending Interrupt Register, but without clearing the contents.

Reading interrupt status of unmasked bits: Reading the Pending Interrupt Masked Status Register yields the contents of the Pending Interrupt Register masked with the contents of the Interrupt Mask Register, but without clearing the contents.

The interrupt registers comprise the following:

• Pending Interrupt Masked Status Register	[FifoPIMSR]	R
• Pending Interrupt Masked Register	[FifoPIMR]	R
• Pending Interrupt Status Register	[FifoPISR]	R
• Pending Interrupt Register	[FifoPIR]	R/W
• Interrupt Mask Register	[FifoIMR]	R/W
• Pending Interrupt Clear Register	[FifoPICR]	W

*Table 401.*Interrupt registers

31 7	6	5	4	3	2	1	0
-	RxParity	RxError	RxFull	RxIrq	TxError	TxEmpty	TxIrq

6:	RxParity	Parity error during reception
5:	RxError	AMBA AHB access error during reception
4:	RxFull	Circular reception buffer full
3:	RxIrq	Successful reception of block of data
2:	TxError	AMBA AHB access error during transmission
1:	TxEmpty	Circular transmission buffer empty
0:	TxIrq	Successful transmission of block of data

All bits in all interrupt registers are reset to 0b after reset.

40.8 Vendor and device identifiers

The module has vendor identifier 0x01 (Gaisler Research) and device identifier 0x035. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

40.9 Configuration options

Table 402 shows the configuration options of the core (VHDL generics).

*Table 402.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the GRFIFO.	0 - NAHBIRQ-1	0
dwidth	Data width	16	16
ptrwidth	Width of data pointers	4 - 16	12
singleirq	Single interrupt output. A single interrupt is assigned to the AMBA APB interrupt bus instead of multiple separate ones. The single interrupt output is controlled by the interrupt registers which are also enabled with this VHDL generic.	0, 1	0
oepol	Output enable polarity	0, 1	1

40.10 Signal descriptions

Table 403 shows the interface signals of the core (VHDL ports).

*Table 403.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBI	*	Input	AMB master input signals	-
AHBO	*	Output	AHB master output signals	-
FIFOI	DIN[31:0]	Input	Data input	-
	PIN[3:0]		Parity input	-
	EFN		Empty flag	Low
	FFN		Full flag	Low
	HFN		Half flag	Low
FIFOO	DOUT[31:0]	Output	Data output	-
	DEN[31:0]		Data output enable	-
	POUT[3:0]		Parity output	-
	PEN[3:0]		Parity output enable	-
	WEN		Write enable	Low
	REN		Read enable	Low

* see GRLIB IP Library User’s Manual

40.11 Library dependencies

Table 404 shows the libraries used when instantiating the core (VHDL libraries).

*Table 404.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GRLIB	AMBA	Signals	AMBA signal definitions
GRLIB	AMBA	Signals, component	DMA2AHB definitions
GAISLER	MISC	Signals, component	Component declarations, signals.

40.12 Instantiation

This example shows how the core can be instantiated.

```
TBD
```

41 GRFPU - High-performance IEEE-754 Floating-point unit

41.1 Overview

GRFPU is a high-performance FPU implementing floating-point operations as defined in the IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and the SPARC V8 standard (IEEE-1754). Supported formats are single and double precision floating-point numbers. The advanced design combines two execution units, a fully pipelined unit for execution of the most common FP operations and a non-blocking unit for execution of divide and square-root operations.

The logical view of the GRFPU is shown in figure 140.

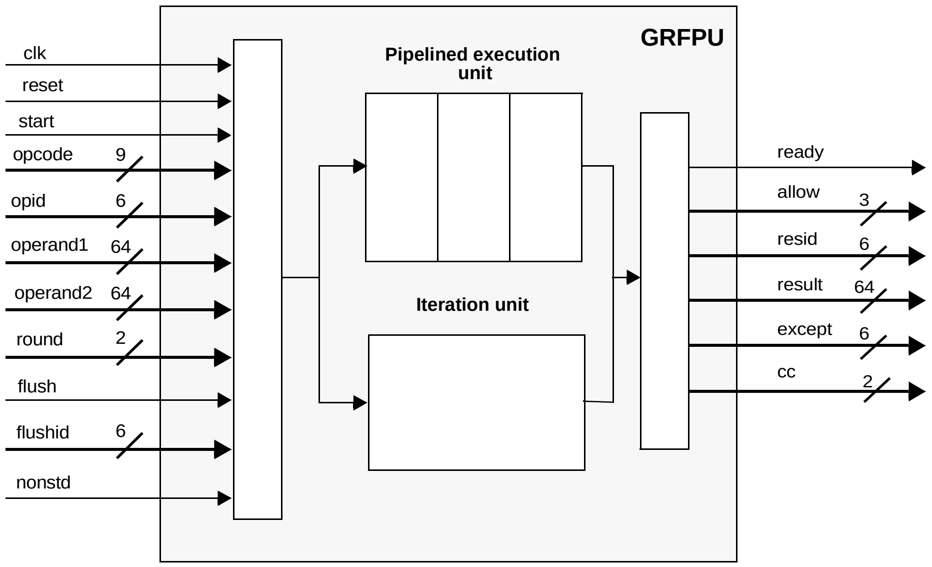


Figure 140. GRFPU Logical View

This document describes GRFPU from functional point of view. Chapter “Functional description” gives details about GRFPU’s implementation of the IEEE-754 standard including FP formats, operations, opcodes, operation timing, rounding and exceptions. “Signals and timing” describes the GRFPU interface and its signals. “GRFPU Control Unit” describes the software aspects of the GRFPU integration into a LEON processor through the GRFPU Control Unit - GRFPC. For implementation details refer to the white paper, “GRFPU - High Performance IEEE-754 Floating-Point Unit” (available at *www.gaisler.com*).

41.2 Functional description

41.2.1 Floating-point number formats

GRFPU handles floating-point numbers in single or double precision format as defined in the IEEE-754 standard with exception for denormalized numbers. See section 41.2.5 for more information on denormalized numbers.

41.2.2 FP operations

GRFPU supports four types of floating-point operations: arithmetic, compare, convert and move. The operations implement all FP instructions specified by SPARC V8 instruction set, and most of the operations defined in IEEE-754. All operations are summarized in table 405, with their opcodes, operands, results and exception codes. Throughputs and latencies and are shown in table 405.

Table 405.: GRFPU operations

Operation	**OpCode[8:0]**	**Op1**	**Op2**	**Result**	**Exceptions**	**Description**
Arithmetic operations						
FADDS FADDD	001000001 001000010	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Addition
FSUBS FSUBD	001000101 001000110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Subtraction
FMULS FMULD FSMULD	001001001 001001010 001101001	SP DP SP	SP DP SP	SP DP DP	UNF, NV, OF, UF, NX UNF, NV, OF, UF, NX UNF, NV, OF, UF	Multiplication, FSMULD gives exact double-precision product of two single-precision operands.
FDIVS FDIVD	001001101 001001110	SP DP	SP DP	SP DP	UNF, NV, OF, UF, NX	Division
FSQRTS FSQRTD	000101001 000101010	- -	SP DP	SP DP	UNF, NV, NX	Square-root
Conversion operations						
FITOS FITOD	011000100 011001000	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	011010001 011010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOI_RND FDTOI_RND	111010001 111010010	-	SP DP	INT	UNF, NV, NX	Floating-point to integer conversion. Rounding according to RND input.
FSTOD FDTOS	011001001 011000110	-	SP DP	DP SP	UNF, NV UNF, NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations						
FCMPS FCMPD	001010001 001010010	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	001010101 001010110	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move						
FABSS	000001001	-	SP	SP	-	Absolute value.
FNEGS	000000101	-	SP	SP	-	Negate.
FMOVS	000000001		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number	CC - condition codes, see table 408
DP - double precision floating-point number	UNF, NV, OF, UF, NX - floating-point exceptions, see section 41.2.3
INT - 32 bit integer	

Arithmetic operations include addition, subtraction, multiplication, division and square-root. Each arithmetic operation can be performed in single or double precision formats. Arithmetic operations have one clock cycle throughput and a latency of four clock cycles, except for divide and square-root operations, which have a throughput of 16 - 25 clock cycles and latency of 16 - 25 clock cycles (see table 406). Add, sub and multiply can be started on every clock cycle, providing high throughput for these common operations. Divide and square-root operations have lower throughput and higher latency due to complexity of the algorithms, but are executed in parallel with all other FP operations in a non-blocking iteration unit. Out-of-order execution of operations with different latencies is easily handled through the GRFPU interface by assigning an id to every operation which appears with the result on the output once the operation is completed (see section 3.2).

Table 406.: Throughput and latency

Operation	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD, FMULS, FMULD, FSMULD	1	4
FITOS, FITOD, FSTOI, FSTOI_RND, FDTOI, FDTOI_RND, FSTOD, FDTOS	1	4
FCMPS, FCMPD, FCMPES, FCMPED	1	4
FDIVS	16	16
FDIVD	16.5 (15/18)*	16.5 (15/18)*
FSQRTS	24	24
FSQRTD	24.5 (23/26)*	24.5 (23/26)*

* Throughput and latency are data dependant with two possible cases with equal statistical possibility.

Conversion operations execute in a pipelined execution unit and have throughput of one clock cycle and latency of four clock cycles. Conversion operations provide conversion between different floating-point numbers and between floating-point numbers and integers.

Comparison functions offering two different types of quiet Not-a-Numbers (QNaNs) handling are provided. Move, negate and absolute value are also provided. These operations do not ever generate unfinished exception (unfinished exception is never signaled since compare, negate, absolute value and move handle denormalized numbers).

41.2.3 Exceptions

GRFPU detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented. Overflow (OF) and underflow (UF) are detected before rounding. If an operation underflows the result is flushed to zero (GRFPU does not support denormalized numbers or gradual underflow). A special Unfinished exception (UNF) is signaled when one of the operands is a denormalized number which is not handled by the arithmetic and conversion operations.

41.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

41.2.5 Denormalized numbers

Denormalized numbers are not handled by the GRFPU arithmetic and conversion operations. A system (microprocessor) with the GRFPU could emulate rare cases of operations on denormals in software using non-FPU operations. A special Unfinished exception (UNF) is used to signal an arithmetic or conversion operation on the denormalized numbers. Compare, move, negate and absolute value operations can handle denormalized numbers and do not raise the unfinished exception. GRFPU does not generate any denormalized numbers during arithmetic and conversion operations on normalized numbers. If the infinitely precise result of an operation is a tiny number (smaller than minimum value representable in normal format) the result is flushed to zero (with underflow and inexact flags set).

41.2.6 Non-standard Mode

GRFPU can operate in a non-standard mode where all denormalized operands to arithmetic and conversion operations are treated as (correctly signed) zeroes. Calculations are performed on zero operands instead of the denormalized numbers obeying all rules of the floating-point arithmetics including rounding of the results and detecting exceptions.

41.2.7 NaNs

GRFPU supports handling of Not-a-Numbers (NaNs) as defined in the IEEE-754 standard. Operations on signaling NaNs (SNaNs) and invalid operations (e.g. inf/inf) generate the Invalid exception and deliver QNaN_GEN as result. Operations on Quiet NaNs (QNaNs), except for FCMPES and FCMPED, do not raise any exceptions and propagate QNaNs through the FP operations by delivering NaN-results according to table 407. QNaN_GEN is 0x7fffe00000000000 for double precision results and 0x7fff0000 for single precision results.

Table 407.: Operations on NaNs

	Operand 2			
Operand 1		FP	QNaN2	SNaN2
	none	FP	QNaN2	QNaN_GEN
	FP	FP	QNaN2	QNaN_GEN
	QNaN1	QNaN1	QNaN2	QNaN_GEN
	SNaN1	QNaN_GEN	QNaN_GEN	QNaN_GEN

41.3 Signal descriptions

Table 408 shows the interface signals of the core (VHDL ports). All signals are active high except for RST which is active low.

Table 408.: Signal descriptions

Signal	I/O	Description
CLK	I	Clock
RST	I	Reset
START	I	Start an FP operation on the next rising clock edge
NONSTD	I	Nonstandard mode. Denormalized operands are converted to zero.
OPCODE[8:0]	I	FP operation. For codes see table 405.
OPID[7:0]	I	FP operation id. Every operation is associated with an id which will appear on the RESID output when the FP operation is completed. This value shall be incremented by 1 (with wrap-around) for every started FP operation. If flushing is used, FP operation id is 6 -bits wide (OPID[5:0] are used for id, OPID[7:6] are tied to “00”). If flushing is not used (input signal FLUSH is tied to ‘0’), all 8-bits (OPID[7:0]) are used.
OPERAND1[63:0] OPERAND2[63:0]	I	FP operation operands are provided on these one or both of these inputs. All 64 bits are used for IEEE-754 double precision floating-point numbers, bits [63:32] are used for IEEE-754 single precision floating-point numbers and 32-bit integers.
ROUND[1:0]	I	Rounding mode. 00 - rounding-to-nearest, 01 - round-to-zero, 10 - round-to-+inf, 11 - round-to--inf.
FLUSH	I	Flush FP operation with FLUSHID.
FLUSHID[5:0]	I	Id of the FP operation to be flushed.
READY	O	The result of a FP operation will be available at the end of the next clock cycle.
ALLOW[2:0]	O	Indicates allowed FP operations during the next clock cycle. ALLOW[0] - FDIVS, FDIVD, FSQRTS and FSQRTD allowed ALLOW[1] - FMULS, FMULD, FSMULD allowed ALLOW[2] - all other FP operations allowed
RESID[7:0]	O	Id of the FP operation whose result appears at the end of the next clock cycle.
RESULT[63:0]	O	Result of an FP operation. If the result is double precision floating-point number all 64 bits are used, otherwise single precision or integer result appears on RESULT[63:32].
EXCEPT[5:0]	O	Floating-point exceptions generated by an FP operation. EXC[5] - Unfinished FP operation. Generated by an arithmetic or conversion operation with denormalized input(s). EXC[4] - Invalid exception. EXC[3] - Overflow. EXC[2] - Underflow. EXC[1] - Division by zero. EXC[0] - Inexact.
CC[1:0]	O	Result (condition code) of an FP compare operation. 00 - equal 01 - operand1 < operand2 10 - operand1 > operand2 11 - unordered

41.4 Timing

An FP operation is started by providing the operands, opcode, rounding mode and id before rising edge. The operands need to be provided a small set-up time before a rising edge while all other signals are latched on rising edge.

The FPU is fully pipelined and a new operation can be started every clock cycle. The only exceptions are divide and square-root operations which require 16 to 26 clock cycles to complete, and which are not pipelined. Division and square-root are implemented through iterative series expansion algorithm. Since the algorithms basic step is multiplication the floating-point multiplier is shared between multiplication, division and square-root. Division and square-root do not occupy the multiplier during the whole operation and allow multiplication to be interleaved and executed parallelly with division or square-root.

One clock cycle before an operation is completed, the output signal RDY is asserted to indicate that the result of an FPU operation will appear on the output signals at the end of the next cycle. The id of the operation to be completed and allowed operations are reported on signals RESID and ALLOW. During the next clock cycle the result appears on RES, EXCEPT and CC outputs.

Table 141 shows signal timing during four arithmetic operations on GRFPU.

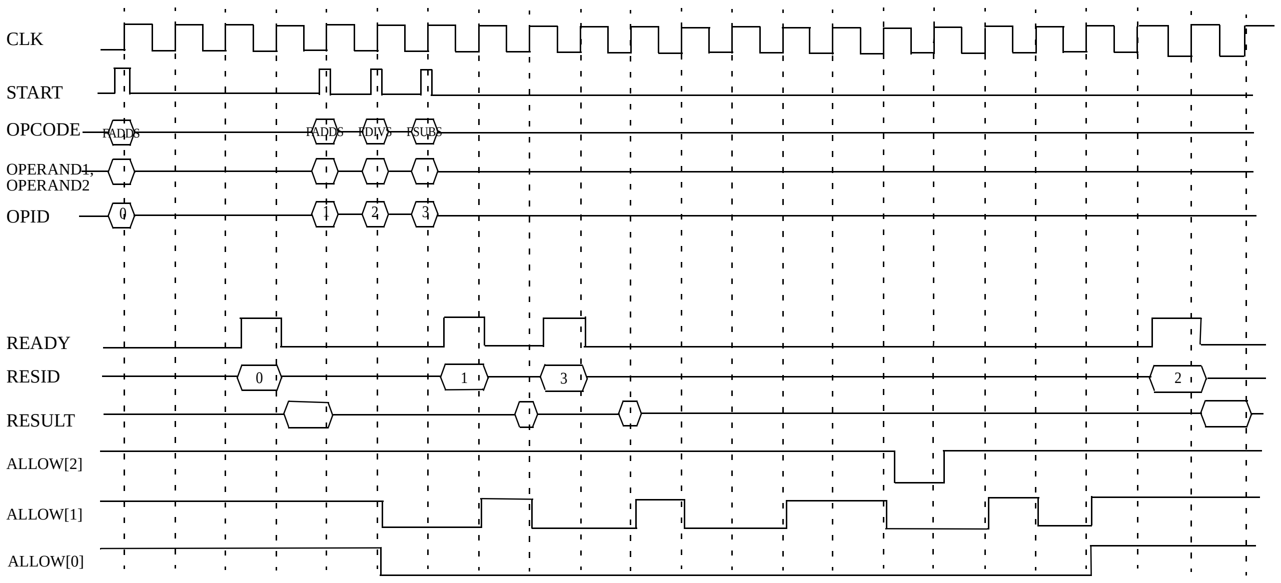


Figure 141. Signal timing

41.5 Shared FPU

In multi-processor systems, asingle GRFPU can be shared between multiple CPU cores providing an area efficient solution. In this configuration, the GRFPU is extended with a wrapper. Each CPU core issues a request to execute an FP operation to the wrapper, which performs fair arbitration using the round-robin algorithm.

In shared FPU configuration, GRFPU uses an 8 bit wide id for each operation. The three high-order bits are used to identify the CPU core which issued the FP operation, while the five low-order bits are used to enumerate FP operations issued by one core. FP operation flushing is not possible in shared FPU configuration.

42 GRFPC - GRFPU Control Unit

The GRFPU Control Unit (GRFPC) is used to attach the GRFPU to the LEON integer unit (IU). GRFPC performs scheduling, decoding and dispatching of the FP operations to the GRFPU as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ). Floating-point operations are executed in parallel with other integer instructions, the LEON integer pipeline is only stalled in case of operand or resource conflicts.

In the FT-version, all registers are protected with TMR and the floating-point register file is protected using parity coding.

42.1 Floating-Point register file

The GRFPU floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

42.2 Floating-Point State Register (FSR)

The GRFPC manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the GRFPU conforming to the SPARC V8 specification and the IEEE-754 standard. Implementation-specific parts of the FSR managing are the NS (non-standard) bit and *ftt* field.

If the NS (non-standard) bit of the FSR register is set, all floating-point operations will be performed in non-standard mode as described in section 41.2.6. When the NS bit is cleared all operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:
- unimplemented_FPop: all FPop operations are implemented
- hardware_error: non-resumable hardware error
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

GRFPU implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise.

The FSR is accessed using LDFSR and STFSR instructions.

42.3 Floating-Point Exceptions and Floating-Point Deferred-Queue

GRFPU implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- an operation on denormalized floating-point numbers (in standard IEEE-mode) raises unfinished_FPop floating-point exception
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.

The trap is deferred to one of the floating-point instructions (FPop, FP load/store, FP branch) following the trap-inducing instruction (note that this may not be next floating-point instruction in the program order due to exception-detecting mechanism and out-of-order instruction execution in the GRFPC). When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction and up to seven FPop instructions that were dispatched in the GRFPC but did not complete.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. The STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code. All instructions in the FQ are FPop type instructions. The first access to the FQ gives a double-word with the trap-inducing instruction, following double-words contain pending floating-point instructions. Supervisor software should emulate FPops from the FQ in the same order as they were read from the FQ.

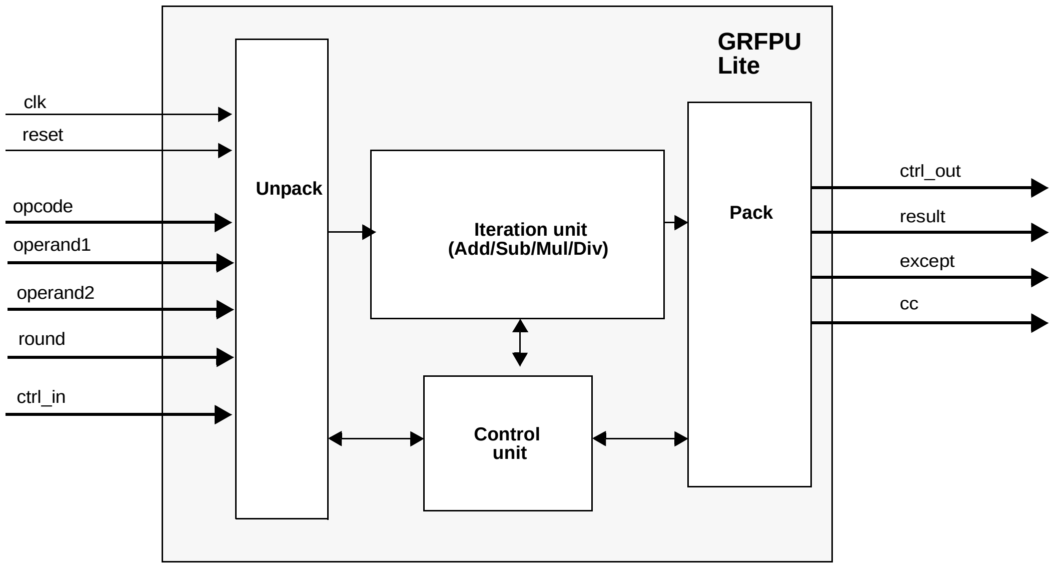
Note that instructions in the FQ may not appear in the same order as the program order since GRFPU executes floating-point instructions out-of-order. A floating-point trap is never deferred past an instruction specifying source registers, destination registers or condition codes that could be modified by the trap-inducing instruction. Execution or emulation of instructions in the FQ by the supervisor software gives therefore the same FPU state as if the instructions were executed in the program order.

43 GRFPU Lite - IEEE-754 Floating-Point Unit

43.1 Overview

The GRFPU Lite floating-point unit implements floating-point operations as defined in IEEE Standard for Binary Floating-Point Arithmetic (IEEE-754) and SPARC V8 standard (IEEE-1754).

Supported formats are single and double precision floating-point numbers. The floating-point unit is not pipelined and executes one floating-point operation at a time.



43.2 Functional Description

43.2.1 Floating-point number formats

The floating-point unit handles floating-point numbers in single or double precision format as defined in IEEE-754 standard.

43.2.2 FP operations

The floating-point unit supports four types of floating-point operations: arithmetic, compare, convert and move. The operations implement all FP instructions specified by SPARC V8 instruction set. All operations are summarized in the table below.

Table 409.:Floating-point operations

Operation	Op1	Op2	Result	Exceptions	Description
Arithmetic operations					
FADDS FADDD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Addition
FSUBS FSUBD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Subtraction
FMULS FMULD FSMULD	SP DP SP	SP DP SP	SP DP DP	NV, OF, UF, NX NV, OF, UF, NX NV,OF, UF	Multiplication
FDIVS FDIVD	SP DP	SP DP	SP DP	NV, OF, UF, NX	Division
FSQRTS FSQRTD	- -	SP DP	SP DP	NV, NX	Square-root
Conversion operations					
FITOS FITOD	-	INT	SP DP	NX -	Integer to floating-point conversion
FSTOI FDTOI	-	SP DP	INT	NV, NX	Floating-point to integer conversion. The result is rounded in round-to-zero mode.
FSTOD FDTOS	-	SP DP	DP SP	NV NV, OF, UF, NX	Conversion between floating-point formats
Comparison operations					
FCMPS FCMPD	SP DP	SP DP	CC	NV	Floating-point compare. Invalid exception is generated if either operand is a signaling NaN.
FCMPES FCMPED	SP DP	SP DP	CC	NV	Floating point compare. Invalid exception is generated if either operand is a NaN (quiet or signaling).
Negate, Absolute value and Move					
FABSS	-	SP	SP	-	Absolute value.
FNEGS	-	SP	SP	-	Negate.
FMOVS		SP	SP	-	Move. Copies operand to result output.

SP - single precision floating-point number	CC - condition codes
DP - double precision floating-point number	NV, OF, UF, NX - floating-point exceptions, see section 43.2.3
INT - 32 bit integer	

Below is a table of worst-case throughput of the floating point unit.

*Table 410.*Worst-case instruction timing

Instruction	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

43.2.3 Exceptions

The floating-point unit detects all exceptions defined by the IEEE-754 standard. This includes detection of Invalid Operation (NV), Overflow (OF), Underflow (UF), Division-by-Zero (DZ) and Inexact (NX) exception conditions. Generation of special results such as NaNs and infinity is also implemented.

43.2.4 Rounding

All four rounding modes defined in the IEEE-754 standard are supported: round-to-nearest, round-to-+inf, round-to--inf and round-to-zero.

44 GRLFPC - GRFPU Lite Floating-point unit Controller

44.1 Overview

The GRFPU Lite Floating-Point Unit Controller (GRLFPC) is used to attach the GRFPU Lite floating-point unit (FPU) to the LEON integer unit (IU). It performs decoding and dispatching of the floating-point (FP) operations to the floating-point unit as well as managing the floating-point register file, the floating-point state register (FSR) and the floating-point deferred-trap queue (FQ).

The GRFPU Lite floating-point unit is not pipelined and executes only one instruction at a time. To improve performance, the controller (GRLFPC) allows the GRFPU Lite floating-point unit to execute in parallel with the processor pipeline as long as no new floating-point instructions are pending.

44.2 Floating-Point register file

The floating-point register file contains 32 32-bit floating-point registers (%f0-%f31). The register file is accessed by floating-point load and store instructions (LDF, LDDF, STD, STDF) and floating-point operate instructions (FPop).

In the FT-version, the floating-point register file is protected using 4-bit parity per 32-bit word. The controller is capable of detecting and correcting one bit error per byte. Errors are corrected using the instruction restart function in the IU.

44.3 Floating-Point State Register (FSR)

The controller manages the floating-point state register (FSR) containing FPU mode and status information. All fields of the FSR register as defined in SPARC V8 specification are implemented and managed by the controller conform to the SPARC V8 specification and IEEE-754 standard.

The non-standard bit of the FSR register is not used, all floating-point operations are performed in standard IEEE-compliant mode.

Following floating-point trap types never occur and are therefore never set in the ftt field:
- unimplemented_FPop: all FPop operations are implemented
- unfinished_FPop: all FPop operation complete with valid result
- invalid_fp_register: no check that double-precision register is 0 mod 2 is performed

The controller implements the *qne* bit of the FSR register which reads 0 if the floating-point deferred-queue (FQ) is empty and 1 otherwise. The FSR is accessed using LDFSR and STFSR instructions.

44.4 Floating-Point Exceptions and Floating-Point Deferred-Queue

The floating-point unit implements the SPARC deferred trap model for floating-point exceptions (fp_exception). A floating-point exception is caused by a floating-point instruction performing an operation resulting in one of following conditions:

- an operation raises IEEE floating-point exception (ftt = IEEE_754_exception) e.g. executing invalid operation such as 0/0 while the NVM bit of the TEM field id set (invalid exception enabled).
- sequence error: abnormal error condition in the FPU due to the erroneous use of the floating-point instructions in the supervisor software.
- hardware_error: uncorrectable parity error is detected in the FP register file

The trap is deferred to the next floating-point instruction (FPop, FP load/store, FP branch) following the trap-inducing instruction. When the trap is taken the floating-point deferred-queue (FQ) contains the trap-inducing instruction.

After the trap is taken the *qne* bit of the FSR is set and remains set until the FQ is emptied. STDFQ instruction reads a double-word from the floating-point deferred queue, the first word is the address of the instruction and the second word is the instruction code.

45 GRGPIO - General Purpose I/O Port

45.1 Overview

The general purpose input output port core is a scalable and provides optional interrupt support. The port width can be set to 2 - 32 bits through the *nbits* VHDL generic (i.e. *nbits* = 16). Interrupt generation and shaping is only available for those I/O lines where the corresponding bit in the *imask* VHDL generic has been set to 1.

Each bit in the general purpose input output port can be individually set to input or output, and can optionally generate an interrupt. For interrupt generation, the input can be filtered for polarity and level/edge detection.

It is possible to share GPIO pins with other signals. The output register can then be bypassed through the bypass register.

The figure 142 shows a diagram for one I/O line.

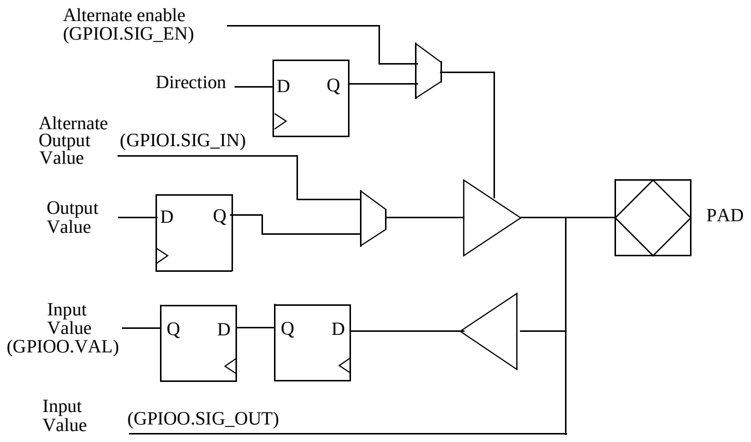


Figure 142. General Purpose I/O Port diagram

45.2 Operation

The I/O ports are implemented as bi-directional buffers with programmable output enable. The input from each buffer is synchronized by two flip-flops in series to remove potential meta-stability. The synchronized values can be read-out from the I/O port data register. They are also available on the GPIOO.VAL signals. The output enable is controlled by the I/O port direction register. A ‘1’ in a bit position will enable the output buffer for the corresponding I/O line. The output value driven is taken from the I/O port output register.

Each I/O port can drive a separate interrupt line on the APB interrupt bus. The interrupt number is equal to the I/O line index (PIO[1] = interrupt 1, etc.). The interrupt generation is controlled by three registers: interrupt mask, polarity and edge registers. To enable an interrupt, the corresponding bit in the interrupt mask register must be set. If the edge register is ‘0’, the interrupt is treated as level sensitive. If the polarity register is ‘0’, the interrupt is active low. If the polarity register is ‘1’, the interrupt is active high. If the edge register is ‘1’, the interrupt is edge-triggered. The polarity register then selects between rising edge (‘1’) or falling edge (‘0’).

A GPIO pin can be shared with other signals. The ports that should have the capability to be shared are specified with the *bypass* generic (the corresponding bit in the generic must be 1). The unfiltered inputs are available through GPIOO.SIG_OUT and the alternate output value must be provided in GPIOI.SIG_IN. The bypass register then controls whether the alternate output is chosen. The direction of the GPIO pin can also be shared, if the correspondig bit is set in the *bpdir* generic. In such case, the output buffer is enabled when GPIOI.SIG_EN is active.

45.3 Registers

The core is programmed through registers mapped into APB address space.

Table 411. General Purpose I/O Port registers

APB address offset	Register
0x00	I/O port data register
0x04	I/O port output register
0x08	I/O port direction register
0x0C	Interrupt mask register
0x10	Interrupt polarity register
0x14	Interrupt edge register
0x18	Bypass register

Table 412. I/O port data register

31 … 16	16-1 … 0
"000..0"	I/O port input value

16-1: 0 I/O port input value

Table 413. I/O port output register

31 … 16	16-1 … 0
"000..0"	I/O port output value

16-1: 0 I/O port output value

Table 414. I/O port direction register

31 … 16	16-1 … 0
"000..0"	I/O port direction value

16-1: 0 I/O port direction value (0=output disabled, 1=output enabled)

Table 415. Interrupt mask register

31 … 16	16-1 … 0
"000..0"	Interrupt mask

16-1: 0 Interrupt mask (0=interrupt masked, 1=intrrupt enabled)

Table 416. Interrupt polarity register

31 … 16	16-1 … 0
"000..0"	Interrupt polarity

16-1: 0 Interrupt polarity (0=low/falling, 1=high/rising)

Table 417. Interrupt edge register

31 … 16	16-1 … 0
"000..0"	Interrupt edge

16-1: 0 Interrupt edge (0=level, 1=edge)

Table 418. Bypass register

31 … 16	16-1 … 0
"000..0"	Bypass

Table 418. Bypass register

16-1: 0 Bypass. (0=normal output, 1=alternate output)

45.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x01A. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

45.5 Configuration options

Table 419 shows the configuration options of the core (VHDL generics).

*Table 419.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the GPIO unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 16#FFF#	0
pmask	The APB address mask	0 to 16#FFF#	16#FFF#
nbits	Defines the number of bits in the I/O port	1 to 32	8
imask	Defines which I/O lines are provided with interrupt generation and shaping	0 - 16#FFFF#	0
oepol	Select polarity of output enable signals. 0 = active low, 1 = active high.	0 - 1	0
syncrst	Selects between syncronous (1) or asyncronous (0) reset during power-up.	0 - 1	0
bypass	Defines which I/O lines are provided bypass capabilities	0 - 16#7FFFFFFF#	0
scantest	Enable scan support for asyncronous-reset flip-flops	0 - 1	0
bpdir	Defines which I/O lines are provided output enable bypass capabilities	0 - 16#7FFFFFFF#	0

45.6 Signal descriptions

Table 420 shows the interface signals of the core (VHDL ports).

*Table 420.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
GPIOO	OEN[31:0]	Output	I/O port output enable	see oepol
	DOUT[31:0]	Output	I/O port outputs	-
	VAL[31:0]	Output	The current (synchronized) value of the GPIO signals	-
	SIG_OUT[31:0]	Output	The current (unsynchronized) value of the GPIO signals	
GPIOI	DIN[31:0]	Input	I/O port inputs	-
	SIG_IN[31:0]	Input	Alternate output	-

* see GRLIB IP Library User's Manual

45.7 Library dependencies

Table 421 shows libraries used when instantiating the core (VHDL libraries).

*Table 421.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Signals, component	Component declaration

45.8 Component declaration

The core has the following component declaration.

```
ibrary gaisler;
use gaisler.misc.all;

entity grgpio is
  generic (
    pindex   : integer := 0;
    paddr    : integer := 0;
    pmask    : integer := 16#fff#;
    imask    : integer := 16#0000#;
    nbits    : integer := 16-- GPIO bits

  );
  port (
    rst    : in  std_ulogic;
    clk    : in  std_ulogic;
    apbi   : in  apb_slv_in_type;
    apbo   : out apb_slv_out_type;
    gpioi  : in  gpio_in_type;
    gpioo  : out gpio_out_type
  );
end;
```

45.9 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

signal gpti : gptimer_in_type;

begin

gpio0 : if CFG_GRGPIO_EN /= 0 generate     -- GR GPIO unit
    grgpio0: grgpio
      generic map( pindex => 11, paddr => 11, imask => CFG_GRGPIO_IMASK, nbits => 8)
      port map( rstn, clkm, apbi, apbo(11), gpioi, gpioo);

      pio_pads : for i in 0 to 7 generate
        pio_pad : iopad generic map (tech => padtech)
            port map (gpio(i), gpioo.dout(i), gpioo.oen(i), gpioi.din(i));
      end generate;
end generate;
```

46 GRSPW - SpaceWire codec with AHB host Interface and RMAP target

46.1 Overview

The SpaceWire core provides an interface between the AHB bus and a SpaceWire network. It implements the SpaceWire standard (ECSS-E-50-12A) with the protocol identification extension (ECSS-E-50-11).The optional Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-50-11).

The core is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface.

Currently, there is one DMA channel but the core can easily be extended to use separate DMA channels for specific protocols. The core can also be configured to have either one or two ports.

There can be up to four clock domains: one for the AHB interface (system clock), one for the transmitter and one or two for the receiver depending on the number of configured ports. The receiver clock can be twice as fast and the transmitter clock four times as fast as the system clock whose frequency should be at least 10 MHz.

The core only supports byte addressed 32-bit big-endian host systems.

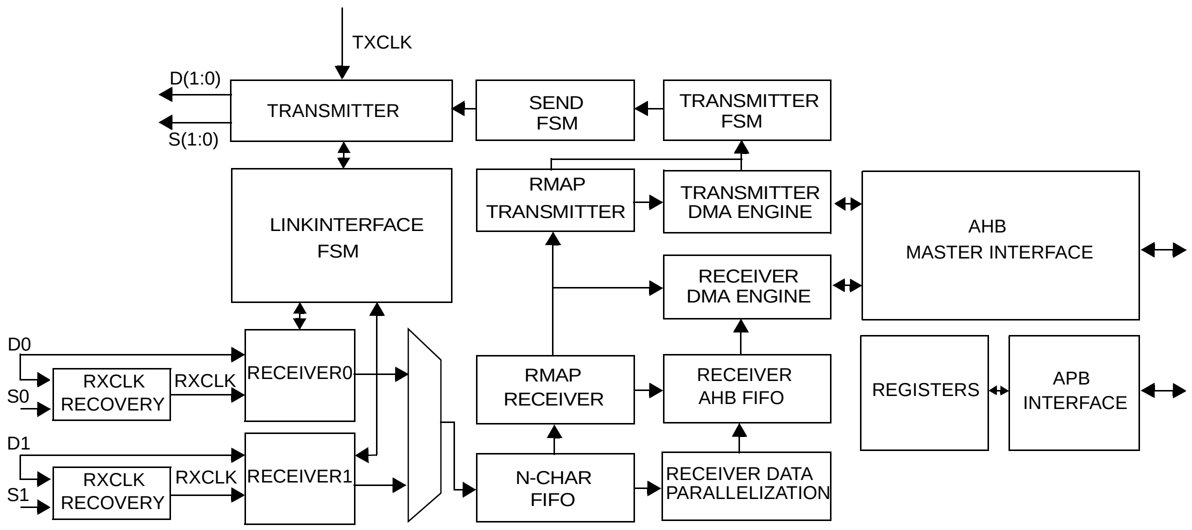


Figure 143. Block diagram

46.2 Operation

46.2.1 Overview

The main sub-blocks of the core are the link-interface, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 143.

The link interface consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP target handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The core is controlled by writing to a set of user registers through the APB interface and three signals: tick-in, rmapen and clkdiv10. The controlled parts are clock-generation, DMA engines, RMAP target and the link interface.

The link interface, DMA engines, RMAP target and AMBA interface are described in section 46.3, 46.4, 46.6 and 46.7 respectively.

46.2.2 Protocol support

The core only accepts packets with a destination address corresponding to the one set in the node address register. Packets with address mismatch will be silently discarded (except in promiscuous mode which is covered in section 46.4.10). The node address register is initialized to the default address 254 during reset. It can then be changed to some other value by writing to the register.

The core also requires that the byte following the destination address is a protocol identifier as specified in part 2 of the SpaceWire standard. It is used to determine to which DMA-channel a packet is destined. Currently only one channel is available to which all packets (except RMAP commands) are stored but the core is prepared to be easily expandable with more DMA channels. Figure 144 shows the packet type expected by the core.

RMAP (Protocol ID = 0x01) commands are handled separately from other packets if the hardware RMAP target is enabled. When enabled, all RMAP commands are processed, executed and replied in hardware. All RMAP replies received are still stored to the DMA channel. If the RMAP target is disabled, all packets are stored to the DMA channel. More information on the RMAP protocol support is found in section 46.6.

All packets arriving with the extended protocol ID (0x00) are stored to the DMA channel. This means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the core. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core.

Figure 144 shows a packet with a normal protocol identifier. The core also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

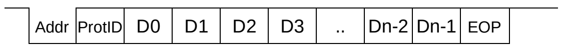


Figure 144. The SpaceWire packet with protocol ID that is expected by the GRSPW.

46.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 143.

46.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM in the host domain handles the exchange level.

The link interface FSM is controlled through the control register. The link can be disabled through the link disable bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started state when either the link start bit is set or when a NULL character has been received and the autostart bit is set.

The current state of the link interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 46.3.5).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

46.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 46.8.1. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 145. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals.

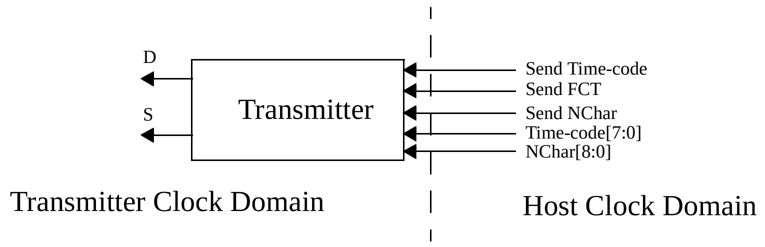


Figure 145. Schematic of the link interface transmitter.

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP CRC values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

46.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream on the data and strobe signals. It is also located in a separate clock domain which runs on a clock generated from the received data and strobe signals. More information on the clock-generation can be found in section 46.8.1.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the system clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 146. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

There are no signals going directly from the transmitter clock domain to the receiver clock domain and vice versa. All the synchronization is done to the system clock.

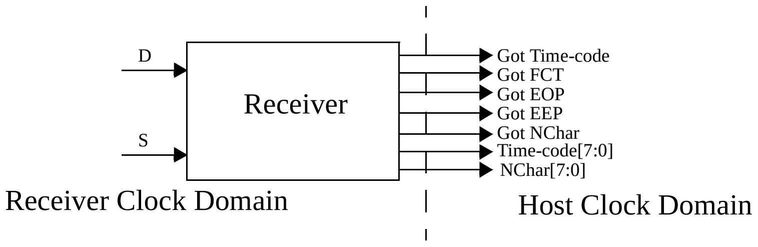


Figure 146. Schematic of the link interface receiver.

46.3.4 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 46.3.2. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port's interface signals (see figure 146) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the noportforce register is zero the portsel register bit selects the active link and when set to one it is determined by the current link activity. In the latter mode the port is changed when no activity is seen on the currently active link while there is activity on the deselected receive port. Activity is defined as a detected null. This definition is selected so that glitches (e.g. port unconnected) do not cause unwanted port switches.

46.3.5 Time interface

The time interface is used for sending Time-codes over the SpaceWire network and consists of a time-counter register, time-ctrl register, tick-in signal, tick-out signal, tick-in register field and a tick-out register field. There are also two control register bits which enable the time receiver and transmitter respectively.

Each Time-code sent from the core is a concatenation of the time-ctrl and the time-counter register. There is a timetxen bit which is used to enable Time-code transmissions. It is not possible to send time-codes if this bit is zero.

Received Time-codes are stored to the same time-ctrl and time-counter registers which are used for transmission. The timerxen bit in the control register is used for enabling time-code reception. No time-codes will be received if this bit is zero.

The two enable bits are used for ensuring that a node will not (accidentally) both transmit and receive time-codes which violates the SpaceWire standard. It also ensures that a the master sending time-codes on a network will not have its time-counter overwritten if another (faulty) node starts sending time-codes.

The time-counter register is set to 0 after reset and is incremented each time the tick-in signal is asserted for one clock-period and the timetxen bit is set. This also causes the link interface to send the new value on the network. Tick-in can be generated either by writing a one to the register field or by asserting the tick-in signal. A Tick-in should not be generated too often since if the time-code after the previous Tick-in has not been sent the register will not be incremented and no new value will be sent. The tick-in field is automatically cleared when the value has been sent and thus no new ticks should be generated until this field is zero. If the tick-in signal is used there should be at least 4 system-clock and 25 transmit-clock cycles between each assertion.

A tick-out is generated each time a valid time-code is received and the timerxen bit is set. When the tick-out is generated the tick-out signal will be asserted one clock-cycle and the tick-out register field is asserted until it is cleared by writing a one to it.

The current time counter value can be read from the time register. It is updated each time a Time-code is received and the timerxen bit is set. The same register is used for transmissions and can also be written directly from the APB interface.

The control bits of the Time-code are always stored to the time-ctrl register when a Time-code is received whose time-count is one more than the nodes current time-counter register. The time-ctrl register can be read through the APB interface. The same register is used during time-code transmissions.

It is possible to have both the time-transmission and reception functions enabled at the same time.

46.4 Receiver DMA engine

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels. Currently there is only one receive DMA channel available but the core has been written so that additional channels can be easily added if needed.

46.4.1 Basic functionality

The receiver DMA engine reads N-Chars from the N-Char FIFO and stores them to a DMA channel. Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the core it reads a descriptor from memory and stores the packet to the memory area pointed to by the descriptor. Then it stores status to the same descriptor and increments the descriptor pointer to the next one.

46.4.2 Setting up the core for reception

A few registers need to be initialized before reception can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum size of packet that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes. If the maximum length is set to zero the receiver will *not* function correctly.

The node address register needs to be set to hold the address of this SpaceWire node. Packets received with the incorrect address are discarded. Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

46.4.3 Setting up the descriptor table address

The core reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1 kbytes aligned address. It is also limited to be 1 kbytes in size which means the maximum number of descriptors is 128.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap automatically by setting a bit in the descriptors. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

46.4.4 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for this to happen.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the *rxunaligned* or *rmap* VHDL generic is set to 1 this restriction is removed and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 422. GRSPW receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24 … 0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise.
29	Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise.
28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.

Table 422. GRSPW receive descriptor word 0 (address offset 0x0)

25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

Table 423. GRSPW receive descriptor word 1 (address offset 0x4)

31 … 0
PACKETADDRESS

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap VHDL generics are both set to zero only bit 31 to 2 are used.

46.4.5 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 46.9). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the core might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

46.4.6 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded. If the receiver is enabled the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is ‘0’ and the nospill bit is ‘0’ the packets will be discarded. If nospill is one the core waits until rxdescav is set.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to ‘0’ and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will cause all packets received afterwards to be discarded. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the core when it reads a disabled descriptor.

46.4.7 Address recognition and packet handling

When the receiver N-Char FIFO is not empty, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address which is compared to the node address register. If it does not match, the complete packet is discarded (up to and including the next EOP/EEP).

If the address matches the next action taken depends on whether RMAP is enabled or not. If RMAP is disabled all packets are stored to the DMA channel and depending on the conditions mentioned in the previous section, the packet will be received or not. If the packet is received complete packet including address and protocol ID but excluding EOP/EEP is stored to the address indicated in the descriptor, otherwise the complete packet is discarded.

If RMAP is enabled the protocol ID and 3rd byte in the packet is first checked before any decisions are made. If incoming packet is an RMAP packet (ID = 0x01) and the command type field is 01b the packet is processed by the RMAP command handler which is described in section 46.6. Otherwise the packet is processed by the DMA engine as when RMAP is disabled.

At least 2 non EOP/EEP N-Chars need to be received for a packet to be stored to the DMA channel. If it is an RMAP packet 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than the minimum size are discarded.

46.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The core can also be made to generate an interrupt for this event as mentioned in section 46.4.4.

RMAP CRC logic is included in the implementation if the *rmapcrc* or *rmap* VHDL generic set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be a RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

46.4.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to ‘1’. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a satisfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

46.4.10 Promiscuous mode

The core supports a promiscuous mode where all the data received is stored to the DMA channel regardless of the node address and possible early EOPs/EEPs. This means that all non-eop/eep N-

Chars received will be stored to the DMA channel. The rxmaxlength register is still checked and packets exceeding this size will be truncated.

RMAP commands will still be handled by the RMAP target when promiscuous mode is enabled if the rmapen bit is set. If it is cleared, RMAP commands will also be stored to the DMA channel.

46.5 Transmitter DMA engine

The transmitter DMA engine handles transmission of data from the DMA channel to the SpaceWire network. There is one DMA channel available but the core has been written so that additional DMA channels can be easily added if needed.

46.5.1 Basic functionality

The transmit DMA engine reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the core reads them and transfer the amount data indicated.

46.5.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the core. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register should be written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

46.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver's corresponding register which was covered in section 46.4.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the core when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation. The CRCs are sent even if the corresponding length is zero.

When both header and data length are zero no packet is sent not even an EOP.

46.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the core to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

Table 424. GRSPW transmit descriptor word 0 (address offset 0x0)

31 18	17	16	15	14	13	12	11 8	7 0
RESERVED	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 18	RESERVED
17	Append data CRC (DC) - Append CRC calculated according to the RMAP specification after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.
15	Link error (LE) - A Link error occurred during the transmission of this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 425. GRSPW transmit descriptor word 1 (address offset 0x4)

31 0
HEADERADDRESS

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 426. GRSPW transmit descriptor word 2 (address offset 0x8)

31 24	23 0
RESERVED	DATALEN

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 427. GRSPW transmit descriptor word 3(address offset 0xC)

31 0
DATAADDRESS

Table 427. GRSPW transmit descriptor word 3(address offset 0xC)

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

46.5.5 The transmission process

When the txen bit is set the core starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

46.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1 kbytes limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

46.5.7 Error handling

Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

Link error

When a link error occurs during the transmission the remaining part of the packet is discarded up to and including the next EOP/EEP. When this is done status is immediately written (with the LE bit set) and the descriptor pointer is incremented. The link will be disconnected when the link error occurs but the core will automatically try to connect again provided that the link-start bit is asserted and the link-disabled bit is deasserted. If the LE bit in the DMA channel's control register is not set the transmitter

DMA engine will wait for the link to enter run-state and start a new transmission immediately when possible if packets are pending. Otherwise the transmitter will be disabled when a link error occurs during the transmission of the current packet and no more packets will be transmitted until it is enabled again.

46.6 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The core has an optional hardware RMAP target which is enabled with a VHDL generic. This section describes the basics of the RMAP protocol and the target implementation.

46.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

46.6.2 Implementation

The core includes a taget for RMAP commands which processes all incoming packets with protocol ID = 0x01 and type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The core implements all three commands defined in the standard with some restrictions. The implementation is based on draft F of the RMAP standard (the only exception being that error code 12 is not implemented). Support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The command handler will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

Packets with a mismatching destination logical address are never passed to the RMAP target. There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to ‘1’.

Detection of all error codes except code 12 is supported. When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 428.

*Table 428.*The order of error detection in case of multiple errors in the GRSPW. The error detected first has number 1.

Detection Order	Error Code	Error
1	2	Unused RMAP packet type or command code
2	3	Invalid destination key
3	9	Verify buffer overrun
4	11	RMW data length error
5	10	Authorization failure
6*	1	General Error (AHB errors during non-verified writes)
7	5/7	Early EOP / EEP (if early)
8	4	Invalid Data CRC
9	1	General Error (AHB errors during verified writes or RMW)
10	7	EEP
11	6	Cargo Too Large
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmission. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 8) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

46.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 B and the address must be aligned to the size. That is 1 B writes can be done to any address, 2 B must be halfword aligned, 3 B are not allowed and 4 B writes must be word aligned. Since there will always be only one AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

46.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

46.6.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

46.6.6 Control

The RMAP command handler mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP command handler. When it is set to ‘0’ no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the command handler stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the command handler to only use one buffer which prevents this situation.

The last control option for the command handler is the possibility to set the destination key which is found in a separate register.

*Table 429.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incrementing address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Modify-Write incrementing address	Executed normally. If length is not one of the allowed rmw values nothing is done and error code is set to 11. If the length was correct, alignment restrictions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, single-address, do not verify before writing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incrementing address, do not verify before writing, no acknowledge	Executed normally. No restrictions. No reply is sent.
0	1	1	0	1	0	Write, single-address, do not verify before writing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.

*Table 429.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	1	1	0	1	1	Write, incre-menting address, do not verify before writ-ing, send acknowledge	Executed normally. No restric-tions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incre-menting address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incre-menting address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

46.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 46.9. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writ-ing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the *rmap* or *rxunaligned* VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

46.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

46.7.2 AHB master interface

The core contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

if rmap and rxunaligned are disabledThe AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

46.8 Synthesis and hardware

46.8.1 Clock-generation

Figure 147 shows the clock recovery scheme for the receiver. Data and strobe are coupled directly from their pads to an xor gate which generates the clock. The output from the xor is then connected to a clock network. The specific type of clock network depends on the technology used. The xor gate is actually all that logically belongs to the Rx clock recovery module in figure 147.

The clock output drives all flip-flops in the receiver module found in figure 143. The data signal which is used for generating the clock is also coupled to the data inputs of several flip-flops clocked by the Rx clock as seen in figure 147. Care must be taken so that the delay from the data and strobe signals through the clock network are longer than the delay to the data input + setup time.

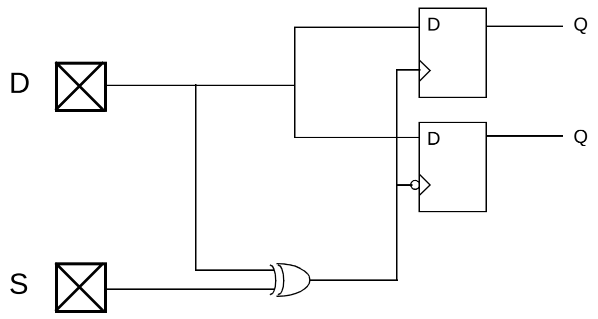


Figure 147. The clocking scheme for the receiver. The clock is

The transmitter clock is generated from the txclk input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the txclk signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called clkdiv10 which sets the clock divisor value during initialization and the reset value for the user accessible clock divisor register. The user register value will be used in run-state. The resulting tx clock frequency will be txclk/(clock divisor value+1). So if no clock division is wanted, the clock divisor should be set to 0.

Since only integer values are allowed for the clock division and the required init-frequency is 10 Mhz the frequency of the txclk input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum txclk frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

46.8.2 Timers

There are two timers in the core: one for generating the 6.4/12.8 us periods and one for disconnect timing. The system clock frequency must be at least 10 MHz to guarantee disconnect timing limits.

There are two user accessible registers which are used to the set the number of clock cycles used for the timeout periods. These registers are described in section 46.9.

The reset value for the timer registers can be set in two different ways selected by the usegen VHDL generic. If usegen is set to 1, the sysfreq VHDL generic is used to generate reset values for the disconnect, 6.4 us and 12.8 us timers. Otherwise, the input signals dcrstval and timerrstval will be used as reset values. If the system clock frequency is 10 MHz or above the disconnect time will be within the limits specified in the SpaceWire standard.

46.8.3 Synchronization

The VHDL generic nsync selects how many synchronization registers are used between clock domains. The default is one and should be used when maximum performance is needed. It allows the transmitter to be clocked 4 times faster than the system clock and the receiver 2 times faster. These are theoretical values without consideration for clock skew and jitter. Note also that the receiver clocks data at both negative and positive edges. Thus, the bitrate is twice as high as the clock-rate.

The synchronization limits the Tx and Rx clocks to be at most 4 and 2 times faster than the system clock. But it might not be possible to achieve such high clock rates for the Tx and Rx clocks for all technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses has a completely asynchronous reset. To make sure that nothing bad happens the is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

46.8.4 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the core will not work unless the *ft* VHDL generic is 0.

46.8.5 Synthesis

Since the receiver and transmitter may run on very high frequency clocks their clock signals have been coupled through a clock buffer with a technology wrapper. This clock buffer will utilize a low skew net available in the selected technology for the clock.

The clock buffer will also enable most synthesis tools to recognize the clocks and it is thus easier to find them and place constraints on them. The fact there are three clock domains in the GRSPW of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths.

In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

The type of clock buffer is selectable with a VHDL generic and the value zero provides a normal feed through which lets the synthesis tool infer the type of net used.

46.8.6 Technology mapping

The core has three generics for technology mapping: *tech*, *techfifo* and *memtech*. *Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

46.8.7 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/ or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

Receiver ahb FIFO

The receiver AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 430 shows the syncram organization for the allowed configurations.

*Table 430.*syncram_2p sizes for GRSPW receiver AHB FIFO.

Fifosize	**Syncram_2p organization**
4	4x32
8	8x32
16	16x32
32	32x32

Transmitter ahb FIFO

The transmitter AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 431 shows the syncram organization for the allowed configurations.

*Table 431.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	**Syncram_2p organization**
4	4x32
8	8x32
16	16x32
32	32x32

Receiver N-Char FIFO

The receiver N-Char fifo consists of one syncram_2p block with a width of 9-bits. The depth is determined by the configured FIFO depth. Table 432 shows the syncram organization for the allowed configurations.

*Table 432.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	**Syncram_2p organization**
16	16x9
32	32x9
64	64x9

RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 433 shows the syncram organization for the allowed configurations.

*Table 433.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	**Syncram_2p organization**
2	64x8
4	128x8
8	256x8

46.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 434.*GRSPW registers

APB address offset	Register
0x0	Control
0x4	Status/Interrupt-source
0x8	Node address
0xC	Clock divisor
0x10	Destination key
0x14	Time
0x18	Timer and Disconnect
0x20	DMA channel 1 control/status
0x24	DMA channel 1 rx maximum length
0x28	DMA channel 1 transmit descriptor table address.
0x2C	DMA channel 1 receive descriptor table address.

Table 435. GRSPW control register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC	RESERVED							PS	NP			RD	RE	RESERVED				TR	TT	LI	TQ		RS	PM	TI	IE	AS	LS	LD

31	RMAP available (RA) - Set to one if the RMAP command handler is available. Only readable.
30	RX unaligned access (RX) - Set to one if unaligned writes are available for the receiver. Only readable.
29	RMAP CRC available (RC) - Set to one if RMAP CRC is enabled in the core. Only readable.
28: 27	RESERVED
26	Number of ports (PO) - The number of available SpaceWire ports minus one. Only readable.
25: 22	RESERVED
21	Port select (PS) - Selects the active port when the no port force bit is zero. ‘0’ selects the port connected to data and strobe on index 0 while ‘1’ selects index 1. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.
20	No port force (NP) - Disable port force. When disabled the port select bit cannot be used to select the active port. Instead, it is automatically selected by checking the activity on the respective receive links. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.

Table 435. GRSPW control register

19: 18	RESERVED
17	RMAP buffer disable (RD) - If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Only available if the rmap VHDL generic is set to 1. Reset value: ‘0’.
16	RMAP Enable (RE) - Enable RMAP command handler. Only available if rmap VHDL generic is set to 1. Reset value: ‘1’.
15: 12	RESERVED
11	Time Rx Enable (TR) - Enable time-code receptions. Reset value: ‘0’.
10	Time Tx Enable (TT) - Enable time-code transmissions. Reset value: ‘0’.
9	Link error IRQ (LI) - Generate interrupt when a link error occurs. Not reset.
8	Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received. Not reset.
7	RESERVED
6	Reset (RS) - Make complete reset of the SpaceWire node. Self clearing. Reset value: ‘0’.
5	Promiscuous Mode (PM) - Enable Promiscuous mode. Reset value: ‘0’.
4	Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal. Reset value: ‘0’.
3	Interrupt Enable (IE) - If set, an interrupt is generated when one or both of bit 8 to 9 is set and its corresponding event occurs. Reset value: ‘0’.
2	Autostart (AS) - Automatically start the link when a NULL has been received. Not reset.
1	Link Start (LS) - Start the link, i.e. allow a transition from ready to started state. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.
0	Link Disable (LD) - Disable the SpaceWire codec. Reset value: ‘0’.

Table 436. GRSPW status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								LS			RESERVED											AP	EE	IA	WE		PE	DE	ER	CE	TO

31: 24	RESERVED
23: 21	Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. Reset value: 0.
20: 10	RESERVED
9	Active port (AP) - Shows the currently active port. ‘0’ = Port 0 and ‘1’ = Port 1 where the port numbers refer to the index number of the data and strobe signals. Only available if the ports generic is set to 2.
8	Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-rmap packet and after the second byte for a RMAP packet. Cleared when written with a one. Reset value: ‘0’.
7	Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the nodeaddr register. Cleared when written with a one. Reset value: ‘0’.
6	Write synchronization Error (WE) - A synchronization problem has occurred when receiving N-Chars. Cleared when written with a one. Reset value: ‘0’.
5	RESERVED
4	Parity Error (PE) - A parity error has occurred. Cleared when written with a one. Reset value: ‘0’.
3	Disconnect Error (DE) - A disconnection error has occurred. Cleared when written with a one. Reset value: ‘0’.
2	Escape Error (ER) - An escape error has occurred. Cleared when written with a one. Reset value: ‘0’.
1	Credit Error (CE) - A credit has occurred. Cleared when written with a one. Reset value: ‘0’.
0	Tick Out (TO) - A new time count value was received and is stored in the time counter field. Cleared when written with a one. Reset value: ‘0’.

Table 437. GRSPW node address register

31 — 8	7 — 0
RESERVED	NODEADDR

31: 8	RESERVED
7: 0	Node address (NODEADDR) - 8-bit node address used for node identification on the SpaceWire network. Reset value: 254.

Table 438. GRSPW clock divisor register

31 — 16	15 — 8	7 — 0
RESERVED	CLKDIVSTART	CLKDIVRUN

31: 16	RESERVED
15: 8	Clock divisor startup (CLKDIVSTART) - 8-bit Clock divisor value used for the clock-divider during startup (link-interface is in other states than run). The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.
7: 0	Clock divisor run (CLKDIVRUN) - 8-bit Clock divisor value used for the clock-divider when the link-interface is in the run-state. The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.

Table 439. GRSPW destination key

31 — 8	7 — 0
RESERVED	DESTKEY

31: 8	RESERVED
7: 0	Destination key (DESTKEY) - RMAP destination key. Only available if the rmap VHDL generic is set to 1. Reset value: 0.

Table 440. GRSPW time register

31 — 8	7 — 6	5 — 0
RESERVED	TCTRL	TIMECNT

31: 8	RESERVED
7: 6	Time control flags (TCTRL) - The current value of the time control flags. Sent with time-code resulting from a tick-in. Received control flags are also stored in this register. Reset value: '0'.
5: 0	Time counter (TIMECNT) - The current value of the system time counter. It is incremented for each tick-in and the incremented value is transmitted. The register can also be written directly but the written value will not be transmitted. Received time-counter values are also stored in this register. Reset value: '0'.

Table 441. GRSPW timer and disconnect register.

31 — 22	21 — 12	11 — 0
RESERVED	DISCONNECT	TIMER64

31: 22	RESERVED

Table 441. GRSPW timer and disconnect register.

21: 12	Disconnect (DISCONNECT) - Used to generate the 850 ns disconnect time period. The disconnect period is the number is the number of clock cycles in the disconnect register + 3. So to get a 850 ns period, the smallest number of clock cycles that is greater than or equal to 850 ns should be calculated and this values - 3 should be stored in the register. Reset value is set with VHDL generics or with input signals depending on the value of the usegen VHDL generic.
11: 0	6.4 us timer (TIMER64) - Used to generate the 6.4 and 12.8 us time periods. Should be set to the smallest number of clock cycles that is greater than or equal to 6.4 us. Reset value is set with VHDL generics or with input signals depending on the value of the usegen VHDL generic.

Table 442. GRSPW dma control register

31 17	16	15 13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	LE	RESERVED	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE

31: 17	RESERVED
16	Link error disable (LE) - Disable transmitter when a link error occurs. No more packets will be transmitted until the transmitter is enabled again. Reset value: ‘0’.
15: 13	RESERVED
12	No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPW will wait for a descriptor to be activated.
11	Rx descriptors available (RD) - Set to one, to indicate to the GRSPW that there are enabled descriptors in the descriptor table. Cleared by the GRSPW when it encounters a disabled descriptor: Reset value: ‘0’.
10	RX active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active otherwise it is ‘0’. Only readable.
9	Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing. Reset value: ‘0’.
8	RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.
7	TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.
6	Packet received (PR) - This bit is set each time a packet has been received. never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.
5	Packet sent (PS) - This bit is set each time a packet has been sent. Never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.
4	AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus. Not reset.
3	Receive interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP. Not reset.
2	Transmit interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not. Not reset.
1	Receiver enable (RE) - Set to one when packets are allowed to be received to this channel. Reset value: ‘0’.
0	Transmitter enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the SW-node to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the SW-node encounters a descriptor which is disabled. Reset value: ‘0’.

Table 443. GRSPW RX maximum length register.

31 25	24 0
RESERVED	RXMAXLEN

Table 443. GRSPW RX maximum length register.

31: 25	RESERVED
24: 0	RX maximum length (RXMAXLEN) - Receiver packet maximum length in bytes. Only bits 24 - 2 are writable. Bits 1 - 0 are always 0. Not reset.

Table 444. GRSPW transmitter descriptor table address register.

31 10	9 4	3 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 4	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. Reset value: 0.
3: 0	RESERVED

Table 445. GRSPW receiver descriptor table address register.

31 10	9 3	2 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 3	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0.
2: 0	RESERVED

46.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x1F. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

46.11 Configuration options

Table 446 shows the configuration options of the core (VHDL generics).

*Table 446.*Configuration options

Generic	Function	Allowed range	Default
tech	Technology for clock buffers	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW.	0 - NAHBIRQ-1	0
sysfreq	Frequency of clock input “clk” in kHz.	-	10000
usegen	Use values calculated from sysfreq generic as reset values for 6.4 us timer and disconnect timer.	0 - 1	1
nsync	Number of synchronization registers.	1 - 2	1
rmap	Include hardware RMAP command handler. RMAP CRC logic will also be added.	0 - 1	0
rmapcrc	Enable RMAP CRC logic.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 32	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxclkbuftype	Select clock buffer type for receiver clock. 0 does not select a buffer, instead i connects the input directly to the output (synthesis tools may still infer a buffer). 1 selects hardwired clock while 2 selects routed clock.	0 - 2	0
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors	0 - 2	0
scantest	Enable support for scan test	0 - 1	0
techfifo	Implement FIFO with RAM cells (1) or flip-flops (0)	0 - 1	1
netlist	Use netlist rather then RTL code	0 - 1	0
ports	Sets the number of ports	1 - 2	1
memtech	Technology for RAM blocks	0 - NTECH	inferred

46.12 Signal descriptions

Table 447 shows the interface signals of the core (VHDL ports).

Table 447. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TXCLK	N/A	Input	Transmitter default run-state clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SWNI	D	Input	Data input	-
	S	Input	Strobe input	-
	TICKIN	Input	Time counter tick input	High
	CLKDIV10	Input	Clock divisor value used during initialization and as reset value for the clock divisor register	-
	RMAPEN	Input	Reset value for the rmapen control register bit	-
	DCRSTVAL	Input	Reset value for disconnect timer. Used if usegen VHDL generic is set to 0.	-
	TIMERRSTVAL	Input	Reset value for 6.4 us timer. Used if usegen VHDL generic is set to 0.	-
SWNO	D	Output	Data output	-
	S	Output	Strobe output	-
	TICKOUT	Output	Time counter tick output	High
* see GRLIB IP Library User's Manual				

46.13 Library dependencies

Table 448 shows libraries used when instantiating the core (VHDL libraries).

*Table 448.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

46.14 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The GRSPW in the example is configured with non-ft memories of size 4, 64 and 8 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively. The system frequency (clk) is 40 MHz and the transmitter frequency (txclk) is 20 MHz.

The memory technology is inferred which means that the synthesis tool will select the appropriate components. The rx clk buffer uses a hardwired clock.

The hardware RMAP command handler is enabled which also automatically enables rxunaligned and rmapcrc. The Finally, the DMA channel interrupt line is 2 and the number of synchronization registers is 1.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- spacewire signals
    di :  in  std_logic_vector(1 downto 0);
    si :  in  std_logic_vector(1 downto 0);
    do :  out std_logic_vector(1 downto 0);
    so :  out std_logic_vector(1 downto 0)
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni : grspw_in_type;
  signal swno : grspw_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSPW
  sw0 : grspw
  generic map (tech => inferred, hindex => 5, pindex => 7, paddr => 7, nsync => 1,
    rmap => 1, rxunaligned => 0, rmapcrc => 0, rxclkbuftype => 0, sysfreq => 40000,
    pirq => 2, fifosize1 => 4, fifosize2 => 64, rmapbufs => 8, ft => 0, ports => 2)
  port map (rstn, clk, apbi, apbo(7), ahbmi, ahbmo(5), swni, swno);

  swni.rmapen   <= ‘1’;
  swni.clkdiv10 <= “00000001”;
  swni.tickin   <= ‘0’;
  swni.d(0)     <= di(0);
  swni.s(0)     <= si(0);
  do(0)         <= swno.d(0);
  so(0)         <= swno.s(0);
  swni.d(1)     <= di(1);
  swni.s(1)     <= si(1);
  do(1)         <= swno.d(1);
  so(1)         <= swno.s(1);
end;
```

46.15 API

A simple Application Programming Interface (API) is provided together with the GRSPW. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW. The different functions are described in this section.

46.15.1 GRSPW Basic API

The basic GRSPW API is based on a struct spwvars which stores all the information for a single GRSPW core. The information includes its address on the AMBA bus as well as SpaceWire parameters such as node address and clock divisor. A pointer to this struct is used as a input parameter to all the functions. If several cores are used, a separate struct for each core is created and used when the specific core is accessed.

*Table 449.*The spwvars struct

Field	Description	Allowed range
regs	Pointer to the GRSPW	-
nospill	The nospill value used for the core.	0 - 1
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
timetxen	The timetxen value used for this core.	0 - 1
timerxen	The timerxen value used for this core.	0 - 1
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int nospill, int timetxen, int
timerxen, int rxmaxlen, int spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

*Table 450.*Return values for spw_setparam

Value	**Description**
0	The function completed successfully
1	One or more of the parameters had an illegal value

*Table 451.*Parameters for spw_setparam

Parameter	**Description**	**Allowed range**
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
nospill	Sets the nospill value of the struct spw passed to the function.	0 - 1
timetxen	Sets the timetxen value of the struct spw passed to the function.	0 - 1
timerxen	Sets the timerxen value of the struct spw passed to the function.	0 - 1
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	0 - 2^{25}-1
spwadr	Sets the address to the GRSPW core which will be associated with the struct passed to the function.	0 - 2^{32}-1

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, receiver maximum length, transmitter descriptor table address, receiver descriptor table address, ctrl and dmactrl. All bits are set to the values found in the spwvars struct. If a register bit is not present in the struct it will be set to zero. The descriptor tables are allocated to an aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the value in clkdiv.

*Table 452.*Return values for spw_init

Value	**Description**
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

*Table 453.*Parameters for spw_init

Parameter	**Description**	**Allowed range**
spw	The spwvars struct associated with the GRSPW core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

*Table 454.*Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 455.*Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

*Table 456.*Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 457.*Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW core (the link disable bit is set to ‘1’).

*Table 458.*Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW core (the link disable bit is set to ‘0’).

*Table 459.*Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW core (the link start bit is set to ‘1’).

*Table 460.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW core (the link start bit is set to ‘0’).

*Table 461.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

*Table 462.*Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

*Table 463.*Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

*Table 464.*Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 465.*Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

*Table 466.*Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 467.*Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

*Table 468.*Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

*Table 469.*Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

*Table 470.*Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

*Table 471.*Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 472.*Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

*Table 473.*Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

*Table 474.*The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 475.*Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

*Table 476.*Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW and transmits the value.

*Table 477.*Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

*Table 478.*Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

*Table 479.*Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

*Table 480.*Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

*Table 481.*Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW.

*Table 482.*Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW.

*Table 483.*Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW

*Table 484.*Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

*Table 485.*Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

*Table 486.*Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set.	-

46.15.2 GRSPW RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds a RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

*Table 487.*Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

*Table 488.*Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to a rmap_pkt struct which contains the data from which the header should be built	
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

*Table 489.*rmap_pkt struct fields

Field	Description	Allowed Range
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-

47 GRSPW2 - SpaceWire codec with AHB host Interface and RMAP target

47.1 Overview

The SpaceWire core provides an interface between the AHB bus and a SpaceWire network. It implements the SpaceWire standard (ECSS-E-50-12A) with the protocol identification extension (ECSS-E-50-11). The optional Remote Memory Access Protocol (RMAP) target implements the ECSS standard (ECSS-E-50-11).

The SpaceWire interface is configured through a set of registers accessed through an APB interface. Data is transferred through DMA channels using an AHB master interface. The number of DMA channels is configurable from one to four.

The core can also be configured with two SpaceWire ports with manual or automatic switching between them.

There can be up to four clock domains: one for the AHB interface (system clock), one for the transmitter and one or two for the receiver depending on the number of configured ports. The receiver clock can be four times as fast and the transmitter clock eight times as fast as the system clock.

The core only supports byte addressed 32-bit big-endian host systems.

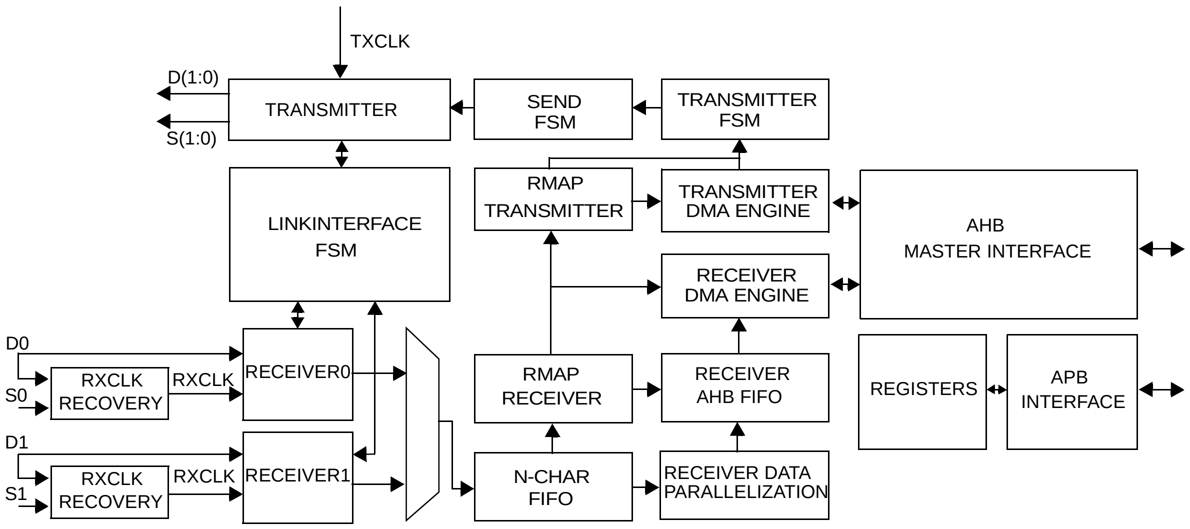


Figure 148. Block diagram

47.2 Operation

47.2.1 Overview

The main sub-blocks of the core are the link interface, the RMAP target and the AMBA interface. A block diagram of the internal structure can be found in figure 148.

The link interface consists of the receiver, transmitter and the link interface FSM. They handle communication on the SpaceWire network. The AMBA interface consists of the DMA engines, the AHB master interface and the APB interface. The link interface provides FIFO interfaces to the DMA engines. These FIFOs are used to transfer N-Chars between the AMBA and SpaceWire domains during reception and transmission.

The RMAP target is an optional part of the core which can be enabled with a VHDL generic. The RMAP handler handles incoming packets which are determined to be RMAP commands instead of the receiver DMA engine. The RMAP command is decoded and if it is valid, the operation is performed on the AHB bus. If a reply was requested it is automatically transmitted back to the source by the RMAP transmitter.

The core is controlled by writing to a set of user registers through the APB interface and three signals: tick-in, rmapen and clkdiv10.

The link interface, DMA engines, RMAP handler and AMBA interface are described in section 47.3, 47.4, 47.6 and 47.7 respectively.

47.2.2 Protocol support

The core only accepts packets with a valid destination address in the first received byte. Packets with address mismatch will be silently discarded (except in promiscuous mode which is covered in section 47.4.10).

The second byte is always interpreted as a protocol ID. The only protocol handled separately in hardware is the RMAP protocol (ID=0x1) while other packets are stored to a DMA channel. If the RMAP target is present and enabled all RMAP commands will be processed, executed and replied automatically in hardware. Otherwise RMAP commands are stored to a DMA channel in the same way as other packets. RMAP replies are always stored to a DMA channel. More information on the RMAP protocol support is found in section 47.6. When the RMAP target is not present or disabled, there is no need to include a protocol ID in the packets and the data can start immediately after the address.

All packets arriving with the extended protocol ID (0x00) are stored to a DMA channel. This means that the hardware RMAP target will not work if the incoming RMAP packets use the extended protocol ID. Note also that packets with the reserved extended protocol identifier (ID = 0x000000) are not ignored by the core. It is up to the client receiving the packets to ignore them.

When transmitting packets, the address and protocol-ID fields must be included in the buffers from where data is fetched. They are *not* automatically added by the core.

Figure 149 shows a packet with a normal protocol identifier. The core also allows reception and transmission with extended protocol identifiers but without support for RMAP CRC calculations and the RMAP target.

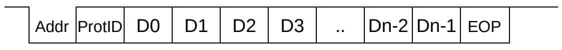


Figure 149. The SpaceWire packet with protocol ID that is expected by the GRSPW.

47.3 Link interface

The link interface handles the communication on the SpaceWire network and consists of a transmitter, receiver, a FSM and FIFO interfaces. An overview of the architecture is found in figure 148.

47.3.1 Link interface FSM

The FSM controls the link interface (a more detailed description is found in the SpaceWire standard). The low-level protocol handling (the signal and character level of the SpaceWire standard) is handled by the transmitter and receiver while the FSM handles the exchange level.

The link interface FSM is controlled through the control register. The link can be disabled through the link disable bit, which depending on the current state, either prevents the link interface from reaching the started state or forces it to the error-reset state. When the link is not disabled, the link interface FSM is allowed to enter the started state when either the link start bit is set or when a NULL character has been received and the autostart bit is set.

The current state of the link interface determines which type of characters are allowed to be transmitted which together with the requests made from the host interfaces determine what character will be sent.

Time-codes are sent when the FSM is in the run-state and a request is made through the time-interface (described in section 47.3.5).

When the link interface is in the connecting- or run-state it is allowed to send FCTs. FCTs are sent automatically by the link interface when possible. This is done based on the maximum value of 56 for the outstanding credit counter and the currently free space in the receiver N-Char FIFO. FCTs are sent as long as the outstanding counter is less than or equal to 48 and there are at least 8 more empty FIFO entries than the counter value.

N-Chars are sent in the run-state when they are available from the transmitter FIFO and there are credits available. NULLs are sent when no other character transmission is requested or the FSM is in a state where no other transmissions are allowed.

The credit counter (incoming credits) is automatically increased when FCTs are received and decreased when N-Chars are transmitted. Received N-Chars are stored to the receiver N-Char FIFO for further handling by the DMA interface. Received Time-codes are handled by the time-interface.

47.3.2 Transmitter

The state of the FSM, credit counters, requests from the time-interface and requests from the DMA-interface are used to decide the next character to be transmitted. The type of character and the character itself (for N-Chars and Time-codes) to be transmitted are presented to the low-level transmitter which is located in a separate clock-domain.

This is done because one usually wants to run the SpaceWire link on a different frequency than the host system clock. The core has a separate clock input which is used to generate the transmitter clock. More information on transmitter clock generation is found in section 47.8.1. Since the transmitter often runs on high frequency clocks (> 100 MHz) as much logic as possible has been placed in the system clock domain to minimize power consumption and timing issues.

The transmitter logic in the host clock domain decides what character to send next and sets the proper control signal and presents any needed character to the low-level transmitter as shown in figure 150. The transmitter sends the requested characters and generates parity and control bits as needed. If no requests are made from the host domain, NULLs are sent as long as the transmitter is enabled. Most of the signal and character levels of the SpaceWire standard is handled in the transmitter. External LVDS drivers are needed for the data and strobe signals.

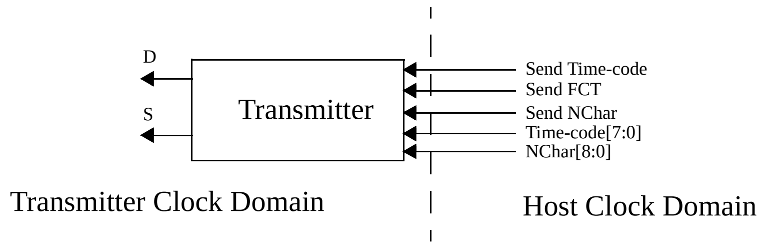


Figure 150. Schematic of the link interface transmitter.

A transmission FSM reads N-Chars for transmission from the transmitter FIFO. It is given packet lengths from the DMA interface and appends EOPs/EEPs and RMAP CRC values if requested. When it is finished with a packet the DMA interface is notified and a new packet length value is given.

47.3.3 Receiver

The receiver detects connections from other nodes and receives characters as a bit stream on the data and strobe signals. It is also located in a separate clock domain which runs on a clock generated from the received data and strobe signals. More information on the clock-generation can be found in section 47.8.1.

The receiver is activated as soon as the link interface leaves the error reset state. Then after a NULL is received it can start receiving any characters. It detects parity, escape and credit errors which causes the link interface to enter the error reset state. Disconnections are handled in the link interface part in the tx clock domain because no receiver clock is available when disconnected.

Received Characters are flagged to the host domain and the data is presented in parallel form. The interface to the host domain is shown in figure 151. L-Chars are the handled automatically by the host domain link interface part while all N-Chars are stored in the receiver FIFO for further handling. If two or more consecutive EOPs/EEPs are received all but the first are discarded.

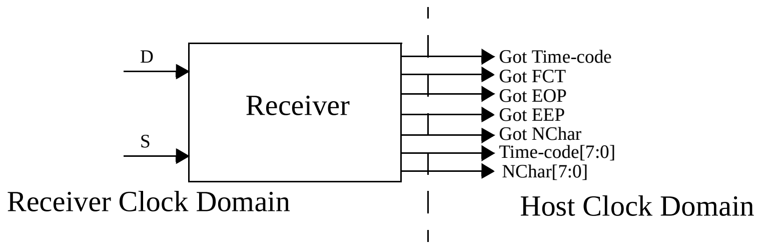


Figure 151. Schematic of the link interface receiver.

47.3.4 Dual port support

The core can be configured to include an additional SpaceWire port. With dual ports the transmitter drives an additional pair of data/strobe output signals and one extra receiver is added to handle a second pair of data/strobe input signals.

One of the ports is set as active (how the active port is selected is explained below) and the transmitter drives the data/strobe signals of the active port with the actual output values as explained in section 47.3.2. The inactive port is driven with zero on both data and strobe.

Both receivers will always be active but only the active port's interface signals (see figure 151) will be propagated to the link interface FSM. Each time the active port is changed, the link will be reset so that the new link is started in a controlled manner.

When the noportforce register is zero the portsel register bit selects the active link and when set to one it is determined by the current link activity. In the latter mode the port is changed when no activity is seen on the currently active link while there is activity on the deselected receive port. Activity is defined as a detected null. This definition is selected so that glitches (e.g. port unconnected) do not cause unwanted port switches.

47.3.5 Time interface

The time interface is used for sending Time-codes over the SpaceWire network and consists of a time-counter register, time-ctrl register, tick-in signal, tick-out signal, tick-in register field and a tick-out register field. There are also two control register bits which enable the time receiver and transmitter respectively.

Each Time-code sent from the grspw is a concatenation of the time-ctrl and the time-counter register. There is a timetxen bit which is used to enable Time-code transmissions. It is not possible to send time-codes if this bit is zero.

Received Time-codes are stored to the same time-ctrl and time-counter registers which are used for transmission. The timerxen bit in the control register is used for enabling time-code reception. No time-codes will be received if this bit is zero.

The two enable bits are used for ensuring that a node will not (accidentally) both transmit and receive time-codes which violates the SpaceWire standard. It also ensures that a the master sending time-codes on a network will not have its time-counter overwritten if another (faulty) node starts sending time-codes.

The time-counter register is set to 0 after reset and is incremented each time the tick-in signal is asserted for one clock-period and the timetxen bit is set. This also causes the link interface to send the new value on the network. Tick-in can be generated either by writing a one to the register field or by asserting the tick-in signal. A Tick-in should not be generated too often since if the time-code after the previous Tick-in has not been sent the register will not be incremented and no new value will be sent. The tick-in field is automatically cleared when the value has been sent and thus no new ticks should be generated until this field is zero. If the tick-in signal is used there should be at least 4 system-clock and 25 transmit-clock cycles between each assertion.

A tick-out is generated each time a valid time-code is received and the timerxen bit is set. When the tick-out is generated the tick-out signal will be asserted one clock-cycle and the tick-out register field is asserted until it is cleared by writing a one to it.

The current time counter value can be read from the time register. It is updated each time a Time-code is received and the timerxen bit is set. The same register is used for transmissions and can also be written directly from the APB interface.

The control bits of the Time-code are stored to the time-ctrl register when a Time-code is received whose time-count is one more than the nodes current time-counter register. The time-ctrl register can be read through the APB interface. The same register is used during time-code transmissions.

It is possible to have both the time-transmission and reception functions enabled at the same time.

47.4 Receiver DMA channels

The receiver DMA engine handles reception of data from the SpaceWire network to different DMA channels.

47.4.1 Address comparison and channel selection

Packets are received to different channels based on the address and whether a channel is enabled or not. When the receiver N-Char FIFO contains one or more characters, N-Chars are read by the receiver DMA engine. The first character is interpreted as the logical address and is compared with the addresses of each channel starting from 0. The packet will be stored to the first channel with an matching address. The complete packet including address and protocol ID but excluding EOP/EEP is stored to the memory address pointed to by the descriptors (explained later in this section) of the channel.

Each SpaceWire address register has a corresponding mask register. Only bits at an index containing a zero in the corresponding mask register are compared. This way a DMA channel can accept a range of addresses. There is a default address register which is used for address checking in all implemented DMA channels that do not have separate addressing enabled and for RMAP commands in the RMAP target. With separate addressing enabled the DMA channels' own address/mask register pair is used instead.

If an RMAP command is received it is only handled by the target if the default address register (including mask) matches the received address. Otherwise the packet will be stored to a DMA channel if one or more of them has a matching address. If the address does not match neither the default address nor one of the DMA channels' separate register, the packet is still handled by the RMAP target if enabled since it has to return the invalid address error code. The packet is only discarded (up to and including the next EOP/EEP) if an address match cannot be found and the RMAP target is disabled.

Packets, other than RMAP commands, that do not match neither the default address register nor the DMA channels' address register will be discarded. Figure 152 shows a flowchart of packet reception.

At least 2 non EOP/EEP N-Chars needs to be received for a packet to be stored to the DMA channel. If it is an RMAP packet with hardware RMAP enabled 3 N-Chars are needed since the command byte determines where the packet is processed. Packets smaller than these sizes are discarded.

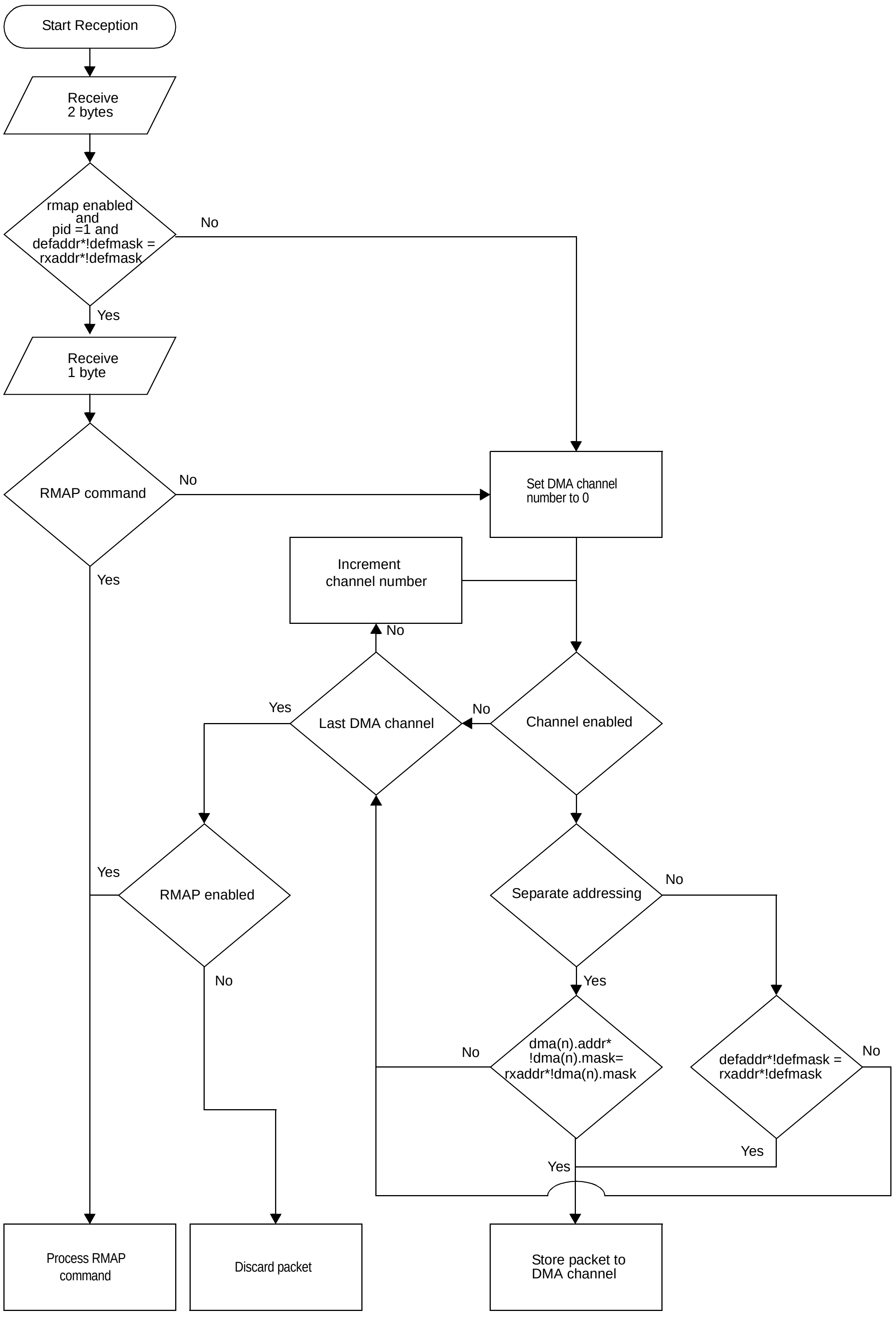


Figure 152. Flow chart of packet reception.

47.4.2 Basic functionality of a channel

Reception is based on descriptors located in a consecutive area in memory that hold pointers to buffers where packets should be stored. When a packet arrives at the core the channel which should receive it is first determined as described in the previous section. A descriptor is then read from the channels' descriptor area and the packet is stored to the memory area pointed to by the descriptor. Lastly, status is stored to the same descriptor and increments the descriptor pointer to the next one. The following sections will describe DMA channel reception in more detail.

47.4.3 Setting up the core for reception

A few registers need to be initialized before reception to a channel can take place. First the link interface need to be put in the run state before any data can be sent. The DMA channel has a maximum length register which sets the maximum packet size in bytes that can be received to this channel. Larger packets are truncated and the excessive part is spilled. If this happens an indication will be given in the status field of the descriptor. The minimum value for the receiver maximum length field is 4 and the value can only be incremented in steps of four bytes up to the maximum value 33554428. If the maximum length is set to zero the receiver will *not* function correctly.

Either the default address register or the channel specific address register (the accompanying mask register must also be set) needs to be set to hold the address used by the channel. A control bit in the DMA channel control register determines whether the channel should use default address and mask registers for address comparison or the channel's own registers. Using the default register the same address range is accepted as for other channels with default addressing and the RMAP target while the separate address provides the channel its own range. If all channels use the default registers they will accept the same address range and the enabled channel with the lowest number will receive the packet.

Finally, the descriptor table and control register must be initialized. This will be described in the two following sections.

47.4.4 Setting up the descriptor table address

The core reads descriptors from an area in memory pointed to by the receiver descriptor table address register. The register consists of a base address and a descriptor selector. The base address points to the beginning of the area and must start on a 1024 bytes aligned address. It is also limited to be 1024 bytes in size which means the maximum number of descriptors is 128 since the descriptor size is 8 bytes.

The descriptor selector points to individual descriptors and is increased by 1 when a descriptor has been used. When the selector reaches the upper limit of the area it wraps to the beginning automatically. It can also be set to wrap at a specific descriptor before the upper limit by setting the wrap bit in the descriptor. The idea is that the selector should be initialized to 0 (start of the descriptor area) but it can also be written with another 8 bytes aligned value to start somewhere in the middle of the area. It will still wrap to the beginning of the area.

If one wants to use a new descriptor table the receiver enable bit has to be cleared first. When the rxactive bit for the channel is cleared it is safe to update the descriptor table register. When this is finished and descriptors are enabled the receiver enable bit can be set again.

47.4.5 Enabling descriptors

As mentioned earlier one or more descriptors must be enabled before reception can take place. Each descriptor is 8 byte in size and the layout can be found in the tables below. The descriptors should be written to the memory area pointed to by the receiver descriptor table address register. When new descriptors are added they must always be placed after the previous one written to the area. Otherwise they will not be noticed.

A descriptor is enabled by setting the address pointer to point at a location where data can be stored and then setting the enable bit. The WR bit can be set to cause the selector to be set to zero when reception has finished to this descriptor. IE should be set if an interrupt is wanted when the reception has finished. The DMA control register interrupt enable bit must also be set for an interrupt to be generated.

The descriptor packet address should be word aligned. All accesses on the bus are word accesses so complete words will always be overwritten regardless of whether all 32-bit contain received data. Also if the packet does not end on a word boundary the complete word containing the last data byte will be overwritten. If the rxunaligned or rmap VHDL generic is set to 1 this restriction is removed and any number of bytes can be received to any packet address without excessive bytes being overwritten.

Table 490. GRSPW receive descriptor word 0 (address offset 0x0)

31	30	29	28	27	26	25	24 - 0
TR	DC	HC	EP	IE	WR	EN	PACKETLENGTH

31	Truncated (TR) - Packet was truncated due to maximum length violation.
30	Data CRC (DC) - 1 if a CRC error was detected for the data and 0 otherwise.
29	Header CRC (HC) - 1 if a CRC error was detected for the header and 0 otherwise.
28	EEP termination (EP) - This packet ended with an Error End of Packet character.
27	Interrupt enable (IE) - If set, an interrupt will be generated when a packet has been received if the receive interrupt enable bit in the DMA channel control register is set.
26	Wrap (WR) - If set, the next descriptor used by the GRSPW will be the first one in the descriptor table (at the base address). Otherwise the descriptor pointer will be increased with 0x8 to use the descriptor at the next higher memory location. The descriptor table is limited to 1 kbytes in size and the pointer will be automatically wrap back to the base address when it reaches the 1 kbytes boundary.
25	Enable (EN) - Set to one to activate this descriptor. This means that the descriptor contains valid control values and the memory area pointed to by the packet address field can be used to store a packet.
24: 0	Packet length (PACKETLENGTH) - The number of bytes received to this buffer. Only valid after EN has been set to 0 by the GRSPW.

Table 491. GRSPW receive descriptor word 1 (address offset 0x4)

31 - 0
PACKETADDRESS

31: 0	Packet address (PACKETADDRESS) - The address pointing at the buffer which will be used to store the received packet. If the rxunaligned and rmap VHDL generics are both set to zero only bit 31 to 2 are used.

47.4.6 Setting up the DMA control register

The final step to receive packets is to set the control register in the following steps: The receiver must be enabled by setting the rxen bit in the DMA control register (see section 47.9). This can be done anytime and before this bit is set nothing will happen. The rxdescav bit in the DMA control register is then set to indicate that there are new active descriptors. This must always be done after the descriptors have been enabled or the core might not notice the new descriptors. More descriptors can be activated when reception has already started by enabling the descriptors and writing the rxdescav bit. When these bits are set reception will start immediately when data is arriving.

47.4.7 The effect to the control bits during reception

When the receiver is disabled all packets going to the DMA-channel are discarded if the packet’s address does not fall into the range of another DMA channel. If the receiver is enabled and the address falls into the accepted address range, the next state is entered where the rxdescav bit is checked. This bit indicates whether there are active descriptors or not and should be set by the external application using the DMA channel each time descriptors are enabled as mentioned above. If the rxdescav bit is ‘0’ and the nospill bit is ‘0’ the packets will be discarded. If nospill is one the grspw waits until rxdescav is set and the characters are kept in the N-Char fifo during this time. If the fifo becomes full further N-char transmissions are inhibited by stopping the transmission of FCTs.

When rxdescav is set the next descriptor is read and if enabled the packet is received to the buffer. If the read descriptor is not enabled, rxdescav is set to ‘0’ and the packet is spilled depending on the value of nospill.

The receiver can be disabled at any time and will stop packets from being received to this channel. If a packet is currently received when the receiver is disabled the reception will still be finished. The rxdescav bit can also be cleared at any time. It will not affect any ongoing receptions but no more descriptors will be read until it is set again. Rxdescav is also cleared by the core when it reads a disabled descriptor.

47.4.8 Status bits

When the reception of a packet is finished the enable bit in the current descriptor is set to zero. When enable is zero, the status bits are also valid and the number of received bytes is indicated in the length field. The DMA control register contains a status bit which is set each time a packet has been received. The core can also be made to generate an interrupt for this event.

RMAP CRC logic is included in the implementation if the rmapcrc or rmap VHDL generic set to 1. The RMAP CRC calculation is always active for all received packets and all bytes except the EOP/EEP are included. The packet is always assumed to be a RMAP packet and the length of the header is determined by checking byte 3 which should be the command field. The calculated CRC value is then checked when the header has been received (according to the calculated number of bytes) and if it is non-zero the HC bit is set indicating a header CRC error.

The CRC value is not set to zero after the header has been received, instead the calculation continues in the same way until the complete packet has been received. Then if the CRC value is non-zero the DC bit is set indicating a data CRC error. This means that the core can indicate a data CRC error even if the data field was correct when the header CRC was incorrect. However, the data should not be used when the header is corrupt and therefore the DC bit is unimportant in this case. When the header is not corrupted the CRC value will always be zero when the calculation continues with the data field and the behaviour will be as if the CRC calculation was restarted

If the received packet is not of RMAP type the header CRC error indication bit cannot be used. It is still possible to use the DC bit if the complete packet is covered by a CRC calculated using the RMAP CRC definition. This is because the core does not restart the calculation after the header has been received but instead calculates a complete CRC over the packet. Thus any packet format with one CRC at the end of the packet calculated according to RMAP standard can be checked using the DC bit.

If the packet is neither of RMAP type nor of the type above with RMAP CRC at the end, then both the HC and DC bits should be ignored.

47.4.9 Error handling

If a packet reception needs to be aborted because of congestion on the network, the suggested solution is to set link disable to ‘1’. Unfortunately, this will also cause the packet currently being transmitted to be truncated but this is the only safe solution since packet reception is a passive operation depending on the transmitter at the other end. A channel reset bit could be provided but is not a satisfactory solution since the untransmitted characters would still be in the transmitter node. The next character (somewhere in the middle of the packet) would be interpreted as the node address which would probably cause the packet to be discarded but not with 100% certainty. Usually this action is performed when a reception has stuck because of the transmitter not providing more data. The channel reset would not resolve this congestion.

If an AHB error occurs during reception the current packet is spilled up to and including the next EEP/EOP and then the currently active channel is disabled and the receiver enters the idle state. A bit in the channels control/status register is set to indicate this condition.

47.4.10 Promiscuous mode

The core supports a promiscuous mode where all the data received is stored to the first DMA channel enabled regardless of the node address and possible early EOPs/EEPs. This means that all non-eop/eep N-Chars received will be stored to the DMA channel. The rxmaxlength register is still checked and packets exceeding this size will be truncated.

RMAP commands will still be handled by it when promiscuous mode is enabled if the rmapen bit is set. If it is cleared, RMAP commands will also be stored to a DMA channel.

47.5 Transmitter DMA channels

The transmitter DMA engine handles transmission of data from the DMA channels to the SpaceWire network. Each receive channel has a corresponding transmit channel which means there can be up to 4 transmit channels. It is however only necessary to use a separate transmit channel for each receive channel if there are also separate entities controlling the transmissions. The use of a single channel with multiple controlling entities would cause them to corrupt each other's transmissions. A single channel is more efficient and should be used when possible.

Multiple transmit channels with pending transmissions are arbitrated in a round-robin fashion.

47.5.1 Basic functionality of a channel

A transmit DMA channel reads data from the AHB bus and stores them in the transmitter FIFO for transmission on the SpaceWire network. Transmission is based on the same type of descriptors as for the receiver and the descriptor table has the same alignment and size restrictions. When there are new descriptors enabled the core reads them and transfer the amount data indicated.

47.5.2 Setting up the core for transmission

Four steps need to be performed before transmissions can be done with the core. First the link interface must be enabled and started by writing the appropriate value to the ctrl register. Then the address to the descriptor table needs to be written to the transmitter descriptor table address register and one or more descriptors must also be enabled in the table. Finally, the txen bit in the DMA control register is written with a one which triggers the transmission. These steps will be covered in more detail in the next sections.

47.5.3 Enabling descriptors

The descriptor table address register works in the same way as the receiver's corresponding register which was covered in section 47.4. The maximum size is 1024 bytes as for the receiver but since the descriptor size is 16 bytes the number of descriptors is 64.

To transmit packets one or more descriptors have to be initialized in memory which is done in the following way: The number of bytes to be transmitted and a pointer to the data has to be set. There are two different length and address fields in the transmit descriptors because there are separate pointers for header and data. If a length field is zero the corresponding part of a packet is skipped and if both are zero no packet is sent. The maximum header length is 255 bytes and the maximum data length is 16 Mbyte - 1. When the pointer and length fields have been set the enable bit should be set to enable the descriptor. This must always be done last. The other control bits must also be set before enabling the descriptor.

The transmit descriptors are 16 bytes in size so the maximum number in a single table is 64. The different fields of the descriptor together with the memory offsets are shown in the tables below.

The HC bit should be set if RMAP CRC should be calculated and inserted for the header field and correspondingly the DC bit should be set for the data field. This field is only used by the GRSPW when the CRC logic is available (*rmap* or *rmapcrc* VHDL generic set to 1). The header CRC will be calculated from the data fetched from the header pointer and the data CRC is generated from data fetched from the data pointer. The CRCs are appended after the corresponding fields. The NON-CRC bytes field is set to the number of bytes in the beginning of the header field that should not be included in the CRC calculation.

The CRCs are sent even if the corresponding length is zero, but when both lengths are zero no packet is sent not even an EOP.

47.5.4 Starting transmissions

When the descriptors have been initialized, the transmit enable bit in the DMA control register has to be set to tell the core to start transmitting. New descriptors can be activated in the table on the fly (while transmission is active). Each time a set of descriptors is added the transmit enable register bit should be set. This has to be done because each time the core encounters a disabled descriptor this register bit is set to 0.

Table 492. GRSPW transmit descriptor word 0 (address offset 0x0)

31 … 18	17	16	15	14	13	12	11 … 8	7 … 0
RESERVED	DC	HC	LE	IE	WR	EN	NONCRCLEN	HEADERLEN

31: 18	RESERVED
17	Append data CRC (DC) - Append CRC calculated according to the RMAP specification after the data sent from the data pointer. The CRC covers all the bytes from this pointer. A null CRC will be sent if the length of the data field is zero.
16	Append header CRC (HC) - Append CRC calculated according to the RMAP specification after the data sent from the header pointer. The CRC covers all bytes from this pointer except a number of bytes in the beginning specified by the non-crc bytes field. The CRC will not be sent if the header length field is zero.
15	Link error (LE) - A Link error occurred during the transmission of this packet.
14	Interrupt enable (IE) - If set, an interrupt will be generated when the packet has been transmitted and the transmitter interrupt enable bit in the DMA control register is set.
13	Wrap (WR) - If set, the descriptor pointer will wrap and the next descriptor read will be the first one in the table (at the base address). Otherwise the pointer is increased with 0x10 to use the descriptor at the next higher memory location.
12	Enable (EN) - Enable transmitter descriptor. When all control fields (address, length, wrap and crc) are set, this bit should be set. While the bit is set the descriptor should not be touched since this might corrupt the transmission. The GRSPW clears this bit when the transmission has finished.
11: 8	Non-CRC bytes (NONCRCLEN)- Sets the number of bytes in the beginning of the header which should not be included in the CRC calculation. This is necessary when using path addressing since one or more bytes in the beginning of the packet might be discarded before the packet reaches its destination.
7: 0	Header length (HEADERLEN) - Header Length in bytes. If set to zero, the header is skipped.

Table 493. GRSPW transmit descriptor word 1 (address offset 0x4)

31 0
HEADERADDRESS

31: 0	Header address (HEADERADDRESS) - Address from where the packet header is fetched. Does not need to be word aligned.

Table 494. GRSPW transmit descriptor word 2 (address offset 0x8)

31 24	23 0
RESERVED	DATALEN

31: 24	RESERVED
23: 0	Data length (DATALEN) - Length of data part of packet. If set to zero, no data will be sent. If both data- and header-lengths are set to zero no packet will be sent.

Table 495. GRSPW transmit descriptor word 3(address offset 0xC)

31 0
DATAADDRESS

31: 0	Data address (DATAADDRESS) - Address from where data is read. Does not need to be word aligned.

47.5.5 The transmission process

When the txen bit is set the core starts reading descriptors immediately. The number of bytes indicated are read and transmitted. When a transmission has finished, status will be written to the first field of the descriptor and a packet sent bit is set in the DMA control register. If an interrupt was requested it will also be generated. Then a new descriptor is read and if enabled a new transmission starts, otherwise the transmit enable bit is cleared and nothing will happen until it is enabled again.

47.5.6 The descriptor table address register

The internal pointer which is used to keep the current position in the descriptor table can be read and written through the APB interface. This pointer is set to zero during reset and is incremented each time a descriptor is used. It wraps automatically when the 1024 bytes limit for the descriptor table is reached or it can be set to wrap earlier by setting a bit in the current descriptor.

The descriptor table register can be updated with a new table anytime when no transmission is active. No transmission is active if the transmit enable bit is zero and the complete table has been sent or if the table is aborted (explained below). If the table is aborted one has to wait until the transmit enable bit is zero before updating the table pointer.

47.5.7 Error handling

Abort Tx

The DMA control register contains a bit called Abort TX which if set causes the current transmission to be aborted, the packet is truncated and an EEP is inserted. This is only useful if the packet needs to be aborted because of congestion on the SpaceWire network. If the congestion is on the AHB bus this will not help (This should not be a problem since AHB slaves should have a maximum of 16 wait-states). The aborted packet will have its LE bit set in the descriptor. The transmit enable register bit is also cleared and no new transmissions will be done until the transmitter is enabled again.

AHB error

When an AHB error is encountered during transmission the currently active DMA channel is disabled and the transmitter goes to the idle mode. A bit in the DMA channel's control/status register is set to indicate this error condition and, if enabled, an interrupt will also be generated. Further error handling depends on what state the transmitter DMA engine was in when the AHB error occurred. If the descriptor was being read the packet transmission had not been started yet and no more actions need to be taken.

If the AHB error occurs during packet transmission the packet is truncated and an EEP is inserted. Lastly, if it occurs when status is written to the descriptor the packet has been successfully transmitted but the descriptor is not written and will continue to be enabled (this also means that no error bits are set in the descriptor for AHB errors).

The client using the channel has to correct the AHB error condition and enable the channel again. No more AHB transfers are done again from the same unit (receiver or transmitter) which was active during the AHB error until the error state is cleared and the unit is enabled again.

Link error

When a link error occurs during the transmission the remaining part of the packet is discarded up to and including the next EOP/EEP. When this is done status is immediately written (with the LE bit set) and the descriptor pointer is incremented. The link will be disconnected when the link error occurs but the grspw will automatically try to connect again provided that the link-start bit is asserted and the link-disabled bit is deasserted. If the LE bit in the DMA channel's control register is not set the transmitter DMA engine will wait for the link to enter run-state and start a new transmission immediately when possible if packets are pending. Otherwise the transmitter will be disabled when a link error occurs during the transmission of the current packet and no more packets will be transmitted until it is enabled again.n immediately when possible if packets are pending.

47.6 RMAP

The Remote Memory Access Protocol (RMAP) is used to implement access to resources in the node via the SpaceWire Link. Some common operations are reading and writing to memory, registers and FIFOs. The core has an optional hardware RMAP command handler which is enabled with a VHDL generic. This section describes the basics of the RMAP protocol and the command handler implementation.

47.6.1 Fundamentals of the protocol

RMAP is a protocol which is designed to provide remote access via a SpaceWire network to memory mapped resources on a SpaceWire node. It has been assigned protocol ID 0x01. It provides three operations write, read and read-modify-write. These operations are posted operations which means that a source does not wait for an acknowledge or reply. It also implies that any number of operations can be outstanding at any time and that no timeout mechanism is implemented in the protocol. Time-outs must be implemented in the user application which sends the commands. Data payloads of up to 16 Mb - 1 is supported in the protocol. A destination can be requested to send replies and to verify data before executing an operation. A complete description of the protocol is found in the RMAP standard.

47.6.2 Implementation

The core includes an handler for RMAP commands which processes all incoming packets with protocol ID = 0x01, type field (bit 7 and 6 of the 3rd byte in the packet) equal to 01b and an address falling in the range set by the default address and mask register. When such a packet is detected it is not stored to the DMA channel, instead it is passed to the RMAP receiver.

The core implements all three commands defined in the standard with some restrictions. The implementation is based on draft F of the RMAP standard (the only exception being that error code 12 is not implemented). Support is only provided for 32-bit big-endian systems. This means that the first byte received is the msb in a word. The command handler will not receive RMAP packets using the extended protocol ID which are always dumped to the DMA channel.

The RMAP receiver processes commands. If they are correct and accepted the operation is performed on the AHB bus and a reply is formatted. If an acknowledge is requested the RMAP transmitter automatically send the reply. RMAP transmissions have priority over DMA channel transmissions.

Packets with a mismatching destination logical address are never passed to the RMAP handler. There is a user accessible destination key register which is compared to destination key field in incoming packets. If there is a mismatch and a reply has been requested the error code in the reply is set to 3. Replies are sent if and only if the ack field is set to ‘1’.

When a failure occurs during a bus access the error code is set to 1 (General Error). There is predetermined order in which error-codes are set in the case of multiple errors in the core. It is shown in table 496.

*Table 496.*The order of error detection in case of multiple errors in the GRSPW. The error detected first has number 1.

Detection Order	Error Code	Error
1	12	Invalid destination logical address
2	2	Unused RMAP packet type or command code
3	3	Invalid destination key
4	9	Verify buffer overrun
5	11	RMW data length error
6	10	Authorization failure
7*	1	General Error (AHB errors during non-verified writes)
8	5/7	Early EOP / EEP (if early)
9	4	Invalid Data CRC
10	1	General Error (AHB errors during verified writes or RMW)
11	7	EEP
12	6	Cargo Too Large
*The AHB error is not guaranteed to be detected before Early EOP/EEP or Invalid Data CRC. For very long accesses the AHB error detection might be delayed causing the other two errors to appear first.		

Read accesses are performed on the fly, that is they are not stored in a temporary buffer before transmitting. This means that the error code 1 will never be seen in a read reply since the header has already been sent when the data is read. If the AHB error occurs the packet will be truncated and ended with an EEP.

Errors up to and including Invalid Data CRC (number 8) are checked before verified commands. The other errors do not prevent verified operations from being performed.

The details of the support for the different commands are now presented. All defined commands which are received but have an option set which is not supported in this specific implementation will not be executed and a possible reply is sent with error code 10.

47.6.3 Write commands

The write commands are divided into two subcategories when examining their capabilities: verified writes and non-verified writes. Verified writes have a length restriction of 4 bytes and the address must be aligned to the size. That is 1 byte writes can be done to any address, 2 bytes must be halfword aligned, 3 bytes are not allowed and 4 bytes writes must be word aligned. Since there will always be only on AHB operation performed for each RMAP verified write command the incrementing address bit can be set to any value.

Non-verified writes have no restrictions when the incrementing bit is set to 1. If it is set to 0 the number of bytes must be a multiple of 4 and the address word aligned. There is no guarantee how many words will be written when early EOP/EEP is detected for non-verified writes.

47.6.4 Read commands

Read commands are performed on the fly when the reply is sent. Thus if an AHB error occurs the packet will be truncated and ended with an EEP. There are no restrictions for incrementing reads but non-incrementing reads have the same alignment restrictions as non-verified writes. Note that the “Authorization failure” error code will be sent in the reply if a violation was detected even if the length field was zero. Also note that no data is sent in the reply if an error was detected i.e. if the status field is non-zero.

47.6.5 RMW commands

All read-modify-write sizes are supported except 6 which would have caused 3 B being read and written on the bus. The RMW bus accesses have the same restrictions as the verified writes. As in the verified write case, the incrementing bit can be set to any value since only one AHB bus operation will be performed for each RMW command. Cargo too large is detected after the bus accesses so this error will not prevent the operation from being performed. No data is sent in a reply if an error is detected i.e. the status field is non-zero.

47.6.6 Control

The RMAP command handler mostly runs in the background without any external intervention, but there are a few control possibilities.

There is an enable bit in the control register of the core which can be used to completely disable the RMAP command handler. When it is set to ‘0’ no RMAP packets will be handled in hardware, instead they are all stored to the DMA channel.

There is a possibility that RMAP commands will not be performed in the order they arrive. This can happen if a read arrives before one or more writes. Since the command handler stores replies in a buffer with more than one entry several commands can be processed even if no replies are sent. Data for read replies is read when the reply is sent and thus writes coming after the read might have been performed already if there was congestion in the transmitter. To avoid this the RMAP buffer disable bit can be set to force the command handler to only use one buffer which prevents this situation.

The last control option for the command handler is the possibility to set the destination key which is found in a separate register.

*Table 497.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Command	Action
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	0	-	-	-	-	Response	Stored to DMA-channel.
0	1	0	0	0	0	Not used	Does nothing. No reply is sent.
0	1	0	0	0	1	Not used	Does nothing. No reply is sent.
0	1	0	0	1	0	Read single address	Executed normally. Address has to be word aligned and data size a multiple of four. Reply is sent. If alignment restrictions are violated error code is set to 10.
0	1	0	0	1	1	Read incrementing address.	Executed normally. No restrictions. Reply is sent.
0	1	0	1	0	0	Not used	Does nothing. No reply is sent.
0	1	0	1	0	1	Not used	Does nothing. No reply is sent.
0	1	0	1	1	0	Not used	Does nothing. Reply is sent with error code 2.
0	1	0	1	1	1	Read-Modify-Write incrementing address	Executed normally. If length is not one of the allowed rmw values nothing is done and error code is set to 11. If the length was correct, alignment restrictions are checked next. 1 byte can be rmw to any address. 2 bytes must be halfword aligned. 3 bytes are not allowed. 4 bytes must be word aligned. If these restrictions are violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	0	0	0	Write, single-address, do not verify before writing, no acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done. No reply is sent.
0	1	1	0	0	1	Write, incrementing address, do not verify before writing, no acknowledge	Executed normally. No restrictions. No reply is sent.
0	1	1	0	1	0	Write, single-address, do not verify before writing, send acknowledge	Executed normally. Address has to be word aligned and data size a multiple of four. If alignment is violated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.

*Table 497.*GRSPW hardware RMAP handling of different packet type and command fields.

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Command**	**Action**
Reserved	**Command / Response**	**Write / Read**	**Verify data before write**	**Acknow-ledge**	**Increment Address**		
0	1	1	0	1	1	Write, incre-menting address, do not verify before writ-ing, send acknowledge	Executed normally. No restric-tions. If AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	0	0	Write, single address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. No reply is sent.
0	1	1	1	0	1	Write, incre-menting address, ver-ify before writing, no acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done. Same alignment restric-tions apply as for rmw. If they are violated nothing is done. No reply is sent.
0	1	1	1	1	0	Write, single address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
0	1	1	1	1	1	Write, incre-menting address, ver-ify before writing, send acknowledge	Executed normally. Length must be 4 or less. Otherwise nothing is done and error code is set to 9. Same alignment restrictions apply as for rmw. If they are vio-lated nothing is done and error code is set to 10. If an AHB error occurs error code is set to 1. Reply is sent.
1	0	-	-	-	-	Unused	Stored to DMA-channel.
1	1	-	-	-	-	Unused	Stored to DMA-channel.

47.7 AMBA interface

The AMBA interface consists of an APB interface, an AHB master interface and DMA FIFOs. The APB interface provides access to the user registers which are described in section 47.9. The DMA engines have 32-bit wide FIFOs to the AHB master interface which are used when reading and writ-ing to the bus.

The transmitter DMA engine reads data from the bus in bursts which are half the FIFO size in length. A burst is always started when the FIFO is half-empty or if it can hold the last data for the packet. The burst containing the last data might have shorter length if the packet is not an even number of bursts in size.

The receiver DMA works in the same way except that it checks if the FIFO is half-full and then performs a burst write to the bus which is half the fifo size in length. The last burst might be shorter. If the rmap or rxunaligned VHDL generics are set to 1 the interface also handles byte accesses. Byte accesses are used for non word-aligned buffers and/or packet lengths that are not a multiple of four bytes. There might be 1 to 3 single byte writes when writing the beginning and end of the received packets.

47.7.1 APB slave interface

As mentioned above, the APB interface provides access to the user registers which are 32-bits in width. The accesses to this interface are required to be aligned word accesses. The result is undefined if this restriction is violated.

47.7.2 AHB master interface

The core contains a single master interface which is used by both the transmitter and receiver DMA engines. The arbitration algorithm between the channels is done so that if the current owner requests the interface again it will always acquire it. This will not lead to starvation problems since the DMA engines always deassert their requests between accesses.

if rmap and rxunaligned are disabledThe AHB accesses can be of size byte, halfword and word (HSIZE = 0x000, 0x001, 0x010) otherwise. Byte and halfword accesses are always NONSEQ.

The burst length will be half the AHB FIFO size except for the last transfer for a packet which might be smaller. Shorter accesses are also done during descriptor reads and status writes.

The AHB master also supports non-incrementing accesses where the address will be constant for several consecutive accesses. HTRANS will always be NONSEQ in this case while for incrementing accesses it is set to SEQ after the first access. This feature is included to support non-incrementing reads and writes for RMAP.

If the core does not need the bus after a burst has finished there will be one wasted cycle (HTRANS = IDLE).

BUSY transfer types are never requested and the core provides full support for ERROR, RETRY and SPLIT responses.

47.8 Synthesis and hardware

47.8.1 Clock-generation

Figure 153 shows the clock recovery scheme for the receiver. Data and strobe are coupled directly from their pads to an xor gate which generates the clock. The output from the xor is then connected to a clock network. The specific type of clock network depends on the technology used. The xor gate is actually all that logically belongs to the Rx clock recovery module in figure 153.

The clock output drives all flip-flops in the receiver module found in figure 148. The data signal which is used for generating the clock is also coupled to the data inputs of several flip-flops clocked by the Rx clock as seen in figure 153. Care must be taken so that the delay from the data and strobe signals through the clock network are longer than the delay to the data input + setup time.

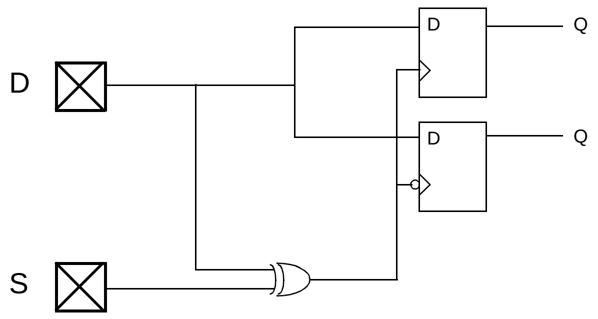


Figure 153. The clocking scheme for the receiver. The clock is

The transmitter clock is generated from the txclk input. A separate clock input is used to allow the transmitter to be run at much higher frequencies than the system clock. The SpaceWire node contains a clock-divider which divides the txclk signal to the wanted frequency. The transmitter clock should be 10 MHz during initialization and any frequency above 2 MHz in the run-state.

There is an input signal called clkdiv10 which sets the reset values for the user accessible clock divisor registers. There is one register value which is used during initialisation and one which is used in run-state The resulting tx clock frequency will be txclk/(clock divisor value+1). So if no clock division is wanted, the clock divisor should be set to 0.

Since only integer values are allowed for the clock division and the required init-frequency is 10 Mhz the frequency of the txclk input must be a multiple of 10 MHz. The clock divisor value is 8-bits wide so the maximum txclk frequency supported is 2.56 GHz (note that there is also a restriction on the relation between the system and transmit clock frequencies).

47.8.2 Timers

There are two timers in the grspw: one for generating the 6.4/12.8 us periods and one for disconnect timing.

The timeout periods are generated from the tx clock whose frequency must be at least 10 MHz to guarantee disconnect timing limits. The same clock divisor is used as for the tx clock during initialisation so it must be set correctly for the link timing to work.

47.8.3 Synchronization

The VHDL generic nsync selects how many synchronization registers are used between clock domains. The default is one and should be used when maximum performance is needed. It allows the transmitter to be clocked 4 times faster than the system clock and the receiver 2 times faster. These are theoretical values without consideration for clock skew and jitter. Note also that the receiver clocks data at both negative and positive edges. Thus, the bitrate is twice as high as the clock-rate.

The synchronization limits the Tx and Rx clocks to be at most 4 and 2 times faster than the system clock. But it might not be possible to achieve such high clock rates for the Tx and Rx clocks for all technologies.

The asynchronous reset to the receiver clock domain has to have a maximum delay of one receiver clock cycle to ensure correct operation. This is needed because the receiver uses has a completely asynchronous reset. To make sure that nothing bad happens the is a synchronous reset guard which prevents any signals from being assigned before all registers have their reset signals released.

47.8.4 Fault-tolerance

The core can optionally be implemented with fault-tolerance against SEU errors in the FIFO memories. The fault-tolerance is enabled through the *ft* VHDL generic. Possible options are byte parity protection (*ft = 1*) or TMR registers (*ft = 2*). Note: the GPL version of GRLIB does not include fault-tolerance, and the core will not work unless the *ft* VHDL generic is 0.

47.8.5 Synthesis

Since the receiver and transmitter may run on very high frequency clocks their clock signals have been coupled through a clock buffer with a technology wrapper. This clock buffer will utilize a low skew net available in the selected technology for the clock.

The clock buffer will also enable most synthesis tools to recognize the clocks and it is thus easier to find them and place constraints on them. The fact there are three clock domains in the core of which all are possibly high frequency clocks makes it necessary to declare all paths between the clock domains as false paths.

In Synplify this is most easily done by declaring all the clocks to be in different clockgroups in the sdc file (if Synplify does not automatically put them in different groups). This will disable any timing considerations between the clock domains and these constraints will also propagate to the place and route tool.

The type of clock buffer is selectable with a VHDL generic and the value zero provides a normal feed through which lets the synthesis tool infer the type of net used.

47.8.6 Technology mapping

The core has three generics for technology mapping: *tech*, *techfifo* and *memtech*. *Tech* selects the technology used for the clock buffers and also adds reset to some registers for technologies where they would otherwise cause problems with gate-level simulations. *Techfifo* selects whether *memtech* should be used to select the technology for the FIFO memories (the RMAP buffer is not affected by the this generic) or if they should be inferred. *Tech* and *memtech* can be set to any value from 0 to NTECH as defined in the GRLIB.TECH package.

47.8.7 RAM usage

The core maps all RAM memories on the syncram_2p component if the *ft* generic is 0 and to the syncram_2pft component for other values. The syncrams are located in the technology mapping library (TECHMAP). The organization of the different memories are described below. If techfifo and/ or memtech is set to 0 the synthesis tool will infer the memories. Either RAM blocks or flip-flops will be used depending on the tool and technology. The number of flip-flops used is *syncram depth x syncram width* for all the different memories. The receiver AHB FIFO with fifosize 32 will for example use 1024 flips-flops.

Receiver ahb FIFO

The receiver AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 498 shows the syncram organization for the allowed configurations.

*Table 498.*syncram_2p sizes for GRSPW receiver AHB FIFO.

Fifosize	**Syncram_2p organization**
4	4x32
8	8x32
16	16x32
32	32x32

Transmitter ahb FIFO

The transmitter AHB fifo consists of one syncram_2p block with a width of 32-bits. The depth is determined by the configured FIFO depth. Table 499 shows the syncram organization for the allowed configurations.

*Table 499.*syncram_2p sizes for transmitter AHB FIFO.

Fifosize	Syncram_2p organization
4	4x32
8	8x32
16	16x32
32	32x32

Receiver N-Char FIFO

The receiver N-Char fifo consists of one syncram_2p block with a width of 9-bits. The depth is determined by the configured FIFO depth. Table 500 shows the syncram organization for the allowed configurations.

*Table 500.*syncram_2p sizes for the receiver N-Char FIFO.

Fifosize	Syncram_2p organization
16	16x9
32	32x9
64	64x9

RMAP buffer

The RMAP buffer consists of one syncram_2p block with a width of 8-bits. The depth is determined by the number of configured RMAP buffers. Table 501 shows the syncram organization for the allowed configurations.

*Table 501.*syncram_2p sizes for RMAP buffer memory.

RMAP buffers	Syncram_2p organization
2	64x8
4	128x8
8	256x8

47.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 502.*GRSPW registers

APB address offset	Register
0x0	Control
0x4	Status/Interrupt-source
0x8	Node address
0xC	Clock divisor
0x10	Destination key
0x14	Time
0x20	DMA channel 1 control/status
0x24	DMA channel 1 rx maximum length
0x28	DMA channel 1 transmit descriptor table address.
0x2C	DMA channel 1 receive descriptor table address.
0x30	DMA channel 1 address register
0x34	Unused
0x38	Unused
0x3C	Unused
0x40 - 0x5C	DMA channel 2 registers
0x60 - 0x7C	DMA channel 3 registers
0x80 - 0x9C	DMA channel 4 registers

Table 503. GRSPW control register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RA	RX	RC	NCH		PO	RESERVED				PS	NP			RD	RE	RESERVED				TR	TT	LI	TQ		RS	PM	TI	IE	AS	LS	LD

31	RMAP available (RA) - Set to one if the RMAP command handler is available. Only readable.
30	RX unaligned access (RX) - Set to one if unaligned writes are available for the receiver. Only readable.
29	RMAP CRC available (RC) - Set to one if RMAP CRC is enabled in the core. Only readable.
28: 27	Number of DMA channels (NCH) - The number of available DMA channels minus one (Number of channels = NCH+1).
26	Number of ports (PO) - The number of available SpaceWire ports minus one.
25: 22	RESERVED
21	Port select (PS) - Selects the active port when the no port force bit is zero. ‘0’ selects the port connected to data and strobe on index 0 while ‘1’ selects index 1. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.
20	No port force (NP) - Disable port force. When disabled the port select bit cannot be used to select the active port. Instead, it is automatically selected by checking the activity on the respective receive links. Only available if the ports VHDL generic is set to 2. Reset value: ‘0’.

Table 503. GRSPW control register

19: 18	RESERVED
17	RMAP buffer disable (RD) - If set only one RMAP buffer is used. This ensures that all RMAP commands will be executed consecutively. Only available if the rmap VHDL generic is set to 1. Reset value: ‘0’.
16	RMAP Enable (RE) - Enable RMAP target. Only available if rmap VHDL generic is set to 1. Reset value: ‘1’.
15: 12	RESERVED
11	Time Rx Enable (TR) - Enable time-code receptions. Reset value: ‘0’.
10	Time Tx Enable (TT) - Enable time-code transmissions. Reset value: ‘0’.
9	Link error IRQ (LI) - Generate interrupt when a link error occurs. Not reset.
8	Tick-out IRQ (TQ) - Generate interrupt when a valid time-code is received. Not reset.
7	RESERVED
6	Reset (RS) - Make complete reset of the SpaceWire node. Self clearing. Reset value: ‘0’.
5	Promiscuous Mode (PM) - Enable Promiscuous mode. Reset value: ‘0’.
4	Tick In (TI) - The host can generate a tick by writing a one to this field. This will increment the timer counter and the new value is transmitted after the current character is transferred. A tick can also be generated by asserting the tick_in signal. Reset value: ‘0’.
3	Interrupt Enable (IE) - If set, an interrupt is generated when one of bit 8 to 10 is set and its corresponding event occurs. Reset value: ‘0’.
2	Autostart (AS) - Automatically start the link when a NULL has been received. Not reset.
1	Link Start (LS) - Start the link, i.e. allow a transition from ready to started state. Reset value: ‘0’ if the RMAP command handler is not available. If available the reset value is set to the value of the rmapen input signal.
0	Link Disable (LD) - Disable the SpaceWire codec. Reset value: ‘0’.

Table 504. GRSPW status register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED								LS			RESERVED											AP	EE	IA			PE	DE	ER	CE	TO

31: 24	RESERVED
23: 21	Link State (LS) - The current state of the start-up sequence. 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. Reset value: 0.
20: 10	RESERVED
9	Active port (AP) - Shows the currently active port. ‘0’ = Port 0 and ‘1’ = Port 1 where the port numbers refer to the index number of the data and strobe signals. Only available if the ports generic is set to 2.
8	Early EOP/EEP (EE) - Set to one when a packet is received with an EOP after the first byte for a non-rmap packet and after the second byte for a RMAP packet. Cleared when written with a one. Reset value: ‘0’.
7	Invalid Address (IA) - Set to one when a packet is received with an invalid destination address field, i.e it does not match the nodeaddr register. Cleared when written with a one. Reset value: ‘0’.

Table 504. GRSPW status register

6: 5	RESERVED
4	Parity Error (PE) - A parity error has occurred. Cleared when written with a one. Reset value: ‘0’.
3	Disconnect Error (DE) - A disconnection error has occurred. Cleared when written with a one. Reset value: ‘0’.
2	Escape Error (ER) - An escape error has occurred. Cleared when written with a one. Reset value: ‘0’.
1	Credit Error (CE) - A credit has occurred. Cleared when written with a one. Reset value: ‘0’.
0	Tick Out (TO) - A new time count value was received and is stored in the time counter field. Cleared when written with a one. Reset value: ‘0’.

Table 505. GRSPW default address register

31 16	15 8	7 0
RESERVED	DEFMASK	DEFADDR

31: 8	RESERVED
15: 8	Default mask (DEFMASK) - Default mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the DEFADDR field are anded with the inverse of DEFMASK before the address check.
7: 0	Default address (DEFADDR) - Default address used for node identification on the SpaceWire network. Reset value: 254.

Table 506. GRSPW clock divisor register

31 16	15 8	7 0
RESERVED	CLKDIVSTART	CLKDIVRUN

31: 16	RESERVED
15: 8	Clock divisor startup (CLKDIVSTART) - Clock divisor value used for the clock-divider during startup (link-interface is in other states than run). The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.
7: 0	Clock divisor run (CLKDIVRUN) - Clock divisor value used for the clock-divider when the link-interface is in the run-state. The actual divisor value is Clock Divisor register + 1. Reset value: clkdiv10 input signal.

Table 507. GRSPW destination key

31 8	7 0
RESERVED	DESTKEY

31: 8	RESERVED
7: 0	Destination key (DESTKEY) - RMAP destination key. Only available if the rmap VHDL generic is set to 1. Reset value: 0.

Table 508. GRSPW time register

31 8	7 6	5 0
RESERVED	TCTRL	TIMECNT

31: 8	RESERVED

Table 508. GRSPW time register

7: 6	Time control flags (TCTRL) - The current value of the time control flags. Sent with time-code resulting from a tick-in. Received control flags are also stored in this register. Reset value: ‘0’.
5: 0	Time counter (TIMECNT) - The current value of the system time counter. It is incremented for each tick-in and the incremented value is transmitted. The register can also be written directly but the written value will not be transmitted. Received time-counter values are also stored in this register. Reset value: ‘0’.

Table 509. GRSPW dma control register

31–17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED	LE	SP	SA	EN	NS	RD	RX	AT	RA	TA	PR	PS	AI	RI	TI	RE	TE

31: 17	RESERVED
16	Link error disable (LE) - Disable transmitter when a link error occurs. No more packets will be transmitted until the transmitter is enabled again. Reset value: ‘0’.
15	Strip pid (SP) - Remove the pid byte (second byte) of each packet. The address byte (first byte) will also be removed when this bit is set independent of the SA bit. Reset value: ‘0’.
14	Strip addr (SA) - Remove the addr byte (first byte) of each packet. Reset value: ‘0’.
13	Enable addr (EN) - Enable separate node address for this channel. Reset value: ‘0’.
12	No spill (NS) - If cleared, packets will be discarded when a packet is arriving and there are no active descriptors. If set, the GRSPW will wait for a descriptor to be activated.
11	Rx descriptors available (RD) - Set to one, to indicate to the GRSPW that there are enabled descriptors in the descriptor table. Cleared by the GRSPW when it encounters a disabled descriptor: Reset value: ‘0’.
10	RX active (RX) - Is set to ‘1’ if a reception to the DMA channel is currently active otherwise it is ‘0’. Only readable.
9	Abort TX (AT) - Set to one to abort the currently transmitting packet and disable transmissions. If no transmission is active the only effect is to disable transmissions. Self clearing. Reset value: ‘0’.
8	RX AHB error (RA) - An error response was detected on the AHB bus while this receive DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.
7	TX AHB error (TA) - An error response was detected on the AHB bus while this transmit DMA channel was accessing the bus. Cleared when written with a one. Reset value: ‘0’.
6	Packet received (PR) - This bit is set each time a packet has been received. never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.
5	Packet sent (PS) - This bit is set each time a packet has been sent. Never cleared by the SW-node. Cleared when written with a one. Reset value: ‘0’.
4	AHB error interrupt (AI) - If set, an interrupt will be generated each time an AHB error occurs when this DMA channel is accessing the bus. Not reset.
3	Receive interrupt (RI) - If set, an interrupt will be generated each time a packet has been received. This happens both if the packet is terminated by an EEP or EOP. Not reset.
2	Transmit interrupt (TI) - If set, an interrupt will be generated each time a packet is transmitted. The interrupt is generated regardless of whether the transmission was successful or not. Not reset.
1	Receiver enable (RE) - Set to one when packets are allowed to be received to this channel. Reset value: ‘0’.
0	Transmitter enable (TE) - Write a one to this bit each time new descriptors are activated in the table. Writing a one will cause the SW-node to read a new descriptor and try to transmit the packet it points to. This bit is automatically cleared when the SW-node encounters a descriptor which is disabled. Reset value: ‘0’.

Table 510. GRSPW RX maximum length register.

31–25	24–0
RESERVED	RXMAXLEN

Table 510. GRSPW RX maximum length register.

31: 25	RESERVED
24: 0	RX maximum length (RXMAXLEN) - Receiver packet maximum length in bytes. Only bits 24 - 2 are writable. Bits 1 - 0 are always 0. Not reset.

Table 511. GRSPW transmitter descriptor table address register.

31 10	9 4	3 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 4	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 16 and eventually wrap to zero again. Reset value: 0.
3: 0	RESERVED

Table 512. GRSPW receiver descriptor table address register.

31 10	9 3	2 0
DESCBASEADDR	DESCSEL	RESERVED

31: 10	Descriptor table base address (DESCBASEADDR) - Sets the base address of the descriptor table. Not reset.
9: 3	Descriptor selector (DESCSEL) - Offset into the descriptor table. Shows which descriptor is currently used by the GRSPW. For each new descriptor read, the selector will increase with 8 and eventually wrap to zero again. Reset value: 0.
2: 0	RESERVED

Table 513. GRSPW dma channel address register

31 16	15 8	7 0
RESERVED	MASK	ADDR

31: 8	RESERVED
15: 8	Mask (MASK) - Mask used for node identification on the SpaceWire network. This field is used for masking the address before comparison. Both the received address and the ADDR field are anded with the inverse of MASK before the address check.
7: 0	Address (ADDR) - Address used for node identification on the SpaceWire network for the corresponding dma channel when the EN bit in the DMA control register is set. Reset value: 254.

47.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x29. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

47.11 Configuration options

Table 514 shows the configuration options of the core (VHDL generics).

*Table 514.*Configuration options

Generic	Function	Allowed range	Default
tech	Technology for fifo memories.	0 - NTECH	inferred
hindex	AHB master index.	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	Addr field of the APB bar.	0 - 16#FFF#	0
pmask	Mask field of the APB bar.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by GRSPW.	0 - NAHBIRQ-1	0
nsync	Number of synchronization registers.	1 - 2	1
rmap	Include hardware RMAP command handler. RMAP CRC logic will also be added.	0 - 1	0
rmapcrc	Enable RMAP CRC logic.	0 - 1	0
fifosize1	Sets the number of entries in the 32-bit receiver and transmitter AHB fifos.	4 - 32	32
fifosize2	Sets the number of entries in the 9-bit receiver fifo (N-Char fifo).	16 - 64	64
rxclkbuftype	Select clock buffer type for receiver clock. 0 does not select a buffer, instead i connects the input directly to the output (synthesis tools may still infer a buffer). 1 selects hardwired clock while 2 selects routed clock.	0 - 2	0
rxunaligned	Receiver unaligned write support. If set, the receiver can write any number of bytes to any start address without writing any excessive bytes.	0 - 1	0
rmapbufs	Sets the number of buffers to hold RMAP replies.	2 - 8	4
ft	Enable fault-tolerance against SEU errors	0 - 2	0
ports	Sets the number of ports	1 - 2	1

47.12 Signal descriptions

Table 515 shows the interface signals of the core (VHDL ports).

Table 515. Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
TXCLK	N/A	Input	Transmitter default run-state clock	-
AHBMI	*	Input	AMB master input signals	-
AHBMO	*	Output	AHB master output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SWNI	D	Input	Data input	-
	S	Input	Strobe input	-
	TICKIN	Input	Time counter tick input	High
	CLKDIV10	Input	Clock divisor value used during initialization and as reset value for the clock divisor register	-
	RMAPEN	Input	Reset value for the rmapen control register bit	-
SWNO	D	Output	Data output	-
	S	Output	Strobe output	-
	TICKOUT	Output	Time counter tick output	High
* see GRLIB IP Library User’s Manual				

47.13 Library dependencies

Table 516 shows libraries used when instantiating the core (VHDL libraries).

*Table 516.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	SPACEWIRE	Signals, component	Component and record declarations.

47.14 Instantiation

This example shows how the core can be instantiated.

Normally di, si, do and so should be connected to input and output pads configured with LVDS drivers. How this is done is technology dependent.

The core in the example is configured with non-ft memories of size 4, 64 and 8 entries for AHB FIFOs, N-Char FIFO and RMAP buffers respectively. The system frequency (clk) is 40 MHz and the transmitter frequency (txclk) is 20 MHz.

The memory technology is inferred which means that the synthesis tool will select the appropriate components. The rx clk buffer uses a hardwired clock.

The hardware RMAP command handler is enabled which also automatically enables rxunaligned and rmapcrc. The Finally, the DMA channel interrupt line is 2 and the number of synchronization registers is 1.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.spacewire.all;

entity spacewire_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- spacewire signals
    di :  in  std_logic_vector(1 downto 0);
    si :  in  std_logic_vector(1 downto 0);
    do :  out std_logic_vector(1 downto 0);
    so :  out std_logic_vector(1 downto 0)
    );
end;

architecture rtl of spacewire_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- Spacewire signals
  signal swni : grspw_in_type;
  signal swno : grspw_out_type;

begin

  -- AMBA Components are instantiated here
  ...

  -- GRSPW
  sw0 : grspw2
  generic map (tech => inferred, hindex => 5, pindex => 7, paddr => 7, nsync => 1,
    rmap => 1, rxunaligned => 0, rmapcrc => 0, rxclkbuftype => 0, pirq => 2,
    fifosize1 => 4, fifosize2 => 64, rmapbufs => 8, ft => 0, ports => 2,
    dmachan => 4)
  port map (rstn, clk, apbi, apbo(7), ahbmi, ahbmo(5), swni, swno);

  swni.rmapen   <= ‘1’;
  swni.clkdiv10 <= “00000001”;
  swni.tickin   <= ‘0’;
  swni.d(0)     <= di(0);
  swni.s(0)     <= si(0);
  do(0)         <= swno.d(0);
  so(0)         <= swno.s(0);
 swni.d(1)     <= di(1);
  swni.s(1)     <= si(1);
  do(1)         <= swno.d(1);
  so(1)         <= swno.s(1);
end;
```

47.15 RTEMS Driver

The RTEMS GRSPW driver supports the standard accesses to file descriptors such as read, write and ioctl. User applications should include the file *spacewire.h* which contains definitions of all necessary data structures used when accessing the driver and a function for registration. An example application using the driver called *rtems-spwtest* is provided in the Gaisler Research RTEMS distribution.

47.15.1 Driver registration

The function *spacewire_register* whose prototype is provided in *spacewire.h* is used for registering the driver. It returns 0 on success and 1 on failure.

47.15.2 Opening the device

After the driver is registered the device should be opened next. It is done with the open call. An example of a open call is shown below.

```
fd = open("/dev/spacewire", O_RDONLY)
```

A file descriptor is returned on success and -1 otherwise. In the latter case errno is set.

*Table 517.*Open errno values.

ERRNO	Description
EINVAL	Illegal device name or not available.
EIO	Error when writing to grspw registers.
ETIMEDOUT	Link did not startup.

47.15.3 Closing the device

The device is closed using the close call. An example is shown below.

```
res = close(fd)
```

Close always returns 0 (success) for the Spacewire driver.

47.15.4 Data structures

The spw_ioctl_packetsize struct is used when changing the size of the drivers' receive and transmit buffers.

```
typedef struct {
   unsigned int rxsize;
   unsigned int txdsize;
   unsigned int txhsize;
} spw_ioctl_packetsize;
```

*Table 518.*spw_ioctl_packetsize member descriptions.

Member	Description
rxsize	Sets the size of the receiver descriptor buffers.
txdsize	Sets the size of the transmitter data buffers.
txhsize	Sets the size of the transmitter header buffers.

The spw_ioctl_pkt_send struct is used for transmissions through the ioctl call. Se the transmission section for more information. The sent variable is set by the driver when returning from the ioctl call while the other are set by the caller.

```
typedef struct {
   unsigned int hlen;
   char *hdr;
   unsigned int dlen;
   char *data;
   unsigned int sent;
} spw_ioctl_pkt_send;
```

*Table 519.*spw_ioctl_pkt_send member descriptions.

Member	Description
hlen	Number of bytes that shall be transmitted from the header buffer.
hdr	Pointer to the header buffer.
dlen	Number of bytes that shall be transmitted from the data buffer.
data	Pointer to the data buffer.
sent	Number of bytes transmitted.

The spw_stats struct contains various statistics gathered from the GRSPW.

```
typedef struct {
   unsigned int tx_link_err;
   unsigned int rx_rmap_header_crc_err;
   unsigned int rx_rmap_data_crc_err;
   unsigned int rx_eep_err;
   unsigned int rx_truncated;
   unsigned int parity_err;
   unsigned int escape_err;
   unsigned int credit_err;
   unsigned int write_sync_err;
   unsigned int disconnect_err;
   unsigned int early_ep;
   unsigned int invalid_address;
   unsigned int packets_sent;
   unsigned int packets_received;
} spw_stats;
```

*Table 520.*spw_stats member descriptions.

Member	Description
tx_link_err	Number of link-errors detected during transmission.
rx_rmap_header_crc_err	Number of RMAP header CRC errors detected in received packets.
rx_rmap_data_crc_err	Number of RMAP data CRC errors detected in received packets.
rx_eep_err	Number of EEPs detected in received packets.
rx_truncated	Number of truncated packets received.
parity_err	Number of parity errors detected.
escape_err	Number of escape errors detected.
credit_err	Number of credit errors detected.
write_sync_err	Number of write synchronization errors detected.
disconnect_err	Number of disconnect errors detected.
early_ep	Number of packets received with an early EOP/EEP.
invalid_address	Number of packets received with an invalid destination address.
packets_sent	Number of packets transmitted.
packets_received	Number of packets received.

The spw_config struct holds the current configuration of the GRSPW.

```
typedef struct {
   unsigned int nodeaddr;
   unsigned int destkey;
   unsigned int clkdiv;
   unsigned int rxmaxlen;
   unsigned int timer;
   unsigned int disconnect;
   unsigned int promiscuous;
   unsigned int timetxen;
   unsigned int timerxen;
   unsigned int rmapen;
   unsigned int rmapbufdis;
   unsigned int linkdisabled;
   unsigned int linkstart;

   unsigned int check_rmap_err;
   unsigned int rm_prot_id;
   unsigned int tx_blocking;
   unsigned int tx_block_on_full;
   unsigned int rx_blocking;
   unsigned int disable_err;
   unsigned int link_err_irq;
   rtems_id event_id;

   unsigned int is_rmap;
   unsigned int is_rxunaligned;
   unsigned int is_rmapcrc;
} spw_config;
```

*Table 521.*spw_config member descriptions.

Member	**Description**
nodeaddr	Node address.
destkey	Destination key.
clkdiv	Clock division factor.
rxmaxlen	Receiver maximum packet length.
timer	Link-interface 6.4 us timer value.
disconnect	Link-interface disconnection timeout value.
promiscuous	Promiscuous mode.
timetxen	Time-code transmission enable.
timerxen	Time-code reception enable.
rmapen	RMAP command handler enable.
rmapbufdis	RMAP multiple buffer enable.
linkdisabled	Linkdisabled.
linkstart	Linkstart.
check_rmap_error	Check for RMAP CRC errors in received packets.
rm_prot_id	Remove protocol ID from received packets.
tx_blocking	Select between blocking and non-blocking transmissions.
tx_block_on_full	Block when all transmit descriptors are occupied.
rx_blocking	Select between blocking and non-blocking receptions.
disable_err	Disable Link automatically when link-error interrupt occurs.
link_err_irq	Enable link-error interrupts.
event_id	Task ID to which event is sent when link-error interrupt occurs.
is_rmap	RMAP command handler available.
is_rxunaligned	RX unaligned support available.
is_rmapcrc	RMAP CRC support available.

47.15.5 Configuration

The GRSPW core and driver are configured using ioctl calls. The table below lists all the supported calls. SPACEWIRE_IOCTRL_ should be concatenated with the call number in the table to get the actual constant used in the code. Return values for all calls are 0 for success and -1 for failure. Errno is set after a failure.

An example of a ioctl is shown below:

```
result = ioctl(fd, SPACEWIRE_IOCTRL_SET_NODEADDR, 0xFE);
```

*Table 522.*ERRNO values for ioctl calls.

ERRNO	**Description**
EINVAL	Null pointer or an out of range value was given as the argument.
EBUSY	Only used for SEND. Returned when no descriptors are available in non-blocking mode.
ENOSYS	Returned for SET_DESTKEY if RMAP command handler is not available or if an non-implemented call is used.
ETIMEDOUT	Returned for SET_PACKETSIZE if the link did not start after the size change.
ENOMEM	Returned for SET_PACKETSIZE if it was unable to allocate the new buff-ers.
EIO	Error when writing to grspw registers.

*Table 523.*Ioctl calls supported by the GRSPW driver.

Call Number	**Description**
SET_NODEADDR	Change node address.
SET_RXBLOCK	Change blocking mode of receptions.
SET_DESTKEY	Change destination key.
SET_CLKDIV	Change clock division factor.
SET_TIMER	Change timer setting.
SET_DISCONNECT	Change disconnection timeout.
SET_PROMISCUOUS	Enable/Disable promiscuous mode.
SET_RMAPEN	Enable/Disable RMAP command handler.
SET_RMAPBUFDIS	Enable/Disable multiple RMAP buffer utilization.
SET_CHECK_RMAP	Enable/Disable RMAP CRC error check for reception.
SET_RM_PROT_ID	Enable/Disable protocol ID removal for reception.
SET_TXBLOCK	Change blocking mode of transmissions.
SET_TXBLOCK_ON_FULL	Change the blocking mode when all descriptors are in use.
SET_DISABLE_ERR	Enable/Disable automatic link disabling when link error occurs.
SET_LINK_ERR_IRQ	Enable/Disable link error interrupts.
SET_EVENT_ID	Change the task ID to which link error events are sent.
SET_PACKETSIZE	Change buffer sizes.
GET_LINK_STATUS	Read the current link status.
SET_CONFIG	Set all configuration parameters with one call.
GET_CONFIG	Read the current configuration parameters.
GET_STATISTICS	Read statistics.
CLR_STATISTICS	Clear all statistics.
SEND	Send a packet with both header and data buffers.
LINKDISABLE	Disable the link.
LINKSTART	Start the link.

SET_NODEADDR

This call sets the node address. It is only used to check the destination of incoming packets. The argument must be an integer in the range 0 to 255. The call will fail if the argument contains an illegal value or if the register can not be written.

SET_RXBLOCK

This call sets the blocking mode for receptions. The argument must be an integer in the range 0 to 1. 0 selects non blocking mode while 1 selects blocking mode. The call will fail if the argument contains an illegal value.

SET_DESTKEY

This call sets the destination key. It can only be used if the RMAP command handler is available. The argument must be an integer in the range 0 to 255. The call will fail if the argument contains an illegal value, if the RMAP command handler is not available or if the register cannot be written.

SET_CLKDIV

This call sets the clock division factor used in the run-state. The argument must be an integer in the range 0 to 255. The call will fail if the argument contains an illegal value or if the register cannot be written.

SET_TIMER

This call sets the counter used to generate the 6.4 and 12.8 us time-outs in the link-interface FSM. The argument must be an integer in the range 0 to 4095. The call will fail if the argument contains an illegal value or if the register cannot be written.

SET_DISCONNECT

This call sets the counter used to generate the 850 ns disconnect interval in the link-interface FSM. The argument must be an integer in the range 0 to 1023. The call will fail if the argument contains an illegal value or if the register cannot be written.

SET_PROMISCUOUS

This call sets the promiscuous mode bit. The argument must be an integer in the range 0 to 1. The call will fail if the argument contains an illegal value or if the register cannot be written.

SET_RMAPEN

This call sets the RMAP enable bit. It can only be used if the RMAP command handler is available. The argument must be an integer in the range 0 to 1. The call will fail if the argument contains an illegal value, if the RMAP command handler is not available or if the register cannot be written.

SET_RMAPBUFDIS

This call sets the RMAP buffer disable bit. It can only be used if the RMAP command handler is available. The argument must be an integer in the range 0 to 1. The call will fail if the argument contains an illegal value, if the RMAP command handler is not available or if the register cannot be written.

SET_CHECK_RMAP

This call selects whether or not RMAP CRC should be checked for received packets. If enabled the header CRC error and data CRC error bits are checked and if one or both are set the packet will be discarded. The argument must be an integer in the range 0 to 1. 0 disables and 1 enables the RMAP CRC check. The call will fail if the argument contains an illegal value.

SET_RM_PROT_ID

This call selects whether or not the protocol ID should be removed from received packets. It is assumed that all packets contain a protocol ID so when enabled the second byte (the one after the node address) in the packet will be removed. The argument must be an integer in the range 0 to 1. 0 disables and 1 enables the RMAP CRC check. The call will fail if the argument contains an illegal value.

SET_TXBLOCK

This call sets the blocking mode for transmissions. The argument must be an integer in the range 0 to 1. 0 selects non blocking mode while 1 selects blocking mode. The call will fail if the argument contains an illegal value.

SET_TXBLOCK_ON_FULL

This call sets the blocking mode for transmissions when all descriptors are in use. The argument must be an integer in the range 0 to 1. 0 selects non blocking mode while 1 selects blocking mode. The call will fail if the argument contains an illegal value.

SET_DISABLE_ERR

This call sets automatic link-disabling due to link-error interrupts. Link-error interrupts must be enabled for it to have any effect. The argument must be an integer in the range 0 to 1. 0 disables automatic link-disabling while a 1 enables it. The call will fail if the argument contains an illegal value.

SET_LINK_ERR_IRQ

This call sets the link-error interrupt bit in the control register. The interrupt-handler sends an event to the task specified with the event_id field when this interrupt occurs. The argument must be an integer in the range 0 to 1. The call will fail if the argument contains an illegal value or if the register write fails.

SET_EVENT_ID

This call sets the task ID to which an event is sent when a link-error interrupt occurs. The argument can be any positive integer. The call will fail if the argument contains an illegal value.

SET_PACKETSIZE

This call changes the size of buffers and consequently the maximum packet sizes. The this cannot be done while the link is running so first it is stopped and then the old buffers are deallocated. Lastly the new buffers are allocated and the link is started again. The configuration before the call will be preserved (except for the packet sizes). The argument must be a pointer to a spw_ioctl_packetsize struct. The call will fail if the argument contains an illegal pointer, the requested buffer sizes cannot be allocated or the link cannot be re-started.

GET_LINK_STATUS

This call returns the current link status. The argument must be a pointer to an integer. The return value in the argument can be one of the following: 0 = Error-reset, 1 = Error-wait, 2 = Ready, 3 = Started, 4 = Connecting, 5 = Run. The call will fail if the argument contains an illegal pointer.

GET_CONFIG

This call returns all configuration parameters in a spw_config struct which is defined in spacewire.h. The argument must be a pointer to a spw_config struct. The call will fail if the argument contains an illegal pointer.

GET_STATISTICS

This call returns all statistics in a spw_stats struct. The argument must be a pointer to a spw_stats struct. The call will fail if the argument contains an illegal pointer.

CLR_STATISTICS

This call clears all statistics. No argument is taken and the call always succeeds.

SEND

This call sends a packet. The difference to the normal write call is that separate data and header buffers can be used. The argument must be a pointer to a spw_ioctl_send struct. The call will fail if the argument contains an illegal pointer, or the struct contains illegal values. See the transmission section for more information.

LINKDISABLE

This call disables the link (sets the linkdisable bit to 1 and the linkstart bit to 0). No argument is taken. The call fails if the register write fails.

LINKSTART

This call starts the link (sets the linkdisable bit to 0 and the linkstart bit to 1). No argument is taken. The call fails if the register write fails.

47.15.6 Transmission

Transmissions are done with either the write call or a special ioctl call. Write calls are used when data only needs to be taken from a single contiguous buffer. An example of a write call is shown below:

```
result = write(fd, tx_pkt, 10))
```

On success the number of transmitted bytes is returned and -1 on failure. Errno is also set in the latter case. Tx_pkt points to the beginning of the packet which includes the destination node address. The last parameter sets the number of bytes that the user wants to transmit.

The call will fail if the user tries to send more bytes than is allocated for a single packet (this can be changed with the SET_PACKETSIZE ioctl call) or if a NULL pointer is passed.

The write call can be configured to block in different ways. If normal blocking is enabled the call will only return when the packet has been transmitted. I non-blocking mode, the transmission is only set up in the hardware and then the function returns immediately (that is before the packet is actually sent). If there are no resources available in the non-blocking mode the call will return with an error.

There is also a feature called Tx_block_on_full which means that the write call blocks when all descriptors are in use.

The ioctl call used for transmissions is SPACEWIRE_IOCTRL_SEND. A spw_ioctl_send struct is used as argument and contains length, and pointer fields. The structure is shown in the data structures section. This ioctl call should be used when a header is taken from one buffer and data from another. The header part is always transmitted first. The hlen field sets the number of header bytes to be transmitted from the hdr pointer. The dlen field sets the number of data bytes to be transmitted from the data pointer. Afterwards the sent field contains the total number (header + data) of bytes transmitted.

The blocking behavior is the same as for write calls. The call fails if hlen+dlen is 0, one of the buffer pointer is zero and its corresponding length variable is nonzero.

*Table 524.*ERRNO values for write and ioctl send.

ERRNO	Description
EINVAL	An invalid argument was passed. The buffers could be null pointers or the length parameters could be 0 or larger than the maximum allowed size.
EBUSY	The packet could not be transmitted because all descriptors are in use (only in non-blocking mode).

47.15.7 Reception

Reception is done using the read call. An example is shown below:

```
len = read(fd, rx_pkt, tmp);
```

The requested number of bytes to be read is given in tmp. The packet will be stored in rx_pkt. The actual number of received bytes is returned by the function on success and -1 on failure. In the latter case errno is also set.

The call will fail if a null pointer is passed.

The blocking behavior can be set using ioctl calls. In blocking mode the call will block until a packet has been received. In non-blocking mode, the call will return immediately and if no packet was available -1 is returned and errno set appropriately. The table below shows the different errno values that can be returned.

*Table 525.*ERRNO values for read calls.

ERRNO	Description
EINVAL	A NULL pointer was passed as the data pointer or the length was illegal.
EBUSY	No data could be received (no packets available) in non-blocking mode.

47.16 API

A simple Application Programming Interface (API) is provided together with the GRSPW. The API is located in $(GRLIB)/software/spw. The files are rmapapi.c, spwapi.c, rmapapi.h, spwapi.h. The spwapi.h file contains the declarations of the functions used for configuring the GRSPW and transferring data. The corresponding definitions are located in spwapi.c. The rmapapi is structured in the same manner and contains a function for building RMAP packets.

These functions could be used as a simple starting point for developing drivers for the GRSPW. The different functions are described in this section.

47.16.1 GRSPW Basic API

The basic GRSPW API is based on a struct spwvars which stores all the information for a single GRSPW core. The information includes its address on the AMBA bus as well as SpaceWire parameters such as node address and clock divisor. A pointer to this struct is used as a input parameter to all the functions. If several cores are used, a separate struct for each core is created and used when the specific core is accessed.

*Table 526.*The spwvars struct

Field	Description	Allowed range
regs	Pointer to the GRSPW	-
nospill	The nospill value used for the core.	0 - 1
rmap	Indicates whether the core is configured with RMAP. Set by spw_init.	0 - 1
rxunaligned	Indicates whether the core is configured with rxunaligned support. Set by spw_init.	0 - 1
rmapcrc	Indicates whether the core is configured with RMAPCRC support. Set by spw_init.	0 - 1
clkdiv	The clock divisor value used for the core.	0 - 255
nodeaddr	The node address value used for the core.	0 - 255
destkey	The destination key value used for the core.	0 - 255
rxmaxlen	The Receiver maximum length value used for the core.	0 - 33554431
rxpnt	Pointer to the next receiver descriptor.	0 - 127
rxchkpnt	Pointer to the next receiver descriptor that will be polled.	0 - 127
txpnt	Pointer to the next transmitter descriptor.	0 - 63
txchkpnt	Pointer to the next transmitter descriptor that will be polled.	0 - 63
timetxen	The timetxen value used for this core.	0 - 1
timerxen	The timerxen value used for this core.	0 - 1
txd	Pointer to the transmitter descriptor table.	-
rxd	Pointer to the receiver descriptor table	-

The following functions are available in the basic API:

```
int spw_setparam(int nodeaddr, int clkdiv, int destkey, int nospill, int timetxen, int
timerxen, int rxmaxlen, int spwadr, struct spwvars *spw);
```

Used for setting the different parameters in the spwvars struct. Should always be run first after creating a spwvars struct. This function only initializes the struct. Does not write anything to the SpaceWire core.

*Table 527.*Return values for spw_setparam

Value	Description
0	The function completed successfully
1	One or more of the parameters had an illegal value

*Table 528.*Parameters for spw_setparam

Parameter	Description	Allowed range
nodeaddr	Sets the node address value of the struct spw passed to the function.	0-255
clkdiv	Sets the clock divisor value of the struct spw passed to the function.	0-255
destkey	Sets the destination key of the struct spw passed to the function.	0-255
nospill	Sets the nospill value of the struct spw passed to the function.	0 - 1
timetxen	Sets the timetxen value of the struct spw passed to the function.	0 - 1
timerxen	Sets the timerxen value of the struct spw passed to the function.	0 - 1
rxmaxlen	Sets the receiver maximum length field of the struct spw passed to the function.	0 - 2^{25}-1
spwadr	Sets the address to the GRSPW core which will be associated with the struct passed to the function.	0 - 2^{32}-1

```
int spw_init(struct spwvars *spw);
```

Initializes the GRSPW core located at the address set in the struct spw. Sets the following registers: node address, destination key, clock divisor, receiver maximum length, transmitter descriptor table address, receiver descriptor table address, ctrl and dmactrl. All bits are set to the values found in the spwvars struct. If a register bit is not present in the struct it will be set to zero. The descriptor tables are allocated to an aligned area using malloc. The status register is cleared and lastly the link interface is enabled. The run state frequency will be set according to the value in clkdiv.

*Table 529.*Return values for spw_init

Value	Description
0	The function completed successfully
1	One or more of the parameters could not be set correctly or the link failed to initialize.

*Table 530.*Parameters for spw_init

Parameter	Description	Allowed range
spw	The spwvars struct associated with the GRSPW core that should be initialized.	-

```
int set_txdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the transmitter descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_tx and spw_checktx (Explained in the section for those functions).

*Table 531.*Return values for spw_txdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 532.*Parameters for spw_txdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int set_rxdesc(int pnt, struct spwvars *spw);
```

Sets a new address to the Receiver descriptor table address register. Should only be used when no transmission is active. Also resets the pointers for spw_rx and spw_checkrx (Explained in the section for those functions).

*Table 533.*Return values for spw_rxdesc

Value	Description
0	The function completed successfully
1	The new address could not be written correctly

*Table 534.*Parameters for spw_rxdesc

Parameter	Description	Allowed range
pnt	The new address to the descriptor table area	0 - 2^{32}-1
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_disable(struct spwvars *spw);
```

Disables the GRSPW core (the link disable bit is set to ‘1’).

*Table 535.*Parameters for spw_disable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_enable(struct spwvars *spw);
```

Enables the GRSPW core (the link disable bit is set to ‘0’).

*Table 536.*Parameters for spw_enable

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_start(struct spwvars *spw);
```

Starts the GRSPW core (the link start bit is set to ‘1’).

*Table 537.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
void spw_stop(struct spwvars *spw);
```

Stops the GRSPW core (the link start bit is set to ‘0’).

*Table 538.*Parameters for spw_start

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_setclockdiv(struct spwvars *spw);
```

Sets the clock divisor register with the clock divisor value stored in the spwvars struct.

*Table 539.*Return values for spw_setclockdiv

Value	Description
0	The function completed successfully
1	The new clock divisor value is illegal.

*Table 540.*Parameters for spw_setclockdiv

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_nodeadr(struct spwvars *spw);
```

Sets the node address register with the node address value stored in the spwvars struct.

*Table 541.*Return values for spw_set_nodeadr

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 542.*Parameters for spw_set_nodeadr

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_set_rxmaxlength(struct spwvars *spw);
```

Sets the Receiver maximum length register with the rxmaxlen value stored in the spwvars struct.

*Table 543.*Return values for spw_set_rxmaxlength

Value	Description
0	The function completed successfully
1	The new node address value is illegal.

*Table 544.*Parameters for spw_set_rxmaxlength

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be configured	-

```
int spw_tx(int crc, int skipcrcsize, int hsize, char *hbuf, int dsize, char *dbuf, struct
spwvars *spw);
```

Transmits a packet. Separate header and data buffers can be used. If CRC logic is available the GSPW inserts RMAP CRC values after the header and data fields if crc is set to one. This function only sets a descriptor and initiates the transmission. Spw_checktx must be used to check if the packet has been transmitted. A pointer into the descriptor table is stored in the spwvars struct to keep track of the next location to use. It is incremented each time the function returns 0.

*Table 545.*Return values for spw_tx

Value	Description
0	The function completed successfully
1	There are no free transmit descriptors currently available
2	There was illegal parameters passed to the function

*Table 546.*Parameters for spw_tx

Parameter	Description	Allowed range
crc	Set to one to append RMAP CRC after the header and data fields. Only available if hardware CRC is available in the core.	0 - 1
skipcrcsize	The number of bytes in the beginning of a packet that should not be included in the CRC calculation	0 - 15
hsize	The size of the header in bytes	0 - 255
hbuf	Pointer to the header data	-
dsize	The size of the data field in bytes	0 - 2^{24}-1
dbuf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should transmit the packet	-

```
int spw_rx(char *buf, struct spwvars *spw);
```

Enables a descriptor for reception. The packet will be stored to buf. Spw_checkrx must be used to check if a packet has been received. A pointer in the spwvars struct is used to keep track of the next location to use in the descriptor table. It is incremented each time the function returns 0.

*Table 547.*Return values for spw_rx

Value	Description
0	The function completed successfully
1	There are no free receive descriptors currently available

*Table 548.*Parameters for spw_rx

Parameter	Description	Allowed range
buf	Pointer to the data area.	-
spw	Pointer to the spwvars struct associated with GRSPW core that should receive the packet	-

```
int spw_checkrx(int *size, struct rxstatus *rxs, struct spwvars *spw);
```

Checks if a packet has been received. When a packet has been received the size in bytes will be stored in the size parameter and status is found in the rxs struct. A pointer in the spwvars struct is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 549.*Return values for spw_checkrx

Value	Description
0	No packet has been received
1	A packet has been received

*Table 550.*Parameters for spw_checkrx

Parameter	Description	Allowed range
size	When the function returns 1 this variable holds the number of bytes received	-
rxs	When the function returns 1 this variable holds status information	-
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

*Table 551.*The rxstatus struct

Field	Description	Allowed range
truncated	Packet was truncated	0 - 1
dcrcerr	Data CRC error bit was set. Only indicates an error if the packet received was an RMAP packet.	0 - 1
hcrcerr	Header CRC error bit was se.t. Only indicates an error if the packet received was an RMAP packet.	0 - 1
eep	Packet was terminated with EEP	0 - 1

```
int spw_checktx(struct spwvars *spw);
```

Checks if a packet has been transmitted. A pointer is used to keep track of the location in the descriptor table to poll. It is incremented each time the function returns nonzero.

*Table 552.*Return values for spw_checktx

Value	Description
0	No packet has been transmitted
1	A packet has been correctly transmitted
2	A packet has been incorrectly transmitted

*Table 553.*Parameters for spw_checktx

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void send_time(struct spwvars *spw);
```

Sends a new time-code. Increments the time-counter in the GRSPW and transmits the value.

*Table 554.*Parameters for send time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int check_time(struct spwvars *spw);
```

Check if a new time-code has been received.

*Table 555.*Return values for check_time

Value	Description
0	No time-code has been received
1	A new time-code has been received

*Table 556.*Parameters for check_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
int get_time(struct spwvars *spw);
```

Get the current time counter value.

*Table 557.*Return values for get_time

Value	Description
0 - 63	Returns the current time counter value

*Table 558.*Parameters for get_time

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be polled	-

```
void spw_reset(struct spwvars *spw);
```

Resets the GRSPW.

*Table 559.*Parameters for spw_reset

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be reset	-

```
void spw_rmapen(struct spwvars *spw);
```

Enables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW.

*Table 560.*Parameters for spw_rmapen

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
void spw_rmapdis(struct spwvars *spw);
```

Disables hardware RMAP. Has no effect if the RMAP command handler is not available in GRSPW

*Table 561.*Parameters for spw_rmapdis

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

```
int spw_setdestkey(struct spwvars *spw);
```

Set the destination key of the GRSPW. Has no effect if the RMAP command handler is not available. The value from the spwvars struct is used.

*Table 562.*Return values for spw_setdestkey

Value	Description
0	The function completed successfully
1	The destination key parameter in the spwvars struct contains an illegal value

*Table 563.*Parameters for spw_setdestkey

Parameter	Description	Allowed range
spw	Pointer to the spwvars struct associated with GRSPW core that should be set.	-

47.16.2 GRSPW RMAP API

The RMAP API contains only one function which is used for building RMAP headers.

```
int build_rmap_hdr(struct rmap_pkt *pkt, char *hdr, int *size);
```

Builds a RMAP header to the buffer pointed to by hdr. The header data is taken from the rmap_pkt struct.

*Table 564.*Return values for build_rmap_hdr

Value	Description
0	The function completed successfully
1	One or more of the parameters contained illegal values

*Table 565.*Parameters for build_rmap_hdr

Parameter	Description	Allowed range
pkt	Pointer to a rmap_pkt struct which contains the data from which the header should be built	
hdr	Pointer to the buffer where the header will be built	
spw	Pointer to the spwvars struct associated with GRSPW core that should be set	-

*Table 566.*rmap_pkt struct fields

Field	Description	Allowed Range
type	Selects the type of packet to build.	writecmd, readcmd, rmwcmd, writerep, readrep, rmwrep
verify	Selects whether the data should be verified before writing	yes, no
ack	Selects whether an acknowledge should be sent	yes, no
incr	Selects whether the address should be incremented or not	yes, no
destaddr	Sets the destination address	0 - 255
destkey	Sets the destination key	0 - 255
srcaddr	Sets the source address	0 - 255
tid	Sets the transaction identifier field	0 - 65535
addr	Sets the address of the operation to be performed. The extended address field is currently always set to 0.	0 - 2^{32}-1
len	The number of bytes to be writte, read or read-modify-written	0 - 2^{24}-1
status	Sets the status field	0 - 11
dstspalen	Number of source path address bytes to insert before the destination address	0 - 228
dstspa	Pointer to memory holding the destination path address bytes	-
srcspalen	Number of source path address bytes to insert in a command. For a reply these bytes are placed before the return address	0 - 12
srcspa	Pointer to memory holding the source path address bytes	-

47.17 Appendix A Clarifications of the GRSPW implementation of the standard

6.3.1 page 9 RMAP draft F

"The user application at destination will be informed that there was an error

in the data transferred. The source will be informed of the data error if the

acknowledge bit in the command has been set."

We view the RMAP command handler as both protocol parser and user application. All commands are parsed in the first stage and various (internal) status bits are set. The next step (which can be viewed as the user application) will make the decision of how to act upon the received command from these bits. Therefore, the various errors that can occur are not externally observable.

If an error occurs when the command handler is accessing the AHB bus through the DMA interface errors will be externally observable using the AHB status register in GRLIB.

6.3.6 page 13 RMAP draft F

“The Write Command packet arrives at the destination and its header is found to be in error. This fact is added to the error statistics in the destination node.”

This text does not state how and if these statistics should be observable. At the moment the error handling is internal to the RMAP command handler and therefore no statistics are internally observable. A counter for this particular error might be added in the future.

6.3.6 page 15 RMAP draft F

"These various errors will be reported to the user application running on the

destination node (Write Data Error Indication)."

Again the RMAP command handler is the user application and all these errors are handled internally.

6.5.6 page 31 RMAP draft F

"The source user application, in fact immediately rejects this as an authorisation

failure as the command is trying to RMW an area of protected memory."

It should probably be destination user application instead of source. It is unclear what immediately means. Should it be rejected before any accesses are done on the bus and thus requiring the RMAP command handler to include a complete bus decoding. The GRSPW does (probably) not comply to this paragraph at the moment. If a bus error occurs a general error code will be returned.

6.5.6 page 32

"If the header of the RMW reply packet is received intact but the data field is

corrupted as indicated by an incorrect data field length (too long or too short)

or by a CRC error, then an error can be flagged to the application immediately

(RMW Data Failure) without having to wait for an application timeout."

This is not applicable to the GRSPW since it does not handle replies. However this is practically an unnecessary comment since it is not specified in the standard in which manner received replies are indicated to the higher layers.

48 GRUSBHC - USB 2.0 Host Controller

48.1 Overview

The Gaisler Research USB 2.0 Host Controller provides a link between the AMBA AHB bus and the Universal Serial Bus. The host controller supports High-, Full-, and Low-Speed USB traffic. USB 2.0 High-Speed functionality is supplied by an enhanced host controller implementing the Enhanced Host Controller Interface revision 1.0. Full- and Low-Speed traffic is handled by up to 15 (USB 1.1) companion controllers implementing the Universal Host Controller Interface, revision 1.1. Each controller has its own AMBA AHB master interface. Configuration and control of the enhanced host controller is done via the AMBA APB bus. Companion controller registers are accessed via an AMBA AHB slave interface. Figure 154 shows a USB 2.0 host system and the organization of the controller types. Figure 155 shows an example with both host controller types present.

The controller supports both UTMI+ and ULPI transceivers and can handle up to 15 ports.

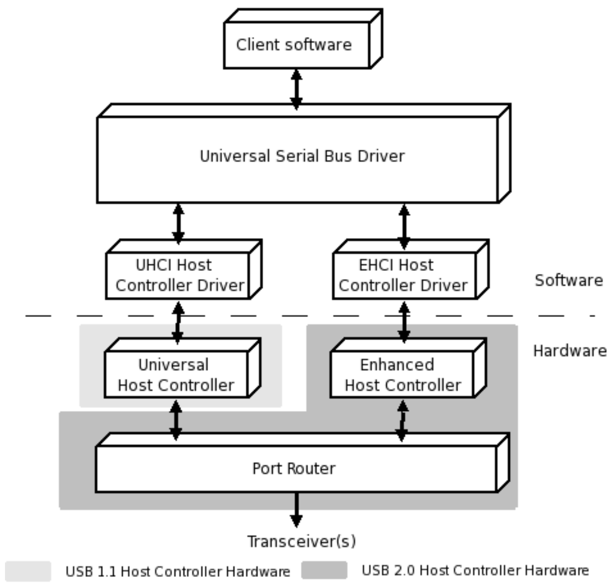

Client software
Universal Serial Bus Driver
UHCI Host Controller Driver
EHCI Host Controller Driver
Software
Hardware
Universal Host Controller
Enhanced Host Controller
Port Router
Transceiver(s)
USB 1.1 Host Controller Hardware
USB 2.0 Host Controller Hardware

Figure 154. Block diagram of USB 2.0 host system

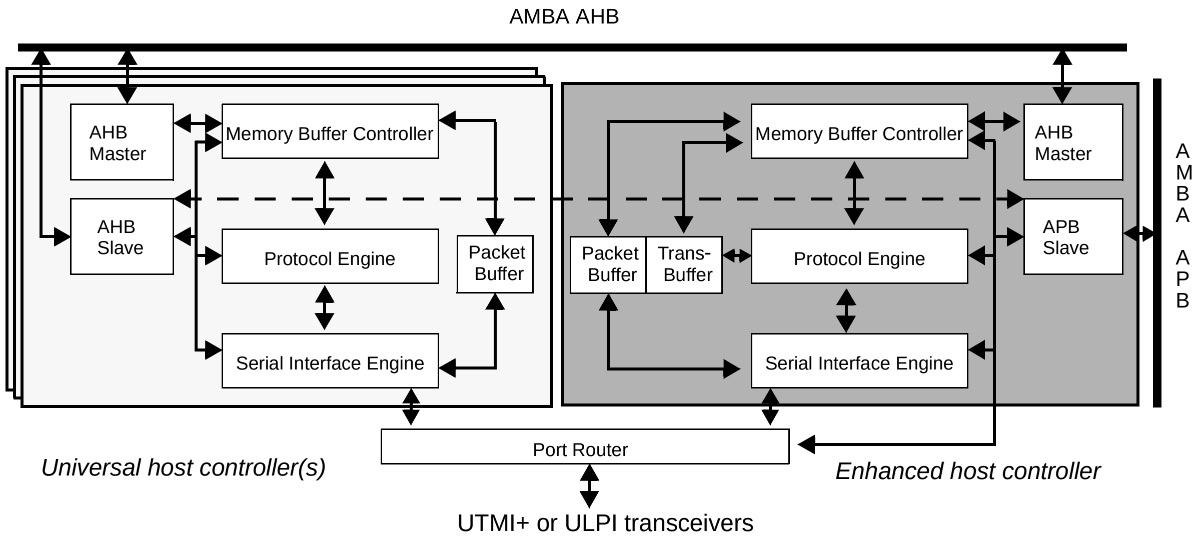


Figure 155. Block diagram of both host controller types

48.2 Operation

48.2.1 System overview

Depending on the core's configuration it may contain both controller types, one enhanced host controller, or up to 15 standalone universal host controllers. If both controller types are present, each universal host controller acts as a companion controller to the enhanced host controller.

The enhanced host controller complies with the Enhanced Host Controller Interface with the exception of the optional Light Host Controller Reset, which is not implemented.

The universal host controller complies with the Universal Host Controller Interface, with exceptions. The HCHalted field in the USB Command register is implemented as Read Only instead of Read/ Write Clear. The Port Status/Control registers have been extended with Over Current and Over Current Change fields. Changes to both registers have been done in accordance with contemporary implementations of the interface. Both changes match the description of corresponding bits in the EHCI specification.

48.2.2 Protocol support

The enhanced host controller has full support for High-Speed traffic as defined in the USB Specification, revision 2.0. In addition Asynchronous Park Mode is supported, and the controller has a NAK counter.

The universal host controller supports Full- and Low-Speed traffic.

48.2.3 Descriptor and data buffering

The enhanced host controller prefetches one frame of isochronous descriptors. All payload data for a transaction is fetched before the transaction is executed. The enhanced host controller has a 2048 byte buffer for descriptors and a 2048 byte buffer for payload data, which can hold data for two transactions.

The universal host controller does not prefetch descriptors. Depending on controller configuration a transaction on the bus may be initiated before all payload data has been fetched from memory. Each universal host controller has a 1024 byte buffer for payload data. A transfer descriptor in UHCI may describe a transaction that has a payload of 1280 bytes. The USB specification limits the maximum allowed data payload to 1023 bytes and the controller will not transfer a larger payload than 1023 bytes. If a descriptor has a, legal, larger payload than 1023 bytes, the controller will only attempt to transfer the first 1023 bytes before the transaction is marked as completed.

In the event that the host controller has just one port, the universal host controller and the enhanced host controller will share the data payload buffer. Thus only two 2048 byte buffers are required.

48.2.4 Clocking

The core has two clock domains; the system clock domain and the USB clock domain which includes the transceiver. All signals that cross a clock domain boundary are synchronized to prevent meta-stability.

48.2.5 Endianness

The core always accesses the least significant byte of a data payload at offset zero. Depending on the core's configuration, registers may be big endian, little endian, or byte swapped little endian.

48.2.6 RAM test facilities

The VHDL generic *ramtest* enables RAM testing. If RAM testing is enabled, controller's internal buffers are mapped into the register space. The universal host controller maps the packet buffer at offset 0x400 - 0x7FF. An enhanced host controller will map the packet buffer at offset 0x1000 - 0x17FF and the transaction buffer at 0x1800 - 0x1FFF. Note that the VHDL generics uhchmask and ehcpmask must be modified to allow access to the increased number of registers. The three least significant bits of the universal host controller’s mask must be set to zero. The enhanced host controller’s mask must have its five least significant bits set to zero. The mapped buffers can be read and written using only WORD accesses, but both controller types must be idle to ensure correct operation. A host controller is idle when Run/Stop is set to zero and HC Halted is set to one.

48.2.7 Scan test support

The VHDL generic scantest enables scan test support. If the core has been implemented with scan test support the core will disable the internal RAM blocks when the test enable and scan enable signals are asserted. The test and scan enable signals are routed via the APB interface to the enhanced host controller and via the AHB slave interface to the universal host controller.

48.3 Port routing

Port routing is implemented according to the EHCI specification but functions regardless of whether the core is configured with or without an enhanced host controller. The VHDL generic prr enables or disables Port Routing Rules. With Port Routing Rules enabled, each port can be individually routed to a specific universal host controller via the VHDL generics portroute1 and portroute2. If Port Routing Rules is disabled the n_pcc lowest ports are routed to the first companion controller, the next n_pcc ports to the second companion controller, and so forth. The HCSP-PORTROUTE array is communicated via the portroute VHDL generics, which are calculated with the following algorithm:

portroute1 = $2^{26}*CC_8 + 2^{22}*CC_7 + 2^{18}*CC_6 + 2^{14}*CC_5 + 2^{10}*CC_4 + 2^6*CC_3 + 2^2*CC_2 + CC_1 / 4$

portroute1 = $2^{26}*CC_{15} + 2^{22}*CC_{14} + 2^{18}*CC_{13} + 2^{14}*CC_{12} + 2^{10}*CC_{11} + 2^6*CC_{10} + 2^2*CC_9 + CC_1 \bmod 4$

where CC_P is the companion controller that port P is routed to. Companion controllers are enumerated starting at 1.

When the enhanced host controller has not been configured by software, or when it is nonexistent, each port is routed to its companion controller. This allows a universal host controller to function even if the host system does not have support for the enhanced host controller. Please see the EHCI specification for a complete description of port routing.

48.4 DMA operations

Both host controller types have configurable DMA burst lengths. The burst length in words is defined by the VHDL generic bwrd. The value of bwrd limits how many words a controller may access in memory during a burst and not the number of memory operations performed after bus access has been granted. When writing a data payload back to memory that requires half-word or byte addressing the number of memory operations may exceed bwrd by one before the bus is released. If a host controller is given a byte-aligned data buffer its burst length may exceed the bwrd limit with one word when fetching payload data from memory.

The universal host controller uses a burst length of four words when fetching descriptors. This descriptor burst length is not affected by the bwrd VHDL generic. The universal host controller may be configured to start transactions on the USB before all data has been fetched from memory. The VHDL generic uhcblo specifies the number of words that must have been fetched from memory before a USB transaction is started. Since the USB traffic handled by the universal host controller can be expected to have significantly lower bandwidth than the system memory bus, this generic should be set to a low value.

48.5 Endianness

The core works internally with little endian. If the core is connected to a big endian bus, endian conversion must be enabled. When the VHDL generic endian_conv is set, all AMBA data lines are byte swapped. With endian_conv correctly set the core will start accessing data payloads from byte offset zero in the buffer, this is the first byte that is moved on the USB. The VHDL generic endian_conv must be set correctly for byte and halfword accesses to work. Therefore it is not possible to change the byte order of the buffer by configuring the controller for a little endian bus when it is connected to a big endian bus or vice versa.

The VHDL generics be_regs and be_desc are used to place the controller into big endian mode when endian conversion is enabled. These configuration options have no effect when the core is connected to a little endian bus, as determined by the value of VHDL generic endian_conv. The VHDL generic be_regs arranges the core's registers in accordance with big endian addressing. In the enhanced host controller this will only affect the placement of the register fields CAPLENGTH and HCIVERSION, and the HCSP-PORTROUTE array. In the universal host controller be_regs will affect the placement of all registers. When be_regs is set, the bus to the register interface is never byte swapped. Tables 567 - 569 below illustrate the difference between big endian, little endian, and little endian layout with byte swapped (32 bit) WORDs on two 16 bit registers. Register R1 is located at address 0x00 and register R2 is located at address 0x02.

*Table 567.*R1 and R2 with big endian addressing

31 16	15 0
R1(15:0)	R2(15:0)

*Table 568.*R1 and R2 with little endian addressing

31 16	15 0
R2(15:0)	R1(15:0)

*Table 569.*R1 and R2 with little endian layout and byte swapped DWORD

31 24	23 16	15 8	7 0
R1(7:0)	R1(15:8)	R2(7:0)	R2(15:8)

The VHDL generic be_desc removes the byte swapping of descriptors on big endian systems. Tables 570 and 571 below list the effects of endian_conv and be_regs on a big endian and a little endian system respectively.

*Table 570.*Effect of endian_conv, be_regs, and be_desc on a big endian system

endian_conv	be_regs	be_desc	System configuration
0	-	-	Illegal. DMA will not function.
1	0	0	Host controller registers will be arranged according to little endian addressing and each DWORD will be byte swapped. In-memory transfer descriptors will also be byte swapped. This is the correct configuration for operating systems, such as Linux, that swap the bytes on big endian systems.
1	0	1	Host controller registers are arranged according to little endian addressing and will be byte swapped. Transfer descriptors will not be byte swapped.
1	1	0	Host controller registers will be arranged according to big endian addressing and will not be byte swapped. In memory transfer descriptors will be byte swapped.
1	1	1	Host controller registers will be arranged according to big endian addressing. In memory transfer descriptors will not be byte swapped.

*Table 571.*Effect of endian_conv, be_regs and be_desc on a little endian system

endian_conv	be_regs	be_desc	System configuration
0	-	-	Host controller registers will be placed as specified in the register interface specifications.
1	-	-	Illegal. DMA will not function.

48.6 Transceiver support

The controller supports UTMI+ 8-bit, UTMI+ 16-bit, and ULPI transceivers. All connected transceivers must be of the same type. Transceiver signals not belonging to the selected transceiver type are not connected and do not need to be driven.When using ULPI transceivers the default, and recommended, configuration is to use an external source for USB bus power (VBUS) as well as external VBUS fault detection. However the core can be configured to support configurations where the ULPI transceiver handles VBUS generation and fault detection internally, and configurations where VBUS generation is external to transceiver but fault detection is handled internally. Also the active level of the VBUS fault indicator can be configured. The configuration is handled by the vbusconf generic.

If UTMI+ transceivers are used it does not matter to the core how VBUS generation and fault detection is handled as long as the VBUS enable signal and VBUS fault indicator are connected to the core's drvvbus and vbusvalid signals respectively. The UTMI+ specification defines these two signals to be active high, however in order to support different types of USB power switches and fault detectors the core can be configured to have active low drvvbus and vbusvalid signals. This configuration is also handled by the vbusconf generic.

48.7 PCI configuration registers and legacy support

The controller does not implement any PCI configuration registers. Legacy support is not implemented.

48.8 Software drivers

The core implements open interface standards and should function with available drivers. Gaisler Research supplies initialization code for both controllers for the Linux 2.6 kernel and VxWorks.

48.9 Registers

48.9.1 Enhanced host controller

The core is programmed through registers mapped into APB address space. The contents of each register is described in the Enhanced Host Controller Interface Specification for Universal Serial Bus revision 1.0.

*Table 572.*Enhanced Host Controller capability registers

APB address offset	Register
0x00	Capability Register Length
0x01	Reserved
0x02	Interface Version Number
0x04	Structural Parameters
0x08	Capability Parameters
0x0C	Companion Port Route Description

*Table 573.*Enhanced Host Controller operational registers

APB address offset	Register
0x14	USB Command*
0x18	USB Status
0x1C	USB Interrupt Enable
0x20	USB Frame Index
0x24	4G Segment Selector (Reserved)
0x28	Frame List Base Address
0x2C	Next Asynchronous List Address
0x54	Configured Flag Register
0x58 - 0x90	Port Status/Control Registers**

*Light Host Controller reset is not implemented.

**One 32-bit register for each port.

48.9.2 Universal host controller

The core is programmed through registers mapped into AHB I/O address space. The contents of each register is described in the Universal Host Controller Interface (UHCI) Design Guide revision 1.1.

*Table 574.*Universal Host Controller I/O registers

AHB address offset	Register
0x00	USB Command
0x02	USB Status*
0x04	USB Interrupt Enable
0x06	Frame Number
0x08	Frame List Base Address
0x0C	Start Of Frame Modify
0x10 - 0x2C	Port Status/Control**

*The HCHalted bit is implemented as Read Only and has the default value 1.

**Over Current and Over Current Change fields have been added. Each port has a 16-bit register.

Table 575. Changes to USB Status register

15 6	5	4 0
UHCI compliant	HCH	UHCI compliant

15: 6	UHCI compliant
5	Host Controller Halted (HCH) - Same behaviour as specified in the UHCI specification but the field has been changed from Read/Write Clear to Read Only and is cleared when Run/Stop is set. The default value of this bit has been changed to 1.
4:0	UHCI compliant

Table 576. Changes to Port Status/Control registers

15	11	10	9 0
UHCI compliant	OCC	OC	UHCI compliant

15: 12	UHCI compliant
11	Over Current Change (OCC) - Set to 1 when Over Current (OC) toggles. Read/Write Clear.
10	Over Current Active (OC) - Set to 1 when there is an over current condition. Read Only.
9:0	UHCI compliant

48.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research), the enhanced host controller has device identifier 0x026, the universal host controller has device identifier 0x027. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

48.11 RAM usage

The core maps all usage of RAM on the *syncram_dp* component from the technology mapping library (TECHMAP). A universal host controller requires a 256x32 *syncram_dp* for its packet buffer. An enhanced host controller requires a 512x32 *syncram_dp* for its packet buffer and two 512x16 *syncram_dp* for its transaction buffer. When the core is instantiated with only one port, the enhanced host controller and universal host controller will share the packet buffer and the core only requires one 512x32 *syncram_dp* for the packet buffers. Table 577 below shows RAM usage for all legal configurations.

*Table 577.*RAM usage for USB Host Controller core

Enhanced Host Controller present	**Number of Universal Host Controllers**	**Number of ports**	**RAM 256x32**	**RAM 512x32**	**RAM 512x16**
No	x*	Don't care	x*	0	0
Yes	1	1	0	1	2
Yes	x*	> 1	x*	1	2

* The number of required 256x32 *syncram_dp* equals the number of instantiated universal host controllers.

48.12 Configuration options

Table 578 shows the configuration options of the core (VHDL generics).

*Table 578.*Configuration options

Generic name	**Function**	**Allowed range**	**Default**
ehchindex	Enhanced host controller AHB master index	0 - NAHBMST-1	0
ehcpindex	Enhanced host controller APB slave index	0 - NAPBSLV-1	0
ehcpaddr	Enhanced host controller ADDR field of the APB BAR.	0 - 16#FFF#	0
ehcpmask	Enhanced host controller MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
ehcpirq	Enhanced host controller interrupt line	0 - NAHBIRQ-1	0
uhchindex	Universal host controller AHB master index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhchindex to uhchindex+n_cc-1.	0 - NAHBMST-1	0
uhchsindex	Universal host controller AHB slave index. If the core contains more than one universal host controller the controllers will be assigned indexes from uhc_hsindex to uhchsindex+n_cc-1.	0 - NAHBSLV-n_cc	0
uhchaddr	Universal host controller ADDR field of the AHB BAR. If the core contains more than one universal host controller the controllers will be assigned the address space uhchaddr to uhchaddr + n_cc.	0 - 16#FFF#	0
uhchmask	Universal host controller MASK field of the AHB BAR.	0 - 16#FFF#	16#FFF#
uhchirq	Universal host controller interrupt line. If the core contains more than one universal host controller the controller will be assigned interrupt lines uhc_hirq to uhchirq+n_cc-1.	0 - NAHBIRQ-1	0
tech	Technology for clock buffers	0 - NTECH	inferred

*Table 578.*Configuration options

Generic name	Function	Allowed range	Default
memtech	Memory Technology used for buffers.	0 - NTECH	inferred
nports	Number of USB ports	1 - 15	1
ehcgen	Enable enhanced host controller	0 - 1	1
uhcgen	Enable universal host controller(s)	0 - 1	1
n_cc	Number of universal host controllers. This value must be consistent with nports and n_pcc, or portroute1 and portroute2, depending on the value of the generic prr. This value must be at least 1, regardless the value of generic uhcgen.	1 - 15	1
n_pcc	Number of ports per universal host controller. This value must be consistent with n_cc and nports: nports <= (n_cc * n_pcc) < (nports + n_pcc) when Port Routing Rules is disabled. The only allowed deviation is if (nports mod n_cc) < n_pcc in which case the last universal host controller will get (nports mod n_cc) ports. This generic is not used then Port Routing Rules (prr) is enabled.	1 - 15	1
prr	Port Routing Rules. Determines if the core’s ports are routed to companion controller(s) with n_cc and n_pcc or with the help of portroute1 and portroute2.	0 - 1	0
portroute1	Defines part of the HCSP-PORTROUTE array	-	0
portroute2	Defines part of the HCSP-PORTROUTE array	-	0
endian_conv	Enable endian conversion. When set, all AMBA data lines are byte swapped. This generic must be set to 1 if the core is attached to a big endian bus, it must be set to 0 if the core is attached to a little endian bus.	0 - 1	1
be_regs	Arrange host controller registers according to big endian addressing. When set no endian conversion is made on the AMBA data lines connected to the host controller registers, regardless of endian_conv. Valid when endian_conv is enabled.	0 - 1	0
be_desc	Disable byte swapping of in-memory descriptors. Valid when endian_conv is enabled.	0 - 1	0
uhcblo	Universal Host Controller Buffer Limit Out. A universal host controller will start OUT bus transactions when uhcblo words of payload data has been fetched from memory. Note that if the core uses the UTMI+ 16 bit interface this generic must have a value larger than 2.	1 - 255	2
bwrd	Burst length in words. A universal host controller has a fixed, not affected by bwrd, burst length of four words when fetching transfer descriptors. See comments under section 48.2 DMA operations.	0 - 256	16
utm_type	Transceiver type: 0: UTMI+ 16 bit data bus 1: UTMI+ 8 bit data bus 2: ULPI	0 - 2	2

*Table 578.*Configuration options

Generic name	Function	Allowed range	Default
vbusconf*	Selects configuration for USB power source and fault detection: ULPI transceivers: 0: ULPI transceiver generates VBUS internally and no external fault indicator present 1: External power source but no external fault indicator 2: External power source and external active high fault indicator 3: External power source and external active low fault indicator UTMI+ transceivers: 0: vbusvalid and drvvbus are both active low 1: vbusvalid is active low, drvvbus is active high 2: vbusvalid is active high, drvvbus is active low 3: vbusvalid and drvvbus are both active high	0 - 3	3
ramtest	When set each controller maps its internal buffers into the controller's register space.	0 - 1	0
urst_time	Amount of time (in ns) that the USB transceiver reset output need to be active	-	250
oepol	The polarity of the output enable signal for the data input/output buffers, 0 means active low and 1 means active high.	0 - 1	0

*see section 48.6 Transceiver support for more information

48.13 Signal descriptions

Table 579 shows the interface signals of the core (VHDL ports).

*Table 579.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
UCLK	N/A	Input	USB clock	-
RST	N/A	Input	Reset	Low
URSTI	N/A	Input	USB reset	Low
APBI	*	Input	APB slave input signals	-
EHC_APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBSI	*	Input	AHB slave input signals	-
EHC_AHBMO	*	Output	AHB master output signals.	-
UHC_AHBMO[]	*	Output	AHB master output vector.	
UHC_AHBSO[]	*	Output	AHB slave output vector.	-
O[]	xcvrsel[1:0]	Output	UTMI+ output signals	-
	termsel	Output	UTMI+ output signal	-
	suspendm	Output	UTMI+ output signal	Low
	opmode[1:0]	Output	UTMI+ output signals	-
	txvalid	Output	UTMI+ output signal	High
	drvvbus	Output	UTMI+ output signal	**
	datah[7:0]	Output	UTMI+ 16-bit interface output data bus	-
	validh	Output	UTMI+ 16-bit interface output signal	High
	host	Output	UTMI+ output signal	High
	stp	Output	ULPI output signal	High
	data[7:0]	Output	UTMI+/ULPI output signals	-
	utm_rst	Output	UTMI+/ULPI output signal	High/Low
	dcrtl	Output	Data bus direction control; High: Input data	-
I[]	linestate[1:0]	Input	UTMI+ input signals	-
	txready	Input	UTMI+ input signal	High
	rxvalid	Input	UTMI+ input signal	High
	rxactive	Input	UTMI+ input signal	High
	rxerror	Input	UTMI+ input signal	High
	vbusvalid	Input	UTMI+ input signal	**
	datah[7:0]	Input	UTMI+ 16-bit interface input data bus	-
	validh	Input	UTMI+ 16-bit interface input signal	High
	hostdisc	Input	UTMI+ input signal	High
	nxt	Input	ULPI input signal	High
	dir	Input	ULPI input signal	-
	data[7:0]	Input	UTMI+/ULPI input data bus	-

* See GRLIB IP Library User’s Manual

** Implementation dependent

48.14 Library dependencies

Table 580 shows the libraries used when instantiating the core (VHDL libraries).

*Table 580.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	GRUSBHC_PKG	Signals, component	Component declaration, USB signals

48.15 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.grusbhc_pkg.all;

-- USB Host controller with 2 ports (ULPI transceiver).One enhanced host controller
-- and two universal host controllers.

entity usbhc_ex is
  generic (
    tech    => tech;
    memtech => memtech;
    padtech => padtech);
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    -- USBHC signals
    usbh_clkin     : in std_ulogic;
    usbh_clkout    : out std_ulogic;
    usbh_d         : inout std_logic_vector(15 downto 0);
    usbh_reset     : out std_logic_vector(1 downto 0);
    usbh_nxt       : in std_logic_vector(1 downto 0);
    usbh_stp       : out std_logic_vector(1 downto 0);
    usbh_dir       : in std_logic_vector(1 downto 0)
    );
end;

architecture rtl of usbhc_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);

  -- USBHC signals
  signal usbhci : usbhc_in_vector(1 downto 0);
  signal usbhco : usbhc_out_vector(1 downto 0);
  signal uhclko, urst, uhclk : std_ulogic;

begin

  -- AMBA Components are instantiated here
  ...

  -- Instantiate pads, one iteration for each port
```

```
multi_pads: for i in 0 to 1 generate
    usbh_d_pad: iopadv
      generic map(tech => padtech, width => 8)
      port map (usbh_d((i*2**3+7) downto (i*2**3)), usbhco(i).data,
                usbhco(i).dctrl, usbhci(i).data);
    usbh_nxt_pad : inpad generic map (tech => padtech)
      port map (usbh_nxt(i),usbhci(i).nxt);
    usbh_dir_pad : inpad generic map (tech => padtech)
      port map (usbh_dir(i),usbhci(i).dir);
    usbh_reset_pad : outpad generic map (tech => padtech)
      port map (usbh_reset(i),usbhco(i).utm_rst);
    usbh_stp_pad : outpad generic map (tech => padtech)
      port map (usbh_stp(i),usbhco(i).stp);
  end generate;

  -- Clock generation. This example generates a clock to the
  -- transceivers. If the clock is genereated outside the FPGA
  -- no clock generator or clock out pads are necessary. When
  -- the clock is generated interally the ursti port of the core
  -- should be connected to lock signal from the DCM (in this case
  -- the urst signal makes that connection). When an external clock
  -- is used and no lock input is provided ursti should be set to ‘1’.

  usbh_clkgen : clkmul_virtex2
    generic map (clk_mul => 6, clk_div => 10)
    port map (’1’, lclk, uhclko, urst);

  -- Clock to transceivers
  usbh_clkout_pad : outpad
      generic map (tech => padtech)
      port map (usbh_clkout, uhclko);

  -- Input clock for USB clock domain. Note: arch => 3 creates
  -- a clock pad which removes input delay by instatiating a BUFGDLL
  -- or similar.
  usbh_clkin_pad :
    clkpad generic map (tech => padtech, arch => 3)
    port map(usbh_clkin, uhclk);

   usbhostcontroller0: grusbhc
    generic map (
      ehchindex => 5,
      ehcpindex => 14,
      ehcpaddr => 14,
      ehcpirq => 9,
      ehcpmask => 16#fff#,
      uhchindex => 6,
      uhchsindex => 3,
      uhchaddr => 16#A00#,
      uhchmask => 16#fff#,
      uhchirq => 10,
      tech => tech,
      memtech => memtech,
      nports => 2,
      ehcgen => 1,
      uhcgen => 1,
      n_cc => 2,
      n_pcc => 1,
      prr => 0,
      portroute1 => 0,
      portroute2 => 0,
      endian_conv => 1,
      be_regs => 0,
      be_desc => 0,
      utm_type => 2,
      uhcblo => 5,
      bwrd => 4,
      vbusconf => 3,
      ramtest => 0,
      urst_time => 250)
    port map (
        clkm,uhclk,rstn,urst,apbi,apbo(14),ahbmi,ahbsi,
        ahbmo(5),
        ahbmo(7 downto 6),
        ahbso(4 downto 3),
        usbhco,usbhci);
end;
```

49 I2CMST - I^{2}C-master

49.1 Overview

The I^{2}C-master core is a modified version of the OpenCores I^{2}C-Master where the WISHBONE interface has been replaced with an AMBA APB interface. The core is compatible with Philips I^{2}C standard and supports 7- and 10-bit addressing. Standard-mode (100 kb/s) and Fast-mode (400 kb/s) operation are supported directly. External pull-up resistors must be supplied for both bus lines.

As of the 1st of October 2006 Philips has declared all remaining patents under the I^{2}C program to be royalty free.

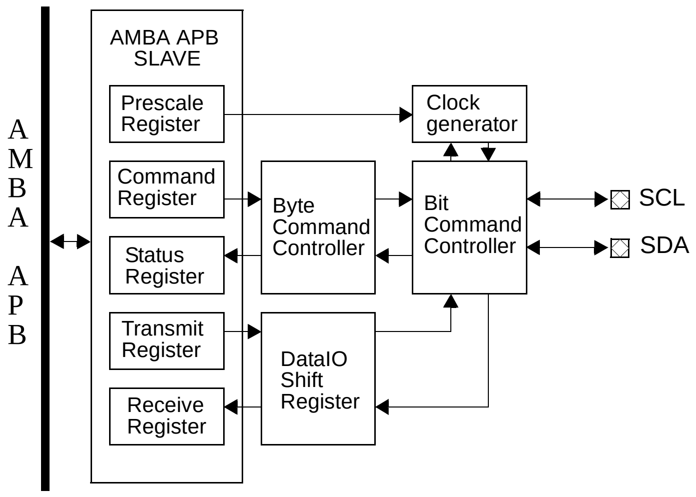


Figure 156. Block diagram

49.2 Operation

49.2.1 Transmission protocol

The I^{2}C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I^{2}C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I^{2}C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I^{2}C-bus specification and is dependent on the bus bit rate.

Figure 157 shows a data transfer taking place over the I^{2}C-bus. The master first generates a START condition and then transmits the 7-bit slave address. The bit following the slave address is the R/$\overline{\text{W}}$ bit which determines the direction of the data transfer. In this case the R/$\overline{\text{W}}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{\text{W}}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the first byte has been acknowledged the master transmits the data byte. If the R/$\overline{\text{W}}$ bit had been set to ‘1’ the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer. Section 49.2.3 contains three more example transfers from the perspective of a software driver.

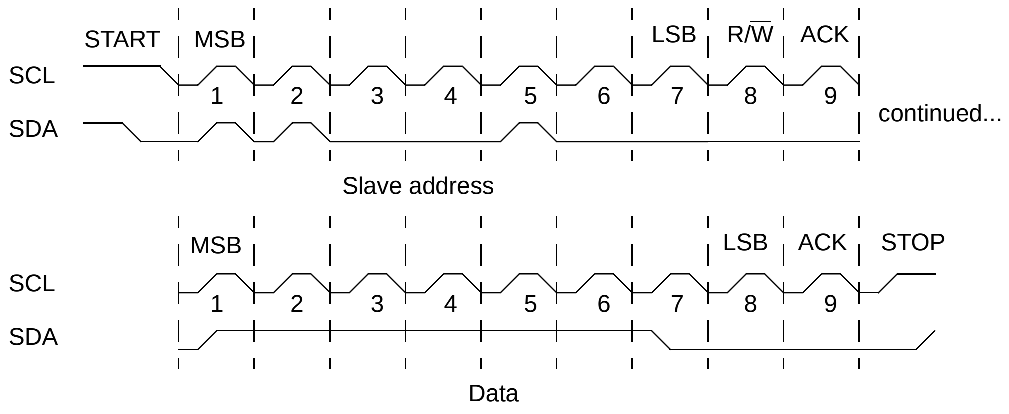


Figure 157. Complete I^{2}C data transfer

If the data bitrate is too high for a slave device, it may stretch the clock period by keeping SCL low after the master has driven SCL low.

49.2.2 Clock generation

The core uses the prescale register to determine the frequency of the SCL clock line and of the 5*SCL clock that the core uses internally. To calculate the prescale value use the formula:

$$Prescale = \frac{AMBAclockfrequency}{5 \cdot SCLfrequency} - 1$$

The *SCLfrequency* is 100 kHz for Standard-mode operation (100 kb/s) and 400 kHz for Fast mode operation. To use the core in Standard-mode in a system with a 60 MHz clock driving the AMBA bus the required prescale value is:

$$Prescale = \frac{60Mhz}{5 \cdot 100kHz} - 1 = 119 = 0x77$$

Note that the prescale register should only be changed when the core is disabled. The minimum recommended prescale value is 3 due to synchronization issues. Lower values may cause the master to violate I^{2}C timing requirements. This limits the minimum system frequency to 2 MHz for operation in Standard-mode.

49.2.3 Software operational model

The core is initialized by writing an appropriate value to the clock prescale register and then setting the enable (EN) bit in the control register. Interrupts are enabled via the interrupt enable (IEN) bit in the control register.

To write a byte to a slave the I^{2}C-master must generate a START condition and send the slave address with the $R/\overline{W}$ bit set to ‘0’. After the slave has acknowledged the address, the master transmits the data, waits for an acknowledge and generates a STOP condition. The sequence below instructs the core to perform a write:

1. Left-shift the I^{2}C-device address one position and write the result to the transmit register. The least significant bit of the transmit register ($R/\overline{W}$) is set to ‘0’.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt, or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the slave-data to the transmit register.

6. Send the data to the slave and generate a stop condition by setting STO and WR in the command register.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Verify that the slave has acknowledged the data by reading the RxACK bit in the status register. RxACK should not be set.

To read a byte from an I^2C-connected memory much of the sequence above is repeated. The data written in this case is the memory location on the I^2C slave. After the address has been written the master generates a repeated START condition and reads the data from the slave. The sequence that software should perform to read from a memory device:

1. Left-shift the I^2C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/W) is set to ‘0’.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the memory location to be read from the slave to the transmit register.

6. Set the WR bit in the command register. Note that a STOP condition is not generated here.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Read RxACK bit in the status register. RxACK should be low.

9. Address the I^2C-slave again by writing its left-shifted address into the transmit register. Set the least significant bit of the transmit register (R/W) to ‘1’ to read from the slave.

10. Set the STA and WR bits in the command register to generate a repeated START condition.

11. Wait for interrupt, or for TIP bit in the status register to go low.

12. Read RxACK bit in the status register. The slave should acknowledge the transfer.

13. Prepare to receive the data read from the I^2C-connected memory. Set bits RD, ACK and STO on the command register. Setting the ACK bit NAKs the received data and signifies the end of the transfer.

14. Wait for interrupt, or for TIP in the status register to go low.

15. The received data can now be read from the receive register.

To perform sequential reads the master can iterate over steps 13 - 15 by not setting the ACK and STO bits in step 13. To end the sequential reads the ACK and STO bits are set. Consult the documentation of the I^2C-slave to see if sequential reads are supported.

The final sequence illustrates how to write one byte to an I^2C-slave which requires addressing. First the slave is addressed and the memory location on the slave is transmitted. After the slave has acknowledged the memory location the data to be written is transmitted without a generating a new START condition:

1. Left-shift the I^2C-device address one position and write the result to the transmit register. The least significant bit of the transmit register (R/W) is set to ‘0’.

2. Generate START condition and send contents of transmit register by setting the STA and WR bits in the command register.

3. Wait for interrupt or for TIP bit in the status register to go low.

4. Read RxACK bit in status register. If RxACK is low the slave has acknowledged the transfer, proceed to step 5. If RxACK is set the device did not acknowledge the transfer, go to step 1.

5. Write the memory location to be written from the slave to the transmit register.

6. Set the WR bit in the command register.

7. Wait for interrupt, or for TIP bit in the status register to go low.

8. Read RxACK bit in the status register. RxACK should be low.

9. Write the data byte to the transmit register.

10. Set WR and STO in the command register to send the data byte and then generate a STOP condition.

11. Wait for interrupt, or for TIP bit in the status register to go low.

12. Check RxACK bit in the status register. If the write succeeded the slave should acknowledge the data byte transfer.

The example sequences presented here can be generally applied to I^{2}C-slaves. However, some devices may deviate from the protocol above, please consult the documentation of the I^{2}C-slave in question. Note that a software driver should also monitor the arbitration lost (AL) bit in the status register.

49.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 581.*I^{2}C-master registers

APB address offset	**Register**
0x00	Clock prescale register
0x04	Control register
0x08	Transmit register*
0x08	Receive register**
0x0C	Command register*
0x0C	Status register**

* Write only

** Read only

Table 582. I^{2}C-master Clock prescale register

31 16	15 7 6 5 4 3 2 1 0
RESERVED	Clock prescale

31 : 16	RESERVED
15:0	Clock prescale - Value is used to prescale the SCL clock line. Do not change the value of this register unless the EN field of the control register is set to ‘0’. The minimum recommended value of this register is 0x0003. Lower values may cause the master to violate I^{2}C timing requirements due to synchronization issues.

Table 583. I^{2}C-master control register

31 8	7	6	5 0
RESERVED	EN	IEN	RESERVED

Table 583. I²C-master control register

31 : 8	RESERVED
7	Enable (EN) - Enable I²C core. The core is enabled when this bit is set to ‘1’.
6	Interrupt enable (IEN) - When this bit is set to ‘1’ the core will generate interrupts upon transfer completion.
5:0	RESERVED

Table 584. I²C-master transmit register

31	8	7	1	0
RESERVED		TDATA		RW

31 : 8	RESERVED
7:1	Transmit data (TDATA) - Most significant bits of next byte to transmit via I²C
0	Read/Write (RW) - In a data transfer this is the data’s least significant bit. In a slave address transfer this is the RW bit. ‘1’ reads from the slave and ‘0’ writes to the slave.

Table 585. I²C-master receive register

31	8	7	0
RESERVED		RDATA	

31 : 8	RESERVED
7:0	Receive data (RDATA) - Last byte received over I²C-bus.

Table 586. I²C-master command register

31	8	7	6	5	4	3	2	1	0
RESERVED		STA	STO	RD	WR	ACK	RESERVED		IACK

31 : 8	RESERVED
7	Start (STA) - Generate START condition on I²C-bus. This bit is also used to generate repeated START conditions.
6	Stop (STO) - Generate STOP condition
5	Read (RD) - Read from slave
4	Write (WR) - Write to slave
3	Acknowledge (ACK) - Used when acting as a receiver. ‘0’ sends an ACK, ‘1’ sends a NACK.
2:1	RESERVED
0	Interrupt acknowledge (IACK) - Clears interrupt flag (IF) in status register.

Table 587. I²C-master status register

31	8	7	6	5	4	3	2	1	0
RESERVED		RxACK	BUSY	AL	RESERVED			TIP	IF

31 : 8	RESERVED
7	Receive acknowledge (RxACK) - Received acknowledge from slave. ‘1’ when no acknowledge is received, ‘0’ when slave has acked the transfer.
6	I²C-bus busy (BUSY) - This bit is set to ‘1’ when a start signal is detected and reset to ‘0’ when a stop signal is detected.
5	Arbitration lost (AL) - Set to ‘1’ when the core has lost arbitration. This happens when a stop signal is detected but not requested or when the master drives SDA high but SDA is low.
4:2	RESERVED

Table 587. I^{2}C-master status register

1	Transfer in progress (TIP) - ‘1’ when transferring data and ‘0’ when the transfer is complete.
0	Interrupt flag (IF) - This bit is set when a byte transfer has been completed and when arbitration is lost. If IEN in the control register is set an interrupt will be generated. New interrupts will ge generated even if this bit has not been cleared.

49.4 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x028. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

49.5 Configuration options

Table 588 shows the configuration options of the core (VHDL generics).

*Table 588.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by I^{2}C-master	0 - NAHBIRQ-1	0
oepol	Output enable polarity	0 - 1	0

49.6 Signal descriptions

Table 589 shows the interface signals of the core (VHDL ports).

*Table 589.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^{2}C clock line input	-
	SDA	Input	I^{2}C data line input	-
I2CO	SCL	Output	I^{2}C clock line output	-
	SCLOEN	Output	I^{2}C clock line output enable	Low
	SDA	Output	I^{2}C data line output	-
	SDAOEN	Output	I^{2}C data line output enable	Low

* see GRLIB IP Library User’s Manual

49.7 Library dependencies

Table 590 shows the libraries used when instantiating the core (VHDL libraries).

*Table 590.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component declaration, I2C signal definitions
OPENCORES	I2C	Component	Component declaration

49.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity i2c_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2c_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- I2C-master
  i2c0 : i2cmst
    generic map (pindex => 12, paddr => 12, pmask => 16#FFF#, pirq => 8)
    port map (rstn, clkm, apbi, apbo(12), i2ci, i2co);
  i2c_scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  i2c_sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
end;
```

50 I2CSLV - I^{2}C slave

50.1 Overview

The I^{2}C slave core is a simple I^{2}C slave that provides a link between the I^{2}C bus and the AMBA APB. The core is compatible with Philips I^{2}C standard and supports 7- and 10-bit addressing with an optionally software programmable address. Standard-mode (100 kb/s) and Fast-mode (400 kb/s) operation are supported directly. External pull-up resistors must be supplied for both bus lines.

As of the 1st of October 2006 NXP/Philips has declared all remaining patents under the I^{2}C program to be royalty free. NXP still provides the I^{2}C address allocation service, more information is available from: *http://www.nxp.com/products/interface_control/i2c/licensing/*

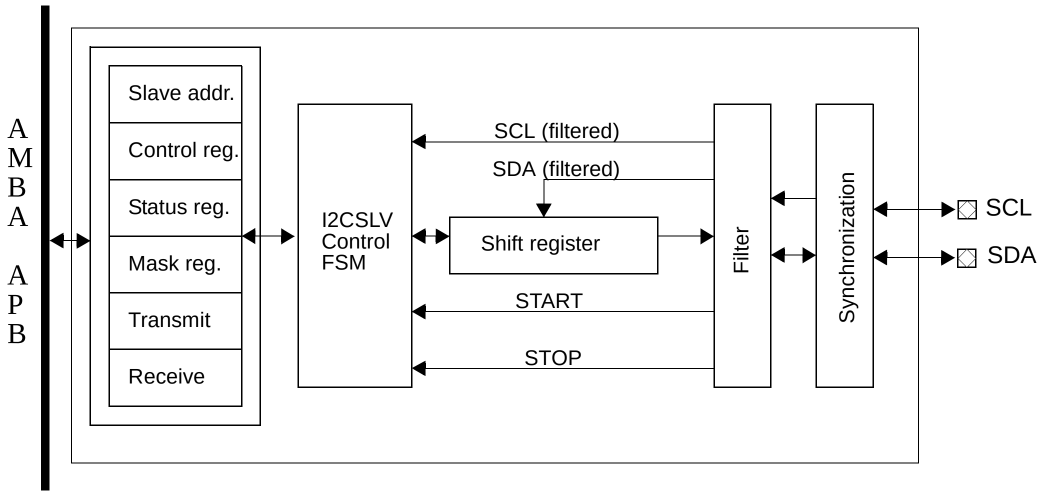


Figure 158. Block diagram

50.2 Operation

50.2.1 Transmission protocol

The I^{2}C-bus is a simple 2-wire serial multi-master bus with collision detection and arbitration. The bus consists of a serial data line (SDA) and a serial clock line (SCL). The I^{2}C standard defines three transmission speeds; Standard (100 kb/s), Fast (400 kb/s) and High speed (3.4 Mb/s).

A transfer on the I^{2}C-bus begins with a START condition. A START condition is defined as a high to low transition of the SDA line while SCL is high. Transfers end with a STOP condition, defined as a low to high transition of the SDA line while SCL is high. These conditions are always generated by a master. The bus is considered to be busy after the START condition and is free after a certain amount of time following a STOP condition. The bus free time required between a STOP and a START condition is defined in the I^{2}C-bus specification and is dependent on the bus bit rate.

Figure 159 shows a data transfer taking place over the I^{2}C-bus. The master first generates a START condition and then transmits the 7-bit slave address. I^{2}C also supports 10-bit addresses, which are discussed briefly below. The bit following the slave address is the R/$\overline{W}$ bit which determines the direction of the data transfer. In this case the R/$\overline{W}$ bit is zero indicating a write operation. After the master has transmitted the address and the R/$\overline{W}$ bit it releases the SDA line. The receiver pulls the SDA line low to acknowledge the transfer. If the receiver does not acknowledge the transfer, the master may generate a STOP condition to abort the transfer or start a new transfer by generating a repeated START condition.

After the address has been acknowledged the master transmits the data byte. If the R/$\overline{W}$ bit had been set to ‘1’ the master would have acted as a receiver during this phase of the transfer. After the data byte has been transferred the receiver acknowledges the byte and the master generates a STOP condition to complete the transfer.

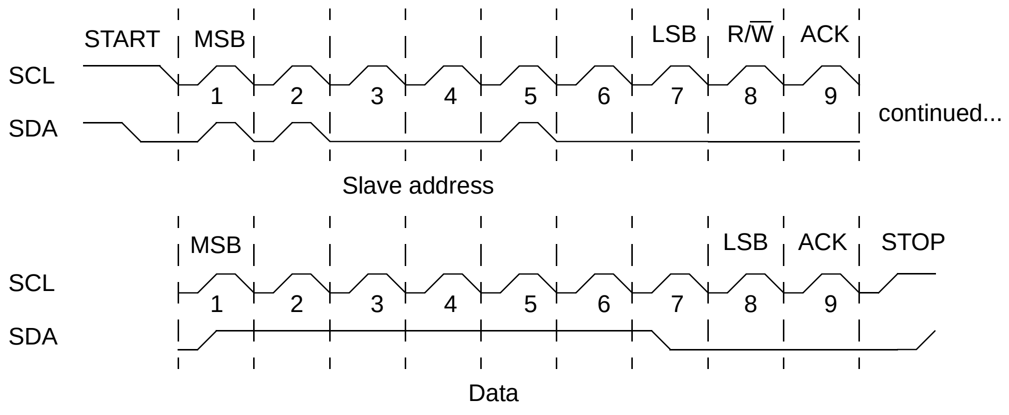


Figure 159. Complete I^{2}C data transfer

An I^{2}C slave may also support 10-bit addressing. In this case the master first transmits a pattern of five reserved bits followed by the two first bits of the 10-bit address and the R/$\overline{W}$ bit set to '0'. The next byte contains the remaining bits of the 10-bit address. If the transfer is a write operation the master then transmits data to the slave. To perform a read operation the master generates a repeated START condition and repeats the first part of the 10-bit address phase with the R/$\overline{W}$ bit set to '1'.

If the data bitrate is too high for a slave device or if the slave needs time to process data, it may stretch the clock period by keeping SCL low after the master has driven SCL low.

50.2.2 Slave addressing

The core's addressing support is implementation dependent. The core may have a programmable address and may support 10-bit addresses. If the core has support for 10-bit addressing, the TBA bit of the Slave address register will be set to '1' after reset. If the core's address is programmable this bit is writable and is used by the core to determine if it should listen to a 7- or 10-bit address.

Software can determine the addressing characteristics of the core by writing and reading the Slave address register. The core supports 10-bit addresses if the TBA bit is, or can be set, to '1'. The core has a software programmable address if the SLVADDR field in the same register can be changed.

50.2.3 System clock requirements and sampling

The core samples the incoming I^{2}C SCL clock and does not introduce any additional clock domains into the system. Both the SCL and SDA lines first pass through two stage synchronizers and are then filtered with a low pass filter consisting of four registers.

START and STOP conditions are detected if the SDA line, while SCL is high, is at one value for two system clock cycles, toggles and keeps the new level for two system clock cycles.

The synchronizers and filters constrain the minimum system frequency. The core requires the SCL signal to be stable for at least four system clock cycles before the core accepts the SCL value as the new clock value. The core's reaction to transitions will be additionally delayed since both lines are taken through two-stage synchronizers before they are filtered. Therefore it takes the core over eight system clock cycles to discover a transition on SCL. To use the slave in Standard-mode operation at 100 kHz the recommended minimum system frequency is 2 MHz. For Fast-mode operation at 400 kHz the recommended minimum system frequency is 6 MHz.

50.2.4 Operational model

The core has four main modes of operation and is configured to use one of these modes via the Control register bits Receive Mode (RMOD) and Transmit Mode (TMOD). The mode setting controls the core's behavior after a byte has been received or transmitted.

The core will always NAK a received byte if the receive register is full when the whole byte is received. If the receive register is free the value of RMOD determines if the core should continue to listen to the bus for the master's next action or if the core should drive SCL low to force the master into a wait state. If the value of the RMOD field is '0' the core will listen for the master's next action. If the value of the RMOD field is '1' the core will drive SCL low until the Receive register has been read and the Status register bit Byte Received (REC) has been cleared. Note that the core has not accepted a byte if it does not acknowledge the byte.

When the core receives a read request it evaluates the Transmit Valid (TV) bit in the Control register. If the Transmit Valid bit is set the core will acknowledge the address and proceed to transmit the data held in the Transmit register. After a byte has been transmitted the core assigns the value of the Control register bit Transmit Always Valid (TAV) to the Transmit Valid (TV) bit. This mechanism allows the same byte to be sent on all read requests without software intervention. The value of the Transmit Mode (TMOD) bit determines how the core acts after a byte has been transmitted and the master has acknowledged the byte, if the master NAKs the transmitted byte the transfer has ended and the core goes into an idle state. If TMOD is set to '0' when the master acknowledges a byte the core will continue to listen to the bus and wait for the master's next action. If the master continues with a sequential read operation the core will respond to all subsequent requests with the byte located in the Transmit Register. If TMOD is '1' the core will drive SCL low after a master has acknowledged the transmitted byte. SCL will be driven low until the Transmit Valid bit in the control register is set to '1'. Note that if the Transmit Always Valid (TAV) bit is set to '1' the Transmit Valid bit will immediately be set and the core will have show the same behavior for both Transmit modes.

When operating in Receive or Transmit Mode '1', the bus will be blocked by the core until software has acknowledged the transmitted or received byte. This may have a negative impact on bus performance and it also affects single byte transfers since the master is prevented to generate STOP or repeated START conditions when SCL is driven low by the core.

The core reports three types of events via the Status register. When the core NAKs a received byte, or its address in a read transfer, the NAK bit in the Status register will be set. When a byte is successfully received the core asserts the Byte Received (REC) bit. After transmission of a byte, the Byte Transmitted (TRA) bit is asserted. These three bits can be used as interrupt sources by setting the corresponding bits in the Mask register.

50.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 591.*I^{2}C slave registers

APB address offset	**Register**
0x00	Slave address register
0x04	Control register
0x08	Status register
0x0C	Mask register
0x10	Receive register
0x14	Transmit register

Table 592. Slave address register

31	30 ALEN	ALEN-1 0
TBA	RESERVED	SLVADDR

31	Ten-bit Address (TBA) - When this bit is set the core will interpret the value in the SLVADDR field as a 10-bit address. If the core has 10-bit address support this bit will have the reset value ‘1’.
30 : ALEN	RESERVED
(ALEN-1):0	Slave address (SLVADDR) - Contains the slave I2C address. The width of the slave address field, ALEN, is 7 bits (6:0) if the core only has support for 7-bit addresses. If the core has support for 10-bit addressing the width of SLVADDR is 10 bits. Depending on the hardware configuration this register may be read only. The core checks the length of the programmed address and will function with 7-bit addresses even if it has support for 10-bit addresses. I^{2}C addresses can be allocated by NXP, please see the link in the core’s overview section.

Reset value: Implementation dependent

Table 593. Control register

31 5	4	3	2	1	0
RESERVED	RMOD	TMOD	TV	TAV	EN

31 : 5	RESERVED
4	Receive Mode (RMOD) - Selects how the core handles writes: ‘0’: The slave accepts one byte and NAKs all other transfers until software has acknowledged the received byte by reading the Receive register. ‘1’: The slave accepts one byte and keeps SCL low until software has acknowledged the received byte by reading the Receive register.
3	Transmit Mode (TMOD) - Selects how the core handles reads: ‘0’: The slave transmits the same byte to all if the master requests more than one byte in the transfer. The slave then NAKs all read requests as long as the Transmit Valid (TV) bit is unset. ‘1’: The slave transmits one byte and then keeps SCL low until software has acknowledged that the byte has been transmitted by setting the Transmit Valid (TV) bit.
2	Transmit Valid (TV) - Software sets this bit to indicate that the data in the transmit register is valid. The core automatically resets this bit when the byte has been transmitted. When this bit is ‘0’ the core will either NAK or insert wait states on incoming read requests, depending on the Transmit Mode (TMOD).
1	Transmit Always Valid (TAV) - When this bit is set, the core will not clear the Transmit Valid (TV) bit when a byte has been transmitted.
0	Enable core (EN) - Enables core. When this bit is set to ‘1’ the core will react to requests to the address set in the Slave address register. If this bit is ‘0’ the core will keep both SCL and SDA inputs in Hi-Z state.

Reset value: 0b000000000000000000000000000UUUU0, where U is undefined.

Table 594. Status register

31 3	2	1	0
RESERVED	REC	TRA	NAK

31 : 3	RESERVED
2	Byte Received (REC) - This bit is set to ‘1’ when the core accepts a byte and is automatically cleared when the Receive register has been read.
1	Byte Transmitted (TRA) - This bit is set to ‘1’ when the core has transmitted a byte and is cleared by writing ‘1’ to this position. Writes of ‘0’ have no effect.
0	NAK Response (NAK) - This bit is set to ‘1’ when the core has responded with NAK to a read or write request. This bit does not get set to ‘1’ when the core responds with a NAK to an address that does not match the cores address. This bit is cleared by writing ‘1’ to this position, writes of ‘0’ have no effect.

Reset value: 0x00000000

Table 595. Mask register

31	3	2	1	0
RESERVED		RECE	TRAE	NAKE

31 : 3	RESERVED
2	Byte Received Enable (RECE) - When this bit is set the core will generate an interrupt when bit 2 in the Status register gets set.
1	Byte Transmitted Enable (TRAE) - When this bit is set the core will generate an interrupt when bit 1 in the Status register gets set.
0	NAK Response Enable (NAKE) - When this bit is set the core will generate an interrupt when bit 0 in the Status register gets set.

Reset value: Undefined

Table 596. Receive register

31	8	7	0
RESERVED		RECBYTE	

31 : 8	RESERVED
7:0	Received Byte (RECBYTE) - Last byte received from master. This field only contains valid data if the Byte received (REC) bit in the status register has been set.

Reset value: Undefined

Table 597. Transmit register

31	8	8	7	0
RESERVED			TRABYTE	

31 : 8	RESERVED
7:0	Transmit Byte (TRABYTE) - Byte to transmit on the next master read request.

Reset value: Undefined

50.4 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x03E. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

50.5 Configuration options

Table 598 shows the configuration options of the core (VHDL generics).

*Table 598.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by I^{2}C slave	0 - NAHBIRQ-1	0
hardaddr	If this generic is set to 1 the core uses the value of generic i2caddr as the hard coded address. If hardaddr is set to 0 the core’s address can be changed via the Slave address register.	0 - 1	0
tenbit	If this generic is set to 1 the core will support 10-bit addresses. Note that the core can still be configured to use a 7-bit address.	0 - 1	0
i2caddr	The slave’s (initial) I^{2}C address.	0 - 1023	0
oepol	Output enable polarity	0 - 1	0

50.6 Signal descriptions

Table 599 shows the interface signals of the core (VHDL ports).

*Table 599.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
I2CI	SCL	Input	I^{2}C clock line input	-
	SDA	Input	I^{2}C data line input	-
I2CO	SCL	Output	I^{2}C clock line output	-
	SCLOEN	Output	I^{2}C clock line output enable	Low
	SDA	Output	I^{2}C data line output	-
	SDAOEN	Output	I^{2}C data line output enable	Low

* see GRLIB IP Library User’s Manual

50.7 Library dependencies

Table 600 shows the libraries used when instantiating the core (VHDL libraries).

*Table 600.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component declaration, I2C signal definitions

50.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity i2cslv_ex is
  port (
    clk  : in std_ulogic;
    rstn : in std_ulogic;

    -- I2C signals
    iic_scl : inout std_ulogic;
    iic_sda : inout std_ulogic
    );
end;

architecture rtl of i2cslv_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  -- I2C signals
  signal i2ci : i2c_in_type;
  signal i2co : i2c_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- I2C-slave
  i2cslv0 : i2cslv
    generic map (pindex => 1, paddr => 1, pmask => 16#FFF#, pirq => 1,
                 hardaddr => 0, tenbit => 1, i2caddr => 16#50#)
    port map (rstn, clk, apbi, apbo(1), i2ci, i2co);
  i2cslv0_scl_pad : iopad generic map (tech => padtech)
    port map (iic_scl, i2co.scl, i2co.scloen, i2ci.scl);
  i2cslv0_sda_pad : iopad generic map (tech => padtech)
    port map (iic_sda, i2co.sda, i2co.sdaoen, i2ci.sda);
end;
```

51 IRQMP - Multiprocessor Interrupt Controller

51.1 Overview

The AMBA system in GRLIB provides an interrupt scheme where interrupt lines are routed together with the remaining AHB/APB bus signals, forming an interrupt bus. Interrupts from AHB and APB units are routed through the bus, combined together, and propagated back to all units. The multiprocessor interrupt controller core is attached to AMBA bus as an APB slave, and monitors the combined interrupt signals.

The interrupts generated on the interrupt bus are all forwarded to the interrupt controller. The interrupt controller prioritizes, masks and propagates the interrupt with the highest priority to the processor. In multiprocessor systems, the interrupts are propagated to all processors.

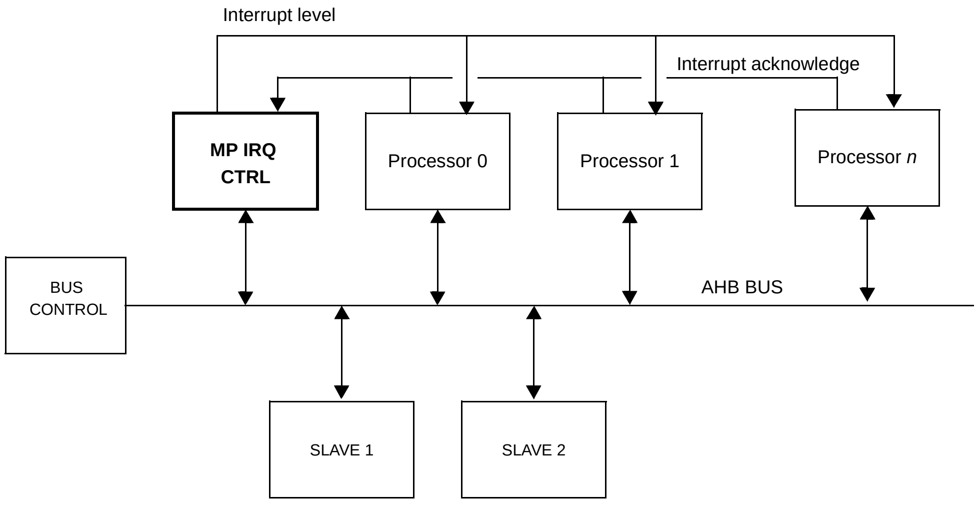


Figure 160. LEON3 multiprocessor system with Multiprocessor Interrupt controller

51.2 Operation

51.2.1 Interrupt prioritization

The interrupt controller monitors interrupt 1 - 15 of the interrupt bus (APBI.PIRQ[15:1]). When any of these lines are asserted high, the corresponding bit in the interrupt pending register is set. The pending bits will stay set even if the PIRQ line is de-asserted, until cleared by software or by an interrupt acknowledge from the processor.

Each interrupt can be assigned to one of two levels (0 or 1) 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 processor. If no unmasked pending interrupt exists on level 1, then the highest unmasked interrupt from level 0 will be forwarded. PIRQ[31:16] are not used by the IRQMP core.

Interrupts are prioritised at system level, while masking and forwarding of interrupts in done for each processor separately. Each processor in an multiprocessor system has separate interrupt mask and force registers. When an interrupt is signalled on the interrupt bus, the interrupt controller will prioritize interrupts, perform interrupt masking for each processor according to the mask in the corresponding mask register and forward the interrupts to the processors.

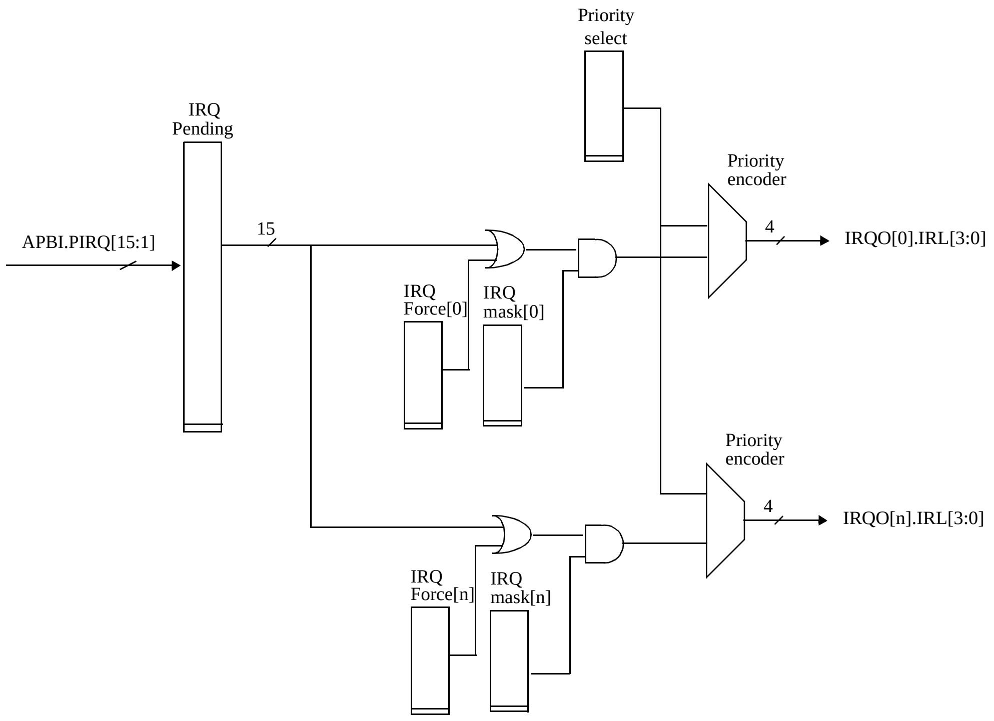


Figure 161. Interrupt controller block diagram

When a processor 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 processor 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. Note that interrupt 15 cannot be maskable by the LEON3 processor and should be used with care - most operating systems do not safely handle this interrupt.

51.2.2 Processor status monitoring

The processor status can be monitored through the Multiprocessor Status Register. The STATUS field in this register indicates if a processor is halted (‘1’) or running (‘0’). A halted processor can be reset and restarted by writing a ‘1’ to its status field. After reset, all processors except processor 0 are halted. When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits.

51.2.3 Irq broadcasting

The Broadcast Register is activated when the generic *ncpu* is > 1. A incoming irq that has its bit set in the Broadcast Register is propagated to the force register of *all* CPUs rather than only to the Pending Register. This can be used to implement a timer that fires to all cpus with that same irq.

51.3 Registers

The core is controlled through registers mapped into APB address space. The number of implemented registers depend on number of processor in the multiprocessor system.

*Table 601.*Interrupt Controller registers

APB address offset	Register
0x00	Interrupt level register
0x04	Interrupt pending register
0x08	Interrupt force register (NCPU = 0)
0x0C	Interrupt clear register
0x10	Multiprocessor status register
0x14	Broadcast register
0x40	Processor interrupt mask register
0x44	Processor 1 interrupt mask register
0x40 + 4 * *n*	Processor *n* interrupt mask register
0x80	Processor interrupt force register
0x84	Processor 1 interrupt force register
0x80 + 4 * *n*	Processor *n* interrupt force register
0xC4	Processor 1 extended interrupt identification register
0xC0 + 4 * *n*	Processor *n* extended interrupt identification register

51.3.1 Interrupt level register

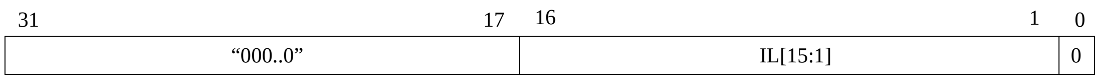


Figure 162. Interrupt level register

[31:16] Reserved.
[15:1] Interrupt Level *n* (IL[*n*]): Interrupt level for interrupt *n*.
[0] Reserved.

51.3.2 Interrupt pending register

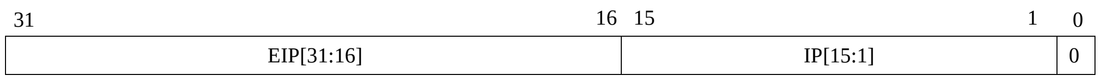


Figure 163. Interrupt pending register

[31:17] Extended Interrupt Pending n (EIP[n]).
[15:1] Interrupt Pending n (IP[n]): Interrupt pending for interrupt n.
[0] Reserved

51.3.3 Interrupt force register (NCPU = 0)

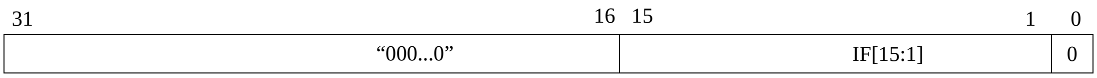


Figure 164. Interrupt force register

[31:16] Reserved.
[15:1] Interrupt Force *n* (IF[*n*]): Force interrupt nr *n*.
[0] Reserved.

51.3.4 Interrupt clear register

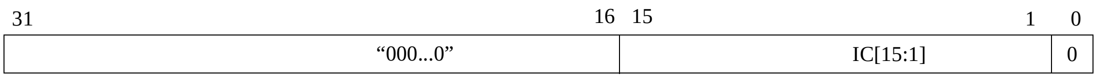


Figure 165. Interrupt clear register

[31:16] Reserved.
[15:1] Interrupt Clear *n* (IC[*n*]): Writing ‘1’ to IC*n* will clear interrupt *n*.
[0] Reserved.

51.3.5 Multiprocessor status register

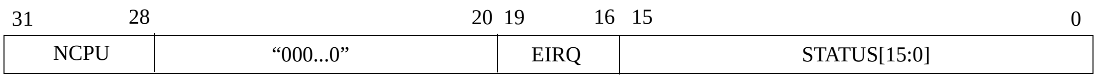


Figure 166. Multiprocessor status register

[31:28] NCPU. Number of CPU’s in the system -1 .
[19:16] EIRQ. Interrupt number (1 - 15) used for extended interrupts. Fixed to 0 if extended interrupts are disabled.
[15:1] Power-down status of CPU [*n*]: ‘1’ = power-down, ‘0’ = running. Write with ‘1’ to start processor *n*.

51.3.6 Processor interrupt mask register

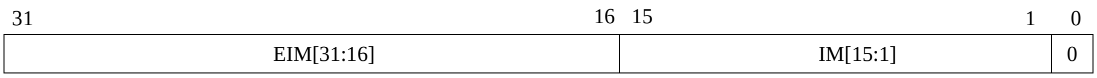


Figure 167. Processor interrupt mask register

[31:16] Interrupt mask for extended interrupts
[15:1] Interrupt Mask *n* (IM[*n*]): If IMn = 0 the interrupt *n* is masked, otherwise it is enabled.
[0] Reserved.

51.3.7 Broadcast register (NCPU > 1)

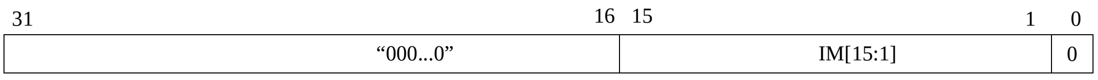


Figure 168. Processor interrupt mask register

[31:16]	Reserved.
[15:1]	Broadcast Mask *n* (BM[*n*]): If BMn = 1 the interrupt *n* is broadcasted (written to the Force Register of all CPUs), otherwise standard semantic applies (Pending Register).
[0]	Reserved.

51.3.8 Processor interrupt force register (NCPU > 0)

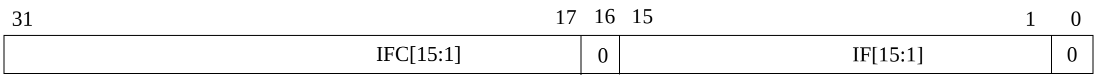


Figure 169. Processor interrupt force register

[31:17]	Interrupt force clear *n* (IFC[*n*]).
[15:1]	Interrupt Force *n* (IF[*n*]): Force interrupt nr *n*.
[0]	Reserved.

51.3.9 Extended interrupt identification register

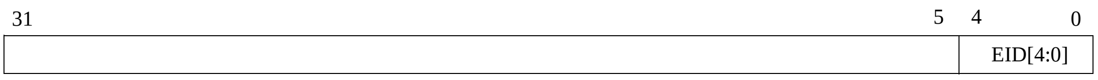


Figure 170. Processor interrupt force register

[4:0]	ID (16 - 31) of the acknowledged extended interrupt

51.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x00D. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

51.5 Configuration options

Table 602 shows the configuration options of the core (VHDL generics).

*Table 602.*Configuration options

Generic	Function	Allowed range	Default
pindex	Selects which APB select signal (PSEL) will be used to access the timer unit	0 to NAPBMAX-1	0
paddr	The 12-bit MSB APB address	0 to 4095	0
pmask	The APB address mask	0 to 4095	4095
ncpu	Number of processors in multiprocessor system	1 to 16	1

51.6 Signal descriptions

Table 603 shows the interface signals of the core (VHDL ports).

*Table 603.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
IRQI[n]	INTACK	Input	Processor *n* Interrupt acknowledge	High
	IRL[3:0]		Processor *n* interrupt level	High
IRQO[n]	IRL[3:0]	Output	Processor *n* Input interrupt level	High
	RST		Reset power-down and error mode of processor *n*	High
	RUN		Start processor *n* after reset (SMP systems only)	High

* see GRLIB IP Library User’s Manual

51.7 Library dependencies

Table 604 shows libraries that should be used when instantiating the core.

*Table 604.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	LEON3	Signals, component	Signals and component declaration

51.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

entity irqmp_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ...  -- other signals
    );
end;

architecture rtl of irqmp_ex is
  constant NCPU : integer := 4;

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);
  signal ahbsi : ahb_slv_in_type;
  -- GP Timer Unit input signals
  signal irqi   : irq_in_vector(0 to NCPU-1);
  signal irqo   : irq_out_vector(0 to NCPU-1);

  -- LEON3 signals
  signal leon3i : l3_in_vector(0 to NCPU-1);
  signal leon3o : l3_out_vector(0 to NCPU-1);

begin

  -- 4 LEON3 processors are instantiated here
  cpu : for i in 0 to NCPU-1 generate
    u0 : leon3s generic map (hindex => i)
    port map (clk, rstn, ahbmi, ahbmo(i), ahbsi,
irqi(i), irqo(i), dbgi(i), dbgo(i));
  end generate;

  -- MP IRQ controller
  irqctrl0 : irqmp
  generic map (pindex => 2, paddr => 2, ncpu => NCPU)
  port map (rstn, clk, apbi, apbo(2), irqi, irqo);
end
```

52 LEON3 - High-performance SPARC V8 32-bit Processor

52.1 Overview

LEON3 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON3 core has the following main features: 7-stage pipeline with Harvard architecture, separate instruction and data caches, hardware multiplier and divider, on-chip debug support and multi-processor extensions.

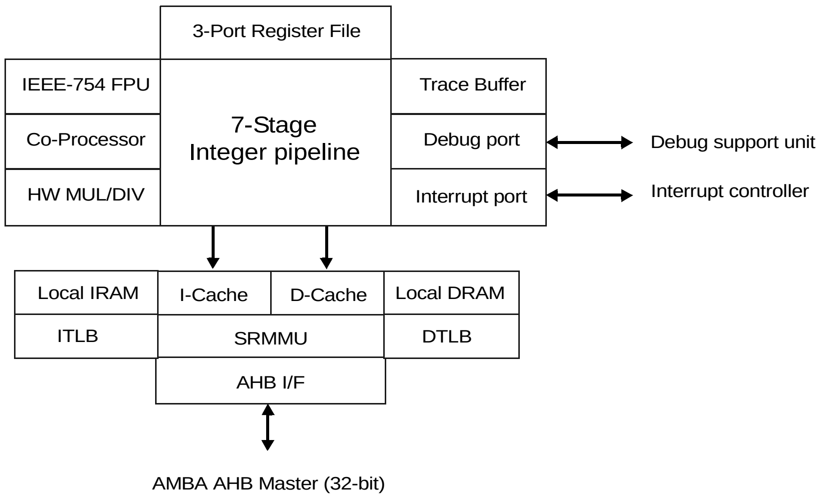


Figure 171. LEON3 processor core block diagram

Note: this manual describes the full functionality of the LEON3 core. Through the use of VHDL generics, parts of the described functionality can be suppressed or modified to generate a smaller or faster implementation.

52.1.1 Integer unit

The LEON3 integer unit implements the full SPARC V8 standard, including hardware multiply and divide instructions. The number of register windows is configurable within the limit of the SPARC standard (2 - 32), with a default setting of 8. The pipeline consists of 7 stages with a separate instruction and data cache interface (Harvard architecture).

52.1.2 Cache sub-system

LEON3 has a highly configurable cache system, consisting of a separate instruction and data cache. Both caches can be configured with 1 - 4 sets, 1 - 256 kbyte/set, 16 or 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 minimize refill latency. The data cache uses write-through policy and implements a double-word write-buffer. The data cache can also perform bus-snooping on the AHB bus. A local scratch pad ram can be added to both the instruction and data cache controllers to allow 0-waitstates access memory without data write back.

52.1.3 Floating-point unit and co-processor

The LEON3 integer unit provides interfaces for a floating-point unit (FPU), and a custom co-processor. Two FPU controllers are available, one for the high-performance GRFPU (available from Gaisler Research) and one for the Meiko FPU core (available from Sun Microsystems). The floating-point processors and co-processor execute in parallel with the integer unit, and does not block the operation unless a data or resource dependency exists.

52.1.4 Memory management unit

A SPARC V8 Reference Memory Management Unit (SRMMU) can optionally be enabled. The SRMMU implements the full SPARC V8 MMU specification, and provides mapping between multiple 32-bit virtual address spaces and 36-bit physical memory. A three-level hardware table-walk is implemented, and the MMU can be configured to up to 64 fully associative TLB entries.

52.1.5 On-chip debug support

The LEON3 pipeline includes functionality to allow non-intrusive debugging on target hardware. To aid software debugging, up to four watchpoint registers can be enabled. Each register can cause a breakpoint trap on an arbitrary instruction or data address range. When the (optional) debug support unit is attached, the watchpoints can be used to enter debug mode. Through a debug support interface, full access to all processor registers and caches is provided. The debug interfaces also allows single stepping, instruction tracing and hardware breakpoint/watchpoint control. An internal trace buffer can monitor and store executed instructions, which can later be read out over the debug interface.

52.1.6 Interrupt interface

LEON3 supports the SPARC V8 interrupt model with a total of 15 asynchronous interrupts. The interrupt interface provides functionality to both generate and acknowledge interrupts.

52.1.7 AMBA interface

The cache system implements an AMBA AHB master to load and store data to/from the caches. The interface is compliant with the AMBA-2.0 standard. During line refill, incremental burst are generated to optimise the data transfer.

52.1.8 Power-down mode

The LEON3 processor core implements a power-down mode, which halts the pipeline and caches until the next interrupt. This is an efficient way to minimize power-consumption when the application is idle, and does not require tool-specific support in form of clock gating. To implement clock-gating, a suitable clock-enable signal is produced by the processor.

52.1.9 Multi-processor support

LEON3 is designed to be use in multi-processor systems. Each processor has a unique index to allow processor enumeration. The write-through caches and snooping mechanism guarantees memory coherency in shared-memory systems.

52.1.10 Performance

Using 8K + 8K caches and a 16x16 multiplier, the dhrystone 2.1 benchmark reports 1,500 iteration/s/MHz using the gcc-3.4.4 compiler (-O2). This translates to 0.85 dhrystone MIPS/MHz using the VAX 11/780 value a reference for one MIPS.

52.2 LEON3 integer unit

52.2.1 Overview

The LEON3 integer unit implements the integer part of the SPARC V8 instruction set. The implementation is focused on high performance and low complexity. The LEON3 integer unit has the following main features:

- 7-stage instruction pipeline
- Separate instruction and data cache interface
- Support for 2 - 32 register windows
- Hardware multiplier with optional 16x16 bit MAC and 40-bit accumulator
- Radix-2 divider (non-restoring)
- Single-vector trapping for reduced code size

Figure 172 shows a block diagram of the integer unit.

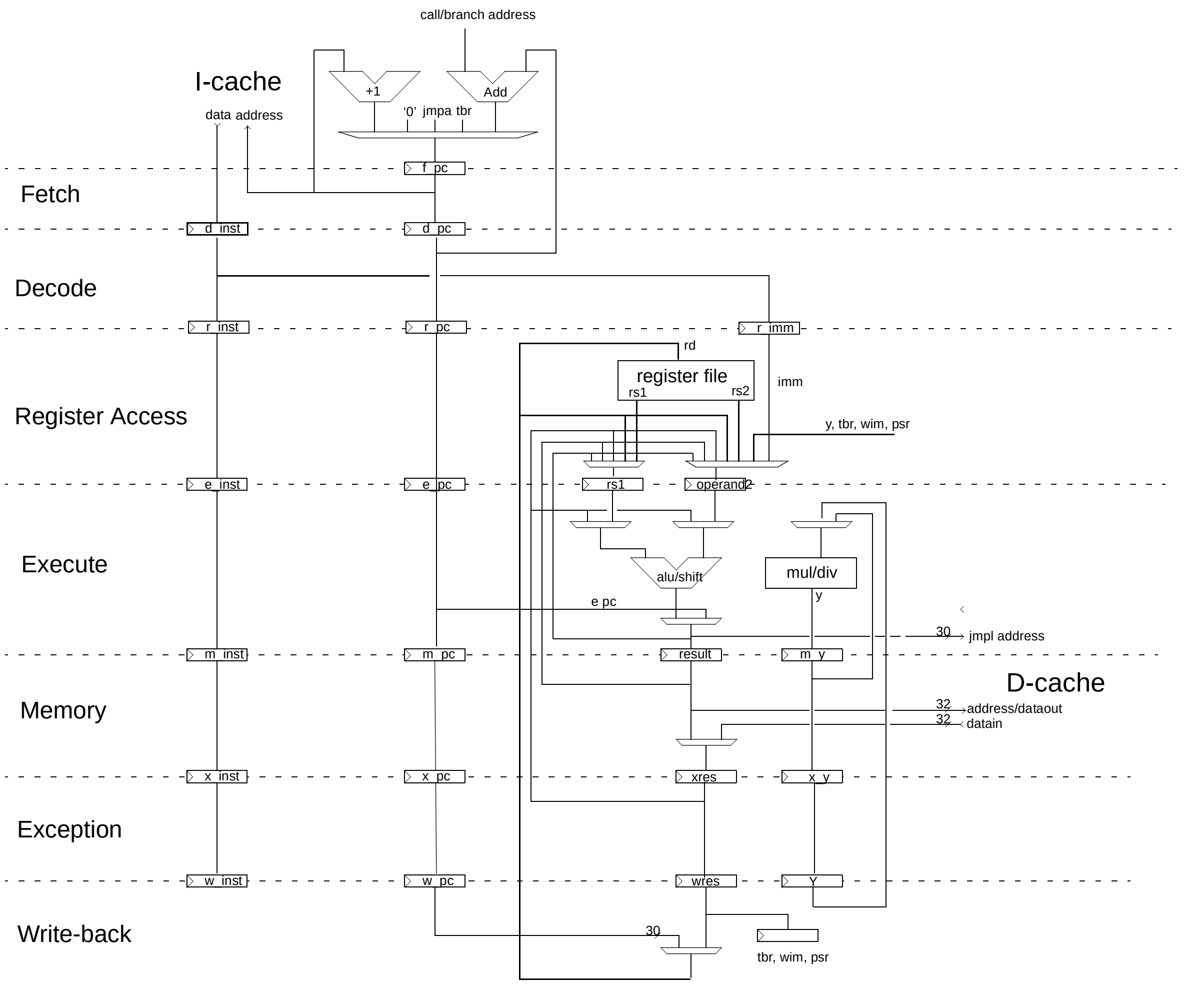


Figure 172. LEON3 integer unit datapath diagram

52.2.2 Instruction pipeline

The LEON integer unit uses a single instruction issue pipeline with 7 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 CALL and Branch target addresses are generated.

3. RA (Register access): Operands are read from the register file or from internal data bypasses.

4. EX (Execute): ALU, logical, and shift operations are performed. For memory operations (e.g., LD) and for JMPL/RETT, the address is generated.

5. ME (Memory): Data cache is accessed. Store data read out in the execution stage is written to the data cache at this time.

6. XC (Exception) Traps and interrupts are resolved. For cache reads, the data is aligned as appropriate.

7. WR (Write): The result of any ALU, logical, shift, or cache operations are written back to the register file.

Table 605 lists the cycles per instruction (assuming cache hit and no icc or load interlock):

*Table 605.*Instruction timing

Instruction	**Cycles (MMU disabled)**	**Cycles (MMU fast-write)**	**Cycles (MMU slow-write)**
JMPL, RETT	3	3	3
Double load	2	2	2
Single store	2	2	4
Double store	3	3	5
SMUL/UMUL	1/4*	1/4*	1/4*
SDIV/UDIV	35	35	35
Taken Trap	5	5	5
Atomic load/store	3	3	5
All other instructions	**1**	**1**	**1**

* Multiplication cycle count is 1 clock for the 32x32 multiplier and 4 clocks for the 16x16 version.

52.2.3 SPARC Implementor’s ID

Gaisler Research is assigned number 15 (0xF) as SPARC implementor’s identification. This value is hard-coded into bits 31:28 in the %psr register. The version number for LEON3 is 3, which is hard-coded in to bits 27:24 of the %psr.

52.2.4 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.

52.2.5 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. The multiply instructions are performed using a 32x32 pipelined hardware multiplier, or a 16x16 hardware multiplier which is iterated four times. To improve the timing, the 16x16 multiplier can optionally be provided with a pipeline stage.

52.2.6 Multiply and accumulate instructions

To accelerate DSP algorithms, two multiply&accumulate instructions are implemented: UMAC and SMAC. The UMAC performs an unsigned 16-bit multiply, producing a 32-bit result, and adds the result to a 40-bit accumulator made up by the 8 lsb bits from the %y register and the %asr18 register. The least significant 32 bits are also written to the destination register. SMAC works similarly but performs signed multiply and accumulate. The MAC instructions execute in one clock but have two clocks latency, meaning that one pipeline stall cycle will be inserted if the following instruction uses the destination register of the MAC as a source operand.

Assembler syntax:

```
umacrs1, reg_imm, rd
smacrs1, reg_imm, rd
```

Operation:

```
prod[31:0] = rs1[15:0] * reg_imm[15:0]
result[39:0] = (Y[7:0] & %asr18[31:0]) + prod[31:0]
(Y[7:0] & %asr18[31:0]) = result[39:0]
rd = result[31:0]
```

%asr18 can be read and written using the RDASR and WRASR instructions.

52.2.7 Hardware breakpoints

The integer unit can be configured to include up to four hardware breakpoints. Each breakpoint consists of a pair of application-specific registers (%asr24/25, %asr26/27, %asr28/29 and %asr30/31) registers; one with the break address and one with a mask:

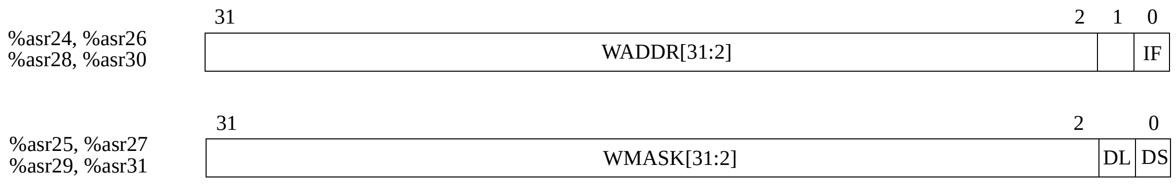


Figure 173. Watch-point registers

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 breakpoint 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 breakpoint function.

52.2.8 Instruction trace buffer

The instruction trace buffer consists of a circular buffer that stores executed instructions. The trace buffer operation is controlled through the debug support interface, and does not affect processor operation (see the DSU description). The size of the trace buffer is configurable from 1 to 64 kB through a VHDL generic. The trace buffer is 128 bits wide, and stores the following information:

- Instruction address and opcode
- Instruction result
- Load/store data and address
- Trap information
- 30-bit time tag

The operation and control of the trace buffer is further described in section 28.4. Note that in multi-processor systems, each processor has its own trace buffer allowing simultaneous tracing of all instruction streams.

52.2.9 Processor configuration register

The application specific register 17 (%asr17) provides information on how various configuration options were set during synthesis. This can be used to enhance the performance of software, or to support enumeration in multi-processor systems. The register can be accessed through the RDASR instruction, and has the following layout:

	31 28		17	16 15	14	13	12	11 10	9	8	7 5	4 0
%asr17	INDEX	RESERVED	CS	CF	DW	SV	LD	FPU	M	V8	NWP	NWIN

Figure 174. LEON3 configuration register (%asr17)

Field Definitions:

[31:28]:	Processor index. In multi-processor systems, each LEON core gets a unique index to support enumeration. The value in this field is identical to the *hindex* generic parameter in the VHDL model.

value in this field is identical to the *hindex* generic parameter in the VHDL model.

[17]:	Clock switching enabled (CS). If set switching between AHB and CPU frequency is available.
[16:15]:	CPU clock frequency (CF). CPU core runs at (CF+1) times AHB frequency.
[14]:	Disable write error trap (DWT). When set, a write error trap (tt = 0x2b) will be ignored. Set to zero after reset.
[13]:	Single-vector trapping (SVT) enable. If set, will enable single-vector trapping. Fixed to zero if SVT is not implemented. Set to zero after reset.
[12]:	Load delay. If set, the pipeline uses a 2-cycle load delay. Otherwise, a 1-cycle load delay i s used. Generated from the *lddel* generic parameter in the VHDL model.
[11:10]:	FPU option. “00” = no FPU; “01” = GRFPU; “10” = Meiko FPU, “11” = GRFPU-Lite
[9]:	If set, the optional multiply-accumulate (MAC) instruction is available
[8]:	If set, the SPARC V8 multiply and divide instructions are available.
[7:5]:	Number of implemented watchpoints (0 - 4)
[4:0]:	Number of implemented registers windows corresponds to NWIN+1.

52.2.10 Exceptions

LEON adheres to the general SPARC trap model. The table below shows the implemented traps and their individual priority. When PSR (processor status register) bit ET=0, an exception trap causes the processor to halt execution and enter error mode, and the external error signal will then be asserted.

*Table 606.*Trap allocation and 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	Hardware breakpoint match
window_overflow	0x05	8	SAVE into invalid window
window_underflow	0x06	8	RESTORE into invalid window
register_hadrware_error	0x20	9	register file EDAC error (LEON-FT only)
mem_address_not_aligned	0x07	10	Memory access to un-aligned address
fp_exception	0x08	11	FPU exception
cp_exception	0x28	11	Co-processor 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
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)

52.2.11 Single vector trapping (SVT)

Single-vector trapping (SVT) is an SPARC V8e option to reduce code size for embedded applications. When enabled, any taken trap will always jump to the reset trap handler (%tbr.tba + 0). The trap type will be indicated in %tbr.tt, and must be decoded by the shared trap handler. SVT is enabled by setting bit 13 in %asr17. The model must also be configured with the SVT generic = 1.

52.2.12 Address space identifiers (ASI)

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 LEON3 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[5:0] are used for the mapping, ASI[7:6] have no influence on operation.

*Table 607.*ASI usage

ASI	**Usage**
0x01	Forced cache miss
0x02	System control registers (cache control register)
0x08, 0x09, 0x0A, 0x0B	Normal cached access (replace if cacheable)
0x0C	Instruction cache tags
0x0D	Instruction cache data
0x0E	Data cache tags
0x0F	Data cache data
0x10	Flush instruction cache
0x11	Flush data cache

52.2.13 Power-down

The processor can be configured to include a power-down feature to minimize power consumption during idle periods. The power-down mode is entered by performing a WRASR instruction to %asr19:

```
wr %g0, %asr19
```

During power-down, the pipeline is halted until the next interrupt occurs. Signals inside the processor pipeline and caches are then static, reducing power consumption from dynamic switching.

52.2.14 Processor reset operation

The processor is reset by asserting the RESET input for at least 4 clock cycles. 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).

*Table 608.*Processor reset values

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

By default, the execution will start from address 0. This can be overridden by setting the RSTADDR generic in the model to a non-zero value. The reset address is always aligned on a 4 kbyte boundary. If RSTADDR is set to 16#FFFFF#, then the reset address is taken from the signal IRQI.RSTVEC. This allows the reset address to be changed dynamically.

52.2.15 Multi-processor support

The LEON3 processor support synchronous multi-processing (SMP) configurations, with up to 16 processors attached to the same AHB bus. In multi-processor systems, only the first processor will start. All other processors will remain halted in power-down mode. After the system has been initialized, the remaining processors can be started by writing to the 'MP status register', located in the multi-processor interrupt controller. The halted processors start executing from the reset address (0 or RSTADDR generic). Enabling SMP is done by setting the *smp* generic to 1 or higher. Cache snooping should always be enabled in SMP systems to maintain data cache coherency between the processors.

52.2.16 Cache sub-system

The LEON3 processor implements a Harvard architecture with separate instruction and data buses, connected to two independent cache controllers. Both instruction and data cache controllers can be separately configured to implement a direct-mapped cache or a multi-set cache with set associativity of 2 - 4. The set size is configurable to 1 - 256 kbyte, divided into cache lines with 16 or 32 bytes of data. In multi-set configurations, one of three replacement policies can be selected: least-recently-used (LRU), least-recently-replaced (LRR) or (pseudo-) random. If the LRR algorithm can only be used when the cache is 2-way associative. A cache line can be locked in the instruction or data cache preventing it from being replaced by the replacement algorithm.

NOTE: The LRR algorithm uses one extra bit in tag rams to store replacement history. The LRU algorithm needs extra flip-flops per cache line to store access history. The random replacement algorithm is implemented through modulo-N counter that selects which line to evict on cache miss.

Cachability for both caches is controlled through the AHB plug&play address information. The memory mapping for each AHB slave indicates whether the area is cachable, and this information is used to (statically) determine which access will be treated as cacheable. This approach means that the cachability mapping is always coherent with the current AHB configuration. The AMBA plug&play cachability can be overriden using the CACHED generic. When this generic is not zero, it is treated as a 16-bit field, defining the cachability of each 256 Mbyte address block on the AMBA bus. A value of 16#00F3# will thus define cachable areas in 0 - 0x20000000 and 0x40000000 - 0x80000000..

52.3 Instruction cache

52.3.1 Operation

The instruction cache can be configured as a direct-mapped cache or as a multi-set cache with associativity of 2 - 4 implementing either LRU or random replacement policy or as 2-way associative cache implementing LRR algorithm. The set size is configurable to 1 - 64 kbyte and divided into cache lines of 16- 32 bytes. Each line has a cache tag associated with it consisting of a tag field, valid field with one valid bit for each 4-byte sub-block and optional LRR and lock bits. On an instruction cache miss to a cachable location, the instruction is fetched and the corresponding tag and data line updated. In a multi-set configuration a line to be replaced is chosen according to the replacement policy.

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. During cache line refill, incremental burst are generated on the AHB bus.

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.

52.3.2 Instruction cache tag

A instruction cache tag entry consists of several fields as shown in figure 175:

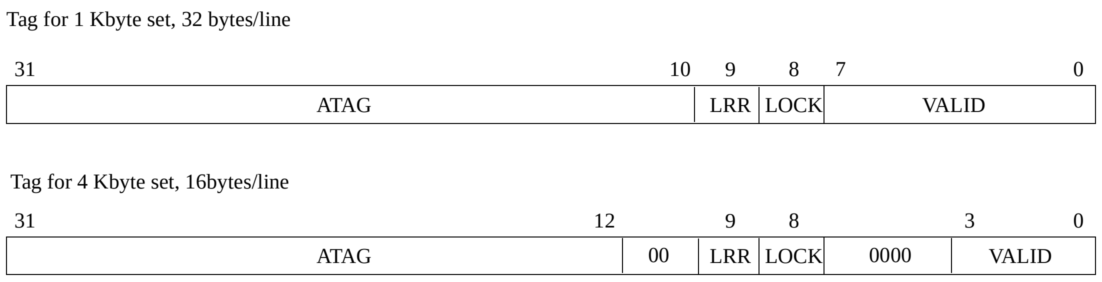

Tag for 1 Kbyte set, 32 bytes/line
31
10
9
8
7
0
ATAG
LRR
LOCK
VALID
Tag for 4 Kbyte set, 16bytes/line
31
12
9
8
3
0
ATAG
00
LRR
LOCK
0000
VALID

Figure 175. Instruction cache tag layout examples

Field Definitions:

[31:10]:	Address Tag (ATAG) - Contains the tag address of the cache line.
[9]:	LRR - Used by LRR algorithm to store replacement history, otherwise 0.
[8]:	LOCK - Locks a cache line when set. 0 if cache locking not implemented.
[7:0]:	Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits are 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.

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 4 kbyte cache with 16 bytes per line would only have four valid bits and 20 tag bits. The cache rams are sized automatically by the ram generators in the model.

52.4 Data cache

52.4.1 Operation

The data cache can be configured as a direct-mapped cache or as a multi-set cache with associativity of 2 - 4 implementing either LRU or (pseudo-) random replacement policy or as 2-way associative cache implementing LRR algorithm. The set size is configurable to 1 - 64 kbyte and divided into cache lines of 16 - 32 bytes. Each line has a cache tag associated with it consisting of a tag field, valid field with one valid bit for each 4-byte sub-block and optional lock and LRR bits. On a data cache read-miss to a cachable location 4 bytes of data are loaded into the cache from main memory. The write policy for stores is write-through with no-allocate on write-miss. In a multi-set configuration a line to be replaced on read-miss is chosen according to the replacement policy. Locked AHB transfers are generated for LDST and SWAP instructions. 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.

52.4.2 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.

52.4.3 Data cache tag

A data cache tag entry consists of several fields as shown in figure 176:

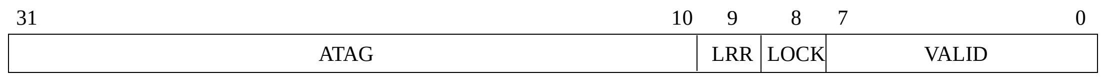


Figure 176. Data cache tag layout

Field Definitions:

[31:10]:	Address Tag (ATAG) - Contains the address of the data held in the cache line.
[9]:	LRR - Used by LRR algorithm to store replacement history. ‘0’ if LRR is not used.
[8]:	LOCK - Locks a cache line when set. ‘0’ if instruction cache locking was not enabled in the configuration.
[3:0]:	Valid (V) - When set, the corresponding sub-block of the cache line contains valid data. These bits are 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. V[0] corresponds to address 0 in the cache line, V[1] to address 1, V[2] to address 2 and V[3] to address 3.

NOTE: only the necessary bits will be implemented in the cache tag, depending on the cache configuration. As an example, a 2 kbyte cache with 32 bytes per line would only have eight valid bits and 21 tag bits. The cache rams are sized automatically by the ram generators in the model.

52.5 Additional cache functionality

52.5.1 Cache flushing

Both instruction and data cache are flushed by executing the FLUSH instruction. The instruction cache is also flushed by setting the FI bit in the cache control register, or by writing to any location with ASI=0x15. The data cache is also flushed by setting the FD bit in the cache control register, or by writing to any location with ASI=0x16. Cache flushing takes one cycle per cache line, during which the IU will not be halted, but during which the caches are disabled. When the flush operation is completed, the cache will resume the state (disabled, enabled or frozen) indicated in the cache control register. Diagnostic access to the cache is not possible during a FLUSH operation and will cause a data exception (trap=0x09) if attempted.

52.5.2 Diagnostic cache access

Tags and data in the instruction and data cache can be accessed through ASI address space 0xC, 0xD, 0xE and 0xF by executing LDA and STA instructions. Address bits making up the cache offset will be used to index the tag to be accessed while the least significant bits of the bits making up the address tag will be used to index the cache set.

Diagnostic read of tags is possible by executing an LDA instruction with ASI=0xC for instruction cache tags and ASI=0xE for data cache tags. A cache line and set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. Similarly, the data sub-blocks may be read by executing an LDA instruction with ASI=0xD for instruction cache data and ASI=0xF for data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

The tags can be directly written by executing a STA instruction with ASI=0xC for the instruction cache tags and ASI=0xE for the data cache tags. The cache line and set are indexed by the address bits making up the cache offset and the least significant bits of the address bits making up the address tag. D[31:10] is written into the ATAG field (see above) and the valid bits are written with the D[7:0] of the write data. Bit D[9] is written into the LRR bit (if enabled) and D[8] is written into the lock bit (if enabled). The data sub-blocks can be directly written by executing a STA instruction with ASI=0xD for the instruction cache data and ASI=0xF for the data cache data. The sub-block to be read in the indexed cache line and set is selected by A[4:2].

In multi-way caches, the address of the tags and data of the ways are concatenated. The address of a tag or data is thus:

ADDRESS = WAY & LINE & DATA & “00”

Examples: the tag for line 2 in way 1 of a 2x4 Kbyte cache with 16 byte line would be:

A[13:12]	= 1	(WAY)
A[11:5]	= 2	(TAG)

=> TAG ADDRESS = 0x1040

The data of this line would be at addresses 0x1040 - 0x104C

52.5.3 Cache line locking

In a multi-set configuration the instruction and data cache controllers can be configured with optional lock bit in the cache tag. Setting the lock bit prevents the cache line to be replaced by the replacement algorithm. A cache line is locked by performing a diagnostic write to the instruction tag on the cache offset of the line to be locked setting the Address Tag field to the address tag of the line to be locked, setting the lock bit and clearing the valid bits. The locked cache line will be updated on a read-miss and will remain in the cache until the line is unlocked. The first cache line on certain cache offset is locked in the set 0. If several lines on the same cache offset are to be locked the locking is performed on the same cache offset and in sets in ascending order starting with set 0. The last set can not be locked and is always replaceable. Unlocking is performed in descending set order.

NOTE: Setting the lock bit in a cache tag and reading the same tag will show if the cache line locking was enabled during the LEON3 configuration: the lock bit will be set if the cache line locking was enabled otherwise it will be 0.

52.5.4 Local instruction ram

A local instruction ram can optionally be attached to the instruction cache controller. The size of the local instruction is configurable from 1-64 kB. The local instruction ram can be mapped to any 16 Mbyte block of the address space. When executing in the local instruction ram all instruction fetches are performed from the local instruction ram and will never cause IU pipeline stall or generate an instruction fetch on the AHB bus. Local instruction ram can be accessed through load/store integer word instructions (LD/ST). Only word accesses are allowed, byte, halfword or double word access to the local instruction ram will generate data exception.

52.5.5 Local scratch pad ram

Local scratch pad ram can optionally be attached to both instruction and data cache controllers. The scratch pad ram provides fast 0-waitstates ram memories for both instructions and data. The ram can be between 1 - 512 kbyte, and mapped on any 16 Mbyte block in the address space. Accessed performed to the scratch pad ram are not cached, and will not appear on the AHB bus. The scratch pads rams do not appear on the AHB bus, and can only be read or written by the processor. The instruction ram must be initialized by software (through store instructions) before it can be used. The default address for the instruction ram is 0x8e000000, and for the data ram 0x8f000000. See section 52.10 for additional configuration details. Note: local scratch pad ram can only be enabled when the MMU is disabled.

52.5.6 Data Cache snooping

To keep the data cache synchronized with external memory, cache snooping can be enabled through the *dsnoop* generic. When enabled, the data cache monitors write accesses on the AHB bus to cacheable locations. If an other AHB master writes to a cacheable location which is currently cached in the data cache, the corresponding cache line is marked as invalid.

52.5.7 Cache Control Register

The operation of the instruction and data caches is controlled through a common Cache Control Register (CCR) (figure 177). 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.

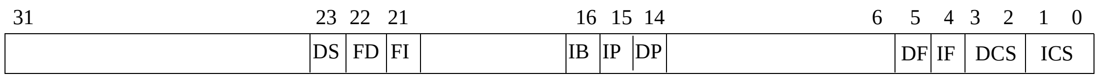


Figure 177. Cache control register

[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.
[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.
[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.

52.5.8 Cache configuration registers

The configuration of the two caches if defined in two registers: the instruction and data configuration registers. These registers are read-only and indicate the size and configuration of the caches.

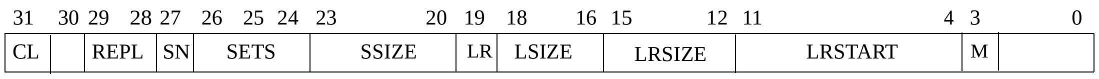


Figure 178. Cache configuration register

[31]:	Cache locking (CL). Set if cache locking is implemented.
[29:28]:	Cache replacement policy (REPL). 00 - no replacement policy (direct-mapped cache), 01 - least recently used (LRU), 10 - least recently replaced (LRR), 11 - random
[27]:	Cache snooping (SN). Set if snooping is implemented.
[26:24]:	Cache associativity (SETS). Number of sets in the cache: 000 - direct mapped, 001 - 2-way associative, 010 - 3-way associative, 011 - 4-way associative
[23:20]:	Set size (SSIZE). Indicates the size (Kbytes) of each cache set. Size = 2^{SIZE}
[19]:	Local ram (LR). Set if local scratch pad ram is implemented.
[18:16]:	Line size (LSIZE). Indicated the size (words) of each cache line. Line size = 2^{LSZ}
[15:12]:	Local ram size (LRSZ). Indicates the size (Kbytes) of the implemented local scratch pad ram. Local ram size = 2^{LRSZ}
[11:4]:	Local ram start address. Indicates the 8 most significant bits of the local ram start address.
[3]:	MMU present. This bit is set to ‘1’ if an MMU is present.

All cache registers are accessed through load/store operations to the alternate address space (LDA/STA), using ASI = 2. The table below shows the register addresses:

*Table 609.*ASI 2 (system registers) address map

Address	**Register**
0x00	Cache control register
0x04	Reserved
0x08	Instruction cache configuration register
0x0C	Data cache configuration register

52.5.9 Software consideration

After reset, the caches are disabled and the cache control register (CCR) is 0. Before the caches may be enabled, a flush operation must be performed to initialized (clear) the tags and valid bits. A suitable assembly sequence could be:

```
flush
set 0x81000f, %g1
sta%g1, [%g0] 2
```

52.6 Memory management unit

A memory management unit (MMU) compatible with the SPARC V8 reference MMU can optionally be configured. For details on operation, see the SPARC V8 manual.

52.6.1 ASI mappings

When the MMU is used, the following ASI mappings are added:

*Table 610.*MMU ASI usage

ASI	**Usage**
0x10	Flush page
0x10	MMU flush page
0x13	MMU flush context
0x14	MMU diagnostic dcache context access
0x15	MMU diagnostic icache context access
0x19	MMU registers
0x1C	MMU bypass
0x1D	MMU diagnostic access
0x1E	MMU snoop tags diagnostic access

52.6.2 MMU/Cache operation

When the MMU is disabled, the caches operate as normal with physical address mapping. When the MMU is enabled, the caches tags store the virtual address and also include an 8-bit context field. If cache snooping is desired, bit 2 of the *dsnoop* generic has to be set. This will store the physical tag in a separate RAM block, which then is used for snooping. In addition, the size of each data cache way has to be smaller or equal to 4 kbyte (MMU page size). This is necessary to avoid aliasing in the cache since the virtual tags are indexed with a virtual offset while the physical tags are indexed with a physical offset. Physical tags and snoop support is needed for SMP systems using the MMU (linux-2.6).

Because the cache is virtually tagged, no extra clock cycles are needed in case of a cache load hit. In case of a cache miss or store hit (write-through cache) at least 2 extra clock cycles are used if there is a TLB hit. If there is a TLB miss the page table must be traversed, resulting in up to 4 AMBA read accesses and one possible writeback operation. If a combined TLB is used by the instruction cache, the translation is stalled until the TLB is free. If fast TLB operation is selected (tlb_type = 2), the TLB will be accessed simultaneously with tag access, saving 2 clocks on cache miss. This will increase the area somewhat, and may reduce the timing, but usually results in better overall throughput.

52.6.3 MMU registers

The following MMU registers are implemented:

*Table 611.*MMU registers (ASI = 0x19)

Address	**Register**
0x000	MMU control register
0x100	Context pointer register
0x200	Context register
0x300	Fault status register
0x400	Fault address register

The definition of the registers can be found in the SPARC V8 manual.

52.6.4 Translation look-aside buffer (TLB)

The MMU can be configured to use a shared TLB, or separate TLB for instructions and data. The number of TLB entries can be set to 2 - 32 in the configuration record. The organisation of the TLB and number of entries is not visible to the software and does thus not require any modification to the operating system.

52.6.5 Snoop tag diagnostic access

If the MMU has been configured to use separate snoop tags, they can be accessed via ASI 0x1E. This is primarily useful for RAM testing, and should not be performed during normal operation. The figure below shows the layout of the snoop tag for a 1 Kbyte data cache:

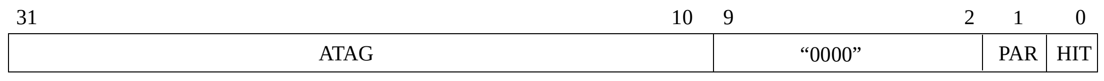


Figure 179. Snoop cache tag layout

[31:10] Address tag. The physical address tag of the cache line.
[1]: Parity. The odd parity over the data tag. LEON3FT only.
[0]: Snoop hit. When set, the cache line is not valid and will cause a cache miss if accessed by the processor.

52.7 Floating-point unit and custom co-processor interface

The SPARC V8 architecture defines two (optional) co-processors: one floating-point unit (FPU) and one user-defined co-processor. The LEON3 pipeline provides an interface port for both of these units. Two different FPU's can be interfaced: Gaisler Research's GRFPU, and the Meiko FPU from Sun. Selection of which FPU to use is done through the VHDL model's generic map. The characteristics of the FPU's are described in the next sections.

52.7.1 Gaisler Research's floating-point unit (GRFPU)

The high-performance GRFPU operates on single- and double-precision operands, and implements all SPARC V8 FPU instructions. The FPU is interfaced to the LEON3 pipeline using a LEON3-specific FPU controller (GRFPC) that allows FPU instructions to be executed simultaneously with integer instructions. Only in case of a data or resource dependency is the integer pipeline held. The GRFPU is fully pipelined and allows the start of one instruction each clock cycle, with the exception is FDIV and FSQRT which can only be executed one at a time. The FDIV and FSQRT are however executed in a separate divide unit and do not block the FPU from performing all other operations in parallel.

All instructions except FDIV and FSQRT has a latency of three cycles, but to improve timing, the LEON3 FPU controller inserts an extra pipeline stage in the result forwarding path. This results in a latency of four clock cycles at instruction level. The table below shows the GRFPU instruction timing when used together with GRFPC:

*Table 612.*GRFPU instruction timing with GRFPC

Instruction	**Throughput**	**Latency**
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	1	4
FDIVS	14	16
FDIVD	15	17
FSQRTS	22	24
FSQRTD	23	25

The GRFPC controller implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to 7 queued instructions when an FPU exception is taken. When the GRFPU is enabled in the model, the version field in %fsr has the value of 2.

52.7.2 GRFPU-Lite

GRFPU-Lite is a smaller version of GRFPU, suitable for FPGA implementations with limited logic resources. The GRFPU-Lite is not pipelined and executes thus only one instruction at a time. To improve performance, the FPU controller (GRLFPC) allows GRFPU-Lite to execute in parallel with the processor pipeline as long as no new FPU instructions are pending. Below is a table of worst-case throughput of the GRFPU-Lite:

*Table 613.*GRFPU-Lite worst-case instruction timing with GRLFPC

Instruction	Throughput	Latency
FADDS, FADDD, FSUBS, FSUBD,FMULS, FMULD, FSMULD, FITOS, FITOD, FSTOI, FDTOI, FSTOD, FDTOS, FCMPS, FCMPD, FCMPES. FCMPED	8	8
FDIVS	31	31
FDIVD	57	57
FSQRTS	46	46
FSQRTD	65	65

When the GRFPU-Lite is enabled in the model, the version field in %fsr has the value of 3.

52.7.3 The Meiko FPU

The Meiko floating-point core operates on both single- and double-precision operands, and implements all SPARC V8 FPU instructions. The Meiko FPU is interfaced through the Meiko FPU controller (MFC), which allows one FPU instruction to execute in parallel with IU operation. The MFC implements the SPARC deferred trap model, and the FPU trap queue (FQ) can contain up to one queued instruction when an FPU exception is taken.

When the Meiko FPU is enabled in the model, the version field in %fsr has the value of 1.

The Meiko FPU is not distributed by Gaisler Research, and must be obtained separately from Sun.

52.7.4 Generic co-processor

LEON can be configured to provide a generic interface to a user-defined co-processor. The interface allows an execution unit to operate in parallel to increase performance. One co-processor instruction can be started each cycle as long as there are no data dependencies. When finished, the result is written back to the co-processor register file.

52.8 Vendor and device identifiers

The core has vendor identifiers 0x01 (Gaisler Research) and device identifiers 0x003. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

52.9 Implementation

52.9.1 Area and timing

Both area and timing of the LEON3 core depends strongly on the selected configuration, target technology and the used synthesis tool. The table below indicates the typical figures for two baseline configurations.

*Table 614.*Area and timing

	Actel AX2000			ASIC (0.13 um)	
Configuration	Cells	RAM64	MHz	Gates	MHz
LEON3, 8 + 8 Kbyte cache	6,500	40	30	25,000	400
LEON3, 8 + 8 Kbyte cache + DSU3	7,500	40	25	30,000	400

52.9.2 Technology mapping

LEON3 has two technology mapping generics, *fabtech* and *memtech*. The *fabtech* generic controls the implementation of some pipeline features, while *memtech* selects which memory blocks will be used to implement cache memories and the IU/FPU register file. *Fabtech* can be set to any of the provided technologies (0 - NTECH) as defined in the GRPIB.TECH package. See the GRLIB Users's Manual for available settings for *memtech*.

52.9.3 RAM usage

The LEON3 core maps all usage of RAM memory on the *syncram*, *syncram_2p* and *syncram_dp* components from the technology mapping library (TECHMAP). The type, configuration and number of RAM blocks is described below.

Register file

The register file is implemented with two *synram_2p* blocks for all technologies where the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p is shown in the following table:

*Table 615.*syncram_2p sizes for LEON3 register file

Register windows	Syncram_2p organization
2 - 3	64x32
4 - 7	128x32
8 - 15	256x32
16-31	512x31
32	1024x32

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the register. On FPGA technologies, it can be in either flip-flops or RAM cells, depending on the tool and technology. On ASIC technologies, it will be flip-flops. The amount of flip-flops inferred is equal to the number of registers:

Number of flip-flops = ((NWINDOWS *16) + 8) * 32

FP register file

If FPU support is enabled, the FP register file is implemented with four *synram_2p* blocks when the *regfile_3p_infer* constant in TECHMAP.GENCOMP is set to 0. The organization of the syncram_2p blocks is 16x32.

If *regfile_3p_infer* is set to 1, the synthesis tool will automatically infer the FP register file. For ASIC technologies the number of inferred flip-flops is equal to number of bits in the FP register file which is 32 * 32 = 1024.

Cache memories

RAM blocks are used to implement the cache tags and data memories. Depending on cache configuration, different types and sizes of RAM blocks are used.

The tag memory is implemented with one *syncram* per cache way when no snooping is enabled. The tag memory depth and width is calculated as follows:

Depth = (cache way size in bytes) / (cache line size in bytes)

Width = 32 - log2(cache way size in bytes) + (cache line size in bytes)/4 + lrr + lock

For a 2 Kbyte cache way with lrr replacement and 32 bytes/line, the tag RAM depth will be (2048/32) = 64. The width will be: 32 - log2(2048) + 32/4 + 1 = 32 - 11 + 8 + 1 = 28. The tag RAM organization will thus be 64x28 for the configuration. If the MMU is enabled, the tag memory width will increase with 8 to store the process context ID, and the above configuration will us a 64x36 RAM.

If snooping is enabled, the tag memories will be implemented using the *syncram_dp* component (dual-port RAM). One port will be used by the processor for cache access/refill, while the other port will be used by the snooping and invalidation logic. The size of the *syncram_dp* block will be the same as when snooping is disabled. If physical snooping is enabled (separate snoop tags), one extra RAM block per data way will be instatiated to hold the physical tags. The width of the RAM block will be the same as the tag address: 32 - log2(way size). A 4 Kbyte data cache way will thus require a 32 - 12 = 20 bit wide RAM block for the physical tags. If fast snooping is enabled, the tag RAM (virtual and physical) will be implemented using *syncram_2p* instead of *syncram_dp*. This can be used to implement snooping on technologies which lack dual-port RAM but have 2-port RAM.

The data part of the caches (storing instructions or data) is always 32 bit wide. The depth is equal to the way size in bytes, divided by 4. A cache way of 2 Kbyte will thus use *syncram* component with and organization of 512x32.

Instruction Trace buffer

The instruction trace buffer will use four identical RAM blocks (*syncram*) to implement the buffer memory. The syncrams will always be 32-bit wide. The depth will depend on the TBUF generic, which indicates the total size of trace buffer in Kbytes. If TBUF = 1 (1 Kbyte), then four RAM blocks of 64x32 will be used. If TBUF = 2, then the RAM blocks will be 128x32 and so on.

Scratch pad RAM

If the instruction scratch pad RAM is enabled, a *syncram* block will be instantiated with a 32-bit data width. The depth of the RAM will correspond to the configured scratch pad size. An 8 Kbyte scratch pad will use a *syncram* with 2048x32 organization. The RAM block for the data scratch pad will be configured in the same way as the instruction scratch pad.

52.9.4 Double clocking

The LEON3 CPU core be clocked at twice the clock speed of the AMBA AHB bus. When clocked at double AHB clock frequency, all CPU core parts including integer unit and caches will operate at double AHB clock frequency while the AHB bus access is performed at the slower AHB clock frequency. The two clocks have to be synchronous and a multicycle paths between the two clock domains have to be defined at synthesis tool level. A separate component (leon3s2x) is provided for the double clocked core. Double clocked versions of DSU (dsu3_2x) and MP interrupt controller (irqmp2x) are used in a double clocked LEON3 system. An AHB clock qualifier signal (*clken* input) is used to identify end of AHB cycle. The AHB qualifier signal is generated in CPU clock domain and is high during the last CPU clock cycle under AHB clock low-phase. Sample *leon3-clk2x* design provides a module that generates an AHB clock qualifier signal.

Double-clocked design has two clock domains: AMBA clock domains (HCLK) and CPU clock domain (CPUCLK). LEON3 (leon3s2x component) and DSU3 (dsu3_2x) belong to CPU clock domain (clocked by CPUCLK), while the rest of the system is in AMBA clock domain (clocked by HCLK). Paths between the two clock domains (paths starting in CPUCLK domain and ending in

HCLK and paths starting in HCLK domain and ending in CPUCLK domain) are multicycle paths with propagation time of two CPUCLK periods (or one HCLK period) with following exceptions:

Start point	Through	End point	Propagation time
leon3s2x core			
CPUCLK	ahbi	CPUCLK	2 CPUCLK
CPUCLK	ahbsi	CPUCLK	2 CPUCLK
CPUCLK	ahbso	CPUCLK	2 CPUCLK
HCLK	irqi	CPUCLK	1 CPUCLK
CPUCLK	irqo	HCLK	1 CPUCLK
CPUCLK		u0_0/p0/c0/sync0/r[*] (register)	1 CPUCLK

Start point	Through	End point	Propagation time
dsu3_2x core			
CPUCLK	ahbmi	CPUCLK	2 CPUCLK
CPUCLK	ahbsi	CPUCLK	2 CPUCLK
	dsui	CPUCLK	1 CPUCLK
r[*] (register)		rh[*] (register)	1 CPUCLK
irqmp2x core			
r2[*] (register)		r[*] (register)	1 CPUCLK

52.9.5 Clock gating

To further reduce the power consumption of the processor, the clock can be gated-off when the processor has entered power-down state. Since the cache controllers and MMU operate in parallel with the processor, the clock cannot be gated immediately when the processor has entered the power-down state. Instead, a power-down signal (DBGO.idle) is generated when all outstanding AHB accesses have been completed and it is safe to gate the clock. This signal should be clocked though a positive-edge flip-flop followed by a negative-edge flip-flop to guarantee that the clock is gated off during the clock-low phase. To insure proper start-up state, the clock should not be gated during reset.

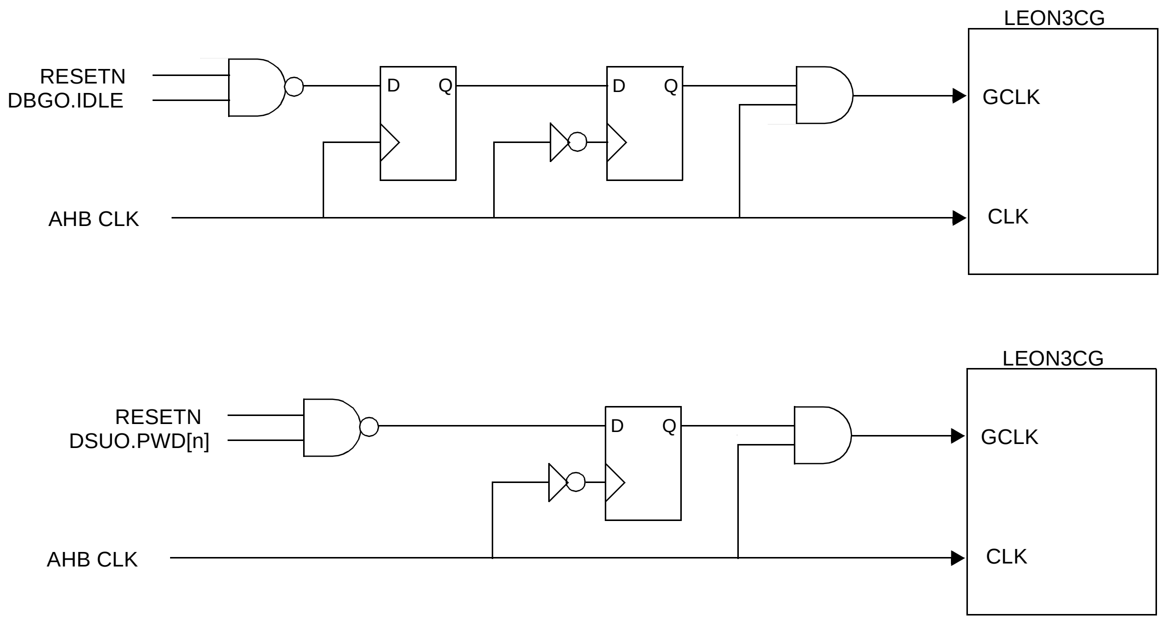


Figure 180. Examples of LEON3 clock gating

The processor should exit the power-down state when an interrupt become pending. The signal DBGO.ipend will then go high when this happen, and should be used to re-enable the clock.

When the debug support unit (DSU3) is used, the DSUO.pwd signal should be used instead of DBGO.idle. This will insure that the clock also is re-enabled when the processor is switched from power-down to debug state by the DSU. The DSUO.pwd is a vector with one power-down signal per CPU (for SMP systems). DSUO.pwd takes DBGO.ipend into account, and no further gating or latching needs to be done of this signal. If cache snooping has been enabled, the continuous clock will insure that the snooping logic is activated when necessary and will keep the data cache synchronized even when the processor clock is gated-off. In a multi-processor system, all processor except node 0 will enter power-down after reset and will allow immediate clock-gating without additional software support.

Clock-tree routing must insure that the continuous clock (CLK) and the gated clock (GCLK) are phase-aligned. The template design *leon3-clock-gating* shows an example of a clock-gated system. The *leon3cg* entity should be used when clock gating is implemented. This entity has one input more (GCLK) which should be driven by the gated clock. Using the double-clocked version of leon3 (leon3s2x), the GCLK2 is the gated double-clock while CLK and CLK2 should be continuous.

52.9.6 Scan support

If the SCANTEST generic is set to 1, support for scan testing is enabled. This will make use of the AHB scan support signals in the following manner: when AHBI.testen and AHBI.scanen are both ‘1’, the select signals to all RAM blocks (cache RAM, register file and DSU trace buffers) are disabled. This means that when the scan chain is shifted, no accidental write or read can occur in the RAM blocks. The scan signal AHBI.testrst is not used as there are no asynchronous resets in the LEON3 core.

52.10 Configuration options

Table 616 shows the configuration options of the core (VHDL generics).

*Table 616.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
fabtech	Target technology	0 - NTECH	0 (inferred)
memtech	Vendor library for regfile and cache RAMs	0 - NTECH	0 (inferred)
nwindows	Number of SPARC register windows. Choose 8 windows to be compatible with Bare-C and RTEMS cross-compilers.	2 - 32	8
dsu	Enable Debug Support Unit interface	0 - 1	0
fpu	Floating-point Unit 0 : no FPU 1 - 7: GRFPU 1 - inferred multiplier, 2 - DW multiplier, 3 - Module Generator multiplier 8 - 14: GRFPU-Lite 8 - simple FPC, 9 - data forwarding FPC, 10 - non-blocking FPC 15: Meiko 16 - 31: as above (modulo 16) but use netlist	0 - 31	0
v8	Generate SPARC V8 MUL and DIV instructions 0 : No multiplier or divider 1 : 16x16 multiplier 2: 16x16 pipelined multiplier 16#32# : 32x32 pipelined multiplier	0 - 16#3F#	0
cp	Generate co-processor interface	0 -1	0
mac	Generate SPARC V8e SMAC/UMAC instruction	0 - 1	0
pclow	Least significant bit of PC (Program Counter) that is actually generated. PC[1:0] are always zero and are normally not generated. Generating PC[1:0] makes VHDL-debugging easier.	0, 2	2
notag	Currently not used	-	-
nwp	Number of watchpoints	0 - 4	0
icen	Enable instruction cache	0 - 1	1
irepl	Instruction cache replacement policy. 0 - least recently used (LRU), 1 - least recently replaced (LRR), 2 - random	0 - 1	0
isets	Number of instruction cache sets	1 - 4	1
ilinesize	Instruction cache line size in number of words	4, 8	4
isetsize	Size of each instruction cache set in kByte	1 - 256	1
isetlock	Enable instruction cache line locking	0 - 1	0
dcen	Data cache enable	0 - 1	1
drepl	Data cache replacement policy. 0 - least recently used (LRU), 1 - least recently replaced (LRR), 2 - random	0 - 1	0
dsets	Number of data cache sets	1 - 4	1
dlinesize	Data cache line size in number of words	4, 8	4
dsetsize	Size of each data cache set in kByte	1 - 256	1
dsetlock	Enable data cache line locking	0 - 1	0

*Table 616.*Configuration options

Generic	Function	Allowed range	Default
dsnoop	Enable data cache snooping Bit 0-1: 0: disable, 1: slow, 2: fast (see text) Bit 2: 0: simple snooping, 1: save extra physical tags (MMU snooping)	0 - 6	0
ilram	Enable local instruction RAM	0 - 1	0
ilramsize	Local instruction RAM size in kB	1 - 512	1
ilramstart	8 MSB bits used to decode local instruction RAM area	0 - 255	16#8E#
dlram	Enable local data RAM (scratch-pad RAM)	0 - 1	0
dlramsize	Local data RAM size in kB	1 - 512	1
dlramstart	8 MSB bits used to decode local data RAM area	0 - 255	16#8F#
mmuen	Enable memory management unit (MMU)	0 - 1	0
itlbnum	Number of instruction TLB entries	2 - 64	8
dtlbnum	Number of data TLB entries	2 - 64	8
tlb_type	0 : separate TLB with slow write 1: shared TLB with slow write 2: separate TLB with fast write	0 - 2	1
tlb_rep	LRU (0) or Random (1) TLB replacement	0 - 1	0
lddel	Load delay. One cycle gives best performance, but might create a critical path on targets with slow (data) cache memories. A 2-cycle delay can improve timing but will reduce performance with about 5%.	1 - 2	2
disas	Print instruction disassembly in VHDL simulator console.	0 - 1	0
tbuf	Size of instruction trace buffer in kB (0 - instruction trace disabled)	0 - 64	0
pwd	Power-down. 0 - disabled, 1 - area efficient, 2 - timing efficient.	0 - 2	1
svt	Enable single-vector trapping	0 - 1	0
rstaddr	Default reset start address	0 - (2**^{20}-1)	0
smp	Enable multi-processor support	0 - 15	0
cached	Fixed cacheability mask	0 - 16#FFFF#	0
scantest	Enable scan test support	0 - 1	0

52.11 Signal descriptions

Table 617 shows the interface signals of the core (VHDL ports).

*Table 617.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	AMBA and processor clock (leon3s, leon3cg)	-
CLK2		Input	Processor clock in 2x mode (leon3sx2)	
GCLK2		Input	Gated processor clock in 2x mode (leon3sx2)	
RSTN	N/A	Input	Reset	Low
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
AHBSI	*	Input	AHB slave input signals	-
IRQI	IRL[3:0]	Input	Interrupt level	High
	RST	Input	Reset power-down and error mode	High
	RUN	Input	Start after reset (SMP system only)	
IRQO	INTACK	Output	Interrupt acknowledge	High
	IRL[3:0]	Output	Processor interrupt level	High
DBGI	-	Input	Debug inputs from DSU	-
DBGO	-	Output	Debug outputs to DSU	-
	ERROR		Processor in error mode, execution halted	Low
GCLK		Input	Gated processor clock for leon3cg	

* see GRLIB IP Library User’s Manual

52.12 Library dependencies

Table 618 shows the libraries used when instantiating the core (VHDL libraries).

*Table 618.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	LEON3	Component, signals	LEON3 component declaration, interrupt and debug signals declaration

52.13 Component declaration

The core has the following component declaration.

```
entity leon3s is
  generic (
    hindex    : integer               := 0;
    fabtech   : integer range 0 to NTECH  := 0;
    memtech   : integer range 0 to NTECH  := 0;
    nwindows  : integer range 2 to 32 := 8;
    dsu       : integer range 0 to 1  := 0;
    fpu       : integer range 0 to 3  := 0;
    v8        : integer range 0 to 2  := 0;
    cp        : integer range 0 to 1  := 0;
    mac       : integer range 0 to 1  := 0;
    pclow     : integer range 0 to 2  := 2;
    notag     : integer range 0 to 1  := 0;
    nwp       : integer range 0 to 4  := 0;
    icen      : integer range 0 to 1  := 0;
    irepl     : integer range 0 to 2  := 2;
    isets     : integer range 1 to 4  := 1;
    ilinesize : integer range 4 to 8  := 4;
    isetsize  : integer range 1 to 256 := 1;
    isetlock  : integer range 0 to 1  := 0;
    dcen      : integer range 0 to 1  := 0;
    drepl     : integer range 0 to 2  := 2;
    dsets     : integer range 1 to 4  := 1;
    dlinesize : integer range 4 to 8  := 4;
    dsetsize  : integer range 1 to 256 := 1;
    dsetlock  : integer range 0 to 1  := 0;
    dsnoop    : integer range 0 to 6:= 0;
    ilram      : integer range 0 to 1 := 0;
    ilramsize  : integer range 1 to 512 := 1;
    ilramstart : integer range 0 to 255 := 16#8e#;
    dlram      : integer range 0 to 1 := 0;
    dlramsize  : integer range 1 to 512 := 1;
    dlramstart : integer range 0 to 255 := 16#8f#;
    mmuen     : integer range 0 to 1  := 0;
    itlbnum   : integer range 2 to 64 := 8;
    dtlbnum   : integer range 2 to 64 := 8;
    tlb_type  : integer range 0 to 1 := 1;
    tlb_rep   : integer range 0 to 1 := 0;
    lddel     : integer range 1 to 2  := 2;
    disas     : integer range 0 to 1  := 0;
    tbuf      : integer range 0 to 64 := 0;
    pwd       : integer range 0 to 2  := 2;     -- power-down
    svt       : integer range 0 to 1  := 1;     -- single vector trapping
    rstaddr   : integer               := 0;
    smp       : integer range 0 to 15 := 0;    -- support SMP systems
    cached    : integer               := 0;     -- cacheability table
    scantest  : integer               := 0
  );

port (
    clk    : in  std_ulogic;
    rstn   : in  std_ulogic;
    ahbi   : in  ahb_mst_in_type;
    ahbo   : out ahb_mst_out_type;
    ahbsi  : in  ahb_slv_in_type;
    ahbso  : in  ahb_slv_out_vector;
    irqi   : in  l3_irq_in_type;
    irqo   : out l3_irq_out_type;
    dbgi   : in  l3_debug_in_type;
    dbgo   : out l3_debug_out_type
  );
end;
```

53 LEON3FT - Fault-Tolerant SPARC V8 Processor

53.1 Overview

LEON3 is a 32-bit processor core conforming to the IEEE-1754 (SPARC V8) architecture. It is designed for embedded applications, combining high performance with low complexity and low power consumption.

The LEON3 core has the following main features: 7-stage pipeline with Harvard architecture, separate instruction and data caches, on-chip debug support and multi-processor extensions.

The LEON3FT processor is a derivate of the standard LEON3 SPARC V8 processor, enhanced with fault-tolerance against SEU errors. The fault-tolerance is focused on the protection of on-chip RAM blocks, which are used to implement IU/FPU register files and the cache memory. The LEON3FT processor is functionally identical to the standard LEON3 processor, and this chapter only outlines the FT features.

53.2 Register file SEU protection

53.2.1 IU SEU protection

The SEU protection for the integer unit register file can be implemented in four different ways, depending on target technology and available RAM blocks. The SEU protection scheme is selected during synthesis, using the *iuft* VHDL generic. Table 619 below shows the implementation characteristics of the four possible SEU protection schemes.

*Table 619.*Integer unit SEU protection schemes

ID	Implementation	Description
0	Hardened flip-flops or TMR	Register file implemented with SEU hardened flip-flops. No error checking.
1	4-bit parity with restart	4-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Pipeline restart on correction.
2	8-bit parity without restart	8-bit checksum per 32-bit word. Detects and corrects 1 bit per byte (4 bits per word). Correction on-the-fly without pipeline restart.
3	7-bit BCH with restart	7-bit BCH checksum per 32-bit word. Detects 2 bits and corrects 1 bit per word. Pipeline restart on correction.

The SEU error detection has no impact on behaviour, but a correction cycle (scheme 1 and 3) will delay the the current instruction with 6 clock cycles. An uncorrectable error in the IU register file will cause trap 0x20 (*register_access_error*).

53.2.2 FPU SEU protection

The FPU register file has similar SEU protection as the IU register file, but with less configuration options. When the GRFPU is selected and the FPU register file protection is enabled, the protection scheme is always 8-bit parity without pipeline restart. For GRFPU-Lite the protection scheme is always 4-bit parity with pipeline restart. An uncorrectable error in the FPU register file will cause an (deferred) FPU exception with %fsr.ftt set to 5 (hardware_error). When FPU register file protection is disabled the FPU register file is implemented using flip-flops.

53.2.3 ASR16 register

ASR register 16 (%asr16) is used to control the IU/FPU register file SEU protection. It is possible to disable the SEU protection by setting the IDI/FDI bits, and to inject errors using the ITE/FTE bits. Corrected errors in the register file are counted, and available in ICNT and FCNT fields. The counters saturate at their maximum value (7), and should be reset by software after read-out.

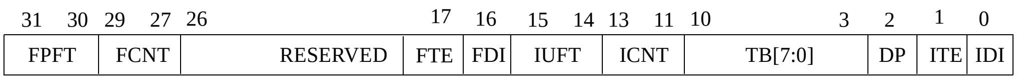


Figure 181. %asr16 - Register protection control register

[31:30]: FP FT ID - Defines which SEU protection is implemented in the FPU (table 619).
[29:27]: FP RF error counter - Number of detected parity errors in the FP register file.
[26:18]: Reserved
[17]: FPU RF Test Enable - Enables FPU register file test mode. Parity bits are xored with TB before written to the FPU register file.
[16]: FP RF protection disable (FDI) - Disables FP RF parity protection when set.
[15:14]: IU FT ID - Defines which SEU protection is implemented in the IU (table 619).
[13:11]: IU RF error counter - Number of detected parity errors in the IU register file.
[10:3]: RF Test bits (FTB) - In test mode, these bits are xored with correct parity bits before written to the register file.
[2]: DP ram select (DP) - Only applicable if the IU or FPU register files consists of two dual-port rams. See text below for details.
[1]: IU RF Test Enable - Enables register file test mode. Parity bits are xored with TB before written to the register file.
[0]: IU RF protection disable (IDI) - Disables IU RF parity protection when set.

*Table 620.*DP ram select usage

ITE/FTE	DP	Function
1	0	Write to IU register (%i, %l, %o, %g) will only write location of %rs2 Write to FPU register (%f) will only write location of %rs2
1	1	Write to IU register (%i, %l, %o, %g) will only write location of %rs1 Write to FPU register (%f) will only write location of %rs1
0	X	IU and FPU registers written nominally

53.2.4 Register file EDAC/parity bits diagnistic read-out

The register file EDAC/parity bits can be read out throught the DSU address space at 0x300800. The ECC bits are read out for both read ports simultaneously as defined in the figure below:

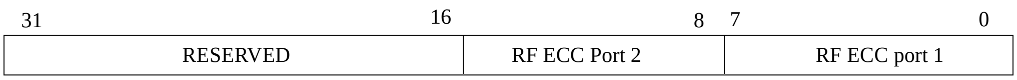


Figure 182. Register file ECC read-out layout

53.2.5 IU/FPU register file error injection

For test purposes, the IU and FPU register file EDAC/parity checkbits can be modified by software. This is done by setting the ITE or FTE bits to ‘1’. In this mode, the EDAC/parity bits are first XORed with the contents of %asr16.FTB before written to the register files.

53.3 Cache memory

Each word in the cache tag or data memories is protected by four check bits. An error during cache access will cause a cache line flush, and a re-excution of the failing instruction. This will insure that the complete cache line (tags and data) is refilled from external memory. For every detected error, a counter in the cache control register is incremented. The counters saturate at their maximum value (3), and should be reset by software after read-out. The cache memory check bits can be diagnostically read by setting the PS bit in the cache control register and then perform a normal tag or data diagnostic read.

53.3.1 Cache Control Register

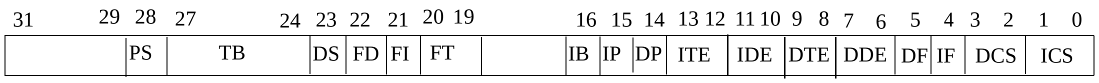


Figure 183. Cache control register

[28]:	Parity Select [PS] - if set diagnostic read will return 4 check bits in the lsb bits, otherwise tag or data word is returned.
[27:24]:	Test Bits [TB] - if set, check bits will be xored with test bits TB during diagnostic write
[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]:	FT scheme: “00” = no FT, “01” = 4-bit checking implemented
[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 Tag Errors (ITE) - Number of detected parity errors in the instruction tag cache.
[11:10]:	Instruction Data Errors (IDE) - Number of detected parity errors in the instruction data cache.
[9:8]:	Data Tag Errors (DTE) - Number of detected parity errors in the data tag cache.
[7:6]:	Data Data Errors (IDE) - Number of detected parity errors in the data data cache.
[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.

53.3.2 Diagnostic cache access

The context and parity bits for data and instruction caches can be read out via ASI 0xC - 0xF when the PS bit in the cache control register is set. The data will be organized as shown below:

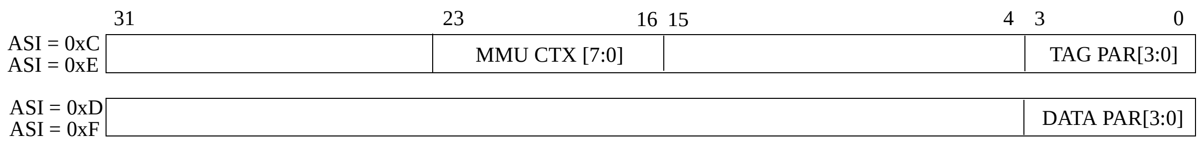


Figure 184. Data cache tag diagnostic access when CCR.PS = ‘1’

53.4 DSU memory map

The FPU register file check bits can be accessed at address 0x301800 - 0x30187C.

*Table 621.*DSU memory map

Address offset	**Register**
0x000000	DSU control register
0x000008	Time tag counter
0x000020	Break and Single Step register
0x000024	Debug Mode Mask register
0x000040	AHB trace buffer control register
0x000044	AHB trace buffer index register
0x000050	AHB breakpoint address 1
0x000054	AHB mask register 1
0x000058	AHB breakpoint address 2
0x00005c	AHB mask register 2
0x100000 - 0x110000	Instruction trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x110000	Intruction Trace buffer control register
0x200000 - 0x210000	AHB trace buffer (..0: Trace bits 127 - 96, ..4: Trace bits 95 - 64, ..8: Trace bits 63 - 32, ..C : Trace bits 31 - 0)
0x300000 - 0x3007FC	IU register file, port1 (%asr16.dpsel = 0) IU register file, port 2 (%asr16.dpsel = 1)
0x300800 - 0x300FFC	IU register file check bits
0x301000 - 0x30107C 0x301800 - 0x30187C	FPU register file FPU register file check bits
0x400000 - 0x4FFFFC	IU special purpose registers
0x400000	Y register
0x400004	PSR register
0x400008	WIM register
0x40000C	TBR register
0x400010	PC register
0x400014	NPC register
0x400018	FSR register
0x40001C	CPSR register
0x400020	DSU trap register
0x400024	DSU ASI register
0x400040 - 0x40007C	ASR16 - ASR31 (when implemented)

0x700000 - 0x7FFFFC	ASI diagnostic access (ASI = value in DSU ASI register, address = address[19:0])
	ASI = 0x9 : Local instruction RAM
	ASI = 0xB : Local data RAM
	ASI = 0xC : Instruction cache tags
	ASI = 0xD : Instruction cache data
	ASI = 0xE : Data cache tags
	ASI = 0xF : Instruction cache data

53.4.1 Data scrubbing

There is generally no need to perform data scrubbing on either IU/FPU register files or the cache memory. During normal operation, the active part of the IU/FPU register files will be flushed to memory on each task switch. This will cause all registers to be checked and corrected if necessary. Since most real-time operating systems performs several task switches per second, the data in the register files will be frequently refreshed.

The similar situation arises for the cache memory. In most applications, the cache memory is significantly smaller than the full application image, and the cache contents is gradually replaced as part of normal operation. For very small programs, the only risk of error build-up is if a part of the application is resident in the cache but not executed for a long period of time. In such cases, executing a cache flush instruction periodically (e.g. once per minute) is sufficient to refresh the cache contents.

53.4.2 Initialisation

After power-on, the check bits in the IU and FPU register files are not initialized. This means that access to an uninitialized (un-written) register could cause a register access trap (tt = 0x20). Such behaviour is considered as a software error, as the software should not read a register before it has been written. It is recommended that the boot code for the processor writes all registers in the IU and FPU register files before launching the main application.

The check bits in the cache memories do not need to be initialized as this is done automatically during cache line filling.

53.5 Vendor and device identifers

The core has vendor identifers 0x01 (Gaisler Research) and device identifers 0x053. For description of vendor and device identiferss see GRLIB IP Library User’s Manual.

53.6 Configuration options

In addition to the configuration of the standard LEON3 processor, the LEON3FT processor has the following configuartion options.

Table 622 shows the configuration options of the core (VHDL generics).

*Table 622.*Configuration options

Generic	Function	Range	Default
iuft, fpft	Register file SEU protection. (0: no protection; 1 : 4-bit parity, 2 : 8-bit parity; 3 : 7-bit BCH)	0 - 3	0
cft	Enable cache memory SEU protection.	0 - 1	0
ceinj	Error injection. Used for simulation only.	0 - 3	0
netlist	Use netlist rather then RTL code	0 - 1	0

53.7 Limitations

The LEON3FT core does not support the following functions present in the LEON3 model:

- Local instruction/data scratch pad RAM

- Cache locking

54 LOGAN - On-chip Logic Analyzer

54.1 Introduction

The LOGAN core implements an on-chip logic analyzer for tracing and displaying of on-chip signals. LOGAN consists of a circular trace buffer and a triggering module. When armed, the logic analyzers stores the traced signals in the circular buffer until a trigger condition occurs. A trigger condition will freeze the buffer, and the traced data can then be read out via an APB interface.

The depth and width of the trace buffer is configurable through VHDL generics, as well as the number of trigger levels.

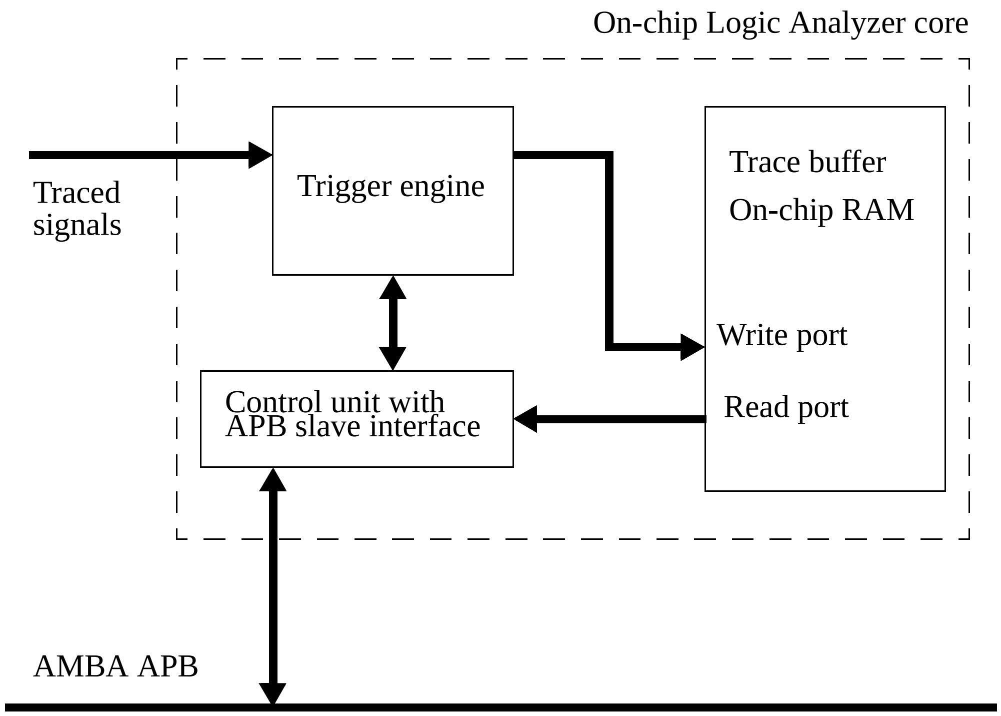


Figure 185. On-chip Logic Analyzer block diagram

54.2 Operation

54.2.1 Trace buffer

When the logic analyzer is armed, the traced signals are sampled and stored to the trace buffer on the rising edge of the sample clock (TCLK). The trace buffer consists of a circular buffer with an index register pointing to the next address in the buffer to be written. The index register is automatically incremented after each store operation to the buffer.

54.2.2 Clocking

LOGAN uses two clocks: TCLK and the APB clock. The trace signals are sampled on the rising edge of the sample clock (TCLK), while the control unit and the APB interface use the APB clock. TCLK and the APB clock does not need to be synchronized or have the same frequency.

54.2.3 Triggering

The logic analyzer contains a configurable number of trig levels. Each trig level is associated with a pattern and a mask. The traced signals are compared with the pattern, only comparing the bits set in the mask. This allows for triggering on any specific value or range. Furthermore each level has a match counter and a boolean equality flag. The equality flag specifies whether a match means that the pattern should equal the traced signals or that it should not be equal. It is possible to configure the trigger engine to stay at a certain level while the traced signals have a certain value using this flag. The match counter is a 6 bit counter which can be used to specify how many times a level should match before proceeding to the next. This is all run-time configurable through registers described in the register section.

To specify post-, center- or pre-triggering mode, the user can set a counter register that controls when the sampling stops relative to the triggering event. It can be set to any value in the range 0 to *depth*-1 thus giving total control of the trace buffer content.

To support the tracing of slowly changing signals, the logic analyzer has a 16-bit sample frequency divider register that controls how often the signals are sampled. The default divider value of 1 will sample the signals every clock cycle.

The *usequal* configuration option has a similar purpose as the sample frequency divider. The user can define one of the traced signals as a qualifier bit that has to have a specified value for the current signals to be stored in the trace buffer. This makes sampling of larger time periods possible if only some easily distinguished samples are interesting. This option has to be enabled with the *usequal* generic and the qualifier bit and value are written to a register.

54.2.4 Arming

To start operation, the logic analyzer needs to be armed. This is done by writing to the status register with bit 0 set to 1. A reset can be performed anytime by writing zero to the status register. After the final triggering event, the trigged flag will be raised and can be read out from the status register. The logic analyzer remains armed and trigged until the trigger counter reaches zero. When this happens the index of the oldest sample can be read from the trace buffer index register.

54.3 Registers

Both trace data and all registers are accessed through an APB interface. The LOGAN core will allocate a 64 kbyte block in the APB address space.

*Table 623.*APB address mapping

APB address offset	Registers
0x0000	Status register
0x0004	Trace buffer index
0x0008	Page register
0x000C	Trig counter
0x0010	Sample freq. divider
0x0014	Storage qualifier setting
0x2000 - 0x20FF	Trig control settings
0x6000 - 0x6FFF	Pattern/mask configuration
0x8000 - 0xFFFF	Trace data

54.3.1 Status register

31	30	29	28	27 20	19 6	5 0
usereg	qualifier	armed	trigged	dbits	depth	trig levels

Figure 186. Status register

[31:28] These bits indicate whether an input register and/or storage qualifier is used and if the Logic Analyzer is armed and/ or trigged.
[27:20] Number of traced signals.
[19:6] Last index of trace buffer. Depth-1.
[5:0] Number of trig levels.

54.3.2 Trace buffer index

31 abits	abits-1 0
"000...0"	the index of the oldest sample

Figure 187. Trace buffer index register

[31:*abits*] - Reserved.
[*abits*-1:0] - The index of the oldest sample in the buffer. *abits* is the number of bits needed to represent the configured depth.
Note that this register is written by the trigger engine clock domain and thus needs to be known stable when read out. Only when the 'armed' bit in the status register is zero is the content of this register reliable.

54.3.3 Page register

31 4	3 0
"000...0"	current page

Figure 188. Page register

[31:4] - Reserved.
[3:0] - This register selects what page that will be used when reading from the trace buffer.
The trace buffer is organized into pages of 1024 samples. Each sample can be between 1 and 256 bits. If the depth of the buffer is more than 1024 the page register has to be used to access the other pages. To access the i:th page the register should be set i (where i=0..15).

54.3.4 Trig counter

31 abits	abits-1 0
"000...0"	trig counter value

Figure 189. Trig counter register

[31:abits] - Reserved.
[nbits-1:0] - Trig counter value. A counter is incremented by one for each stored sample after the final triggering event and when it reaches the value stored in this register the sampling stops. 0 means posttrig and *depth-1* is pretrig. Any value in between can be used.

54.3.5 Sample frequency divider

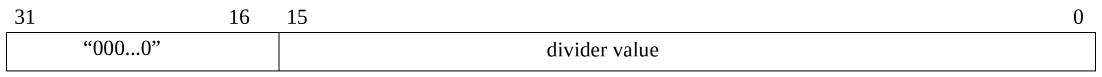


Figure 190. Sample freq. divider register

[31:16] - Reserved.
[15:0] - A sample is stored on every i:th clock cycle where i is specified through this register. This resets to 1 thus sampling occurs every cycle if not changed.

54.3.6 Storage qualifier

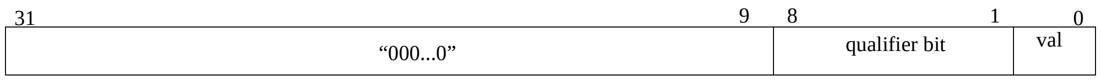


Figure 191. Storage qualifier register

[31:9] - Reserved.
[8:1] - Which bit to use as qualifier.
[0] - Qualify storage if bit is 1/0.

54.3.7 Trig control registers

This memory area contains the registers that control when the trigger engine shall proceed to the next level, i.e the match counter and a one bit field that specifies if it should trig on equality or inequality. There are *trigl* words where each word is used like in the figure below.

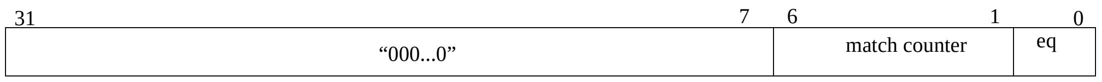


Figure 192. Trigger control register

[31:7] - Reserved.
[6:1] - Match counter. A counter is increased with one on each match on the current level and when it reaches the value stored in this register the trigger engine proceeds to the next level or if it is the last level it raises the trigged flag and starts the count of the trigger counter.
[0] - Specifies if a match is that the pattern/mask combination is equal or inequal compared to the traced signals.

54.3.8 Pattern/mask configuration

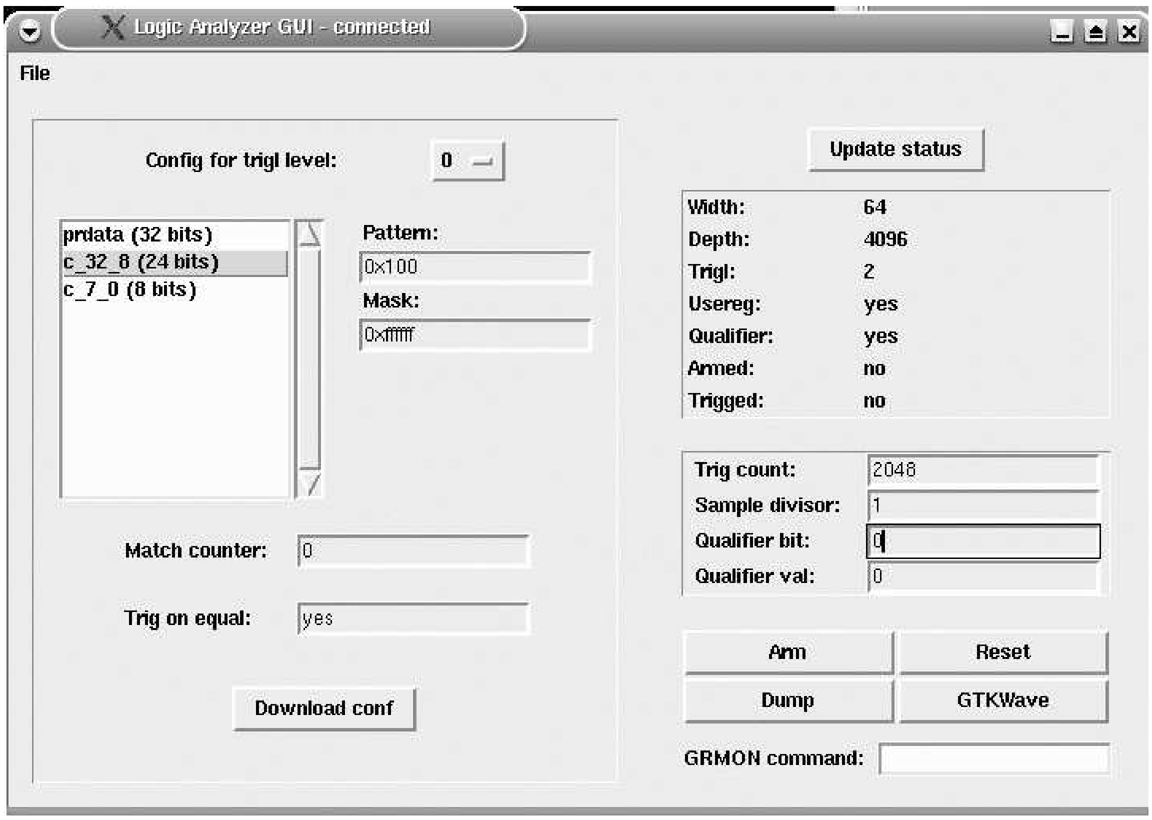
In these registers the pattern and mask for each trig level is configured. The pattern and mask can contain up to 8 words (256 bits) each so a number of writes can be necessary to specify just one pattern. They are stored with the LSB at the lowest address. The pattern of the first trig level is at 0x6000 and the mask is located 8 words later at 0x6020. Then the next trig levels starts at address 0x6040 and so on.

54.3.9 Trace data

It is placed in the upper half of the allocated APB address range. If the configuration needs more than the allocated 32 kB of the APB range the page register is used to page into the trace buffer. Each stored word is *dbits* wide but 8 words of the memory range is always allocated so the entries in the trace buffer are found at multiples of 0x20, i.e. 0x8000, 0x8020 and so on.

54.4 Graphical interface

The logic analyzer is normally controlled by the LOGAN debug driver in GRMON. It is also possible to control the LOGAN operation using a graphical user interface (GUI) written in Tcl/Tk. The GUI is provided with GRMON, refer to the GRMON manual for more details.



Config for trigl level: 0

prdata (32 bits)
c_32_8 (24 bits)
c_7_0 (8 bits)

Pattern: 0x100
Mask: 0xffffff

Match counter: 0
Trig on equal: yes

Width: 64
Depth: 4096
Trigl: 2
Usereg: yes
Qualifier: yes
Armed: no
Trigged: no

Trig count: 2048
Sample divisor: 1
Qualifier bit: 0
Qualifier val: 0

GRMON command:

54.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x062. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

54.6 Configuration options

Table 624 shows the configuration options of the core (VHDL generics).

*Table 624.*Configuration options

Generic	Function	Allowed range	Default
dbits	Number of traced signals	1 - 255	32
depth	Number of stored samples	256 - 16384	1024
trigl	Number of trigger levels	1 - 63	1
usereg	Use input register	0 - 1	1
usequal	Use storage qualifier	0 - 1	0
pindex	APB slave index	0 - NAPBSLV - 1	0
paddr	The 12-bit MSB APB address	0 -16#FFF#	0
pmask	The APB address mask	16#000 - 16#F00#	F00
memtech	Memory technology	0 - NTECH	0

The usereg VHDL generic specifies whether to use an input register to synchronize the traced signals and to minimize their fan out. If usereg=1 then all signals will be clocked into a register on the positive edge of the supplied clock signal, otherwise they are sent directly to the RAM.

54.7 Signal descriptions

Table 625 shows the interface signals of the core (VHDL ports).

*Table 625.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
TCLK	N/A	Input	Sample clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SIGNALS	N/A	Input	Vector of traced signals	-

* See GRLIB IP Library users manual

54.8 Library dependencies

Table 626 shows libraries used when instantiating the core (VHDL libraries).

*Table 626.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component	Component declaration

54.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.misc.all;

entity logan_ex is
  port (
    clk : in std_ulogic;
    rstn : in std_ulogic;

    ... -- other signals
    );
end;

architecture rtl of logan_ex is

  -- AMBA signals
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal signals : std_logic_vector(63 downto 0);

begin

-- Logic analyzer core
```

```
  logan0 : logan
  generic map (dbits=>64,depth=>4096,trigl=>2,usereg=>1,usequal=>0,
            pindex => 3, paddr => 3, pmask => 16#F00#, memtech => memtech)
  port map (rstn, clk, clk, apbi, apbo(3), signals);

end;
```

55 MCTRL - Combined PROM/IO/SRAM/SDRAM Memory Controller

55.1 Overview

The memory controller handles a memory bus hosting PROM, memory mapped I/O devices, asynchronous static ram (SRAM) and synchronous dynamic ram (SDRAM). 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- or 16-bit mode for applications with low memory and performance demands.

Chip-select decoding is done for two PROM banks, one I/O bank, five SRAM banks and two SDRAM banks.

The controller decodes three address spaces (PROM, I/O and RAM) whose mapping is determined through VHDL-generics.

Figure 193 shows how the connection to the different device types is made.

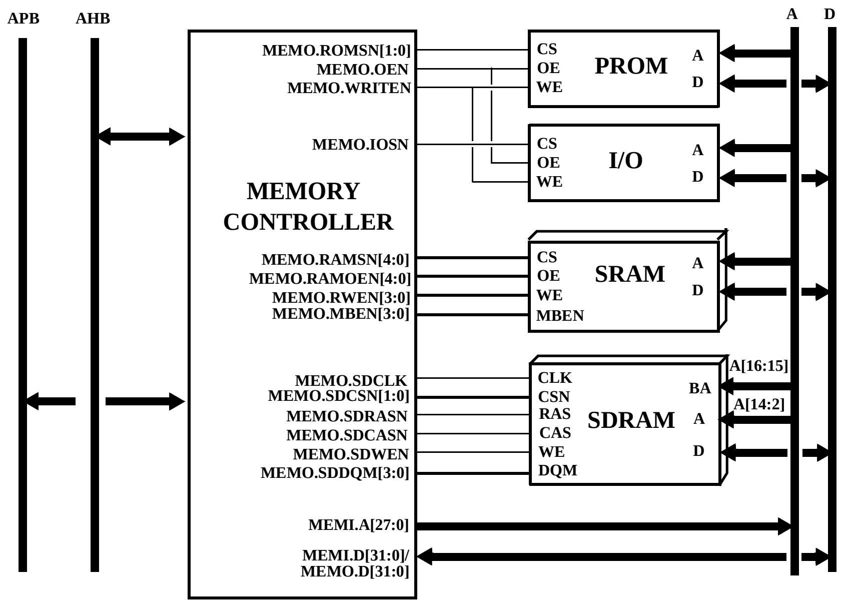


Figure 193. **Memory controller conected to AMBA bus and different types of memory devices**

55.2 PROM access

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

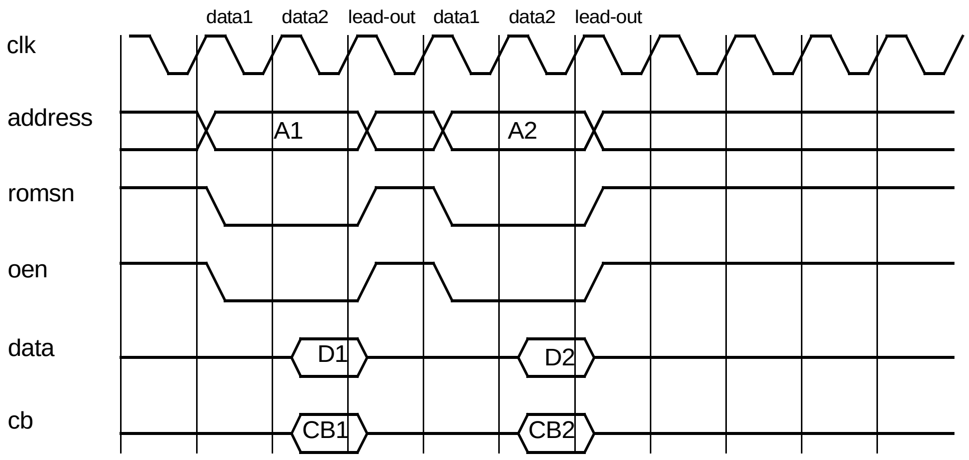


Figure 194. **Prom non-consecutive read cyclecs.**

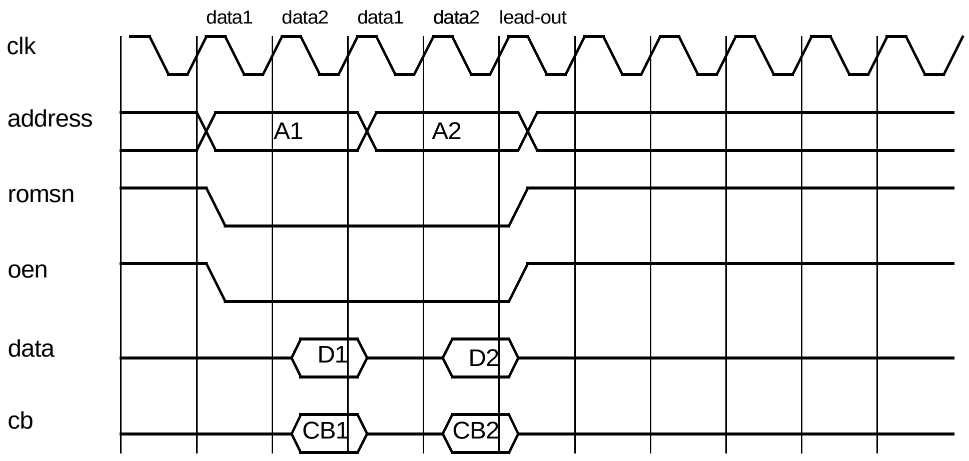


Figure 195. **Prom consecutive read cyclecs.**

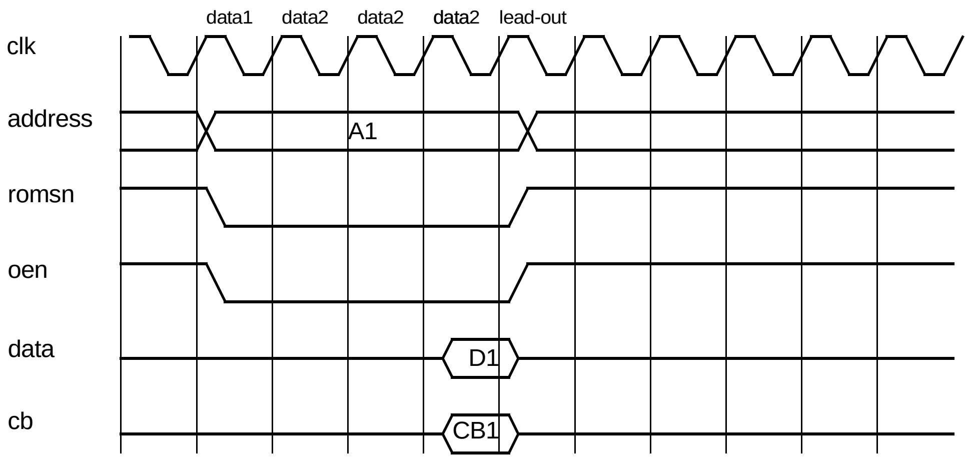


Figure 196. **Prom read access with two waitstates.**

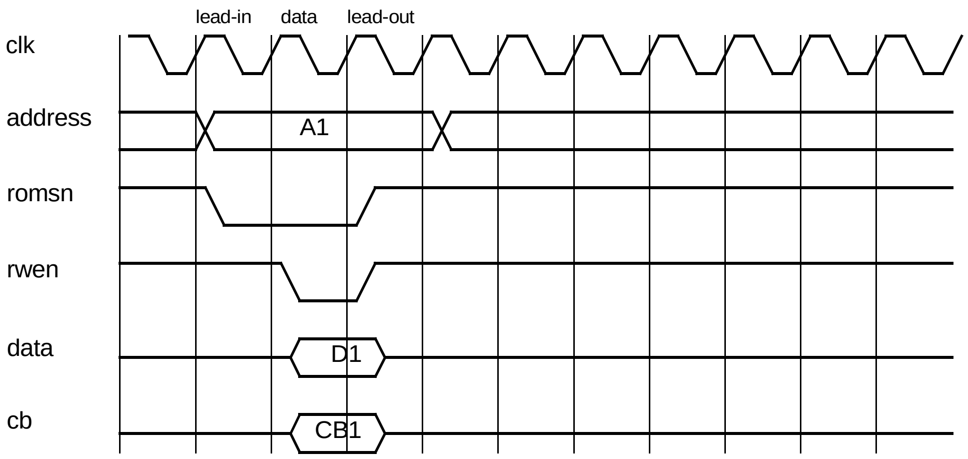


Figure 197. **Prom write cycle (0-waitstates)**

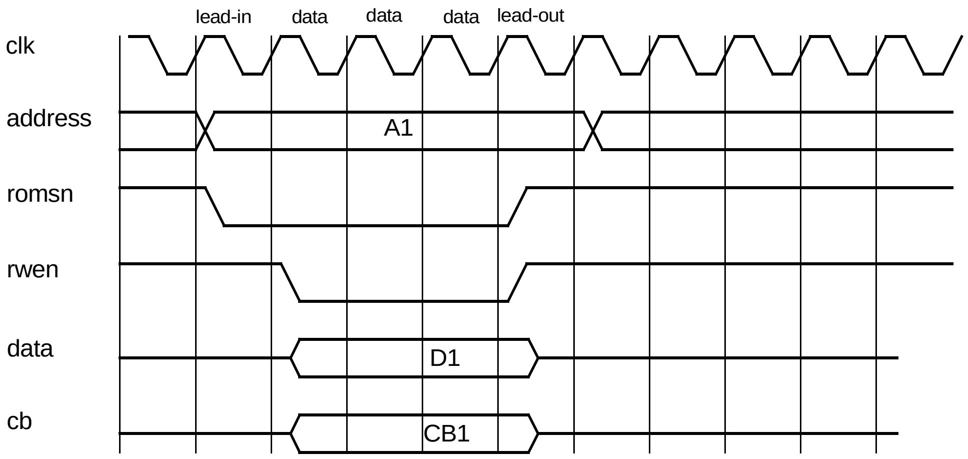


Figure 198. **Prom write cycle (2-waitstates)**

Two PROM chip-select signals are provided, MEMO.ROMSN[1:0]. MEMO.ROMSN[0] is asserted when the lower half of the PROM area as addressed while MEMO.ROMSN[1] is asserted for the upper half. When the VHDL model is configured to boot from internal prom, MEMO.ROMSN[0] is never asserted and all accesses to the lower half of the PROM area are mapped on the internal prom.

55.3 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 MEMI.BRDYN signal. The I/O select signal (MEMO.IOSN) is delayed one clock to provide stable address before MEMO.IOSN is asserted.

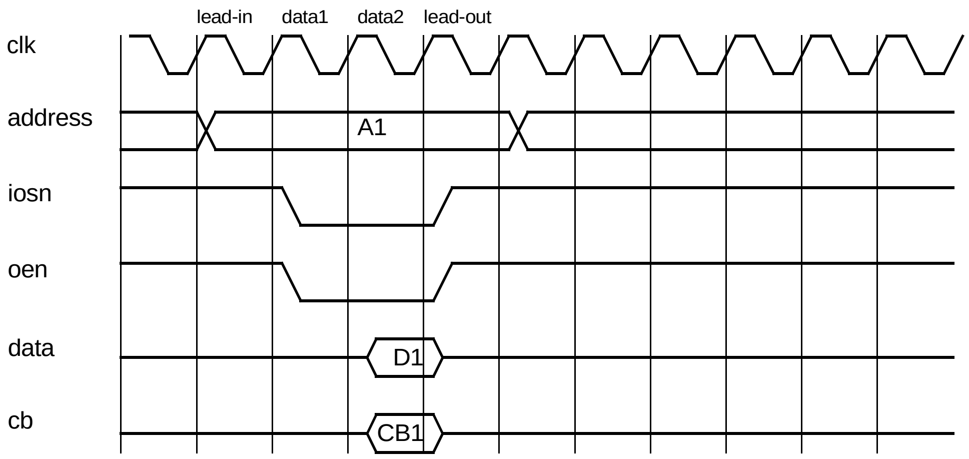


Figure 199. **I/O read cycle (0-waitstates)**

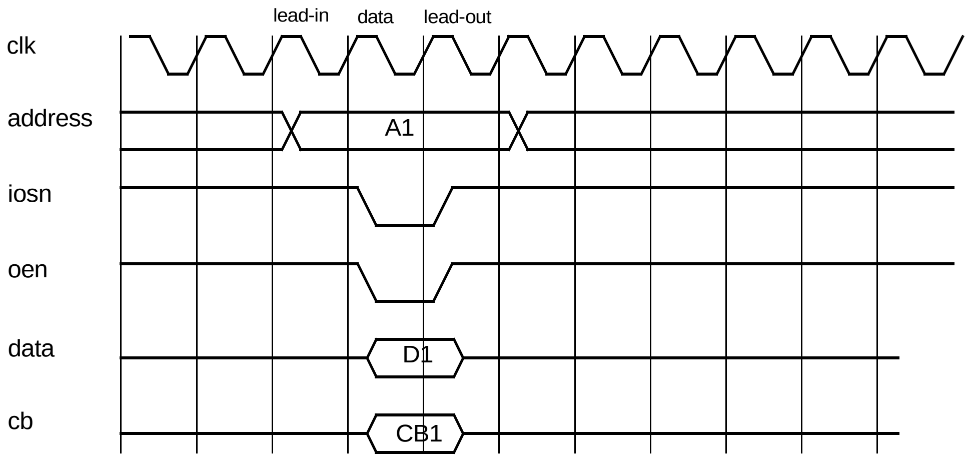


Figure 200. **I/O write cycle (0-waitstates)**

55.4 SRAM access

The SRAM area can be up to 1 Gbyte, divided on up to five RAM banks. The size of banks 1-4 (MEMO.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 and cannot be used simultaneously with SDRAM memory. A read access to SRAM consists of two data cycles and between zero and three waitstates. Accesses to MEMO.RAMSN[4] can further be stretched by de-asserting MEMI.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 201 shows the basic read cycle waveform (zero waitstate).

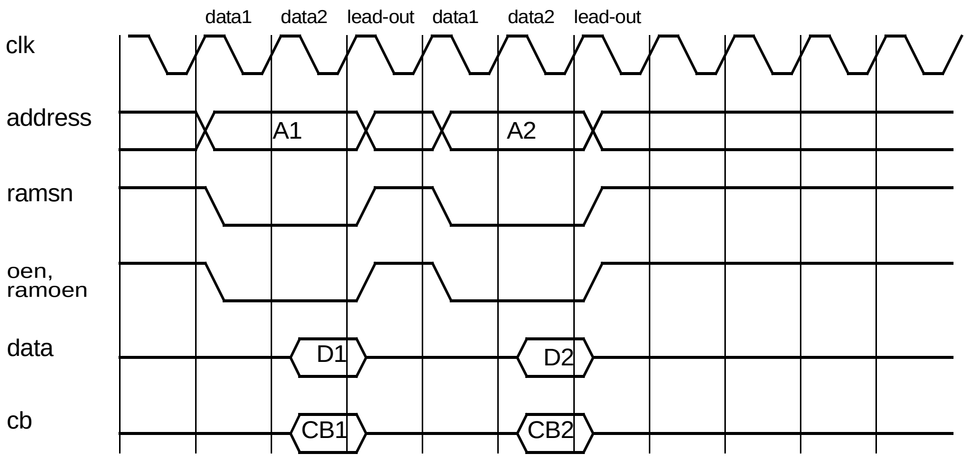


Figure 201. **SRAM non-consecutive read cyclecs.**

For read accesses to MEMO.RAMSN[4:0], a separate output enable signal (MEMO.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 takes a minimum of three cycles:

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 the memory uses a common write strobe for the full 16- or 32-bit data, the read-modify-write bit in the MCFG2 register should be set to enable read-modify-write cycles for sub-word writes.

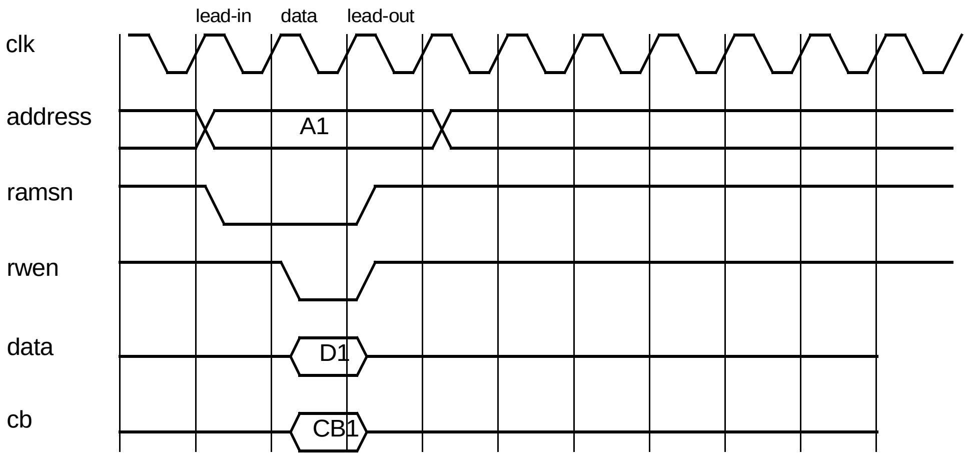


Figure 202. **Sram write cycle (0-waitstates)**

A drive signal vector for the data I/O-pads is provided which has one drive signal for each data bit. It can be used if the synthesis tool does not generate separate registers automatically for the current technology. This can remove timing problems with output delay.

55.5 8-bit and 16-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- or 16-bit operation by programming the ROM and RAM size fields in the memory configuration registers. Since read 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 while access to 16-bit memory will generate a burst of two 16-bits reads. During writes, only the necessary bytes will be writen. Figure 203 shows an interface example with 8-bit PROM and 8-bit SRAM. Figure 204 shows an example of a 16-bit memory interface.

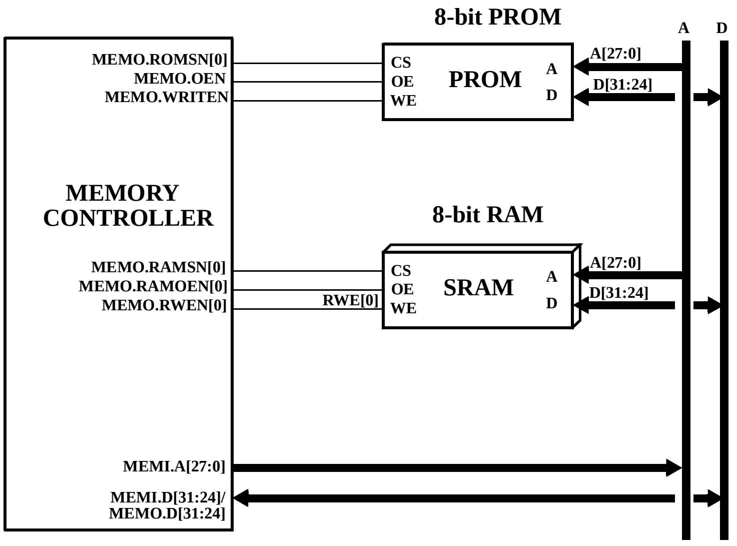


Figure 203. **8-bit memory interface example**

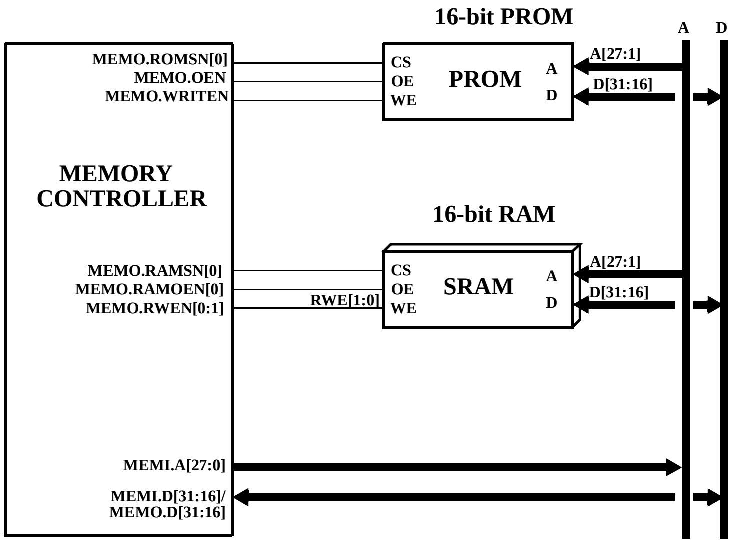


Figure 204. **16-bit memory interface example**

55.6 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.

55.7 8- and 16-bit I/O access

Similar to the PROM/RAM areas, the I/O area can also be configured to 8- or 16-bit mode. However, the I/O device will NOT be accessed by multiple 8/16 bit accesses as the memory areas, but only with one single access just as in 32-bit mode. To access an I/O device on a 16-bit bus, LDUH/STH instructions should be used while LDUB/STB should be used with an 8-bit bus.

55.8 SDRAM access

55.8.1 General

Synchronous dynamic RAM (SDRAM) access is supported to two banks of PC100/PC133 compatible devices. This is implemented by a special version of the SDCTRL SDRAM controller core from Gaisler Research, which is optionally instantiated as a sub-block. The SDRAM controller supports 64M, 256M and 512M devices with 8 - 12 column-address bits, and up to 13 row-address bits. The size 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). Both 32- and 64-bit data bus width is supported, allowing the interface of 64-bit DIMM modules. The memory controller can be configured to use either a shared or separate bus connecting the controller and SDRAM devices. When the VHDL generic **mobile** is set to a value not equal to 0, the controller supports mobile SDRAM.

55.8.2 Address mapping

The two SDRAM chip-select signals are decoded. SDRAM area is mapped into the upper half of the RAM area defined by BAR2 register. When the SDRAM enable bit is set in MCFG2, the controller is enabled and mapped into upper half of the RAM area 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 into the lower half of the RAM area.

55.8.3 Initialisation

When the SDRAM controller is enabled, it automatically performs the SDRAM initialisation sequence of PRECHARGE, 2x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled the initialization sequence is appended by a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read and single location access on write.

55.8.4 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 tabel 627.

***Table 627.*SDRAM programmable timing parameters**

Function	Parameter	Range	Unit
CAS latency, RAS/CAS delay	t_{CAS}, t_{RCD}	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

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

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 628.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	Minimum timing (clocks)
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

55.9 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 µs (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.

55.9.1 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available then the VHDL generic **mobile** >=1.

55.9.2 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available then the VHDL generic **mobile** >=1.

55.9.3 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

55.9.4 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

55.9.5 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

55.9.6 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in MCFG2: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). 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. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. To issue the EMR command, the EMR bit in the MCFG4 register has to be set. 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.

55.9.7 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.

55.9.8 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.

55.9.9 Address bus connection

The memory controller can be configured to either share the address and data buses with the SRAM, or to use separate address and data buses. When the buses are shared, the address bus of the SDRAMs should be connected to A[14:2], the bank address to A[16:15]. The MSB part of A[14:2] can be left unconnected if not used. When separate buses are used, the SDRAM address bus should be connected to SA[12:0] and the bank address to SA[14:13].

55.9.10 Data bus

SDRAM can be connected to the memory controller through the common or separate data bus. If the separate bus is used the width is configurable to 32 or 64 bits. 64-bit data bus allows the 64-bit SDRAM devices to be connected using the full data capacity of the devices. 64-bit SDRAM devices can be connected to 32-bit data bus if 64-bit data bus is not available but in this case only half the full data capacity will be used. There is a drive signal vector and separate data vector available for SDRAM. The drive vector has one drive signal for each data bit. These signals can be used to remove timing problems with the output delay when a separate SDRAM bus is used. SDRAM bus signals are described in section 55.13, for configuration options refer to section 55.15.

55.9.11 Clocking

The SDRAM clock typically requires special synchronisation at layout level. For Xilinx and Altera device, the GR Clock Generator can be configured to produce a properly synchronised SDRAM clock. For other FPGA targets, the GR Clock Generator can produce an inverted clock.

55.10 Using bus ready signalling

The MEMI.BRDYN signal can be used to stretch access cycles to the I/O area and the ram area decoded by MEMO.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 MEMI.BRDYN is asserted. MEMI.BRDYN should be asserted in the cycle preceding the last one. The use of MEMI.BRDYN can be enabled separately for the I/O and RAM areas.

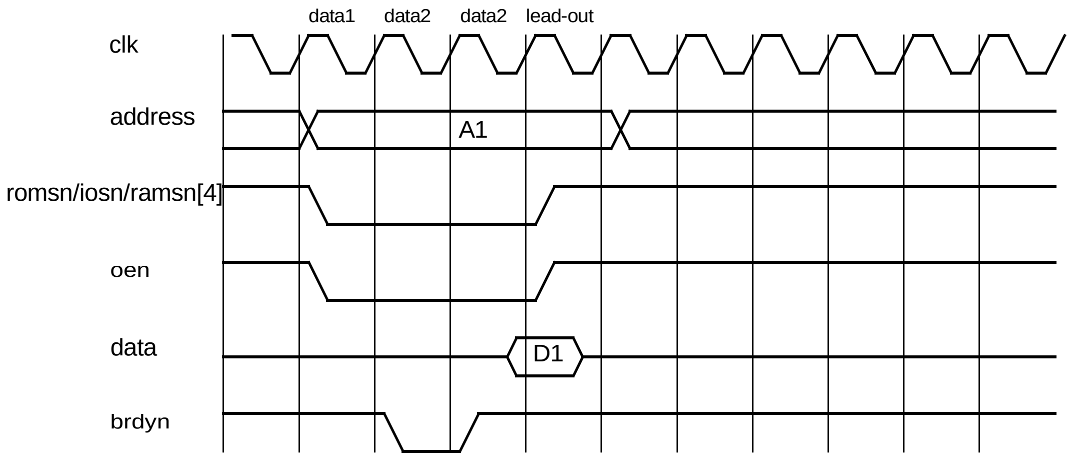


Figure 205. **READ cycle with one extra data2 cycle added with BRDYN (synchronous sampling). Lead-out cycle is only applicable for I/O accesses.**

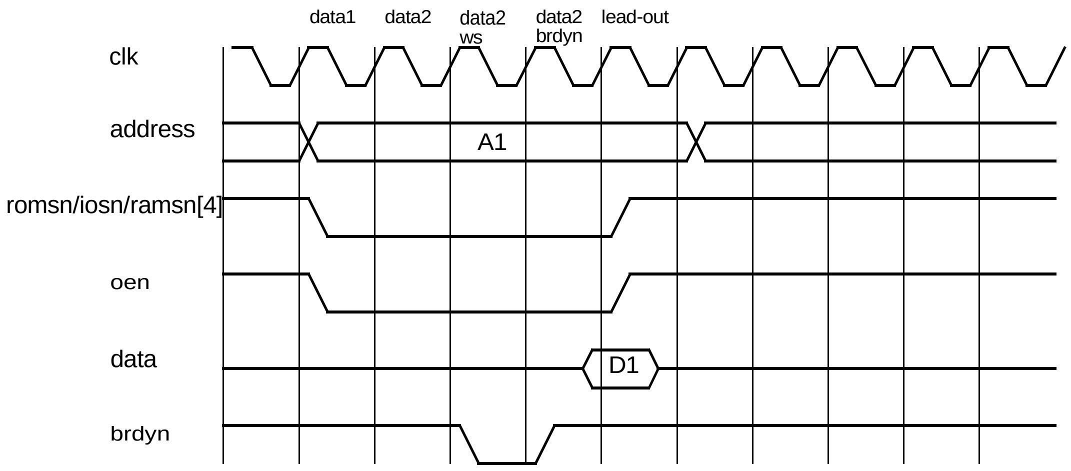


Figure 206. **Read cycle with one waitstate (configured) and one BRDYN generated waitstate (synchronous sampling).**

55.11 Access errors

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

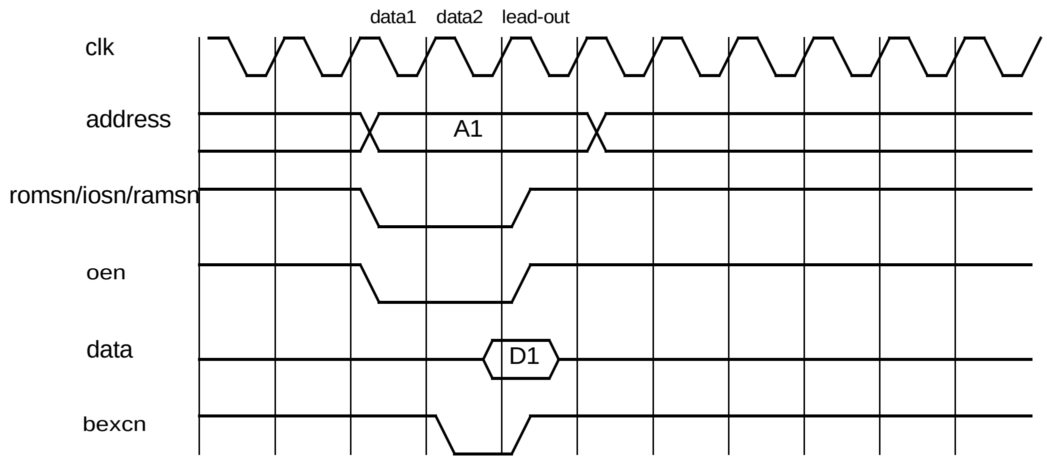


Figure 207. **Read cycle with BEXCN.**

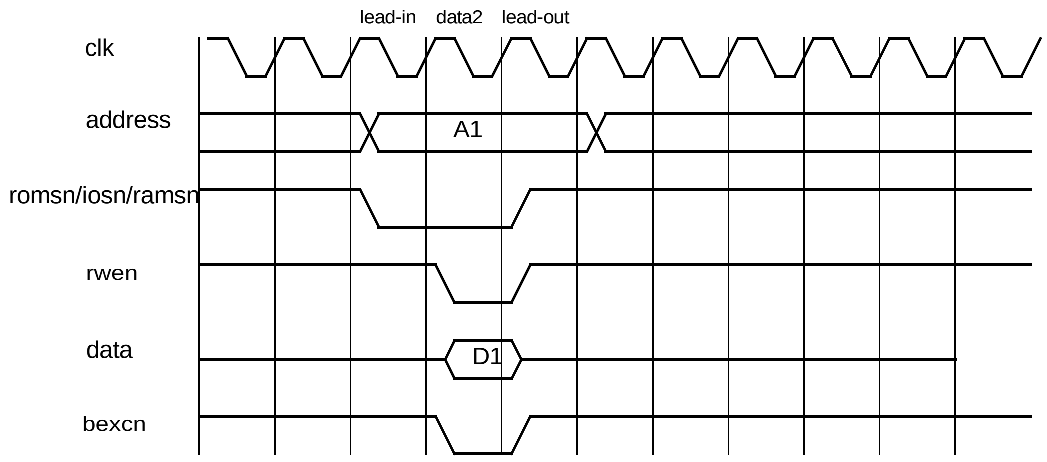


Figure 208. **Write cycle with BEXCN. Chip-select (iosn) is not asserted in lead-in cycle for io-accesses.**

55.12 Attaching an external DRAM controller

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

55.13 Registers

The memory controller is programmed through registers mapped into APB address space.

Table 629.**Memory controller registers**

APB address offset	Register
0x0	MCFG1
0x4	MCFG2
0x8	MCFG3
0xC	MCFG4 (Power-Saving configuration register)

55.13.1 Memory configuration register 1 (MCFG1)

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

Table 630. Memory configuration register 1.

31 29	28 27	26	25	24	23 20	19	18
RESERVED	IOBUSW	IBRDY	BEXCN		IO WAITSTATES	IOEN	

12	11	10	9 8	7 4	3 0
RESERVED	PWEN		PROM WIDTH	PROM WRITE WS	PROM READ WS

31 : 29	RESERVED
28 : 27	I/O bus width (IOBUSW) - Sets the data width of the I/O area (“00”=8, “01”=16, “10” =32).
26	I/O bus ready enable (IBRDY) - Enables bus ready (BRDYN) signalling for the I/O area. Reset to ‘0’.
25	Bus error enable (BEXCN) - Enables bus error signalling. Reset to ‘0’.
24	RESERVED
23 : 20	I/O waitstates (IO WAITSTATES) - Sets the number of waitstates during I/O accesses (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
19	I/O enable (IOEN) - Enables accesses to the memory bus I/O area.
18:12	RESERVED
11	PROM write enable (PWEN) - Enables write cycles to the PROM area.
10	RESERVED
9 : 8	PROM width (PROM WIDTH) - Sets the data width of the PROM area (“00”=8, “01”=16, “10”=32).
7 : 4	PROM write waitstates (PROM WRITE WS) - Sets the number of wait states for PROM write cycles (“0000”=0, “0001”=1, “0010”=2,..., “1111”=15).
3 : 0	PROM read waitstates (PROM READ WS) - Sets the number of wait states for PROM read cycles (“0000”=0, “0001”=1, “0010”=2,...,”1111”=15). Reset to “1111”.

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

55.13.2 Memory configuration register 2 (MCFG2)

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

Table 631. Memory configuration register 2.

31	30	29 28 27	26	25 24 23	22 21	20 19	18	17	16
SDRF	TRP	SDRAM TRFC	TCAS	SDRAM BANKSZ	SDRAM COLSZ	SDRAM CMD	D64	RES	MS

15	14	13	12 9	8	7	6	5 4	3 2	1 0
RES	SE	SI	RAM BANK SIZE		RBRDY	RMW	RAM WIDTH	RAM WRITE WS	RAM READ WS

31	SDRAM refresh (SDRF) - Enables SDRAM refresh.
30	SDRAM TRP parameter (TRP) - t_{RP} will be equal to 2 or 3 system clocks (0/1).
29 : 27	SDRAM TRFC parameter (SDRAM TRFC) - t_{RFC} will be equal to 3+field-value system clocks.
26	SDRAM TCAS parameter (TCAS) - Selects 2 or 3 cycle CAS delay (0/1). When changed, a LOAD-COMMAND-REGISTER command must be issued at the same time. Also sets RAS/CAS delay (t_{RCD}).
25 : 23	SDRAM bank size (SDRAM BANKSZ) - Sets the bank size for SDRAM chip selects (“000”=4 Mbyte, “001”=8 Mbyte, “010”=16 Mbyte.... “111”=512 Mbyte).
22 : 21	SDRAM column size (SDRAM COLSZ) - “00”=256, “01”=512, “10”=1024, “11”=4096 when bit 25:23=”111” 2048 otherwise.

Table 631. Memory configuration register 2.

20 : 19	SDRAM command (SDRAM CMD) - Writing a non-zero value will generate a SDRAM command. “01”=PRECHARGE, “10”=AUTO-REFRESH, “11”=LOAD-COMMAND-REGISTER. The field is reset after the command has been executed.
18	64-bit SDRAM data bus (D64) - Reads ‘1’ if the memory controller is configured for 64-bit SDRAM data bus width, ‘0’ otherwise. Read-only.
17	RESERVED
16	Mobile SDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
15	RESERVED
14	SDRAM enable (SE) - Enables the SDRAM controller.
13	SRAM disable (SI) - Disables accesses RAM if bit 14 (SE) is set to ‘1’.
12 : 9	RAM bank size (RAM BANK SIZE) - Sets the size of each RAM bank (“0000”=8 kbyte, “0001”=16 kbyte, ..., “1111”=256 Mbyte).
8	RESERVED
7	RAM bus ready enable (RBRDY) - Enables bus ready signalling for the RAM area.
6	Read-modify-write enable (RMW) - Enables read-modify-write cycles for sub-word writes to 16- bit 32-bit areas with common write strobe (no byte write strobe).
5 : 4	RAM width (RAM WIDTH) - Sets the data width of the RAM area (“00”=8, “01”=16, “1X”=32).
3 : 2	RAM write waitstates (RAM WRITE WS) - Sets the number of wait states for RAM write cycles (“00”=0, “01”=1, “10”=2, “11”=3).
1 : 0	RAM read waitstates (RAM READ WS) - Sets the number of wait states for RAM read cycles (“00”=0, “01”=1, “10”=2, “11”=3).

55.13.3 Memory configuration register 3 (MCFG3)

MCFG3 is contains the reload value for the SDRAM refresh counter.

31 30 29 28 27	26 25 24 23 22 21 20 19 18 17 16 15 14 13 12	11 10 9 8 7 6 5 4 3 2 1 0
RESERVED	SDRAM REFRESH RELOAD VALUE	RESERVED

31: 27	RESERVED
26: 12	SDRAM refresh counter reload value (SDRAM REFRESH RELOAD VALUE)
11: 0	RESERVED

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

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

Table 632. MCFG4 Power-Saving configuration register

31	30	29	28 24	23 20	19	18 16	15 7	6 5	4 3	2 0
ME	CE	EM	Reserved	tXSR	res	PMODE	Reserved	DS	TCSR	PASR

31	Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
29	EMR. When set, the LOAD-COMMAND-REGISTER command issued by the SDRAM command filed in MCFG2 will be interpret as a LOAD-EXTENDED-COMMAND-REGISTER command.
28: 24	Reserved

Table 632. MCFG4 Power-Saving configuration register

23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved
18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

55.14 Vendor and device identifiers

The core has vendor identifier 0x04 (ESA) and device identifier 0x00F. For description of vendor and device identifier see GRLIB IP Library User’s Manual.

55.15 Configuration options

Table 633 shows the configuration options of the core (VHDL generics).

Table 633.**Configuration options**

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0x1FFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#E00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space. Default I/O area is 0x20000000 - 0x2FFFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#E00#
ramaddr	ADDR field of the AHB BAR2 defining RAM address space. Default RAM area is 0x40000000-0x7FFFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR2 defining RAM address space.	0 -16#FFF#	16#C00#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
wprot	RAM write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
fast	Enable fast SDRAM address decoding.	0 - 1	0
romasel	log2(PROM address space size) - 1. E.g. if size of the PROM area is 0x20000000 romasel is log2(2^29)-1 = 28.	0 - 31	28
sdrasel	log2(RAM address space size) - 1. E.g if size of the RAM address space is 0x40000000 sdrasel is log2(2^30)-1= 29.	0 - 31	29
srbanks	Number of SRAM banks.	0 - 5	4
ram8	Enable 8-bit PROM and SRAM access.	0 - 1	0
ram16	Enable 16-bit PROM and SRAM access.	0 - 1	0
sden	Enable SDRAM controller.	0 - 1	0
sepbus	SDRAM is located on separate bus.	0 - 1	1
sdbits	32 or 64 -bit SDRAM data bus.	32, 64	32
oepol	Select polarity of drive signals for data pads. 0 = active low, 1 = active high.	0 - 1	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

55.16 Signal descriptions

Table 634 shows the interface signals of the core (VHDL ports).

Table 634.**Signal descriptions**

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low

Table 634. **Signal descriptions**

Signal name	Field	Type	Function	Active
MEMI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Bus ready strobe	Low
	BEXCN	Input	Bus exception	Low
	WRN[3:0]	Input	SRAM write enable feedback signal	Low
	BWIDTH[1:0]	Input	Sets the reset value of the PROM data bus width field in the MCFG1 register	High
	SD[31:0]	Input	SDRAM separate data bus	High
MEMO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	-
	SDDATA[63:0]	Output	Sdram memory data	-
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Local I/O select	Low
	ROMSN[1:0]	Output	PROM chip-select	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus.Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0].	Low/High
	VBDRIVE[31:0]	Output	Vectored I/O-pad drive signals.	Low/High
	SVBDRIVE[63:0]	Output	Vectored I/O-pad drive signals for separate sdram bus.	Low/High
	READ	Output	Read strobe	High
	SA[14:0]	Output	SDRAM separate address bus	High
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
WPROT	WPROTHIT	Input	Unused	-

*Table 634.***Signal descriptions**

Signal name	Field	Type	Function	Active
SDO	SDCASN	Output	SDRAM column address strobe	Low
	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDDQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	SDRASN	Output	SDRAM row address strobe	Low
	SDWEN	Output	SDRAM write enable	Low

* see GRLIB IP Library User’s Manual

55.17 Library dependencies

Table 635 shows libraries used when instantiating the core (VHDL libraries).

*Table 635.***Library dependencies**

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals Components	Memory bus signals definitions SDMCTRL component
ESA	MEMORYCTRL	Component	Memory controller component declaration

55.18 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0x1FFFFFFF, I/O-area is 0x20000000-0x3FFFFFFF and RAM area is 0x40000000 - 0x7FFFFFFF. SDRAM controller is enabled. SDRAM clock is synchronized with system clock by clock generator.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
library esa;
use esa.memoryctrl.all;

entity mctrl_ex is
```

```
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdram_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  -- Memory controller
  mctrl0 : mctrl generic map (srbanks => 1, sden => 1)
    port map (rstn, clkm, memi, memo, ahbsi, ahbso(0), apbi, apbo(0), wprot, sdo);

  -- memory controller inputs not used in this configuration
  memi.brdyn <= ’1’; memi.bexcn <= ’1’; memi.wrn <= "1111";
```

```
    memi.sd <= sd;

    -- prom width at reset
    memi.bwidth <= "10";

    -- I/O pads driving data memory bus data signals
    datapads : for i in 0 to 3 generate
        data_pad : iopadv generic map (width => 8)
        port map (pad => data(31-i*8 downto 24-i*8),
                  o => memi.data(31-i*8 downto 24-i*8),
                  en => memo.bdrive(i),
                  i => memo.data(31-i*8 downto 24-i*8));
    end generate;

    -- connect memory controller outputs to entity output signals
    address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
    oen <= memo.oen; rwen <= memo.wrn; ramoen <= "1111" & memo.ramoen(0);
    sa <= memo.sa;
    writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
    sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
    sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;
  end;
```

56 MUL32 - Signed/unsigned 32x32 multiplier module

56.1 Overview

The multiplier module is highly configurable module implementing 32x32 bit multiplier. Multiplier takes two signed or unsigned numbers as input and produces 64-bit result. Multiplication latency and hardware complexity depend on multiplier configuration. Variety of configuration option makes it possible to configure the multiplier to meet wide range of requirements on complexity and performance.

For DSP applications the module can be configured to perform multiply & accumulate (MAC) operation. In this configuration 16x16 multiplication is performed and the 32-bit result is added to 40-bit value accumulator.

56.2 Operation

The multiplication is started when ‘1’ is samples on MULI.START on positive clock edge. Operands are latched externally and provided on inputs MULI.OP1 and MULI.OP2 during the whole operation. The result appears on the outputs during the clock cycle following the clock cycle when MULO.READY is asserted if multiplier if 16x16, 32x8 or 32x16 configuration is used. For 32x32 configuration result appears on the output during the second clock cycle after the MULI.START was asserted.

Signal MULI.MAC shall be asserted to start multiply & accumulate (MAC) operation. This signal is latched on positive clock edge. Multiplication is performed between two 16-bit values on inputs MULI.OP1[15:0] and MULI.OP2[15:0]. The 32-bit result of the multiplication is added to the 40-bit accumulator value on signal MULI.ACC to form a 40-bit value on output MULO.RESULT[39:0]. The result of MAC operation appears during the second clock cycle after the MULI.MAC was asserted.

56.3 Synthesis

Table 636 shows hardware complexity in ASIC gates and latency for different multiplier configurations.

*Table 636.*Multiplier latencies and hardware complexity

Multiplier size (multype)	**Pipelined (pipe)**	**Latency (clocks)**	**Approximate area (gates)**
16x16	1	5	6 500
16x16	0	4	6 000
32x8	-	4	5 000
32x16	-	2	9 000
32x32	-	1	15 000

56.4 Configuration options

Table 637 shows the configuration options of the core (VHDL generics).

*Table 637.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
infer	If set the multipliers will be inferred by the synthesis tool. Use this option if your synthesis tool i capable of inferring efficient multiplier implementation.	0 to 1	1
multype	Size of the multiplier that is actually implemented. All configuration produce 64-bit result with different latencies. 0 - 16x16 bit multiplier 1 - 32x8 bit multiplier 2 - 32x16 bit multiplier 3 - 32x32 bit multiplier	0 to 3	0
pipe	Used in 16x16 bit multiplier configuration with inferred option enabled. Adds a pipeline register stage to the multiplier. This option gives better timing but adds one clock cycle to latency.	0 to 1	0
mac	Enable multiply & accumulate operation. Use only with 16x16 multiplier option with no pipelining (*pipe* = 0)	0 to 1	0

56.5 Signal descriptions

Table 638 shows the interface signals of the core (VHDL ports).

*Table 638.*Signal declarations

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
HOLDN	N/A	Input	Hold	Low
MULI	OP1[32:0]	Input	Operand 1 OP1[32] - Sign bit. OP1[31:0] - Operand 1 in 2's complement format	High
	OP2[32:0]		Operand 2 OP2[32] - Sign bit. OP2[31:0] - Operand 2in 2's complement format	High
	FLUSH		Flush current operation	High
	SIGNED		Signed multiplication	High
	START		Start multiplication	High
	MAC		Multiply & accumulate	High
	ACC[39:0]		Accumulator. Accumulator value is held externally.	High
MULO	READY	Output	Result is ready during the next clock cycle for 16x16, 32x8 and 32x16 configurations. Not used for 32x32 configuration or MAC operation.	High
	NREADY		Not used	-
	ICC[3:0]		Condition codes ICC[3] - Negative result (not used in 32x32 conf) ICC[1] - Zero result (not used in 32x32 conf) ICC[1:0] - Not used	High
	RESULT[63:0]		Result. Available at the end of the clock cycle if MULO.READY was asserted in previous clock cycle. For 32x32 configuration the result is available during second clock cycle after the MULI.START was asserted.	High

56.6 Library dependencies

Table 639 shows the libraries used when instantiating the core (VHDL libraries).

*Table 639.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	ARITH	Signals, component	Signals, component declaration

56.7 Component declaration

The core has the following component declaration.

```
component mul32
generic (
    infer   : integer := 1;
    multype : integer := 0;
    pipe    : integer := 0;
    mac     : integer := 0
);
port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    holdn   : in  std_ulogic;
    muli    : in  mul32_in_type;
    mulo    : out mul32_out_type
);
end component;
```

56.8 Instantiation

This example shows how the core can be instantiated.

The module is configured to implement 16x16 pipelined multiplier with support for MAC operations.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use gaisler.arith.all;

.
.
.

signal muli  : mul32_in_type;
signal mulo  : mul32_out_type;

begin

mul0 : mul32 generic map (infer => 1, multype => 0, pipe => 1, mac => 1)
    port map (rst, clk, holdn, muli, mulo);


end;
```

57 MULTLIB - High-performance multipliers

57.1 Overview

The GRLIB.MULTLIB VHDL-library contains a collection of high-performance multipliers from the Arithmetic Module Generator at Norwegian University of Science and Technology. 32x32, 32x8, 32x16, 16x16 unsigned/signed multipliers are included. 16x16-bit multiplier can be configured to include a pipeline stage. This option improves timing but increases latency with one clock cycle.

57.2 Configuration options

Table 640 shows the configuration options of the core (VHDL generics).

*Table 640.*Configuration options

Generic	Function	Allowed range	Default
mulpipe	Include a pipeline stage (0 -pipelining disabled, 1 - pipelining enabled)	0 - 1	0

57.3 Signal descriptions

Table 641 shows the interface signals of the core (VHDL ports).

*Table 641.*Signal descriptions

Signal name	Type	Function	Active
CLK (16x16 multiplier only)	Input	Clock	-
HOLDN (16x16 multiplier only)	Input	Hold. When active, the pipeline register is not updates	Low
X[16:0] (16x16 mult) X[32:0] (32x8 mult) X[32:0] (32x16 mult) X[32:0] (32x32 mult)	Input	Operand 1. MBS bit is sign bit.	High
Y[16:0] (16x16 mult) Y[8:0] (32x8 mult) Y[16:0] (32x16 mult) Y[32:0] (32x32 mult)	Input	Operand 2. MSB bit is sign bit.	High
P[33:0] (16x16 mult) P[41:0] (32x8 mult) P[49:0] (32x16 mult) P[65:0] (32x32 mult)		Result. Two MSB bits are sign bits.	High

57.4 Library dependencies

Table 642 shows libraries used when instantiating the core (VHDL libraries).

*Table 642.*Library dependencies

Library	Package	Imported unit	Description
GRLIB	MULTLIB	Component	Multiplier component declarations

57.5 Component declaration

The core has the following component declaration.

```
component mul_33_33
  port (
    x    : in  std_logic_vector(32 downto 0);
    y    : in  std_logic_vector(32 downto 0);
    p    : out std_logic_vector(65 downto 0)
  );
end component;
```

57.6 Instantiation

This example shows how the core can be instantiated.

The core is configured to implement 16x16 pipelined multiplier with support for MAC operations.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.multlib.all;

.
.

signal op1, op2 : std_logic_vector(32 downto 0);
signal prod : std_logic_vector(65 downto 0);

begin

m0 : mul_33_33
         port map (op1, op2, prod);

end;
```

58 GRPCI - 32-bit PCI Master/Target with configurable FIFOs and AHB back end

58.1 Overview

The PCI Master/Target is a bridge between the PCI bus and the AMBA AHB bus. The core is connected to the PCI bus through two interfaces, a target and master. The PCI master interface is optional and can be disabled through a VHDL generic. The AHB side of the core uses one slave interface and one master interface. Configuration registers are available through the AMBA APB bus.

The PCI and AMBA interfaces belong to two different clock domains. Synchronization is performed inside the core through FIFOs with configurable depth.

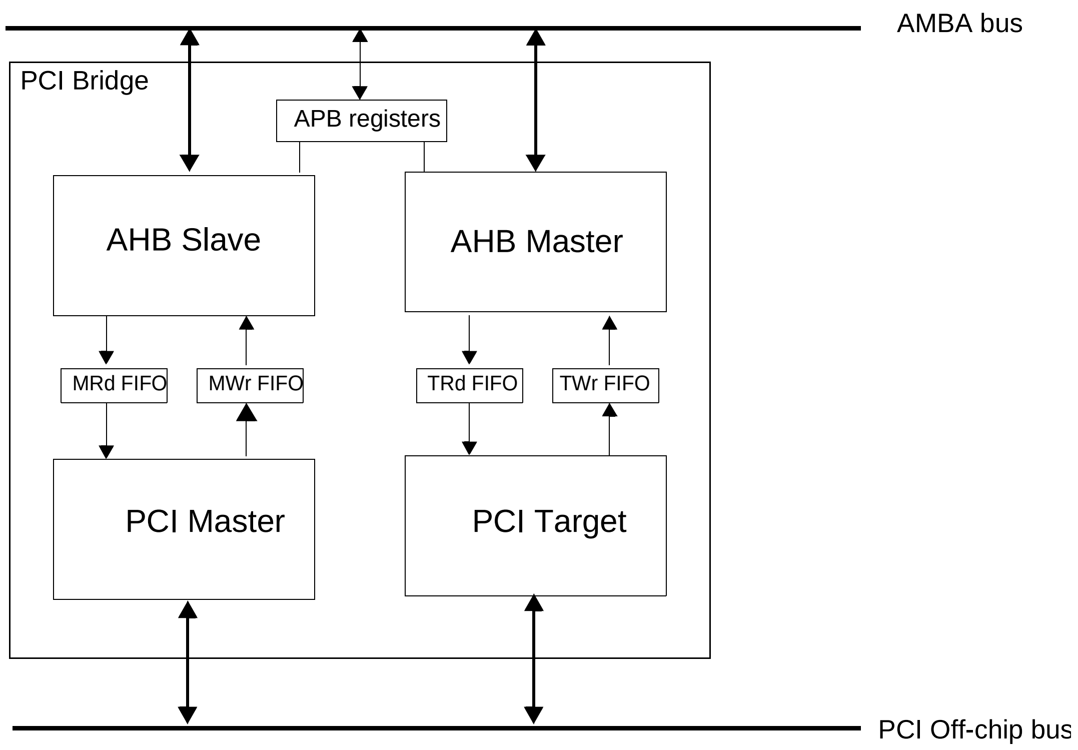


Figure 209. GRPCI master/target

A summary of the GRPCI key features:

- 32-bit PCI interface
- PCI bus master and target
- AMBA AHB/APB 2.0 back end interface
- Configurable FIFOs for both master and target operation
- Supports incremental bursts and single accesses
- Bus master capabilities:

  o Memory read, memory write

  o Memory read multiple

  o Memory read line

  o I/O read, I/O write

  o Type 0 and 1 configuration read and write

  - o Host bridging
- Target capabilities:
  - o Type 0 configuration space header
  - o Configuration read and write
  - o Parity generation (PAR), Parity error detection (PERR, SERR)
  - o 2 Memory BARs
  - o Memory read, memory write
  - o Memory read multiple
  - o Memory read line
  - o Memory write and invalidate
- Optional DMA engine add on (see PCIDMA IP core)

58.2 Operation

PCI transactions are initiated by accessing the GRPCI AHB slave. The AHB slave has one memory bank of configurable size (128 MB - 2 GB) and one 128 KB IO bank. Accesses to the memory bank are translated into PCI memory cycles while accesses to the lower half of the IO bank are translated into IO cycles. Configuration cycles are generated through accesses to the upper half of the IO bank. The AHB slave supports 8/16/32-bit single accesses and 32-bit bursts. The address translation from AHB to PCI is determined by the memory map register and the IO map register.

A connection from the PCI bus to the AHB bus is provided by the core’s PCI target interface and AHB master. The PCI target is capable of handling configuration cycles, 8/16/32-bit single access memory cycles and burst memory cycles of any alignment on the PCI bus. Configuration cycles are used to access the PCI targets configuration space while the memory cycles are translated to AHB accesses. The PCI target provides two memory space Base Address Registers (BARs) of configurable size and can thus occupy two areas in the PCI memory space. Each BAR is associated with a PAGE register which determines the address translation from PCI to AHB.

For burst transactions the data is buffered internally in FIFOs with configurable size. For more information about the flow of data through the FIFOs and how it affects the operation see section 58.8.

Since PCI is little endian and LEON3 big endian GRPCI defaults to performing byte twisting on all accesses to preserve the byte ordering. See 58.7 for more information.

58.3 PCI target interface

The PCI target interface occupies two memory areas in the PCI address space and the memory mapping is determined by the BAR0 and BAR1 registers in the targets configurations space. The size of the PCI memory areas is determined by number of bits actually implemented by the BAR registers (configurable through *abits* and *dmaabits* VHDL-generics).

58.3.1 PCI commands

The GRPCI target interface handles the following PCI commands:

- **Configuration Read/Write:** Single access to the configuration space. No AHB access is performed.
- **Memory Read:** If prefetching is enabled through the *readpref* generic (it is disabled per default), the units AHB master interface fetches a cache line, otherwise a single AHB access is performed.

- **Memory Read Line:** The unit prefetches data according to the value of the cache line size register.
- **Memory Read Multiple:** The unit performs maximum prefetching. This can cause long response time, depending of the user defined FIFO depth.
- **Memory Write, Memory Write and Invalidate:** Handled similarly.

58.3.2 PCI responses

The target interface can terminate a PCI transaction with one of the following responses:

- **Retry:** This response indicates that the master should perform the same request later as the target is temporarily busy. This response is always given at least one time for read accesses, but can also occur for write accesses.
- **Disconnect with data:** Terminate the transaction and transfer data in the current data phase. This occurs if a master tries to transfer more data that fits in the FIFO or if prefetching is disabled and a Memory Read command is performed.
- **Disconnect without data:** Terminate the transaction without transfering data in the current data phase. This can occur during a PCI read burst if the PCI target is forced to insert more than 8 wait states while waiting for the AHB master to complete.
- **Target-Abort:** Indicates that the current access caused an internal error, and the target never will be able to finish it.

An AHB transaction with ‘retry’ responses is repeated by the AHB master until an ‘ok’ or ‘error’ response is received. The ‘error’ response on AHB bus will result in ‘target abort’ response for the PCI memory read cycle. In case of PCI memory write cycle, AHB access will not be finished with error response since write data is posted to the destination unit. Instead the write error (WE) bit will be set in the APB configuration/status register.

58.3.3 Bursts and byte enables

The target is capable of handling burst transactions. A PCI burst crossing 1 kB address boundary will be performed as multiple AHB bursts by the AHB master interface. The AHB master interface will insert an idle-cycle before requesting a new AHB burst to allow re-arbitration on the AHB.

A PCI master is allowed to change the byte enables on any data phase during a transaction. The GRPCI core handles this but each non 32-bit access in a PCI write burst will be translated into an AHB single access of the corresponding size by the AHB master. For PCI reads the byte enables are ignored by the PCI target and all byte lanes are driven with valid data.

58.3.4 Configuration Space

The following registers are implemented in the GRPCI configuration space. Please refer to the PCI specification for more information than is given here.

*Table 643.*Configuration Space Header registers

Address offset	**Register**
0x00	Device ID, Vendor ID
0x04	Status, Command
0x08	Class Code & Revision ID
0x0C	BIST, Header Type, Latency Timer, Cache Line Size
0x10	BAR0
0x14	BAR1
0x3C	Max_lat, min_gnt, interrupt pin, interrupt line.

31 16	15 0
DEVICE_ID	VENDOR_ID

Figure 210. Device ID & Vendor ID register

[31:16]: Device ID (read-only). Returns value of *device_id* VHDL-generic.
[15:0]: Vendor ID (read-only). Returns value of *vendor_id* VHDL-generic.

31	30	29	28	27	26 25	24	23	22	21	20 16	15 10	9	8	7	6	5	4	3	2	1	0
DPE	‘0’	RMA	RTA	STA	‘10’	DPD	‘0’	‘0’	‘0’	RESERVED	RESERVED	‘0’	‘0’	‘0’	PER	‘0’	MIE	‘0’	BM	MS	‘0’

Figure 211. Status & Command register

[31:16]: Status Register - Writing one to a bit 31 - 16 clears the bit. Writes can not set a bit.
[31]: Detected parity Error (DPE).
[30]: Signalled System Error (SSE) - Not implemented. Always reads 0.
[29]: Received Master Abort (RMA) - Set by the PCI Master interface when its transaction is terminated with Master-Abort.
[28]: Received Target Abort (RTA) - Set by the PCI Master interface when its transaction is terminated with Target-Abort.
[27]: Signalled Target Abort (STA) - Set by the PCI Target Interface when the target terminates transaction with Target-Abort.
[26:25]: DEVSEL timing (DST) -Always reads “10” - medium DEVSEL timing.
[24]: Data Parity Error Detected (DPD).
[23]: Fast Back-to-Back Capable - The Target interface is not capable of fast back-to-back transactions. Always reads ‘0’.
[22]: UDF Supported - Not supported. Always reads ‘0’,
[21]: 66 Mhz Capable - Not supported. Always reads ‘0’.
[20:16]: Reserved. Always reads ‘00..0’.
[15:0]: Command Register - Writing one to an implemented bit sets the bit. Writing zero clears the bit.
[15:10]: Reserved - Always reads as ‘00..0’.
[9]: Fast back-to-Back Enable - Not implemented. Always reads ‘0’.
[8]: SERR# enable - Not implemented. Always reads ‘0’.
[7]: Wait cycle control - Not implemented. Always reads ‘0’.
[6]: Parity Error Response (PER) - Controls units response on parity error.
[5]: VGA Palette Snoop - Not implemented. Always reads ‘0’.

[4]: Memory Write and Invalidate Enable (MIE) - Enables the PCI Master interface to generate Memory Write and Invalidate Command.

[3]: Special Cycles - Not implemented. Always reads ‘0’.

[2]: Bus Master (BM) - Enables the Master Interface to generate PCI cycles.

[1]: Memory Space (MS) - Allows the unit to respond to Memory space accesses.

[0]: I/O Space (IOS) - The unit never responds to I/O cycles. Always reads as ‘0’.

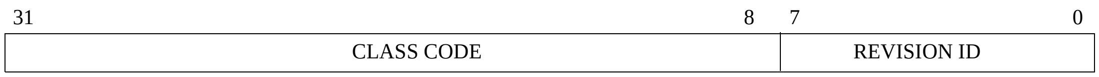


Figure 212. Class Code & revision ID

[31:8]: Class Code - Processor device class code. Set with *class_code* generic (read-only).

[7:0]: Revision ID - Set with *rev* generic (read-only).

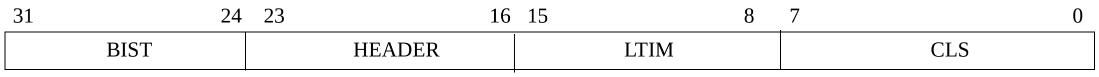


Figure 213. BIST, Header Type, Latency Timer and Cache Line Size register

[31:24]: BIST - Not supported. Reads always as ‘00..0’.

[23:16]: Header Type (HEADER)- Header Type 0. Reads always as ‘00..0’.

[15:8]: Latency Timer (LTIM) - Maximum number of PCI clock cycles that Master can own the bus.

[7:0]: Cache Line Size (CLS) - System cache line size. Defines the prefetch length for ‘Memory Read’ and ‘Memory Read Line’ commands.

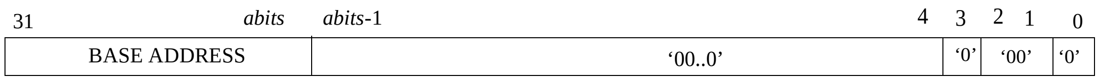


Figure 214. BAR0 register

[31:*abits*]: PCI Base Address - PCI Targets interface Base Address 0. The number of implemented bits depend on the VHDL-generic *abits*. Memory area of size 2^*abits* bytes at Base Address is occupied through this Base Address register. Register PAGE0 is accessed through upper half of this area. PCI memory accesses to the lower half of this area is translated to AHB accesses using PAGE0 map register.

[*abits*-1:4]: These bits are read-only and always read as ‘00..0’. This field can be used to determine devices memory requirement by writing value of all ones to this register and reading the value back. The device will return zeroes in unimplemented bits positions effectively defining memory area requested.

[3]: Prefetchable: Not supported. Always reads ‘0’.

[2:1]: Base Address Type - Mapping can be done anywhere in the 32-bit memory space. Reads always as ‘00’.

[0]: Memory Space Indicator - Register maps into Memory space. Read always as ‘0’.

PAGE0 register is mapped into upper half of the PCI address space defined by BAR0 register.

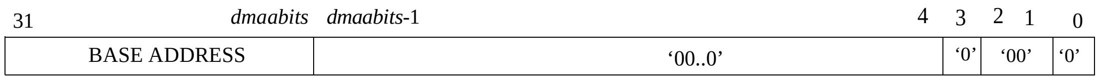


Figure 215. BAR1 register

[31:*dmaabits*]: PCI Base Address - PCI Targets interface Base Address 1. The number of implemented bits depends on the VHDL-generic *dmaabits*. Memory area of size 2^*dmaabits* bytes at Base Address is occupied through this Base Address register. PCI memory accesses to this memory space are translated to AHB accesses using PAGE1 map register.

[*dmaabits*-1:4]: These bits are read-only and always read as ‘00..0’. This field can be used to determine devices memory requirement by writing value of all ones to this register and reading the value back. The device will return zeroes in unimplemented bits positions effectively defining memory area requested.

[3]: Prefetchable: Not supported. Always reads as ‘0’.

[2:1]: Base Address Type - Mapping can be done anywhere in the 32-bit memory space. Reads always as ‘00’.

[0]: Memory Space Indicator - Register maps in Memory space. Read always as ‘0’.

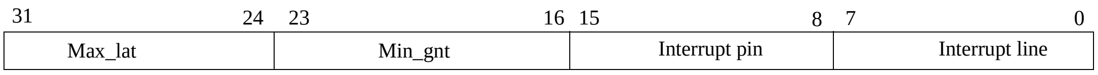


Figure 216. Max_lat, min_gnt and interrupt settings

[31-24]: Max_lat. Reads 0.

[23-16]: Min_gnt. Reads 2^(*fifodepth-3).*

[15:8]: Interrupt pin. Always reads 1, indicating that the device uses PCI interrupt pin A. If used the interrupt must be driven from outside the GRPCI core.

[7-0]: Interrupt line. Write able register used by the operating system to store information about interrupt routing.

58.3.5 The PAGE0/1 map registers

The PAGE0 and PAGE1 registers are used to translate PCI addresses to AHB addresses for BAR0 and BAR1 respectively. PAGE0 is mapped into the PCI address space defined by BAR0, while PAGE1 is an APB register.

*Table 644.*PCI target map registers

Register	Address	Address space
PAGE0	Upper half of PCI address space defined by BAR0 register	PCI
PAGE1	APB base address + 0x10	APB

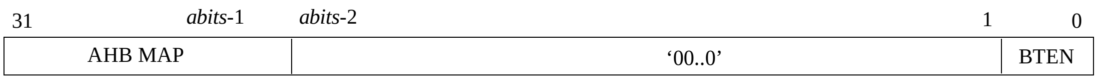


Figure 217. PAGE0 register

[31:*abits*-1]: AHB Map Address - Maps PCI accesses to PCI BAR0 address space to AHB address space. AHB address is formed by concatenating AHB MAP with LSB of the PCI address.

[*abits*-2:1]: Reserved. Reads always as ‘00..0’.

[0]: BTEN - Byte twisting enabled if ‘1’. Reset value ‘1’. May only be altered when bus mastering is disabled.

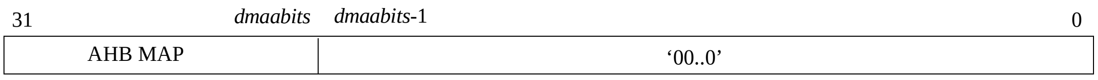


Figure 218. PAGE1 register

[31:*dmaabits*]: AHB Map Address (AHB MAP) - Maps PCI accesses to PCI BAR1 address space to AHB address space. AHB address is formed by concatenating AHB MAP with LSB of the PCI address.

[*dmaabits*-1:0]: Reserved. Reads always as ‘00..0’.

Note that it is possible to set the PAGE1 register from PCI by configuring PAGE0 to point to the APB base address and then writing to BAR0 + GRPCI_APB_OFFSET + 0x10.

58.3.6 Calculating the address of PAGE0

Since the size of BAR0 is configurable the address of the PAGE0 register is not constant between designs. It is possible to calculate the address of PAGE0 in a simple manner in order to write code that is portable between devices with differently sized BAR0 registers. This is shown below using C syntax.

```
/* Save original BAR0 (address 0x10 in the conf. space) */
pci_read_config_dword(bus,slot,function,0x10, &tmp);

/* Write 0xffffffff to BAR0 */
pci_write_config_dword(bus, slot, function, 0x10, 0xffffffff);

/* Read it back */
pci_read_config_dword(bus, slot, function, 0x10, &addr);

/* Restore to original */
pci_write_config_dword(bus, slot, function, 0x10, tmp);

/* Calculate PAGE0 offset from BAR0 (upper half of BAR0) */
addr = (~addr+1)>>1;

/* Set PAGE0 to point to start of APB */
page0 = tmp + addr;
*page0 = (unsigned int *) 0x80000000;
```

58.4 PCI master Interface

The PCI Master interface occupies 128 MB to 2 GB of AHB memory address space and 128 kB of AHB I/O address space. It handles AHB accesses to its AHB slave interface and translates them to PCI configuration (host only), memory or I/O cycles.

The mapping of the AHB slave into AHB address space is configurable through VHDL generics (see the GRLIB User’s Manual for a detailed description of the AHB address mapping). The PCI cycles performed on the PCI bus depends on the AHB access and the values in the configuration/status register.

If the PCI host signal is asserted (active low) during reset, the PCI master function will be enabled after reset. Otherwise the PCI host must enable the device to act as a master through the Bus Master bit in the command register in PCI configuration space.

The PCI master interface is capable of performing the following PCI cycles:

58.4.1 Configuration cycles

PCI configuration cycles are performed by accessing the upper 64 kB of the AHB I/O address space allocated by the AHB slave. If the bus number field in the AHB configuration register is set to 0 then type 0 cycles are generated with the mapping shown in the figure below.

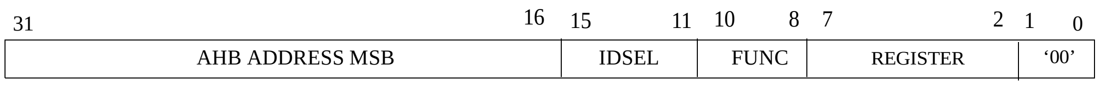


Figure 219. Mapping of AHB I/O addresses to PCI address for PCI Configuration cycle, type 0

[31:16]:	AHB Address MSB - Not used for configuration cycle address mapping.
[15:11]:	IDSEL - This field is decoded to drive PCI AD[IDSEL+10]. AD[31:11] signal lines are supposed to drive IDSEL lines during configuration cycles.
[10:8]:	Function Number (FUNC) - Selects function on multi-function device.
[7:2]:	Register Number (REGISTER) - Used to index a PCI DWORD in configuration space.
[1:0]:	Always driven to ‘00’ to generate Type 0 configuration cycle.

If the bus number field in the AHB configuration register is set to a value between 1 and 15 then type 1 cycles are generated with the mapping shown in the figure below.

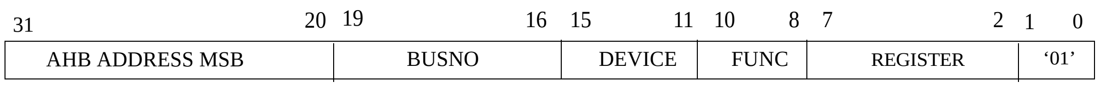


Figure 220. Mapping of AHB I/O addresses to PCI address for PCI Configuration cycle, type 0

[31:20]:	AHB Address MSB - Not used for configuration cycle address mapping.
[19:16]:	Bus number (BUSNO) - Which bus to access.
[15:11]:	Device number (DEVICE) - Which device to select.
[10:8]:	Function Number (FUNC) - Selects function on multi-function device.
[7:2]:	Register Number (REGISTER) - Used to index a PCI DWORD in configuration space.
[1:0]:	Always driven to ‘01’ to generate Type 1 configuration cycle.

When a configuration cycle does not get a response the configuration timeout (CTO) bit is set in the APB configuration/status register. This bit should be checked when scanning the bus for devices in order to detect the slots which are used.

58.4.2 I/O cycles

Single access PCI I/O cycles are supported. Accesses to the lower 64 kB of the GRPCI AHB I/O address space are translated into PCI I/O cycles. The address translation is determined by the value in the I/O map register.

58.4.3 Memory cycles

PCI memory cycles are performed by accessing the AHB memory slave. Mapping and PCI command generation are determined by the values in the AMBA configuration/status register. Burst operation is supported for PCI memory cycles.

The PCI commands generated by the master are directly dependant of the AMBA access and the value of the configuration/status register. The configuration/status register can be programmed to issue Memory Read, Memory Read Line, Memory Read Multiple, Memory Write or Memory Write and Invalidate.

If a burst AHB access is made to PCI master’s AHB slave it is translated into burst PCI memory cycles. When the PCI master interface is busy performing the transaction on the PCI bus, its AHB slave interface will not be able to accept new requests. A ‘retry’ response will be given to all accesses to its AHB slave interface. A requesting AHB master should repeat its request until ‘ok’ or ‘error’ response is given by the PCI master’s AHB slave interface.

Note that ‘retry’ responses on the PCI bus will automatically be retried by the PCI master interface until the transfer is either finished or aborted.

For burst accesses, only linear-incremental mode is supported and is directly translated from the AMBA commands.

58.4.4 PCI byte enable generation

The byte-enables on the PCI bus are translated from the AHB HSIZE control signal and the AHB address according to the table below. Note that only word, half-word and byte values of HSIZE are valid.

*Table 645.*Byte enable generation (in PCI little endian configuration)

HSIZE	**Address[1:0]**	**CBE[3:0]**
00 (8 bit)	00	1110
00 (8 bit)	01	1101
00 (8 bit)	10	1011
00 (8 bit)	11	0111
01 (16 bit)	00	1100
01 (16 bit)	10	0011
10 (32 bit)	00	0000

58.5 PCI host operation

The GRPCI core provides a host input signal that must be asserted (active low) for PCI host operation. If this signal is asserted the bus master interface is automatically enabled (BM bit set in PCI configuration space command register).

An asserted PCI host signal also makes the PCI target respond to configuration cycles when no IDSEL signals are asserted (none of AD[31:11] are asserted). This is done for the master to be able to configure its own target.

For designs intended to be hosts or peripherals only the pci_host signal can be tied low or high internally in the design. For multi-purpose designs it should be connected to a pin. The PCI Industrial Computers Manufacturers Group (PICMG) cPCI specification uses pin C2 on connector P2 for this purpose. The pin should have pull-up resistors since peripheral slots leave it unconnected.

It is possible to enable the GRPCI core to drive the PCI reset signal if in the host slot. Normally the PCI reset is used as an input but if the *hostrst* generic is enabled it will drive the reset signal when located in the host slot. The GRPCI reset output signal should then be connected to an open drain pad with a pull up on the output.

PCI interrupts are supported as inputs for PCI hosts. See section 58.6.

58.6 Interrupt support

When acting as a PCI host the GRPCI core can take the four PCI interrupt lines as inputs and use them to forward an interrupt to the interrupt controller.

If any of the PCI interrupts lines specified through the *irqmask* generic is asserted the PCI core will drive the internal irq line specified through the *irq* generic. E.g. if *irqmask* is 1 only PCI int A will be used but if it is 3 PCI int A and B will be used.

There is no built in support in the PCI core to generate PCI interrupts. These should be generated by the respective IP core and drive an open-drain pad connected to the correct PCI interrupt line. Note that all single function PCI devices should drive PCI interrupt A.

58.7 Byte twisting

To maintain the correct byte order on transfers going from AHB to PCI and vice versa the GRPCI core defaults to byte twisting on all such accesses. This means that all the byte lanes are swapped as shown in the figure below.

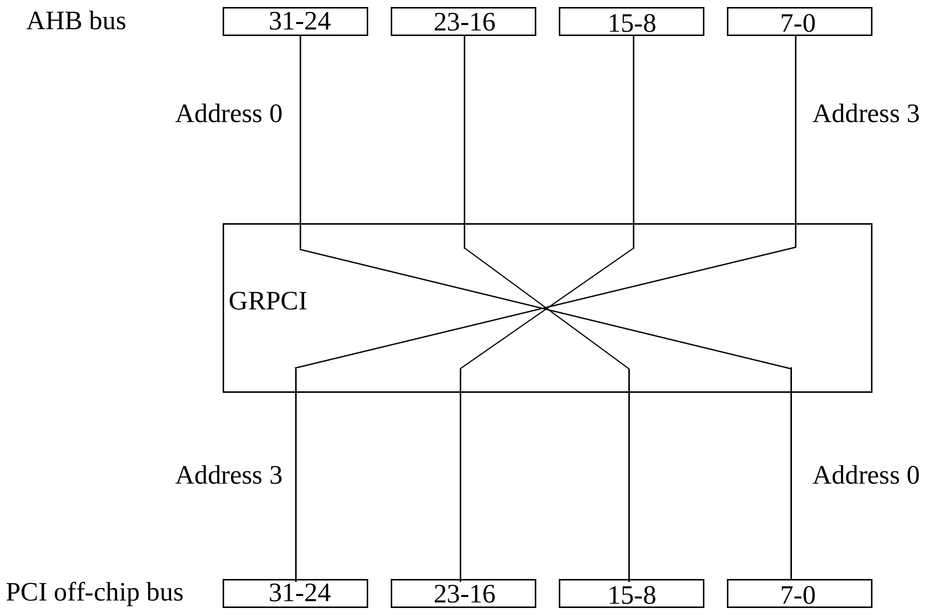


Figure 221. GRPCI byte twisting

The byte twisting can be disabled by writing 0 to bit 0 in the PAGE0 register. This should be done if the AHB bus is little endian or if no twisting is wanted for another reason. It it also possible to configure the PCI bus to be big endian through the *endian* generic (0, meaning little endian, is default).

NOTE: Only accesses that go from AHB to PCI and vice versa are twisted, i.e not accesses to configuration space or the PAGE0 register as they are little endian.

58.7.1 Byte twisting for hosts

Byte twisting should be enabled for big endian PCI hosts. Otherwise DMA transfers from PCI peripherals into the host memory will not have the correct byte ordering.

When the byte twisting is enabled byte sized PIO accesses work as expected but 16 bit and larger PIO accesses need to be twisted before being sent to the PCI core. I.e. if the value 0x12345678 is supposed to be written to a 32-bit register in a PCI peripheral the CPU will need to twist this into 0x78563412 before doing the access. Then the hardware will twist this value back and the correct 32-bit value of 0x12345678 is presented on the PCI bus. Non 8-bit descriptors must also be twisted.

58.7.2 Byte twisting for peripherals

Byte twisting must be enabled if the GRPCI core is used in a peripheral that does DMA to a little endian (or byte twisting big endian) PCI host and the correct byte order is of importance. In this case the data will keep the original byte order but non 8-bit descriptors and PIO accesses must be pre-twisted in software.

If the host is a LEON3 with GRPCI and all peripherals are big endian systems then the PCI bus could be defined big endian and byte twisting disabled for all devices (including the host).

58.8 FIFO operation

Asynchronous FIFOs of configurable size are used for all burst transactions. They are implemented with two port RAM blocks as described in 58.13.2. The PCI master and target interface have two FIFOs each, one used for read transactions and one for write transactions. Each FIFO is divided into two ping pong buffers. How the FIFOs operate during the possible transactions are described below.

58.8.1 PCI target write

When the GRPCI core is the target of a PCI write burst the PCI target begins to fill the write FIFO. After the first ping pong buffer has been filled by the PCI target it continues with filling the second buffer and the AHB master initiates an AHB burst and reads the first ping pong buffer. When the PCI target has finished filling the second buffer it is also emptied by the AHB master. If the PCI write burst is larger than the size of the FIFO the PCI target terminates the PCI transaction with a disconnect with data.

The PCI target does not accept any new transactions until the AHB master has finished emptying the complete FIFO. Any incoming access will receive a retry response.

58.8.2 PCI target read

How PCI target reads are treated depend on the PCI command used. If prefetching is disabled (*read-pref* generic = 0) then for Memory Read (0x6) commands data is fetched one word at a time. After each word the target terminates using disconnect with data. If prefetching is enabled one cache line (as defined by the cache line register) is prefetched. For bursts the Memory Read Multiple (0xC) command should be used. In this case the PCI target requests the AHB master to start filling the FIFO. The PCI targets gives retry responses until the AHB master has filled the first buffer. When the first buffer is full the PCI target responds with the data while the AHB master fills the second buffer. When the PCI target has read out the complete FIFO it terminates the read transaction with disconnect with data.

If the PCI target is forced to insert more than 8 wait states while waiting for the AHB master to to fill the second buffer it will generate a disconnect without data response.

During the period before the target responds it will give retry responses to masters trying to read an address different from the address of the already initiated read transaction.

58.8.3 PCI master write

A PCI write transaction is initiated by an access to the AHB slave of GRPCI. The AHB slave interface fills the FIFO with data. When the first buffer is full the PCI master begins the PCI write transaction while the AHB slave continues filling the second buffer. Accesses to the AHB slave that occurs when the PCI master is emptying the FIFO receives retry responses.

58.8.4 PCI master read

When the AHB slave receives a read access it requests the PCI master to issue a read transaction on the PCI bus and to fill the FIFO with the result. After the first buffer has been filled the AHB slave reads the buffer and begins to empty it to the AHB bus. When the second buffer is full it is also emptied by the AHB slave. While the AHB slave is waiting for the PCI master to fill the FIFO it gives retry responses.

58.9 Registers

The core is programmed through registers mapped into APB address space.

*Table 646.*AMBA registers

Address offset	Register	Note
0x00	Configuration/Status register	-
0x04	BAR0 register	Read-only access from AMBA, write/read access from PCI (see section 58.3.4).
0x08	PAGE0 register	Read-only access from AMBA, write/read access from PCI (see section 58.3.5).
0x0C	BAR1 register	Read-only access from AMBA, write/read access from PCI (see section 58.3.4).
0x10	PAGE1 register	-
0x14	IO Map register	-
0x18	Status & Command register (PCI Configuration Space Header)	Read-only access from AMBA, write/read access from PCI (see section 58.3.4).

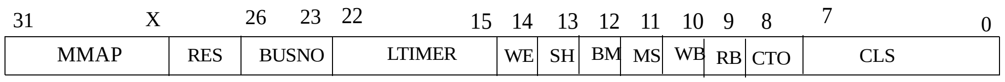


Figure 222. Configuration/Status register

[31:X]: Memory Space Map register - Defines mapping between PCI Master's AHB memory address space and PCI address space when performing PCI memory cycles. Value of this field is used as the MSB of the PCI address. LSB bits are taken from the AHB address (X = 32 - number of bits not masked with the *hmask* generic).
[X:27]: Reserved.
[26:23]: Bus number (BUSNO) - Which bus to access when generating configuration cycles.
[22:15]: Latency Timer (LTIMER) - Value of Latency Timer Register in Configuration Space Header. (Read-only)
[14]: Write Error (WE) - Target Write Error. Write access to units target interface resulted in error. (Read-only)
[13]: System Host (SH) - Set if the unit is system host. (Read-only)
[12]: Bus Master (BM) - Value of BM field in Command register in Configuration Space Header. (Read-only)
[11]: Memory Space (MS) - Value of MS field in Command register in Configuration Space Header. (Read-only)
[10]: Write Burst Command (WB) - Defines PCI command used for PCI write bursts.
'0' - 'Memory Write'
'1' - 'Memory Write and Invalidate'
[9]: Read Burst Command (RB) - Defines PCI command used for PCI read bursts.
'0' - Memory Read Multiple'
'1' - Memory Read Line'
[8]: Configuration Timeout (CTO) - Received timeout when performing Configuration cycle. (Read-only)
[7:0]: Cache Line Size (CLS) - Value of Cache Line Size register in Configuration Space Header. (Read-only)

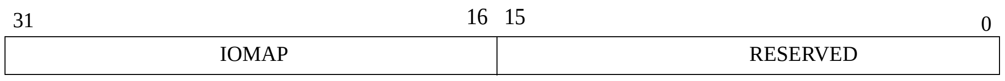


Figure 223. I/O Map register

[31:16]: I/O Map (IOMAP) - Most significant bits of PCI address when performing PCI I/O cycle. Concatenated with low bits of AHB address to from PCI address.
[15:0]: Reserved.

58.10 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x014. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

58.11 Scan support

When the SCANEN generic is 1, scan support is enabled. All asynchronous reset are then connected to AHBMI.testrst when AHBMI.testen = ‘1’. Note that the PCI clock is not multiplexed, and should be driven with the same clock as the AHB clk when AHBMI.testen = ‘1’.

58.12 Configuration options

Table 647 shows the configuration options of the core (VHDL generics).

*Table 647.*Configuration options

Generic	Function	Allowed range	Default
memtech	The memory technology used for the FIFO instantiation	-	0
mstndx	The AMBA master index for the target backend AHB master interface.	0 - NAHBMST-1	0
dmamst	The AMBA master index for the DMA controller, if present. This value is used by the PCI core to detect when the DMA controller accesses the AHB slave interface.	0 - NAHBMS	NAHBMST (= disabled)
readpref	Prefetch data for the ‘memory read’ command. If set, the target prefetches a cache line, otherwise the target will give a single word response.	0 -1	0
abits	Least significant implemented bit of BAR0 and PAGE0 registers. Defines PCI address space size.	16 - 28	21
dmaabits	Least significant implemented bit of BAR1 and PAGE1 registers. Defines PCI address space size.	16 - 28	26
fifodepth	Size of each FIFO is 2^fifodepth 32-bit words.	>= 3	5
device_id	PCI device ID number	0 -16#FFE#	0
vendor_id	PCI vendor ID number	0 - 16#FFF#	0
master	Disables/enables PCI master interface.	0 - 1	0
slvndx	The AHB index of the master backend AHB slave interface.	0 - NAHBSLV-1	0
apbndx	The AMBA APB index for the configuration/status APB interface	0 - NAPBMAX-1	0
paddr	APB interface base address	0 - 16#FFF#	0
pmask	APB interface address mask	0 - 16#FFF#	16#FFF#
haddr	AHB slave base address	0 - 16#FFF#	16#F00#
hmask	AHB address mask. 128 MB - 2 GB.	16#800# - 16#F80#	16#F00#
ioaddr	AHB I/O area base address	0 - 16#FFF#	0
irq	IRQ line driven by the PCI core	0 - NAHBIRQ-1	0
irqmask	Specifies which PCI interrupt lines that can cause an interrupt	0 - F	0
nsync	The number of clock registers used by each signal that crosses the clock regions.	1 - 2	2
oepol	Polarity of pad output enable signals. 0=active low, 1=active high	0 - 1	0
endian	Endianess of the PCI bus. 0 is little and 1 big.	0 - 1	0
class_code	Class code. Defaults to base class 0x0B (processor), sub class 0x40 (co-processor).	See PCI spec.	16#0B4000#
rev	Revision	0 - 16#FF#	0
scanen	Enable scan support	0 - 1	0
hostrst	Enable driving of pci_rst when host	0 - 1	0

58.13 Implementation

58.13.1 Technology mapping

GRPCI has one technology mapping generic, *memtech*, which controls how the memory cells used will be implemented. See the GRLIB Users's Manual for available settings.

58.13.2 RAM usage

The FIFOs in GRPCI are implemented with the *syncram_2p* (with separate clocks for each port) component from the technology mapping library (TECHMAP). The number of FIFOs used depends on whether the core is configured to be master/target or target only (as selected with the *master* generic). The master and target interface both use two 32 bits wide FIFOs. The depth of all FIFOs is the same and is controlled by the *fifodepth* generic.

*Table 648.*RAM usage

Configuration	**Number of *fifodepth* x 32 bit RAM blocks**
Master/target	4
Target only	2

58.13.3 Area

The GRPCI is portable and can be implemented on most FPGA and ASIC technologies. The table below shows the approximate area usage.

*Table 649.*Approximate area requirements

FIFO size	**VirtexII LUTs**	**StratixII LUTs**	**ASIC gates**
8	1500	1200	8000
32	1900	1300	10000

58.13.4 Timing

In order for the PCI core to function properly in a PCI system it is necessary to meet the PCI timing constraints. The PCI clock to out should not exceed 11 ns and the setup time must be below 7 ns. If you experience excessive clock to out make sure that the synthesizer has not removed the output registers. This can happen with too aggressive pipelining/retiming.

58.13.5 Pull-ups

For PCI hosts we recommend that the following signals are provided with pull-ups (if these are not available on the motherboard/rack).

*Table 650.*PCI host pull-ups

pci_devsel
pci_frame
pci_irdy
pci_lock
pci_perr
pci_serr
pci_stop
pci_trdy
pci_gnt
pci_par
pci_rst
pci_arb_req(x)

58.14 Signal description

Table 651 shows the interface signals of the core (VHDL ports).

*Table 651.*Signal descriptions

Signal name	**Field**	**Type**	**Function**	**Active**
RST	N/A	Input	Reset	Low
CLK	N/A	Input	AMBA system clock	-
PCICLK	N/A	Input	PCI clock	-
PCII	*1	Input	PCI input signals	-
PCIO	*1	Output	PCI output signals	-
APBI	*2	Input	APB slave input signals	-
APBO	*2	Output	APB slave output signals	-
AHBMI	*2	Input	AHB master input signals	-
AHBMO	*2	Output	AHB master output signals	-
AHBSI	*2	Input	AHB slave input signals	-
AHBSO	*2	Output	AHB slave output signals	-

*1) see PCI specification
*2) see GRLIB IP Library User’s Manual

The PCIO record contains an additional output enable signal VADEN. It is has the same value as aden at each index but they are all driven from separate registers. A directive is placed on this vector so that the registers will not be removed during synthesis. This output enable vector can be used instead of aden if output delay is an issue in the design.

For a system host, the (active low) PCII.host signal should to be connected to the PCI SYSEN signal. For a device that is not a system host, this signal should have a pull-up connection.

58.15 Library dependencies

Table 652 shows libraries used when instantiating the core (VHDL libraries).

*Table 652.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Signals, component	PCI signals and component declaration

58.16 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.stdlib.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.pci.all;

.
.
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);


signal pcii : pci_in_type;
signal pcio : pci_out_type;

begin

pci0 : pci_mtf generic map (memtech => memtech,
hmstndx => 1,
fifodepth => log2(CFG_PCIDEPTH), device_id => CFG_PCIDID, vendor_id => CFG_PCIVID,
hslvndx => 4, pindex => 4, paddr => 4, haddr => 16#E00#,
ioaddr => 16#400#, nsync => 2)
port map (rstn, clkm, pciclk, pcii, pcio, apbi, apbo(4), ahbmi,
ahbmo(1), ahbsi, ahbso(4));

pcipads0 : pcipads generic map (padtech => padtech)-- PCI pads
    port map ( pci_rst, pci_gnt, pci_idsel, pci_lock, pci_ad, pci_cbe,
               pci_frame, pci_irdy, pci_trdy, pci_devsel, pci_stop, pci_perr,
               pci_par, pci_req, pci_serr, pci_host, pci_66, pcii, pcio );
```

58.17 Software support

Support for LEON3 systems acting as PCI hosts using the GRPCI is available in Linux 2.6, RTEMS and VxWorks. In the GRLIB IP library there is a simple Bare C (BCC) example of how configure and use PCI in GRLIB/software/leon3/pcitest.c. See also 58.18, Appendix A for BCC source code examples.

The debug monitor GRMON supports PCI bus scanning and configuration for GRPCI hosts.

58.18 Appendix A - Software examples

Examples of PCI configurations functions.

pci_read_config_dword - generate a configuration read (type 0) cycle

pci_write_config_dword - generate a configuration write (type 0) cycle

pci_mem_enable - enable the memory space of a device

pci_master_enable - enable bus master for a device

```
/* Upper half of IO area */
#define PCI_CONF 0xfff10000

typedef struct {
volatile unsigned int cfg_stat;
volatile unsigned int bar0;
volatile unsigned int page0;
volatile unsigned int bar1;
volatile unsigned int page1;
volatile unsigned int iomap;
volatile unsigned int stat_cmd;
} LEON3_GRPCI_Regs_Map;

int
pci_read_config_dword(unsigned char bus, unsigned char slot, unsigned char function, unsigned
char offset, unsigned int *val) {

    volatile unsigned int *pci_conf;

    if (offset & 3) return -1;

    if (slot >= 21) {
        *val = 0xffffffff;
        return 0;
    }

    pci_conf = PCI_CONF + ((slot<<11) | (function<<8) | offset);

    *val =  *pci_conf;

    if (pcic->cfg_stat & 0x100) {
        *val = 0xffffffff;
    }

    return 0;
}

int
pci_write_config_dword(unsigned char bus, unsigned char slot, unsigned char function,
unsigned char offset, unsigned int val) {

    volatile unsigned int *pci_conf;

    if (offset & 3) return -1;

    pci_conf = PCI_CONF + ((slot<<11) | (function<<8) | (offset & ~3));

    *pci_conf = val;

    return 0;
}

void pci_mem_enable(unsigned char bus, unsigned char slot, unsigned char function) {
    unsigned int data;

    pci_read_config_dword(0, slot, function, 0x04, &data);
    pci_write_config_dword(0, slot, function, 0x04, data | 0x2);
}

void pci_master_enable(unsigned char bus, unsigned char slot, unsigned char function) {
    unsigned int data;

    pci_read_config_dword(0, slot, function, 0x04, &data);
    pci_write_config_dword(0, slot, function, 0x04, data | 0x4);

}
```

58.19 Appendix B - Troubleshooting

58.19.1 GRPCI does not operate correctly.

Make sure that the PCI timing constraints are met (see 58.13.4) and that all necessary pull-ups are available. The generic nsync should be set to 2 for reliable operation.

58.19.2 It is impossible to meet the PCI clock to out constraint due to too many levels of logic.

The PCI output registers have been removed. This can happen when the tools use pipelining and retiming to aggressively.

58.19.3 I write 0x12345678 but get 0x78563412, what is the matter?

Please read about byte twisting in section 58.7.

58.19.4 The PCI target responds by asserting stop when I try to read data.

The PCI target gives a ‘retry’ response (stop but not trdy asserted on initial data phase). This is normal since the PCI target needs to request the data from the AHB master before it can deliver it on the PCI bus.

If a Memory Read Multiple command is issued the PCI target will give retry responses until half the FIFO has been prefetched. Thus the initial latency is dependant on the FIFO size and the PCI command. See section 58.8.

58.19.5 Configuration space accesses do not work.

In order to initiate any access the device must have the Bus Master bit set in the PCI configuration space command register. This bit is automatically set if the GRPCI host input signal is asserted (low). See section 58.5 (PCI host operation).

59 PCIDMA - DMA Controller for the GRPCI interface

59.1 Introduction

The DMA controller is an add-on interface to the GRPCI interface. This controller perform bursts to or from PCI bus using the master interface of GR PCI Master/target unit.

Figure 1 below illustrates how the DMA controller is attached between the AHB bus and the PCI master interface.

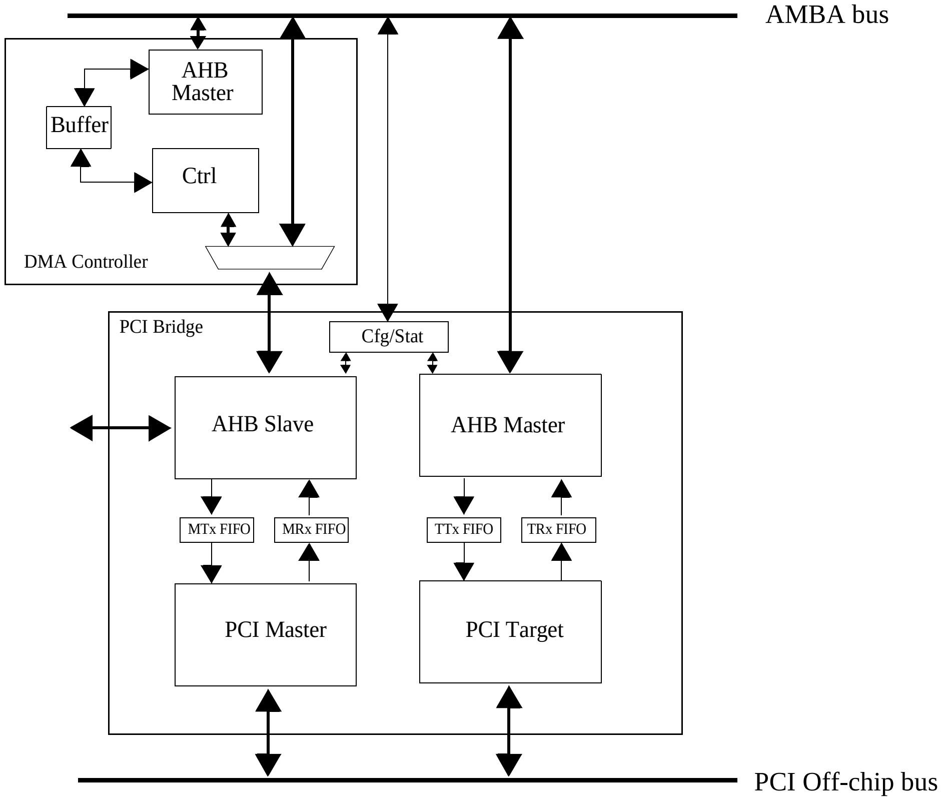


Figure 224. DMA Controller unit

59.2 Operation

The DMA controller is set up by defining the location of memory areas between which the DMA will take place in both PCI and AHB address space as well as direction, length and type of the transfer. Only 32-bit word transfer are supported.

The DMA transfer is automatically aborted when any kind of error is detected during a transfer. The DMA controller does not detect deadlocks in its communication channels. If the system concludes that a deadlock has occurred, it can manually abort the DMA transfer. It is allowed to perform burst over a 1 Kbyte boundary of the AHB bus. When this happens, an AHB idle cycle will be automatically inserted to break up the burst over the boundary.

When the DMA is not active the AHB slave interface of PCI Master/Target unit will be directly connected to AMBA AHB bus.

59.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 653.*DMA Controller registers

Address offset	Register
0x00	Command/status register
0x04	AMBA Target Address
0x08	PCI Target Address
0x0C	Burst length

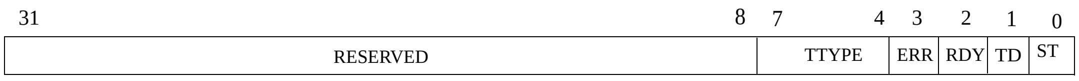


Figure 225. Status/Command register

[31:8]: Reserved.
[7:4]: Transfer Type (TTYPE) - Perform either PCI Memory or I/O cycles. “1000” - memory cycles, “0100” - I/O cycles. This value drives directly HMBSEL signals on PCI Master/Targets units AHB Slave interface.
[3]: Error (ERR) - Last transfer was abnormally terminated. If set by the DMA Controller this bit will remain zero until cleared by writing ‘1’ to it.
[2]: Ready (RDY) - Current transfer is completed. When set by the DMA Controller this bit will remain zero until cleared by writing ‘1’ to it.
[1]: Transfer Direction (TD) - ‘1’ - write to PCI, ‘0’ - read form PCI.
[0]: Start (ST) - Start DMA transfer. Writing ‘1’ will start the DMA transfer. All other registers have to be set up before setting this bit. Set by the PCI Master interface when its transaction is terminated with Target-Abort. Writing ‘1’

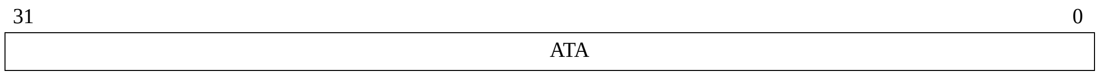


Figure 226. AMBA Target Address

[31:0]: AMAB Target Address (ATA) - AHB start address for the data on AMBA bus. In case of error, it indicated failing address.

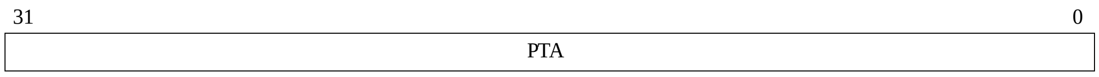


Figure 227. PCI Target Address

[31:0]: PCI Target Address (PTA) - PCI start address on PCI bus. This is a complete 32-bit PCI address and is not further mapped by the PCI Master/Target unit. In case of error, it indicated failing address.

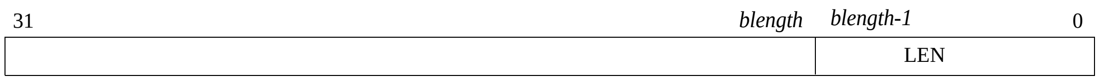


Figure 228. Length register

[*blentgh-1*:0]: DMA Transfer Length (LEN) - Number of 32-bit words to be transferred.

59.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x016. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

59.5 Configuration options

Table 654 shows the configuration options of the core (VHDL generics).

*Table 654.*Configuration options

Generic	Function	Allowed range	Default
mstndx	DMA Controllers AHB Master interface index	0 - NAHBMST-1	0
apbndx	The AMBA APB index for the configuration/status APB interface	0 - NAPBMAX-1	0
apbaddr	APB interface base address	0 - 16#FFF#	0
apbmask	APB interface address mask	0 - 16#FFF#	16#FFF#
blength	Number of bits in the Burst length register	-	16

59.6 Signal description

Table 655 shows the interface signals of the core (VHDL ports).

*Table 655.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	AMBA system clock	-
PCICLK	N/A	Input	PCI clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-
AHBSI0	*	Input	AHB slave input signals, main AHB bus	-
AHBSO0	*	Output	AHB slave output signals, main AHB bus	-
AHBSI1	*	Input	AHB slave input signals, connected to PCI Target/Master unit	-
AHBSO1	*	Output	AHB slave output signals, connected to PCI Target/Master unit	-

* see GRLIB IP Library User's Manual

59.7 Library dependencies

Table 656 shows libraries used when instantiating the core (VHDL libraries).

*Table 656.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Component	Component declaration

59.8 Instantiation

This example shows how the core can be instantiated

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.pci.all;
use gaisler.pads.all;

signal pcii : pci_in_type;
signal pcio : pci_out_type;

dma : pcidma generic map (memtech => memtech, dmstndx => 1,
  dapbndx => 5, dapbaddr => 5, blength => blength, mstndx => 0,
  fifodepth => log2(fifodepth), device_id => CFG_PCIDID, vendor_id => CFG_PCIVID,
  slvndx => 4, apbndx => 4, apbaddr => 4, haddr => 16#E00#, ioaddr => 16#800#,
  nsync => 1)
port map (rstn, clkm, pciclk, pcii, pcio, apbo(5), ahbmo(1),
apbi, apbo(4), ahbmi, ahbmo(0), ahbsi, ahbso(4));


pcipads0 : pcipads generic map (padtech => padtech)
port map ( pci_rst, pci_gnt, pci_idsel, pci_lock, pci_ad, pci_cbe,
      pci_frame, pci_irdy, pci_trdy, pci_devsel, pci_stop, pci_perr,
      pci_par, pci_req, pci_serr, pci_host, pci_66, pcii, pcio );
```

60 PCITB_MASTER_SCRIPT - Scriptable PCI testbench master

60.1 Overview

The PCITB_MASTER_SCRIPT IP core provides a simulation model for a PCI master with system host capabilities. The user specifies the requested PCI commands and check their result in a script file which has an easy to use syntax.

Features:

- Support for PCI memory, i/o and configuration cycles
- Easy to use scripting syntax
- Write/read PCI data to/from files
- Flexible burst length (1-65535 words)
- User can specify byte enables for each data phase

60.2 Operation

As soon as the PCI reset is released the PCI master begins to parse the specified script file. It will display the operations performed in the simulation console.

The following commands are supported. Comments are supported in script files and such lines must begin with ‘#’. Note that address and data must be specified in hexadecimal width 8 digits and that length must be hexadecimal with 4 digits. Each command should be ended with a semicolon.

60.2.1 wait <cycles>;

The master waits the specified number of clock cycles.

Example:

```
wait 5;
```

60.2.2 comp <file1> <file2>;

Compare file1 to file1 and display the result.

Example:

```
comp test1.log test2.log;
```

60.2.3 stop;

End PCI master operation.

60.2.4 halt;

Halt simulation.

60.2.5 print <string>;

Prints everything between ‘print’ and ‘;’ to the simulation console.

Example:

```
print PCI testbench print example;
```

60.2.6 estop <0|1>;

Turn on (1) or off (0) stop on error. If on, the PCI test master will stop the testbench with a failure if any error is detected.

Example:

```
estop 1;
```

60.2.7 rcfg <addr> <data | *>;

Perform a configuration read cycle from address ‘addr’ and compare to ‘data’. If a ‘*’ is specified instead of a data word then no checking is done.

Examples:

```
rcfg 10000000 12345678;
rcfg 10000000 *;
```

60.2.8 wcfg <addr> <data.cbe>;

Perform a configuration write cycle with the specified data to address ‘addr’. Specifying the byte enables is optional.

Examples:

```
wcfg 10000000 12345678;
wcfg 10000000 12345678.C;
```

60.2.9 rmem <cmd> <addr> <length> <data | filename | *>;

Perform a PCI read transaction using command ‘cmd’ from address ‘addr’ and compare to the specified data. If a ‘*’ is specified then no checking is done. If a filename is specified then the read data is stored in that file. Note that data must be specfied between braces. If byte enables are specified for a word then only the enabled bytes will be compared.

Examples:

```
# Burst read 4 words and compare against specified data
rmem C 10000000 0004 {
12345678
87654321
AA55AA55
11223344
};
# Read a single word and compare using CBE=3
rmem 6 10000000 0001 {
12345678.3
};
# Read single word and ignore result
rmem 6 10000000 0001 *;

# Burst read 64 words and store in file read.log
rmem C 10000000 0040 read.log;
```

60.2.10 wmem <cmd> <addr> <length> <data | filename>;

Perform a PCI write transaction using command ‘cmd’ to address ‘addr’. Note that data must be specfied between braces. If a filename is specified then the data is read from that file.

Examples:

```
# Burst write 4 words
wmem 7 10000000 0004 {
12345678
87654321
AA55AA55
11223344
};

# Write a single word using CBE=3
wmem 7 10000000 0001 {
12345678.3
};

# Write burst 64 words from file write.txt (write.txt should have
# one word per line)
wmem 7 10000000 0040 write.txt;
```

60.2.11 Example script file

```
#########################
# PCI TEST SCRIPT
#########################

# Wait 10 clock cycles
wait 10;

# Read vendor/device id and compare to 0xBACCFEED (of device with idsel <= ad31)
rcfg 80000000 BACCFEED;

# Enable memory space
wcfg 80000004 00000002;

# Set BAR0 to 0x10000000
wcfg 80000010 10000000;

# Set BAR1 to 0x20000000
wcfg 80000014 20000000;

# Write single word to PAGE0 (BAR0 + 0x8000).
# BAR0 -> APB, byte twisting enabled
wmem 7 10008000 0001 {
80000001
};

# Read single word from BAR0 and compare to 0x80000000
rmem 6 10008000 0001 {
80000001
};

# Set GRPCI PAGE1 to 0x40000000 (RAM) through BAR0 (BAR0 + 0x410)
wmem 7 10000410 0001 {
00000040.0
};

# Read back PAGE1 and  compare
rmem 6 10000410 0001 {
00000040
};

# Burst write 8 words to BAR1 (memory)
wmem 7 20000000 0008 {
11223344.3
22334411.0
33441122.0
44112233.0
12345678.0
87654321.0
a5a5a5a5.0
00001111.c
};

# Burst read 8 words and check against specified data
rmem C 20000000 0008 {
11223344.3
22334411
33441122
44112233
12345678
87654321
a5a5a5a5
00001111.c
};

# Burst write 64 words from file wf1.txt
wmem 7 20000000 0040 wf1.txt;

# Burst read 64 words and store result in rf1.txt
rmem C 20000000 0040 rf1.txt;

# Compare wf1.txt with rf1.txt
comp wf1.txt rf1.txt;

# End of Simulation
stop;
```

60.3 Configuration options

Table 657 shows the configuration options of the core (VHDL generics).

*Table 657.*Configuration options

Generic	Function	Allowed range	Default
slot	PCI slot used by master. Determines which req/gnt pair to use	0-20	0
tval	Output delay for signals that are driven by this unit	0-7 ns	7 ns
dbglevel	Debug level. Higher value means more debug information	0-2	1
maxburst	Maximum burst length supported	1-65535	1024
filename	PCI command script file	-	pci.cmd

60.4 Signal descriptions

Table 658 shows the interface signals of the core (VHDL ports).

*Table 658.*Signals descriptions

Signal name	Field	Type	Function	Active
PCIIN	*	Input	PCI input signals	-
PCIOUT	*	Output	PCI output signals	-

*1) PCI_TYPE (declared in pcitb.vhd). See PCI specification for more info on PCI signals

60.5 Library dependencies

Table 659 shows the libraries used when instantiating the core (VHDL libraries).

*Table 659.*Library dependencies

Library	**Package**	**Imported unit(s)**	**Description**
GAISLER	PCITB	Signals, component	PCI TB signals and component declaration

61 PCITARGET - Simple 32-bit PCI target with AHB interface

61.1 Overview

This core implements PCI interface with a simple target-only interface. The interface is developed primarily to support DSU communication over the PCI bus. Focus has been put on small area and robust operation, rather than performance. The interface has no FIFOs, limiting the transfer rate to about 5 Mbyte/s. This is however fully sufficient to allow fast download and debugging using the DSU.

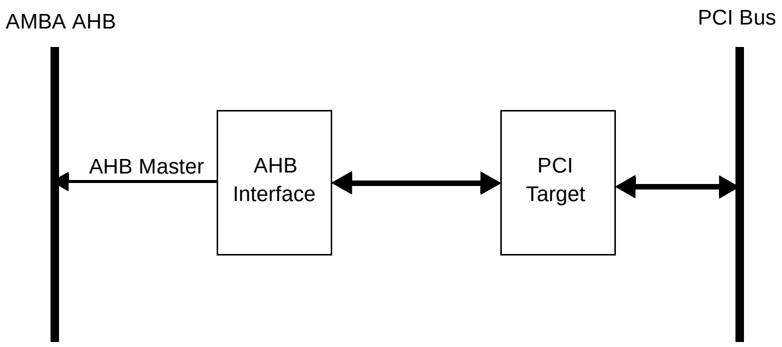


Figure 229. Target-only PCI interface

61.2 Registers

The core implements one PCI memory BAR.

31 *abits*-1	*abits*-2 0
AHB address [31:*abits*-1]	UNUSED

Figure 230. AHB address register (BAR0, 0x100000)

The interface consist of one PCI memory BAR occupying (2^*abits*) bytes (default: 2 Mbyte) of the PCI address space, and an AHB address register. Any access to the lower half of the address space (def.: 0 - 0xFFFFF) will be forwarded to the internal AHB bus. The AHB address will be formed by concatenating the AHB address field of AHB address register with the LSB bits of the PCI address. An access to the upper half of the address space (default: 1 Mbyte on 0x100000 - 0x1FFFFF) of the BAR will read or write the AHB address register.

61.3 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x012. For description of vendor and device identifies see GRLIB IP Library User’s Manual.

61.4 Configuration options

Table 660 shows the configuration options of the core (VHDL generics).

*Table 660.*Configuration options

Generic	Function	Allowed range	Default
hindex	Selects which AHB select signal (HSEL) will be used to access the PCI target core	0 to NAHBMAX-1	0
abits	Number of bits implemented for PCI memory BAR	0 to 31	21
device_id	PCI device id	0 to 65535	0
vendor_id	PCI vendor id	0 to 65535	0
nsync	One or two synchronization registers between clock regions	1 - 2	1
oepol	Polarity of output enable signals. 0=active low, 1=active high	0 - 1	0

61.5 Signal descriptions

Table 661 shows the interface signals of the core (VHDL ports).

*Table 661.*Signals descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	AHB system clock	-
PCICLK	N/A	Input	PCI clock	-
PCII	*1	Input	PCI input signals	-
PCIO	*1	Output	PCI output signals	-
APBI	*2	Input	APB slave input signals	-
APBO	*2	Output	APB slave output signals	-

*1) see PCI specification
*2) see GRLIB IP Library User’s Manual

The PCIO record contains an additional output enable signal vaden. It is has the same value as aden at each index but they are all driven from separate registers. A directive is placed on this vector so that the registers will not be removed during synthesis. This output enable vector can be used instead of aden if output delay is an issue in the design.

61.6 Library dependencies

Table 662 shows the libraries used when instantiating the core (VHDL libraries).

*Table 662.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	PCI	Signals, component	PCI signals and component declaration

62 PHY - Ethernet PHY simulation model

62.1 Overview

The PHY is a simulation model of an IEEE 802.3 compliant Ethernet PHY. It provides a complete MII and GMII interface with the basic, extended status and extended capability registers accessible through the management interface (MDIO). Not all of the functionality is implemented and many of the register bits are therefore only writable and readable but do not have any effect. Currently only the loopback is supported.

62.2 Operation

The PHY simulation model was designed to make it possible to perform simple simulations on the GRETH and GRETH_GBIT cores in GRLIB. It provides the complete set of basic, extended capability and extended status registers through the MII management interface (MDIO) and a loopback mode for data transfers. Figure 1 shows a block diagram of a typical connection.

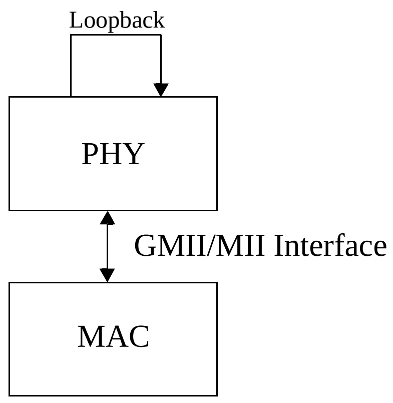


Figure 231. Block diagram of the PHY simulation model connected to a MAC.

The PHY model provides the complete GMII and MII interface as defined by the IEEE 802.3 standard. The model can be used in any of the following modes: 10 Mbit half- or full duplex, 100 Mbit half- or full-duplex and 1000 Mbit half- or full-duplex. This support refers only to the configuration settings available through the MDIO registers. Since the datapath implementation is loopback no collisions will ever be seen on the network and operation will essentially be full-duplex all the time. In loopback mode, rx_clk and tx_clk are identical in both frequency and phase and are driven by the PHY when not in gigabit mode. In gigabit mode the gtx_clk input is used as the transmitter clock and it also drives rx_clk.

When not configured to loopback mode the PHY just sits idle and ignores transmitted packet and does not insert any activity on the receive interface. Clocks are still generated but in this case rx_clk and tx_clk does have the same frequency but not the same phase when not in gigabit mode.

A simple auto-negotiation function is provided and the supported and advertised modes are set through vhdl generics. The generic values will be directly reflected in the reset values and read-only values of all corresponding MII management registers.

62.3 Configuration options

Table 663 shows the configuration options of the model (VHDL generics).

*Table 663.*Configuration options

Generic	Function	Allowed range	Default
address	Address of the PHY on the MII management interface	0 - 31	0
extended_regs	Include extended register capability	0 - 1	1
aneg	Enable auto-negotiation functionality	0 - 1	1
base100_t4	Enable support for 100Base-T4	0 - 1	0
base100_x_fd	Enable support for 100Base-X full-duplex	0 - 1	1
base100_x_hd	Enable support for 100Base-X half-duplex	0 - 1	1
fd_10	Enable support for 10Base-T full-duplex	0 - 1	1
hd_10	Enable support for 10Base-T half-duplex	0 - 1	1
base100_t2_fd	Enable support for 100Base-T2 full-duplex	0 - 1	1
base100_t2_hd	Enable support for 100Base-T2 half-duplex	0 - 1	1
base1000_x_fd	Enable support for 1000Base-X full-duplex	0 - 1	0
base1000_x_hd	Enable support for 1000Base-X half-duplex	0 - 1	0
base1000_t_fd	Enable support for 1000Base-T full-duplex	0 - 1	1
base1000_t_hd	Enable support for 1000Base-T half-duplex	0 - 1	1

62.4 Signal descriptions

Table 664 shows the interface signals of the model (VHDL ports).

*Table 664.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	-	Input	Reset	Low
MDIO	-	Input/ Output	Data signal for the management interface (Currently not used)	-
TX_CLK	-	Output	Transmitter clock	-
RX_CLK	-	Output	Receiver clock	-
RXD	-	Output	Receiver data	-
RX_DV	-	Output	Receiver data valid	High
RX_ER	-	Output	Receiver error	High
RX_COL	-	Output	Collision	High
RX_CRS	-	Output	Carrier sense	High
TXD	-	Input	Transmitter data	-
TX_EN	-	Input	Transmitter enable	High
TX_ER	-	Input	Transmitter error	High
MDC	-	Input	Management interface clock (Currently not used)	-
GTX_CLK	-	Input	Gigabit transmitter clock	-

see the IEEE 802.3 standard for a description of how the signals are used.

62.5 Library dependencies

Table 665 shows the libraries used when instantiating the model (VHDL libraries).

*Table 665.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	SIM	Component	Component declaration

62.6 Instantiation

This example shows how the model can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.sim.all;

entity phy_ex is
  port (
rst : std_ulogic;
clk : std_ulogic;
    );
end;

architecture rtl of phy_ex is

  -- Signals

 signal etx_clk   : std_logic;
 signal gtx_clk   : std_logic;
 signal erx_clk   : std_logic;
 signal erxd      : std_logic_vector(7 downto 0);
 signal erx_dv    : std_logic;
 signal erx_er    : std_logic;
 signal erx_col   : std_logic;
 signal erx_crs   : std_logic;
 signal etxd      : std_logic_vector(7 downto 0);
 signal etx_en    : std_logic;
 signal etx_er    : std_logic;
 signal emdc      : std_logic;

begin

  -- Other components are instantiated here
  ...


  -- PHY model
  phy0 : phy
  generic map (address => 1)
  port map(resetn => rst, mdio => open, tx_clk => etx_clk, rx_clk => erx_clk, rxd => erxd,
  rx_dv => erx_dv, rx_er => erx_er,
  rx_col => erx_col, rx_crs => erx_crs, txd => etxd, tx_en => etx_en,
  tx_er => etx_er, mdc => emdc, gtx_clk => gtx_clk);
end;
```

63 REGFILE_3P 3-port RAM generator (2 read, 1 write)

63.1 Overview

The 3-port register file has two read ports and one write port. Each port has a separate address and data bus. All inputs are latched on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Note: on most technologies, the register file is implemented with two 2-port RAMs with combined write ports. Address width, data width and target technology is parametrizable through generics.

Write-through is supported if the function *syncram_2p_write_through(tech)* returns 1 for the target technology.

63.2 Configuration options

Table 666 shows the configuration options of the core (VHDL generics).

*Table 666.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table 667	-
dbits	Data width	see table 667l	-
wrfst	Write-first (write-through). Only applicable to inferred technology	0 - 1	0
numregs	Not used		

Table 667 shows the supported technologies for the core.

*Table 667.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
axcel	Actel AX/RTAX	RAM64K36	2 - 12	unlimited
altera	All Altera devices	altsyncram	unlimited	unlimited
ihp25	IHP 0.25	flip-flops	unlimited	unlimited
inferred	Behavioural description	synthesis tool dependent		
rhumc	Rad-hard UMC 0.18	flip-flops	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2, Spartan3, Virtex4	RAMB16_Sn	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited

63.3 Signal descriptions

Table 668 shows the interface signals of the core (VHDL ports).

*Table 668.*Signal descriptions

Signal name	Field	Type	Function	Active
WCLK	N/A	Input	Write port clock	
WADDR	N/A	Input	Write address	
WDATA	N/A	Input	Write data	
WE	N/A	Input	Write enable	High
RCLK	N/A	Input	Read ports clock	-
RADDR1	N/A	Input	Read port1 address	-
RE1	N/A	Input	Read port1 enable	High
RDATA1	N/A	Output	Read port1 data	-
RADDR2	N/A	Input	Read port2 address	-
RE2	N/A	Input	Read port2 enable	High
RDATA2	N/A	Output	Read port2 data	-

63.4 Library dependencies

Table 669 shows libraries used when instantiating the core (VHDL libraries).

*Table 669.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

63.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component regfile_3p
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8;
           wrfst : integer := 0; numregs : integer := 64);
  port (
    wclk   : in  std_ulogic;
    waddr  : in  std_logic_vector((abits -1) downto 0);
    wdata  : in  std_logic_vector((dbits -1) downto 0);
    we     : in  std_ulogic;
    rclk   : in  std_ulogic;
    raddr1 : in  std_logic_vector((abits -1) downto 0);
    re1    : in  std_ulogic;
    rdata1 : out std_logic_vector((dbits -1) downto 0);
    raddr2 : in  std_logic_vector((abits -1) downto 0);
    re2    : in  std_ulogic;
    rdata2 : out std_logic_vector((dbits -1) downto 0)
  );
  end component;
```

64 RSTGEN - Reset generation

64.1 Overview

The RSTGEN reset generator implements input reset signal synchronization with glitch filtering and generates the internal reset signal. The input reset signal can be asynchronous.

64.2 Operation

The reset generator latches the value of the clock lock signal on each rising edge of the clock. The lock signal serves as input to a five-bit shift register. The three most significant bits of this shift register are clocked into the reset output register. The reset signal to the system is high when both the reset output register and the reset input signal are high. Since the output register depends on the system clock the active low reset output from the core will go high synchronously to the system clock. The raw reset output does not depend on the system clock or clock lock signal and is polarity adjusted to be active low.

The VHDL generic *syncrst* determines how the core resets its shift register and the reset output register. When *syncrst* is set to 1 the core's shift register will have an synchronous reset and no reset signal will be connected to the output reset register, see figure 232. Note that the core's reset output signal will always go low when the input reset signal is activated.

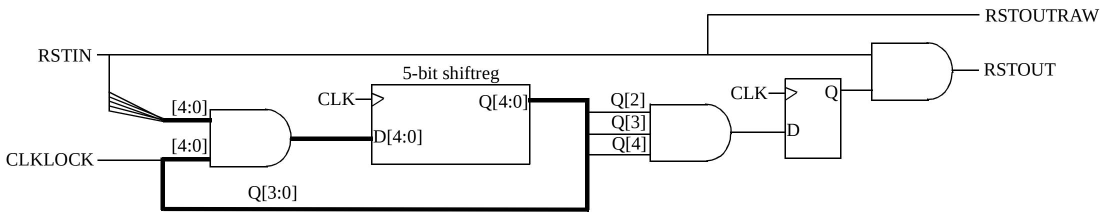


Figure 232. Reset generator with VHDL generic syncrst set to 1

When *syncrst* is 0 the shift register will be reset asynchronously together with the reset output register. Figure 233 shows the reset generator when scan test support is disabled. The shift register reset will be connected to the core's normal reset input and the test reset input will be unused. When scan test support is enabled, the core's test reset input can be connected to the reset input on both registers. The reset signal to use for the registers is selected with the test enable input, see figure 234.

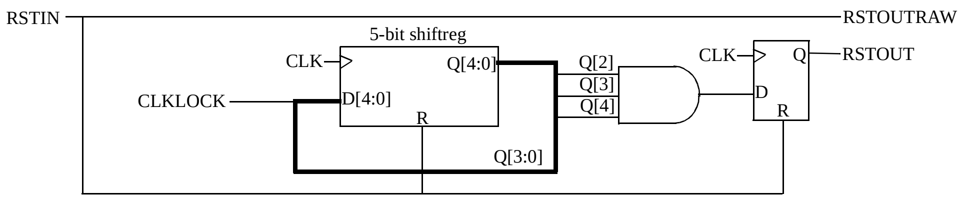


Figure 233. Reset generator with VHDL generic syncrst set to 0 and scan test disabled

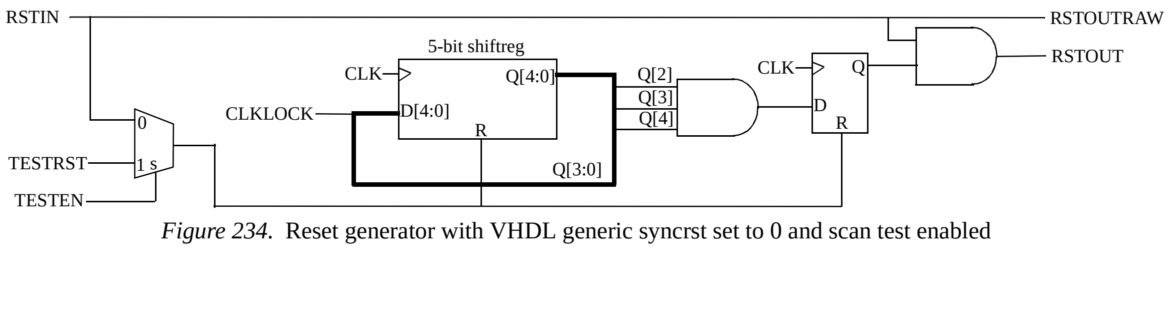


Figure 234. Reset generator with VHDL generic syncrst set to 0 and scan test enabled

64.3 Configuration options

Table 670 shows the configuration options of the core (VHDL generics).

*Table 670.*Configuration options

Generic name	Function	Allowed range	Default
acthigh	Set to 1 if reset input is active high. The core outputs an active low reset.		0
syncrst	When this generic is set to 1 the reset signal will use a synchronous reset to reset the filter registers. When this generic is set to 1 the TESTRST and TESTEN inputs will not be used.		0
scanen	Setting this generic to 1 enables scan test support. This connects the TESTRST input via a multiplexer so that the TESTRST and TESTEN signals can be used to asynchronously reset the core's registers. This also requires that the generic syncrst is set to 1.		0

64.4 Signal descriptions

Table 671 shows the interface signals of the core (VHDL ports).

*Table 671.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTIN	N/A	Input	Reset	-
CLK	N/A	Input	Clock	-
CLKLOCK	N/A	Input	Clock lock	High
RSTOUT	N/A	Output	Filtered reset	Low
RSTOUTRAW	N/A	Output	Raw reset	Low
TESTRST	N/A	Input	Test reset	-
TESTEN	N/A	Input	Test enable	High

64.5 Library dependencies

Table 672 shows the libraries used when instantiating the core (VHDL libraries).

*Table 672.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MISC	Component	Component definition

64.6 Instantiation

This example shows how the core can be instantiated together with the GRLIB clock generator.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
library gaisler;
use gaisler.misc.all;

entity rstgen_ex is
  port (
    resetn  : in  std_ulogic;
    clk     : in  std_ulogic; -- 50 MHz main clock
    pllref  : in  std_ulogic;
    testrst : in std_ulogic;
    testen  : in std_ulogic
  );
end;

architecture example of rstgen_ex is

signal lclk, clkm, rstn, rstraw, sdclkl, clk50: std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;

begin
  cgi.pllctrl <= "00"; cgi.pllrst <= rstraw;
  pllref_pad : clkpad generic map (tech => padtech) port map (pllref, cgi.pllref);
  clk_pad : clkpad generic map (tech => padtech) port map (clk, lclk);
  clkgen0 : clkgen  -- clock generator
    generic map (clktech, CFG_CLKMUL, CFG_CLKDIV, CFG_MCTRL_SDEN,
                 CFG_CLK_NOFB, 0, 0, 0, BOARD_FREQ)
    port map (lclk, lclk, clkm, open, open, sdclkl, open, cgi, cgo, open, clk50);
  sdclk_pad : outpad generic map (tech => padtech, slew => 1, strength => 24)
    port map (sdclk, sdclkl);

  resetn_pad : inpad generic map (tech => padtech) port map (resetn, rst);

  rst0 : rstgen  -- reset generator
    generic map (acthigh => 0, syncrst => 0, scanen => 1)
    port map (rst, clkm, cgo.clklock, rstn, rstraw, testrst, testen);
end;
```

65 SDCTRL - 32/64-bit PC133 SDRAM Controller

65.1 Overview

The SDRAM controller handles PC133 SDRAM compatible memory devices attached to 32 or 64 bit wide data bus. The controller acts as a slave on the AHB bus where it occupies configurable amount of address space for SDRAM access. The SDRAM controller function is programmed by writing to a configuration register mapped into AHB I/O address space.

Chip-select decoding is provided for two SDRAM banks.

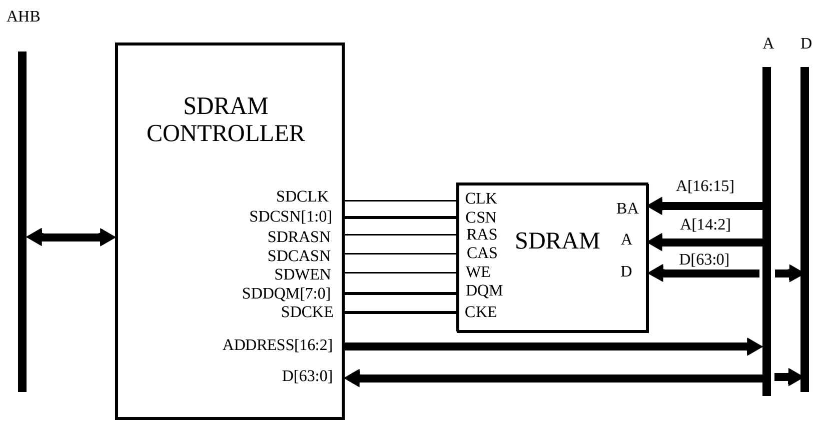


Figure 235. SDRAM Memory controller conected to AMBA bus and SDRAM

65.2 Operation

65.2.1 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 the configuration register SDCFG (see section 65.3). SDRAM banks data bus width is configurable between 32 and 64 bits. When the VHDL generic **mobile** is set to a value not equal to 0, the controller supports mobile SDRAM.

65.2.2 Initialization

When the SDRAM controller is enabled, it automatically performs the SDRAM initialization sequence of PRECHARGE, 8x AUTO-REFRESH and LOAD-MODE-REG on both banks simultaneously. When mobile SDRAM functionality is enabled the initialization sequence is appended by a LOAD-EXTMODE-REG command. The controller programs the SDRAM to use page burst on read and single location access on write. If the **pwron** VHDL generic is 1, the initialization sequence is also sent automatically when reset is released. Note that some SDRAM require a stable clock of 100 us before any commands might be sent. When using on-chip PLL, this might not always be the case and the **pwron** VHDL generic should be set to 0 in such cases.

65.2.3 Configurable SDRAM timing parameters

To provide optimum access cycles for different SDRAM devices (and at different frequencies), three SDRAM parameters can be programmed through memory configuration register 2 (MCFG2): TCAS, TRP and TRFCD. The value of these field affects the SDRAM timing as described in table 673.

*Table 673.*SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
CAS latency, RAS/CAS delay (t_{CAS}, t_{RCD})	TCAS + 2
Precharge to activate (t_{RP})	TRP + 2
Auto-refresh command period (t_{RFC})	TRFC + 3
Activate to precharge (t_{RAS})	TRFC + 1
Activate to Activate (t_{RC})	TRP + TRFC + 4

If the TCAS, TRP and TRFC are programmed such that the PC100/133 specifications are fulfilled, the remaining SDRAM timing parameters will also be met. The table below shows typical settings for 100 and 133 MHz operation and the resulting SDRAM timing (in ns):

*Table 674.*SDRAM example programming

SDRAM settings	t_{CAS}	t_{RC}	t_{RP}	t_{RFC}	t_{RAS}
100 MHz, CL=2; TRP=0, TCAS=0, TRFC=4	20	80	20	70	50
100 MHz, CL=3; TRP=0, TCAS=1, TRFC=4	30	80	20	70	50
133 MHz, CL=2; TRP=1, TCAS=0, TRFC=6	15	82	22	67	52
133 MHz, CL=3; TRP=1, TCAS=1, TRFC=6	22	82	22	67	52

When mobile SDRAM support is enabled, one additional timing parameter (TXSR) can be programmed though the Power-Saving configuration register.

*Table 675.*Mobile SDRAM programmable minimum timing parameters

SDRAM timing parameter	**Minimum timing (clocks)**
Exit Self Refresh mode to first valid command (t_{XSR})	tXSR

65.2.4 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 SDCFG register. Depending on SDRAM type, the required period is typically 7.8 or 15.6 µs (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 SDCFG register.

65.2.5 Self Refresh

The self refresh mode can be used to retain data in the SDRAM even when the rest of the system is powered down. When in the self refresh mode, the SDRAM retains data without external clocking and refresh are handled internally. The memory array that is refreshed during the self refresh operation is defined in the extended mode register. These settings can be changed by setting the PASR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the PASR bits are changed. The supported “Partial Array Self Refresh” modes are: Full, Half, Quarter, Eighth, and Sixteenth array. “Partial Array Self Refresh” is only supported when mobile SDRAM functionality is enabled. To enable the self refresh mode, set the PMODE bits in the Power-Saving configuration register to “010” (Self Refresh). The controller will enter self refresh mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits are cleared. When exiting this mode the controller introduce a delay defined by tXSR in the Power-Saving configuration register and a AUTO REFRESH command before any other memory access is allowed. The minimum duration of this mode is defined by tRAS. This mode is only available then the VHDL generic **mobile** >=1.

65.2.6 Power-Down

When entering the power-down mode all input and output buffers, excluding SDCKE, are deactivated. All data in the SDRAM is retained during this operation. To enable the power-down mode, set the PMODE bits in the Power-Saving configuration register to “001” (Power-Down). The controller will enter power-down mode after every memory access (when the controller has been idle for 16 clock cycles), until the PMODE bits is cleared. The REFRESH command will still be issued by the controller in this mode. When exiting this mode a delay of one clock cycles are added before issue any command to the memory. This mode is only available then the VHDL generic **mobile** >=1.

65.2.7 Deep Power-Down

The deep power-down operating mode is used to achieve maximum power reduction by eliminating the power of the memory array. Data will not be retained after the device enters deep power-down mode. To enable the deep power-down mode, set the PMODE bits in the Power-Saving configuration register to “101” (Deep Power-Down). To exit the deep power-down mode the PMODE bits in the Power-Saving configuration register must be cleared. The controller will respond with an AMBA ERROR response to an AMBA access, that will result in a memory access, during Deep Power-Down mode. This mode is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

65.2.8 Temperature-Compensated Self Refresh

The settings for the temperature-compensation of the Self Refresh rate can be controlled by setting the TCSR bits in the Power-Saving configuration register. The extended mode register is automatically updated when the TCSR bits are changed. Note that some vendors implements a Internal Temperature-Compensated Self Refresh feature, which makes the memory to ignore the TCSR bits. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

65.2.9 Drive Strength

The drive strength of the output buffers can be controlled by setting the DS bits in the Power-Saving configuration register. The extended mode register is automatically updated when the DS bits are changed. The available options are: full, three-quarter, one-half, and one-quarter drive strengths. This functionality is only available then the VHDL generic **mobile** >=1 and mobile SDRAM functionality is enabled.

65.2.10 SDRAM commands

The controller can issue four SDRAM commands by writing to the SDRAM command field in SDCFG: PRE-CHARGE, AUTO-REFRESH, LOAD-MODE-REG (LMR) and LOAD-EXTMODE-REG (EMR). If the LMR command is issued, the CAS delay as programmed in SDCFG will be used, remaining fields are fixed: page read burst, single location write, sequential burst. If the EMR command is issued, the DS, TCSR and PASR as programmed in Power-Saving configuration register will be used. 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.

65.2.11 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. Note that only word bursts are supported by the SDRAM controller. The AHB bus supports bursts of different sizes such as bytes and half-words but they cannot be used.

65.2.12 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. As in the read case, only word bursts are supported.

65.2.13 Address bus connection

The SDRAM address bus should be connected to SA[12:0], the bank address to SA[14:13], and the data bus to SD[31:0] or SD[63:0] if 64-bit data bus is used.

65.2.14 Data bus

The external SDRAM data bus is configurable to either 32 or 64 bits width, using the **sdbits** generic. 64-bit data bus allows 64-bit (SO)DIMS to be connected using the full data capacity of the devices. The polarity of the output enable signal to the data pads can be selected with the oepol generic. Sometimes it is difficult to fulfil the output delay requirements of the output enable signal. In this case, the vbdrive signal can be used instead of bdrive. Each index in this vector is driven by a separate register and a directive is placed on them so that they will not be removed by the synthesis tool.

65.2.15 Clocking

The SDRAM controller is designed for an external SDRAM clock that is in phase or slightly earlier than the internal AHB clock. This provides the maximum margin for setup and hold on the external signals, and allows highest possible frequency. For Xilinx and Altera device, the GRLIB Clock Generator (CLKGEN) can be configured to produce a properly synchronized SDRAM clock. For other FPGA targets, the custom clock synchronization must be designed, or the inverted clock option can be used (see below). For ASIC targets, the SDRAM clock can be derived from the AHB clock with proper delay adjustments during place&route.

If the VHDL generic INVCLK is set, then all outputs from the SDRAM controller are delayed for 1/2 clock. This is done by clocking all output registers on the falling clock edge. This option can be used on FPGA targets where proper SDRAM clock synchronization cannot be achieved. The SDRAM clock can be the internal AHB clock without further phase adjustments. Since the SDRAM signals will only have 1/2 clock period to propagate, this option typically limits the maximum SDRAM frequency to 40 - 50 MHz.

65.3 Registers

The memory controller is programmed through register(s) mapped into the AHB I/O space defined by the controllers AHB BAR1.

*Table 676.*SDRAM controller registers

AHB address offset	Register
0x0	SDRAM Configuration register
0x4	SDRAM Power-Saving configuration register

Table 677. SDRAM configuration register

31	30	29 27	26	25 23	22 21	20 18	17	16	15	14 0
Refresh	tRP	tRFC	tCD	SDRAM bank size	SRAM col. size	SDRAM commnad	Page-Burst	MS	D64	SDRAM refresh load value

31	SDRAM refresh. If set, the SDRAM refresh will be enabled.
30	SDRAM tRP timing. tRP will be equal to 2 or 3 system clocks (0/1). When mobile DDR support is enabled, this bit also represent the MSB in the tRFC timing.
29: 27	SDRAM tRFC timing. tRFC will be equal to 3 + field-value system clocks. When mobile DDR support is enabled, this field is extended with the bit 30.
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. Also sets RAS/CAS delay (tRCD).
25: 23	SDRAM banks size. Defines the decoded memory size for each SDRAM chip select: “000”= 4 Mbyte, “001”= 8 Mbyte, “010”= 16 Mbyte .... “111”= 512 Mbyte.
22: 21	SDRAM column size. “00”=256, “01”=512, “10”=1024, “11”=4096 when bit[25:23]= “111”, 2048 otherwise.
20: 18	SDRAM command. Writing a non-zero value will generate an SDRAM command: “010”=PRE-CHARGE, “100”=AUTO-REFRESH, “110”=LOAD-COMMAND-REGISTER, “111”=LOAD-EXTENDED-COMMAND-REGISTER. The field is reset after command has been executed.
17	1 = pageburst is used for read operations, 0 = line burst of length 8 is used for read operations. (Only available when VHDL generic pageburst i set to 2)
16	Mobile SDR support enabled. ‘1’ = Enabled, ‘0’ = Disabled (read-only)
15	64-bit data bus (D64) - Reads ‘1’ if memory controller is configured for 64-bit data bus, otherwise ‘0’. Read-only.
14: 0	The period between each AUTO-REFRESH command - Calculated as follows: tREFRESH = ((reload value) + 1) / SYSCLK

Table 678. SDRAM Power-Saving configuration register

31	30	29 24	23 20	19	18 16	15 7	6 5	4 3	2 0
ME	CE	Reserved	tXSR	res	PMODE	Reserved	DS	TCSR	PASR

31	Mobile SDRAM functionality enabled. ‘1’ = Enabled (support for Mobile SDRAM), ‘0’ = disabled (support for standard SDRAM)
30	Clock enable (CE). This value is driven on the CKE inputs of the SDRAM. Should be set to ‘1’ for correct operation. This register bit is read only when Power-Saving mode is other then none.
29: 24	Reserved
23: 20	SDRAM tXSR timing. tXSR will be equal to field-value system clocks. (Read only when Mobile SDR support is disabled).
19	Reserved

Table 678. SDRAM Power-Saving configuration register

18: 16	Power-Saving mode (Read only when Mobile SDR support is disabled). “000”: none “001”: Power-Down (PD) “010”: Self-Refresh (SR) “101”: Deep Power-Down (DPD)
15: 7	Reserved
6: 5	Selectable output drive strength (Read only when Mobile SDR support is disabled). “00”: Full “01”: One-half “10”: One-quarter “11”: Three-quarter
4: 3	Reserved for Temperature-Compensated Self Refresh (Read only when Mobile SDR support is disabled). “00”: 70ªC “01”: 45ªC “10”: 15ªC “11”: 85ªC
2: 0	Partial Array Self Refresh (Read only when Mobile SDR support is disabled). “000”: Full array (Banks 0, 1, 2 and 3) “001”: Half array (Banks 0 and 1) “010”: Quarter array (Bank 0) “101”: One-eighth array (Bank 0 with row MSB = 0) “110”: One-sixteenth array (Bank 0 with row MSB = 00)

65.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x009. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

65.5 Configuration options

Table 679 shows the configuration options of the core (VHDL generics).

*Table 679.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
haddr	ADDR field of the AHB BAR0 defining SDRAM area. Default is 0xF0000000 - 0xFFFFFFFF.	0 - 16#FFF#	16#000#
hmask	MASK field of the AHB BAR0 defining SDRAM area.	0 - 16#FFF#	16#F00#
ioaddr	ADDR field of the AHB BAR1 defining I/O address space where SDCFG register is mapped.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR1 defining I/O address space.	0 - 16#FFF#	16#FFF#
wprot	Write protection.	0 - 1	0
invclk	Inverted clock is used for the SDRAM.	0 - 1	0
pwron	Enable SDRAM at power-on initialization	0 - 1	0
sdbits	32 or 64-bit data bus width.	32, 64	32
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
pageburst	Enable SDRAM page burst operation. 0: Controller uses line burst of length 8 for read operations. 1: Controller uses pageburst for read operations. 2: Controller uses pageburst/line burst depending on PageBurst bit in SDRAM configuration register.	0 - 2	0
mobile	Enable Mobile SDRAM support 0: Mobile SDRAM support disabled 1: Mobile SDRAM support enabled but not default 2: Mobile SDRAM support enabled by default 3: Mobile SDRAM support only (no regular SDR support)	0 - 3	0

65.6 Signal descriptions

Table 680 shows the interface signals of the core (VHDL ports).

*Table 680.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDI	WPROT	Input	Not used	-
	DATA[63:0]	Input	Data	High
SDO	SDCKE[1:0]	Output	SDRAM clock enable	High
	SDCSN[1:0]	Output	SDRAM chip select	Low
	SDWEN	Output	SDRAM write enable	Low
	RASN	Output	SDRAM row address strobe	Low
	CASN	Output	SDRAM column address strobe	Low
	DQM[7:0]	Output	SDRAM data mask: DQM[7] corresponds to DATA[63:56], DQM[6] corresponds to DATA[55:48], DQM[5] corresponds to DATA[47:40], DQM[4] corresponds to DATA[39:32], DQM[3] corresponds to DATA[31:24], DQM[2] corresponds to DATA[23:16], DQM[1] corresponds to DATA[15:8], DQM[0] corresponds to DATA[7:0].	Low
	BDRIVE	Output	Drive SDRAM data bus	Low/High^2
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High^2
	ADDRESS[16:2]	Output	SDRAM address	Low
	DATA[31:0]	Output	SDRAM data	Low

1) see GRLIB IP Library User's Manual

2) Polarity selected with the oepol generic

65.7 Library dependencies

Table 681 shows libraries used when instantiating the core (VHDL libraries).

*Table 681.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

65.8 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the SDRAM controller. The external SDRAM bus is defined on the example designs port map and connected to the SDRAM controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

SDRAM controller decodes SDRAM area:0x60000000 - 0x6FFFFFFF. SDRAM Configuration register is mapped into AHB I/O space on address (AHB I/O base address + 0x100).

```
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;

entity mctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;
sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
    );
end;

architecture rtl of mctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

signal sdi   : sdctrl_in_type;
  signal sdo   : sdctrl_out_type;

  signal clkm, rstn : std_ulogic;
signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;
  signal gnd : std_ulogic;

begin

  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                                tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);

  -- SDRAM controller
  sdc : sdctrl generic map (hindex => 3, haddr => 16#600#, hmask => 16#F00#,
    ioaddr => 1, pwron => 0, invclk => 0)
    port map (rstn, clkm, ahbsi, ahbso(3), sdi, sdo);

  -- input signals
  sdi.data(31 downto 0) <= sd(31 downto 0);

  -- connect SDRAM controller outputs to entity output signals
  sa <= sdo.address; sdcke <= sdo.sdcke; sdwen <= sdo.sdwen;
  sdcsn <= sdo.sdcsn; sdrasn <= sdo.rasn; sdcasn <= sdo.casn;
  sddqm <= sdo.dqm;

--Data pad instantiation with scalar bdrive
sd_pad : iopadv generic map (width => 32)
port map (sd(31 downto 0), sdo.data, sdo.bdrive, sdi.data(31 downto 0));
end;

--Alternative data pad instantiation with vectored bdrive
sd_pad : iopadvv generic map (width => 32)
port map (sd(31 downto 0), sdo.data, sdo.vbdrive, sdi.data(31 downto 0));
end;
```

66 SPICTRL - SPI Controller

66.1 Overview

The core provides a link between the AMBA APB bus and the Serial Peripheral Interface (SPI) bus. Through registers mapped into APB address space the core can be configured to work either as a master or a slave. The SPI bus parameters are highly configurable via registers, the core has configurable word length, bit ordering and clock gap insertion. All SPI modes are supported and also a 3-wire mode where the core uses a bidirectional data line. To facilitate reuse of existing software drivers the controller's interface is compatible with the SPI controller interface in the Freescale MPC83xx series. In slave mode the core synchronizes the incoming clock and can operate in systems where other SPI devices are driven by asynchronous clocks.

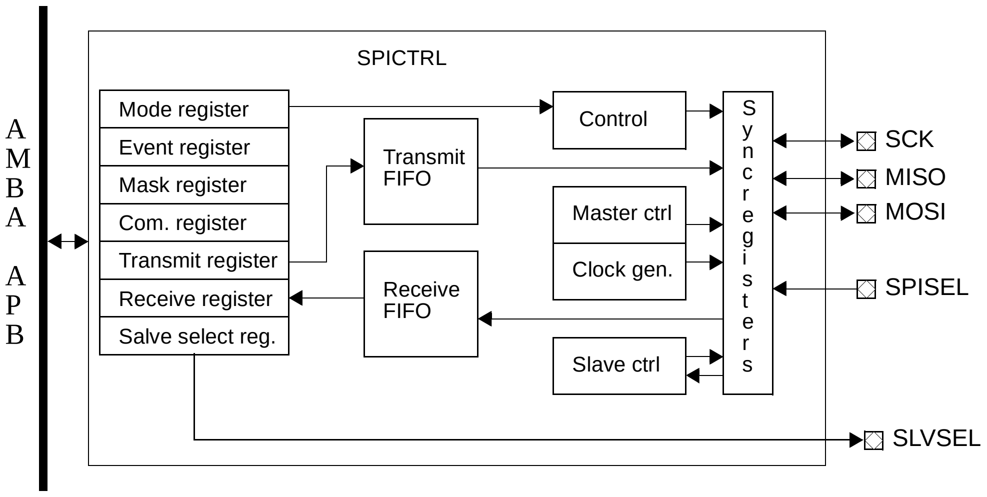


Figure 236. Block diagram

66.2 Operation

66.2.1 SPI transmission protocol

The SPI bus is a full-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal and from the slave through the Master-Input-Slave-Output (MISO) signal. In a system with only one master and one slave, the Slave Select input of the slave may be always active and the master does not need to have a slave select output. If the core is configured as a master it will monitor the SPISEL signal to detect collisions with other masters, if SPISEL is activated the master will be disabled.

During a transmission on the SPI bus data is either changed or read at a transition of SCK. If data has been read at edge n, data is changed at edge n+1. If data is read at the first transition of SCK the bus is said to have clock phase 0, and if data is changed at the first transition of SCK the bus has clock phase 1. The idle state of SCK may be either high or low. If the idle state of SCK is low, the bus has clock polarity 0 and if the idle state is high the clock polarity is 1. The combined values of clock polarity (CPOL) and clock phase (CPHA) determine the mode of the SPI bus. Figure 237 shows one byte (0x55) being transferred MSb first over the SPI bus under the four different modes. Note that the idle state of the MOSI line is '1' and that CPHA = 0 means that the devices must have data ready before the first transition of SCK. The figure does not include the MISO signal, the behavior of this line is the same as for the MOSI signal. However, due to synchronization issues the MISO signal will be delayed when the core is operating in slave mode, please see section 66.2.5 for details.

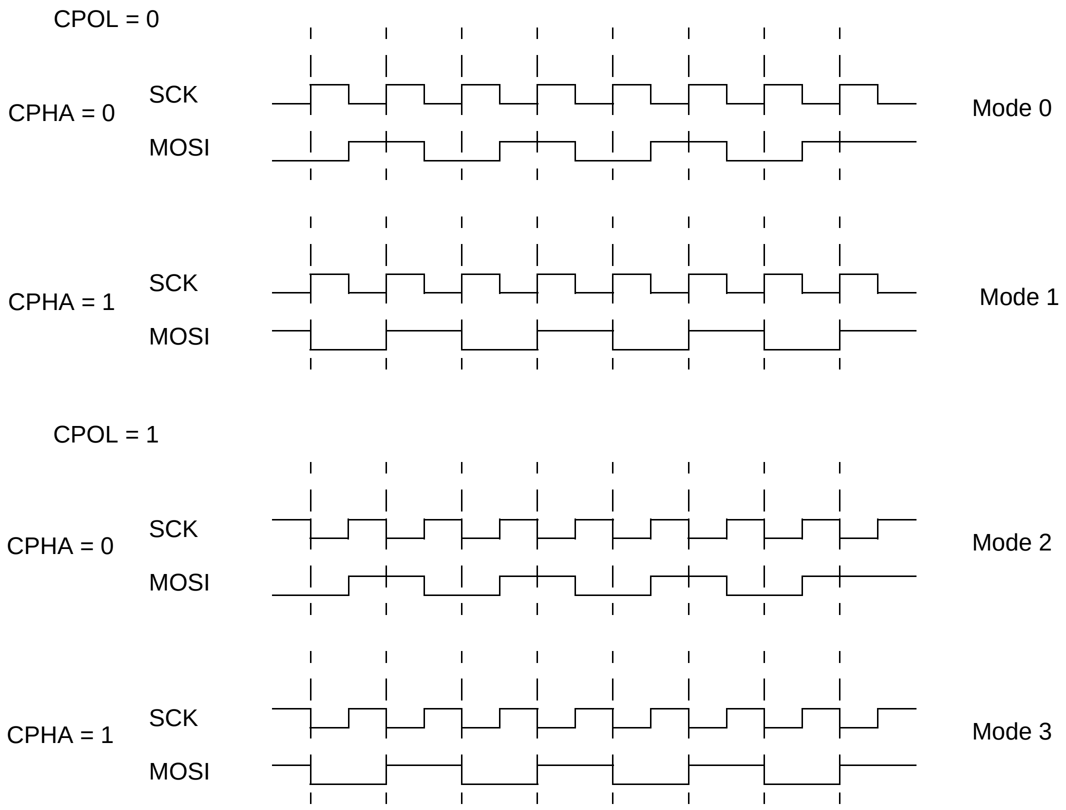


Figure 237. SPI transfer of byte 0x55 in all modes

66.2.2 3-wire transmission protocol

The core can be configured to operate in 3-wire mode where the controller uses a bidirectional data-line instead of separate data lines for input and output data. In 3-wire mode the bus is thus a half-duplex synchronous serial bus. Transmission starts when a master selects a slave through the slave's Slave Select (SLVSEL) signal and the clock line SCK transitions from its idle state. Data is transferred from the master through the Master-Output-Slave-Input (MOSI) signal. After a word has been transferred the slave uses the same Master-Output-Slave-Input (MOSI) data line to transfer a word back to the master. The MISO signal is not used in 3-wire mode.

The data line transitions depending on the clock polarity and clock phase in the same manner as in SPI mode. The afformentioned slave delay of the MISO signal in SPI mode will affect the MOSI signal in 3-wire mode, when the core operates as a slave.

66.2.3 Receive and transmit queues

The core's transmit queue consists of the transmit register and the transmit FIFO. The receive queue consists of the receive register and the receive FIFO. The total number of words that can exist in each queue is thus the FIFO depth plus one. When the core has one or more free slots in the transmit queue it will assert the Not full (NF) bit in the event register. Software may only write to the transmit register when this bit is asserted. When the core has received a word, as defined by word length (LEN) in the Mode register, it will place the data in the receive queue. When the receive queue has one or more elements stored the Event register bit Not empty (NE) will be asserted. The receive register will only contain valid data if the Not empty bit is asserted and software should not access the receive register unless this bit is set. If the receive queue is full and the core receives a new word, an overrun condition will occur. The received data will be discarded and the Overrun (OV) bit in the Event register will be set.

The core will also detect underrun conditions. An underrun condition occurs when the core is selected, via SPISEL, and the SCK clock transitions while the transmit queue is empty. In this scenario the core will respond with all bits set to ‘1’ and set the Underrun (UN) bit in the Event register. An underrun condition will never occur in master mode. When the master has an empty transmit queue the bus will go into an idle state.

66.2.4 Clock generation

The core only generates the clock in master mode, the generated frequency depends on the system clock frequency and the Mode register fields DIV16 and PM. Without DIV16 the SCK frequency is:

$$SCKFrequency = \frac{AMBAclockfrequency}{4 \cdot (PM + 1)}$$

With DIV16 enabled the frequency of SCK is derived through:

$$SCKFrequency = \frac{AMBAclockfrequency}{16 \cdot (4 \cdot (PM + 1))}$$

Note that the fields of the Mode register, which includes DIV16 and PM, should not be changed when the core is enabled.

66.2.5 Slave operation

When the core is configured for slave operation it does not drive any SPI signal until the core is selected, via the SPISEL input, by a master. If the core operates in SPI mode when SPISEL goes low the core configures MISO as an output and drives the value of the first bit scheduled for transfer. If the core is configured into 3-wire mode the core will first listen to the MOSI line and when a word has been transfered drive the response on the MOSI line. If the core is selected when the transmit queue is empty it will transfer a word with all bits set to ‘1’ and the core will report an underflow.

Since the core synchronizes the incoming clock it will not react to transitions on SCK until two system clock cycles have passed. This leads to a delay of three system clock cycles when the data output line should change as the result of a SCK transition. This constrains the maximum input SCK frequency of the slave to (system clock) / 8 or less. The controlling master must also allow the decreased setup time on the slave data out line.

66.2.6 Master operation

When the core is configured for master operation it will transmit a word when there is data available in the transmit queue. When the transmit queue is empty the core will drive SCK to its idle state. If the SPISEL input goes low during master operation the core will abort any active transmission and the Multiple-master error (MME) bit will be asserted in the Event register. If a Multiple-master error occurs the core will be disabled. Note that the core will react to changes on SPISEL even if the core is operating in loop mode.

66.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 682.*SPI controller registers

APB address offset	Register
0x00	Capability register
0x04-0x1C	Reserved
0x20	Mode register
0x24	Event register
0x28	Mask register
0x2C	Command register
0x30	Transmit register
0x34	Receive register
0x38	Slave Select register (optional)

Table 683. SPI controller Capability register

31 24	23 16	15 8	7 0
SSSZ	SSEN	FDEPTH	REV

31 : 24	Slave Select register size (SSSZ) - If the core has been configured with slave select signals this field contains the number of available signals.
23:16	Slave Select Enable (SSEN) - If the core has a slave select register, and corresponding slave select lines, the value of this field is one. Otherwise the value of this field is zero.
15:8	FIFO depth (FDEPTH) - This field contains the depth of the core’s internal FIFOs. The number of words the core can store in each queue is FDEPTH+1, since the transmit and receive registers can contain one word each.
7:0	Core revision (REV)

Table 684. SPI controller Mode register

31	30	29	28	27	26	25	24	23 20	19 16
R	LOOP	CPOL	CPHA	DIV16	REV	MS	EN	LEN	PM

15	14 12	11 7	6 0
TWEN		CG	R

31	RESERVED (R) - Read as zero and should be written as zero to ensure forward compatibility.
30	Loop mode (LOOP) - When this bit is set, and the core is enabled, the core’s transmitter and receiver are interconnected and the core will operate in loopback mode. The core will still detect, and will be disabled, on Multiple-master errors.
29	Clock polarity (CPOL) - Determines the polarity (idle state) of the SCK clock.
28	Clock phase (CPHA) - When CPOL is ‘0’ data will be read on the first transition of SCK. When CPOL is ‘1’ data will be read on the second transition of SCK.
27	Divide by 16 (DIV16) - Divide system clock by 16, see description of PM field below and see section 66.2.4 on clock generation. This bit has no significance in slave mode.
26	Reverse data (REV) - When this bit is ‘0’ data is transmitted LSB first, when this bit is ‘1’ data is transmitted MSB first. This bit affects the layout of the transmit and receive registers.
25	Master/Slave (MS) - When this bit is set to ‘1’ the core will act as a master, when this bit is set to ‘0’ the core will operate in slave mode.
24	Enable core (EN) - When this bit is set to ‘1’ the core is enabled. No fields in the mode register should be changed while the core is enabled. This can bit can be set to ‘0’ by software, or by the core if a multiple-master error occurs.

Table 684. SPI controller Mode register

23:20	Word length (LEN) - The value of this field determines the length in bits of a transfer on the SPI bus. Values are interpreted as: 0b0000 - 32-bit word length 0b0001-0b0010 - Illegal values 0b0011-0b1111 - Word length is LEN+1, allows words of length 4-16 bits.
19:16	Prescale modulus (PM) - This value is used in master mode to divide the system clock and generate the SPI SCK clock. The system clock is divided by 4*(PM+1) if the DIV16 field is ‘0’ and 16*4*(PM+1) if the DIV16 field is set to ‘1’. The highest SCK frequency is attained when PM is set to 0b0000 and DIV16 to ‘0’, this configuration will give a SCK frequency that is (system clock)/4.
15	Three-wire mode (tw) - If this bit is set to ‘1’ the core will operate in 3-wire mode.
14:12	RESERVED (R) - Read as zero and should be written as zero to ensure forward compatibility.
11:7	Clock gap (CG) - The value of this field is only significant in master mode. The core will insert CG SCK clock cycles between each consecutive word. This only applies when the transmit queue is kept non-empty. After the last word of the transmit queue has been sent the core will go into an idle state and will continue to transmit data as soon as a new word is written to the transmit register, regardless of the value in CG. A value of 0b00000 in this field enables back-to-back transfers.
6:0	RESERVED (R) - Read as zero and should be written as zero to ensure forward compatibility.

Table 685. SPI controller Event register

31 15	14	13	12	11	10	9	8	7 0
R	LT	R	OV	UN	MME	NE	NF	R

31 : 15	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
14	Last character (LT) - This bit is set when a transfer completes if the transmit queue is empty and the LST bit in the Command register has been written. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
13	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
12	Overrun (OV) - This bit gets set when the receive queue is full and the core receives new data. The core continues communicating over the SPI bus but discards the new data. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
11	Underrun (UN) - This bit is only set when the core is operating in slave mode. The bit is set if the core’s transmit queue is empty when a master initiates a transfer. When this happens the core will respond with a word where all bits are set to ‘1’. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
10	Multiple-master error (MME) - This bit is set when the core is operating in master mode and the SPISEL input goes active. In addition to setting this bit the core will be disabled. This bit is cleared by writing ‘1’, writes of ‘0’ have no effect.
9	Not empty (NE) - This bit is set when the receive queue contains one or more elements. It is cleared automatically by the core, writes have no effect.
8	Not full (NF) - This bit is set when the transmit queue has room for one or more words. It is cleared automatically by the core when the queue is full, writes have no effect.
7:0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

Table 686. SPI controller Mask register

31 15	14	13	12	11	10	9	8	7 0
R	LTE	R	OVE	UNE	MMEE	NEE	NFE	R

31 : 15	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
14	Last character enable (LTE) - When this bit is set the core will generate an interrupt when the LT bit in the Event register transitions from ‘0’ to ‘1’.
13	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.
12	Overrun enable (OVE) - When this bit is set the core will generate an interrupt when the OV bit in the Event register transitions from ‘0’ to ‘1’.
11	Underrun enable (UNE) - When this bit is set the core will generate an interrupt when the UN bit in the Event register transitions from ‘0’ to ‘1’.

Table 686. SPI controller Mask register

10	Multiple-master error enable (MMEE) - When this bit is set the core will generate an interrupt when the MME bit in the Event register transitions from ‘0’ to ‘1’.
9	Not empty enable (NEE) - When this bit is set the core will generate an interrupt when the NE bit in the Event register transitions from ‘0’ to ‘1’.
8	Not full enable (NFE) - When this bit is set the core will generate an interrupt when the NF bit in the Event register transitions from ‘0’ to ‘1’.
7:0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatibility.

Table 687. SPI controller Command register

31	23	22	21	0
R		LST	R	

31 : 23	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatiability.
22	Last (LST) - After this bit has been written to ‘1’ the core will set the Event register bit LT when a character has been transmitted and the transmit queue is empty. If the core is operating in 3-wire mode the Event register bit is set when the whole transfer has completed. This bit is automatically cleared when the Event register bit has been set and is always read as zero.
21:0	RESERVED (R) - Read as zero and should be written to zero to ensure forward compatiability.

Table 688. SPI controller Transmit register

31	23	22	21	0
TDATA				

31 : 0	Transmit data (TDATA) - Writing a word into this register places the word in the transmit queue. This register will only react to writes if the Not full (NF) bit in the Event register is set. The layout of this register depends on the value if the REV field in the Mode register: Rev = ‘0’: The word to transmit should be written with its least significant bit at bit 0. Rev = ‘1’: The word to transmit should be written with its most significant bit at bit 31.

Table 689. SPI controller Receive register

31	23	22	21	0
RDATA				

31 : 0	Receive data (RDATA) - This register contains valid receive data when the Not empty (NE) bit of the Event register is set. The placement of the received word depends on the Mode register fields LEN and REV: For LEN = 0b0000 - The data is placed with its MSb in bit 31 and its LSb in bit 0. For other lengths and REV = ‘0’ - The data is placed with its MSB in bit 15. For other lengths and REV = ‘1’ - The data is placed with its LSB in bit 16. To illustrate this, a transfer of a word with eight bits (LEN = 7) that are all set to one will have the following placement: REV = ‘0’ - 0x0000FF00 REV = ‘1’ - 0x00FF0000

Table 690. SPI Slave select register (optional)

31	SSSZ	SSSZ-1	0
R		SLVSEL	

31 : SSSZ	RESERVED (R) - The lower bound of this register is determined by the Capbility register field SSSZ if the SSEN field is set to 1. If SSEN is zero bits 31:0 are reserved.
(SSSZ-1) : 0	Slave select (SLVSEL) - If SSEN in the Capability register is 1 the core’s slave select signals are mapped to this register on bits (SSSZ-1):0. Software is soley responsible for activating the correct slave select signals, the core does not assert or deassert any slave select signal automatically.

66.4 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x02D. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

66.5 Configuration options

Table 691 shows the configuration options of the core (VHDL generics).

*Table 691.*Configuration options

Generic name	Function	Allowed range	Default
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	ADDR field of the APB BAR.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR.	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by SPI controller	0 - NAHBIRQ-1	0
fdepth	FIFO depth. The FIFO depth in the core is 2^{fdepth}. Note that the depth of the transmit and receive queues is FIFO depth + 1 since the Transmit and Receive registers can hold one word.	1 - 7	1
slvselen	Enable Slave Select register. When this value is set to 1 the core will include a slave select register that controls slvselsz slave select signals.	0 - 1	0
slvselsz	Number of Slave Select (slvsel) signals that the core will generate. These signals can be controlled via the Slave select register if the generic slvselen has been set to 1, otherwise they are driven to '1'.	1 - 32	1
oepol	Selects output enable polarity	0 - 1	0

66.6 Signal descriptions

Table 692 shows the interface signals of the core (VHDL ports).

*Table 692.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
SPII	MISO	Input	Master-Input-Slave-Output data line, not used in 3-wire mode.	-
	MOSI	Input	Master-Output-Slave-Input data line	-
	SCK	Input	Serial Clock	-
	SPISEL	Input	Slave select input	-
SPIO	MISO	Output	Master-Input-Slave-Output data line, not used in 3-wire mode.	-
	MISOOEN	Output	Master-Input-Slave-Output output enable, not used in 3-wire mode.	Low
	MOSI	Output	Master-Output-Slave-Input	-
	MOSIOEN	Output	Master-Output-Slave-Input output enable	Low
	SCK	Output	Serial Clock	-
	SCKOEN	Output	Serial Clock output enable	Low

* see GRLIB IP Library User’s Manual

66.7 Library dependencies

Table 693 shows the libraries used when instantiating the core (VHDL libraries).

*Table 693.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	SPI component and signal definitions.

66.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.misc.all;

entity spi_ex is
  port (
    clk    : in std_ulogic;
    rstn   : in std_ulogic;

    -- SPI signals
    sck    : inout std_ulogic;
    miso   : inout std_ulogic;
    mosi   : inout std_ulogic;
    spisel : in std_ulogic
    );
end;

architecture rtl of spi_ex is

  -- AMBA signals
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);

  -- SPIsignals
  signal spii : spi_in_type;
  signal spio : spi_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- SPI controller with FIFO depth 2 and no slave select register
  spictrl0 : spictrl generic map (pindex => 10, paddr => 10, pirq => 10,
                                  fdepth => 1, slvselen => 0, slvselsz => 1)
      port map (rstn, clkm, apbi, apbo(10), spii, spio, open);

    misopad : iopad generic map (tech => padtech)
      port map (miso, spio.miso, spio.misooen, spii.miso);
    mosipad : iopad generic map (tech => padtech)
      port map (mosi, spio.mosi, spio.mosioen, spii.mosi);
    sckpad : iopad generic map (tech => padtech)
      port map (sck, spio.sck, spio.sckoen, spii.sck);
    spiselpad : inpad generic map (tech => padtech)
      port map (spisel, spii.spisel);
end;
```

67 SPIMCTRL - SPI Memory Controller

67.1 Overview

The core maps a memory device connected via the Serial Peripheral Interface (SPI) into AMBA address space. Read accesses are performed by performing normal AMBA read operations in the mapped memory area. Other operations, such as writes, are performed by directly sending SPI commands to the memory device via the core's register interface. The core is highly configurable and supports most SPI Flash memory devices. The core also has limited, SPI-mode, SD card support.

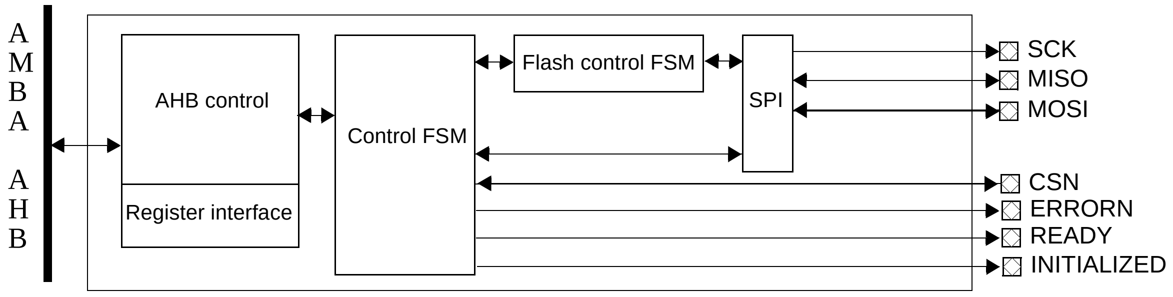


Figure 238. Block diagram

67.2 Operation

67.2.1 Operational model

The core has two memory areas that can be accessed via the AMBA bus; the I/O area and the ROM area. The ROM area maps the memory device into AMBA address space and the I/O area is utilized for status reporting and to issue user commands to the memory device.

When transmitting SPI commands directly to the device the ROM area should be left untouched. The core will issue an AMBA ERROR response if the ROM area is accessed when the core is busy performing an operation initiated via I/O registers.

Depending on the type of device attached the core may need to perform an initialization sequence. Accesses to the ROM area during the initialization sequence receive AMBA error responses. The core has successfully performed all necessary initialization when the Initialized bit in the core's status register is set, the value of this bit is also propagated to the core's output signal *spio.initialized*.

67.2.2 I/O area

The I/O area contains registers that are used when issuing commands directly to the memory device. By default, the core operates in System mode where it will perform read operations on the memory device when the core's ROM area is accessed. Before attempting to issue commands directly to the memory device, the core must be put into User mode. This is done by setting the User Control (USRC) bit in the core's Control register. Care should be taken to not enter User mode while the core is busy, as indicated by the bits in the Status register. The core should also have performed a successful initialization sequence before User mode accesses (INIT bit in the Status register should be set).

The following steps are performed to issue a command to the memory device after the core has been put into User mode:

1. Check Status register and verify that the BUSY and DONE bits are cleared. Also verify that the core is initialized and not in error mode.
2. Optionally enable DONE interrupt by setting the Control register bit IEN.
3. Write command to Transmit register.
4. Wait for interrupt (if enabled) or poll DONE bit in Status register.

5. When the DONE bit is set the core has transferred the command and will have new data available in the Receive register.
6. Clear the Status register's DONE bit by writing one to its position.

The core should not be brought out of User mode until the transfer completes. Accesses to ROM address space will receive an AMBA ERROR response when the core is in User mode and when an operation initiated under User mode is active.

67.2.3 ROM area

The ROM area only supports AMBA read operations. Write operations will receive AMBA ERROR responses. When a read access is made to the ROM area the core will perform a read operation on the memory device. If the system has support for AMBA SPLIT responses the core will SPLIT the master until the read operation on the memory device has finished, unless the read operation is a locked access. A locked access never receives a SPLIT response and the core inserts wait states instead. If the system lacks AMBA SPLIT support the core will always insert wait states until the read operation on the memory device has finished. The core uses the value of the VHDL generic *spliten* to determine if the system has AMBA SPLIT support.

When the core is configured to work with a SD card, two types of timeouts are taken into account. The SD card is expected to respond to a command within 100 bytes and the core will wait for a data token following a read command for 312500 bytes. If the SD card does not respond within these limits the core will issue an AMBA error response to the access.

If an error occurs during an access to the ROM area the core will respond with an AMBA ERROR response. It will also set one or both of the Status register bits Error (ERR) and Timeout (TO). If the Error (ERR) bit remains set, subsequent accesses to the ROM area will also receive AMBA ERROR responses. The core can only detect failures when configured for use with a SD card.

67.2.4 Supported memory devices

The core supports a wide range of memory devices due to its configuration options of read instruction, dummy byte insertion and dual output. Table 694 below lists configurations for some memory devices.

*Table 694.*Configurations for some memory devices

		VHDL generic*			
Manufacturer	**Memory device**	**sdcard**	**readcmd****	**dummybyte**	**dualoutput**
SD card	SD card	1	-	-	-
Spansion	S25FL-series	0	0x0B	1	0
Winbond	W25X-series	0	0x0B	1	0
	W25X-series with dual output read.	0	0x3B	1	1
* '-' means don't care ** Available in the core's Configuration register.					

When the core is configured for use with a SD card it does not use the VHDL generics *readcmd*, *dummybyte* nor *dualoutput*. All SD cards are initialized to use a block length of four bytes and accesses to the ROM area lead to a READ_SINGLE_BLOCK command being issued to the card.

When the VHDL generic *sdcard* is set to 0 the core is configured to issue the instruction defined by the VHDL generic *readcmd* to obtain data from the device. After an access to the ROM area the core will issue the read instruction followed by 24 address bits. If the VHDL generic *dummybyte* is set to 1 the core will issue a dummy byte following the address. After the possible dummy byte the core expects to receive data from the device. If the VHDL generic *dualoutput* is set to 1 the core will read data on both the MISO and MOSI data line. Otherwise the core will only use the MISO line for input data.

Many memory devices support both a READ and a FAST_READ instruction. The FAST_READ instruction can typically be issued with higher device clock frequency compared to the READ instruction, but requires a dummy byte to be present after the address. The most suitable choice of read instruction depends on the system frequency and on the memory device's characteristics.

67.2.5 Clock generation and power-up timing

The core generates the device clock by scaling the system clock. The VHDL generic *scaler* selects the divisor to use for the device clock that is used when issuing read instructions. The VHDL generic *altscaler* defines an alternate divisor that is used to generate the clock during power-up. This alternate divisor is used during initialization of SD cards and should be selected to produce a clock of 400 kHz or less when the core is configured for use with an SD card. The alternate clock can also be used when issuing user mode commands by setting the Enable Alternate Scaler (EAS) bit in the Control register.

When the core is configured for a non-SD card (VHDL generic *sdcard* set to 0), the VHDL generic *pwrupcnt* specifies how many system clock cycles after reset the core should be idle before issuing the first command.

67.3 Registers

The core is programmed through registers mapped into AHB address space.

*Table 695.*GRSLINK registers

APB address offset	**Register**
0x00	Configuration register
0x04	Control register
0x08	Status register
0x0C	Receive register
0x10	Transmit register

Table 696. SPIMCTRL Configuration register

31 8	7 0
RESERVED	READCMD

31 :8	RESERVED
7:0	Read instruction (READCMD) - Read instruction that the core will use for reading from the memory device. When the core is configured to interface with a SD card this field will be set to 0.

Reset value: Implementation dependent

Table 697. SPIMCTRL Control register

31 5	4	3	2	1	0
RESERVED	RST	CSN	EAS	IEN	USRC

31 :5	RESERVED
4	Reset core (RST) - By writing '1' to this bit the user can reset the core. This bit is automatically cleared when the core has been reset. Reset core should be used with care. Writing this bit has the same effect as system reset. Any ongoing transactions, both on AMBA and to the SPI device will be aborted.

Table 697. SPIMCTRL Control register

3	Chip select (CSN) - Controls core chip select signal. This field always shows the level of the core's internal chip select signal. This bit is always automatically set to '1' when leaving User mode by writing USRC to '0'.
2	Enable Alternate Scaler (EAS) - When this bit is set the SPI clock is divided by using the alternate scaler.
1	Interrupt Enable (IEN) - When this bit is set the core will generate an interrupt when a User mode transfer completes.
0	User control (USRC) - When this bit is set to '1' the core will accept SPI data via the transmit register. Accesses to the memory mapped device area will return AMBA ERROR responses.

Reset value: 0x00000008

Table 698. SPIMCTRL Status register

31 6	5	4	3	2	1	0
RESERVED	CD	TO	ERR	INIT	BUSY	DONE

31:6	RESERVED
5	Card Detect (CD) - This bit shows the value of the core's CD input signal if the core has been configured to work with an SD card. When using the core with a device that is hot-pluggable it may be necessary to monitor this bit and reset the core if a device has been disconnected and reconnected. This bit is only valid if the core has been configured for use with SD cards, otherwise it is always '0'.
4	Timeout (TO) - This bit is set to '1' when the core times out while waiting for a response from a SD card. This bit is only used when the core is configured for use with a SD card. The state is refreshed at every read operation that is performed as a result of an access to the ROM memory area. User mode accesses can never trigger a timeout. This bit is read only.
3	Error (ERR) - This bit is set to '1' when the core has entered error state. When the core is in this state all accesses to the ROM memory area will receive AMBA ERROR responses. The error state can be cleared by writing '1' to this bit. If the core entered error state during a read operation the ROM area will be available for read accesses again. This bit is follows the negated errorn output signal. The core will only detect errors when configured for use with an SD card. User mode accesses can never trigger an error.
2	Initialized (INIT) - This read only bit is set to '1' when the SPI memory device has been initialized. Accesses to the ROM area should only be performed when this bit is set to '1'.
1	Core busy (BUSY) - This bit is set to '1' when the core is performing an SPI operation.
0	Operation done (DONE) - This bit is set to '1' when the core has transferred an SPI command in user mode.

Reset value: 0x00000000

Table 699. SPIMCTRL Receive register

31 8	7 0
RESERVED	RDATA

31 :8	RESERVED
7:0	Receive data (RDATA) : Contains received data byte

Reset value: 0x000000UU, where U is undefined

Table 700. SPIMCTRL Transmit register

31 8	7 0
	TDATA

Table 700. SPIMCTRL Transmit register

31 :8	RESERVED
7:0	Transmit data (TDATA) - Data byte to transmit

Reset value: 0x00000000

67.4 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x045. For description of vendor and device identifiers see GRLIB IP Library User's Manual.

67.5 Implementation

67.5.1 Technology mapping

The core does not instantiate any technology specific primitives.

67.5.2 RAM usage

The core does not use any RAM components.

67.6 Configuration options

Table 701 shows the configuration options of the core (VHDL generics).

*Table 701.*Configuration options

Generic name	Function	Allowed range	Default
hindex	AHB slave index	0 - (NAHBSLV-1)	0
hirq	Interrupt line	0 - (NAHBIRQ-1)	0
faddr	ADDR field of the AHB BAR1 defining ROM address space.	0 - 16#FFF#	16#000#
fmask	MASK field of the AHB BAR1 defining ROM address space.	0 - 16#FFF#	16#FFF#
ioaddr	ADDR field of the AHB BAR0 defining register address space.	0 - 16#FFF#	16#000#
iomask	MASK field of the AHB BAR0 defining register space.	0 - 16#FFF#	16#FFF#
spliten	If this generic is set to 1 the core will issue AMBA SPLIT responses when it is busy performing an operation on the memory device. Otherwise the core will insert wait states until the operation completes.	0 - 1	0
oepol	Select polarity of output enable signals. 0 = active low.	0 - 1	0
sdcard	Enable support for SD card	0 - 1	0
readcmd	Read instruction of memory device	0 - 16#FF#	16#0B#
dummybyte	Output dummy byte after address	0 - 1	0
dualoutput	Use dual output when reading data from device	0 - 1	0
scaler	Clock divisor used when generating device clock is 2^{scaler}	1 - 512	1
altscaler	Clock divisor used when generating alternate device clock is $2^{altscaler}$	1 - 512	1
pwrupcnt	Number of clock cycles to wait before issuing first command to memory device	N/A	0

67.7 Signal descriptions

Table 702 shows the interface signals of the core (VHDL ports).

*Table 702.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTN	N/A	Input	Reset	Low
CLK	N/A	Input	Clock	-
AHBSI	*	Input	AHB slave input signals	-
AHBSO	*	Output	AHB slave output signals	-
SPII	MISO	Input	Master-input slave-output data line SD card connection: DAT0	-
	MOSI	Input	Master-output slave-input data line SD card connection: None	-
	CD	Input	Card detection. Used in SD card mode to detect if a card is present. Must be pulled high if this functionality is not used. SD card connection: CD/DAT3	High
SPIO	MOSI	Output	Master-output slave-input data line SD card connection: CMD	-
	MOSIOEN	Output	Master-output slave-input output enable	-
	SCK	Output	SPI clock SD card connection: CLK	-
	CSN	Output	Chip select SD card connection: CD/DAT3	Low
	CDCSNOEN	Output	Chip select output enable. If the core is configured to work with an SD card this signal should be connected to the I/O pad that determines if CSN should drive the CD/DAT3 line. During initialization the CSN signal does not drive CD/DAT3 and the core monitors the CD input to see if a card is present. When a card has been detected CD/DAT3 is used as the chip select signal via CSN. For other SPI memory devices this signal can be left unconnected and the CSN signal can be connected to an output pad.	-
	ERRORN	Output	Error signal. Negated version of Error bit in the core’s Status register.	Low
	READY	Output	When this signal is low the core is busy performing an operation.	High
	INITIALIZED	Output	This bit goes high when the SPI memory device has been initialized and can accept read accesses. This signal has the same value as the Initialized (INIT) bit in the core’s Status register.	High

* see GRLIB IP Library User’s Manual

67.8 Library dependencies

Table 703 shows the libraries used when instantiating the core (VHDL libraries).

*Table 703.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	MEMCTRL	Component, signals	Component and signal definitions
GRLIB	AMBA	Signals	AMBA signal definitions

67.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib, techmap;
use grlib.amba.all;
use techmap.gencomp.all;

library gaisler;
use gaisler.memctrl.all;

entity spimctrl_ex is
  port (
    clk       : in  std_ulogic;
    rstn      : in  std_ulogic;
    -- SPIMCTRL signals
    -- For SD Card
    sd_dat    : in  std_ulogic;
    sd_cmd    : out std_ulogic;
    sd_sck    : out std_ulogic;
    sd_dat3   : inout std_ulogic;
    -- For SPI Flash
    spi_c     : out std_ulogic;
    spi_d     : out std_ulogic;
    spi_q     : in  std_ulogic;
    spi_sn    : out std_ulogic
    );
end;

architecture rtl of spimctrl_ex is
  -- AMBA signals
  signal ahbsi  : ahb_slv_in_type;
  signal ahbso  : ahb_slv_out_vector := (others => ahbs_none);
  ...
  -- SPIMCTRL signals
  signal spmi0, spmi1 : spimctrl_in_type;
  signal spmo0, spmo1 : spimctrl_out_type;
begin

  -- AMBA Components are instantiated here
  ...

  -- Two cores are instantiated below. One configured for use with an SD card and one
  -- for use with a generic SPI memory device. Usage of the errorn, ready and
  -- initialized signals is not shown.

  -- SPMCTRL core, configured for use with SD card
  spimctrl0 : spimctrl
      generic map (hindex => 3, hirq => 3, faddr => 16#a00#, fmask  => 16#ff0#,
                   ioaddr => 16#100#, iomask => 16#fff#, spliten => CFG_SPLIT,
                   sdcard => 1, scaler => 1, altscaler => 7)
      port map (rstn, clk, ahbsi, ahbso(3), spmi0, spmo0);

    sd_miso_pad : inpad generic map (tech => padtech)
      port map (sd_dat, spmi0.miso);
    sd_mosi_pad : outpad generic map (tech => padtech)
      port map (sd_cmd, spmo0.mosi);
    sd_sck_pad  : outpad generic map (tech => padtech)
      port map (sd_clk, spmo0.sck);
    sd_slvsel0_pad : iopad generic map (tech => padtech)
      port map (sd_dat3, spmo0.csn, spmo0.cdcsnoen, spmi0.cd);
    -- Alternative use of cd/dat3 if connection detect is not wanted or available:
    -- sd_slvsel0_pad : outpad generic map (tech => padtech)
    --   port map (sd_dat3, spmo0.csn);
    --  spmi0.cd <= ‘1’; -- Must be set if cd/dat3 is not bi-directional

  -- SPMCTRL core, configured for use with generic SPI Flash memory with read
  -- command 0x0B and a dummy byte following the address.
  spimctrl1 : spimctrl
      generic map (hindex => 4, hirq => 4, faddr => 16#b00#, fmask  => 16#fff#,
                   ioaddr => 16#200#, iomask => 16#fff#, spliten => CFG_SPLIT,
                   sdcard => 0, readcmd => 16#0B#, dummybyte => 1, dualoutput => 0,
                   scaler => 1, altscaler => 1)
      port map (rstn, clk, ahbsi, ahbso(4), spmi1, spmo1);

    spi_miso_pad : inpad generic map (tech => padtech)
      port map (spi_q, spmi1.miso);
    spi_mosi_pad : outpad generic map (tech => padtech)
      port map (spi_d, spmo1.mosi);
    spi_sck_pad  : outpad generic map (tech => padtech)
      port map (spi_c, spmo1.sck);
    spi_slvsel0_pad : outpad generic map (tech => padtech)
      port map (spi_sn, spmo1.csn);
end;
```

68 SRCTRL- 8/32-bit PROM/SRAM Controller

68.1 Overview

SRCTRL is an 8/32-bit PROM/SRAM/IO controller that interfaces external asynchronous SRAM, PROM and I/O to the AMBA AHB bus. The controller can handle 32-bit wide SRAM and I/O, and either 8- or 32-bit PROM.

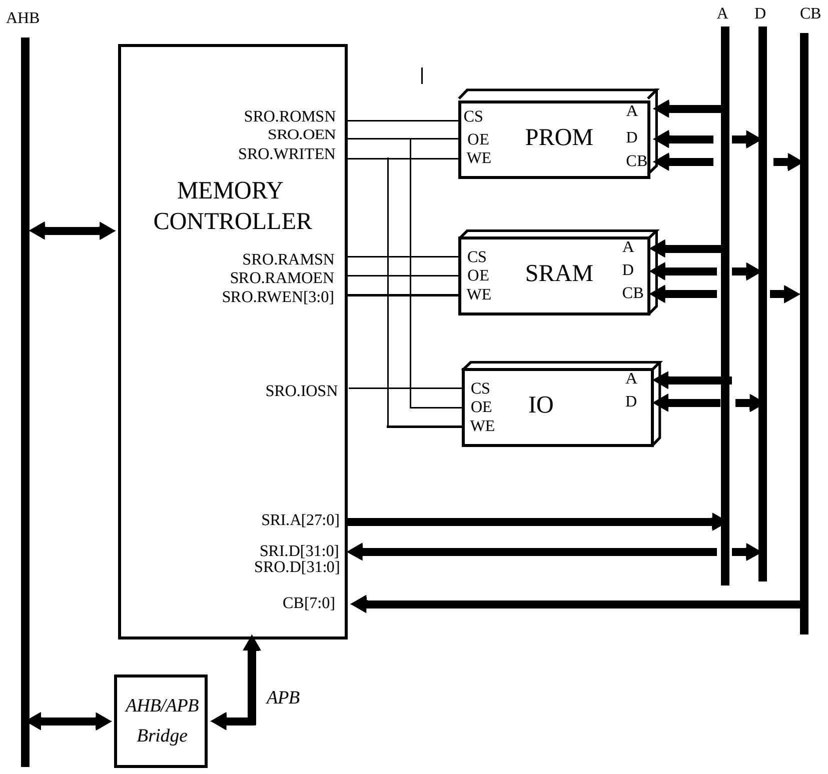


Figure 239. 8/32-bit PROM/SRAM/IO controller

The controller is configured through VHDL-generics to decode three address ranges: PROM, SRAM and I/O area. By default PROM area is mapped into address range 0x0 - 0x00FFFFFF, the SRAM area is mapped into address range 0x40000000 - 0x40FFFFFF, and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is decoded for the I/O area, while SRAM and PROM can have up to four and two select signals respectively. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WREN). If the SRAM uses a common write enable signal the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses. Number of waitstates is separately configurable for the three address ranges.

A single write-enable signal is generated for the PROM area (WRITEN), while four byte-write enable signals (RWEN[3:0]) are provided for the SRAM area. If the external SRAM uses common write enable signal, the controller can be configured to perform read-modify-write cycles for byte and half-word write accesses.

Number of waitstates is configurable through VHDL generics for both PROM and SRAM areas.

A signal (BDRIVE) is provided for enabling the bidirectional pads to which the data signals are connected. The oepol generic is used for selecting the polarity of these enable signals. If output delay is an issue, a vectored output enable signal (VBDRIVE) can be used instead. In this case, each pad has its own enable signal driven by a separate register. A directive is placed on these registers so that they will not be removed during synthesis (if the output they drive is used in the design).

68.2 8-bit PROM access

The SRCTRL controller can be configured to access a 8-bit wide PROM. The data bus of external PROM should be connected to the upper byte of the 32-bit data bus, i.e. D[31:24]. The 8-bit mode is enabled with the prom8en VHDL generic. When enabled, read accesses to the PROM area will be done in four-byte bursts. The whole 32-bit word is then presented on the AHB data bus. Writes should be done one byte at a time and the byte should always be driven on bit 31-24 on the AHB data bus independent of the byte address.

It is possible to dynamically switch between 8- and 32-bit PROM mode using the BWIDTH[1:0] input signal. When BWIDTH is “00” then 8-bit mode is selected. If BWIDTH is “10” then 32-bit mode is selected. Other BWIDTH values are reserved for future use.

SRAM access is not affected by the 8-bit PROM mode.

68.3 PROM/SRAM waveform

Read accesses to 32-bit PROM and SRAM has the same timing, see figure below.

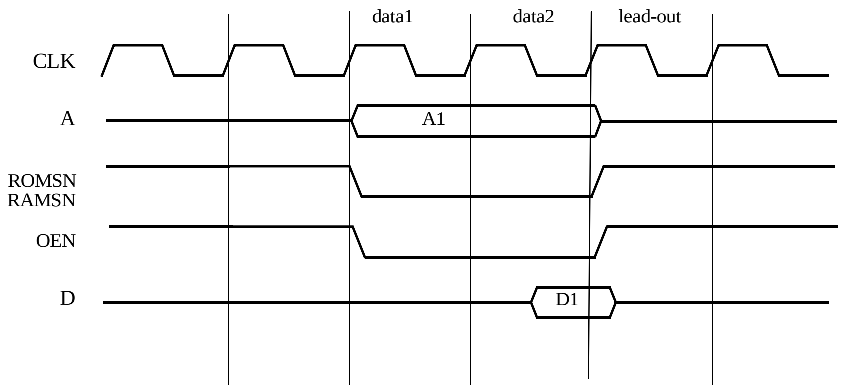


Figure 240. 32-bit PROM/SRAM/IO read cycle

The write access for 32-bit PROM and SRAM can be seen below.

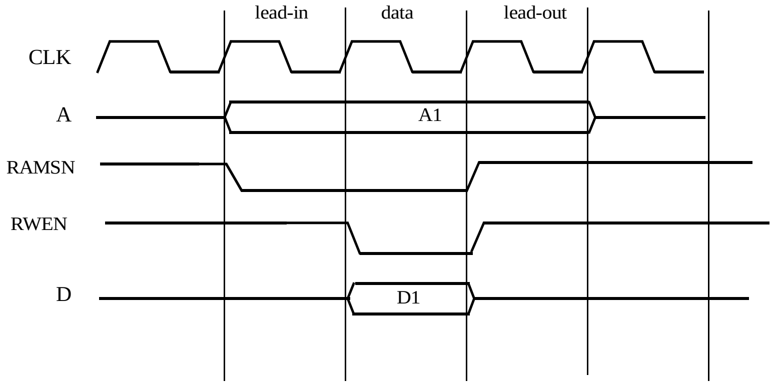


Figure 241. 32-bit PROM/SRAM/IO write cycle

If waitstates are configured through the VHDL generics, one extra data cycle will be inserted for each waitstate in both read and write cycles.

68.4 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 and burst from DMA masters. 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.

68.5 Registers

The core does not implement any user programmable registers.

All configuration is done through the VHDL generics.

68.6 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x008. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

68.7 Configuration options

Table 705 shows the configuration options of the core (VHDL generics).

*Table 704.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining SRAM address space. Default SRAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining SRAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default IO area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 -16#FFF#	16#FF0#
ramws	Number of waitstates during access to SRAM area	0 - 15	0
romws	Number of waitstates during access to PROM area	0 - 15	2
iows	Number of waitstates during access to IO area	0 - 15	2
rmw	Enable read-modify-write cycles.	0 - 1	0
prom8en	Enable 8 - bit PROM accesses	0 - 1	0
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
srbanks	Set the number of SRAM banks	1 - 5	1
banksz	Set the size of bank 1 - 4. 0 = 8 Kbyte, 1 = 16 Kbyte, ... , 13 = 64Mbyte.	0 - 13	13
romasel	address bit used for PROM chip select.	0 - 27	19

68.8 Signal description

Table 704 shows the interface signals of the core (VHDL ports).

*Table 705.*Signal descriptions

Signal name	Field	Type	Function	Polarity
CLK	N/A	Input	Clock	-
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Not used	-
	BEXCN	Input	Not used	-
	WRN[3:0]	Input	Not used	-

	BWIDTH[1:0]	Input	BWIDTH=”00” => 8-bit PROM mode
			BWIDTH=”10” => 32-bit PROM mode
				-

	SD[31:0]	Input	Not used	-
SRO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	RAMSN[4:0]	Output	SRAM chip-select	Low
	RAMOEN[4:0]	Output	SRAM output enable	Low
	IOSN	Output	Not used. Driven to ‘1’ (inactive)	Low
	ROMSN[1:0]	Output	PROM chip-select	Low
	RAMN	Output	Common SRAM chip-select. Asserted when one of the RAMSN[4:0] signals is asserted.	Low
	ROMN	Output	Common PROM chip-select. Asserted when one of the ROMSN[1:0] signals is asserted.	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SRAM write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Byte enable: MBEN[0] corresponds to DATA[31:24], MBEN[1] corresponds to DATA[23:16], MBEN[2] corresponds to DATA[15:8], MBEN[3] corresponds to DATA[7:0].	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0].	Low/High^2
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High^2
	READ	Output	Read strobe	High
	SA[14:0]	Output	Not used	High

*Table 705.*Signal descriptions

Signal name	Field	Type	Function	Polarity
AHBSI	1)	Input	AHB slave input signals	-
AHBSO	1)	Output	AHB slave output signals	-
SDO	SDCASN	Output	Not used. All signals are driven to inactive state.	Low

1) See GRLIB IP Library User's Manual

2) Polarity is selected with the oepol generic

68.9 Library dependencies

Table 706 shows libraries used when instantiating the core (VHDL libraries).

*Table 706.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

68.10 Component declaration

The core has the following component declaration.

```
component srctrl
  generic (
    hindex  : integer := 0;
    romaddr : integer := 0;
    rommask : integer := 16#ff0#;
    ramaddr : integer := 16#400#;
    rammask : integer := 16#ff0#;;
    ioaddr  : integer := 16#200#;
    iomask  : integer := 16#ff0#;
    ramws   : integer := 0;
    romws   : integer := 2;
    iows    : integer := 2;
    rmw     : integer := 0;-- read-modify-write enable
    prom8en : integer := 0;
    oepol   : integer := 0;
    srbanks : integer range 1 to 5 := 1;
    banksz  : integer range 0 to 13:= 13;
    romasel : integer range 0 to 27:= 19
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    sri     : in  memory_in_type;
    sro     : out memory_out_type;
    sdo     : out sdctrl_out_type
  );
end component;
```

68.11 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it including the memory controller. The external memory bus is defined on the example designs port map and connected to the memory controller. System clock and reset are generated by GR Clock Generator and Reset Generator.

Memory controller decodes default memory areas: PROM area is 0x0 - 0xFFFFFF and SRAM area is 0x40000000 - 0x40FFFFF. The 8-bit PROM mode is disabled. Two SRAM banks of size 64 Mbyte are used and the fifth chip select is disabled.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.pads.all;   -- used for I/O pads
use gaisler.misc.all;
library esa;
use esa.memoryctrl.all;

entity srctrl_ex is
  port (
    clk : in std_ulogic;
    resetn : in std_ulogic;
    pllref : in  std_ulogic;

    -- memory bus
    address  : out   std_logic_vector(27 downto 0); -- memory bus
    data     : inout std_logic_vector(31 downto 0);
    ramsn    : out   std_logic_vector(4 downto 0);
    ramoen   : out   std_logic_vector(4 downto 0);
    rwen     : inout std_logic_vector(3 downto 0);
    romsn    : out   std_logic_vector(1 downto 0);
    iosn     : out   std_logic;
    oen      : out   std_logic;
    read     : out   std_logic;
    writen   : inout std_logic;
    brdyn    : in    std_logic;
    bexcn    : in    std_logic;
    modesel  : in    std_logic; --PROM width select
-- sdram i/f
    sdcke    : out std_logic_vector ( 1 downto 0);  -- clk en
    sdcsn    : out std_logic_vector ( 1 downto 0);  -- chip sel
    sdwen    : out std_logic;                       -- write en
    sdrasn   : out std_logic;                       -- row addr stb
    sdcasn   : out std_logic;                       -- col addr stb
    sddqm    : out std_logic_vector (7 downto 0);  -- data i/o mask
    sdclk    : out std_logic;                       -- sdram clk output
    sa       : out std_logic_vector(14 downto 0); -- optional sdram address
    sd       : inout std_logic_vector(63 downto 0) -- optional sdram data
       );
end;

architecture rtl of srctrl_ex is

  -- AMBA bus (AHB and APB)
  signal apbi  : apb_slv_in_type;
  signal apbo  : apb_slv_out_vector := (others => apb_none);
  signal ahbsi : ahb_slv_in_type;
  signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

  -- signals used to connect memory controller and memory bus
  signal memi : memory_in_type;
  signal memo : memory_out_type;

  signal sdo : sdctrl_out_type;

  signal wprot : wprot_out_type;  -- dummy signal, not used
  signal clkm, rstn : std_ulogic; -- system clock and reset

-- signals used by clock and reset generators
```

```
  signal cgi : clkgen_in_type;
  signal cgo : clkgen_out_type;

  signal gnd : std_ulogic;

begin

  -- AMBA Components are defined here ...


  -- Clock and reset generators
  clkgen0 : clkgen generic map (clk_mul => 2, clk_div => 2, sdramen => 1,
                               tech => virtex2, sdinvclk => 0)
  port map (clk, gnd, clkm, open, open, sdclk, open, cgi, cgo);

  cgi.pllctrl <= "00"; cgi.pllrst <= resetn; cgi.pllref <= pllref;

  rst0 : rstgen
  port map (resetn, clkm, cgo.clklock, rstn);


  -- Memory controller
srctrl0 : srctrl generic map (rmw => 1, prom8en => 0, srbanks => 2,
  banksz => 13, ramsel5 => 0)
    port map (rstn, clkm, ahbsi, ahbso(0), memi, memo, sdo);


  -- I/O pads driving data memory bus data signals
  datapads : for i in 0 to 3 generate
      data_pad : iopadv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(i),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- Alternative I/O pad instantiation with vectored enable instead
  datapads : for i in 0 to 3 generate
      data_pad : iopadvv generic map (width => 8)
      port map (pad => data(31-i*8 downto 24-i*8),
                o => memi.data(31-i*8 downto 24-i*8),
                en => memo.bdrive(31-i*8 downto 24-i*8),
                i => memo.data(31-i*8 downto 24-i*8));
  end generate;

  -- connect memory controller outputs to entity output signals
  address <= memo.address; ramsn <= memo.ramsn; romsn <= memo.romsn;
  oen <= memo.oen; rwen <= memo.wrn; ramoen <= memo.ramoen;
  writen <= memo.writen; read <= memo.read; iosn <= memo.iosn;
  sdcke <= sdo.sdcke; sdwen <= sdo.sdwen; sdcsn <= sdo.sdcsn;
  sdrasn <= sdo.rasn; sdcasn <= sdo.casn; sddqm <= sdo.dqm;

end;
```

69 SSRCTRL- 32-bit SSRAM/PROM Controller

69.1 Overview

The memory controller (SSRCTRL) is an 32-bit SSRAM/PROM/IO controller that interfaces external Synchronous pipelined SRAM, PROM, and I/O to the AMBA AHB bus. The controller acts as a slave on the AHB bus and has a configuration register accessible through an APB slave interface. Figure 242 illustrates the connection between the different devices.

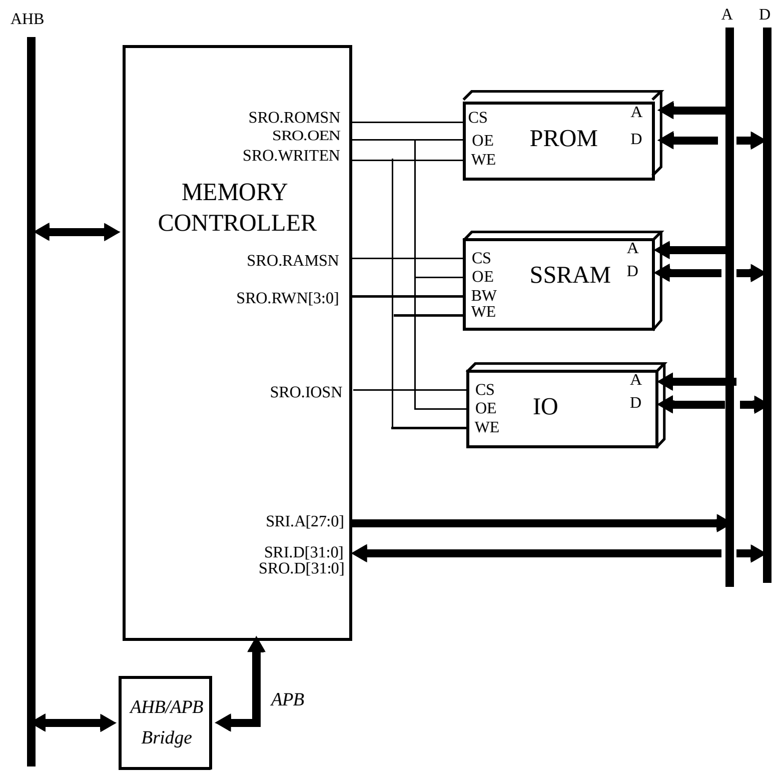


Figure 242. 32-bit SSRAM/PROM/IO controller

The controller is configured by VHDL-generics to decode three address ranges: PROM, SSRAM and I/O area. By default PROM area is mapped into address range 0x0 - 0x00FFFFFF; the SSRAM area is mapped into address range 0x40000000 - 0x40FFFFFF; and the I/O area is mapped to 0x20000000 - 0x20FFFFFF.

One chip select is generated for each of the address areas. The controller generates both a common write-enable signal (WRITEN) as well as four byte-write enable signals (WRN). The byte-write enable signal enables byte and half-word write access to the SSRAM.

A signal (BDRIVE) is provided for enabling the bidirectional pads to which the data signals are connected. The oepol generic is used to select the polarity of these enable signals. If output delay is an issue, a vectored output enable signal (VBDRIVE) can be used instead. In this case, each pad has its own enable signal driven by a separate register. A directive is placed on these registers so that they will not be removed during synthesis (in case the output they drive is used in the design).

The SSRCTRL conteoller can optionally support 16-bit PROM/IO devices. This is enabled through the BUS16 generic. A 32-bit access to the PROM or IO area will be translated into two 16-bit accesses with incrementing address.

69.2 SSRAM/PROM waveform

Because the SSRAM (Synchronous pipelined SRAM) has a pipelined structure, the data output has a latency of three clock cycles. The pipelined structure enables a new memory operation to be issued each clock cycle. Figure 242 and figure 243 show timing diagrams for the SSRAM read and write accesses.

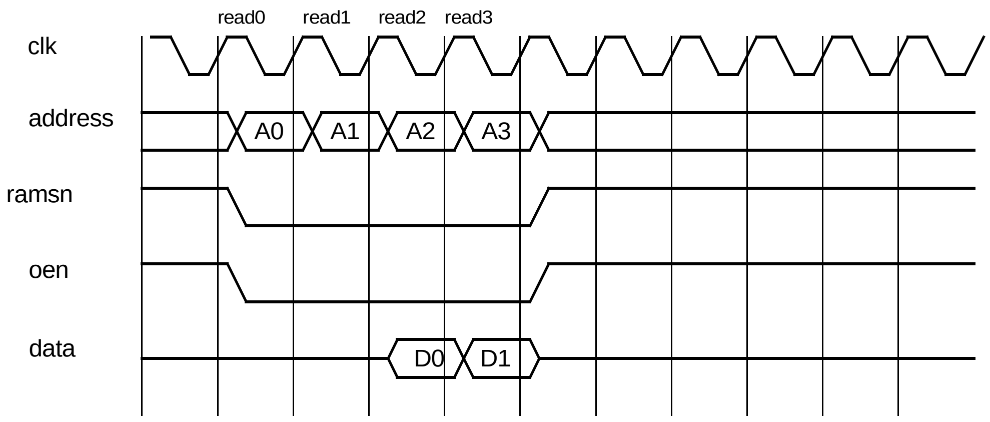


Figure 243. 32-bit SSRAM read cycle

As shown in the figure above, the controller always perform a burst read access to the memory. This eliminates all data output latency except for the first word when a burst read operation is executed.

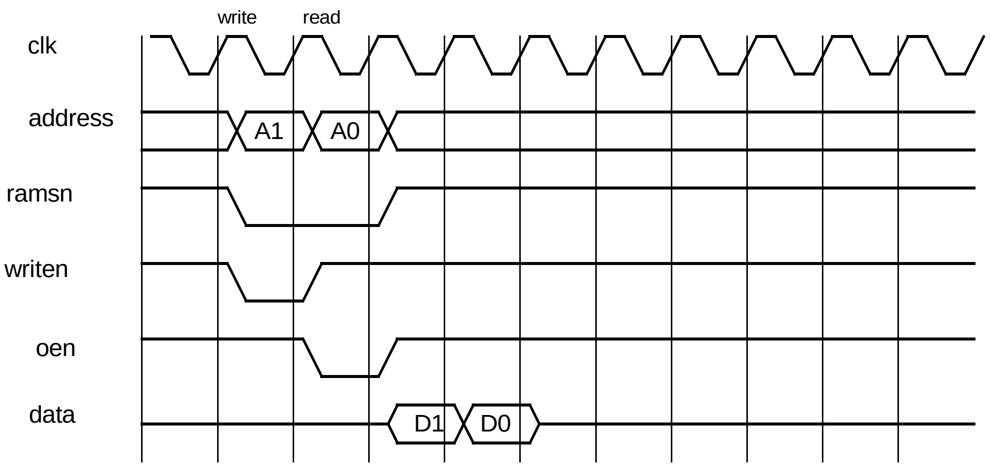


Figure 244. 32-bit SSRAM write cycle

A write operation takes three clock cycles. On the rising edge of the first clock cycle, the address and control signals are latched into the memory. On the next rising edge, the memory puts the data bus in high-impedance mode. On the third rising edge the data on the bus is latched into the memory and the write is complete. The controller can start a new memory (read or write) operation in the second clock cycle. In figure 244 this is illustrated by a read operation following the write operation.

Due to the memory automatically putting the data bus in high-impedance mode when a write operation is performed, the output-enable signal (OEN) is held active low during all SSRAM accesses (including write operations).

69.2.1 PROM and IO access

For the PROM and I/O operations, a number of waitstates can be inserted to increase the read and write cycle. The number of waitstates can be configured separately for the I/O and PROM address ranges, through a programmable register mapped into the APB address space. After a reset the wait-states for PROM area is set to its maximum (15). Figure 245 and figure 246 show timing diagrams for the PROM read and write accesses.

Read accesses to 32-bit PROM and I/O has the same timing, see figure 245

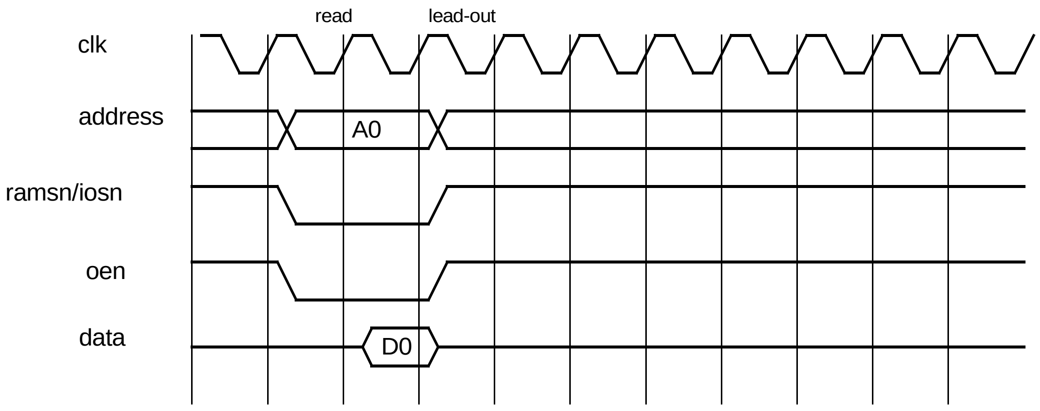


Figure 245. 32-bit PROM/IO read cycle

The write access for 32-bit PROM and I/O can be seen in figure 246

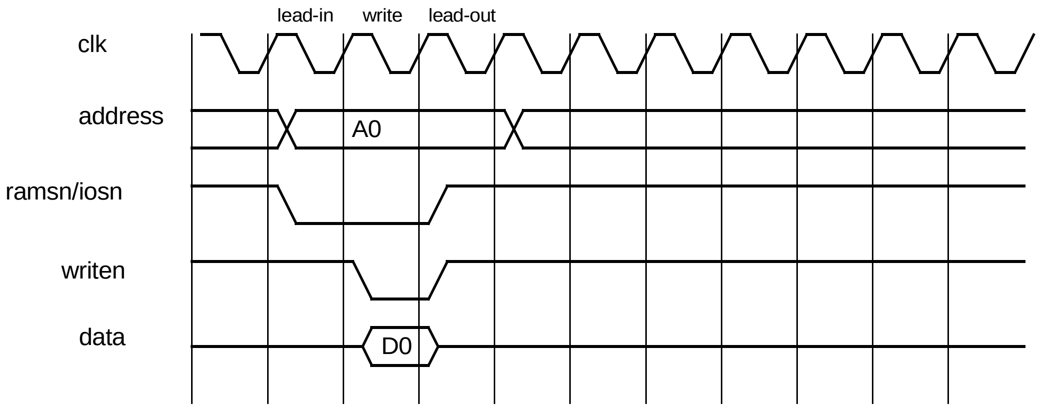


Figure 246. 32-bit PROM/IO write cycle

The SSRCTRL conteoller can optionally support 16-bit PROM/IO devices. This is enabled through the BUS16 generic. A 32-bit access to the PROM or IO area will be translated into two 16-bit accesses with incrementing address. A 16-bit access will result in one bus access only. 8-bit accesses are not allowed.

16-bit PROM/IO operation is enabled by writing “01” to the romwidth field in SSRAM control register. At reset, the romwidth field is set by the MEMI.BWIDTH input signal.

Read accesses to 16-bit PROM and I/O has the same timing, see figure 247

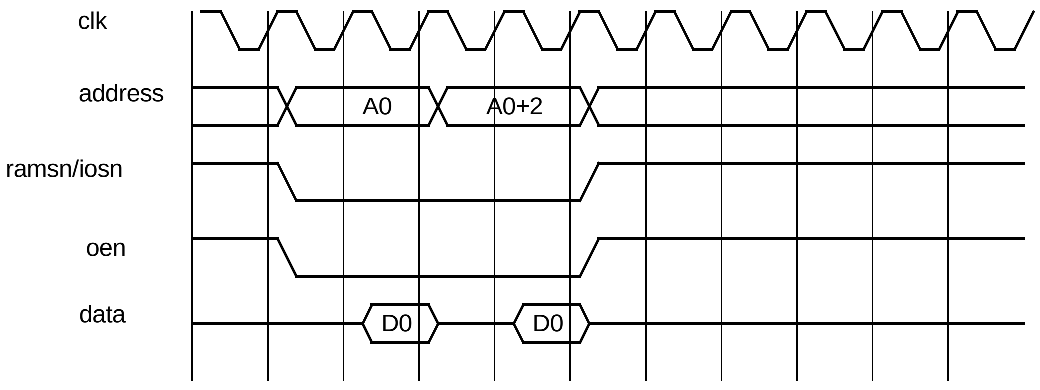


Figure 247. 32-bit PROM/IO read cycle in 16-bit mode

The write access for 32-bit PROM and I/O can be seen in figure 248

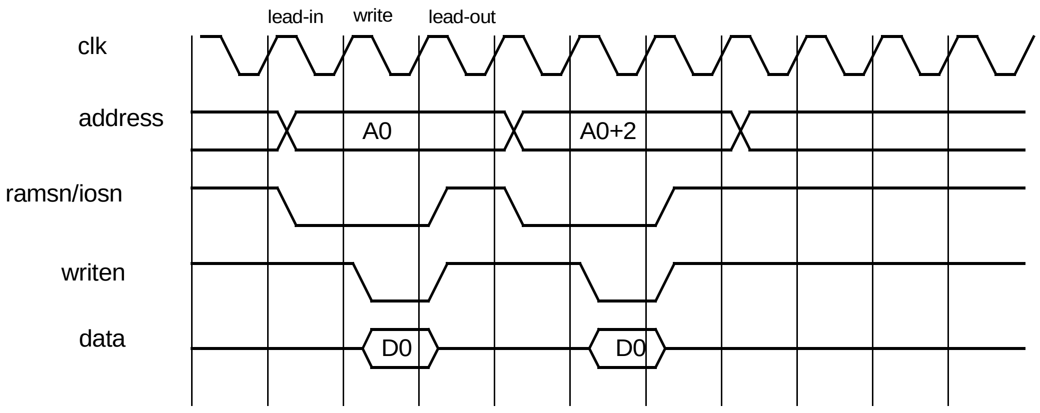


Figure 248. 32-bit PROM/IO write cycle in 16-bit mode

69.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 707.*SSRAM controller registers

APB address offset	**Register**
0x00	Memory configuration register

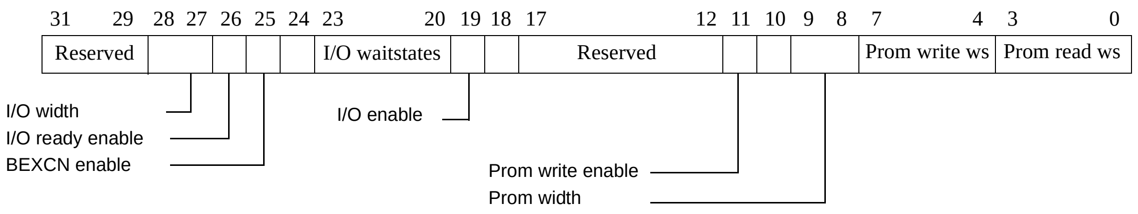


Figure 249. Memory configuration register

[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 with of the prom area (“01”=16, “10”=32).
[10]:	Reserved
[11]:	Prom write enable. If set, enables write cycles to the prom area. NOT USED.
[17:12]:	Reserved
[19]:	I/O enable. If set, the access to the memory bus I/O area are enabled. NOT USED.
[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. NOT USED.
[26]:	Bus ready (BRDYN) enable. NOT USED.
[28:27]:	I/O bus width. Defines the data with of the I/O area (“01”=16, “10”=32).

During power-up (reset), the PROM waitstates fields are set to 15 (maximum) and the PROM bus width is set to the value of MEMI.BWIDTH. All other fields are initialized to zero.

69.4 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x00A. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

69.5 Configuration options

Table 708 shows the configuration options of the core (VHDL generics).

*Table 708.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB slave index	1 - NAHBSLV-1	0
pindex	APB slave index	0 - NAPBSLV-1	0
romaddr	ADDR field of the AHB BAR0 defining PROM address space. Default PROM area is 0x0 - 0xFFFFFF.	0 - 16#FFF#	16#000#
rommask	MASK field of the AHB BAR0 defining PROM address space.	0 - 16#FFF#	16#FF0#
ramaddr	ADDR field of the AHB BAR1 defining RAM address space. Default RAM area is 0x40000000-0x40FFFFFF.	0 - 16#FFF#	16#400#
rammask	MASK field of the AHB BAR1 defining RAM address space.	0 -16#FFF#	16#FF0#
ioaddr	ADDR field of the AHB BAR2 defining IO address space. Default IO area is 0x20000000-0x20FFFFFF.	0 - 16#FFF#	16#200#
iomask	MASK field of the AHB BAR2 defining IO address space.	0 -16#FFF#	16#FF0#
paddr	ADDR field of the APB BAR configuration registers address space.	0 - 16#FFF#	0
pmask	MASK field of the APB BAR configuration registers address space.	0 - 16#FFF#	16#FFF#
oepol	Polarity of bdrive and vbdrive signals. 0=active low, 1=active high	0 - 1	0
bus16	Enable support for 16-bit PROM/IO accesses	0 - 1	0

69.6 Signal descriptions

Table 709 shows the interface signals of the core (VHDL ports).

*Table 709.*Signal descriptions

Signal name	Field	Type	Function	Polarity
CLK	N/A	Input	Clock	-

*Table 709.*Signal descriptions

Signal name	Field	Type	Function	Polarity
RST	N/A	Input	Reset	Low
SRI	DATA[31:0]	Input	Memory data	High
	BRDYN	Input	Not used	-
	BEXCN	Input	Not used	-
	WRN[3:0]	Input	Not used	-
	BWIDTH[1:0]	Input	PROM bus width at reset	-
	SD[63:0]	Input	Not used	-
	CB[7:0]	Input	Not used	-
	SCB[7:0]	Input	Not used	-
	EDAC	Input	Not used	-
SRO	ADDRESS[27:0]	Output	Memory address	High
	DATA[31:0]	Output	Memory data	High
	SDDATA[63:0]	Output	Not used	-
	RAMSN[7:0]	Output	SSRAM chip-select, only bit 0 is used	Low
	RAMOEN[7:0]	Output	Same as OEN	Low
	IOSN	Output	I/O chip-select	Low
	ROMSN[7:0]	Output	PROM chip-select, only bit 0 is used	Low
	OEN	Output	Output enable	Low
	WRITEN	Output	Write strobe	Low
	WRN[3:0]	Output	SSRAM byte write enable: WRN[0] corresponds to DATA[31:24], WRN[1] corresponds to DATA[23:16], WRN[2] corresponds to DATA[15:8], WRN[3] corresponds to DATA[7:0].	Low
	MBEN[3:0]	Output	Not used	Low
	BDRIVE[3:0]	Output	Drive byte lanes on external memory bus. Controls I/O-pads connected to external memory bus: BDRIVE[0] corresponds to DATA[31:24], BDRIVE[1] corresponds to DATA[23:16], BDRIVE[2] corresponds to DATA[15:8], BDRIVE[3] corresponds to DATA[7:0]. Any BDRIVE[ ] signal can be used for CB[ ].	Low/High[2]
	VBDRIVE[31:0]	Output	Identical to BDRIVE but has one signal for each data bit. Every index is driven by its own register. This can be used to reduce the output delay.	Low/High[2]
	SVBDRIVE	Output	Not used	-
	READ	Output	Not used	-
	SA[14:0]	Output	Not used	-
	CB[7:0]	Output	Not used	-
	SCB[7:0]	Output	Not used	-
	VCDRIVE[7:0]	Output	Not used	-
	SVCDRIVE[7:0]	Output	Not used	-
	CE	Output	Not used	-
AHBSI	1)	Input	AHB slave input signals	-

*Table 709.*Signal descriptions

Signal name	Field	Type	Function	Polarity
AHBSO	1)	Output	AHB slave output signals	-
APBI	1)	Input	APB slave input signals	-
APBO	1)	Output	APB slave output signals	-

1) See GRLIB IP Library User’s Manual

2) Polarity is selected with the oepol generic

69.7 Library dependencies

Table 710 shows libraries used when instantiating the core (VHDL libraries).

*Table 710.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AHB signal definitions
GAISLER	MEMCTRL	Signals, component	Memory bus signals definitions, component declaration

69.8 Component declaration

The core has the following component declaration.

```
component ssrctrl
  generic (
    hindex  : integer := 0;
    pindex  : integer := 0;
    romaddr : integer := 0;
    rommask : integer := 16#ff0#;
    ramaddr : integer := 16#400#;
    rammask : integer := 16#ff0#;
    ioaddr  : integer := 16#200#;
    iomask  : integer := 16#ff0#;
    paddr   : integer := 0;
    pmask   : integer := 16#fff#;
     oepol   : integer := 0;
     bus16 : integer := 0
  );
  port (
    rst     : in  std_ulogic;
    clk     : in  std_ulogic;
    ahbsi   : in  ahb_slv_in_type;
    ahbso   : out ahb_slv_out_type;
    apbi    : in  apb_slv_in_type;
    apbo    : out apb_slv_out_type;
    sri     : in  memory_in_type;
    sro     : out memory_out_type

  );
end component;
```

69.9 Instantiation

This example shows how the core can be instantiated.

The example design contains an AMBA bus with a number of AHB components connected to it, including the memory controller. The external memory bus is defined in the example designs port map and connected to the memory controller. System clock and reset are generated by the Clkgen_ml401 Clock Generator and GR Reset Generator.

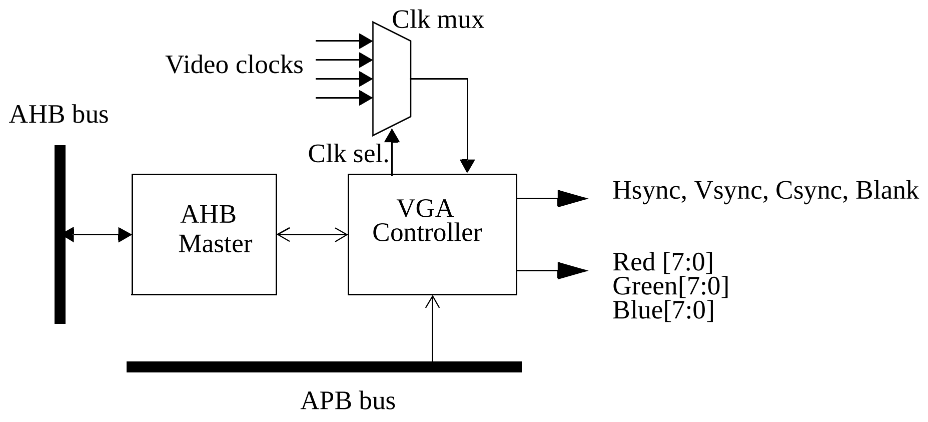
The memory controller decodes default memory areas: PROM area is 0x0 - 0x00FFFFFF, I/O-area is 0x20000000-0x20FFFFFF and RAM area is 0x40000000 - 0x40FFFFFF.

```
library ieee;
use ieee.std_logic_1164.all;
library grlib, techmap;
use grlib.amba.all;
use grlib.stdlib.all;
use techmap.gencomp.all;
library gaisler;
use gaisler.memctrl.all;
use gaisler.misc.all;

entity ssrctrl_ex is
 port (
    sys_rst_in: in  std_ulogic;
    sys_clk: in  std_ulogic; -- 100 MHz main clock
    sram_flash_addr : out std_logic_vector(22 downto 0);
    sram_flash_data : inout std_logic_vector(31 downto 0);
    sram_cen  : out std_logic;
    sram_bw   : out std_logic_vector (0 to 3);
    sram_flash_oe_n : out std_ulogic;
    sram_flash_we_n : out std_ulogic;
    flash_ce  : out std_logic;
    sram_clk  : out std_ulogic;
    sram_clk_fb: in  std_ulogic;
    sram_mode : out std_ulogic;
    sram_adv_ld_n : out std_ulogic;
    sram_zz : out std_ulogic;
    iosn    : out std_ulogic;
);
end;

architecture rtl of ssrctrl_ex is

-- Clock generator component
component clkgen_ml401
  generic (
    clk_mul  : integer := 1;
    clk_div  : integer := 1;
    freq     : integer := 100000);-- clock frequency in KHz
  port (
    clkin   : in  std_logic;
    clk     : out std_logic;-- main clock
    ddrclk  : out std_logic;-- DDR clock
    ddrclkfb: in  std_logic;-- DDR clock feedback
    ddrclk90  : out std_logic;-- DDR 90 clock
    ddrclk180 : out std_logic;-- 180 clock
    ddrclk270 : out std_logic;-- DDR clock
    ssrclk  : out std_logic;-- SSRAM clock
    ssrclkfb: in  std_logic;-- SSRAM clock feedback
    cgi     : in clkgen_in_type;
    cgo     : out clkgen_out_type);
end component;


-- signals used to connect memory controller and memory bus
signal memi  : memory_in_type;
signal memo  : memory_out_type;

-- AMBA bus (AHB and APB)
signal apbi  : apb_slv_in_type;
signal apbo  : apb_slv_out_vector := (others => apb_none);
signal ahbsi : ahb_slv_in_type;
signal ahbso : ahb_slv_out_vector := (others => ahbs_none);
signal ahbmi : ahb_mst_in_type;
signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

-- Signals used by clock and reset generators
signal clkm, rstn, rstraw, srclkl : std_ulogic;
signal cgi   : clkgen_in_type;
signal cgo   : clkgen_out_type;
signal ddrclkfb, ssrclkfb, ddr_clkl, ddr_clknl : std_ulogic;


begin

  clkgen0 : clkgen_ml401  -- clock generator
  port map (sys_clk, clkm, ddr_clkl, ddrclkfb, open, ddr_clknl, open, sram_clk,
      sram_clk_fb, cgi, cgo);

  rst0 : rstgen-- reset generator
  port map (sys_rst_in, clkm, cgo.clklock, rstn, rstraw);

  -- AMBA Components are defined here ...

  -- Memory controller
  mctrl0 : ssrctrl generic map (hindex => 0, pindex => 0)
  port map (rstn, clkm, ahbsi, ahbso(0), apbi, apbo(0), memi, memo);

  -- connect memory controller outputs to entity output signals
  sram_adv_ld_n <=  ’0’; sram_mode <=  ’0’; sram_zz <= ’0’;
  sram_flash_addr <= memo.address(24 downto 2); sram_cen <= memo.ramsn(0);
  flash_ce <= memo.romsn(0); sram_flash_oe_n <= memo.oen; iosn <= memo.iosn;
  sram_bw <= memo.wrn; sram_flash_we_n <= memo.writen;

  -- I/O pad instantiation with vectored enable instead
  bdr : for i in 0 to 31 generate
      data_pad : iopad generic map (tech => padtech)
      port map (sram_flash_data(i), memo.data(i),
             memo.vbdrive(i), memi.data(i));
  end generate;

end;
```

70 SVGACTRL - VGA Controller Core

70.1 Overview

The core is a pixel based video controller (frame buffer), capable of displaying standard and custom resolutions with variable bit depth and refresh rates. The video controller consists of a synchronization unit, main control unit, FIFO unit and an AHB master as shown in the figure below.



70.2 Operation

The core uses external frame buffer memory located in the AHB address space. A frame on the display is created by fetching the pixel data from memory, and sending it to the screen through an external DAC using three 8-bit color vectors. To hide the AHB bus latency, the pixel data is buffered in a FIFO inside the core. The start address of the frame buffer is specified in the Frame buffer Memory Position register, and can be anywhere in the AHB address space. In addition to the color vectors the, video controller also generates HSYNC, VSYNC, CSYNC and BLANK signals control signals.

The video timing is programmable through the Video Length, Front Porch, Sync Length and Line Length registers. The bit depth selection and enabling of the controller is done through the status register. These values makes it possible to display a wide range of resolutions and refresh rates.

The pixel clock can be either static or a dynamic multiplexed. The frequency of the pixel clock is calculated as "Horizontal Line Length * Vertical Line Length * refresh rate". When using a dynamically multiplexed clock, bits [5:4] in the status register are used to control the clock selector. The dynamic pixel clocks should be defined in the core's VHDL generics to allow software to read out the available pixel clock frequencies.

The core can use bit depths of 8, 16 and 32 bits. When using 32 bits, bits[23:0] are used, when 16 bits a [5,6,5] color scheme is used and when using 8 bits a color lookup table "CLUT" is used. The CLUT has 256 addresses each 24 bits wide and the 8 bit values read from memory are used to index the CLUT to obtain the actual color.

70.3 Registers

The core is programmed through registers mapped into APB address space.

*Table 711.*VGA controller registers

APB address offset	**Register**
0x00	Status register
0x04	Video length register
0x08	Front Porch register
0x0C	Sync Length register
0x10	Line Length register
0x14	Framebuffer Memory Position register
0x18	Dynamic Clock 0 register
0x1C	Dynamic Clock 1 register
0x20	Dynamic Clock 2 register
0x24	Dynamic Clock 3 register
0x28	CLUT Access register

Table 712. VGA controller Status register

31 10	9	8	7 6	5 4	3	2	1	0
RESERVED	VPOL	HPOL	CLKSEL	BDSEL	RUN	VR	RST	EN

31:10	RESERVED
9	V polarity (VPOL)- Sets the polarity for the vertical sync pulse.
8	H polarity (HPOL) - Sets the polarity for the horizontal sync pulse.
7:6	Clock Select (CLKSEL) Clock selector when using dynamic pixelclock
5:4	Bit depth selector (BDSEL) - “01” = 8-bit mode; “10” = 16-bit mode; “10” or “11” = 32-bit mode
3	Running (RUN) - Displays the current mode active high
2	Vertical Refresh (VR) - Displays current refresh state, active high
1	Reset (RST) - Resets the core
0	Enable (EN) - Enables the core

Table 713. VGA controller Video Length register

31 16	15 0
VRES	HRES

31:16	Vertical screen resolution (VRES) - Vertical screen resolution in pixels -1
15:0	Horisontal screen resolution (HRES) - Horizontal screen resolution in pixels -1.

Table 714. VGA controller Front porch register

31 16	15 0
VPORCH	HPORCH

31:16	Vertical front porch (VPORCH) - Vertical front porch in pixels.
15:0	Horisontal front porch (HPORCH) - Horizontal front porch in pixels.

Table 715. VGA controller Sync pulse register

31 … 16	15 … 0
VPLEN	HPLEN

31:16	Vertical sync pulse length (VPLEN) - Vertical sync pulse length in pixels.
15:0	Horisontal sync pulse length (HPLEN) - Horizontal sync pulse length in pixels.

Table 716. VGA controller Line Length register

31 … 16	15 … 0
VLLEN	HLLEN

31:16	Vertical line length (VLLEN) - The length of the total line with front and back porch, sync pulse length and vertical screen resolution.
15:0	Horisontal line length (HLLEN) - The length of the total line with front and back porch, sync pulse length and horizontal screen resolution,

Table 717. VGA controller Framebuffer Memory Position register

31 … 0
FMEM

31:0	Framebuffer memory position (FMEM) - Holds the memory position of the framebuffer, must be aligned on a 1 Kbyte boundary.

Table 718. VGA controller Dynamic clock 0 register

31 … 0
CLK0

31:0	Dynamic pixel clock 0 (CLK0) - Dynamic pixel clock defined in ps.

Table 719. VGA controller Dynamic clock 1 register

31 … 0
CLK1

31:0	Dynamic pixel clock 1 (CLK1) - Dynamic pixel clock defined in ps.

Table 720. VGA controller Dynamic clock 2 register

31 … 0
CLK2

31:0	Dynamic pixel clock 2 (CLK2) - Dynamic pixel clock defined in ps.

Table 721. VGA controller Dynamic clock 3 register

31	0
CLK3	

31:0	Dynamic pixel clock 3 (CLK3) - Dynamic pixel clock defined in ps.

Table 722. VGA controller CLUT Access register

31 24	23 16	15 6	7 0
CREG	RED	GREEN	BLUE

31:24	Color lookup table register (CREG) - Color lookup table register to set.
23:16	Red color data (RED) - Red color data to set in the specified register.
15:8	Green color data (GREEN) - Green color data to set in the specified register.
7:0	Blue color data (BLUE) - Blue color data to set in the specified register.

70.4 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x063. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

70.5 Configuration options

Table 723 shows the configuration options of the core (VHDL generics).

*Table 723.*Configuration options

Generic name	Function	Allowed range	Default
length	Size of the pixel FIFO	3 - 1008	384
part	Pixel FIFO part length	1 - 336	128
memtech	Memory technology	0 - NTECH	0
pindex	APB slave index	0 - NAPBSLV-1	0
paddr	12-bit MSB APB address	0 - 16#FFF#	0
pmask	APB address mask	0 - 16#FFF#	16#FFF#
hindex	AHB master index	0 - NAHBMST-1	0
hirq	Interrupt line	0 - NAHBIRQ-1	0
clk0	Period of dynamic clock 0 in ps	0- 16#FFFFFFFF#	40000
clk1	Period of dynamic clock 1 in ps	0- 16#FFFFFFFF#	20000
clk2	Period of dynamic clock 2 in ps	0- 16#FFFFFFFF#	15385
clk3	Period of dynamic clock 3 in ps	0- 16#FFFFFFFF#	0
burstlen	AHB burst length. The core will burst $2^{burstlen}$ words.	2 - 8	8

70.6 Signal descriptions

Table 724 shows the interface signals of the core (VHDL ports).

*Table 724.*Signal descriptions

Signal name	Field	Type	Function	Active
RST	N/A	Input	Reset	Low
CLK	N/A	Input	System clock	-
VGACLK	N/A	Input	Pixel clock	-
APBI	*	Input	APB slave input signals	-
APBO	*	Output	APB slave output signals	-
VGAO	HSYNC	Output	Horizontal sync	-
	VSYNC	Output	Vertical sync	-
	CSYNC	Output	Composite sync	-
	BLANK	Output	Blanking	-
	VIDEO_OUT_R[7:0]	Output	Video out, red.	-
	VIDEO_OUT_G[7:0]	Output	Video out, green.	-
	VIDEO_OUT_B[7:0]	Output	Video out, blue.	-
AHBI	*	Input	AHB master input signals	-
AHBO	*	Output	AHB master output signals	-
CLK_SEL[1:0]	N/A	Output	2-bit clock selector	-

* see GRLIB IP Library User’s Manual

70.7 Library dependencies

Table 725 shows the libraries used when instantiating the core (VHDL libraries).

*Table 725.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	MISC	Component, signals	Component and signal definitions.

70.8 Instantiation

This example shows how the core can be instantiated.

```
library grlib;
use grlib.amba.all;
library Gaisler;
use gaiser.misc.all;
.
architecture rtl of test is
signal apbi : apb_slv_in_type;
signal apbo : apb_slv_out;
signal vgao : apbvga_out_type;
signal ahbi : ahb_mst_in_type;
signal ahbo : ahb_mst_out_type;
signal clk_sel :std_logic_vector(1 downto 0));
signal clkmvga : std_logic;
begin
.
.
-- VGA Controller
  vga0 : svgactrl
  generic map(memtech => memtech, pindex => 6, paddr => 6, hindex => 6,
    clk0 => 40000, clk1 => 20000, clk2 => 15385, clk3 => 0)
  port map(rstn,clkm,clkmvga, apbi, apbo(6), vgao,ahbmi,ahbmo(6),clk_sel);
end;
```

70.9 Linux-2.6 command line options

A video driver for SVGACTRL is provided Snapgear Linux (-p27 and later) from Gaisler Research. The proper kernel command line options must be used for the driver to detect the core. The table below lists the boot options and the order they should appear in.

71 SYNCRAM - Single-port RAM generator

71.1 Overview

The single port RAM has a common address bus, and separate data-in and data-out buses. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge.

71.2 Configuration options

Table 726 shows the configuration options of the core (VHDL generics).

*Table 726.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-

Table 727 shows the supported technologies for the core.

*Table 727.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
altera	All Altera devices	altsyncram	unlimited	unlimited
ihp15	IHP 0.25	sram2k (512x32)	2 - 9	unlimited
inferred	Behavioral description	Tool dependent	unlimited	unlimited
virtex	Xilinx Virtex, VirtexE, Spartan2	RAMB4_Sn	2 - 12	unlimited
virtex2	Xilinx Virtex2, Spartan3, Virtex4, Spartan3e	RAMB16_Sn	2 - 14	unlimited
axcel	Actel AX, RTAX	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	sp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss1_128x32cm4sw0 hdss1_256x32cm4sw0 hdss1_512x32cm4sw0 hdss1_1024x32cm8sw0	7 - 11	32
memartisan	Artisan ASIC RAM	sp_256x32m32 sp_512x32m32 sp_1kx32m32 sp_2kx32m32 sp_4kx32m32 sp_8kx32m32 sp_16kx32m32	8 - 14	32
memvirage90	Virage 90 nm ASIC RAM	SPRAM_HS_32x30 SPRAM_HS_128x32 SPRAM_HS_256x32 SPRAM_HS_1024x32	2 - 10	128
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram, bram	2 - 15	unlimited

71.3 Signal descriptions

Table 728 shows the interface signals of the core (VHDL ports).

*Table 728.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK	N/A	Input	Clock. All input signals are latched on the rising edge of the clock.	-
ADDRESS	N/A	Input	Address bus. Used for both read and write access.	-
DATAIN	N/A	Input	Data inputs for write data	-
DATAOUT	N/A	Output	Data outputs for read data	-
ENABLE	N/A	Input	Chip select	High
WRITE	N/A	Input	Write enable	High

71.4 Library dependencies

Table 729 shows libraries used when instantiating the core (VHDL libraries).

*Table 729.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

71.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

  component syncram
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8);
  port (
    clk      : in std_ulogic;
    address  : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    enable   : in std_ulogic;
    write    : in std_ulogic);
  end component;
```

71.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;
.

clk      : std_ulogic;
address  : std_logic_vector((abits -1) downto 0);
datain   : std_logic_vector((dbits -1) downto 0);
dataout  : std_logic_vector((dbits -1) downto 0);
enable   : std_ulogic;
write    : std_ulogic);

ram0 : syncram generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( clk, addr, datain, dataout, enable, write);
```

72 SYNCRAM_2P - Two-port RAM generator

72.1 Overview

The two-port RAM generator has a one read port and one write port. Each port has a separate address and data bus. All inputs are registered on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Address width, data width and target technology is parametrizable through generics.

Write-through is supported if the function *syncram_2p_write_through(tech)* returns 1 for the target technology.

72.2 Configuration options

Table 730 shows the configuration options of the core (VHDL generics).

*Table 730.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-
sepclk	If 1, separate clocks (rclk/wclk) are used for the two ports. If 0, rclk is used for both ports.	0 - 1	0

Table 731 shows the supported technologies for the core.

*Table 731.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
Inferred	Behavioural description	Tool dependent	unlimited	unlimited
altera	All Altera devices	altsyncram	umlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2, Spartan3, Virtex4, Spartan3e	RAMB16_Sn	2 - 14	unlimited
axcel	Actel AX, RTAX	RAM64K36	2 - 12	unlimited
proasic	Actel Proasic	RAM256x9SST	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9, ram512x18	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	rf2_256x32m4 rf2_512x32m4	8 - 9	32
eclipse	Aeroflex/Quicklogic FPGA	RAM128x18_25um RAM256X9_25um RAM512X4_25um RAM1024X2_25um	2 - 10	unlimited
easic90	eASIC 90 nm Nextreme	eram	2 - 12	unlimited

72.3 Signal descriptions

Table 732 shows the interface signals of the core (VHDL ports).

*Table 732.*Signal descriptions

Signal name	Field	Type	Function	Active
RCLK	N/A	Input	Read port clock	-
RENABLE	N/A	Input	Read enable	High
RADDRESS	N/A	Input	Read address bus	-
DATAOUT	N/A	Output	Data outputs for read data	-
WCLK	N/A	Input	Write port clock	-
WRITE	N/A	Input	Write enable	High
WADDRESS	N/A	Input	Write address	-
DATAIN	N/A	Input	Write data	-

72.4 Library dependencies

Table 733 shows libraries used when instantiating the core (VHDL libraries).

*Table 733.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

72.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component syncram_2p
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8; sepclk : integer
:= 0);
  port (
    rclk     : in std_ulogic;
    renable  : in std_ulogic;
    raddress : in std_logic_vector((abits -1) downto 0);
    dataout  : out std_logic_vector((dbits -1) downto 0);
    wclk     : in std_ulogic;
    write    : in std_ulogic;
    waddress : in std_logic_vector((abits -1) downto 0);
    datain   : in std_logic_vector((dbits -1) downto 0));
  end component;
```

72.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
use techmap.gencomp.all;

rclk     : in std_ulogic;
renable  : in std_ulogic;
raddress : in std_logic_vector((abits -1) downto 0);
dataout  : out std_logic_vector((dbits -1) downto 0);
wclk     : in std_ulogic;
```

```
write    : in std_ulogic;
waddress : in std_logic_vector((abits -1) downto 0);
datain   : in std_logic_vector((dbits -1) downto 0));

ram0 : syncram_2p generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( rclk, renable, raddress, dataout, wclk, write, waddress, datain, enable,
write);
```

73 SYNCRAM_DP - Dual-port RAM generator

73.1 Overview

The dual-port RAM generator has two independent read/write ports. Each port has a separate address and data bus. All inputs are latched on the on the rising edge of clk. The read data appears on dataout directly after the clk rising edge. Address width, data width and target technology is parametrizable through generics. Simultaneous write to the same address is technology dependent, and generally not allowed.

73.2 Configuration options

Table 734 shows the configuration options of the core (VHDL generics).

*Table 734.*Configuration options

Name	Function	Range	Default
tech	Technology selection	0 - NTECH	0
abits	Address bits. Depth of RAM is $2^{abits-1}$	see table below	-
dbits	Data width	see table below	-

Table 735 shows the supported technologies for the core.

*Table 735.*Supported technologies

Tech name	Technology	RAM cell	abit range	dbit range
altera	All altera devices	altsyncram	unlimited	unlimited
virtex	Xilinx Virtex, Virtex-E, Spartan-2	RAMB4_Sn	2 - 10	unlimited
virtex2	Xilinx Virtex2, Spartan3, Virtex4, Spartan3e	RAMB16_Sn	2 - 14	unlimited
proasic3	Actel Proasic3	ram4k9	2 - 12	unlimited
lattice	Lattice XP/EC/ECP	dp8ka	2 - 13	unlimited
memvirage	Virage ASIC RAM	hdss2_64x32cm4sw0 hdss2_128x32cm4sw0 hdss2_256x32cm4sw0 hdss2_512x32cm4sw0	6 - 9	32
memartisan	Artisan ASIC RAM	dp_256x32m4 dp_512x32m4 dp_1kx32m4	8 - 10	32
memvirage90	Virage 90 nm ASIC RAM	DPRAM_HS_256x20 DPRAM_HS_256x32	2 - 8	128

73.3 Signal descriptions

Table 736 shows the interface signals of the core (VHDL ports).

*Table 736.*Signal descriptions

Signal name	Field	Type	Function	Active
CLK1	N/A	Input	Port1 clock	-
ADDRESS1	N/A	Input	Port1 address	-
DATAIN1	N/A	Input	Port1 write data	-
DATAOUT1	N/A	Output	Port1 read data	-
ENABLE1	N/A	Input	Port1 chip select	High
WRITE1	N/A	Input	Port 1 write enable	High
CLK2	N/A	Input	Port2 clock	-
ADDRESS2	N/A	Input	Port2 address	-
DATAIN2	N/A	Input	Port2 write data	-
DATAOUT2	N/A	Output	Port2 read data	-
ENABLE2	N/A	Input	Port2 chip select	High
WRITE2	N/A	Input	Port 2 write enable	High

73.4 Library dependencies

Table 737 shows libraries used when instantiating the core (VHDL libraries).

*Table 737.*Library dependencies

Library	Package	Imported unit(s)	Description
TECHMAP	GENCOMP	Constants	Technology contants

73.5 Component declaration

The core has the following component declaration.

```
library techmap;
use techmap.gencomp.all;

component syncram_dp
  generic (tech : integer := 0; abits : integer := 6; dbits : integer := 8);
  port (
    clk1     : in std_ulogic;
    address1 : in std_logic_vector((abits -1) downto 0);
    datain1  : in std_logic_vector((dbits -1) downto 0);
    dataout1 : out std_logic_vector((dbits -1) downto 0);
    enable1  : in std_ulogic;
    write1   : in std_ulogic;
    clk2     : in std_ulogic;
    address2 : in std_logic_vector((abits -1) downto 0);
    datain2  : in std_logic_vector((dbits -1) downto 0);
    dataout2 : out std_logic_vector((dbits -1) downto 0);
    enable2  : in std_ulogic;
    write2   : in std_ulogic);
  end component;
```

73.6 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;
library techmap;
```

```
use techmap.gencomp.all;

clk1     : in std_ulogic;
address1 : in std_logic_vector((abits -1) downto 0);
datain1  : in std_logic_vector((dbits -1) downto 0);
dataout1 : out std_logic_vector((dbits -1) downto 0);
enable1  : in std_ulogic;
write1   : in std_ulogic;
clk2     : in std_ulogic;
address2 : in std_logic_vector((abits -1) downto 0);
datain2  : in std_logic_vector((dbits -1) downto 0);
dataout2 : out std_logic_vector((dbits -1) downto 0);
enable2  : in std_ulogic;
write2   : in std_ulogic);

ram0 : syncram_dp generic map ( tech => tech, abits => addrbits, dbits => dbits)
      port map ( clk1, address1, datain1, dataout1, enable1, write1, clk2, address2, datain2,
dataout2, enable2, write2);
```

74 TAP - JTAG TAP Controller

74.1 Overview

JTAG TAP Controller provides an Test Access Port according to IEEE-1149 (JTAG) Standard. The core implements the Test Access Port signals, the synchronous TAP state-machine, a number of JTAG data registers (depending on the target technology) and an interface to user-defined JTAG data registers.

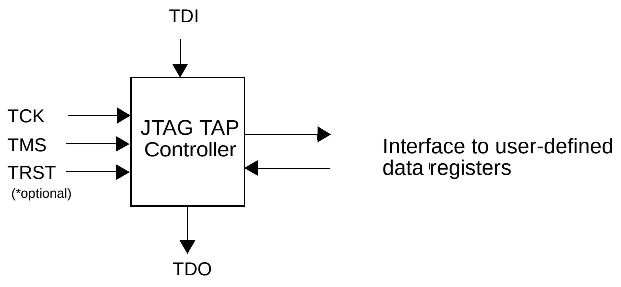


Figure 250. TAP controller block diagram

74.2 Operation

74.2.1 Generic TAP Controller

The generic TAP Controller implements JTAG Test Access Point interface with signals TCK, TMS, TDI and TDO, a synchronous state-machine compliant to the IEEE-1149 standard, JTAG instruction register and two JTAG data registers: bypass and device identification code register. The core is capable of shifting and updating the JTAG instruction register, putting the device into bypass mode (BYPASS instruction) and shifting out the devices identification number (IDCODE instruction). User-defined JTAG test registers are accessed through user-defined data register interface.

The access to the user-define test data registers is provided through the user-defined data register interface. The instruction in the TAP controller instruction register appears on the interface as well as shift-in data and signals indicating that the TAP controller is in Capture-Data-Register, Shift-Data-Register or Update-Data-Register state. Logic controlling user-defined data registers should observe value in the instruction register and TAP controller state signals in order to capture data, shift data or update data-registers.

JTAG test registers such as boundary-scan register can be interfaced to the TAP controller through the user data register interface.

74.3 Technology specific TAP controllers

The core instantiates technology specific TAP controller for Altera and Xilinx devices.

74.4 Registers

The core implements three JTAG registers: instruction, bypass and device identification code register.

74.5 Vendor and device identifiers

The core does not have vendor and device identifiers since it does not have AMBA interfaces.

74.6 Configuration options

Table 738 shows the configuration options of the core (VHDL generics).

*Table 738.*Configuration options

Generic	Function	Allowed range	Default
tech	Target technology	0 - NTECH	0
irlen	Instruction register length (generic tech only)	2 - 8	4
idcode	JTAG IDCODE instruction code(generic tech only)	0 - 255	9
manf	Manufacturer id. Appears as bits 11-1 in TAP controllers device identification register. Used only for generic technology. Default is Gaisler Research manufacturer id.	0 - 2047	804
part	Part number (generic tech only). Bits 27-12 in device id. reg.	0 - 65535	0
ver	Version number (generic tech only). Bits 31-28 in device id. reg.	0-15	0
trsten	Support optional TRST signal (generic tech only)	0 - 1	1

74.7 Signal descriptions

Table 739 shows the interface signals of the core (VHDL ports).

*Table 739.*Signal declarations

Signal name	Field	Type	Function	Active
TRST	N/A	Input	JTAG TRST signal*	Low
TCK	N/A	Input	JTAG clock*	-
TMS	N/A	Input	JTAG TMS signal*	High
TDI	N/A	Input	JTAG TDI signal*	High
TDO	N/A	Output	JTAG TDO signal*	High
TDOEN	N/A	Output	JTAG TDO enable signal*	High
User-defined data register interface				
TAPO_TCK	N/A	Output	TCK signal	High
TAPO_TDI	N/A	Output	TDI signal	High
TAPO_INST[7:0]	N/A	Output	Instruction in the TAP Ctrl instruction register	High
TAPO_RST	N/A	Output	TAP Controller in Test-Logic_Reset state	High
TAPO_CAPT	N/A	Output	TAP Controller in Capture-DR state	High
TAPO_SHFT	N/A	Output	TAP Controller in Shift-DR state	High
TAPO_UPD	N/A	Output	TAP Controller in Update-DR state	High
TAPO_XSEL1	N/A	Output	Xilinx User-defined Data Register 1 selected (Xilinx tech only)	High
TAPO_XSEL2	N/A	Output	Xilinx User-defined Data Register 2 selected (Xilinx tech only)	High
TAPI_EN1	N/A	Input	Enable shift-out data port 1 (TAPI_TDO1), when disabled data on port 2 is used	High
TAPI_TDO1	N/A	Input	Shift-out data from user-defined register port 1	High
TAPI_TDO2	N/A	Input	Shift-out data from user-defined register port 2	High

*) If the target technology is Xilinx or Altera the cores JTAG signals TCK, TCKN, TMS, TDI and TDO are not used. Instead the dedicated FPGA JTAG pins are used. These pins are implicitly made visible to the core through technology-specific TAP macro instantiation.

74.8 Library dependencies

Table 740 shows libraries used when instantiating the core (VHDL libraries).

*Table 740.*Library dependencies

Library	Package	Imported unit(s)	Description
GAISLER	JTAG	Component	TAP Controller component declaration

74.9 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library gaisler;
use gaisler.jtag.all;

entity tap_ex is
  port (
    clk : in std_ulogic;
    rst : in std_ulogic;

    -- JTAG signals
    tck  : in std_ulogic;
    tms  : in std_ulogic;
    tdi  : in std_ulogic;
    tdo  : out std_ulogic
);
end;

architecture rtl of tap_ex is

signal gnd : std_ulogic;

signal tapo_tck, tapo_tdi, tapo_rst, tapo_capt : std_ulogic;
signal tapo_shft, tapo_upd : std_ulogic;
signal tapi_en1, tapi_tdo : std_ulogic;
signal tapo_inst : std_logic_vector(7 downto 0);


begin

 gnd <= ‘0’;
 tckn <= not tck;


-- TAP Controller

  tap0 : tap (tech => 0)
    port map (rst, tck, tckn, tms, tdi, tdo, open, tapo_tck, tapo_tdi, tapo_inst,
      tapo_rst, tapo_capt, tapo_shft, tapo_upd, open, open,
       tapi_en1, tapi_tdo, gnd);


-- User-defined JTAG data registers

  ...


end;
```

75 USBDCL - USB Debug Communication Link

75.1 Overview

The Universal Serial Bus Debug Communication Link (USBDCL) provides an interface between an USB 2.0 bus and an AMBA-AHB bus. An external High-/Full-Speed Universal Transceiver Macrocell Interface (UTMI) with an 8-bit interface is needed to connect to the USB. The USBDCL is an AHB master and provides read and write access to the whole AHB address space using a simple protocol over two USB bulk endpoints. Figure 251 shows how the USBDCL should be connected to the UTMI.

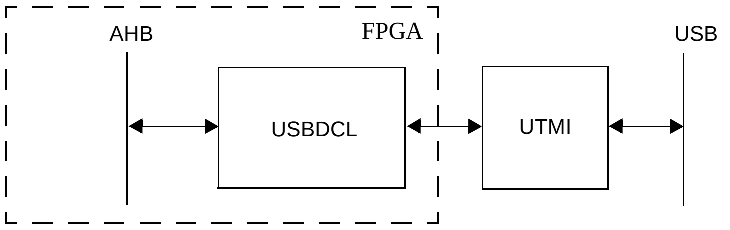


Figure 251. USBDCL connected to an external UTMI device.

75.2 Operation

75.2.1 System overview

Figure 252 shows the internal structure of the USBDCL. This section describes the function of the different blocks briefly.

The Speed Negotiation Engine (SNE) detects connection by monitoring VBUS on the USB connector. When a steady 5 V voltage is detected the SNE waits for a reset and then starts the High-speed negotiation. When the Speed negotiation and reset procedure is finished the selected speed mode is notified to the Serial Interface Engine (SIE).

The SIE is enabled when the SNE notifies that the reset procedure has finished. It then waits for packets to arrive and processes them according to the USB 2.0 specification. There are four endpoints: one in/out pair of control type with number 0 (default pipe) and one in/out pair of bulk type with number 1. Data received to an endpoint is stored in the endpoint's buffer located in block ram.

The AIE reads the packets from the endpoint buffers and if the packet was received to endpoint 0 the device request is processed directly and a response is stored in the buffer for IN endpoint 0. This is also notified to the SIE which transmits the data to the host when the next IN token arrives. Device requests never access the AHB bus.

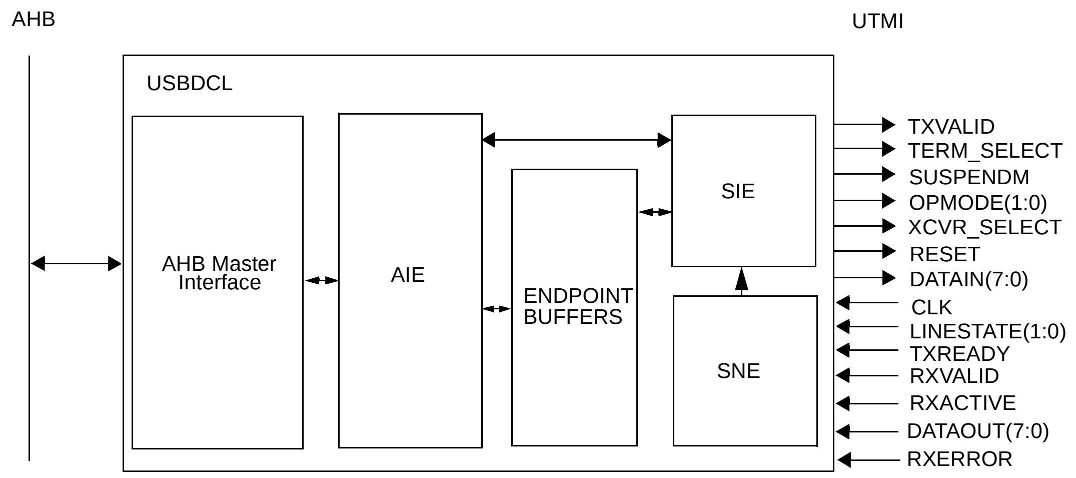


Figure 252. Block diagram of the internal structure of the USBDCL.

If the packet was received to endpoint 1 it is an AHB command and the AIE performs the operation on the AHB bus. Read commands arrive to OUT endpoint 1 and the read data from the AHB bus is stored in the buffer of IN endpoint 1 and transmitted in the same manner as for endpoint 0.

Write commands also arrive to OUT endpoint 1 and are executed on the bus immediately. No reply is sent for writes.

Each endpoint has two buffers that can hold one max payload packet each. The USBDCL automatically alternates between them when a packet has been received/transmitted. When it operates in high-speed mode OUT transactions are replied with a ACK handshake if the other buffer (the one to which the packet is not stored) is empty and with a NYET if is full. A NAK is sent if both are full since the packet cannot be received.

75.2.2 Protocol

The protocol used for the AHB commands is very simple and consists of two 32-bit control words. The first word consists of the 32-bit AHB address and the second consists of a read/write bit at bit 31 and the number of words to be written at bits 16 downto 2. All other bits in the second word are reserved for future use and must be set to 0. The read/write bit must be set to 1 for writes.

Figure 253 shows the layout of a write command. The command should be sent as the data cargo of an OUT transaction to endpoint 1. The data for a command must be included in the same packet. The maximum payload is 512 B when running in high-speed mode and 64 B in full-speed mode. Since the control information takes 8 B the maximum number of bytes per command is 504 B and 56 B respectively. Subword writes are not supported so the number of bytes must be a multiple of four between 0 and 504.

The words should be sent with the one to be written at the start address first. Individual bytes should be transmitted msb first, i.e. the one at bits 31-24.

There is no reply sent for writes since the USB handshake mechanism for bulk writes guarantees that the packet has been correctly received by the target.

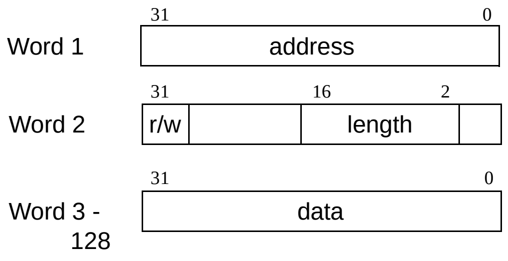


Figure 253. Layout of USBDCL write commands.

Figure 88 shows the layout of read commands and replies. In this case the command only consists of two words containing the same control information as the two first words for write commands. However, for reads the r/w bit must be set to 0.

When the read is performed data is read to the buffer belonging to IN endpoint 1. The reply packet is sent when the next IN token arrives after all data has been stored to the buffer. The reply packets only contains the read data (no control information is needed) with the word read from the start address transmitted first. Individual bytes are sent with most significant byte first, i.e. the byte at bit 31 downto 24.

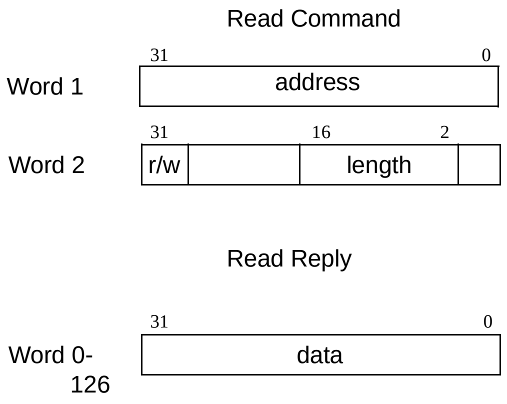


Figure 254. Layout of USBDCL read commands and replies.

75.2.3 AHB operations

All AHB operations are performed as incremental bursts of unspecified length. Only word size accesses are done.

75.3 Registers

The core does not contain any user accessible registers.

75.4 Vendor and device identifier

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x022. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

The USB vendor identifier is 0x1781 and product identifier is 0x0AA0.

75.5 Configuration options

Table 741 shows the configuration options of the core (VHDL generics).

*Table 741.*Configuration options

Generic	Function	Allowed range	Default
hindex	AHB master index.	0 - NAHBMST-1	0
memtech	Memory technology used for blockrams (endpoint buff-ers).	0 - NTECH	0

75.6 Signal descriptions

Table 742 shows the interface signals of the core (VHDL ports).

*Table 742.*Signal descriptions

Signal name	Field	Type	Function	Active
UCLK	N/A	Input	USB UTMI Clock	-
USBI		Input	USB UTMI Input signals	-
USBO		Output	USB UTMI Output signals	-
HCLK		Input	AMBA Clock	-
HRST		Input	AMBA Reset	Low
AHBMI	*	Input	AHB master input signals	-
AHBMO	*	Output	AHB master output signals	-

* see GRLIB IP Library User’s Manual

75.7 Library dependencies

Table 743 shows libraries used when instantiating the core (VHDL libraries).

*Table 743.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	USB	Signals, component	USBDCL component declarations, USB signals

75.8 Instantiation

This example shows how the core can be instantiated.

```
library ieee;
use ieee.std_logic_1164.all;

library grlib;
use grlib.amba.all;
use grlib.tech.all;
library gaisler;
use gaisler.usb.all;

entity usbdcl_ex is
  port (
    clk           : in std_ulogic; --AHB Clock
    rstn          : in std_ulogic;

    -- usb signals
    usb_clkout    : in std_ulogic;
    usb_d         : inout std_logic_vector(7 downto 0);
    usb_linestate : in std_logic_vector(1 downto 0);
    usb_opmode    : out std_logic_vector(1 downto 0);
    usb_reset     : out std_ulogic;
    usb_rxactive  : in std_ulogic;
    usb_rxerror   : in std_ulogic;
    usb_rxvalid   : in std_ulogic;
    usb_suspend   : out std_ulogic;
    usb_termsel   : out std_ulogic;
    usb_txready   : in std_ulogic;
    usb_txvalid   : out std_ulogic;
    usb_xcvrsel   : out std_ulogic;
    usb_vbus      : in std_ulogic);
end;

architecture rtl of usbdcl_ex is
  constant padtech : integer := inferred;
  constant memtech : integer := inferred;

  -- AMBA signals
  signal ahbmi : ahb_mst_in_type;
  signal ahbmo : ahb_mst_out_vector := (others => ahbm_none);

begin

  -- AMBA Components are instantiated here
  ...

  -- USBDCL
  usb_d_pads: for i in 0 to 7 generate
     usb_d_pad: iopad generic map(tech => padtech)
port map (usb_d(i), usbo.dout(i), usbi.rxactive, usbi.din(i));
    end generate;

  usbi0pad : inpad generic map (tech => padtech) port map (usb_txready,usbi.txready);
  usbi1pad : inpad generic map (tech => padtech) port map (usb_rxvalid,usbi.rxvalid);
  usbi2pad : inpad generic map (tech => padtech) port map (usb_rxerror,usbi.rxerror);
  usbi3pad : inpad generic map (tech => padtech) port map (usb_rxactive,usbi.rxactive);
  usbi4pad : inpad generic map (tech => padtech) port map
(usb_linestate(0),usbi.linestate(0));
  usbi5pad : inpad generic map (tech => padtech) port map
(usb_linestate(1),usbi.linestate(1));
  usbi6pad : inpad generic map (tech => padtech) port map (usb_vbus, usbi.vbus);
  usbo0pad : outpad generic map (tech => padtech) port map (usb_reset,usbo.reset);
  usbo1pad : outpad generic map (tech => padtech) port map (usb_suspend,usbo.suspend);
  usbo2pad : outpad generic map (tech => padtech) port map (usb_termsel,usbo.termselect);
  usbo3pad : outpad generic map (tech => padtech) port map (usb_xcvrsel,usbo.xcvrselect);
  usbo4pad : outpad generic map (tech => padtech) port map (usb_opmode(0),usbo.opmode(0));
  usbo5pad : outpad generic map (tech => padtech) port map (usb_opmode(1),usbo.opmode(1));
  usbo6pad : outpad generic map (tech => padtech) port map (usb_txvalid,usbo.txvalid);

  usb_clk_pad : clkpad generic map (tech => padtech) port map (usb_clkout, uclk);


usb_ctrl : usbdcl
    generic map (
      hindex => 0, memtech => memtech)
    port map (
      uclk, usbi, usbo, clkm, rstn, ahbmi, ahbmo(0));
end;
```

76 WILD2AHB - WildCard Debug Interface with AHB Master Interface

76.1 Overview

The WildCard debug interface provides access to the on-chip AMBA AHB bus through the WildCard CardBus Controller. Through this debug interface, a 32-bit read or write transfer can be generated to any address on the AMBA AHB bus.

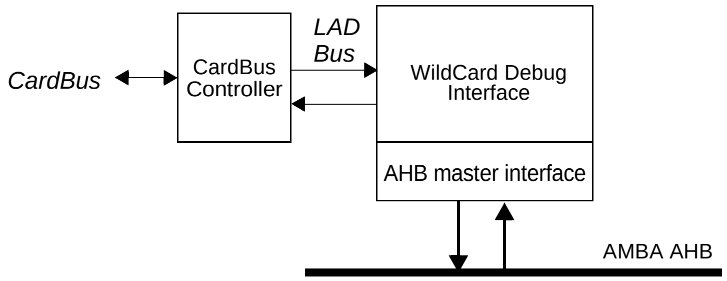


Figure 255. WildCard Debug Interface block diagram

76.2 WildCard

The WildCard™ is a development board from Annapolis Micro Systems. The WildCard is a PCMCIA card with the following features:

- 32-bit CardBus interface with multichannel DMA controller and programmable clock generator
- single Processing Element: Virtex™ XCV300E -6 FPGA
- two independent memory ports connected to two 10 ns SSRAM devices
- two independent 15 pin I/O connectors
- Windows® CardBus driver or Linux® CardBus driver

The Local Address and Data (LAD) Bus interface connects the Processing Element (i.e. Xilinx FPGA) on the WildCard with the proprietary CardBus Controller device. The CardBus Controller device communicates with the host PC via the CardBus using a WildCard specific API.

The WildCard comes with templates, example VHDL designs and software routines. It is only possible to access the WildCard via a specific Application Programming Interface (API). The necessary Windows drivers and API are delivered with the card. To simulate a WildCard system, the VHDL template design delivered with the WildCard can be used.

For more information, visit http://www.annapmicro.com

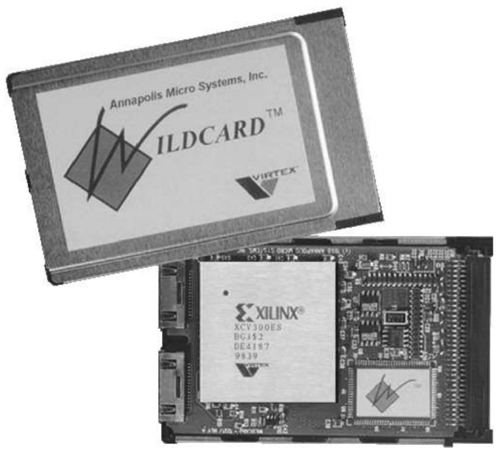


Figure 256. WildCard

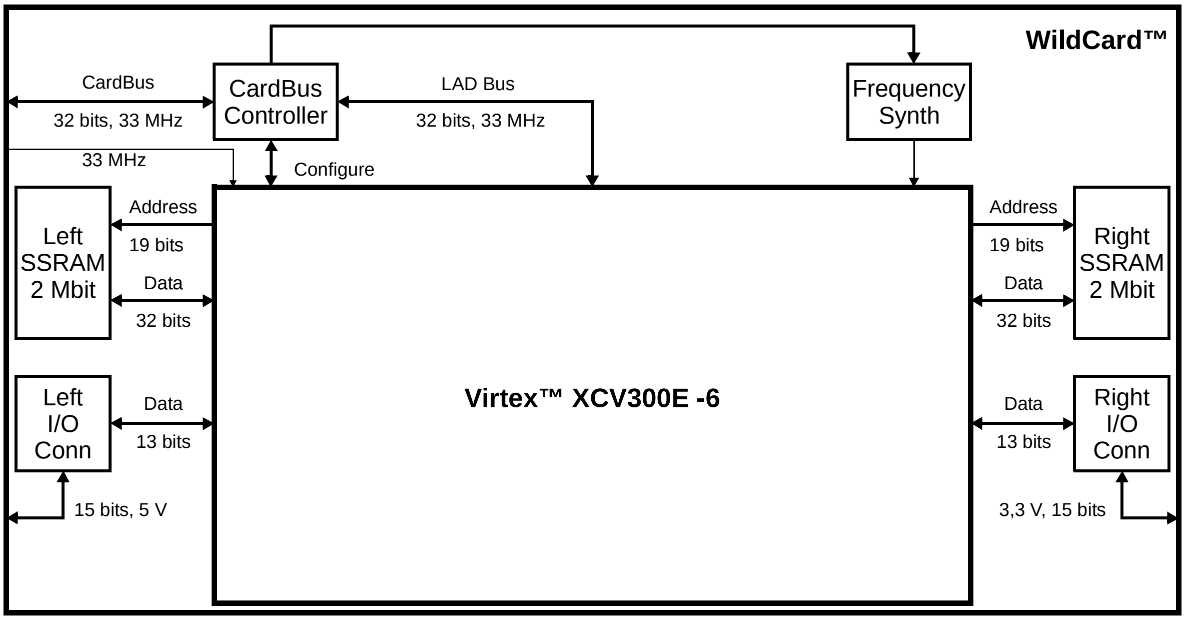


Figure 257. WildCard block diagram

76.3 Operation

The WildCard debug interface implements a simple protocol for access the AMBA AHB bus from the CardBus Controller device. The WildCard API provides read and write access from the WildCard CardBus Controller device to the Processing Element (i.e. FPGA), using the Local Address and Data (LAD) Bus interface. The WildCard debug interface is connected to the LAD bus on one side, and the AMBA AHB bus on the other side.

An AMBA write access is performed by writing the data to the Write Data Register, followed by writing the AMBA address to the Write Address Register which starts the access on the AMBA bus. The progress of the AMBA access is observed via the Status Register. When the AMBA access is completed, the Status Registers must be cleared by writing any data to the Status Register. The size of the data transfer, in number of words, can be specified by writing to the Size Register before writing to the Write Address Register, this is only required when the size needs to be changed.

An AMBA read access is performed by writing the AMBA address to the Read Address Register which starts the access on the AMBA bus. The progress of the AMBA access is observed via the Status Register. When the AMBA access is completed, the data can be read from the Read Data Register, and the Status Registers must be cleared by writing any data to the Status Register. The size of the data transfer, in number of words, can be specified by writing to the Size Register before writing to the Read Address Register, this is only required when the size needs to be changed.

The burst capability of the interface can be determined by reading the Version Register, indicating the size of the largest possible data transfer in number of words.

The WildCard debug interface supports 32-bit word accesses. It supports AMBA retry and split, automatically retrying the access till completed successfully or with an AMBA error. On the completion of an AMBA access, the Ready bit in the Status Register will be set. If the AMBA access completed with an AMBA error, the Error bit in the Status Register will also be set. Both bits are cleared when the Status Register is written to.

Table 744. WildCard registers

Address offset	API address	Register
`0x10000`	`0x4000`	Status register
`0x10004`	`0x4001`	Control register (unused)
`0x10008`	`0x4002`	Size register
`0x1000C`	`0x4003`	Version register (read only)
`0x10010`	`0x4004`	Read address register (starts read access) (write only)
`0x10020`	`0x400C`	Write address register (starts write access) (write only)
`0x10200-0x102FF`	`0x4080-0x40BF`	Read data (read only)
`0x10300-0x103FF`	`0x40C0-0x40FF`	Write data (write only)
`0x08000`	`0x2000`	Processing Element version register (read only)
`0x08000-0x0FFFC`	`0x2000-0x2FFF`	Reserved address space
`0x10000-0x1FFFC`	`0x4000-0x7FFF`	User address space

Table 745. Status register

31 ... 3	2	1	0
“000 ... 0”	ERROR	ACITVE	READY

2	Error	AMBA error when access completed and Error=1. Cleared when register written to.
1	Active	Access is on-going (read only)
0	Ready	Access completed when Ready=1. Cleared when register written to.

Table 746. Size register

31 ... 0
SIZE

31:	0	Size	Number of words to be transferred

Table 747. Version register

31 ... 12	11 ... 4	3 ... 0
“000 ... 0”	BURST LENGTH	REVISION

11:	4	Supported burst length in number of words
3:	0	Revision

Table 748. Read address register

31 ... 0
Read Address

31:	0	Read address (write only)

Table 749. Write address register

31 ... 0
Write Address

31:	0	Write address (write only)

Table 750. Read data

31 … 0
Read Data

31: 0 Read data (read only)

Table 751. Write data

31 … 0
Write Data

31: 0 Write data (write only)

Table 752. Processing element version

31 … 16	15 … 8	7 … 0
RESERVED	MAJOR VERSION	MINOR VERSION

31:	16	Reserved (read only)
15:	8	Major version (read only)
731:	0	Minor version (read only)

76.4 Registers

The core does not implement any registers mapped in the AMBA AHB or APB address space.

76.5 Vendor and device identifiers

The core has vendor identifier 0x01 (Gaisler Research) and device identifier 0x079. For description of vendor and device identifiers see GRLIB IP Library User’s Manual.

76.6 Configuration options

Table 753 shows the configuration options of the core (VHDL generics).

*Table 753.*Configuration options

Generic	**Function**	**Allowed range**	**Default**
hindex	AHB master index	0 - NAHBMST-1	0
burst	Burst length = 2 ** *burst*	0, 3-6	0
syncrst	Choose between synchronous and asynchronous reset.	0 - 1	0

76.7 Signal descriptions

Table 754 shows the interface signals of the core (VHDL ports).

*Table 754.*Signal descriptions

Signal name	Field	Type	Function	Active
RSTKN	N/A	Input	Reset (PCI clock domain)	Low
CLKK	N/A	Input	CardBus clock (PCI clock domain)	Falling
RSTFN	N/A	Input	Reset (AHB clock domain)	Low
CLKF	N/A	Input	System clock (AHB clock domain)	Rising
AHBI	*	Input	AHB Master interface input	-
AHBO	*	Output	AHB Master interface output	-
LADI	Addr_Data	Input	Shared address/data bus input	-
	AS_n		Address strobe	Low
	DS_n		Data strobe	Low
	WR_n		Write select	Low
	CS_n		Chip select	Low
	Reg_n		Register select	Low
LADO	Addr_Data	Output	Shared address/data bus output	-
	Addr_Data_OE_n		Address/data bus output enable	High
	Ack_n		Acknowledge strobe	Low
	Int_Req_n		Interrupt request	Low
	DMA_0_Data_OK_n		DMA channel 0 data OK flag	Low
	DMA_0_Burst_OK		DMA channel 0 burst OK flag	High
	DMA_1_Data_OK_n		DMA channel 1 data OK flag	Low
	DMA_1_Burst_OK		DMA channel 1 burst OK flag	High
	Reg_Data_OK_n		Register space data OK flag	Low
	Reg_Burst_OK		Register space burst OK flag	High
	Force_K_Clk_n		K_Clk forced-run select	Low
	Reserved		Reserved for future use	-

*) see GRLIB IP Library User’s Manual

76.8 Library dependencies

Table 755 shows libraries used when instantiating the core (VHDL libraries).

*Table 755.*Library dependencies

Library	Package	Imported unit(s)	Description
GRLIB	AMBA	Signals	AMBA signal definitions
GAISLER	WILD	Signals, component	Signals and component declaration

76.9 Instantiation

This example shows how the core can be instantiated.

```
library  ieee;
use      ieee.std_logic_1164.all;

entity wildfpga is
   port (
      ...
      LAD_Bus_Addr_Data:           inout  std_logic_vector (31 downto 0);
      LAD_Bus_AS_n:                in     std_logic;
      LAD_Bus_DS_n:                in     std_logic;
      LAD_Bus_WR_n:                in     std_logic;
      LAD_Bus_CS_n:                in     std_logic;
      LAD_Bus_Reg_n:               in     std_logic;
      LAD_Bus_Ack_n:               out    std_logic;
      LAD_Bus_Int_Req_n:           out    std_logic;
      LAD_Bus_DMA_0_Data_OK_n:     out    std_logic;
      LAD_Bus_DMA_0_Burst_OK:      out    std_logic;
      LAD_Bus_DMA_1_Data_OK_n:     out    std_logic;
      LAD_Bus_DMA_1_Burst_OK:      out    std_logic;
      LAD_Bus_Reg_Data_OK_n:       out    std_logic;
      LAD_Bus_Reg_Burst_OK:        out    std_logic;
      LAD_Bus_Force_K_Clk_n:       out    std_logic;
      LAD_Bus_Reserved:            out    std_logic;
      ...
   );
end entity wildfpga;

library  ieee;
use      ieee.std_logic_1164.all;

library  work;
use      work.config.all;

library  grlib;
use      grlib.amba.all;

library  gaisler;
use      gaisler.wild.all;

library  unisim;
use      unisim.vcomponents.fd;

rchitecture rtl of wildfpga is
   ...
   -- local address and data bus
   signal   ladi:             lad_in_type;
   signal   lado:             lad_out_type;

   -- amba interface
   signal   ahbmi:            ahb_mst_in_type;
   signal   ahbmo:            ahb_mst_out_vector := (others => ahbm_none);
   ...
begin
   ...
   -----------------------------------------------------------------------------
   -- Local Address and Data Bus to AMBA AHB bus DMA interface
   -----------------------------------------------------------------------------
   wild2ahb0: wild2ahb
      generic map (
         hindex   => 1,
         burst    => 0,
         syncrst  => 1)
      port map(rstkn, clkk, rstfn, clkf, ahbmi, ahbmo(1), ladi, lado);

   -----------------------------------------------------------------------------
   -- Local Address and Data Bus pads
   -----------------------------------------------------------------------------
   addr_data_pad : for i in LAD_Bus_Addr_Data'range generate
    ad_pad : iopad
      generic map(
         slew     => 1,
         strength => 24)
      port map(
         pad      => LAD_Bus_Addr_Data(i),
         i        => lado.Addr_Data(i),
         en       => lado.Addr_Data_OE_n(i),
         o        => ladi.Addr_Data(i));
   end generate;

   ladi.AS_n                  <= LAD_Bus_AS_n;
   ladi.DS_n                  <= LAD_Bus_DS_n;
   ladi.WR_n                  <= LAD_Bus_WR_n;
   ladi.CS_n                  <= LAD_Bus_CS_n;
   ladi.Reg_n                 <= LAD_Bus_Reg_n;

   LAD_Bus_Ack_n              <= lado.Ack_n;
   LAD_Bus_Int_Req_n          <= lado.Int_Req_n;
   LAD_Bus_DMA_0_Data_OK_n    <= lado.DMA_0_Data_OK_n;
   LAD_Bus_DMA_0_Burst_OK     <= lado.DMA_0_Burst_OK;
   LAD_Bus_DMA_1_Data_OK_n    <= lado.DMA_1_Data_OK_n;
   LAD_Bus_DMA_1_Burst_OK     <= lado.DMA_1_Burst_OK;
   LAD_Bus_Reg_Data_OK_n      <= lado.Reg_Data_OK_n;
   LAD_Bus_Reg_Burst_OK       <= lado.Reg_Burst_OK;
   LAD_Bus_Force_K_Clk_n      <= lado.Force_K_Clk_n;
   LAD_Bus_Reserved           <= lado.Reserved;

  ...
end architecture rtl;
```

Gaisler Research AB
Första Långgatan 19
413 27 Göteborg
Sweden

tel +46 31 7758650
fax +46 31 421407
sales@gaisler.com
www.gaisler.com