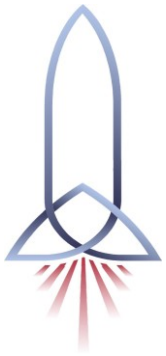
School of Computing
UNIVERSITY OF DUNDEE

SPACE SYSTEMS RESEARCH GROUP



Space
Technology
Centre
University of Dundee

SpaceNet – RMAP IP Core

End User Test-bench Manual

Revision: Issue 1.1

Date: 12th February 2009

ESA Contract Number 220774-07-NL/LvH

Ref: RMAP IP WP2-400.7-1

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

Document Authors

Chris McClements (CMC)

Document Change Log

Date	Revision No	Comments
30h January 2009	Issue 1.0	CMC: First issue
12th February	Issue 1.1	CMC: Normal and Embedded Test-bench CMC: Detailed information on end user test-bench expanded.

A comprehensive list of the changes which have been made to this document in each revision is provided in section 6.

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.

This document gives an overview of the test-bench architecture and user instructions to run the end user test-bench and check the results. The end user test-bench is intended to run a series of packet checks on an RTL, netlist or placed and routed model of a design where the RMAP core has been embedded in the logic of the design.

1.2 GUIDE TO DOCUMENT

Section 2 defines introduces the end user test-bench and the embedded end user test-bench.

Section 3 describes the test-bench environment

Section 4 describes the end user test-bench in detail giving descriptions of the test-bench architecture and the tests performed.

Section 5 describes the embedded end user test-bench in detail giving descriptions of the test-bench architecture and the tests performed.

Section 6 lists the changes to the document.

1.3 ACRONYMS AND ABBREVIATIONS

UUT Unit Under Test

SpW SpaceWire

RMAP Remote Memory Access Protocol

FIFO First In First Out

RTL Register Transfer Level

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.

1.5 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-E-ST-50-11C	SpaceWire Protocols, Draft 1.3 – July 2008
AD2	RMAP IP WP2-100.1	SpaceNet RMAP IP Core Requirements 6th Feb 2008
AD3	RMAP IP WP2-100.2	SpaceNet RMAP IP Core Functional Specification 4th April 2008
AD4	RMAP IP WP2-100.2	SpaceNet RMAP IP Core Interface Specification 4th April 2008
AD5	RMAP IP WP2-400.3	SpaceNet RMAP IP Core User Manual

2 END USER TEST-BENCH SUMMARY

The end user test-bench is intended to run a series of tests on an RTL, netlist or placed and routed model or a design where the RMAP core has been embedded in the logic of the design. The end user test-bench is not intended to be a complete validation of all the functions of the RMAP core. A separate validation test-bench which checks the functions of the core is not supplied with the user release.

Two types of testing can be performed in the end user test-bench, the first test-bench provides tests to check the interfaces of the core, Figure 2-1, and the second provides tests to check the response of the core when it is embedded in the end users system and the only interface available is the SpaceWire interface, Figure 2-2.

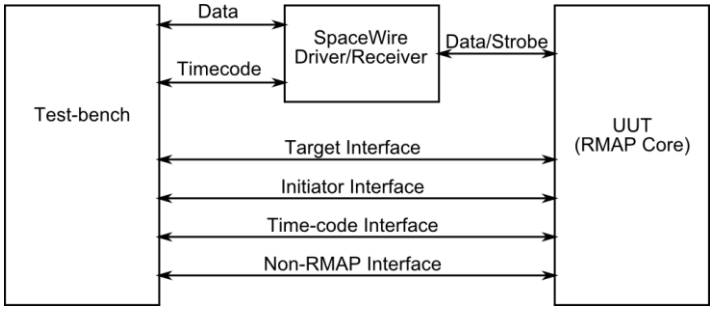


Figure 2-1 End user test-bench (TB_CORE) with access to the UUT interfaces. The test-bench can check all ports on the RMAP core, enabling it to initiate commands and authorise target commands.

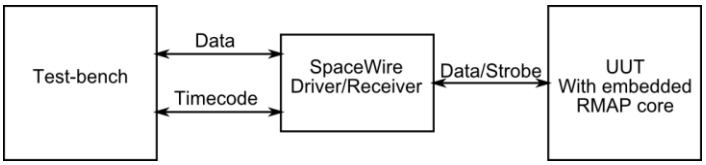


Figure 2-2 End user test-bench (TB_EMB) with embedded RMAP IP core inside the end users system. The test-bench interface to the system is through the SpaceWire interface. Command initiation and target authorisation must be performed by the embedded system.

2.1 END USER TEST-BENCH FLOW

A typical flow for testing of the RMAP core in a target system is given in the following list:

- Run the RMAP core end user test-bench to check the behaviour of the RMAP core is correct and gain experience with the test-bench.
- Synthesise and/or place and route the RMAP core using the end user synthesis tool and environment. Run the end user test-bench (TB_CORE) on the synthesised and/or placed and routed model to ensure the correct behaviour has survived synthesis (see section 2.2).

- Embed the RMAP core in the user system. Connect the test-bench to the embedded RMAP core in the end users system. Setup the embedded end user test-bench to send and receive RMAP packets to and from the end user system. Functions and packet array structures are provided (see section 2.3)
- Run the embedded core test-bench (TB_EMB) on the end user embedded model.

2.2 END USER TEST-BENCH (TB_CORE)

The end user test-bench provides a method to check the interfaces, target and initiator, of the RMAP core as shown in Figure 2-1.

The test-bench works by controlling the inputs to the RMAP core, either through the test-bench SpaceWire interface or by directly driving the inputs and checking the expected behaviour of the core against the actual output behaviour. A SpaceWire interface is used to connect to the SpaceWire ports on the RMAP core and the test-bench transmits and receives SpaceWire packets through the interface.

The tests which are run on the interface are listed below.

1) Initialise test-bench
2) Reset
3) Non-RMAP loopback
4) Target non-verified write operation with reply
5) Target verified write operation with reply
6) Target read
7) Target read modify write
8) Initiator initiate command and receive reply
9) Initiator initiate command and delete expected transaction record
10) Send and receive time-code

An overview of the end user test-bench architecture and tests is given in section 4.

2.3 END USER TEST-BENCH WITH EMBEDDED RMAP CORE (TB_EMB)

The end user test-bench with embedded RMAP core provides a method to transmit and receive RMAP and SpaceWire packets through the SpaceWire interface to the RMAP core as illustrated in Figure 2-2.

The embedded test-bench works by controlling the FIFO and time-code interfaces to the SpaceWire link in the test-bench which allows it to transmit and receive SpaceWire commands. When the test-bench starts it starts the SpaceWire link and expects the link to be running after a period of 100 µs.

Two example SpaceWire target operations are given in the test-bench file “enduser_tb_embedded.vhd”. The tests perform a non-verified write and a verified write operation with reply expected.

An overview of the embedded test-bench architecture and example commands with instructions on how to implement additional commands is defined in section 5.

3 TEST-BENCH ENVIRONMENT

3.1 SUPPORTED SIMULATORS

The RMAP core end user test-bench script files are written for the Mentor Graphics Modelsim simulator. The test-bench is tested on version 6.4c of the simulator but is compatible with all 6.3 and 6.4 versions.

3.2 SOURCE FILES AND DIRECTORY STRUCTURE

The core directory src/enduser_tb holds the source VHDL and script files required to run the test-bench. The source directory structure is listed in Table 3-1.

File	Description
src/enduser_tb/mem	Memory structures for the test-bench SpaceWire link
src/enduser_tb/scripts	Modelsim scripts
src/enduser_tb/spw	SpaceWire files to interface to the RMAP core
src/enduser_tb/tb	Test-bench modules
src/enduser_tb/top	Top level test-bench files
src/enduser_tb/uut	UUT control files

Table 3-1 Source file directory structure.

3.3 SOURCE FILES

The source files for the RMAP IP core are listed in Table 3-2. The root directory for each is not shown and should be assumed to be <ip_dir>/src/enduser_tb/

File	Description
mem/asyncfifologic.vhd	SpaceWire link test-bench memory file. Dual port FIFO logic
mem/dpfifo.vhd	SpaceWire link test-bench memory file. Dual port FIFO wrapper
mem/fifo_out_valid.vhd	SpaceWire link test-bench memory file. FIFO output encoder
mem/memblock.vhd	SpaceWire link test-bench memory file.
mem/readptr.vhd	SpaceWire link test-bench memory file. Dual port read pointer.
mem/writeptr.vhd	SpaceWire link test-bench memory file. Dual port write pointer
scripts/enduser_tb.do	End user test-bench Modelsim script
scripts/enduser_tb_embedded.do	Embedded end user test-bench Modelsim script
spw/enduser_tb_spwrlinkwrap_verif.vhd	SpaceWire link wrapper for test-bench including memory blocks.

spw/enduser_tb_spwrlink_pkg.vhd	SpaceWire link configuration file for test-bench
tb/monitor_fifo.vhd	VHDL module to record reads and writes over a SpaceWire interface to a text file
top/enduser_tb_pkg.vhd	Common test-bench functions and constants
top/enduser_tb_tests_pkg.vhd	Tests to perform on the RMAP core
top/enduser_tb.vhd	Top level of test-bench (TB_CORE)
top/enduser_tb_embedded_pkg.vhd	Functions and constants for the embedded test-bench.
top/enduser_tb_embedded.vhd	Top level of the end user test-bench (TB_EMB)
uut/rmap_codec_ip_spwrlink_pkg.vhd	Configuration of RMAP core SpaceWire link
uut/embedded_core.vhd	Dummy embedded core which automatically authorises target commands, grants bus access and returns dummy data when a bus read is performed.

Table 3-2 Source File Descriptions

A number of helper procedures are available in enduser_tb_pkg.vhd. The procedures are listed in the Table 3-3.

Filename	Function	Description
enduser_tb_pkg.vhd	check	Check if an output of the RMAP core is equal to the expected output
enduser_tb_pkg.vhd	reverse	Swap bits in a vector
enduser_tb_pkg.vhd	rmap_calccrc	Calculate the next value of the CRC register using the current value and the next byte.
enduser_tb_pkg.vhd	compute_data_crc	Compute the CRC of a SpaceWire packet
enduser_tb_pkg.vhd	check_spw_data	Check the contents of a SpaceWire packet against an expected packet.
enduser_tb_pkg.vhd	transmit_spw_packet	Transmit a SpaceWire packet to a FIFO interface
enduser_tb_pkg.vhd	receive_spw_packet	Receive a SpaceWire packet from a FIFO interface
enduser_tb_pkg.vhd	run_for_cycles	Run the test-bench for a number of clock cycles.

Table 3-3 Procedures from enduser_tb_pkg.vhd

3.4 LOG FILES

The simulation produces a Modelsim waveform (WLF) file, which can be viewed after simulation completes, and a number of text log files which output recorded data from the interfaces on the test-bench. The log files are listed below.

File	Description
enduser_tb.log	Modelsim text log file which records simulator output and information on the tests being performed.
enduser_tb.wlf	Modelsim waveform log file. All signals are logged to the log file using the Modelsim command “add log –r /*”.
auth_data.txt	Text file which records target authorisation parameters when the test-bench authorises a target command. This file is generated by the test-bench VHDL code.
non_rmap_data.txt	Text file which records read and write operations over the non RMAP interface. If a read and write are performed on the same cycle then they are both recorded on the same line. This text file is only available in the normal end user test-bench and is not available in the embedded test-bench, as the non RMAP interface is not available.
spw_data.txt	Text file which records read and write operations over test-bench SpaceWire link transmit and receive interface. If a read and write are performed on the same cycle then they are both recorded on the same line.
ports.txt	Text file which records the state of the inputs and outputs from the RMAP core model on each cycle. This file is only available in the normal end user test-bench.

Table 3-4 Test-bench log files

3.5 TEST-BENCH SCRIPTS

The test-bench scripts, normal and embedded test-benches, are located the src/enduser_tb/scripts directory. The script files are listed in Table 3-5.

File	Description
enduser_tb.do	Modelsim script to run the normal test-bench. The script is in Modelsim “do” file TCL format.
enduser_tb_embedded.do	Modelsim script to run the embedded test-bench. The script is in Modelsim “do” file TCL format.

Table 3-5 Test-bench scripts.

Both test-bench scripts accept the options listed in Table 3-6.

option	Description
-restart	After compilation restart the test with “restart -f; run –all” command
-top_only	Re-compile the top level verification files only. The files which are recompiled are top/enduser_tb_tests_pkg.vhd and top/enduser_tb.vhd.
-netlist <file>	Specify a netlist file to compile instead of the RMAP core RTL files
-timing <sdf> <target>	Specify an SDF timing file to be linked against the netlist target in the testbench file.

Table 3-6 Test-bench script options

An example command to run the test-bench with a netlist and timing file is shown below. The /UUT parameter specifies the path to the target model in the test-bench VHDL top level file.

```
do ../../src/enduser_tb/scripts/enduser_tb.do \
   -netlist ../../design/test/rmap_core.vhd \
   -timing ../../design/test/rmap_core.sdf /UUT
```

3.6 VENDOR LIBRARIES

When the CFG_TECH generic is set to TECH_MEM_PROASIC the Actel proasic3 library should be available in the Modelsim library path.

When CFG_TECH generic is set to TECH_MEM_AXCELERATOR the Actel axcelerator library should be available in the Modelsim library path.

If the Modelsim version is not an Actel specific version then the libraries can be compiled into Modelsim using the source files located in the designer installation directory “`Model\actel\Vhdl\src`”.

3.7 TEST-BENCH COMPLETION

The end user test-bench (TB_CORE) completes with the message:

```
# ** Note: ----- [  ]: Completed
#    Time: 111995520 ps  Iteration: 1  Instance: /enduser_tb
```

The end user test-bench (TB_EMB) completes with the message:

```
# ** Note: ----- [  ]: Completed
#    Time: 33248670 ps  Iteration: 1  Instance: /enduser_tb_embedded
```

The message is displayed after all tests have completed successfully and the test-bench clocks have stopped.

4 END USER TEST-BENCH (TB_CORE)

4.1 ARCHITECTURE

The end user test-bench architecture is defined in Figure 4-1.

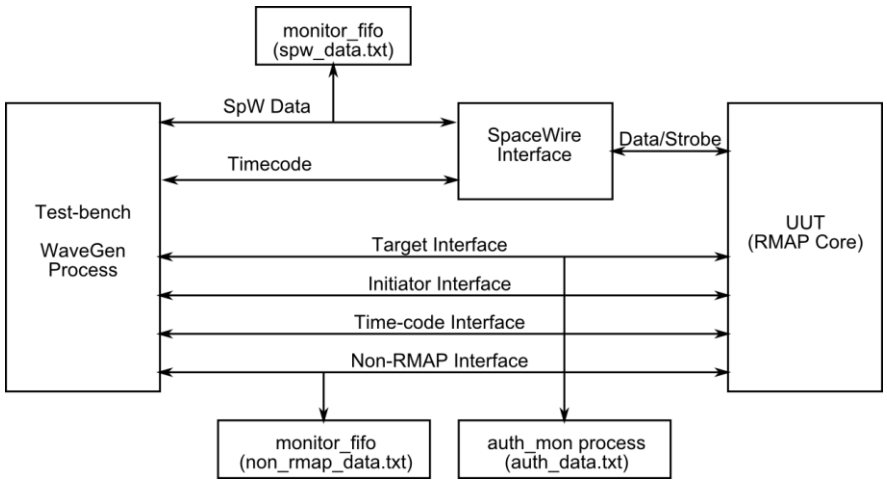


Figure 4-1 End User Test-bench Architecture. The WaveGen process sets the inputs and checks the outputs of the RMAP core. The monitor_fifo components are used to record data transmitted and receive to and from a FIFO. The SpaceWire interface is used to interface to the SpaceWire port

The test-bench WaveGen process controls the inputs to the RMAP core and checks the outputs from the core are the expected values. For example the WaveGen process writes a packet to the SpaceWire interface, causing the target interface to request authorisation.

The monitor_fifo components monitor a FIFO read/write interface and write operations over the interface to a text file specified in the generics of the component.

The auth_mon_process process monitors the authorisation interface and writes authorisation information to a text file.

The SpaceWire interface connects to the serial SpaceWire port of the RMAP core and allows packets and time-codes to be received.

The UUT is the RMAP core model.

A number of helper procedures are available in enduser_tb_tests_pkg.vhd and are listed in the Table 3-3.

Filename	Function	Description
enduser_tb_tests_pkg.vhd	validate_outputs	Check the actual outputs of the RMAP core are equal to the expected outputs
enduser_tb_tests_pkg.vhd	run_for_cycles	Run for a number of clock cycles and validate the outputs

		of the core on each cycle.
enduser_tb_tests_pkg.vhd	write_to_file	Write the RMAP core inputs and outputs to a text file
enduser_tb_tests_pkg.vhd	check_bus_data	Check data received from the external bus is valid
enduser_tb_tests_pkg.vhd	reverse_bytes	Swap bytes in a vector
enduser_tb_tests_pkg.vhd	reverse_bus_data	Swap bytes in each vector of a bus data structure
enduser_tb_tests_pkg.vhd	swap_bits	Swap bits in each byte of a bus data structure
enduser_tb_tests_pkg.vhd	slave_bus_write	Receive data from the external bus when a bus write is performed.
enduser_tb_tests_pkg.vhd	slave_bus_read	Make data available to the external bus when a bus read is performed.

Table 4-1 Procedures from enduser_tb_pkg.vhd

4.2 RMAP CORE CONFIGURATION

The RMAP core configuration constants are defined in the VHDL file src/enduser_tb/top/enduser_tb.vhd. The configuration section is identified by the VHDL comments:

```
-- {START RMAP CODEC CONFIGURATION CONSTANTS}
...
-- {END RMAP CODEC CONFIGURATION CONSTANTS}
```

When netlist or post place and route simulation is performed the configuration should be set to the configuration used to synthesise the core. This allows the test-bench to insert and check the correct data patterns dependent on the bus parameters; CFG_WORD_SIZE, CFG_TARGET_MSB_FIRST, CFG_TARGET_BITSWAP, CFG_INI_CODEC_MSB_FIRST and CFG_INI_CODEC_BITSWAP.

4.3 RMAP CORE INTERFACES

The interfaces of the RMAP core are checked by the end user test-bench. The interfaces are listed in the following list:

1. SpaceWire serial interface
2. Time-code interface
3. Initiator status interface
4. Initiator command encode interface
5. Initiator reply decode interface

6. Initiator delete interface
7. Target authorisation interface
8. Target status interface
9. Target RMW interface
10. Bus interface
11. Non RMAP port

4.4 TEST DESCRIPTIONS

The tests which are performed by the end user test-bench are described in this section. The tests are defined in the VHDL file src/enduser_tb/top/enduser_tb_tests_pkg.vhd.

The tests which are performed are listed below:

1. Initialise test-bench
2. Reset
3. Non-RMAP loopback
4. Target non-verified write operation with reply
5. Target verified write operation with reply
6. Target read
7. Target read modify write
8. Initiator initiate command and receive reply
9. Initiator initiate command and delete expected transaction record
10. Send and receive time-code

The tests are described in detail in the following sections.

4.4.1 Test 1: Initialise test-bench

The initialise test-bench operation is performed in the VHDL function “do_init_and_reset”. In the function the inputs and initialise to default values and the expected outputs are set.

4.4.2 Test 2: Reset

Reset of the RMAP core is performed in the VHDL function “do_init_and_reset”. Reset is pulsed low for 4 clock cycles and then released.

4.4.3 Test 3: Non RMAP Loopback

The non RMAP loopback test writes a 1024 byte packet to the non RMAP port and checks the reply. The 1024 byte packet consists of an incrementing pattern starting at 0 and ends with an EOP. If the packet is received correctly the test is a success.

4.4.4 Test 4: Target Non-Verified Write

The non-verified write test checks the targets response to a non-verified write operation. The authorisation parameters used in the target command are shown in Table 4-2.

Parameter	Value
Logical Address	0xfa
Command	0x6c, write=1, reply=1, acknowledge=1
Key	0x20
Initiator Logical Address	0xfe
Transaction ID	0xabcd
Extended Address	0x00
Address	0x001f4502
Data Length	8

Table 4-2 Non-Verified write test parameters

The test procedure is listed below:

1. Generate test data accounting for word size, byte order and bit swapping.
2. Send RMAP command packet with correct header and data CRC.
3. Check authorisation parameters on the RMAP core and authorise the command with no error.
4. Grant bus access and check the data written to the external bus accounting for word size, byte order and bit swapping.
5. Receive RMAP reply and check format, CRCs and data.

4.4.5 Test 5: Target verified write operation with reply

The verified write test checks the targets response to a verified write operation. The verify write buffer is used to store data by the target before it is written to the external bus. The authorisation parameters used in the target command are shown in Table 4-3.

Parameter	Value
Logical Address	0xb5
Command	0x7c, write=1, verify=1, reply=1, acknowledge=1
Key	0xa5
Initiator Logical Address	0x5a
Transaction ID	0x1122
Extended Address	0x00
Address	0xa5a5a5a5
Data Length	8

Table 4-3 Verified write test parameters

The test procedure is listed below:

1. Generate test data accounting for word size, byte order and bit swapping.
2. Send RMAP command packet with correct header and data CRC.
3. Check authorisation parameters on the RMAP core and authorise the command with no error. Verify bit should be set on the command byte
4. Grant bus access and check the data written to the external bus accounting for word size, byte order and bit swapping.
5. Receive RMAP reply and check format, CRCs and data.

4.4.6 Test 6: Target read

The target read test checks the targets ability to read data from the external bus and return the data in an RMAP packet. The target read command uses return path addresses in the RMAP command to add leading bytes to the return packer. The authorisation parameters used in the target command are shown in Table 4-4.

Parameter	Value
Logical Address	0x23
Command	0x4d, write=0, reply=1, acknowledge=1, path length=1
Key	0x1f
Return Path Address	0x00 0x00 0x01 0x02
Initiator Logical Address	0x22

Transaction ID	0x0001
Extended Address	0x00
Address	0x51a40dc7
Data Length	16

Table 4-4 Read command test parameters

The test procedure is listed below:

1. Send RMAP command packet with correct header CRC.
2. Check authorisation parameters on the RMAP core and authorise the command with no error. Command should be a read command with return path address length set to 1.
3. Grant bus access and set data in to return data to the RMAP core from the external bus.
4. Receive the RMAP command and check the format, reply data, header CRC and data CRC. The leading path bytes should be received before the RMAP command.

4.4.7 Test 7: Target read modify write

The target read modify write checks a read modify write operation to a target memory address. A read modify write operation reads data from the bus, provides the data from the bus and the command packet to the RMW interface, writes data back to the bus and returns an RMAP reply packet with the original bus data. The authorisation parameters used in the target command are shown in Table 4-5.

Parameter	Value
Logical Address	0x43
Command	0x4d, command=0111b
Key	0xfc
Initiator Logical Address	0xe5
Transaction ID	0x0112
Extended Address	0x00
Address	0x12345678
Data Length	8, Data (4) + Mask (4)

Table 4-5 Read Modify Write test parameters

The test procedure is listed below:

1. Send RMAP command packet data and mask information.

2. Check authorisation parameters on the RMAP core and authorise the command with no error. Command should be a read modify write command.
3. Grant bus access and set data in to return original RMW data to the RMAP core from the external bus.
4. Check RMW command parameters and return modified data by acknowledging the RMW operation.
5. Grant bus access and check the modified data is written back to the bus in the correct format.
6. Receive a RMAP command and check the returned data length is 4 and the returned data is the original data read from the external bus. Check the packet format and the header and data CRCs.

4.4.8 Test 8: Initiator initiate command and receive reply

The initiator command and reply test checks the ability of the RMAP core to generate and send an RMAP command and to decode an RMAP reply to the command. The parameters of the RMAP command are given in Table 4-6.

Parameter	Value
Command Pointer	0x11234456
Generate TID	0
Transaction Flags	MSB first=1, notify sent=1, notify reply=1, header inc=1, DMA inc=1, reply expected=1
Header Pointer	0x0000c000
Data Pointer	0x0000c100
Sent Notify Pointer	0x0000c200
Reply Data Pointer	0x0000c300
Reply Notify Pointer	0x0000c400
Data Length	8
Reply Timeout	0xffffffff, infinite timeout
Header Logical Address	0xe3
Command	0x5c, command=0111 (Read Modify Write)
Key	0x22
Initiator Logical Address	0x3a
Transaction ID	0x5678
Extended Address	0
Address	0x12345678

Data Length	8

Table 4-6 Initiator command test parameters

The test procedure is listed below:

1. Initiate command using the initiator interface. The command pointer sets the address in memory to read the transaction record.
2. Grant bus access for the initiator to read the transaction flags from the transaction record.
3. Grant bus access for the initiator to read the remainder of the transaction record.
4. Grant bus access for the initiator to read the header. The header is read in two passes; pass 1 reads and checks the header for errors and pass 2 reads and sends the header as an RMAP packet.
5. Grant bus access for the initiator to send the command header to the SpaceWire core.
6. Grant bus access for the initiator to send data and mask information.
7. Grant bus access for the initiator to write command indication status to the external bus.
8. Check the command indication interface for command success.
9. Receive and check the RMAP command sent by the initiator.
10. Send an RMAP reply packet with the returned RMW data.
11. Grant bus access for the reply data bus transfer.
12. Grant bus access for the reply indication status to be written to the reply notification register.
13. Check reply indication status for success.

4.4.9 Test 9: Initiator initiate command and delete expected transaction record

The initiator command and delete expected transaction checks the delete interface of the RMAP core. A command is started and the expected reply is deleted by the test-bench. The parameters of the RMAP command are given in Table 4-6.

Parameter	Value
Command Pointer	0x11234456
Generate TID	0
Transaction Flags	MSB first=1, notify sent=1, notify reply=1, header inc=1, DMA inc=1, reply expected=1
Header Pointer	0x0000c000
Data Pointer	0x0000c100

Sent Notify Pointer	0x0000c200
Reply Data Pointer	0x0000c300
Reply Notify Pointer	0x0000c400
Data Length	8
Reply Timeout	0xffffffff, infinite timeout
Header Logical Address	0xe3
Command	0x4c, write=0, acknowledge=1, reply=1
Key	0x22
Initiator Logical Address	0x3a
Transaction ID	0x5678
Extended Address	0
Address	0x12345678
Data Length	8

Table 4-7 Initiator command test parameters

The test procedure is listed below:

1. Initiate command using the initiator interface. The command pointer sets the address in memory to read the transaction record.
2. Grant bus access for the initiator to read the transaction flags from the transaction record.
3. Grant bus access for the initiator to read the remainder of the transaction record.
4. Grant bus access for the initiator to read the header. The header is read in two passes; pass 1 reads and checks the header for errors and pass 2 reads and sends the header as an RMAP packet.
5. Grant bus access for the initiator to send the command header to the SpaceWire core.
6. Grant bus access for the initiator to send data and mask information.
7. Grant bus access for the initiator to write command indication status to the external bus.
8. Check the command indication interface for command success.
9. Receive and check the RMAP command sent by the initiator.
10. Delete the command from the transaction table.

4.4.10 Test 10: Send and receive time-code

The send and receive time-code test checks that the RMAP core time-code interface can send and receive a time-code.

4.5 RUNNING THE TEST-BENCH

Testing is performed in the “verif/enduser_tb” directory. To run the test-bench start the Modelsim simulator in the “verif/enduser_tb” directory and run the do file “run.do”. To run the test-bench with options then the test-bench script should be referenced directly from the Modelsim command line, for example:

```
do ../../src/enduser_tb/scripts/enduser_tb.do <options>
```

4.6 NETLIST OR BACK ANNOTATED TIMING SIMULATION

Netlist simulation can be performed using the test-bench script as described in section 3.5, test-bench scripts. To perform netlist simulation the VHDL file src/enduser_tb/top/enduser_tb.vhd should be modified to instantiate the users design. The component declaration and component instantiation is enclosed by the comments:

```
-- {START DECLARATION}
....
-- {END DECLARATION}
-- {START INSTANTIATION}
....
-- {END INSTANTIATION}
```

When instantiating a new netlist model into the VHDL code the interfaces should be connected exactly as they are in the RTL model.

5 EMBEDDED TEST-BENCH (TB_CORE)

5.1 ARCHITECTURE AND USAGE

The end user embedded test-bench architecture is defined in Figure 4-1.

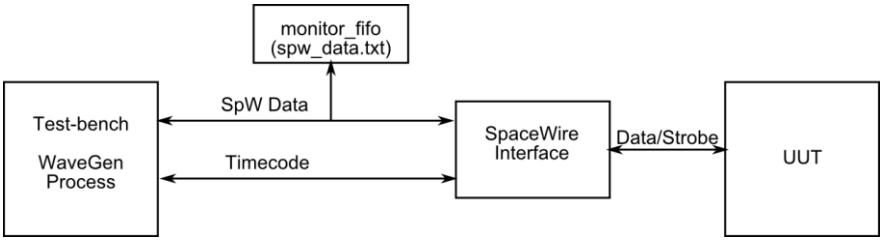


Figure 5-1 End user test-bench embedded architecture

The UUT is the end user system with an embedded RMAP core. In the default test-bench model a dummy UUT core is located at “src/enduser_tb/uut/embedded_core.vhd”. The dummy core automatically authorises target commands, grants bus access and returns dummy data when a bus read is performed.

In the end user test-bench the WaveGen process is used to setup a series of packets which will be sent and received by the test-bench. Procedures in the VHDL file src/enduser_tb/top/enduser_tb_embedded_pkg.vhd provided a method to send and receive packets from an array of packet data structures automatically.

Two example packets are provided in the WaveGen process, a verified and non-verified write operation to a dummy RMAP core which authorises all target operations and returns dummy data.

The monitor_fifo component monitors a FIFO read/write interface. The component writes operations over the interface to a text file specified in the generics of the component.

The enduser_tb_embedded_pkg.vhd procedures which can send and receive packets are listed below:

- transmit_and_receive_spw_packets: Transmit a number of SpaceWire packets to the UUT through the SpaceWire interface and for each packet transmitted expect a reply. Check the reply packet against the expected reply packet.
- receive_and_transmit_spw_packets: Receive a number of SpaceWire packets from the UUT through the SpaceWire interface and check the format and size of the packets against the expected packets. For each packet received transmit a reply packet.

Send and reply packets are built using the spw_packet record and the SpaceWire packet array spw_packet_array_t type. The types are listed in the VHDL code below:

```
-- byte buffer array of data to send
type spw_byte_array_t is array (0 to MAX_PACKET_SIZE)
   of std_ulogic_vector(8 downto 0);
```

```
-- packet with size
type spw_packet is record
   packet_data : spw_byte_array_t;
   size        : integer;
end record;

-- array of spacewire packets
type spw_packet_array_t is array (0 to MAX_PACKETS)
   of spw_packet;
```

5.2 EXAMPLE TEST DESCRIPTIONS

The example test packets send a verified and non-verified write command to the UUT using the spw_packet_array_t array of packets and the transmit_and_receive_spw_packets procedure in the embedded test package.

5.2.1 Test 1: Non-Verified Write Packet

The non-verified write packet parameters are defined in table xx.

Parameter	Value
Logical Address	0xfa
Command	0x6c, write=1, reply=1, acknowledge=1
Key	0x40
Initiator Logical Address	0xfe
Transaction ID	0xabcd
Extended Address	0x00
Address	0xffffff00
Data Length	4
Data	{0xaa 0xbb 0xcc 0xdd}

Table 5-1 Non-Verified write test parameters

5.2.2 Test 2: Verified Write Packet

The example verified write packet parameters are defined in table xx.

Parameter	Value
Logical Address	0x21
Command	0x7c, write=1, verify=1, reply=1, acknowledge=1
Key	0x10
Initiator Logical Address	0xab
Transaction ID	0x1234
Extended Address	0x00
Address	0x5a5a5a5a
Data Length	16
Data	{0x00 0x01 0x02 ... 0x0e 0x0f}

Table 5-2 Verified write test parameters

5.3 RUNNING THE TEST-BENCH

Testing is performed in the "verif/enduser_tb_embedded" directory. To run the test-bench start the Modelsim simulator in the "verif/enduser_tb_embedded" directory and run the do file "run.do". To run the test-bench with options then the test-bench script should be referenced directly from the Modelsim command line, for example:

```
do ../../src/enduser_tb/scripts/enduser_tb_embedded.do <options>
```

5.4 NETLIST OR BACK ANNOTATED TIMING SIMULATION

Netlist simulation can be performed using the test-bench script as described in section 3.5, test-bench scripts. To perform netlist simulation the VHDL file src/enduser_tb/top/enduser_tb_embedded.vhd should be modified to instantiate the users design. The component declaration and component instantiation is enclosed by the comments:

```
-- {START DECLARATION}
....
-- {END DECLARATION}
-- {START INSTANTIATION}
....
-- {END INSTANTIATION}
```

When instantiating a new netlist model into the VHDL code the interfaces should be connected exactly as they are in the RTL model.

6 DOCUMENT CHANGES

The changes made this document are listed in **Error! Reference source not found.**.

(Issue 1.0 to Issue 1.1)	
Section/Reference	**Change**
2	Add overview of the test-benches
3	Add test-bench environment detailed description
4	Add tests, architecture and usage of test-bench
5	Add tests, architecture and usage of embedded test-bench
All	Major revisions to all sections and added new sections detailing architecture and tests for end user and embedded test-benches
All	End user test-bench = TB_CORE and embedded test-bench = TB_EMB
3.3	Inserted table on helper functions for enduser_tb_pkg.vhd
3.6	Inserted section on vendor libraries
3.7	Added detailed text on completion message and completion text for test-benches
4.1	Procedures from enduser_tb_tests_pkg.vhd added
5.1	Procedures from enduser_tb_embedded_pkg.vhd added

Table 6-1 Changes to Document