Provartec

PR201
Configurable Dual-Core High Performance AHB DMA
Reference Guide

Revision 1.5

Preface

About This Manual

This document describes the Configurable Dual-Core High Performance AHB DMA (PR201).

Notational Conventions

This document uses the following conventions:

- Hexadecimal numbers are shown with the prefix 0x.

Related Documents

- “AMBA AHB-Lite Protocol”, can be downloaded from http://infocenter.arm.com
- "AMBA 3 APB Protocol Specification”, can be downloaded from http://infocenter.arm.com
- DMA_Builder_manual.pdf - DMA Builder user guide

Release information

Date	Revision	Change
26/July/2010	Rev 1.0	Initial revision
29/Aug/2010	Rev 1.4	Version number updated to parallel PR200
5/Oct/2010	Rev 1.5	Added endianness byte swapping

Table of Contents

1. Introduction....6
   1.1 General....6
   1.2 The Dual-Core concept....6
   1.3 Construction options....8
   1.4 DMA Builder application....9
   1.5 Main Feature List....9
2. External connections....10
   2.1 Port list....10
   2.2 Connecting to AHB....12
   2.3 Connecting to APB3....12
   2.4 Connecting to peripherals....12
3. Operation modes....13
   3.1 General....13
   3.2 Independent mode – normal channel mode....13
   3.3 Independent mode – outstanding channel mode....14
   3.3 Joint mode....15
4. Concepts....18
   4.1 DMA commands....18
   4.2 DMA command lists - optional feature (Command lists)....18
   4.3 Peripheral control - optional feature (Peripheral control)....19
   4.4 Peripheral to peripheral transfer - optional feature (Peripheral control)....20
   4.5 Arbitration – Build can remove high and top priority modes (Priority modes)....20
   4.6 Tokens (Windowed arbitration) - optional feature (Tokens)....21
   4.7 Block transfer - optional feature (Block support)....21
   4.8 Block scatter gather - optional feature (Block support)....21
   4.9 Peripheral block transfer - optional feature (Block support & Peripheral control)....21
   4.10 Scheduled channels - optional feature (Scheduler)....21
   4.11 Interrupt depth - optional feature (Command lists)....22
   4.12 Software control of peripheral request - optional feature (Peripheral control)....22
   4.13 Multiple processor control - optional feature (Number of interrupts)....22
   4.14 AHB timeouts - optional feature (AHB timeout)....22
   4.15 Watchdog timer - optional feature (Watchdog timer)....23
   4.16 Clock gating - optional feature (Clock gating)....23
   4.17 Multiple output port control - optional feature (Interconnect)....23
   4.18 Core 1 clock divider....23
   4.19 AHB port mux....23
   4.20 Endianness byte swapping - optional feature (Endianness)....24
5. Configuration Flows....24
   5.1 General configuration....24
   5.2 Configure and start a channel....24
   5.3 Stop a channel....24
   5.4 Pause and resume a channel....25
   5.5 Restart a channel....25
   5.6 Interrupt handling....25
6. Performance....26
   6.1 General....26
   6.2 Independent mode, 64 bit data bus....26
   6.3 Independent mode, 32 bit data bus....27

6.4 Independent mode – outstanding channel mode....28
6.5 Independent mode - multiple channels....30
6.6 Independent mode - dual cores with a shared AHB bus....30
6.7 Independent mode - block transfer....32
6.8 Joint mode....33
7. Area and frequency examples....34
7.1 Single core design....34
7.2 Dual core design....35
8. Bus activity....36
8.1 AHB bursts used....36
8.2 Error interrupts....36
9. Registers....37
9.1 General....37
9.2 Channel registers....37
9.3 Shared registers....51

Index of Tables

Table 1: Compare DMA functionalities....6
Table 2: Pin list....11
Table 3: Burst flow example – independent mode....14
Table 4: Burst flow example – joint mode....17
Table 5: Scatter list example....19
Table 6: Cyclic buffer example....19
Table 7: Normal priority example....20
Table 8: High priority example....20
Table 9: Top priority example....20
Table 10: Performance conclusion....31
Table 11: Single core gate count....32
Table 12: Dual core gate count....33

Index of Illustrations

Illustration 1: Dual cores, single clock....7
Illustration 2: Dual cores, dual clocks....7
Illustration 3: Dual cores with interconnect....7
Illustration 4: Bus activity, single channel, 64 bit, buffer size 32 bytes....26
Illustration 5: Bus activity, single channel, 64 bit, buffer size 128 bytes....26
Illustration 6: Bus activity, single channel, 64 bit, buffer size 64 bytes....26
Illustration 7: Bus activity, single channel, 32 bit, buffer size 32 bytes....27
Illustration 8: Bus activity, single channel, 32 bit, buffer size 64 bytes....27
Illustration 9: Bus activity, single channel, 32 bit, buffer size 128 bytes....27
Illustration 10: Bus activity, single channel, 32 bit, buffer size 32 bytes, read outstanding....28
Illustration 11: Bus activity, single channel, 32 bit, buffer size 32 bytes, write outstanding....28
Illustration 12: Bus activity, single channel, 32 bit, buffer size 32 bytes, read and write outstanding....28
Illustration 13: Bus activity, multiple channels, 64 bit, buffer size 64 bytes....29
Illustration 14: Bus activity, dual cores, single channel per core, shared clock....29
Illustration 15: Bus activity, dual cores, single channel per core, clock ratio 4....30
Illustration 16: Illustration 19: Bus activity, block mode, single channel, single block 32x8....30
Illustration 17: Bus activity, block mode, two channels, single block 32x8....30
Illustration 18: Bus activity, joint mode, single channel, unaligned addresses....31
Illustration 19: Bus activity, joint mode, multiple channels, minimal buffer size....31

Configurable Dual-Core DMA

This document describes the Configurable Dual-Core DMA (PR201).

1. Introduction

1.1 General

The Dual-Core DMA is a high performance 64 bit AHB master.

1.2 The Dual-Core concept

The DMA transfers data between different points in the memory space without intervention of the CPU. The DMA is generally used to replace two CPU functions, memory copy and peripheral control (slow peripheral devices such as SPI, UART, etc.). These two functionalities are very different in nature as shown in the table below:

	Memory copy	**Peripheral control**
Latency required	Short as possible	Can be long
Transfer speed	Fast as possible	Can be slow
Buffer size	Preferably large	Can be small
Read & write bursts	Usually same	Often different
Channels required	Usually few	Usually many

Table 1: Compare DMA functionalities

The Dual-Core solution separates these two functionalities. One core services memory copy requests and the other services all slow peripheral devices. The different cores can be configured to completely different topographies, saving area, power and improving performance.

Let us consider the Dual-Core design.

In illustration 1 two independent AHB master cores are shown, powered by the same clock. The memory transfer core is constructed with deep buffers while the peripheral core is constructed to the minimum.

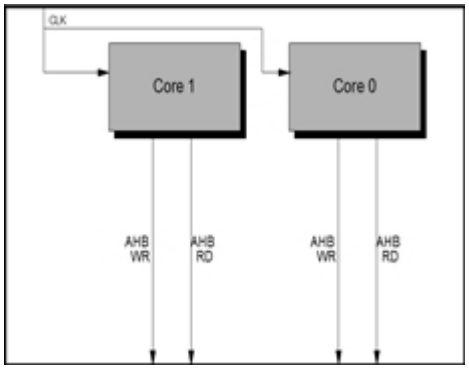

Illustration 1: Dual cores, single clock

Since the peripheral core is not required to work as quickly as the memory core, a clock divider can be added on the peripheral core clock.

In illustration 2 a clock divider and an AHB bus synchronizer have been added on the peripheral core.

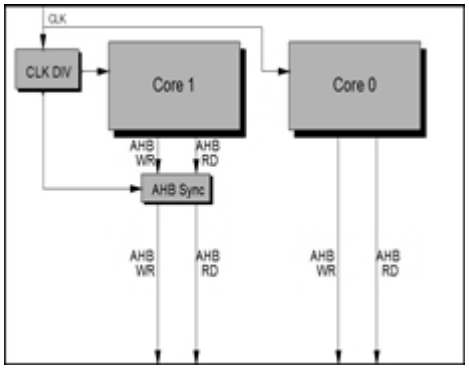

Illustration 2: Dual cores, dual clocks

Finally, in illustration 3, an AHB matrix can be added to allow each channel to simultaneously access two AHB ports or output a single AHB port.

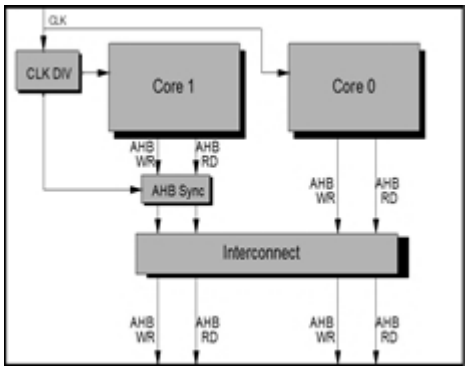

Illustration 3: Dual cores with interconnect

1.3 Construction options

The design is constructed according to the following configurations:

General build options:

- Single or Dual core
- Number of interrupts (number of controlling processors)
- Use single output or dual output interconnect
- Use clock divider for Core 1
- Insert clock gating

Core build options:

- Channel number (1-8)
- Data width (32 or 64 bits)
- Data buffer size (16-512 bytes)
- Address bits (16-32)
- Buffer size bits (9-16)
- AHB mux (on read and write buses)

Core optional features:

- Block support
- Scheduled channels
- Three level priority modes
- Joint mode support
- Independent mode support
- Outstanding mode support
- Command lists support
- Usage of tokens (windowed arbitration)
- Timeouts on all AHB buses
- Watchdog timer
- Peripheral control

1.4 DMA Builder application

The design can be easily configured using the *DMA Builder* application.

For more information see related document or go to:

www.provartec.com/dma-builder

1.5 Main Feature List

Key features:

- Dual Core design
- Configurable build and optional features
- Clock divider for slow channels
- Block transfer in a frame context
- Three operation modes: independent, outstanding and joint
- Three level priority arbitration
- Windowed channel arbitration (tokens)
- Configurable interrupt controller with multiple processor support
- Supports any address alignment
- Supports any buffer size alignment
- Supports command lists, including block lists
- Peripheral flow control, including peripheral block transfer
- Peripheral to peripheral transfer
- Scheduled transfers
- Endianness byte swapping
- Software control peripheral request
- Watchdog timer
- Channel pause and resume
- APB3 registers
- Complete status register set for debug

AHB main capabilities:

- Compliant to AMBA AHB protocol
- Each channel can control up to two AHB read ports and two AHB write ports
- Independent and simultaneous read and write control (each channel can simultaneously read and write on different ports)
- 32 or 64 bit bus support
- Maximum throughput transfers, regardless of alignment
- Supports AHB response error
- Support timeout on all AHB buses

2. External connections

2.1 Port list

Name	Bits	Direction	Description
clk	1	In	Main clock
reset	1	In	Reset
scan_en	1	In	Scan enable
INT	1-8 (configurable)	Out	Interrupt bus
idle	1	Out	Idle indication
***The following periph ports can be removed in build**			
periph_tx_req	[31:1]	Out	Peripheral TX request
periph_tx_clr	[31:1]	In	Peripheral TX clear
periph_rx_req	[31:1]	Out	Peripheral RX request
periph_rx_clr	[31:1]	In	Peripheral RX clear
pclken	1	In	APB3 clock enable (integer ratio only)
psel	1	In	APB3 select
penable	1	In	APB3 enable
paddr	13	In	APB3 address
pwrite	1	In	APB3 write qualifier
pwdata	32	In	APB3 write data
prdata	32	Out	APB3 read data
pslverr	1	Out	APB3 slave error
pready	1	Out	APB3 ready
WHADDR0	16-32 (configurable)	Out	CORE0 AHB write address
WHBURST0	3	Out	CORE0 AHB write burst
WHSIZE0	2	Out	CORE0 AHB write size
WHTRANS0	2	Out	CORE0 AHB write trans
WHWDATA0	32 or 64 (configurable)	Out	CORE0 AHB write data
WHREADY0	1	In	CORE0 AHB write ready
WHRESP0	1	In	CORE0 AHB write response

RHADDR0	16-32 (configurable)	Out	CORE0 AHB read address
RHBURST0	3	Out	CORE0 AHB read burst
RHSIZE0	2	Out	CORE0 AHB read size
RHTRANS0	2	Out	CORE0 AHB read trans
RHWDATA0	32 or 64 (configurable)	In	CORE0 AHB read data
RHREADY0	1	In	CORE0 AHB read ready
RHRESP0	1	In	CORE0 AHB read response
***The following AHB port depends on build (see details below)**			
WHADDR1	16-32 (configurable)	Out	CORE1 AHB write address
WHBURST1	3	Out	CORE1 AHB write burst
WHSIZE1	2	Out	CORE1 AHB write size
WHTRANS1	2	Out	CORE1 AHB write trans
WHWDATA1	32 or 64 (configurable)	Out	CORE1 AHB write data
WHREADY1	1	In	CORE1 AHB write ready
WHRESP1	1	In	CORE1 AHB write response
RHADDR1	16-32 (configurable)	Out	CORE1 AHB read address
RHBURST1	3	Out	CORE1 AHB read burst
RHSIZE1	2	Out	CORE1 AHB read size
RHTRANS1	2	Out	CORE1 AHB read trans
RHWDATA1	32 or 64 (configurable)	In	CORE1 AHB read data
RHREADY1	1	In	CORE1 AHB read ready
RHRESP1	1	In	CORE1 AHB read response

Table 2: Pin list

* The second AHB port will be present if the design is a dual-core design without an interconnect, or a design using a dual output port interconnect (single or dual core).

2.2 Connecting to AHB

AHB bus is part of the AMBA AHB specification. For further information see the "AMBA AHB-Lite Protocol" (related documents).

2.3 Connecting to APB3

APB bus is part of the AMBA3 specification. For further information see the "AMBA 3 APB Protocol Specification" (related documents).

Configuration registers use the main clock. Core 1 uses the main clock for configuration also when it uses a clock divider. The APB bus can be slowed down by using the pclken input, pclken ratio must be an integer value.

2.4 Connecting to peripherals

Each peripheral can either be an RX or a TX device and is attached by 2 wires, periph_req and periph_clr. The master on this interface is the peripheral, raising the periph_req when it requests data to be written or read from it. Once the burst has been transferred (on AHB) the DMA will issue a pulse on periph_clr indicating the peripheral to update its periph_req. The peripherals periph_req and periph_clr signals must be connected on the same bit number on the DMA's periph buses, the channel servicing this peripheral will have the same bit number set in its PERIPH_NUM register.

3. *Operation modes*

3.1 General

The operation mode determines how the channels use the AHB read and write buses. Each core is configured to work in 'Independent mode' or in 'Joint mode'. If using a dual core design, the two cores can be configured differently. When using 'Independent mode' each channel can work in normal or in outstanding mode. When using 'Joint mode', each channel can work in normal or in joint mode.

3.2 Independent mode – normal channel mode

When using this mode, each core will use an independent arbiter for read operations and for write operations. Read operations on the AHB bus will be issued without considering the write operations.
This mode is configured in the CORE0_JOINT_MODE, CORE1_JOINT_MODE registers.

Each channel will work in the following manner:

1. The channel calculates the next read and write bursts sizes according to AHB, address and software restrictions.
2. The channel will issue read bursts as long as the data buffer can hold the data.
3. The channel will issue write bursts as long as the write data is present in the data buffer.

This mode is most efficient when:

- Channels read and write bursts are of different sizes (like when transferring data from peripheral devices to DRAM).
- AHB slaves have unpredicted response times or response times are long.
- Several channels work simultaneously and overall performance is the main goal.

Advantages:

- Maximum overall performance for the above cases.

Disadvantages:

- The channel will not start a write burst until the read data has arrived, this causes long latency and in order to achieve maximum throughput the data buffer must be enlarged.
- When using read or write addresses that are not aligned to the data buffer size, the maximum burst supported will be half of the data buffer size.

Burst flow:

In order to achieve maximum throughput regardless to alignments, the controller will start with single AHB commands until reaching a round address (according to data width) from where full strobe bursts are possible.
The calculation of burst length considers the following:

- Does not exceed the value set in BURST_MAX_SIZE register
- Does not exceed FIFO size
- Burst width must be aligned to burst address

Data flow example using a 32 bit data bus:

FIFO_SIZE = 64 bytes
RD_START_ADDR = 0x30000001
RD_BURST_MAX_SIZE = 64 bytes
WR_START_ADDR = 0x40000017
WR_BURST_MAX_SIZE = 64 bytes
BUFFER_SIZE = 128 bytes

Read					Write			
Burst address	**Burst type**	**Burst size**	**Buffer remain**		**Burst address**	**Burst type**	**Burst size**	**Buffer remain**
0x30000001	SINGLE	1	128		0x40000017	SINGLE	1	128
0x30000002	SINGLE	2	127		0x40000018	SINGLE	4	127
0x30000004	SINGLE	4	125		0x4000001C	SINGLE	4	123
0x30000008	SINGLE	4	121		0x40000020	INCR8	32	119
0x3000000C	SINGLE	4	117		0x40000040	INCR16	64	8
0x30000010	INCR4	16	113		0x40000080	INCR4	16	23
0x30000020	INCR8	32	97		0x40000090	SINGLE	4	7
0x30000040	INCR16	64	65		0x40000094	SINGLE	2	3
0x30000080	SINGLE	1	1		0x40000096	SINGLE	1	1

Table 3: Burst flow example – independent mode

3.3 Independent mode – outstanding channel mode

When the core is configured to 'Independent mode', each channel can be configured to 'read outstanding', 'write outstanding' or both. This mode works like the normal independent mode, the difference is that when using 'read outstanding' the read commands will be issued once the write commands have been issued, before the write data has actually been written out. When using 'write outstanding' the write commands will be issued once the read commands have been issued, before the read data has actually been read out of the data buffer. This mode works under the assumption that the AHB slave will respond quickly, otherwise the data buffer will overflow (or underflow). If the channel's buffer overflows or underflows the channel will issue an error interrupt and will automatically stop its operation.
In case of an error interrupt the following options are possible:

- If the slave is just too slow to work under these conditions, reconfigure the channel to 'normal mode' and restart the channel.
- If the slave is too slow but rarely, the channel can be restarted (see 'Restart a channel' configuration flow).
- The retry option can also be issued automatically by writing to the AUTO_RETRY register. If this is set, whenever an overflow or an underflow occurs, the channel will flush its buffers and restart at the beginning of the current command. The

interrupt error will still be issued.

This mode is configured in the RD_OUTSTANDING and WR_OUTSTANDING registers in each channel.

This mode is most efficient when:

- Working with fast responding channels.

Advantages:

- Improved latency and throughput for small data buffers.

Disadvantages:

- If the slave does not respond in time the channel will overflow or underflow and stop.

Restrictions:
Outstanding mode can not work with the following configurations:

- Joint mode
- Peripherals
- Non aligned block transfer (works with normal non aligned transfer)

3.3 Joint mode

When using this mode, each core will use a single arbiter for both read and write operations. This mode is used for channels that work at the same pace for their read and write operations. The current channel locks both read and write AHB buses and transfers the read data directly to the write data. When working in this mode the channel's configuration is done in the read registers only and they affect both read and write operations that are simulations.
This mode is configured in the CORE0_JOINT_MODE, CORE1_JOINT_MODE registers and JOINT_MODE register in each channel.

Each channel will work in the following manner:

1. The channel will read single bursts until it will reach an aligned address from which it can perform its requested bursts.
2. The channel will write single bursts until it will reach an aligned address from which it can perform it requested bursts.
3. The channel moves into its joint stage, during this stage the channel will read and write simultaneously on both AHB buses.
4. The channel will perform single bursts to finish its remaining last bytes.

This mode is most efficient when:

- The read and write slaves work in same size bursts (must).
- The slave does not have many wait cycles (wait cycles on the read bus stall the write bus and vice-versa).

•Working with large transfer buffers.

Advantages:

•Improved latency, improved performance.

•The maximum burst size is not restricted regardless to alignments and regardless to data buffer size. The maximum supported burst is 16 strobes (128 bytes for 64 bit data bus or 64 bytes for 32 bit data bus).

Disadvantages:

•It is possible to use in the same core channels working in 'joint mode' and 'normal mode' but these do not work very well together since there is a flush stage when transferring from a 'joint' working channel to a 'normal' working channel.

•The 'normal' working channels in a 'joint' working core can still use only a single arbiter.

Burst flow:

In joint mode each channel will issue single read and write bursts until it is possible to issue bursts that equal in length the BURST_MAX_SIZE register value. Then the channel will go into the joint phase, performing simultaneous read and write bursts. At the end of the buffer the channel will flush the last bytes by issuing smaller bursts again.

Remark: When using joint mode the data buffer can be minimal without causing a restriction on the joint burst length. But it will restrict the bursts at the initial and final stages of the channel (when joint bursts are not possible).

Restrictions:

Joint mode can not work with the following configurations:

- Peripherals
- Non aligned block transfer (works with normal non aligned transfer)
- When using a 16 bytes data buffer the read start address and the write start address must be aligned to the burst size. A 32 bytes data buffer or larger does not restrict joint mode.

Example using a 32 bit bus:

FIFO_SIZE = 32 bytes
RD_START_ADDR = 0x30000031
RD_BURST_MAX_SIZE = 64 bytes (affects WR_BURST_MAX_SIZE as well
WR_START_ADDR = 0x40000037
BUFFER_SIZE = 256 bytes

- Joint mode is not supported with peripherals.
- In order to support joint mode with block mode, the block width, frame width, read start address and write start address must be align to data bus width.

Read				**Write**		
Burst address	**Burst size**	**Buffer remain**		**Burst address**	**Burst size**	**Buffer remain**
0x30000039	1 (SINGLE)	256		0x400000BF	1 (SINGLE)	256
0x3000003A	2 (SINGLE)	255		Ready for joint – wait for read		
0x3000003C	4 (SINGLE)	253		Ready for joint – wait for read		
Going into joint mode				Going into joint mode		
0x30000040	64 (INCR16)	249	joint	0x400000C0	64 (INCR16)	255
0x30000080	64 (INCR16)	185	joint	0x40000100	64 (INCR16)	191
0x300000C0	64 (INCR16)	121	joint	0x40000140	64 (INCR16)	127
Going back to normal mode				Going back to normal mode		
0x30000100	32 (INCR8)	57		0x40000180	32 (INCR8)	63
0x30000120	16 (INCR4)	25		0x400001A0	16 (INCR4)	31
0x30000130	4 (SINGLE)	9		0x400001B0	4 (SINGLE)	15
0x30000134	4 (SINGLE)	5		0x400001B4	4 (SINGLE)	11
0x30000138	1 (SINGLE)	1		0x400001B8	4 (SINGLE)	7
-	-	-		0x400001BC	2 (SINGLE)	3
-	-	-		0x400001BE	1 (SINGLE)	1

Table 4: Burst flow example – joint mode

4. Concepts

4.1 DMA commands

A DMA command is constructed of four 32 bit fields. Each channel holds its current command in its CMD0-3 registers. The channel's command can be configured directly by the APB3 configuration bus or the channel can load the command by reading it from on the AHB bus. The fields of the DMA command are explained in the Register chapter.

4.2 DMA command lists - optional feature (Command lists)

DMA command lists are linked lists of DMA commands that can be placed anywhere in the memory space. When the channel completes its current command, if the CMD_LAST field is 0, the channel control will read the next command on the AHB bus from the address specified in the CMD_NEXT_ADDR field. If CMD_LAST is 1, the channel will stop.
The first command can be written directly to the channel's CMD registers, but it is better practice to write the entire command list to memory and set in the current command (CMD registers) an empty buffer that points to the beginning of the list.
The configuration sequence:

1. Write command list to memory
2. Set BUFFER_SIZE = 0
3. Set CMD_SET_INT = 0
4. Set CMD_LAST = 0
5. Set CMD_NEXT_ADDR = address of first command in memory.

Remark: The next command is read using an INCR4 burst, if the bus width is 64 bit the last 2 data strobes will be ignored.

Remark: CMD_NEXT_ADDR must be aligned to a command size (16 bytes for a 32 bit data bus, 32 bytes for a 64 bit data bus).

Command lists can be used for two purposes:

1. Scatter – Gather. When an operation system allocates large memory blocks the memory is continuous in the virtual address space but non-continuous in the physical address space. The list of address-size pairs of the allocation process can be written to memory as a list of DMA commands allowing the DMA to transfer all these chunks of data without CPU intervention.

   Example: A scatter list describing a 20KB memory block fragmented into five 4KB pages. After the last page is written the list indicated to issue a completion interrupt to the CPU.

Command's address in memory	Command number	RD START ADDR	WR START ADDR	BUFFER SIZE	CMD SET INT	CMD LAST	CMD NEXT ADDR
0x30000000	0 (first)	0x40001000	0x50001000	0x1000	0	0	0x30000010/4
0x30000010	1	0x40002000	0x50008000	0x1000	0	0	0x30000020/4
0x30000020	2	0x40003000	0x50015000	0x1000	0	0	0x30000030/4
0x30000030	3	0x40004000	0x50017000	0x1000	0	0	0x30000040/4
0x30000040	4 (last)	0x40005000	0x50025000	0x1000	1	1	0

Table 5: Scatter list example

2.Cyclic buffers for peripheral control. When servicing peripheral devices it is good practice to use at least double buffers to hold the peripheral RX or TX data. Cyclic buffers can be easily configured by setting a cyclic command list.

Example: A cyclic double buffer at address 0x30000000, services an RX peripheral client at address 0xBE000000, buffer sits at 0x50001000. Notice that the list is cyclic and endless, this is since most peripherals such as SPI, UART, audio devices, etc. need endless servicing.

Command's address in memory	Command number	RD START ADDR	WR START ADDR	BUFFER SIZE	CMD SET INT	CMD LAST	CMD NEXT ADDR
0x30000000	0	0xBE000000	0x50001000	0x1000	1	0	0x30000010/4
0x30000010	1	0xBE000000	0x50002000	0x1000	1	0	0x30000000/4

Table 6: Cyclic buffer example

4.3 Peripheral control - optional feature (Peripheral control)

DMA is a very easy and efficient way to service peripheral devices with minimum CPU intervention. This is specially beneficial when attending to slow devices. Usually peripherals are either RX or TX and hold a fixed width FIFO.
Usually each peripheral will have a specific channel that will service it exclusively.

Channel configuration for peripheral control (example is for an RX peripheral, TX peripheral is exactly the same just replace TX with RX and RD with WR):

1.Set the channel to issue bursts that match the functionality of its peripheral. Write the number of bytes in each burst in register RD_BURST_MAX_SIZE.
2.Set the number of the peripheral in register RD_PERIPH_NUM for RX. This corresponds to the bit number the peripheral is connected to on the periph_rx_req and periph_rx_clr buses.
3.Set the maximum number of consecutive bursts to be issued every time the peripheral is serviced. Write the number of bursts to the register RD_TOKENS.

4.Set the number of cycles to wait for the periph_rx_req to update after giving periph_rx_clr, this is dependent on peripheral latency (only has effect is RD_TOKENS is larger than 1). Write this number to the register RD_PERIPH_DELAY.

4.4 Peripheral to peripheral transfer - optional feature (Peripheral control)

Peripheral to peripheral transfer is possible and is configured as normal peripheral transfer, setting both read and write peripheral registers. There is no restriction regarding different burst sizes for the RX and TX peripherals.

4.5 Arbitration – Build can remove high and top priority modes (Priority modes)

Arbitration takes place separately in each core independently for read and for write. Generally channels work in a round robin order. They are three levels of priorities; normal, high and top. Each channel has a separate priority level for read and for write. This is set in the RD_PRIO and WR_PRIO registers. Normal priority channels work in round robin order, high priority channels are granted every other slot and top priority channels work as long as they request the bus.

Remark: It is not recommended to use top priority with joint mode since the top priority channel can starve the other channels.

CH0	CH1	CH2	CH3	CH0	CH1	CH2	CH3	CH0	CH1	CH2	CH3

Table 7: Normal priority example

CH0	CH3	CH1	CH3	CH2	CH3	CH3	CH3	CH0	CH3	CH1	CH3

Table 8: High priority example

CH3	CH3	CH3	CH3	CH3	CH3	CH0	CH1	CH2	CH0	CH1	CH2

Table 9: Top priority example

4.6 Tokens (Windowed arbitration) - optional feature (Tokens)

Every time a channel starts working it will transfer a maximum number of bursts according to the value in the RD_TOKENS or WR_TOKENS registers. Using many tokens improves the channel's overall performance but can result in lengthening other channels latency.

4.7 Block transfer - optional feature (Block support)

Some applications might require to read or write blocks in a frame context. A channel transfers blocks if its BLOCK register is set. The block size is set in the X_SIZE and Y_SIZE registers. The frame width is set in the FRAME_WIDTH register. In this case the START_ADDR registers refer to the upper left corner of the block. Each DMA command refers to one block. It is recommended that the TOKENS register will be set to the number of lines of the block in order to allow transferring the entire block once the channel started operating.

4.8 Block scatter gather - optional feature (Block support)

Some applications, will find it very efficient to use block command lists (block scatter gather). This is a normal command list, when each command refers to a single block. It is recommended to set the TOKENS register to the maximum Y_SIZE expected, so block transfer will not stop in the middle.

4.9 Peripheral block transfer - optional feature (Block support & Peripheral control)

Peripherals can transfer block in two modes, normal peripheral mode and block peripheral mode. This is configured in the PERIPH_BLOCK register. When working in peripheral block mode, the periph_clr signal is given at the end of the entire block, in normal peripheral mode, it is given, normally, after each burst. In both modes the BURST_MAX_SIZE must be equal to X_SIZE (block width), resulting in transferring a single block line in each burst.

4.10 Scheduled channels - optional feature (Scheduler)

It is possible for low priority channels to clog the bus activity, especially low priority memory copy channels. To prevent this, it is possible to slow down such channels by setting a number of cycles in which they will not request the bus after they operate. This period can be set in the RD_WAIT_LIMIT and WR_WAIT_LIMIT registers for adding a delay after read or after write.

4.11 Interrupt depth - optional feature (Command lists)

It is normal for processors to take a long time before handling interrupts. In such cases a channel can complete a number of commands before the processor had a chance to handle the channel's end interrupt. In order to support this the number of unserviced interrupts is kept in the INT_COUNT register. When clearing an end interrupt, if they are pending interrupts (INT_COUNT > 1), the end interrupt will be reissued.

4.12 Software control of peripheral request - optional feature (Peripheral control)

A channel can be controlled directly by software by using the peripheral control mechanism.

Example: Using this feature to access configuration registers off-line.

Writing to configuration registers:
1.An unused peripheral TX bit is set to service the channel.
2.The CPU writes the configuration data to the memory space, preferably to a fast memory close to it.
3.The DMA loads the data to the channel's data buffer but does not write it out.
4.When the CPU wishes the actual configuration to take place it sets the appropriate bit in the PERIPH_TX_CTRL register.
5.The DMA writes out the data, clearing PERIPH_TX_CTRL.

Reading from status registers:
1.An unused peripheral RX bit is set to service the channel.
2.When the status registers are ready to be read the CPU sets the appropriate bit in the PERIPH_RX_CTRL register.
3.The DMA copies to status registers to a desired address, clearing PERIPH_RX_CTRL.
4.The CPU can access the status registers when it finds the time.

4.13 Multiple processor control - optional feature (Number of interrupts)

In case the system contains more than 1 processor it might be productive to connect all processors to the DMA, allowing different processors to control different channels simultaneously. Each processor's interrupt should be connected to a different bit of the INT output bus and each channel's INT_NUM register should be configured on which interrupt bit to output its interrupts. Up to 8 processors can be connected to the DMA.

4.14 AHB timeouts - optional feature (AHB timeout)

Timeouts can be issued on all five AHB buses. In case any of the buses does not respond in 1024 cycles a timeout interrupt will be issued by the corresponding channel. In this way not only the interrupt indicates that the slave is not responding but also indicates which channel issued the request.

4.15 Watchdog timer - optional feature (Watchdog timer)

A watchdog timer is present in each core. The watchdog timer checks each active channel (enabled and not ended) in a round robin order. If the checked channel does not start working in 2048 cycles a timeout interrupt will be issued by the checked channel.

4.16 Clock gating - optional feature (Clock gating)

Clock gates are inserted into the design to reduce dynamic power. The design has the following clock gates:

•Each core's clock stops when all of its channels have ended

•The general configuration register block's clock stops when not accessing its registers.

•Each channel's configuration register block's clock stops when not accessing its registers.

4.17 Multiple output port control - optional feature (Interconnect)

Each channel can control up to two output ports. In each channel the read port is configured in the RD_PORT_NUM register, the write port in configured in the WR_PORT_NUM register. The channel's commands can be read from a different port than the read data port, the port to read the commands from is configured in the RD_CMD_PORT_NUM.

4.18 Core 1 clock divider

Slow channels can be gathered in core 1 and a clock divider can be added to slow down the core's clock. The APB clock will not be slowed down since the APB bus is joined with core 0 and adding wait states on core 0 APB bus is not desired.
An AHB synchronizer will be added on core 1 AHB bus so that the output bus will work at the same frequency as the input clock. In order not to insert wait states on the output AHB bus a read data buffer and a write data buffer are present in the synchronizer. The data buffers are at the same size as the data buffers of the core's channels. These buffers allow core 1 to read and write burst at full speed, one burst at a time. This is why when working with a clock divider core 1 write command depth should be set to 1.
When using a clock divider core 1 can not use 'Joint mode'.

4.19 AHB port mux

Each core simultaneously controls a read AHB port and a write AHB port, this allows a channel to read and write at the same time. In case the AHB slave does not support simultaneous read and write it is possible to add an AHB mux on its port. This will add the HWRITE port in order to indicate whether the read or write bus is operating. Obviously, such a mux significantly reduces bus performance.

4.20 Endianness byte swapping - optional feature (Endianness)

Each channel can manipulate the data written out in order to support little / big endian ports. Byte swapping is configured in the END_SWAP register. Supports byte swapping within 16, 32 or 64 bit data.
Restrictions: The following parameters must be aligned to the byte swapping size: RD_START_ADDR, WR_START_ADDR, BUFFER_SIZE or X_SIZE for block mode, FRAME_WIDTH. Example: if END_SWAP is set to 1 (swap within 16 bits) the parameters above must be aligned to 16 bits.

5. *Configuration Flows*

5.1 General configuration

1.If core 1 is present with a clock divider, initially core 1 clock ratio should be set using the CORE1_CLKDIV_RATIO register. The divided clock controls the entire core except its configuration registers that keep working on the main clock, this is to prevent long wait states on the APB bus and to support APB backward compatibility.

2.Set each core to work in 'Independent mode' or 'Joint mode' using the CORE0_JOINT_MODE and CORE1_JOINT_MODE registers.

5.2 Configure and start a channel

All channels are completely independent and use no shared configuration registers. The channel configuration is basically made out of two parts, the static configuration and the command. The static configuration holds the information that does not change during the life of an application, the command holds the current activity of the channel.

Generally, setting up a channel is made out of:

1.Configure static registers
2.Configure interrupt controller (all interrupts are enabled by default)
3.Configure command or command list
4.Enable the channel (all channels are enabled by default)
5.Start the channel

5.3 Stop a channel

A channel will work until it completes its last command and then will stop by itself. After stopping the CH_RD_ACTIVE and CH_WR_ACTIVE will both be 0.
A channel can be stopped by clearing the CH_ENABLE register, later on the channel can be resumed by setting it.

5.4 Pause and resume a channel

A channel can be paused by clearing the CH_ENABLE register, the channel can be resumed by resetting it.

5.5 Restart a channel

In order to restart a channel the flowing sequence should be done:

- Stop the channel by clearing the CH_ENABLE register.
- Read the CMD_OUTS_REG register until it is 0.
- Restart the channel by setting CH_START.
- Restart the channel by setting CH_ENABLE.

5.6 Interrupt handling

When a processor receives an interrupt it should already know on which bit of the INT bus it is connected.
The following actions should be performed:

1. Read the correct INTX_STATUS register to figure out which channel caused the interrupt.
2. Read the channel's INT_STATUS_REG to figure out which interrupt to handle.
3. Do what should be done.
4. Clear the interrupt by writing to the corresponding INT_CLEAR_REG.

5.7 Power down sequence

In order to power down the DMA:

1. Clear the CH_ENABLE register in all active channels (this will stop the channel at the completion of the current pending transactions) or set the CMD_LAST register in all active channels (this will stop the channel at the completion of the current buffer).
2. Wait for the idle pin to set or wait until the IDLE register is set.
3. DMA is ready for power down.

6. *Performance*

6.1 General

Constructing the design under different configurations will result in different performance, basically larger buffers will result in smaller burst to burst latency. Better performance can also be achieved by different register configurations.

- Generally, best performance is achieved using joint mode.
- Both latency and throughput is improved using outstanding mode.
- When multiple channels operate overall throughput can be higher by channel interleaving.
- Smaller bursts have smaller burst to burst latency.

6.2 Independent mode, 64 bit data bus

Example: A single channel transfers 1024 bytes, data bus is 64 bit. The following screen-shots show AHB bus activity using different data buffers.

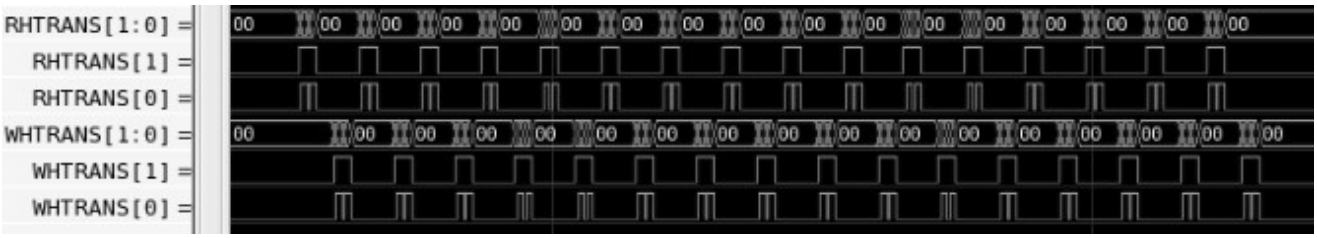

Illustration 4: Bus activity, single channel, 64 bit, buffer size 32 bytes

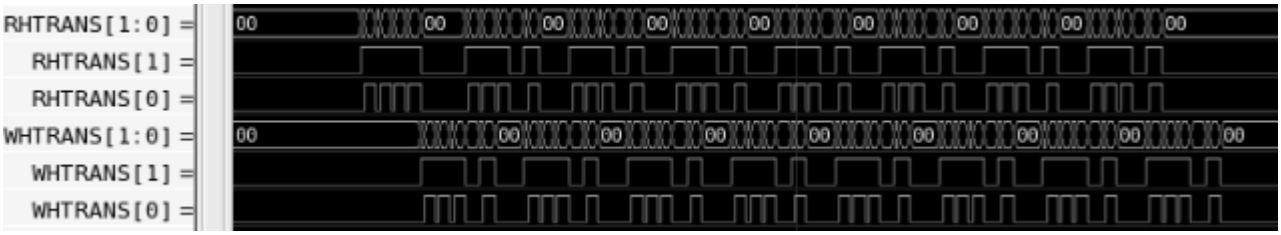

Illustration 5: Bus activity, single channel, 64 bit, buffer size 128 bytes

6.3 Independent mode, 32 bit data bus

Since data transfer is twice slower when using a 32 bit bus the data buffer size needed to achieve maximum throughput is half the size needed when using a 64 bit data bus.

Example: A single channel transfers 1024 bytes, data bus is 32 bit. The following screen-shots show AHB bus activity using different data buffers.

Illustration 7: Bus activity, single channel, 32 bit, buffer size 32 bytes

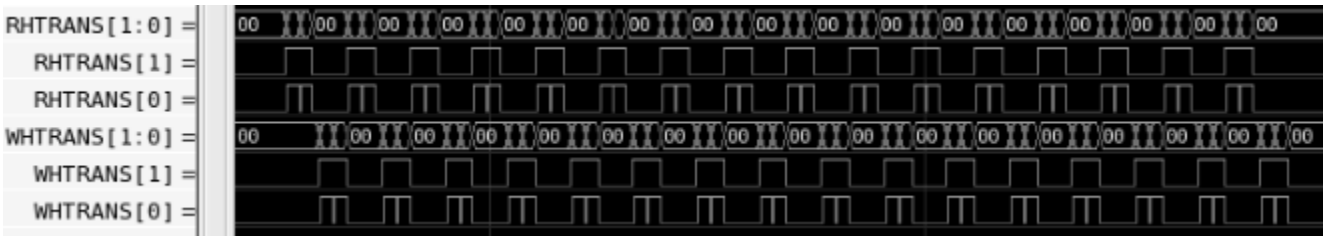

Illustration 8: Bus activity, single channel, 32 bit, buffer size 64 bytes

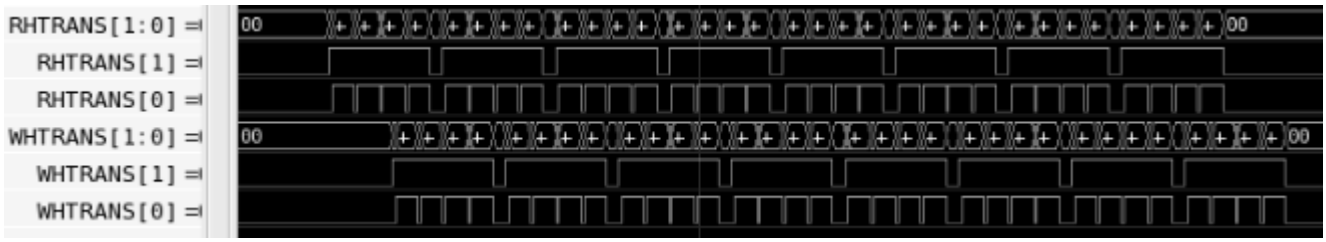

Illustration 9: Bus activity, single channel, 32 bit, buffer size 128 bytes

6.4 Independent mode – outstanding channel mode

Both bus latency and throughput are improved using outstanding requests, but these can only be used if the slave is quick enough to operate in this mode, otherwise erogenous result are expected.

Example: A single channel transfers 1024 bytes, data bus is 32 bit, buffer size is 64 bytes. The following screen-shots show AHB bus activity using different outstanding configurations.

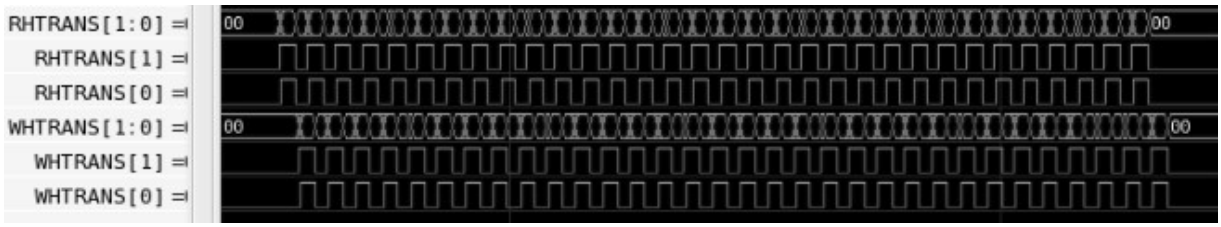

Illustration 10: Bus activity, single channel, 32 bit, buffer size 32 bytes, read outstanding

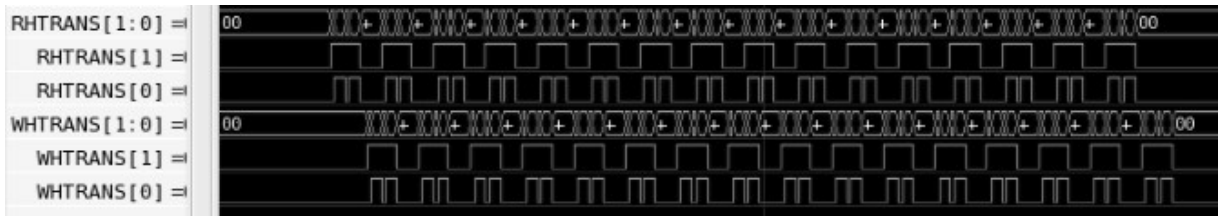

Illustration 11: Bus activity, single channel, 32 bit, buffer size 32 bytes, write outstanding

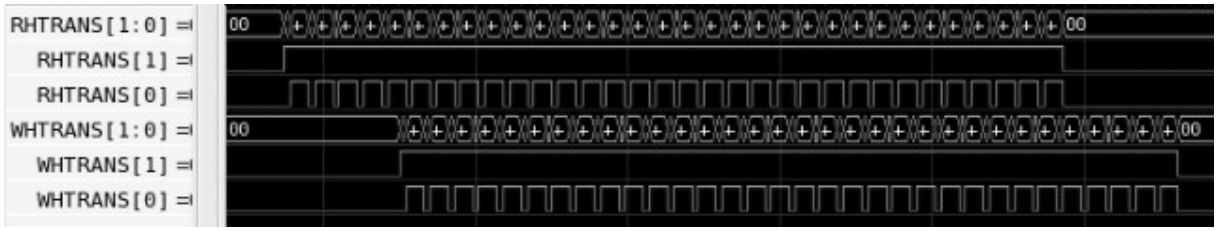

Illustration 12: Bus activity, single channel, 32 bit, buffer size 32 bytes, read and write outstanding

Notice the improvement in bus latency.

6.5 Independent mode - multiple channels

Example: 8 channels transferring each 1024 bytes, data bus is 64 bit, buffer size is 32 bytes

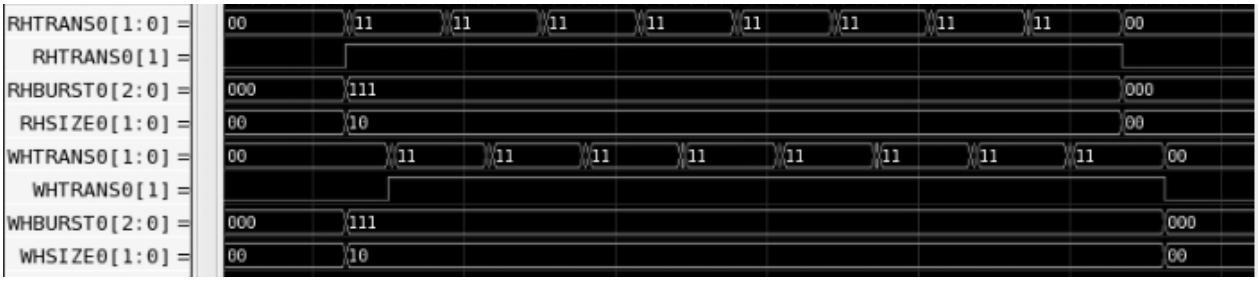

Illustration 13: Bus activity, multiple channels, 64 bit, buffer size 64 bytes

Sine the channels interleave, maximum throughput is reached although using a small data buffer.

6.6 Independent mode - dual cores with a shared AHB bus

When using dual cores with a shared AHB bus (AHB matrix) the bus operation will round robin between the cores.

Example: Dual cores, 1 channel per core. M0 is core 0 AHB bus, M1 is core 1 AHB bus and S0 is the output of the interconnect. The following screen-shots show AHB bus activity using different clock ratio.

Illustration 14: Bus activity, dual cores, single channel per core, shared clock

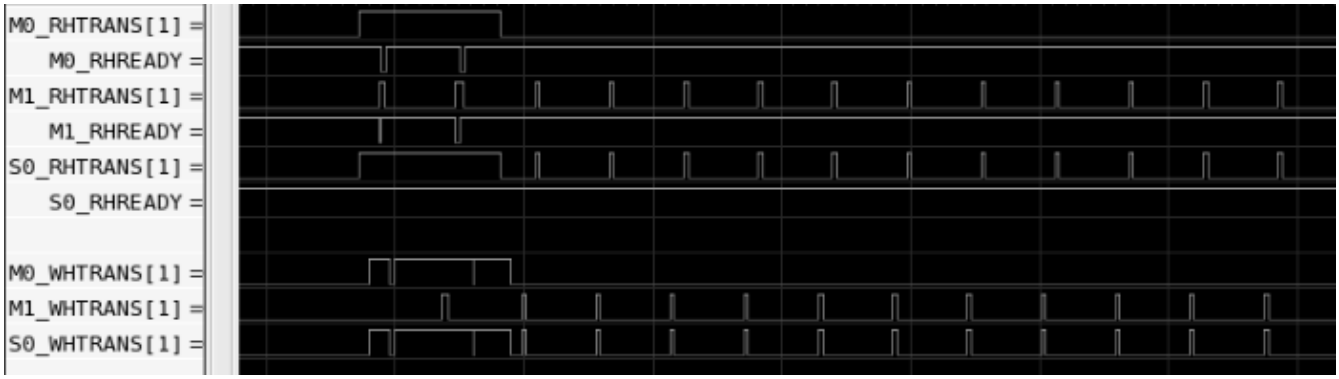

Illustration 15: Bus activity, dual cores, single channel per core, clock ratio 4

6.7 Independent mode - block transfer

When working in block mode, there will be a two cycle gap on both data buses between block lines. It is recommended to allow the entire block to be transferred in a single operation of the channel using the TOKENS register.

Example: Single channel, block mode, single block 32x8.

Illustration 16: Illustration 19: Bus activity, block mode, single channel, single block 32x8

Example: Two channels, block mode, each channel transfers a single block 32x8.

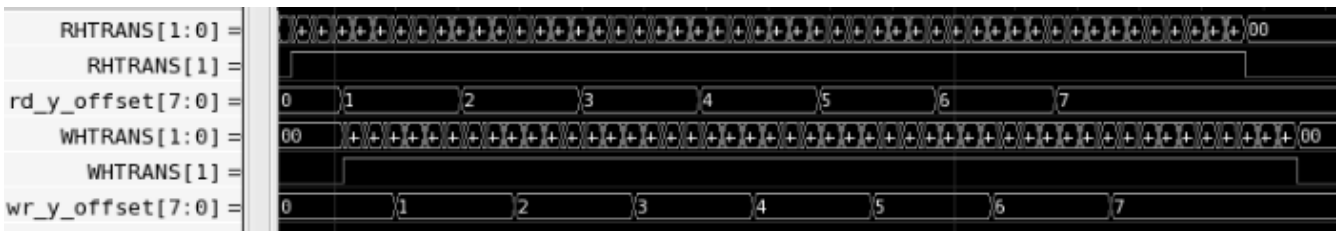

Illustration 17: Bus activity, block mode, two channels, single block 32x8

6.8 Joint mode

Joint mode is the most efficient regarding throughput, latency and the minimal data buffer possible for achieving maximum throughput.

Example: Single channel, data buffer is 32 bytes, joint bursts are 32 bursts, read and write addresses are not aligned.

Illustration 18: Bus activity, joint mode, single channel, unaligned addresses

Example: Four channels, data buffer is 32 bytes, joint bursts are 64 bursts, read and write addresses are aligned.

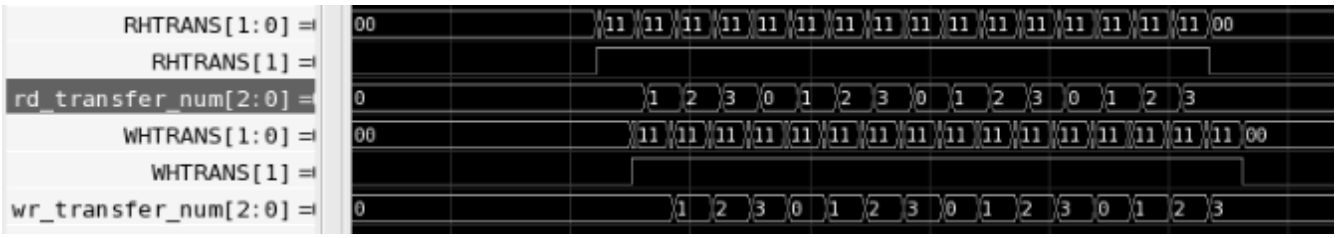

Illustration 19: Bus activity, joint mode, multiple channels, minimal buffer size

Notice that although a 32 byte data buffer is used the burst size is not restricted to the data buffer size.

6.9 Conclusion

The following configurations support back-to-back continuous bus activity (100% throughput):

Channels	Operation mode	Bus width	FIFO size
1	Independent	32 bits	128 bytes
1	Independent	64 bits	256 bytes
1	Outstanding	32 bits	64 bytes
1	Outstanding	64 bits	128 bytes
1 to 8	Joint	32 bits	32 bytes
1 to 8	Joint	64 bits	32 bytes
6 to 8	Independent	32 bits	32 bytes

Table 10: Performance conclusion

7. *Area and frequency examples*

The gate count of the design differs by the build configuration, this chapter presents the expected gate count for different typical builds.

7.1 Single core design

	Design 0	Design 1	Design 2	Design 3	Design 4	Design 5
Channel number	1	1	1	2	8	8
Address restrictions	None	None	None	None	None	Burst aligned
Mode	Independent	Joint	Joint	Joint	Joint	Joint
Data width	32 bits	32 bits	64 bits	32 bits	32 bits	32 bits
Data buffer size	16 bytes	32 bytes	32 bytes	32 bytes	32 bytes	32 bytes
Address bits	16	16	20	20	20	20
Buffer bits	9	9	11	11	10	10
AHB matrix	None	None	None	Dual port	None	None
AHB mux	None	None	Port 0	None	None	None
Block support	Yes	No	No	No	No	No
Watchdog timer	No	No	Yes	Yes	Yes	Yes
AHB timeout	No	No	Yes	Yes	Yes	Yes
Priority modes	No	No	No	No	Yes	Yes
Tokens	No	No	No	No	Yes	Yes
Gate count	**14000**	**16000**	**23000**	**28000**	**91000**	**78000**
Frequency @TSMC90	**440 MHz**	**440MHz**	**420 MHz**	**420 MHz**	**400 MHz**	**400 MHz**

Table 11: Single core gate count

Remark: All designs were synthesized with half a cycle input and output delays.

7.2 Dual core design

	Design 0	
	Core 0	**Core 1**
Channel number	2	4
Address restrictions	None	None
Mode	Joint	Independent
Data width	64 bits	32 bits
Data buffer size	32 bytes	16 bytes
Address bits	20	16
Buffer bits	10	9
AHB matrix	Dual port	
AHB mux	Port 1	
Block support	No	No
Watchdog timer	Yes	Yes
AHB timeout	Yes	Yes
Priority modes	No	No
Tokens	No	No
Clock divider	No	Yes
Gate count	**83000**	
Frequency @TSMC90	**390 MHz**	

Table 12: Dual core gate count

Remark: All designs were synthesized with half a cycle input and output delays.

8. Bus activity

8.1 AHB bursts used

- AHB bursts used: INCR16, INCR8, INCR4, SINGLE byte, SINGLE halfword, SINGLE word, SINGLE double word (64 bit data bus).
- BUSY on HTRANS will only be used in joint mode when HREADY goes low in the middle of a joint burst.

8.2 Error interrupts

All interrupts will be set in the channel that issued the erroneous burst, in this way finding out the cause of the error is almost immediate. They are 3 error types:

- An error has been received on AHB bus (HRESP).
- Timeout errors. A timeout interrupt is issued if the slave did not respond for 1024 cycles, the timeout value is non-configurable.
- Underflow-overflow errors. These interrupts are used to indicate that the slave is not working fast enough to support outstanding mode. In case of such an interrupt the system configuration should be altered (for example raising the channel's priority level or increasing its tokens), or the outstanding mode should be turned off.
- Watchdog timeout. An interrupt is issued if an active channel did not start a burst in 1024 cycles, the timeout value is non-configurable.

All the interrupts are described in the *registers* chapter.

9. Registers

9.1 General

Configuration registers are accessed using the standard APB3 control bus.

Systems that use older APB control bus can simply not connect PREADY and PSLVERR. All registers could still be accessed since they are all have one cycle latency. Core 1 will return read data after one cycle even if it uses a divided clock, this is because its configuration registers use the main clock.

APB slave error (PSLVERR) will be given on the following events:

- Accessing non mapped addresses
- Accessing a non existing core
- Accessing a non existing channel
- Writing to read only registers
- Reading from write only registers

9.2 Channel registers

Constructing channel registers addresses

All channels have the exact same register set, in order to access a specific register the address is constructed in the following manner:

(Register address) = (Core base address) + (Channel base address) + (register offset)

Core 0 base address is 0x0

Core 1 base address is 0x800

Channel base address = Channel number << 8 (Channel number * 256)

•CMD_REG0

Offset:	0x00
Read/Write:	R/W
Description:	Channel's command, first line out of 4.When using command lists this register will be overwritten by the next command.
Fields:	

➢**RD_START_ADDR [31:0]**

Start address of read buffer. This address will be read on AHB bus. Value is in bytes and has no alignment restrictions. If the address is not aligned to the channel's FIFO size, the maximum burst size will be restricted to half of the FIFO size, instead of being restricted to the FIFO size.

Default value is 0.

•CMD_REG1

Offset: 0x04

Read/Write: R/W

Description: Channel's command, second line out of 4.When using command lists this register will be overwritten by the next command.

Fields:

➢**WR_START_ADDR [31:0]**

Start address of write buffer. This address will be written on AHB bus. Value is in bytes and has no alignment restrictions. If the address is not aligned to the channel's FIFO size, the maximum burst size will be restricted to half of the FIFO size, instead of being restricted to the FIFO size.

Default value is 0.

•CMD_REG2

Offset: 0x08

Read/Write: R/W

Description: Channel's command, third line out of 4.When using command lists this register will be overwritten by the next command.

Remark: Buffer size bits can be reduced in build down to 9 bits.

Fields:

If not in block mode (according to BLOCK register):

➢**BUFFER_SIZE [15:0]**

Size of buffer to transfer. Value is in bytes and has no size restrictions.

Default value is 0.

If in block mode (according to BLOCK register):

➢**X_SIZE [7:0]**

Block width. Value is in bytes and has no size restrictions.

Default value is 0.

➢**Y_SIZE [15:8]**

Number of block lines.

Default value is 0.

•CMD_REG3

Offset: 0x0C

Read/Write: R/W

Description: Channel's command, last line out of 4. When using command lists this register will be overwritten by the next command.

Fields:

➢**CMD_SET_INT [0]**

If set the channel will issue an interrupt once the entire buffer has been transfered.

Default value is 0.

➢**CMD_LAST [1]**

If set the channel will stop once the entire buffer has been transferred, if not the next command will be loaded from the address specified in the CMD_NEXT_ADDR field.

Default value is 0.

➢**CMD_NEXT_ADDR [31:4]**

Address of next command. Value is in chunks of 16 bytes (command size). The next command will be read on the AHB bus in case the CMD_LAST field is not set.

Default value is 0.

•STATIC_REG0

Offset: 0x10

Read/Write: R/W

Description: Channel's static configuration. These parameters should not be changed while channel is active. These registers are used for both are and write in joint mode.

Fields:

➢**RD_BURST_MAX_SIZE [9:0]**

Maximum number of bytes of an AHB read burst. Possible values: 1, 2, 4, data_width*N (N=4,8,16). If the channel is reading from a peripheral, the number is set according to the peripherals FIFO.

When working with a peripheral in block mode, the value must match X_SIZE (block width).

Default value is 0.

➢**RD_ALLOW_FULL_BURST [12]**

Status register, indicates if burst size can exceed data buffer size (joint mode).

Reset value is 0.

➢**RD_ALLOW_FULL_FIFO [13]**

Status register, indicates if burst can use entire data buffer. It is allowed when both read and write start addresses are aligned to data buffer size, otherwise the maximum allowed burst is half of the buffer size.

Reset value is 0.

➢**RD_TOKENS [21:16]**

Number of AHB read commands to issue before the channel is released.

Default value is 1.

➢**RD_OUTSTANDING [30]**

Caution, might cause erroneous results!

If set it allows the controller to issue the AHB read command while the FIFO is full, expecting the data to be outputted before the read data arrives. If this does not happen the FIFO will be overflown, data lost, and an OVERFLOW interrupt will be issued.

Default value is 0.

➢**RD_INCR [31]**

If set the controller will increment the next burst address. Should be set for all memory copy channels. Should be cleared for all peripheral clients that use a static address FIFO.

Default value is 1.

•**STATIC_REG1**

Offset: 0x14

Read/Write: R/W

Description: Channel's static configuration. These parameters should not be changed while channel is active. These registers are not used in joint mode.

Fields:

➢**WR_BURST_MAX_SIZE [9:0]**

Maximum number of bytes of an AHB write burst. Possible values: 1, 2, 4, data_width*N (N=4.8.16). If the channel is writing to a peripheral, the number is set according to the peripherals FIFO. When working with a peripheral in block mode, the value must match X_SIZE (block width).

Default value is 0.

➢**WR_ALLOW_FULL_BURST [12]**

Status register, indicates if burst size can exceed data buffer size (joint mode).

Reset value is 0.

➢**WR_ALLOW_FULL_FIFO [13]**

Status register, indicates if burst can use entire data buffer. It is allowed when both read and write start addresses are aligned to data buffer size, otherwise the maximum allowed burst is half of the buffer size.

Reset value is 0.

➢**WR_TOKENS [21:16]**

Number of AHB write commands to issue before the channel is released.

Default value is 1.

➢**WR_OUTSTANDING [30]**

Caution, might cause erroneous results!

If set it allows the controller to issue the AHB write command immediately after the AHB read command has been given, before the read data actually arrived. This is effective for high priority memory copies, especially if the write AHB slave can benefit by an early command. If the write data is outputted before the read data arrives, an UNDERFLOW interrupt will be issued.

Default value is 1.

➢**WR_INCR [31]**

If set the controller will increment the next burst address. Should be set for all memory copy channels. Should be cleared for all peripheral clients that use a static address FIFO.

Default value is 1.

•**STATIC_REG2**

Offset: 0x18

Read/Write: R/W

Description: Block mode

Fields:

➢**FRAME_WIDTH [11:0]**

Frame width for block mode. Value is in bytes and has no restrictions.

Default value is 0.

➢**BLOCK [15]**

If set the channel will work in block mode.

Default value is 0.

➢**JOINT [16]**

If set the channel will work in joint mode, has effect only if COREx_JOINT_MODE is set.

Default value is 0.

➢**AUTO_RETRY [17]**

If set the channel will automatically restart if an underflow or overflow error occurs.

Default value is 0.

➢**RD_CMD_PORT_NUM [20]**

Number of AHB output port to read commands from.

Default value is 0.

➢**RD_PORT_NUM [21]**

Number of AHB output port to read data from.

Default value is 0.

- **WR_PORT_NUM [22]**

  Number of AHB output port to write data to.

  Default value is 0.

- **INT_NUM[26:24]**

  Number of interrupt bit to use. Can be used when different channels are used by different processors. Each processor can be connected to a different interrupt bit.

  Default value is 0.

- **END_SWAP[29:28]**

  Endianness byte swapping. Notice byte swapping restrictions (section 4.20).

  0 – No swapping.

  1 – Swap bytes within 16 bits.

  2 – Swap bytes within 32 bits.

  3 – Swap bytes within 64 bits.

  Default value is 0.

•STATIC_REG3

Offset: 0x1C

Read/Write: R/W

Description: Channel's static configuration. These parameters should not be changed while channel is active.

Fields:

- **RD_WAIT_LIMIT[11:0]**

  Minimum number of cycles to wait after a channel is released and before it starts issuing more read commands. This helps prevent low priority channels clog the AHB bus. Value must be a multiplication of 16. Controls also WR_WAIT_LIMIT in joint mode.

  Default value is 0.

- **WR_WAIT_LIMIT[27:16]**

  Minimum number of cycles to wait after a channel is released and before it starts issuing more write commands. This helps prevent low priority channels clog the AHB bus. Value must be a multiplication of 16. Is not used in joint mode.

  Default value is 0.

•STATIC_REG4

Offset: 0x20

Read/Write: R/W

Description: Channel's static configuration. These parameters should not be changed while channel is active.

Fields:

- **RD_PERIPH_NUM[4:0]**

  Number of peripheral to read from. Set 0 if the channel reads from a memory, or reads from a peripheral that does not use peripheral flow control.

  Default value is 0.

- **RD_PERIPH_DELAY[10:8]**

  Number of cycles to wait for the peripheral read request signal to update after issuing the read clear signal. This is determined by the peripheral latency.

  Default value is 0.

- **RD_PERIPH_BLOCK[15]**

  If set the peripheral control works in block mode. In block mode, the periph_rx_clr signal will be given only at the end of the block, otherwise it will be given normally, after every burst (every block line).

  Default value is 0.

- **WR_PERIPH_NUM[20:16]**

  Number of peripheral to write to. Set 0 if the channel writes to a memory, or writes to a peripheral that does not use peripheral flow control.

  Default value is 0.

- **WR_PERIPH_DELAY[26:24]**

  Number of cycles to wait for the peripheral write request signal to update after issuing the write clear signal. This is determined by the peripheral latency.

  Default value is 0.

- **WR_PERIPH_BLOCK[31]**

  If set the peripheral control works in block mode. In block mode, the periph_tx_clr signal will be given only at the end of the block, otherwise it will be given normally, after every burst (every block line).

  Default value is 0.

•RESTRICT_REG

Offset: 0x2C

Read/Write: R

Description: Channel's restrictions status register

Fields:

- **RD_ALLOW_FULL_FIFO[0]**

  Read start address does not restrict burst size.

➢**WR_ALLOW_FULL_FIFO[1]**

Write start address does not restrict burst size.

➢**ALLOW_FULL_FIFO[2]**

Burst sizes can equal data buffer size, otherwise the maximum burst is half of the data buffer size.

➢**ALLOW_FULL_BURST[3]**

Maximum burst of 16 strobes can be used (joint mode only).

➢**ALLOW_JOINT_BURST[4]**

Joint bursts are currently active.

➢**RD_OUTSTANDING_STAT[5]**

Read outstanding is currently active.

➢**WR_OUTSTANDING_STAT[6]**

Write outstanding is currently active.

➢**BLOCK_NON_ALIGN_STAT[7]**

Block configuration is not aligned, either start address, x_size or frame width.

➢**SIMPLE_STAT[8]**

Configuration is aligned and peripherals are not used.

•**READ_OFFSET_REG**

Offset: 0x30

Read/Write: R

Description: Channel's read offset status register.

Fields:

If not in block mode (according to BLOCK register):

➢**RD_OFFSET[15:0]**

Offset from the beginning of the buffer. Value is in bytes.

Reset value is 0.

If in block mode (according to BLOCK register):

➢**RD_X_OFFSET[7:0]**

Offset from the beginning of the current block line. Value is in bytes.

Reset value is 0.

➢**RD_Y_OFFSET[23:16]**

Number of current block line.

Reset value is 1.

•WRITE_OFFSET_REG

Offset: 0x34

Read/Write: R

Description: Channel's write offset status register.

Fields:

If not in block mode (according to BLOCK register):

➢**WR_OFFSET[15:0]**

Offset from the beginning of the buffer. Value is in bytes.

Reset value is 0.

If in block mode (according to BLOCK register):

➢**WR_X_OFFSET[7:0]**

Offset from the beginning of the current block line. Value is in bytes.

Reset value is 0.

➢**WR_Y_OFFSET[23:16]**

Number of current block line.

Reset value is 1.

•FIFO_FULLNESS_REG

Offset: 0x38

Read/Write: R

Description: FIFO fullness status register.

Fields:

➢**RD_GAP[9:0]**

Remaining space in channel's FIFO for read data. Value is in bytes. Value is decremented when read command is issued and incremented according to the value of the RD_OUTSTANDING register. If RD_OUTSTANDING is set the value is incremented after issuing the write command otherwise it is incremented after data is written.

Reset value is equal to FIFO size.

➢**WR_FULLNESS[25:16]**

Occupied space in channel's FIFO by write data. Value is in bytes. Value is decremented when write command is issued and incremented according to the value of the WR_OUTSTANDING register. If WR_OUTSTANDING is set the value is incremented after issuing the read command otherwise it is incremented after data is

read.

Reset value is 0.

•CH_ENABLE_REG

Offset: 0x40

Read/Write: R/W

Description: Channel enable.

Fields:

➢**CH_ENABLE[0]**

Channel enable. Part of the initialization sequence. Also used for pause and resume.

Default value is 1.

•CH_START_REG

Offset: 0x44

Read/Write: W

Description: Channel start.

Fields:

➢**CH_START[0]**

Channel start. Part of the initialization sequence.

•CH_ACTIVE_REG

Offset: 0x48

Read/Write: R

Description: Channel active status register.

Fields:

➢**CH_RD_ACTIVE[0]**

This value is set if channel is enabled and all read data has been received.

Reset value is 0

➢**CH_WR_ACTIVE[0]**

This value is set if channel is enabled and all write data has been transfered.

Reset value is 0

•COUNT_REG

Offset: 0x50

Read/Write: R

Description: Buffer counter status register.

Fields:

- **BUFF_COUNT[15:0]**

  Number of buffers transferred by channel since started. When using a command list this status indicates how many DMA commands have been completed.

  Reset value is 0

- **INT_COUNT[5:0]**

  Number of unserviced end interrupts. Value is incremented each time an end interrupt is issued and is decremented when the INT_CLR_CH_END register is written.

  Reset value is 0

•INT_RAWSTAT_REG

Offset: 0xA0

Read/Write: R/W

Description: Interrupt raw status

Fields:

- **INT_RAWSTAT_CH_END[0]**

  Indicates an unserviced channel end interrupt. The total number of unserviced end interrupts can be read in the INT_COUNT register.

  Interrupt can be issued by writing to this field.

  Default value is 0.

- **INT_RAWSTAT_RD_SLVERR[1]**

  Indicates that a read issued by this channel caused an AHB read slave error.

  Interrupt can be issued by writing to this field.

  Default value is 0.

- **INT_RAWSTAT_WR_SLVERR[2]**

  Indicates that a write issued by this channel caused an AHB read slave error.

  Interrupt can be issued by writing to this field.

  Default value is 0.

- **INT_RAWSTAT_OVERFLOW[3]**

  Indicates that the data FIFO has been overflown, this can only occur when RD_OUTSTANING is set, indicating the slave was not fast enough to operate under this mode.

Interrupt can be issued by writing to this field.

Default value is 0.

- **INT_RAWSTAT_UNDERFLOW[4]**

  Indicates that the data FIFO has been underflown, this can only occur when WR_OUTSTANING is set, indicating the slave was not fast enough to operate under this mode.

  Interrupt can be issued by writing to this field.

  Default value is 0.

- **INT_RAWSTAT_TIMEOUT_RD[5]**

  Indicates that a read issued by this channel caused a timeout on the AHB bus. Timeout value is fixed to 1024 cycles.

  Interrupt can be issued by writing to this field.

  Default value is 0.

- **INT_RAWSTAT_TIMEOUT_WR[6]**

  Indicates that a write issued by this channel caused a timeout on the AHB bus. Timeout value is fixed to 1024 cycles.

  Interrupt can be issued by writing to this field.

  Default value is 0.

- **INT_RAWSTAT_WDT[7]**

  Indicates that the channel is active but did not start a burst for 2048 cycles.

  Interrupt can be issued by writing to this field.

  Default value is 0.

•INT_CLEAR_REG

Offset: 0xA4

Read/Write: W

Description: Interrupt clear

Fields:

- **INT_CLR_CH_END[0]**

  Clear channel end interrupt. Decrements INT_COUNT register.

- **INT_CLR_RD_SLVERR[1]**

  Clears INT_RAWSTAT_RD_SLVERR.

- **INT_CLR_WR_SLVERR[2]**

  Clears INT_RAWSTAT_WR_SLVERR.

- **INT_CLR_OVERFLOW[3]**

  Clears INT_RAWSTAT_OVERFLOW.

➢**INT_CLR_OVERFLOW[4]**

Clears INT_RAWSTAT_UNDERFLOW.

➢**INT_CLR_TIMEOUT_RD[5]**

Clears INT_RAWSTAT_TIMEOUT_RD.

➢**INT_CLR_TIMEOUT_WR[6]**

Clears INT_RAWSTAT_TIMEOUT_WR.

➢**INT_CLR_WDT[7]**

Clears INT_RAWSTAT_WDT.

•**INT_ENABLE_REG**

Offset: 0xA8

Read/Write: R/W

Description: Interrupt enable. Each bit that is set enables its corresponding INT_RAWSTAT register to be present in the INT_STATUS register and outputted on the INT output pin.

Fields:

➢**INT_ENABLE_CH_END[0]**

Enables INT_RAWSTAT_RD_DECERR.

Default value is 1.

➢**INT_ENABLE_RD_SLVERR[1]**

Enables INT_RAWSTAT_RD_SLVERR.

Default value is 1.

➢**INT_ENABLE_WR_SLVERR[2]**

Enables INT_RAWSTAT_WR_SLVERR.

Default value is 1.

➢**INT_ENABLE_OVERFLOW[3]**

Enables INT_RAWSTAT_OVERFLOW.

Default value is 1.

➢**INT_ENABLE_OVERFLOW[4]**

Enables INT_RAWSTAT_UNDERFLOW.

Default value is 1.

➢**INT_ENABLE_TIMEOUT_RD[5]**

Enables INT_RAWSTAT_TIMEOUT_RD.

Default value is 1.

➢**INT_ENABLE_TIMEOUT_WR[6]**

Enables INT_RAWSTAT_TIMEOUT_WR.

Default value is 1.

- **INT_ENABLE_WDT[7]**

  Enables INT_RAWSTAT_WDT.

  Default value is 1.

•INT_STATUS_REG

Offset: 0xAC

Read/Write: R

Description: Interrupt status. Indicates which interrupts are currently outputted on the INT output pin.

Fields:

- **INT_STATUS_CH_END[0]**

  INT_RAWSTAT_CH_END is set and enabled.

- **INT_STATUS_RD_SLVERR[1]**

  INT_RAWSTAT_RD_SLVERR is set and enabled.

- **INT_STATUS_WR_SLVERR[2]**

  INT_RAWSTAT_WR_SLVERR is set and enabled.

- **INT_STATUS_OVERFLOW[3]**

  INT_RAWSTAT_OVERFLOW is set and enabled.

- **INT_STATUS_UNDERFLOW[4]**

  INT_RAWSTAT_UNDERFLOW is set and enabled.

- **INT_STATUS_TIMEOUT_RD[5]**

  INT_RAWSTAT_TIMEOUT_RD is set and enabled.

- **INT_STATUS_TIMEOUT_WR[6]**

  INT_RAWSTAT_TIMEOUT_WR is set and enabled.

- **INT_STATUS_WDT[7]**

  INT_RAWSTAT_WDT is set and enabled.

9.3 Shared registers

•INT0_STATUS

Offset: 0x1000

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[0]

Fields:

- **CORE0_CH0_INT0_STAT[0]**

  Interrupt caused by Channel 0 in Core 0.

  Reset value is 0.

- **CORE0_CH1_INT0_STAT[1]**

  Interrupt caused by Channel 1 in Core 0.

  Reset value is 0.

- **CORE0_CH2_INT0_STAT[2]**

  Interrupt caused by Channel 2 in Core 0.

  Reset value is 0.

- **CORE0_CH3_INT0_STAT[3]**

  Interrupt caused by Channel 3 in Core 0.

  Reset value is 0.

- **CORE0_CH4_INT0_STAT[4]**

  Interrupt caused by Channel 4 in Core 0.

  Reset value is 0.

- **CORE0_CH5_INT0_STAT[5]**

  Interrupt caused by Channel 5 in Core 0.

  Reset value is 0.

- **CORE0_CH6_INT0_STAT[6]**

  Interrupt caused by Channel 6 in Core 0.

  Reset value is 0.

- **CORE0_CH7_INT0_STAT[7]**

  Interrupt caused by Channel 7 in Core 0.

  Reset value is 0.

- **CORE1_CH0_INT0_STAT[8]**

  Interrupt caused by Channel 0 in Core 1.

  Reset value is 0.

- **CORE1_CH1_INT0_STAT[9]**

  Interrupt caused by Channel 1 in Core 1.

Reset value is 0.

➢**CORE1_CH2_INT0_STAT[10]**

Interrupt caused by Channel 2 in Core 1.

Reset value is 0.

➢**CORE1_CH3_INT0_STAT[11]**

Interrupt caused by Channel 3 in Core 1.

Reset value is 0.

➢**CORE1_CH4_INT0_STAT[12]**

Interrupt caused by Channel 4 in Core 1.

Reset value is 0.

➢**CORE1_CH5_INT0_STAT[13]**

Interrupt caused by Channel 5 in Core 1.

Reset value is 0.

➢**CORE1_CH6_INT0_STAT[14]**

Interrupt caused by Channel 6 in Core 1.

Reset value is 0.

➢**CORE1_CH7_INT0_STAT[15]**

Interrupt caused by Channel 7 in Core 1.

Reset value is 0.

•**INT1_STATUS**

Offset: 0x1004

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[1]

Fields: Identical to INT0_STATUS.

•**INT2_STATUS**

Offset: 0x1008

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[2]

Fields: Identical to INT0_STATUS.

•**INT3_STATUS**

Offset: 0x100C

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[3]

Fields: Identical to INT0_STATUS.

•INT4_STATUS

Offset: 0x1010

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[4]

Fields: Identical to INT0_STATUS.

•INT5_STATUS

Offset: 0x1014

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[5]

Fields: Identical to INT0_STATUS.

•INT6_STATUS

Offset: 0x1018

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[6]

Fields: Identical to INT0_STATUS.

•INT7_STATUS

Offset: 0x101C

Read/Write: R

Description: Status register indicating which channels caused an interrupt on INT[7]

Fields: Identical to INT0_STATUS.

•CORE0_JOINT_MODE

Offset: 0x1030

Read/Write: R/W

Description: Core 0 joint mode

Fields:

- **CORE0_JOINT_MODE[0]**

  If set core 0 works in joint mode otherwise in independent mode.

  Reset value is 0.

•**CORE1_JOINT_MODE**

Offset: 0x1034

Read/Write: R/W

Description: Core 1 joint mode

Fields:

➢**CORE0_JOINT_MODE[0]**

If set core 1 works in joint mode otherwise in independent mode.

Reset value is 0.

•**CORE0_PRIORITY**

Offset: 0x1038

Read/Write: R/W

Description: Core 0 priority channels

Fields:

➢**CORE0_RD_PRIO_TOP_NUM[2:0]**

Core 0 read top priority channel number.

Reset value is 0.

➢**CORE0_RD_PRIO_TOP[3]**

Core 0 read top priority enable.

Reset value is 0.

➢**CORE0_RD_PRIO_HIGH_NUM[6:4]**

Core 0 read high priority channel number.

Reset value is 0.

➢**CORE0_RD_PRIO_HIGH[7]**

Core 0 read high priority enable.

Reset value is 0.

➢**CORE0_WR_PRIO_TOP_NUM[10:8]**

Core 0 write top priority channel number.

Reset value is 0.

➢**CORE0_WR_PRIO_TOP[11]**

Core 0 write top priority enable.

Reset value is 0.

➢**CORE0_WR_PRIO_HIGH_NUM[14:12]**

Core 0 write high priority channel number.

Reset value is 0.

➢**CORE0_WR_PRIO_HIGH[15]**

Core 0 write high priority enable.

Reset value is 0.

•CORE1_PRIORITY

Offset: 0x103C

Read/Write: R/W

Description: Core 1 priority channels

Fields:

➢**CORE1_RD_PRIO_TOP_NUM[2:0]**

Core 1 read top priority channel number.

Reset value is 0.

➢**CORE1_RD_PRIO_TOP[3]**

Core 1 read top priority enable.

Reset value is 0.

➢**CORE1_RD_PRIO_HIGH_NUM[6:4]**

Core 1 read high priority channel number.

Reset value is 0.

➢**CORE1_RD_PRIO_HIGH[7]**

Core 1 read high priority enable.

Reset value is 0.

➢**CORE1_WR_PRIO_TOP_NUM[10:8]**

Core 1 write top priority channel number.

Reset value is 0.

➢**CORE1_WR_PRIO_TOP[11]**

Core 1 write top priority enable.

Reset value is 0.

➢**CORE1_WR_PRIO_HIGH_NUM[14:12]**

Core 1 write high priority channel number.

Reset value is 0.

➢**CORE1_WR_PRIO_HIGH[15]**

Core 1 write high priority enable.

Reset value is 0.

•CORE1_CLKDIV

Offset: 0x1040

Read/Write: R/W

Description: Core 1 clock divider

Fields:

- **CORE1_CLKDIV_RATIO[3:0]**

  Ratio between main clock and core 1 clock.

  Reset value is 1.

•CORE0_CH_START

Offset: 0x1048

Read/Write: W

Description: Core 0 channel start

Fields:

- **CORE0_CHANNEL_START[7:0]**

  Allow to start multiple channels simultaneously. Each bit set starts the corresponding channel.

•CORE1_CH_START

Offset: 0x104C

Read/Write: W

Description: Core 1 channel start

Fields:

- **CORE1_CHANNEL_START[7:0]**

  Allow to start multiple channels simultaneously. Each bit set starts the corresponding channel.

•PERIPH_RX_CTRL

Offset: 0x1050

Read/Write: R/W

Description: Direct control of peripheral RX request

Fields:

- **PERIPH_RX_REQ[31:1]**

  Allows direct control of the peripheral RX request bus. Particularly useful when the application wishes to read results without waiting for them to arrive. Cleared automatically by HW periph_rx_clr singal.

  Bit 0 is reserved for uncontrolled transfers.

  Reset value is 0.

•PERIPH_TX_CTRL

Offset: 0x1054

Read/Write: R/W

Description: Direct control of peripheral TX request

Fields:

➢**PERIPH_TX_REQ[31:1]**

Allows direct control of the peripheral TX request bus. Particularly useful when the application wishes to write configuration registers to a slow device. Cleared automatically by HW periph_tx_clr singal.

Bit 0 is reserved for uncontrolled transfers.

Reset value is 0.

•IDLE

Offset: 0x10D0

Read/Write: R

Description: Idle indication register

Fields:

➢**IDLE[0]**

Indicates that all channels have stopped working and all bus transactions have completed.

•USER_DEF_STATUS

Offset: 0x10E0

Read/Write: R

Description: Status register indicating user defined configurations

Fields:

➢**USER_DEF_INT_NUM[3:0]**

Number of bits in interrupt bus INT.

➢**USER_DEF_CORE1_CLKDIV[4]**

If set core 1 uses clock divider and AHB synchronizer.

➢**USER_DEF_DUAL_CORE[5]**

If set the design has two cores else a single core.

➢**USER_DEF_IC[6]**

If set an AHB matrix is used.

➢**USER_DEF_IC_DUAL_PORT[7]**

If set the AHB matirx has two output ports otherwise it has a single port.

➢**USER_DEF_CLKGATE[8]**

If set the design contains functional clock gates.

➢**USER_DEF_PORT0_MUX[9]**

- AHB port 0 is using an AHB mux.

➢**USER_DEF_PORT1_MUX[10]**

- AHB port 1 is using an AHB mux.

•USER_CORE0_DEF_STATUS0

Offset: 0x10F0

Read/Write: R

Description: Status register indicating user defined configurations

Fields:

➢**USER_DEF_CORE0_CH_NUM[3:0]**

Number of channels in core 0.

➢**USER_DEF_CORE0_FIFO_SIZE[7:4]**

Log2 of core 0 FIFO size per channel.

➢**USER_DEF_CORE0_WCMD_DEPTH[11:8]**

Log2 of core 0 maximum number of pending write commands.

➢**USER_DEF_CORE0_AHB_32[12]**

If set core 0 AHB bus is 32 bit otherwise 64 bit.

➢**USER_DEF_CORE0_ADDR_BITS[21:16]**

Number of bits in all core 0 address buses.

➢**USER_DEF_CORE0_BUFF_BITS[28:24]**

Number of bits in core 0 BUFFER_SIZE

•USER_CORE0_DEF_STATUS1

Offset: 0x10F4

Read/Write: R

Description: Status register indicating user defined configurations

Fields:

➢**USER_DEF_CORE0_WDT[0]**

If set core 0 has a watchdog timer.

➢**USER_DEF_CORE0_TIMEOUT[1]**

If set core 0 supports timeouts on AHB read and write buses.

➢**USER_DEF_CORE0_TOKENS[2]**

If set core 0 has tokens support.

➢**USER_DEF_CORE0_PRIO[3]**

If set core 0 has priority modes support.

➢**USER_DEF_CORE0_OUTS[4]**

If set core 0 supports outstanding mode.

➢**USER_DEF_CORE0_WAIT[5]**

If set core 0 supports scheduled channels .

- **USER_DEF_CORE0_BLOCK[6]**

  If set core 0 supports block transfer.

- **USER_DEF_CORE0_JOINT[7]**

  If set core 0 supports joint mode.

- **USER_DEF_CORE0_INDEPENDENT[8]**

  If set core 0 supports independent mode.

- **USER_DEF_CORE0_PERIPH[9]**

  If set core 0 supports peripherals.

- **USER_DEF_CORE0_LISTS[10]**

  If set core 0 supports command lists.

- **USER_DEF_CORE0_END[11]**

  If set core 0 supports endianness swapping.

•USER_CORE1_DEF_STATUS0

Offset: 0x10F8

Read/Write: R

Description: Status register indicating user defined configurations

Fields:

- **USER_DEF_CORE1_CH_NUM[3:0]**

  Number of channels in core 1.

- **USER_DEF_CORE1_FIFO_SIZE[7:4]**

  Log2 of core 1 FIFO size per channel.

- **USER_DEF_CORE1_WCMD_DEPTH[11:8]**

  Log2 of core 1 maximum number of pending write commands.

- **USER_DEF_CORE1_AHB_32[12]**

  If set core 1 AHB bus is 32 bit otherwise 64 bit.

- **USER_DEF_CORE1_ADDR_BITS[21:16]**

  Number of bits in all core 1 address buses.

- **USER_DEF_CORE1_BUFF_BITS[28:24]**

  Number of bits in core 1 BUFFER_SIZE

•USER_CORE1_DEF_STATUS1

Offset: 0x10FC

Read/Write: R

Description: Status register indicating user defined configurations

Fields:

- **USER_DEF_CORE1_WDT[0]**

  If set core 1 has a watchdog timer.

- **USER_DEF_CORE1_TIMEOUT[1]**

  If set core 1 supports timeouts on AHB read and write buses.

- **USER_DEF_CORE1_TOKENS[2]**

  If set core 1 has tokens support.

- **USER_DEF_CORE1_PRIO[3]**

  If set core 1 has priority modes support.

- **USER_DEF_CORE1_OUTS[4]**

  If set core 1 supports outstanding mode.

- **USER_DEF_CORE1_WAIT[5]**

  If set core 1 supports scheduled channels .

- **USER_DEF_CORE1_BLOCK[6]**

  If set core 1 supports block transfer.

- **USER_DEF_CORE1_JOINT[7]**

  If set core 1 supports joint mode.

- **USER_DEF_CORE1_INDEPENDENT[8]**

  If set core 1 supports independent mode.

- **USER_DEF_CORE1_PERIPH[9]**

  If set core 1 supports peripherals.

- **USER_DEF_CORE1_LISTS[10]**

  If set core 1 supports command lists.

- **USER_DEF_CORE1_END[11]**

  If set core 1 supports endianness swapping.