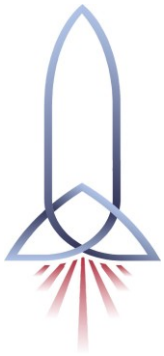
School of Computing
UNIVERSITY OF DUNDEE

SPACE SYSTEMS RESEARCH GROUP



Space
Technology
Centre
University of Dundee

SpaceNet – RMAP IP Core

User Manual

Revision:	Issue 1.6
Date:	2nd March 2010
ESA Contract Number	220774-07-NL/LvH
Ref:	RMAP IP WP2-400.3

Space Technology Centre
School of Computing
University of Dundee
Dundee, DD1 4HN
Scotland, UK
spacetech.computing.dundee.ac.uk

Document Authors

Chris McClements (CMC)

Martin Dunstan (MND)

Document Change Log

Date	Revision No	Comments
29th September 2008	Issue 1.0	MND: First issue
24th December 2008	Issue 1.1	CMC: Added initiator interfaces, initiator data structures and synthesis sections
19th January 2009	Issue 1.2	CMC: Initiator bit order correction
12th February 2009	Issue 1.3	CMC: User comment updates
9th March 2009	Issue 1.4	CMC: See numerous document changes in section 10
18th May 2009	Issue 1.5	CMC: Add additional transaction debug/status signals
2nd March 2010	Issue 1.6	CMC: Updated synthesis results
2nd March 2010	Issue 1.7	CMC: Update RMAP document reference number

A comprehensive list of the changes which have been made to this document in each revision is provided in section 10.

CONTENTS

I LIST OF FIGURES

II LIST OF TABLES

1 INTRODUCTION

1.1 AIMS AND OBJECTIVES

WP2 in the SpaceNet activity aims to provide a SpaceWire interface VHDL core that includes the RMAP protocol extension to SpaceWire. This will enable users to readily implement the RMAP protocols in FPGAs or ASICs.

The objective of this document is to provide an introduction to the VHDL source code of the RMAP IP core, define the configuration parameters and describe the RMAP IP core interfaces.

1.2 GUIDE TO DOCUMENT

Section 3 provides an overview of the IP core source code tree and provides a list of the files, in compilation order, needed for synthesis.

Section 4 describes the RMAP IP core architecture covering target and initiator interfaces.

Section 5 describes the configuration and interfaces of the RMAP codec.

Section 6 describes the external DMA bus interface in more detail.

Section 10 contains a comprehensive list of the changes which have been made to this document in each revision.

1.3 Acronyms and Abbreviations

AD	Applicable Document
ASIC	Application Specific Integrated Circuit
DMA	Direct Memory Access
CODEC	Coder Decoder
ECSS	European Cooperation for Space Standardization
ESA	European Space Agency
ESTEC	ESA Space Technology and Research Centre
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
I/F	Interface
IP	Intellectual Property
LVDS	Low Voltage Differential Signalling
PC	Personal Computer
PCB	Printed Circuit Board
RD	Reference Document
RMW	Read/Modify/Write
SpW	SpaceWire
SpW-10X	The SpaceWire Router ASIC device under test
UoD	University of Dundee
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

1.4 Terms and Definitions

1.4.1 Numbers

In this document hexadecimal numbers are written with the prefix 0x, for example 0x34 and 0xdf15. Binary numbers are written with the prefix 0b, for example 0b01001100 and 0b01.

1.4.2 SpaceWire Definitions

Cargo is a sequence of data characters containing the information transmitted in a SpaceWire packet.

Destination is the SpaceWire node that a SpaceWire packet is to be delivered to.

Destination address is a SpaceWire path or logical address of a destination.

EEP is the Error End of Packet marker of a SpaceWire packet which indicates that the SpaceWire packet was terminated prematurely.

EOP is the End Of Packet marker of a SpaceWire packet.

Logical Address is an identifier of a destination which can be used to route a SpaceWire packet to the destination or, if path addressing is being used, to simply confirm that the final destination is the correct one i.e. that the logical address of the destination matches the logical address in the packet.

Packet is a SpaceWire packet which comprises a destination address, a cargo and an end of packet marker.

Path Address is a sequence of one or more SpaceWire data characters that defines the route to a target by specifying, for each router encountered on the way to the destination, the output port that a SpaceWire packet is to be forwarded through. A path address comprises one byte for each router on the path to the destination. Once a path address byte has been used to specify an output port of a router it is deleted to expose the next path address byte for the next router. All path address bytes have been deleted by the time the packet reaches the destination.

Source is the SpaceWire node that sent a SpaceWire packet.

SpaceWire Fabric a SpaceWire point-to-point link or SpaceWire network of links and routers.

SpaceWire Interface is the ESA SpaceWire IP core designed by UoD.

1.4.3 RMAP Definitions

Address is a 32-bit field in an RMAP command that contains the bottom 32-bits of the address to which the data in a write command is to be written or from where data is to be read for a read command. Input/output registers and control/status registers are assumed to be memory mapped. When combined with the Extended Write Address byte a 40-bit memory address is provided. The address can be separated into different fields and interpreted in a variety of different ways provided that the initiator and target both agree on the interpretation. For example, the 40-bit address may be used as a single address space, it may be interpreted as a memory/register bank field followed by an address, it may reference a mailbox or it may use one field to identify a specific application, another to reference a bank of memory or mail box related to that application and a third field to reference the memory location within the memory bank. There are many possible ways in which the address fields can be used. The important feature of the Extended Memory and Address fields is that together they define where in the target data is to be written to or read from.

Command is an RMAP packet sent by an initiator to a target that reads, writes or read-modify-writes data to the target.

Command Header is the header of an RMAP command.

Data is the data that is written in a write command or the data that is read in a read response.

Data CRC is an 8-bit Cyclic Redundancy Check (CRC) used to confirm that the data in the data field is correct before being written in a verified write command or was correctly transferred in a non-verified write command or read reply. The data CRC starts with the character after the header CRC and covers all the data characters.

Data Length is a 24-bit field in an RMAP command that contains the length in bytes of the data that is written or read. The most-significant byte of the length is sent first.

Key is an 8-bit field in an RMAP command that provides a key which is matched by the target user application in order for the RMAP command to be accepted.

Extended Address is an 8-bit field in an RMAP command that contains the most-significant 8-bits of the memory address extending the 32-bit memory address to 40-bits allowing a 1 Terabyte address space to be accessed directly in each node. The Extended Address can be used to differentiate between various address spaces in the target. For example when set to 0x00 it can reference a 4G location directly addressable memory space and when set to 0x01 it can reference an array of mailboxes, which provide indirect addressing.

Header is the first part of a command or reply packet that defines the command or reply. It contains all of the command or reply except the data and data CRC fields.

Header CRC field is an 8-bit Cyclic Redundancy Check (CRC) used to confirm that the header is correct before executing the command. Each character in the header starting with the target logical address and ending with the character before the header CRC itself is used in the CRC.

Initiator is a SpaceWire node that supports the sending of RMAP commands and the receiving of RMAP replies.

Initiator Logical Address is the logical address of the initiator.

Initiator RMAP Interface generates the RMAP command when requested to do so by user logic and sends out it over the SpaceWire interface. It also receives any replies over the SpaceWire interface and passes them back to the initiator user logic.

Initiator User Logic is the user logic that wants to send an RMAP command.

Protocol Identifier identifies the particular protocol being used for communication.

Reply an RMAP packet sent by an RMAP target to an Initiator that contains data read from the target by a read or read-modify-write command, or indicates that a write command was completed successfully, or that contains an error code indicating why a command failed.

Reply Header is the header of an RMAP reply.

Reply SpaceWire Address is the path and/or regional logical address used to route the reply to a command to a node on the SpaceWire network that expects the reply (normally the initiator).

Target is a SpaceWire node that supports the receiving of RMAP commands and the sending of RMAP replies.

Target Logical Address is the logical address of the target.

Target RMAP Interface receives RMAP commands over the SpaceWire interface, executes these commands and sends any replies back over the SpaceWire interface.

Target SpaceWire Address is the path and/or regional logical address to the target on the SpaceWire network.

Target User Logic is the user logic that responds to RMAP commands.

Transaction Identifier is a 24-bit field in RMAP commands and replies used to associate replies with the command that caused the reply. The initiator of the command gives the command a unique transaction identity. This transaction identifier is returned in the reply to the command. This allows the command initiator to send many commands without having to wait for a reply to each command before sending the next command. When a reply comes in it can be quickly associated with the command that caused it by the transaction identifier.

1.5 REFERENCE DOCUMENTS

The documents referenced in this document are listed in Table 1-1.

Table 1-1: Reference Documents

REF	Document Number	Document Title
RD1	UoD-SpaceNet v7, 23rd April 2007	Proposal for SpaceWire Network and Future Onboard Data-Handling, Technical, Management and Administrative Proposal
RD2	TEC-ED/WG/2005.15	SpaceWire Network “SpW-Net” SpaceWire and Future Onboard Data Handling SpaceNet Statement of Work Annex1

1.6 APPLICABLE DOCUMENTS

The documents applicable to this document are listed in Table 1-2.

Table 1-2: Applicable Documents		
REF	**Document Number**	**Document Title**
AD1	ECSS-Q-60-02	Space Product Assurance or ASIC/FPGA development. Final Draft, Issue 5, February 2004
AD2	WDN/PS/70	ASIC Design and Manufacturing Requirements ftp://ftp.estec.esa.nl/pub/vhdl/doc/DesignReq.pdf
AD3	ECSS-E-ST-50-12C	SpaceWire: Links, nodes, routers and networks, 31 July 2008
AD4	ECSS-E-ST-50-52C	SpaceWire - Remote memory access protocol, 5th February 2010
AD5	RMAP IP WP2-100.1	SpaceNet RMAP IP Core Requirements 6th Feb 2008
AD6	RMAP IP WP2-100.2	SpaceNet RMAP IP Core Functional Specification 4th April 2008
AD7	RMAP IP WP2-100.2	SpaceNet RMAP IP Core Interface Specification 4th April 2008

2 LIMITATIONS AND INFORMATION

This list below contains the limitations of the RMAP core.

- **The initiator currently supports an external bus size 4 words, 32 bits.**

The list below provides information on common problems and pitfalls which the user should avoid when implementing the core

- **The initiator reads transaction and header information records from the external bus dependent on the configuration generics CFG_INI_CODEC_MSBFIRST. Write and read data transfers are dependent on the transaction header flags held in the transaction record.**
- **When CFG_ALLOW_LOOPBACK=1 and the initiator and target are enabled, CFG_INITIATOR_EN=1 and CFG_TARGET_EN=1, there is a timing path in the design between the initiator DMA controller and the target verified data buffer and target DMA controller through the SpaceWire Loopback controller.**

3 VHDL File Hierarchy

The RMAP IP core is distributed with the following top-level directories:

Directory	Contents/Purpose
doc	Documentation such as this user manual
extern	Sources for external components such as the SpaceWire link codec IP
src	Sources for the RMAP codec IP
design/synth_example	Synthesis directories for the RMAP IP core and individual units for Mentor Graphics precision for gate count estimations and example synthesis scripts.

The top-level file for the codec is `src/vhdl/top/rmap_codec_ip.vhd` and all related RMAP codec source files can be found in the `src/vhdl` tree. The top-level codec file refers to the design unit `spwrlinkwrap` which represents the SpaceWire codec to be used with the RMAP core. The source code for this unit is in `src/vhdl/spw/spwrlinkwrap.vhd` and is a wrapper around the UoD SpaceWire link codec IP found in the `extern` directory. The SpaceWire link codec configuration settings are in `src/vhdl/spw/spwrlink_pkg.vhd`. For more details about using the SpaceWire link codec IP please refer to the separate documentation released with that IP: only the VHDL source code files are provided in the `extern` directory.

A complete list of all the VHDL source files needed to synthesise the SpaceWire link codec IP for an FPGA for use with the RMAP codec IP is given below in compilation order:

Root	Directory	Filename
src/vhdl/	spw/	spwrlink_pkg.vhd
extern/spw_v2_03/src/vhdl/	initfsm/	initfsm_counter.vhd
extern/spw_v2_03/src/vhdl/	initfsm/	initfsm_sync.vhd
extern/spw_v2_03/src/vhdl/	initfsm/	init_fsm.vhd
extern/spw_v2_03/src/vhdl/	other/	clk10gen.vhd
extern/spw_v2_03/src/vhdl/	other/	clkmux.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxclock.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxcredit.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxdataformat.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxdecode.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxdiscerr.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxnchar_resync_valid.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxnchar_resync_ffstore_inferfpgaram.vhd

extern/spw_v2_03/src/vhdl/	receive/	rxnchar_resync_ff.vhd
extern/spw_v2_03/src/vhdl/	receive/	rxtcode_resync.vhd
extern/spw_v2_03/src/vhdl/	transmit/	txddrreg.vhd
extern/spw_v2_03/src/vhdl/	transmit/	txddrreg_noenable.vhd
extern/spw_v2_03/src/vhdl/	transmit/	txencode.vhd
extern/spw_v2_03/src/vhdl/	transmit/	txtcode_send.vhd
extern/spw_v2_03/src/vhdl/	txclk/	txclk_divider.vhd
extern/spw_v2_03/src/vhdl/	txclk/	txclk_en_gen.vhd
extern/spw_v2_03/src/vhdl/	txclk/	txclkgen.vhd
extern/spw_v2_03/src/vhdl/	top/	spwrlink.vhd

A complete list of all the VHDL source files needed to synthesise the RMAP codec IP for an FPGA is given below in compilation order:

Root	Directory	Filename
src/vhdl/	mem	fifo_out_valid.vhd
src/vhdl/	sync_fifo	sync_fifo_logic.vhd
src/vhdl/	sync_fifo	sync_dpfifo.vhd
src/vhdl/	sync_fifo	sync_memblock_fpga_memory.vhd
src/vhdl/	async_fifo	async_dpfifo.vhd
src/vhdl/	async_fifo	async_fifo_logic.vhd
src/vhdl/	async_fifo	async_fifo_readptr.vhd
src/vhdl/	async_fifo	async_fifo_writeptr.vhd
src/vhdl/	async_fifo	async_memblock_fpga_memory.vhd
src/vhdl/	spw	spwrlinkwrap.vhd
src/vhdl/	pkg	support_pkg.vhd
src/vhdl/	rmap	rmap_pkg.vhd
src/vhdl/	dma	dma_pkg.vhd
src/vhdl/	pkg	tech_pkg.vhd
src/vhdl/	rmap	pack_rmap_word.vhd
src/vhdl/	rmap	unpack_rmap_word.vhd
src/vhdl/	rmap	target_verify_control.vhd
src/vhdl/	rmap	target_command_decode.vhd
src/vhdl/	rmap	target_reply_encode.vhd
src/vhdl/	rmap	target_controller.vhd
src/vhdl/	rmap	rmap_target.vhd
src/vhdl/	rmap	ini_command_encode.vhd

src/vhdl/	rmap	ini_reply_decode.vhd
src/vhdl/	rmap	ini_trans_controller.vhd
src/vhdl/	rmap	ini_delete.vhd
src/vhdl/	rmap	rmap_initiator.vhd
src/vhdl/	dma	dma_burst_fifo_out.vhd
src/vhdl/	dma	dma_burst_fifo_in.vhd
src/vhdl/	dma	dma_controller.vhd
src/vhdl/	dma	bus_master.vhd
src/vhdl/	dma	bus_arbiter.vhd
src/vhdl/	spw	protocol_mux.vhd
src/vhdl/	spw	protocol_demux.vhd
src/vhdl/	spw	timecode_handler.vhd
src/vhdl/	top	verify_buffer.vhd
src/vhdl/	mem	ax_table_32x4.vhd
src/vhdl/	mem	ax_table_32x5.vhd
src/vhdl/	mem	ax_table_32x6.vhd
src/vhdl/	mem	ax_table_32x7.vhd
src/vhdl/	mem	ax_table_32x8.vhd
src/vhdl/	mem	pa3_table_32x4.vhd
src/vhdl/	mem	pa3_table_32x5.vhd
src/vhdl/	mem	pa3_table_32x6.vhd
src/vhdl/	mem	pa3_table_32x7.vhd
src/vhdl/	mem	pa3_table_32x8.vhd
src/vhdl/	mem	transaction_table.vhd
src/vhdl/	top	rmap_kernel.vhd
src/vhdl/	top	rmap_codec_ip.vhd

Alternatives for files with “fpga” in the name can be found in the `extern` and `src` directories with similar names. For example, `src/vhdl/async_fifo/async_memblock.vhd` can be used to implement an asynchronous FIFO from registers as an alternative to FPGA RAM which ought to be inferred when synthesising `async_memblock_fpga_memory.vhd`.

Due to the limitations of some synthesisers for Actel Axcelerator and ProAsic parts the transaction table cannot be inferred correctly from a typical generic synchronous RAM VHDL process. Smartgen Actel memory cores are provided in the “src/vhdl/mem/” directory for use by the transaction table.

The design/synth_example directory contains a TCL script and spwrlink_pkg.vhd configuration file to synthesis the core using the Mentor Graphics precision synthesis tool. The script synthesises the core for the Actel AX2000, Actel ProASIC3E and Xilinx Spartan3E technologies.

4 ARCHITECTURE OVERVIEW

The RMAP IP Core architecture overview is shown in illustrated in Figure 4-1.

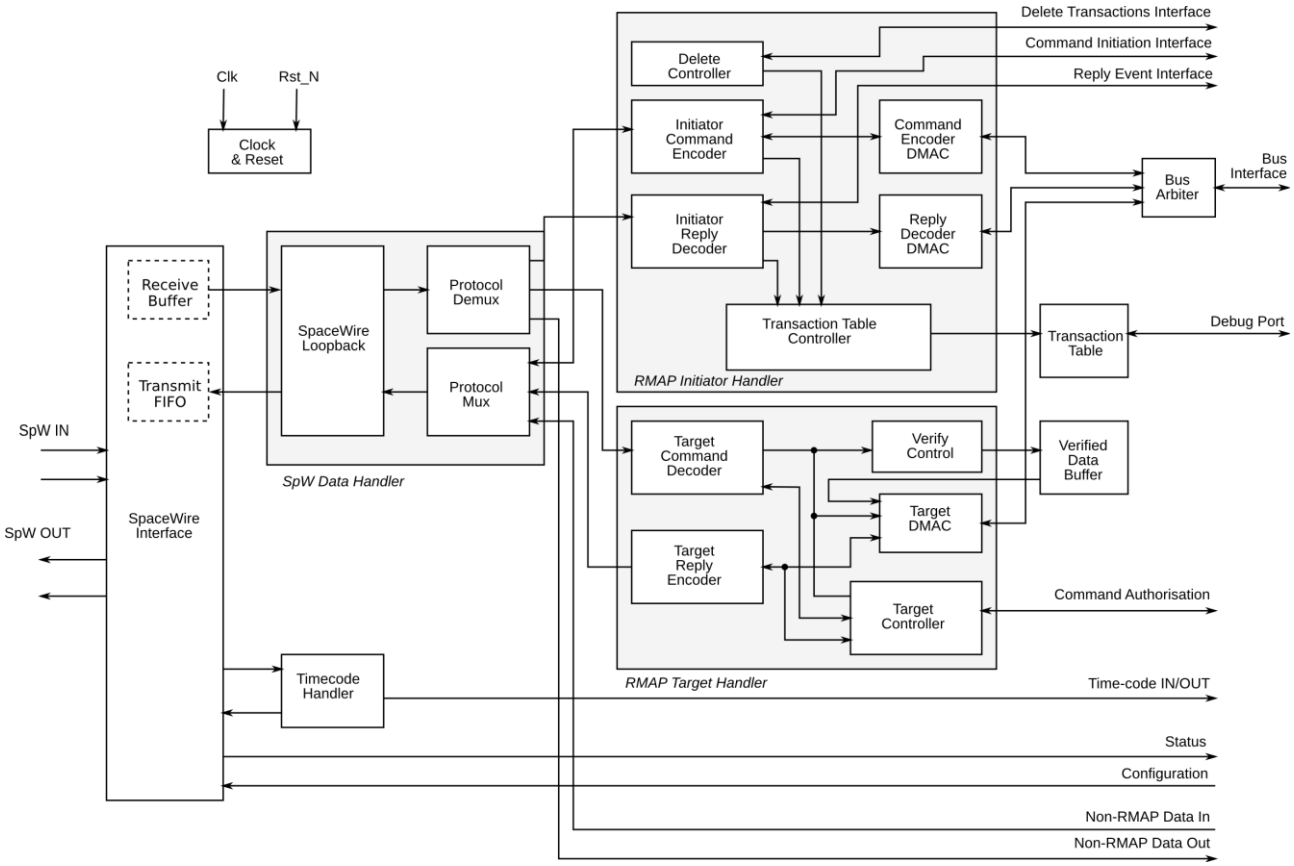


Figure 4-1: RMAP IP Core Architecture Overview

The following paragraphs give an overview of each architectural block in the SpaceWire RMAP IP core design.

4.1 SPACEWIRE INTERFACE

The SpaceWire Interface is responsible for passing received SpaceWire packets and time-codes and passing to the RMAP Handler and Time-Code Handler respectively. It also transmits SpaceWire packets when requested to do so by the RMAP handler. The packets it sends are RMAP reply or acknowledgement packets. The SpaceWire Interface is configured by CONFIG inputs and SpaceWire link status information is made available on the STATUS outputs.

4.1.1 Receive Buffer

The SpaceWire receive buffer holds RMAP packet data received from the SpaceWire link.

4.1.2 Transmit FIFO

The SpaceWire interface transmit FIFO stores RMAP packet data to be send over the SpaceWire link.

4.2 SpaceWire Loopback

The Loop-Back block provides a means of looping back SpaceWire data characters, EOPs and EEPs. When Loop-Back is enabled no SpaceWire packets reach the RMAP handler. Time-codes are not affected by the Loop-Back block.

4.3 Time-code Handler

The Time-Code Handler is responsible for checking time-codes and maintaining the value of the time-code counter. It will assert the TICK_OUT signal when a valid time code is received and put the value of each valid time-code on the TIME-CODE output.

4.4 Protocol Demux

The Protocol De-multiplexer is responsible for de-multiplexing the received SpaceWire data to the initiator reply decoder, target command decoder or the non-RMAP interface. The Demux block decodes the first few bytes of the RMAP packet to determine the destination of the packet.

4.5 Protocol Mux

The Protocol Multiplexer is responsible for the multiplexing of initiator command encode data, target reply encode data or data from the non-RMAP interface. The multiplexed data is written to the SpaceWire interface through the loopback controller.

4.6 Initiator Command Encoder

The Initiator Command Encoder is responsible for generating and sending RMAP commands based on information provided in user memory. User memory is accessed via the initiator command DMA controller. The command encoder uses the contents of the transaction records array to determine the location of the header and data information fin user memory.

4.7 Initiator Transaction Table

The transaction table is responsible for holding details of the current transaction and the outstanding transactions. The current transaction is used by the initiator command encoder to retrieve the command details from user memory. The outstanding transactions are checked by the reply decoder to match command replies to sent commands. Outstanding transactions waiting on reply packets can timeout

dependent on the user transactions array. If an outstanding transaction times out the time out is signalled to the initiator reply decoder which writes the notification status to the notify user memory location.

4.8 Initiator Transaction Table Controller

The transaction table controller controls accesses to the transaction table for the command encoder, reply decoder, delete controller and the transaction table debug port. The controller has five main functions; initialise the transaction table to its default empty state on reset and when instructed to by the delete controller, create a new transaction and add it to the table, delete an existing transaction from the table, search though the table for existing transactions by transaction ID and allow direct read access to the table data fields.

4.9 Initiator Reply Decoder

The Initiator Reply Decoder is responsible for reading and decoding RMAP reply packets from the Protocol Demux component. Data from read reply packets is written to the user memory through the initiator reply DMA controller block. The status of read and write replies is written to the notify location of user memory.

4.10 Initiator Delete Controller

The initiator delete controller allows the user to clear the transaction table or delete an individual transaction from the table which is likely to never receive a response. This is normally handled by a command time-out function of the transaction table but can be manually performed by host software.

4.11 Target Command Decoder

The Target Command Decoder is responsible for decoding RMAP command packets. RMAP command headers are checked for validity and the authorisation parameters are passed to the Target Controller for authorisation by the host. When the RMAP command is a valid write command data is read from the RMAP packet and placed in the user memory by the target DMA controller.

4.12 Target Reply Encoder

The target reply encoder is responsible for sending RMAP reply packets with the status of write command or the data and status from a read command. The status is dependent on the validity of the RMAP command packet and the authorisation request on the host. Reply data is read from the host user memory by the DMA controller and sent in the RMAP packet.

4.13 TARGET CONTROLLER

The target controller controls the reception, authorisation and reply of an RMAP target transaction. The target controller sets up the target DMA controller to perform reads and writes from user memory. Authorisation for RMAP commands is requested from the host through the authorisation parameters.

4.14 TARGET VERIFY CONTROLLER

The target verify controller handles access to the verify buffer when verified data is available from the command decoder.

4.15 DMA CONTROLLER

Each DMA controller provides the interface to user memory and registers. It is responsible for gaining access to the user data bus and performing memory or register, read or write operations.

4.16 STATUS

The Configuration and Status registers, not shown in Figure 4-1, hold configuration and status information for the SpaceWire interface and RMAP Handler. On power up certain configuration registers are loaded with default values specified by the CONFIG interface. Thereafter the configuration values may be changed by writing to the configuration registers either by a SpaceWire-RMAP command or by the User logic writing to the appropriate registers. Status information from the SpaceWire interface and RMAP Handler is held in status registers which can be read by SpaceWire-RMAP command or by the user logic. Certain status information is also available on dedicated signals, STATUS, from the SpW/RMAP IP core.

4.17 CLOCK AND RESET

The Clock and Reset block is responsible for providing the user reset signal, RESET, to the relevant parts of the SpW/RMAP IP core ensuring a clean condition after the reset signal has been asserted. It is also responsible for generating any necessary clock signals from the single clock input signal, CLK.

5 CONFIGURATION AND INTERFACES

The internal and external interfaces are described in this section.

5.1 CONFIGURATION GENERICS

The configuration generics are defined in the following section. All configuration generics are integers unless otherwise stated. When a generic represents a choice, a value of 0 shall be used to mean false/disable/exclude and 1 to mean true/enable/include.

5.1.1 Overview

An overview of the generics is given in Table 5-1

Generic	Description
CFG_TECH	Technology for internal memory structures
CFG_INITIATOR_EN	Enable the initiator
CFG_TARGET_EN	Enable the target
CFG_TARGET_MSB_FIRST	Target byte order
CFG_TARGET_BITSWAP	Target bit swapping
CFG_WORD_SIZE	Byte size of the RMAP data bus
CFG_TARGET_VERIFY_BUF_ABITS	Size of the internal target verified data buffer
CFG_TRGT_FIFO_OUT_ABITS	Size of the DMA output FIFO
CFG_TRGT_FIFO_IN_ABITS	Size of the DMA output FIFO
CFG_TRGT_BURST_SIZE	Number of DMA words per burst
CFG_TRGT_WATCHDOG_TIMEOUT	Enable watchdog timeout on bus transfers
CFG_TRGT_EN_WATCHDOG	Enable watchdog mode
CFG_REQ_TIMEOUT_NBITS	Size of request timeout counter
CFG_SEND_REPLY_ON_EEP_AFTER_CRC	Enable the target to send a reply if an EEP is received after the header CRC
CFG_SEND_REPLY_ON_RESERVED_PKT	Enable the target to send a reply if the packet type has the reserved bit set (bit 7)
CFG_ALLOW_LOOPBACK	Enable loopback logic for testing purposes
CFG_DEMUX_ROUTE_RESERVED_TO_TARGET	When initiator is not enabled route RMAP reserved packets to target command handler
CFG_DEMUX_ROUTE_REPLIES_TO_TARGET	When initiator is not enabled route RMAP reply packets to target command handler
CFG_INI_MAX_COMMANDS	Maximum commands supported by transaction table
CFG_INI_TRTABLE_ABITS	Number of address bits for transaction table and therefore table size
CFG_INI_OUTSTANDING_BITS	Number of bits for outstanding transactions counter

CFG_INI_EXTRA_CHECKS	Enable extra initiator reply packet checks
CFG_INI_CODEC_MSB_FIRST	Order of bytes in initiator table data
CFG_INI_CODEC_BITSWAP	Order of bits in initiator table data
CFG_INI_TIMEOUT_CHECK_WAIT	Time to wait between checking the transaction table for timeouts.
CFG_INI_TIMEOUT_CHECK_MAX	Time to perform timeout checks
CFG_INI_BURST_SIZE	Size of burst transfers for initiator
CFG_INI_WATCHDOG_TIMEOUT	Timeout of watchdog for initiator bus transfers
CFG_INI_EN_WATCHDOG	Enable initiator bus transfer watchdog
CFG_INI_FIFO_OUT_ABITS	Size of Initiator DMA controller FIFO output buffer
CFG_INI_FIFO_IN_ABITS	Size of initiator DMA controller FIFO input buffer

Table 5-1 Generics Overview

5.1.2 CFG_TECH (default TECH_GENERIC)

Set the type of memory block which will be instantiated for the transaction table. The package src/vhdl/pkg/tech_pkg.vhd provides constants for all possible values of CFG_TECH as listed below.

```
constant TECH_MEM_GENERIC     : integer := 0;

constant TECH_MEM_PROASIC     : integer := 1;

constant TECH_MEM_AXCELERATOR : integer := 2;
```

When a prosaic device is used then CFG_TECH should be set to TECH_MEM_PROASIC and if an Axcelerator device is used then CFG_TECH should be set to TECH_MEM_AXCELERATOR.

5.1.3 CFG_INITIATOR_EN (default 0)

Include (1) or exclude (0) the initiator RMAP command packet generator and RMAP reply packet decoder. The initiator reads command packet parameters from host memory and sends RMAP command packets. Any command packet which is expecting a reply will have its transaction identifier placed in a table where the reply packet can be checked against. Reply data is written back to external memory when a read command is performed.

5.1.4 CFG_TARGET_EN (default 1)

Include (1) or exclude (0) the target RMAP command packet decoder and reply packet generator. The target part of the RMAP core responds to RMAP commands and returns the appropriate reply. Commands which have a valid header will be passed to user logic for authorisation. If the command is authorised and no other errors are detected then it will be executed.

5.1.5 CFG_TARGET_MSB_FIRST (default 1)

RMAP packet bytes are transferred to/from the data bus in response to RMAP commands most significant byte first (1) or least significant byte first (0). This setting controls how write command data are converted from sequential packet bytes into external data bus words and controls how read command reply data are converted from external data bus words into sequential packet bytes.

For example, if the target is set to MSB first, the word size is 4 (32 bit external bus) and the length of data sent is 5 the packet data and data in memory interaction will be as shown in Figure 5-1 and Figure 5-2.

Figure 5-1 MSB First

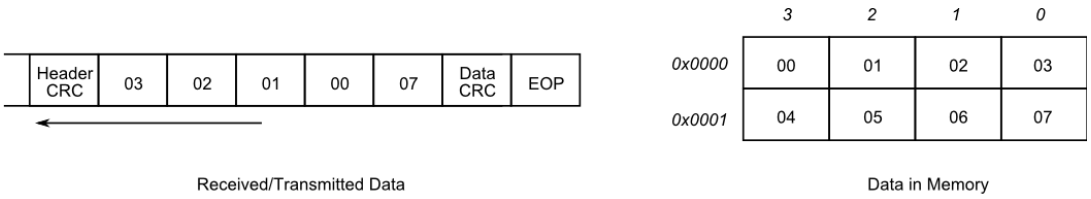


Figure 5-2 LSB First

Note the target setting does not take account of the user bus endianness. If the bus is big endian and data should be sent and received MSB first then CFG_TARGET_MSB_FIRST should be set to 0.

5.1.6 CFG_TARGET_BITSWAP (default 0)

The bits in each RMAP packet byte are reversed in order (1) or left unchanged (0) when packing and unpacking external data bus words.

5.1.7 CFG_WORD_SIZE (default 4)

Defines the width of the user logic data bus measured in 8-bit bytes. The RMAP codec uses word addressing for all DMA transfers over the user logic bus.

Note:	***When the RMAP initiator is enabled the bus size is restricted to 4 bytes (32 bits).***

5.1.8 CFG_TARGET_VERIFY_BUF_ABITS (default 8)

The number of address bits to use for the verified-write buffer. The verify buffer in the current implementation can hold CFG_WORD_SIZE*$2^{CFG\_TARGET\_VERIFY\_BUF\_ABITS}$ bytes. Note that the data CRC is stored in this buffer so the number of data bytes is one less than the buffer size.

5.1.9 CFG_TRGT_FIFO_OUT_ABITS (default 8)

The number of address bits to use for the RMAP target write DMA burst FIFO. The burst FIFO will be able to hold $CFG\_WORD\_SIZE * 2^{(CFG\_TRGT\_FIFO\_OUT\_ABITS)}$ bytes and must be equal or greater than the burst size defined by CFG_TRGT_BURST_SIZE.

5.1.10 CFG_TRGT_FIFO_IN_ABITS (default 8)

The number of address bits to use for the RMAP target read DMA burst FIFO. The burst FIFO will be able to hold $CFG\_WORD\_SIZE * 2^{(CFG\_TRGT\_FIFO\_IN\_ABITS)}$ bytes and must be equal or greater than the burst size defined by CFG_TRGT_BURST_SIZE.

5.1.11 CFG_TRGT_BURST_SIZE (default 8)

The maximum number of words which may be transferred in a single burst. Large RMAP data transfers are divided into one or more bursts. The DMA controller will request control over the bus before each burst and release it afterwards. See also the restrictions placed on this parameter by generics CFG_TRGT_FIFO_OUT_ABITS and CFG_TRGT_FIFO_IN_ABITS.

5.1.12 CFG_TRGT_EN_WATCHDOG (default 1)

Include (1) or exclude (0) the DMA watchdog timer. The DMA watchdog timer is used to prevent the bus from being held indefinitely due to failure of bus source or sink.

5.1.13 CFG_TRGT_WATCHDOG_TIMEOUT (default 1024)

The maximum number of cycles that the DMA bus may be held for. If the watchdog timer is included in the core (see CFG_DMA_EN_WATCHDOG) and the bus has been held for more than CFG_DMA_WATCHDOG_TIMEOUT cycles then the RMAP command will be aborted and a bus error will be reported. This timer is used to prevent a failed device asserting BUS_WAIT indefinitely and thereby prevent the RMAP command from completing. The watchdog timeout ought to be set to a value that is much larger than the expected duration of the longest DMA burst.

5.1.14 CFG_REQ_TIMEOUT_NBITS (default 8)

The number of bits used to hold DMA bus request timeouts. This generic defines the width of the REQ_TIMEOUT input and places an upper limit on the length of request timeouts.

5.1.15 CFG_SEND_REPLY_ON_EEP_AFTER_CRC (default 0)

Send (1) or do not send (0) a reply if an EEP is received immediately after the header CRC of a command. The RMAP standard states (§6.4.3.4.c, §6.5.3.4.c and §6.6.3.4.c) that a reply packet **should not** be sent if an EEP is received immediately after the complete header including the header CRC.

5.1.16 CFG_SEND_REPLY_ON_RESERVED_PKT (default 1)

Send (1) or do not send (0) a reply if a command packet is received with the reserved bit (7) and reply bit (3) set. The RMAP standard states (§6.4.3.4.f.1.c, §6.5.3.4.f.1.c and §6.6.3.4.f.1.c) that a reply packet **shall not** be sent if the instruction field contains an unused packet type. It also states (§6.4.3.4.f.1.d, §6.5.3.4.f.1.d and §6.6.3.4.f.1.d) that a reply packet **may** be sent.

5.1.17 CFG_ALLOW_LOOPBACK (default 0)

Include (1) or exclude (0) the internal/external loopback unit. The loopback unit allows the codec to be placed in loopback mode when LINK_LOOPBACK is asserted high. In loopback mode packets received from the SpaceWire interface are transmitted back through the SpaceWire interface while packets emitted from the protocol mux are passed directly to the protocol demux.

Note:	**When CFG_ALLOW_LOOPBACK=1 and the initiator and target are enabled, CFG_INITIATOR_EN=1 and CFG_TARGET_EN=1, there is a timing path in the design between the initiator DMA controller and the target verified data buffer and target DMA controller through the SpaceWire Loopback controller**

5.1.18 CFG_DEMUX_ROUTE_REPLIES_TO_TARGET (default 0)

If the initiator is not included then send reply packets to the target handler (1) or to the non-RMAP port (0). If the target handler receives a reply packet it will record the error and spill the packet. If reply packets are sent to the non-RMAP port then user logic must process them promptly.

This generic can be set to 1 if the link is not expected to receive reply packets and the designer does not want to add extra logic to the non-RMAP port to deal with them. This generic can be set to 0 if the designer wants to implement an RMAP initiator in user logic (*e.g.* software) using the non-RMAP port to transmit commands and accept replies.

5.1.19 CFG_DEMUX_ROUTE_RESERVED_TO_TARGET (default 0)

Send command packets with the reserved bit set in the packet type field to the target handler (1) or to the non-RMAP port (0). If the target handler receives a packet with the reserved bit set it will record the error and

spill the packet. If packets with the reserved bit set in the packet type field are sent to the non-RMAP port then user logic must process them promptly.

5.1.20 CFG_INI_MAX_COMMANDS

Control the maximum amount of outstanding transactions which the core can be waiting on at any one time. Typically this will be set to the maximum size which the transaction table can support, dependent on CFG_INI_TRTABLE_ABITS as shown below.

$$CFG\_INI\_MAX\_COMMANDS = 2^{(CFG\_IN\_TRTABLE\_ABITS)} / 7$$

For example if CFG_INI_TRTABLE_ABITS=8 then setting CFG_INI_MAX_COMMANDS to 36 will use the maximum amount of transactions space in the transaction table.

The user may wish to reduce the number of outstanding transactions if a target node cannot handle multiple requests.

5.1.21 CFG_INI_TRTABLE_ABITS

Sets the address size of the transaction table. The transaction table is expected to implemented as a memory block.

5.1.22 CFG_INI_CODEC_MSB_FIRST (default 1)

Set the order of transaction and header record data transfers RMAP packet bytes are transferred to/from the data bus in response to RMAP commands most significant byte first (1) or least significant byte first (0). This setting controls the order of initiator records words transferred over the external bus where; MSB first expects the most significant byte to be placed at the bits 31 to 24 and LSB first expects the most significant byte to be placed at bits 7 do 0.

The transaction record flags determine the byte order of data transfers for write command data, read reply data and read modify write data.

5.1.23 CFG_INI_CODEC_BITSWAP (default 0)

The bits in each external bus byte are reversed in order (1) or left unchanged (0) when reading the transaction table. The initiator command data byte order and bit order when sent over the SpaceWire interface is specified as a flag in the command transaction record.

5.1.24 CFG_INI_OUTSTANDING_BITS

Set the number of bits to use for the outstanding counter status port. The outstanding counter is a status port to indicate the number of RMAP replies the core is expecting and the number of transactions in the transaction table.

5.1.25 CFG_INI_TIMEOUT_CHECK_WAIT (default 50)

Sets the number of micro-seconds the RMAP core will wait between checking the transaction table for transactions which have timed out.

5.1.26 CFG_INI_TIMEOUT_CHECK_MAX (default 5)

Sets the number of micro-seconds the RMAP core will spend checking the table for timeouts. During this time the other RMAP core units cannot access the table so no new commands can be started and no replies can be processed.

5.1.27 CFG_INI_BURST_SIZE (default 8)

The maximum number of words which may be transferred in a single burst. Large RMAP data transfers are divided into one or more bursts. The DMA controller will request control over the bus before each burst and release it afterwards. See also the restrictions placed on this parameter by generics CFG_INI_FIFO_OUT_ABITS and CFG_INI_FIFO_IN_ABITS.

5.1.28 CFG_INI_EN_WATCHDOG (default 1)

Include (1) or exclude (0) the DMA watchdog timer. The DMA watchdog timer is used to prevent the bus from being held indefinitely due to failure of bus source or sink.

5.1.29 CFG_INI_WATCHDOG_TIMEOUT (default 1024)

The maximum number of cycles that the DMA bus may be held for. If the watchdog timer is included in the core (see CFG_DMA_EN_WATCHDOG) and the bus has been held for more than CFG_DMA_WATCHDOG_TIMEOUT cycles then the RMAP command will be aborted and a bus error will be reported. This timer is used to prevent a failed device asserting BUS_WAIT indefinitely and thereby prevent the RMAP command from completing. The watchdog timeout ought to be set to a value that is much larger than the expected duration of the longest DMA burst.

5.1.30 CFG_INI_FIFO_OUT_ABITS (default 8)

The number of address bits to use for the RMAP target write DMA burst FIFO. The burst FIFO will be able to hold $CFG\_WORD\_SIZE * 2^{(CFG\_INI\_FIFO\_OUT\_ABITS)}$ bytes and must be equal or greater than the burst size defined by CFG_INI_BURST_SIZE.

5.1.31 CFG_INI_FIFO_IN_ABITS (default 8)

The number of address bits to use for the RMAP target read DMA burst FIFO. The burst FIFO will be able to hold $CFG\_WORD\_SIZE * 2^{(CFG\_INI\_FIFO\_IN\_ABITS)}$ bytes and must be equal or greater than the burst size defined by CFG_INI_BURST_SIZE.

5.2 Clock/Reset Interface

This interface provides an asynchronous reset and the system clock which all the other interfaces are synchronous with unless otherwise stated.

Signal	Width	I/O	Description
RST_N	1	In	Active low asynchronous reset
CLK	1	In	System clock

5.3 Low-Level SpaceWire Interface

The low-level SpaceWire interface provides control over the SpaceWire link used by the RMAP interface. Refer to the separate documentation of the SpaceWire codec IP core for more details about the configuration and control of this part of the interface.

Signal	Width	I/O	Description
SLOWCLK	1	in	Slow clock used for transmit during link initialisation
TXCLK	1	In	Transmit clock used for transmit when link running
DIN	1	In	SpaceWire data in
SIN	1	In	SpaceWire strobe in
DOUT	1	Out	SpaceWire data out (non-DDR)
SOUT	1	Out	SpaceWire strobe out (non-DDR)
SLOW_EN	1	In	Clock enable for the slow clock
SLOWRATE_SYSCLK	8	In	Divider used to obtain the slow clock from CLK
SLOWRATE_TXCLK	8	In	Divider used to obtain the slow clock from TXCLK
TXRATE	8	In	Divider used to obtain transmit bit clock from TXCLK
TXBITCLK	1	Out	Transmit bit clock for DDR
DOUT_R	1	Out	SpaceWire data out DDR rising edge
DOUT_F	1	Out	SpaceWire data out DDR falling edge
SOUT_R	1	Out	SpaceWire strobe out DDR rising edge
SOUT_F	1	Out	SpaceWire strobe out DDR falling edge

5.4 High-Level SpaceWire Interface

The high-level SpaceWire interface allows user logic to start and stop the SpaceWire link and to detect errors occurring on the link.

Signal	Width	I/O	Description
LINK_START	1	In	Start SpaceWire link if not disabled
LINK_DISABLE	1	In	Disable SpaceWire link
LINK_AUTO	1	In	Auto-start SpaceWire link if not disabled

LINK_LOOPBACK	1	In	Set loopback mode
STAT_LINK_RUNNING	1	Out	SpaceWire link is running (Run state)
STAT_LINK_DISCONNECT	1	Out	The SpaceWire link has disconnected
STAT_LINK_PARITY_ERROR	1	Out	A parity error has been detected
STAT_LINK_CREDIT_ERROR	1	Out	A credit error has been detected
STAT_LINK_ESCAPE_ERROR	1	Out	An escape error has been detected

The LINK_DISABLE signal takes priority over LINK_START and LINK_AUTO: if LINK_DISABLE is asserted then the SpaceWire link will not start even if LINK_START is asserted. If LINK_AUTO is asserted then the SpaceWire link will automatically start when the first NULL is received. Otherwise the link can be explicitly started by asserting LINK_START. When the link reaches the Run state STAT_LINK_RUNNING will be asserted. The link can be stopped at any time by asserting LINK_DISABLE. The STAT_LINK_DISCONNECT, STAT_LINK_PARITY_ERROR, STAT_LINK_CREDIT_ERROR and STAT_LINK_ESCAPE_ERROR signals will be asserted for one CLK cycle when an error of the corresponding type is detected.

If CFG_ALLOW_LOOPBACK=1 then loopback mode will be entered when LINK_LOOPBACK is asserted. See Section 5.1.2 for more details.

5.5 TIMECODE INTERFACE

Allows SpaceWire timecodes to be sent and received.

Signal	Width	I/O	Description
TIME_MASTER_EN	1	In	Tick-in codes accepted
TICK_IN	1	In	Send a time-code
TIME_IN	8	In	Time-code value to send
TICK_OUT	1	Out	Time-code received
TIME_OUT	8	Out	Time-code value received

The TICK_IN/TIME_IN interface will be used only if TIME_MASTER_EN is asserted. If it is then a timecode with value TIME_IN will be transmitted after TICK_IN is asserted. Whenever a timecode is received TICK_OUT will be asserted and TIME_OUT will hold the value of the timecode if and only if the timecode received is one larger than the previous timecode received.

5.6 NON-RMAP RECEIVE INTERFACE

This interface allows user logic to receive packets non-RMAP packets.

Signal	Width	I/O	Description
NR_RX_EMPTY	1	Out	No non-RMAP data characters to be read.
NR_RX_READ	1	In	Read non-RMAP data character
NR_RX_DATA	9	Out	Non-RMAP data character to be read

This port provides a simple FIFO interface to allow non-RMAP packets received by the codec to be passed to user logic. When data is available to be read NR_RX_EMPTY will be deasserted. The 9-bit SpaceWire character on NR_RX_DATA will be consumed on the rising edge of CLK if the read enable NR_RX_READ is asserted.

If CFG_ALLOW_LOOPBACK=1 and LINK_LOOPBACK is asserted then any commands from the initiator handler, any replies from the target handler and any packets from the non-RMAP transmit interface will be passed to user logic via this interface.

If there is no initiator handler and CFG_DEMUX_ROUTE_REPLIES_TO_TARGET=0 then RMAP reply packets will be sent to this port. If CFG_DEMUX_ROUTE_RESERVED_TO_TARGET=0 and an RMAP packet with type field 0b11 is received then it will be sent to this port. Similarly if there is no initiator handler and CFG_DEMUX_ROUTE_RESERVED_TO_TARGET=0 then any RMAP packets with type field 0b10 will be sent to this port.

5.7 NON-RMAP TRANSMIT INTERFACE

This interface allows user logic to transmit packets of any form over the SpaceWire link.

Signal	Width	I/O	Description
NR_TX_FULL	1	Out	Unable to accept more non-RMAP data characters
NR_TX_WRITE	1	In	Submit non-RMAP data character for transmission
NR_TX_DATA	9	In	Non-RMAP data character to be transmitted

This port provides a modified FIFO interface to allow non-RMAP packets to be submitted by user logic to the codec for transmission over the SpaceWire link. Each character of the packet must be put on NR_TX_DATA before the rising edge of CLK and NR_TX_WRITE asserted. The character will be accepted on the rising edge of CLK if NR_TX_FULL is not asserted. NR_TX_FULL is valid *only* while NR_TX_WRITE is asserted. Thus user logic must attempt to send data before it can tell whether the data will be accepted.

If CFG_ALLOW_LOOPBACK=1 and LINK_LOOPBACK is asserted then any RMAP commands sent through this interface will be routed to the target handler, any RMAP replies will be routed to the initiator handler and all other packets will be sent to the non-RMAP receive interface.

5.8 EXTERNAL BUS INTERFACE

Provides a pipelined bus interface to user logic for DMA. It is used by the target handler when processing read, write and RMW commands; it is used by the initiator handler to retrieve details of commands to be sent from user logic and when generating read, write and RMW commands; it is used by the initiator handler to pass status information back to user logic for commands in progress. The external bus interface is designed to be compatible with the AMBA AHB interface specification revision 2.0. For information about bus operation see Section 6.

Signal	Width	I/O	Description
REQ_TIMEOUT	Variable	In	Set the RMAP bus request timeout
BUS_REQ	1	Out	Request access to the bus
BUS_GNT	1	In	Granted access to the bus
BUS_TRANS	2	Out	Type of transaction (IDLE, START, ENABLED, BUSY)
BUS_WRITE	1	Out	Write operation enable (‘0’=read, ‘1’=write)
BUS_BANK	8	Out	Memory bank (RMAP external address)
BUS_ADDR	32	Out	Memory word (RMAP address)
BUS_DATA_IN	Variable	In	Word read from user logic
BUS_DATA_OUT	Variable	Out	Word to write to user logic
BUS_DATA_OUT_BYTE_EN	Variable	Out	Marks which bytes of BUS_DATA_OUT are valid
BUS_WAIT	1	In	Host is not ready to perform the operation
BUS_ERROR	1	In	Host cannot perform the operation due to an error

The REQ_TIMEOUT input defines the maximum number of CLK cycles that the DMA interface will wait for BUS_GNT to be asserted when asserting BUS_REQ. The size of this port is controlled by the CFG_REQ_TIMEOUT_NBITS generic.

When the bus has been granted to the codec the BUS_TRANS signal will define the state of the transaction in progress: IDLE (no operation), START (address phase of the first read/write), ENABLED (address phase of additional read/writes) and BUSY (unable to present another address in this CLK cycle). The bus is pipelined so that BUS_WRITE, BUS_BANK and BUS_ADDR are asserted in the address phase of an operation with the corresponding data being captured from BUS_DATA_IN or presented on BUS_DATA_OUT on the following cycle. The address phase of one operation is expected to overlap with the data phase of the previous operation. If user logic is unable to supply or capture data for a given address in the data phase of an operation then it must assert BUS_WAIT until it can do. If user logic cannot satisfy a request then it must assert BUS_ERROR to abort the transaction. BUS_DATA_OUT_BYTE_EN is used by the codec to tell user logic which bytes of BUS_DATA_OUT are valid. This is used, for example, when a single byte write is made over a multi-byte bus.

CFG_WORD_SIZE defines the width of BUS_DATA_IN and BUS_DATA_OUT. The order in which bytes and bits on the bus are converted to/from RMAP serial byte order is controlled by generics CFG_TARGET_MSB_FIRST and CFG_TARGET_BITSWAP.

5.9 Read Modify Write Interface

This interface allows user logic to define the behaviour of the modify step of a RMW command.

Signal	Width	I/O	Description
RMW_VALID	1	Out	RMW outputs are valid; supply modified value
RMW_MEM_DATA	32	Out	Data read from memory
RMW_DATA_OUT	32	Out	Data from the RMAP packet
RMW_MASK	32	Out	Mask from the RMAP packet
RMW_BYTE_EN	4	Out	Which bytes of RMW_MEM_DATA etc to process
RMW_DATA_IN	32	In	Modified data value to write back to memory
RMW_ACK	1	In	Set when RMW_DATA_IN is valid

When the target performs a RMW command it reads the data and mask fields from the RMAP packet and places them on RMW_DATA_OUT and RMW_MASK in bus data order. The data word is read from user logic via the bus (see Section 5.8) and placed directly on RMW_MEM_DATA. RMW_BYTE_EN is updated to indicate which bytes are valid based on the length of the RMW. The codec then asserts RMW_VALID; user logic must compute the modified value to be written back to memory and place it on RMW_DATA_IN before asserting RMW_ACK. The code will take the value from RMW_DATA_IN and write it directly to user logic via the bus.

5.10 Target Authorisation Interface

This interface allows user logic to authorise all valid RMAP commands received by the target.

Signal	Width	I/O	Description
AUTH_REQUEST	1	Out	Request authorisation of the target parameters
AUTH_LOGICAL_ADDR	8	Out	Destination logical address of the command
AUTH_COMMAND	8	Out	Command byte of the command being authorised
AUTH_KEY	8	Out	Key of the command being authorised
AUTH_INIT_LOGICAL_ADDR	8	Out	Initiator logical address of the command
AUTH_TRANS_ID	16	Out	Transaction ID of the command being authorised
AUTH_EXT_ADDRESS	8	Out	Extended address of the command being authorised
AUTH_ADDRESS	32	Out	Address of the command being authorised
AUTH_DATA_LENGTH	24	Out	Length of the command being authorised
AUTH_RESPONSE	1	In	Authorisation request processed

AUTH_GRANT	1	In	Command accepted
AUTH_DEST_KEY_ERROR	1	In	Command rejected because of the key
AUTH_LOGICAL_ADDR_ERROR	1	In	Command rejected because of a logical address

After processing the header of a valid RMAP command received by the target handler, details of the command are output to user logic for authorisation and AUTH_REQUEST asserted. User logic must assert AUTH_GRANT if the command is authorised; if the command is not authorised because of the destination key user logic must assert AUTH_DEST_KEY_ERROR; if the command is not authorised because the destination (or initiator) logical address user logic must assert AUTH_LOGICAL ADDR_ERROR; if the command is not authorised for any other reason then AUTH_GRANT, AUTH_DEST_KEY_ERROR and AUTH_LOGICAL_ADDR_ERROR must be left unasserted. User logic must assert AUTH_RESPONSE when these three inputs have been assigned to notify the codec of the authorisation result.

5.11 TARGET STATUS INTERFACE

This interface is used to report the status of commands received by the target interface.

Signal	Width	I/O	Description
STAT_TARGET_INDICATE	1	Out	Indicates completion of a target command
STAT_TARGET_STATUS	8	Out	Value of the target status register

After the target has processed an RMAP command the codec will place the status result of the operation on STAT_TARGET_STATUS and assert STAT_TARGET_INDICATE for one cycle.

Target Status Name	Value	Meaning
TARGET_SUCCESS	0	Command processed successfully
TARGET_GENERAL_ERROR	1	Unspecified/general error
TARGET_HEADER_EOP_ERROR	2	EOP while reading packet header
TARGET_HEADER_EEP_ERROR	3	EEP while reading packet header
TARGET_PID_ERROR	4	Invalid protocol ID
TARGET_REPLY_ERROR	5	Received reply packet
TARGET_HEADER_CRC_ERROR	6	Header CRC is invalid
TARGET_HEADER_EEP_AFTER_CRC	7	EEP immediately after header CRC
TARGET_PKT_TYPE_ERROR	8	Reserved bit set in type field
TARGET_CMD_TYPE_ERROR	9	Bad read/RMW command flags
TARGET_RMW_DATALEN_ERROR	10	Invalid data length for RMW command
TARGET_CARGO_TOO_LARGE	11	Extra bytes before end of packet marker
TARGET_KEY_ERROR	12	Authorisation rejected due to bad key
TARGET_LOGICAL_ADDR_ERROR	13	Authorisation rejected due to logical address

TARGET_AUTHORISED_ERROR	14	Authorisation rejected for other reasons
TARGET_VERIFY_BUFFER_OVERRUN	15	Too much data for verified-write command
TARGET_DATA_CRC_ERROR	16	Invalid data CRC
TARGET_BUS_REQUEST_ERROR	17	Timeout waiting for BUS_GNT
TARGET_BUS_ERROR	18	BUS_ERROR asserted during bus transfer
TARGET_BUS_TRANSFER_TIMEOUT	19	Bus watchdog timer expired during transfer
TARGET_DATA_EOP_ERROR	20	EOP while reading write/RMW data/mask
TARGET_DATA_EEP_ERROR	21	EEP while reading write/RMW data/mask
TARGET_DATA_EEP_AFTER_CRC	22	EEP after data CRC

A table of all possible STAT_TARGET_STATUS values is given above. The symbolic names are those used in the RMAP IP core VHDL and are defined in `src/vhdl/rmap/rmap_pkg.vhd`.

5.12 Initiator Configuration/Status Interface

The initiator configuration and status interface ports are used to monitor the transaction table and set the time period for the internal 1 µs timer.

Signal	Width	I/O	Description
CFG_1US_CLOCKS	8	In	Number of clock cycles in a 1 µs period.
INI_GOT_ROOM	1	Out	Room for one more RMAP command to be sent
INI_GOT_NONE	1	Out	Set HIGH when there are no outstanding RMAP commands
INI_OUTSTANDING	Variable	Out	Number of RMAP replies which are expected by the RMAP core
DOING_TIMEOUT_CHECK	1	Out	Set HIGH when the RMAP core initiator is check the transaction table for transactions which have timed out and should be deleted.

The length of the INI_OUTSTANDING vector is set dependent on the generic CFG_INI_OUTSTANDING_BITS. The most common method to set CFG_INI_OUTSTANDING_BITS is $Log_2(CFG\_MAX\_COMMANDS)$. If INI_OUTSTANDING is not large enough to represent the maximum number of outstanding commands the lower bits are truncated.

5.13 Initiator Command Interface

The initiator commands interface is used by the host system to send RMAP commands to a target. A pointer to the command parameters in memory is given when a command is requested. If the command parameters indicate that a reply is expected the command cannot be started until there is room in the internal transaction records table.

Signal	Width	I/O	Description
INI_SEND_COMMAND	1	In	Set by the host to initiate an RMAP command.
INI_COMMAND_PTR	32	In	Pointer to the transaction record array in user memory
INI_GENERATE_TID	1	In	Set by the host when the RMAP core should internally generate the transaction identifier and ignore the transaction identifier in the header information record in user memory
INI_COMMAND_DONE	1	Out	Set by the RMAP core when the command has completed
INI_COMMAND_SENT	1	Out	Set HIGH when the command is done and the command was sent without error
INI_COMMAND_TID	16	Out	Transaction identifier of the sent command. This value can either be the generated ID or the value from the header information record.
INI_COMMAND_STATUS	8	Out	Status of the command. The command status types are listed below.
INI_COMMAND_ACK	1	In	Acknowledgement from the host system when command done is set and the host has checked the command status and transaction ID fields.

To initiate a command the host should setup the correct transaction and header information records in user memory and allocate the read and write buffers dependent on the command. If command notification is required, notify send and reply memory locations should also be allocated. When user memory is setup the host should set INI_SEND_COMMAND and set the pointer to the transaction record on INI_COMMAND_PTR. If the transaction identifier should be 1 more than the previous transaction ID the host interface can set INI_GENERATE_TID. When the command has been sent or an error occurred during the command the core will set INI_COMMAND_DONE and the correct status and transaction identifier parameters. If no error was detected in the command header the output INI_COMMAND_SENT is set.

The initiator command status fields output on INI_COMMAND_STATUE are listed below.

Initiator Command Status Name	Value	Description
INITIATOR_ENCODE_SUCCESS	0	Command interpreted correctly
INITIATOR_ENCODE_GENERAL_ERROR	1	An unknown general error occurred during command processing
INITIATOR_ENCODE_TARGET_ADDR_ERROR	2	Target logical address error in the header information record. Outside the range 32-255.
INITIATOR_ENCODE_PROTOCOL_ERROR	3	Protocol identifier is not 0x01 in the header information record.
INITIATOR_ENCODE_PACKET_TYPE_ERROR	4	Error in the packet type field of the RMAP command code in the header information record.
INITIATOR_ENCODE_COMMAND_ERROR	5	Error in the command field of the RMAP command code in the header information record.
INITIATOR_ENCODE_INITIATOR_ADDR_ERROR	6	Initiator logical address in the header information

		record is invalid. Outside the range 32-255.
INITIATOR_ENCODE_RMW_DATALEN_ERROR	7	Error in the read modify write data length in the header information record in user memory.
INITIATOR_ENCODE_BUS_ERROR	8	The host bus controller sets BUS_ERROR when the RMAP core is trying to perform a DMA operation.
INITIATOR_ENCODE_BUS_REQUEST_ERROR	9	Timeout while waiting for BUS_GNT.
INITIATOR_ENCODE_BUS_TRANSFER_TIMEOUT	10	A transfer on the external bus timed out due to the slave setting BUS_WAIT.

5.14 Initiator Reply Interface

The initiator reply interface is used to indicate to the host when reply packet have been received and processed. The reply field, the initiator reply status and the transaction identifier of the command are output after a command has been processed.

Signal	Width	I/O	Description
INI_REPLY_EVENT	1	Out	Reply received event
INI_REPLY_TID	16	Out	Transaction identifier of the RMAP reply packet
INI_REPLY_FIELD	8	Out	Reply field from the RMAP packet
INI_REPLY_STATUS	8	Out	Status of the reply decoder and RMAP reply handler
INI_REPLY_ACK	1	In	Host acknowledge

When a reply is received the initiator reply decoder decodes the reply packet, checking the reply status field and the packet bytes for errors. If the packet is valid the transaction table is searched for the transaction identifier and if a match is found the command can be processed. If a match is not found a status code of lost transaction is recorded. If the RMAP packet is a read or read modify write reply then the packet data is written to user memory over the external bus.

The initiator reply interface status codes are listed in the table below.

Initiator Reply Status Name	Value	Description
INITIATOR_SUCCESS	0	Reply processed successfully
INITIATOR_GENERAL_ERROR	1	Unknown/unspecified error
INITIATOR_HEADER_EOP_ERROR	2	End of packet received during header decoding
INITIATOR_HEADER_EEP_ERROR	3	Error end of packet received during header decoding
INITIATOR_PID_ERROR	4	Protocol identifier not equal to 0x01
INITIATOR_REPLY_TIMEOUT	5	Outstanding reply timed out
INITIATOR_HEADER_CRC_ERROR	6	Header CRC is invalid

INITIATOR_HEADER_EEP_AFTER_CRC	7	Error end of packet error
INITIATOR_PKT_TYPE_ERROR	8	Packet type not a reply
INITIATOR_CMD_TYPE_ERROR	9	Command type unrecognised error
INITIATOR_RMW_DATALEN_ERROR	10	Read Modify Write length error
INITIATOR_CARGO_TOO_LARGE	11	Extra packet bytes after the header CRC and data CRC
INITIATOR_LOST_TRANSACTION	12	Transaction identifier not found in the transaction table
INITIATOR_COMMAND_MISMATCH	13	Reply command type does not match the command type sent in the command packet (1)
INITIATOR_INITIATOR_MISMATCH	14	Reply initiator field does not match the initiator field sent in the command packet (1)
INITIATOR_TARGET_MISMATCH	15	Reply target field does not match the target field sent in the command byte (1)
INITIATOR_DATA_CRC_ERROR	16	Reply data CRC is invalid
INITIATOR_BUS_REQUEST_ERROR	17	Timeout while waiting for BUS_GNT
INITIATOR_BUS_ERROR	18	Host bus controller sets BUS_ERROR when the RMAP core is trying to perform a DMA operation
INITIATOR_BUS_TRANSFER_TIMEOUT	19	Transfer on the external bus timed out due to the slave setting BUS_WAIT
INITIATOR_DATA_EOP_ERROR	20	End of packet received during data processing
INITIATOR_DATA_EEP_ERROR	21	Error end of packet received during data processing
INITIATOR_DATA_EEP_AFTER_CRC	22	Error end of packet received instead of normal end of packet after the data CRC
INITIATOR_HEADER_DATA_AFTER_CRC	23	Data received where the header end of packet should be receive
INITIATOR_BUFFER_TOO_SMALL	24	Reply data length is greater than expected data length

(1) The command, initiator address and target address are checked if CFG_INI_EXTRA_CHECK is equal to 1.

5.15 INITIATOR DELETE/CLEAR INTERFACE

The initiator delete/clear interface is used to delete single or all transactions from the transactions table. If a single transaction is deleted the host specifies the transaction identifier to be deleted. If the transaction identifier is found in the table the transaction is deleted otherwise the table is unchanged. If a clear operation is requested by the host any outstanding transactions are cleared and the table is returned to its original state. The delete interface will wait until the command and reply units are idle before starting a delete or clear

operation. This ensures invalid RMAP command packets are not generated or the core does not try to access invalid memory locations.

Signal	Width	I/O	Description
INI_CLEAR	1	In	Clear all transactions from the records array.
INI_DELETE	1	In	Status of the reply decoder and RMAP reply handler
INI_DELETE_STATUS	1	In	Host acknowledge
INI_DELETE_TID	16	In	Transaction identifier to delete
INI_DELETE_ACK	1	Out	Acknowledge from the RMAP core

The delete status is 1 when a delete is successful or a clear operation has completed or 0 when the transaction identifier is not found in the transaction table.

5.16 Initiator Transaction Table Debug Port Interface

The transaction table stores information on the replies expected by the initiator core. When a reply does not arrive the transaction remains in the table indefinitely or until a times-out occurs and the transaction record is removed by the core. The debug port allows the host to view the outstanding transactions in the table. The interfaces are listed in the following table

Signal	Width	I/O	Description
DBG_RD_REQ_ACCESS	1	In	Request access to the transaction table
DBG_RD_GNT_ACCESS	1	Out	Granted access to the transaction table
DBG_RD_ADDRESS	Variable	In	Address to read from in the transaction table
DBG_RD_EN	1	In	Enable output data from the transaction table
DBG_RD_DATA	32	Out	Returned data from the transaction table address when DBG_RD_EN is set
DBG_CURRENT_PTR	Variable	Out	Pointer to the next free location in the transaction table maintained by the RMAP initiator. The size is dependent on CFG_INI_TRTABLE_ABITS.
DBG_TIMER	34	Out	Transaction timeout timer.

The length of the read address is dependent on the generic CFG_INI_TRTABLE_ABITS.

When the host wishes to use the debug read port it sets DBG_RD_REQ_ACCESS and waits for DBG_RD_GNT_ACCESS to be set. DBG_RD_GNT_ACCESS is set for one system clock cycle to grant access. The debug port retains access to the bus while DBG_RD_REQ_ACCESS is held high after grant. When access to the debug port is completed the host should set DBG_RD_REQ_ACCESS low before performing another read. Commands and replies cannot be processed while the debug read port is active. A

pipelined read port is used. To access the port the address DBG_RD_ADDRESS and DBG_RD_EN should be set and on the next clock cycle the read data DBG_RD_DATA is available.

6 EXTERNAL BUS INTERFACE

The external bus interface is the most important for connecting the RMAP IP core to other IP. The sections below describe the bus operation in more detail and have been taken from the architecture document. See Section 5.8 for the list of bus signals.

6.1 EXTERNAL BUS CONNECTIONS

The external bus is expected to be connected directly to a similar bus system or connected through a bridge. The expected usage of the external bus is illustrated in the following figures.

6.1.1 Direct Connection to Host Bus

In Figure 6-1 the RMAP IP core is connected directly to the host system bus.

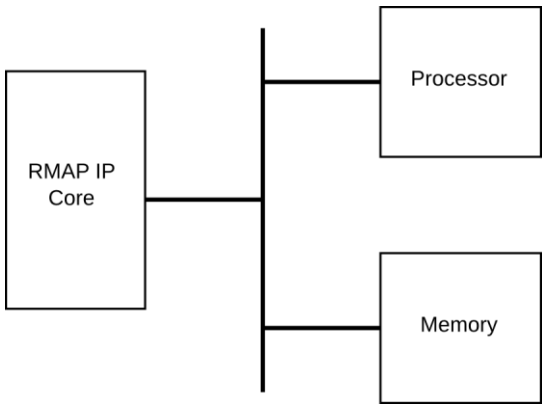


Figure 6-1: RMAP IP Core connected directly to host bus

Target and initiator transactions occur between the RMAP IP Core and the host memory. This is the most efficient usage of the IP core. Control interfaces can be set up between the core and the processor using directly mapped IO or configuration and status registers. In the case of the AMBA bus extra logic external to the RMAP IP core is required to implement all the AMBA bus signals.

6.1.2 Connection to Host Bus Through a Bridge

In Figure 6-2 the RMAP IP core is connected to the host system through a bridge.

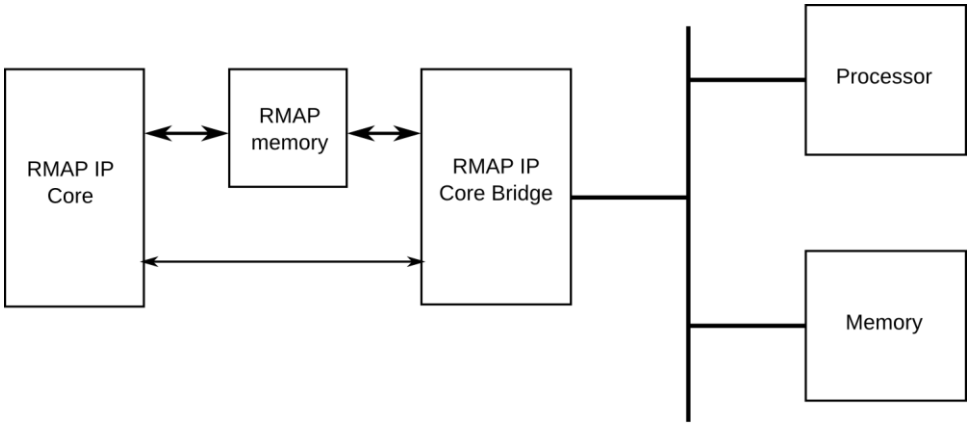


Figure 6-2: RMAP IP Core connected through a bridge

The core has a separate memory space specifically used for target and initiator operations. Initiator and target, header and/or data information is copied between the RMAP memory and the host memory by the RMAP IP core bridge. Control interfaces are set up by the processor and the RMAP IP Core bridge.

6.1.3 Direct Connection to Peripheral/Controller

In Figure 6-3 the RMAP IP core is connected directory to a host peripheral or controller.

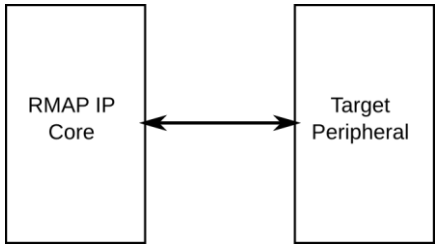


Figure 6-3: RMAP IP Core connected directly to peripheral/controller

In this configuration it is likely the host is a target only and the RMAP IP core is used to configure the host, write data to the host and read data from the host.

6.2 External Bus Operation

An external bus operation consists of an address phase and a data phase.

- In the address phase the bus master sets the transaction type, address and write flag.
- In the data phase the data is presented or captured and the wait/error flags are driven.

Bus operations are pipelined so that the instruction cycle of one operation can overlap with the data cycle of the previous operation.

Note that in the remaining parts of this section, the master has been granted control over the bus via BUS_REQ/BUS_GNT unless otherwise stated.

6.2.1 Bus Transfer States

The bus transfer state driven on BUS_TRANS has one of four possible values:

State	Value	Purpose
IDLE	00	Used when the codec is not issuing the address of an operation.
START	01	Used to mark the first address phase after an IDLE state.
ENABLED	10	Used to mark the address phase of remaining operations of a burst.
BUSY	11	Used to mark an idle phase in the middle of a burst.

From the table above it can be seen that the START and ENABLED states are similar: the START state may be useful to smart buses which split transfers into smaller groups. From the table it can also be seen that IDLE and BUSY states are similar: the BUSY state allows gaps to be inserted into a burst without the need to start a new burst.

6.2.2 General Basic Transfer

A general bus transfer operation is illustrated in Figure 6-4.

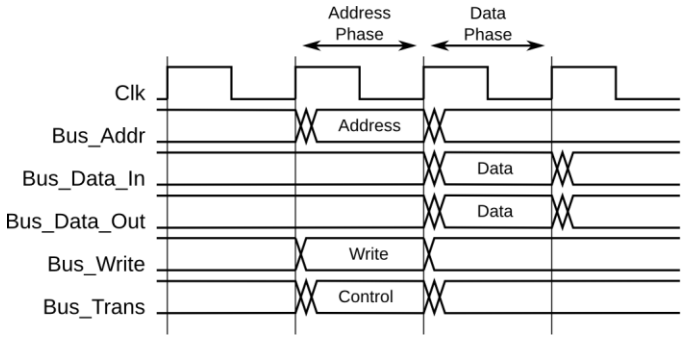


Figure 6-4: External bus basic transfer operation

In a basic transfer with no wait cycle or error:

- The RMAP IP core drives the BUS_ADDR, BUS_WRITE and BUS_TRANS signals during the address phase of the operation.
- In the data phase of a write operation the data is written to BUS_DATA_OUT by the core. In the data phase of a read operation the data is captured from BUS_DATA_IN by the core.

With no waits the data arrives on the cycle after the address phase.

6.2.3 Transfer with Bus Wait

The middle of a transfer with wait states is illustrated in Figure 6-5.

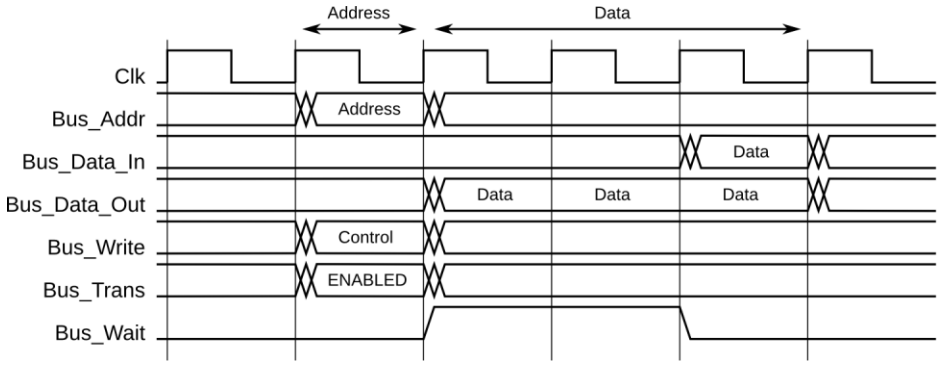


Figure 6-5: External bus operation with wait states

In a transfer with wait cycles:

- The RMAP IP core drives the BUS_ADDR, BUS_WRITE and BUS_TRANS signals during the address phase of the operation.
- In the data phase the BUS_WAIT is set HIGH by the external user memory controller. In a write operation BUS_DATA_OUT is valid until BUS_WAIT is set LOW. In a read operation the data on BUS_DATA_IN is not ready until BUS_WAIT is set LOW.

Due to the BUS_WAIT the data isn't captured until the third data cycle. Bus waits are inserted by the slave when it is unable to supply or capture data during a data phase cycle.

6.2.4 Transfer with Bus Error

The middle of a transfer with a bus error is illustrated in Figure 6-6.

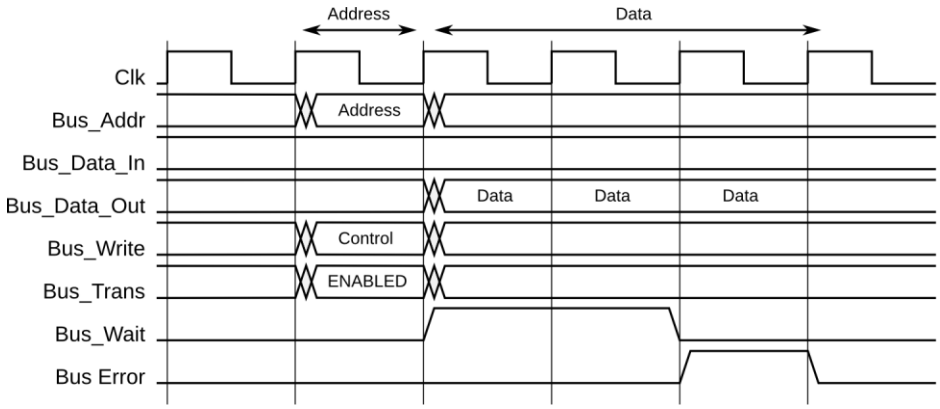


Figure 6-6: External bus operation with bus error

In the bus error terminated transfer shown above:

- The RMAP IP core drives the BUS_ADDR, BUS_WRITE and BUS_TRANS signals during the address phase of the operation.
- In the data phase the BUS_WAIT is set HIGH by the external user memory controller. Eventually the host realises it cannot perform the transfer and sets BUS_ERROR and the transfer is terminated by the RMAP IP core DMA controller.

A bus error may be raised for a variety of reasons including attempts to access memory that isn't mapped, has the wrong access permissions, or perhaps due to a timeout in the slave.

6.2.5 Multiple Transfer Operation

The middle of a multiple transfer operation is illustrated in Figure 6-7.

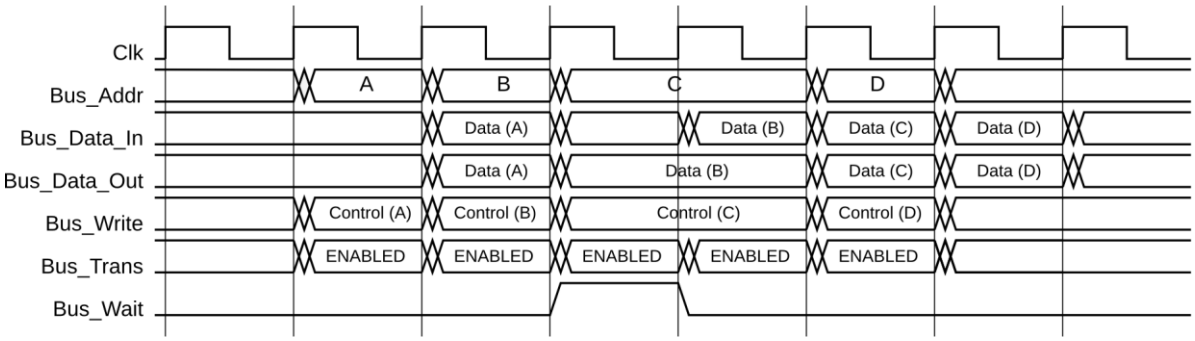


Figure 6-7: External bus multiple transfer operation

In the multiple transfer operation shown above:

- The RMAP IP core drives the BUS_ADDR, BUS_WRITE and BUS_TRANS signals during the address phase of the operation.
- In the data phase of a write operation the data is written to BUS_DATA_OUT by the core. The data is held stable if BUS_WAIT is asserted and the concurrent address phase of the next operation is also held stable.
- In the data phase of a read operation the data is captured from BUS_DATA_IN by the core. The data is invalid if BUS_WAIT is asserted and the concurrent address phase of the next operation is also held stable.
- Transfer B had one wait state.

Notice how the data phase of A overlaps with the address phase of B *etc*.

6.2.6 Transfer with Busy States

The middle of a multiple transfer operation with busy states is illustrated in Figure 6-8.

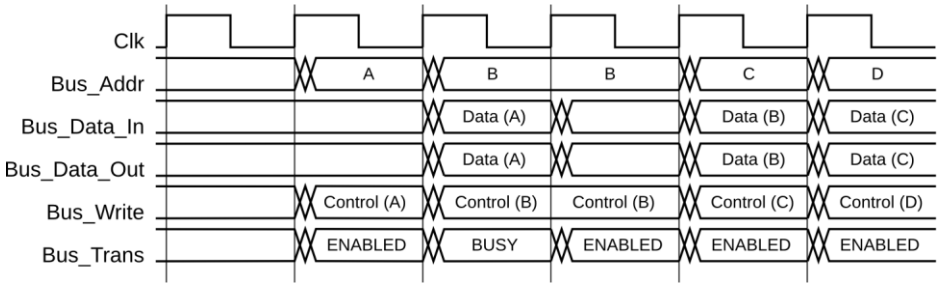


Figure 6-8: External bus operation with BUSY state

In a bus operation with a BUSY state such as that shown above:

- The RMAP IP core drives the BUS_ADDR, BUS_WRITE and BUS_TRANS signals during the address phase of the operation.
- In the data phase of a write operation the data is written to BUS_DATA_OUT by the core. In the data phase of a read operation the data is captured from BUS_DATA_IN by the core.
- If the codec is unable to accept more data it may insert a busy state such as the one for transfer B. The data for transfer A must be provide or accepted but the request for transfer B is delayed by one cycle by setting BUS_TRANS to BUSY.

The codec may use BUSY states to control the flow of data. However, in practice the codec will release the bus when it is unable to provide data for a write or consume data for a read. It will wait until it is able to continue a burst of transfers before requesting the bus again.

6.2.7 Complete Example of a Read Burst

A complete example of a read burst operation is shown in Figure 6-9.

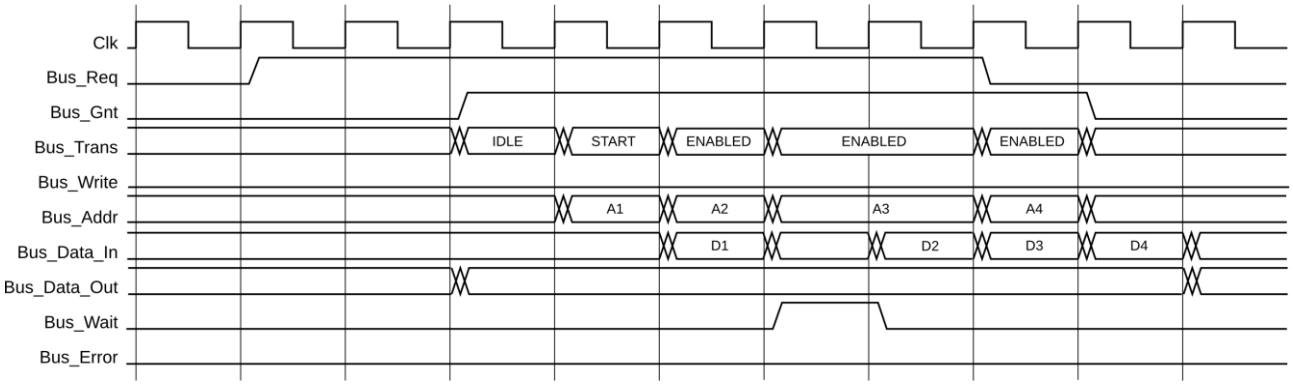


Figure 6-9: Complete example of a read burst

The codec asserts BUS_REQ and waits for BUS_GNT to be asserted in response. After a couple of cycles the bus is granted to the codec. In this example the codec marks the bus IDLE during the cycle in which it was granted but it need not do this (it could issue a START immediately).

On the next cycle the codec issues a read for address A1 using the START state because this is the first operation of a burst. On the next cycle the data D1 can be captured for address A1; at the same time the codec issues a read for address A2 using state ENABLED for this and subsequent operations. On the next cycle the codec expects the data D2 to arrive for address A2 and issues a read for address A3. However, the slave is unable to provide D2 and asserts BUS_WAIT for one cycle. The codec holds BUS_TRANS, BUS_WRITE and BUS_ADDR stable for another cycle during which the expected data D2 arrives.

The last operation of the burst is to read from address A4. However, since the bus is pipelined the BUS_GNT signal will remain asserted for the next address cycle so the codec is permitted to deassert BUS_REQ at the same time as issuing the last read operation for address A4. Because the bus is still granted during the address phase of the A4 operation the data D4 is expected in the following cycle even though BUS_GNT has been deasserted.

Note that if the data D4 was not ready the slave would assert BUS_WAIT as normal and the codec would wait for the data to become ready. If data D3 was not ready then the bus master may not deassert BUS_GNT until the end of address phase A4.

6.2.8 Complete Example of a Write Burst

A complete example of a read burst operation is shown in Figure 6-10.

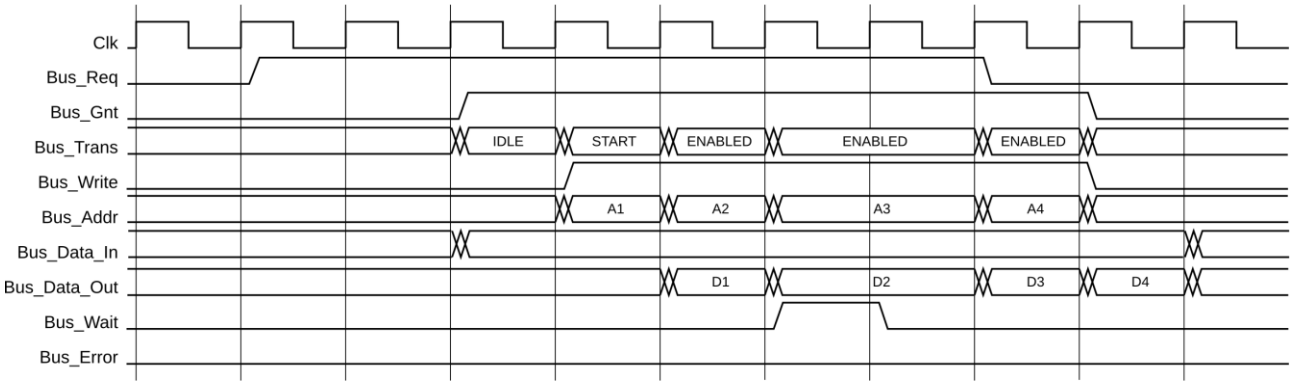


Figure 6-10: Complete example of a write burst

The codec asserts BUS_REQ and waits for BUS_GNT to be asserted in response. After a couple of cycles the bus is granted to the codec. In this example the codec marks the bus IDLE during the cycle in which it was granted but it need not do this (it could issue a START immediately).

On the next cycle the codec issues a write for address A1 using the START state because this is the first operation of a burst. On the next cycle the data D1 is driven for address A1; at the same time the codec issues a write for address A2 using state ENABLED for this and subsequent operations. On the next cycle

the codec drives data D2 for address A2 and issues a write for address A3. However, the slave is unable to accept D2 and asserts BUS_WAIT for one cycle. The codec holds BUS_TRANS, BUS_WRITE, BUS_ADDR and BUS_DATA_OUT stable for another cycle during which the data D2 is captured by the slave.

The last operation of the burst is to write from address A4. However, since the bus is pipelined the BUS_GNT signal will remain asserted for the next address cycle so the codec is permitted to deassert BUS_REQ at the same time as issuing the last write operation for address A4. Because the bus is still granted during the address phase of the A4 operation the data D4 will be driven in the following cycle even though BUS_GNT has been deasserted.

Note that if the data D4 could not be captured the slave would assert BUS_WAIT as normal and the codec would wait for the data to be captured. If data D3 could not be captured then the bus master may not deassert BUS_GNT until the end of address phase A4.

7 INITIATOR

The RMAP Initiator Handler uses several memory structures inside the RMAP IP core and also inside initiator user memory. The structures are used to control the passing of commands from initiator user memory to the RMAP IP core and the passing of replies from the RMAP IP core to initiator user memory.

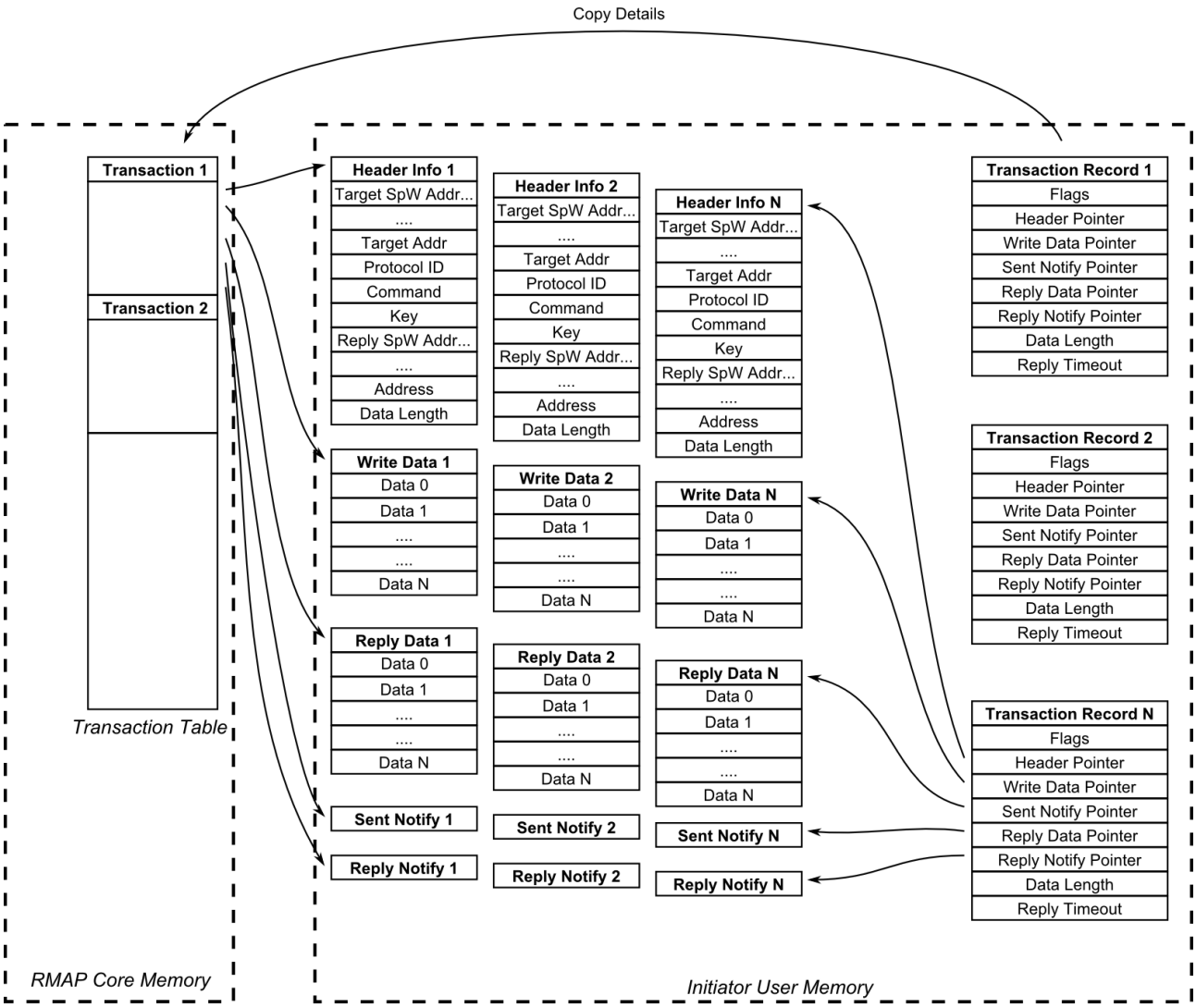


Figure 7-1: Initiator Data Structures

In the initiator user memory there are four possible memory areas or buffers associated with each RMAP command: transaction details record, header information, write data, and reply data.

The transaction details array holds the following information: pointer to command header in user memory, pointer to any data to be sent with a write or read-modify-write command, pointer to memory location to be written for sent notification, pointer to space for a reply to a read or read-modify-write command, pointer to memory location to be written for reply notification, length of data to be read or written, reply time-out value and transaction status.

The header information buffer holds the RMAP command header information including the Target SpW Address and the Reply Address.

The write data buffer holds any data to be sent with a write or read-modify-write command.

The reply data buffer is reserved space into which any data associated with a read or read-modify-write command will be written.

7.1 SENDING A NEW COMMAND

To send an RMAP command the user sets up the header of the command in a header information buffer, any data to be sent with the command in a write data buffer and space for any reply in a reply data buffer. The user then creates a transaction record with pointers to the header information buffer, write data buffer and reply data buffer along with information about the amount of data in these buffers. It also provides pointers to memory locations (or registers) where sent and reply notifications are to be made. Finally it adds into the transaction record a reply time-out value (0 = no wait, 0x00000001-0xFFFFFFFE wait time in µs, 0xFFFFFFFF = infinite wait). Once the transaction record is complete the initiator user application informs the RMAP IP core that is has an RMAP command to send and passes the RMAP IP core a pointer to the corresponding transaction record.

If the transaction details record flags field indicates that the command is expecting a reply the command is not started (sent) until there is room for another transaction in its outstanding transaction array. The RMAP IP core will then send the command by copying the header information from user memory to the SpaceWire interface, adding any detail necessary (e.g. header CRC). The header information is copied twice so the information can be checked and then sent. Any errors which are detected in the header are recorded and output on the status interface and to the notify sent register, if used.

If there is any write data to be sent this will be copied from the write data buffer in user memory to the SpaceWire interface and appending the data CRC. Finally an EOP marker will be added to complete the packet. The initiator user application will be informed that the command has been sent by the RMAP IP core writing the transaction ID and status to the memory location specified by the sent notify pointer in the transaction details array element.

7.2 RECEIVING A REPLY

When an RMAP reply is received the RMAP IP core searches the outstanding transaction array for an entry with a transaction identifier that matches the transaction identifier of the reply. Assuming there is a match the RMAP IP core then writes any data from a read or read-modify-write reply to the user memory location specified by the reply data pointer for the corresponding entry in the transaction details array. The RMAP IP core writes the transaction identifier and status to the memory location specified by the reply notification

pointer in the transaction details array entry. When this has been done the relevant entry in the outstanding transaction array is cleared freeing it for use by another RMAP transaction.

7.3 TRANSACTION DETAILS RECORD

The transaction details record is setup in user memory by the host application when it wishes to send an RMAP command. The format of the command is shown in Figure 7-2 Transaction Details Record Memory SetupFigure 7-2. The flags field is a bit mask which holds properties on the transaction record.

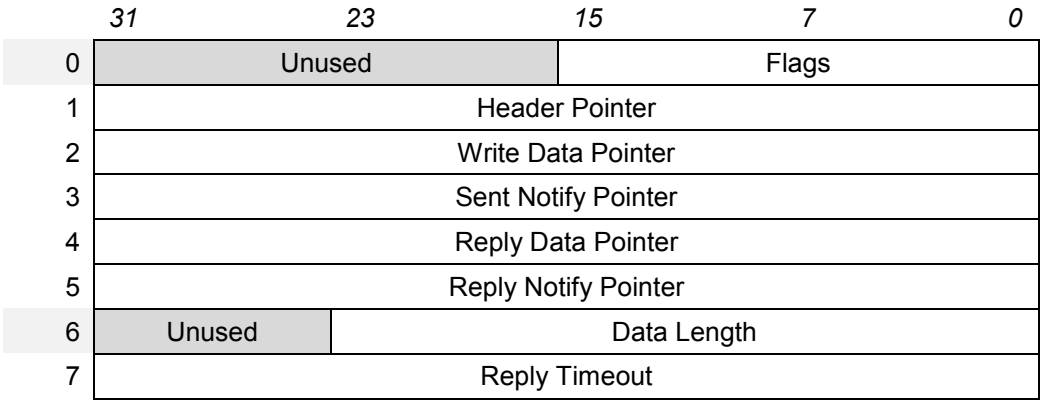


Figure 7-2 Transaction Details Record Memory Setup

The transaction details record fields are described in the table below.

Field	Size (bits)	Description
Flags(7:0) – Target SpW Length	8	Number of target SpaceWire addresses. Up to 256 SpaceWire path addresses can be added to the front of an RMAP command
Flags(9:8) – Reply Addresses	2	Number of SpaceWire reply addresses to send in the RMAP command. The actual number is multiplied by 4 when
Flags(10) – MSB First	1	Send the RMAP data bytes most significant byte first
Flags(11) – Bit swap	1	Reverse the bits in the RMAP data byte
Flags(12) – Notify Sent	1	When set the notify sent memory location is valid
Flags(13) – Notify Reply	1	When set the notify reply memory location is valid
Flags(14) – DMA Header Inc	1	When set the external bus address is incremented when accessing the header information array
Flags(15) – DMA data Inc	1	When set the external bus address is incremented when accessing the write and reply data buffers
Flags(16) – Reply Expected	1	When set a reply is expected and a location in the transaction table is required
Header Pointer	32	Pointer to the header information buffer in user memory
Write data pointer	32	Pointer to the write data buffer in user memory which contains data to be sent in a write or RMW command for

		writing into target memory.
Sent notify pointer	32	Pointer to the memory location (or register) which is to be written to when the RMAP command has been sent. The transaction identifier and status are written to the memory location.
Reply data pointer	32	A pointer to the buffer in user memory reserved for any data received in a reply from the RMAP command.
Reply notify pointer	32	Pointer to the memory location (or register) which is to be written to when the RMAP reply has been received and any associated data written to memory. The transaction identifier and status are written to the memory location.
Data Length	24	The amount of data to be read in a read command or written in a write command. For a read-modify-write command this field holds the length of the data plus mask, which is twice the size of the data to be read.
Reply time-out	32	The amount of time to wait for a reply to an RMAP command. 0x00000000 - do not wait. 0x00000001 to 0xFFFFFFFE - time-out time in µs. 0xFFFFFFFF - wait forever.

Table 7-1 Transaction Record Fields

7.4 HEADER INFORMATION RECORD

The header information record holds information on the RMAP command parameters to be sent. The header information record is stored in memory as shown in Figure 7-3. In the example there are 4 target SpaceWire addresses and 1 block of reply SpaceWire addresses.

	31	*23*	*15*	*7* *0*
0	Target Path Addr 1	Target Path Addr 2	Target Path Addr 3	Target Path Addr 4
1	Target Address	Protocol ID	Instruction	Key
2	Reply Path Addr 1	Reply Path Addr 2	Reply Path Addr 3	Reply Path Addr 4
3	Initiator Address	Transaction ID 1	Transaction TID 0	Extended Address
4	Address 3	Address 2	Address 1	Address 0
5	Data Length 2	Data Length 1	Data Length 0	Unused

Figure 7-3 Header Information Record Setup

The header information is checked by the initiator core before it is sent.

Field	Size	Description
Target SpaceWire Address	Variable	Target SpaceWire path addresses. Any leading zeros are removed before transmission.
Target Logical Address	1	A logical address of the target that the RMAP command is intended for.
Protocol Identifier	1	0x01 for RMAP
Instruction	1	The RMAP packet type and command.
Key	1	A value that is checked by the target for security.
Reply Address	Variable	Reply path addresses
Initiator logical address	1	A logical address of the initiator.
Transaction identifier	2	The transaction identifier value to be sent in the command. If automatic transaction identifier generation is selected the RMAP Initiator Handler will ignore the transaction identifier value in the header information buffer.
Extended address	1	The most-significant 8-bits of the address of the memory that is to be accessed in the target.
Address	4	The least-significant 32-bits of the address of the memory that is to be accessed in the target.
Data length	3	The amount of data to be written to, read from or read-modify-written in target memory.

Table 7-2 Header information record fields

7.5 NOTIFY SENT/REPLY RECORD

The notify sent and reply records are used to store the transaction ID and status of the commands and replies which are processed by the initiator RMAP core.

The command sent notify record is shown in Figure 7-4

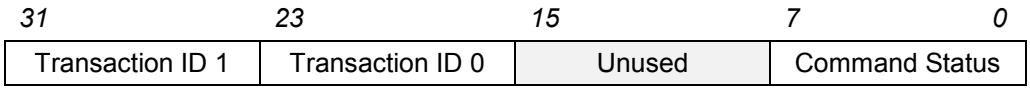


Figure 7-4 Notify Sent Record

The command sent notify record is shown in Figure 7-5

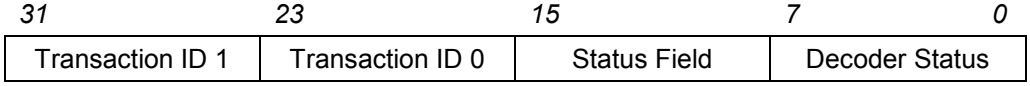


Figure 7-5 Notify Reply Record

The status field is a copy of the reply status from the RMAP command and the reply decoder status is the reply status as shown in §5.14.

7.6 Debug Read Port Details

The debug read port can be used to view the contents of the transaction records table in the RMAP IP core memory block. Each record in the transaction memory block is setup when a reply is expected. A transaction record requires 7 locations in the transaction memory block and each transaction record is sequential, i.e. transaction 1 is at address 0 and transaction 2 is at address 7 and so on.

	31	*23*	*15*	*7*	*0*
0	Core Field	Core Field	Core Field	Status/Used	
1	Unused	Core Field	Transaction ID 1	Transaction ID 0	
2	Unused	Instruction	Initiator Address	Target Address	
3	Reply Data Pointer				
4	Reply Notify Pointer				
5	Timeout Used	Data Length			
6	Core Field				

Figure 7-6 Debug port transaction record details

The core fields are fields which are used only by the RMAP core.

Bit 0 of the Status/Used field indicates if the transaction record is in use and represents a valid transaction record for an outstanding reply. Bit 1 indicates if the command has been sent and bit 2 indicates if a reply has been received. Note a reply may be received before the complete command has been sent, for example a long write operation may not be authorised so the target can return an RMAP reply packet with the status set to not authorised before the command packet has completed.

Bit 2 of the Timeout Used field indicates if the transaction timeout is active and if the transaction will be deleted by the core after the specified timeout period has expired and no reply has been received.

The other fields are copied directly from the transaction record and header information record as the command is sent.

7.7 Reply Packet Timeout Detection

A timeout field in the transaction details record can be set to force the initiator to stop waiting for the packet reply after a period of time. When a timeout occurs the initiator frees up the location in the transaction table and discards the reply if it eventually is returned. The reply timeout field details are listed in Table 7-1.

The configuration generics CFG_INI_TIMEOUT_CHECK_WAIT and CFG_INI_TIMEOUT_CHECK_MAX, defined in sections 5.1.25 and 5.1.26, controls the timing of initiator timeout checks. The initiator does not hold a counter for each outstanding transaction which could be costly for device area. Instead the absolute timeout value for each packet is held in the internal transaction table array. When a transaction is started the value of the transaction timeout in the transaction details array is added to the value of the internal timer and then stored in the internal array. The value is then checked at a regular intervals to determine if it exceeds the current value of the internal timer and if so a timeout is detected. Note using this method multiple expected replies may have timed out and remain in the transaction table but they will eventually be caught when the next timeout check is performed. It is also possible for the reply packet to be received when an expected reply has already timed out but has not been caught. This situation is guarded against by checking the timeout field when a reply is matched in the transaction table before the reply is acted upon.

The regular interval for timeout checks is controlled by the configuration signal CFG_INI_TIMEOUT_CHECK_WAIT and the duration of the timeout check is controlled by CFG_INI_TIMEOUT_CHECK_MAX. When timeout check is being performed by the initiator no command or reply packets can be processed so CFG_INI_TIMEOUT_CHECK_MAX should be kept to a few micro seconds. The interaction of CFG_INI_TIMEOUT_CHECK_WAIT and CFG_INI_TIMEOUT_CHECK_MAX is shown in Figure 7-7.

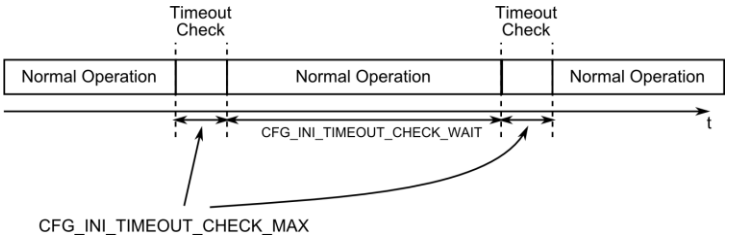


Figure 7-7 Timeout check settings interaction

8 VERIFICATION

Verification of the RMAP core is performed using the RMAP core validation test-bench. The validation test-bench it is not provided with the core, instead a cut down test-bench which is used to allow the end user to test a netlist or placed and routed model is supplied.

Two end user test-benches are supplied with the core. The first test-bench tests the functions and the interfaces of the core where all interfaces are available i.e. the implementation is the RMAP IP core alone. The second test-bench checks packets sent and received by the core when only the SpaceWire interface is available and the other RMAP core interfaces are embedded in the end user system. The user defines the packets to be sent and received according to the characteristics of their system.

9 SYNTHESIS

9.1 CLOCK PERFORMANCE

The RMAP CODEC IP has one system clock input (Clk) which clocks all flip-flops in the design except the receive clock domain of the SpaceWire link. The SpaceWire transmitter can also be clocked from a separate clock input on the core Txclk dependent on the SpaceWire link configuration setting CFG_BITCLK in src/vhdl/spw/spwrlink_pkg.vhd.

A typical way to implement the RMAP core design is to run the system clock (Clk) at the byte rate of the system and use a separate transmit clock to transmit the bytes at the required bit rate. For example a system which processes RMAP data at 20 Mbytes/s requires a 100 MHz transmit clock to transmit the byte data at 100 Mbps, taking into account the SpaceWire data character length of 10 bits.

9.2 SYNTHESIS RESULTS

The configuration of the RMAP core used to get synthesis results are listed in Table 9-1.

Generic Name	Value Used	Comments
CFG_TECH	TECH_MEM_PROASIC TECH_MEM_AXCELERATOR	Correct memory type for architecture
CFG_INITIATOR_EN	1	Initiator enabled
CFG_TARGET_EN	1	Target enabled
CFG_TARGET_MSB_FIRST	1	
CFG_TARGET_BITSWAP	0	
CFG_WORD_SIZE	4	32 bit bus size
CFG_TARGET_VERIFY_BUF_ABITS	8	256 byte verify buffer
CFG_TRGT_FIFO_OUT_ABITS	6	64 word buffer
CFG_TRGT_FIFO_IN_ABITS	6	64 word buffer
CFG_TRGT_BURST_SIZE	64	64 word per DMA transaction
CFG_TRGT_WATCHDOG_TIMEOUT	1024	
CFG_TRGT_EN_WATCHDOG	1	Watchdog mode enabled
CFG_REQ_TIMEOUT_NBITS	8	
CFG_SEND_REPLY_ON_EEP_AFTER_CRC	1	Return packet
CFG_SEND_REPLY_ON_RESERVED_PKT	0	
CFG_ALLOW_LOOPBACK	1	Loopback mode enabled
CFG_DEMUX_ROUTE_RESERVED-_TO_TARGET	0	
CFG_DEMUX_ROUTE_REPLIES-	0	

_TO_TARGET		
CFG_INI_MAX_COMMANDS	36	Use maximum table size
CFG_INI_TRTABLE_ABITS	8	
CFG_INI_OUTSTANDING_BITS	8	
CFG_INI_EXTRA_CHECKS	0	No extra packet checks
CFG_INI_CODEC_MSB_FIRST	1	
CFG_INI_CODEC_BITSWAP	0	
CFG_INI_TIMEOUT_CHECK_WAIT	50	
CFG_INI_TIMEOUT_CHECK_MAX	20	
CFG_INI_BURST_SIZE	64	64 word burst size
CFG_INI_WATCHDOG_TIMEOUT	1024	
CFG_INI_EN_WATCHDOG	1	
CFG_INI_FIFO_OUT_ABITS	6	64 word buffer
CFG_INI_FIFO_IN_ABITS	6	64 word buffer

Table 9-1 "rmap_codec_ip.vhd" Generic settings for area usage figures

The results of synthesis runs on the Mentor Graphics Precision synthesiser are given below. The figures for "Kernel Only" are the RMAP core with imitator and target but without the SpaceWire link interface.

Model	AX2000			Spartan3E 1600	ProASIC3E1500
	FF	Comb	Modules	Slices	Tiles
Core	2957	6249	9206 (29.06%)	3095 (20.97%)	11261 (29.33%)
Initiator Only	2029	4434	6463 (20.04%)	2213 (15.00%)	7987 (20.80%)
Target Only	1425	2962	4464 (13.84%)	1134 (7.69%)	4576 (11.92%)
Kernel Only	2599	5634	8233 (26.20%)	2584 (17.52%)	10206 (26.58%)

Table 9-2 Area usage of RMAP core synthesised with Mentor Graphics Precision

The results of synthesis runs on the Synplicity Synplify synthesiser are given below. In Synplify the syn_hierarchy attribute is set to "firm", syn_encoding is set to "gray". The figures for "Kernel Only" are the RMAP core with imitator and target but without the SpaceWire link interface.

Model	AX2000			ProASIC3E1500
	FF	Comb	Modules	Tiles
Core	3665	6955	10620 (33%)	19420 (50%)
Initiator Only	2491	5040	7531 (24%)	15365 (40%)
Target Only	1845	2745	4590 (15%)	6238 (16%)
Kernel Only	2491	5040	7531 (24%)	17796 (46%)

Table 9-3 Area usage of RMAP core synthesised with Synplicity Synplify

9.3 AREA OPTIMISATION

The RMAP core units designed using finite state machines (FSM), therefore the type and settings of the synthesis tool will greatly affect the area usage. To reduce area further the generic settings of the RMAP core are modified as listed in Table 9-4.

Generic Name	Initial Value	Value Used
CFG_TARGET_VERIFY_BUF_ABITS	*8*	4
CFG_TRGT_FIFO_OUT_ABITS	*6*	3
CFG_TRGT_FIFO_IN_ABITS	*6*	3
CFG_TRGT_BURST_SIZE	*64*	2
CFG_INI_MAX_COMMANDS	*36*	2
CFG_INI_TRTABLE_ABITS	*8*	4
CFG_INI_OUTSTANDING_BITS	*8*	2
CFG_INI_TIMEOUT_CHECK_MAX	*50*	20
CFG_INI_BURST_SIZE	*64*	4
CFG_INI_FIFO_OUT_ABITS	*6*	3
CFG_INI_FIFO_IN_ABITS	*6*	3

Table 9-4 Generics modified for reduced area consumption results

To reduce the area usage of the core the FSM encoding should be set to gray or binary. Note binary and gray encoding can increase the number of combinatorial cells used. One hot encoding will increase the size of the core but also increase the performance.

Model	AX2000	ProASIC3E1500
	Modules	Tiles
Core, one hot FSM	11184 (34.7%)	11397 (29.7)%
Core, gray coded FSM	10352 (32.1%)	11794 (31%)

Table 9-5 Area optimisation synthesis results (Mentor Graphics Precision)

The gray coded FSM has the precision “setup_design –compile_for_area” option set in the command line.

Note: one hot encoding produces better results for ProASIC3E devices but may reduce performance for “safe mode” state machines. Area optimisation does not provide a significant area decrease therefore it is not recommended.

9.4 MEMORY BLOCKS AND FIFOS

The RMAP IP core block uses a number of memory blocks and FIFO structures to store and buffer data.

The memory blocks used in the core are listed in Table 9-6.

Memory block	Description
SpaceWire transmit FIFO	FIFO for SpaceWire transmit data. This FIFO can be relatively small, as little as 8 locations, as its purpose is to synchronise data to the transmit clock. The FIFO size is configurable in src/vhdl/spw/spwrlinkwrap.vhd
SpaceWire receive buffer	Receive buffer to store received SpaceWire characters. The receive buffer size is set by the constant CFG_RXBUF_ADDRLEN in src/vhdl/spw/spwrlink_pkg.vhd
DMAC input/output FIFO	Each DMA controller has an input and output FIFO to facilitate burst transfers over the external bus. The size of the DMAC FIFO is configurable through the VHDL generics.
Verify buffer	The verify buffer is used to store data from an RMAP packet until the data CRC can be checked and the data can then be written to external memory. The size of the verify buffer can be set through the generic CFG_TARGET_VERIFY_BUF_ABITS.
Transaction Table	The transaction table is used to store information on RMAP commands sent by the initiator. The table size is set through the generic CFG_INI_TRTABLE_ABITS.

Table 9-6 Core memory blocks

The SpaceWire transmit FIFO uses a two clock dual ported memory block to store SpaceWire data. The memory block is implemented in src/vhdl/async_fifo/async_memblock.vhd. Three models are available, each with a different filename. The first model builds the memory block using flip-flop resources, the second model infers a dual port RAM on most synthesisers and the third block generates a Xilinx block RAM component.

The other memory blocks are implemented using a single clock dual ported memory block implemented in src/vhdl/sync_fifo/sync_memblock.vhd. Again three models are available, each with a different filename. The first model builds the memory block using flip-flop resources, the second model infers a dual port RAM on most synthesisers and the third block generates a Xilinx block RAM component.

The memory structures available in the VHDL code are listed in Figure 9-1.

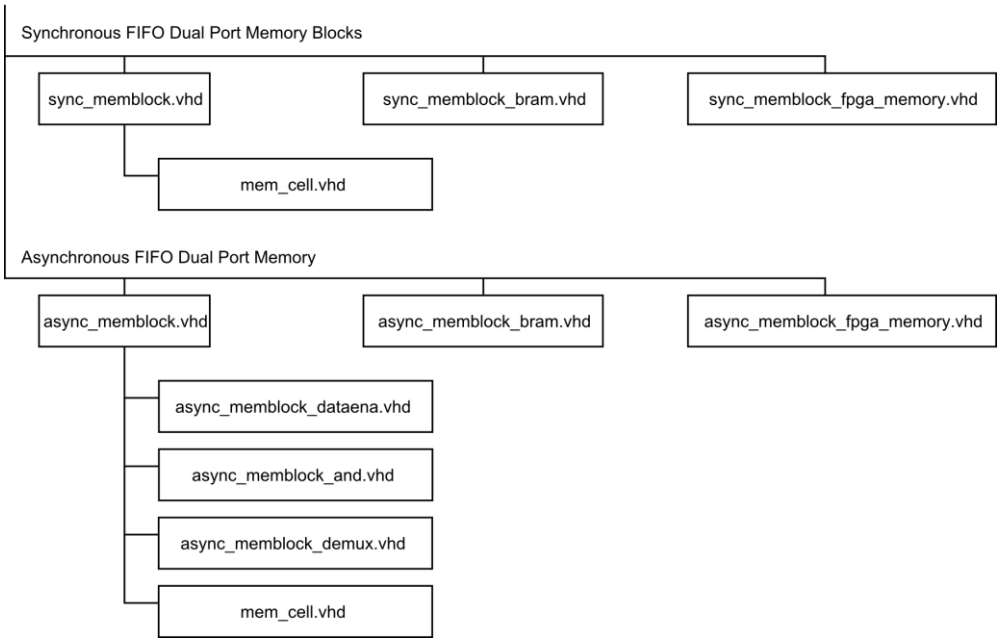


Figure 9-1 Synchronous and asynchronous dual port memory blocks

A description of each VHDL file is given in Table 9-7.

VHDL file	Description
mem/mem_cell.vhd	Register with write enable intended for memory block build with flip-flop banks.
sync_fifo/sync_memblock.vhd	Single clock, dual port memory built from mem_cell blocks.
sync_fifo/sync_memblock_bram.vhd	Single clock, dual port memory using Xilinx BRAM components. Uses RAMB4_S16_S16 which is compatible with Virtex and Spartan designs.
sync_fifo/sync_memblock_fpga_memory.vhd	Single clock, dual port memory using VHDL code to instantiate technology memory blocks.
async_fifo/async_memblock.vhd	Two clock, dual port memory build from mem_cell blocks and with additional asynchronous output filtering to the memory output register.
async_fifo/async_memblock_and.vhd	AND gate for output data multiplexer.
async_fifo/async_memblock_dataena.vhd	Output data multiplexer build from AND gates and OR gates.

async_fifo/async_memblock_demux.vhd	Binary to one-hot output register selection block. Selects the row of memory which should be enabled to the output register through the data enabler AND gates.
async_fifo/async_memblock_bram.vhd	Two clock, dual port memory using Xilinx BRAM components. Uses RAMB4_S16_S16 which is compatible with Virtex and Spartan designs.
sync_fifo/sync_memblock_fpga_memory.vhd	Two clock, dual port memory using VHDL code to instantiate technology memory blocks.

Table 9-7 VHDL memory block files.

When synthesising, the correct technology memory block should be added to the synthesis project.

9.5 SEU PROTECTION

SEU protection is not provided in the RMAP IP core model. It is expected that the fabric of the FPGA or ASIC technology will provide SEU protection for synchronous elements in the design (flip-flops).

Typically memory blocks are not protected, therefore they should either be implemented as flip-flops; or a drop in replacement for the single and dual clocked memory blocks should be used in the final synthesised model. For example memory blocks with error detection and correction (EDAC) using error correcting codes (ECC) are provided with the Actel Libero and designer toolchain.

Critical memory blocks for SEU protection in the design are the verity buffer, transaction table and DMA controller FIFOs. Scrubbing may also be desirable in the transaction table memory block as the transactions may be in the memory block for a long period of time and be susceptible to repeated single events before the data is read from the memory. Scrubbing support is not compiled in the transaction table controller, and may be provided in a later version of the core.

The SpaceWire interface transmit and receive FIFOs are also critical but if the RMAP protocol is used the packet data is protected by header and data CRCs. In this case SEU protection may not be required, dependent on the initiator hosts ability to resend an RMAP command on a corrupted data byte in the RMAP packet.

9.6 SYNTHESIS EXAMPLE FOLDER “DESIGN/SYNTH_EXAMPLE/”

A Mentor Graphics precision script and example spwrlink_pkg.vhd configuration file are available in the design/synth_example folder. The script synthesises the core for the Actel AX2000, Actel ProASIC3E and Xilinx Spartan3E technologies.

10 DOCUMENT CHANGES

The changes made this document are listed in table 10.1

(Issue 1.6 to Issue 1.7)	
Section/Reference	**Change**
	Update RMAP and SpaceWire standard document reference numbers
(Issue 1.5 to Issue 1.6)	
Section/Reference	**Change**
9.2	Update synthesis results after initiator reply decoder change.
(Issue 1.4 to Issue 1.5)	
Section/Reference	**Change**
5.16	Add additional transaction debug/status signals
(Issue 1.3 to Issue 1.4)	
Section/Reference	**Change**
2	Add note on CFG_ALLOW_LOOPBACK=1 results in long static timing path.
3	design/units directory becomes design/synth_example
4	Update architecture diagram with FIFOs and verify control path flow
4.x	Add section headings for architecture blocks. Add sections on receive and transmit FIFOs, transaction table controller, delete controller and target verify controller.
5	Add overview table of configuration generics.
5	CFG_ALLOW_LOOPBACK default set to 0 and note added on static timing results.
6	Fix diagram background when converting to PDF and missing references
9.2	Use AX2000 and remove references to AX1000 usage
9.4	Add descriptions of memory block VHDL files and hierarchy
9.6	Brief description of design/synth_example folder

(Issue 1.2 to Issue 1.3)	
Section/Reference	**Change**
3	Description of design/units directory added to source files section
5.1.6	Fix text in section on CFG_WORD_SIZE so note describes correct use of CFG_WORD_SIZE when initiator en = 1
5.12	Fix description of doing time-out check
8	Add section on verification of the core in the user manual
9.2	Add configuration of the RMAP core for usage estimate
9.6	Add information on synthesis of units and pie charts for those figures
(Issue 1.1 to Issue 1.2)	
Section/Reference	**Change**
4 (CMC)	Correct initiator bit order generic descriptions and usage
(Issue 1.0 to Issue 1.1)	
Section/Reference	**Change**
4 (CMC)	Add initiator interfaces
6 (CMC)	Add initiator data structures and procedures
7 (CMC)	Add synthesis information

Table 10-1 Document changes