ESA 13345/#3 : Building Block for System On a chip

SPACEWIRE IP CORE HARDWARE USER MANUAL

	Name and Function	Date	Signature
Prepared by	**Tam Le Ngoc**		
Verified by	**Marc Lefebvre**		
Approved by			
Authorised by	**Marc Souyri**		

Document type	Nb WBS	Keywords

DOCUMENT CHANGE LOG

Issue/ Revision	Date	Modification Nb	Modified pages	Observations
0/0				Creation
0/1	25/04/02			Included comments from ESA
	04/03/03			Adding TX clock configuration.

PAGE ISSUE RECORD

Issue of this document comprises the following pages at the issue shown

Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.	Page	Issue/ Rev.
all	**0/1**										

TABLE OF CONTENTS

LIST OF FIGURES

1 SCOPE

The present document is written in the frame of the ESA 13345/#3 contract " Building block for System on a Chip". It is part of Phase 3 of the contract related to the design of a System On a Chip for Space application. The present activity concerns the design of a Spacewire VHDL core to be integrated in the System On a CHip.

This document is the Hardware User Manual of the VHDL core. It is intended for users that would like to use the VHDL block. It explains the following elements :

- structure of the UNIX directories containing the core and the testbench,
- adaptation of the core to other technologies,
- structure of the two testbenches used to verify the core,
- test plan of the testbenches,
- run of the simulation,
- synthesis of the core.

2 DOCUMENTS AND ACRONYMS

2.1 APPLICABLE DOCUMENTS

AD8	*SCOC Requirement Specification*	R&D-RP-SOC-214-MMV, Issue 2, June 2000
AD9	*AMBA™ Specification*	*Rev 2.0, ARM IHI 0011A*
AD10	*Spacecraft Controller On a Chip Architectural Design Document*	
AD11	*ECSS-E-50-12 Draft 1 (ESA SpaceWire Specification)*	*March 2001*

2.2 REFERENCE DOCUMENTS

RD21	*System-On-a-Chip Feasibility Study*	December 99, Issue 2, R&D-RP-SOC-154-MMV
RD26	*Austrian Aerospace Spacewire Test-bench User Manual*	*November 2000*
RD27	*Austrian Aerospace Spacewire Protocol Verification*	*November 2000*

2.3 ACRONYMS

AD	Applicable Document
APB	Advanced Peripheral Bus
AHB	Advanced High-Performance Bus
DMA	Direct Memory Access
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
HKPF	Housekeeping Function
HKAPB	Housekeeping Advanced Peripheral Bus
IP	Intellectual Property
IT	Interrupt
LVDS	Low Voltage Differential Signals
SCoC	Spacecraft Controller on a Chip
SWB	SpaceWire Block
RD	Reference Document
RHI	RX Host Interface
SOC	System-On-a-Chip
THI	TX Host Interface

3 DESIGN DESCRIPTION

3.1 ACCOUNT STRUCTURE

The path of the working account used for the design is called "your_path" in the present document.

The organization is shown hereafter:

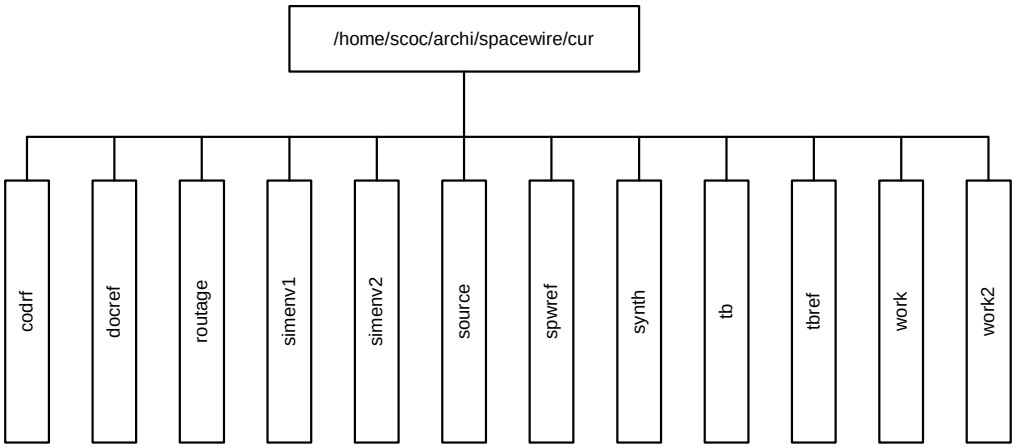

Figure 3.3.1-1 Account structure

The SOURCE directory contains all the VHDL files of the Spacewire VHDL core (SWB).

The CODRF, DOCREF, SPWREF, TBREF directories contain documentations, testbench and emulators provided by Austrian Aerospace to test the spacewire protocol on the link side.

The Modelsim compilation of the Austrian Aerospace test bench is stored in the WORK2 directory.

The SIMENV2 directory is used to simulate the Austrian Aerospace bench.

The TB directory contains the second test bench mainly used to test the host interface of the spacewire core.

The result of the Modelsim compilation of the second testbench is stored in the WORK directory.

The SIMENV1 directory is dedicated to launch the second testbench simulation. The *sw.tb* file contains all the tests performed on the spacewire.

The SYNTH directory contains specific files for the synthesis made with the Synplify software. These files are *spacewire.prj* (project file) and *spacewire.sdc* (constraints file). All the result files are in the REV_X sub-directory such as the *spacewire.edf* edif file and the *spacewire.srr* report file.

The ROUTAGE directory contains the result files of the Xilinx software that performs the place and route task.

3.2 THE VHDL SOURCE DESCRIPTION

All the VHDL source files and the level hierarchy are shown hereafter:

TOP LEVEL	LEVEL 2	LEVEL 3
spacewire.vhd	sw.vhd	tx_mgt.vhd rx_mgt.vhd sw_counters.vhd sw_resync.vhd sw_reg.vhd delay_cnt.vhd init_fsm.vhd
	host_int.vhd	ahb_mst_slv_tx.vhd sw_fifo.vhd (AHB FIFO) ahb_tx_int.vhd ahb_mst_rx.vhd
	rx.vhd	rx_resync.vhd rx_decod.vhd rx_shiftreg.vhd disconnection.vhd
	tx.vhd	tx_resync.vhd tx_select.vhd txcnt.vhd txshiftreg.vhd ds_gen.vhd tx_ack.vhd
	clk_tx_gen.vhd sw_fifo.vhd (TX FIFO) sw_fifo.vhd (RX FIFO)	

Figure 3.3.1-1 Source files hierarchy

The top file spacewire_noamba.vhd does not include the host interface. The architecture spacewire_noamba is used to the Austrian Aerospace testbench. In SCOC project, the top file spacewire.vhd is used. In this case the sw_noamba.vhd file is replaced by the sw.vhd file.

3.3 CONFIGURATION OF THE SPACEWIRE BLOCK

Some parameters can be personalized in the VHDL code in order to dimension the FIFO size, to change counter widths and to choose the TX clock type (gated or not).

3.3.1 FIFO configuration

For each FIFO (RX, TX or AHB) , the user can specify :

- The address width of the FIFO corresponding to the FIFO depth (xxFIFOABITS). The recommended values are the following:
  - for RX RXFIFOABITS must be greater than or equal to 6
  - for TX TXFIFOABITS must be greater than or equal to 3
  - for AHB AHBFIFOABITS must be greater than or equal to 2. This FIFO is a 32 bit word size FIFO, its size depends on the AHB latency.

xxFIFODBITS is the width of the data. Its is set at 9 and shall not be changed.

xxFIFODEPTH is the number of words of the FIFO. It is worked out from xxFIFOABITS and shall not be modified.

These parameters are included in the *sw_pack.vhd* file.

The configuration of the RX FIFO, TX FIFO and AHB FIFO is described hereafter.

```
constant RXFIFOABITS  : integer:= 6;
constant RXFIFODBITS  : integer:= 9;                    -- shall not be modified
constant RXFIFODEPTH  : integer:= 2** RXFIFOABITS  ;    -- shall not be modified
constant TXFIFOABITS  : integer:= 3;
constant TXFIFODBITS  : integer:= 9;                    -- shall not be modified
constant TXFIFODEPTH  : integer:= 2** TXFIFOABITS  ;    -- shall not be modified
constant AHBFIFOABITS : integer:= 2;
constant AHBFIFODBITS : integer:= 32;                   -- shall not be modified
constant AHBFIFODEPTH : integer:= 2** AHBFIFOABITS  ;   -- shall not be modified
```

3.3.2 Counter configuration

The size of the counter used for the 6.4 µs delay generation is defined by the DELAYWIDTH constant. The maximum value for DELAYWIDTH is 16 since only 16 bits are reserved in the Time_Out Register to store the maximum value of the counter.

The size of the counter used to store the number of RX FIFO empty space is defined by the CWIDTH constant.

The CMAX constant has the value of RXFIFODEPTH but in std_logic_vector format.

These constants are included in the *sw_pack.vhd*.

```
constant DELAYWIDTH   : integer:= 8;
constant CWIDTH       : integer:= 7;
constant CMAX         : std_logic_vector((CWIDTH-1) downto 0):= conv_std_logic_vector(2**
RXFIFOABITS, CWIDTH); -- shall not be modified
```

3.3.3 TX clock configuration

The TX frequency can be generated by either a gated clock with a 2(n+1) frequency divider or a not-gated clock using an enable clock signal. In the last case, a n+1 frequency divider is used.

Configuration:

⇒ To use a gated TX clock, you must have: “constant GATED_TX_CLK : Boolean := true;”

⇒ To use a not-gated TX clock, you must have: “constant GATED_TX_CLK : Boolean := false;”

3.4 PORTING OF THE SPACEWIRE CORE TO DIFFERENT TECHNOLOGY

All the Spacewire code is written in synthesizable VHDL. Most of the code is independent from the target technology. The specific parts concern only :

- The three FIFOs : RX, TX and AHB. Each FIFO is made by a dual port RAM. By now, the VHDL code is targeted for Xilinx and Synplify. Thus the synthesis tool is able to recognize that the VHDL code corresponds to Xilinx RAMB block. This part would have to be changed for another technology, otherwise fil-flops will be inferred.
- The TX clock generator that is not included in the spacewire block. The user has to generate this high frequency clock, the Spacewire Block contains only the divider that generates the TX clock.

In addition there are 4 clocks used in the Spacewire Block that the user must balance according to the capability of the target technology :

- The System clock clk_sw that controls about 700 flip flops

- The Input TX clock clk_txin that controls about 20 flip flops
- The TX clock obtained after divide of clk_txin clock that controls about 150 flip flops.
- The RX clock clk_rx that controls about 150 flip flops.

4 SIMULATION PLAN

4.1 INTRODUCTION

Two testbenches are used:

- A testbench based on the Austrian Aerospace testbench. This testbench is used to test the Spacewire VHDL core on the link side. Some sequences were added to the Austrian Aerospace testbench provided by ESA to test the time code.
- A testbench developed by Astrium that allows testing the host interface.

4.2 TEST OF THE SPACEWIRE WITHOUT THE HOST INTERFACE

The spacewire VHDL core without its host interface is connected to the Austrian Aerospace spacewire emulator as depicted in Figure 3.3.3-1. The verification is done by reading the FIFOs data and comparing them to expected results.

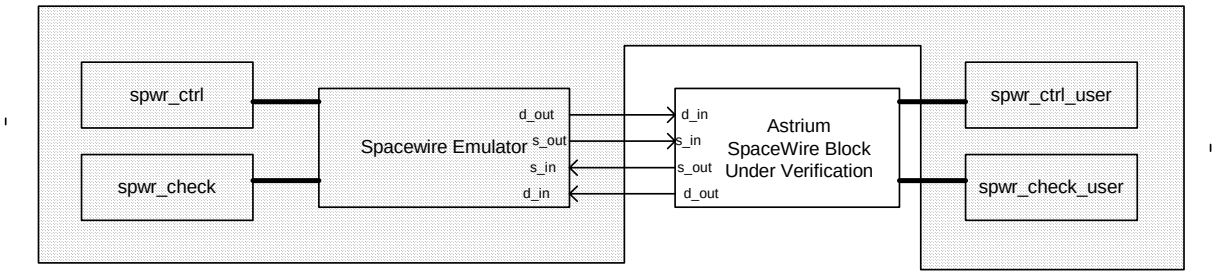

Figure 3.3.3-1 : Austrian Aerospace test bench

The Austrian Aerospace test bench checks the protocol initialization, the TX interface, the RX interface and the Time Code, FCT and data management.

The test cases performed are:

- Link Startup
- Normal operation
- Error cases
- Stress Cases (TX and RX rates are different)

For more information, read *protocol_verification.pdf* and *testbench_user_manual.pdf* included in the /your_path/docref directory.

4.3 TEST OF THE HOST INTERFACE

Two spacewire blocks are connected together in Astrium testbench as depicted in Figure 3.3.3-1.

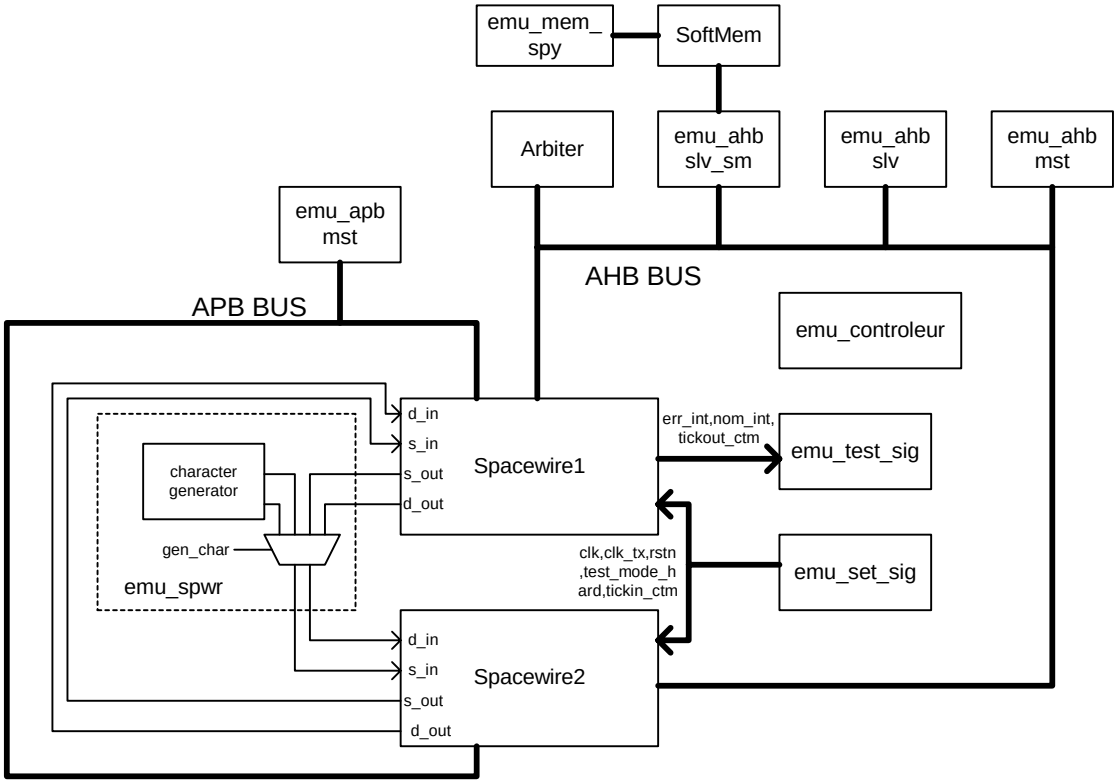

Figure 3.3.3-1 : Test bench for host interface test

This test checks the TX AHB master and slave interfaces and the RX AHB master interface. Checking the TX AHB master/slave of one spacewire is done by verifying that the data received by the other spacewire block is correct. The RX AHB master is checked for normal operations and for operations performed when the host memory area limit is reached.

VHDL blocks have been developed to emulate the behaviour of the spacewire block environment. The VHDL emulators used for the test are:

- emu_spwr generates error characters for Spacewire 2 to simulate transmission errors.
- emu_apbmst is used for the spacewire block configuration through APB bus.
- emu_set_sig sets signals.
- emu_test_sig checks signals.
- arbiter manages the AHB master/slave traffic
- emu_ahbslv_sm interfaces between AHB slave and SoftMem

- SoftMem is an advanced memory block
- emu_mem_spy checks the memory data
- emu_ahbslv is an AHB slave used to give RETRY, SPLIT or ERROR response to the spacewire
- emu_ahbmst: is an AHB master used to communicate with the spacewire AHB slave interface

4.3.1 Test sequences run during the test of the host interface

4.3.1.1 introduction

Only one simulation is run that contains 39 test sequences. The testbench checks the correctness of the results by reading in the memory or by checking signal activation. The following paragraphs give a brief description of each test.

4.3.1.2 TX AHB master interface test 1: nominal operation

Data packets are transmitted using TX AHB master and the received data are checked in the SoftMem.

- Spacewire 1 & 2: programmation: tx_mode=master, linkstart=1
- Spacewire 1& 2: descriptor initialization
- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1 & 2: memory area 1 validation
- Received data are stored in the Softmem
- Comparison between reference data and received data

4.3.1.3 RX AHB master interface test 1: area middle address= area end address

- Spacewire 1& 2 programmation: tx_mode=1, test_mode_tx=1linkstart=1
- Spacewire 1& 2: descriptor initialization
- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1 & 2: memory area 1 validation
- Spacewire 1 & 2: interrupts clearing
- Spacewire 2: verify that no null is written at area 1 end
- Spacewire 1 & 2: check the exceed_mem IT and no_area_valid IT generation

4.3.1.4 RX AHB master interface test 2: RX FIFO is full and RX FIFO dumping

- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1 & 2: memory area 1 validation

4.3.1.5 RX AHB master interface test 3: area middle address<area end address

- Spacewire 1& 2: programmation: tx_mode=1, test_mode_tx=1linkstart=1
- Spacewire 1& 2: descriptor initialization
- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1 & 2: memory area 1 validation
- Spacewire 1 & 2: interrupts clearing

- Spacewire 1 & 2: verify that a null is written at area 1 end
- Spacewire 1 & 2: check the no_area_valid IT generation

4.3.1.6 RX AHB master interface test 4: RX FIFO dumping

- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1 & 2: memory area 1 validation

4.3.1.7 RX AHB master interface test 5: memory area swap

- Spacewire 1& 2: programmation: tx_mode=1, test_mode_tx=1linkstart=1
- Spacewire 1& 2: descriptor initialization
- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1& 2: memory area 2 initialization
- Spacewire 1 & 2 memory area 1 & 2 validation
- Spacewire 1 & 2: verify the memory area swap by reading the area1_used and area2_used flag
- Spacewire 1 & 2: verify that the status "10" is in the header of the last packet written in the area 1 because this packet is not entirely stored in the area1
- Spacewire 1 & 2: verify that the last part of this packet is stored in the area 2
- Spacewire 1& 2: memory area 1 reinitialization
- Spacewire 1 & 2: memory area 1 validation
- Spacewire 1 & 2: check the change from the area 2 to area 1

4.3.1.8 RX AHB master interface test 6: RX FIFO dumping

- Spacewire 1& 2: memory area 1 initialization
- Spacewire 1 & 2: memory area 1 validation

4.3.1.9 TX AHB master interface test 2: packet abortion

The spacewire 1 transmits data to the spacewire2 then receives an abortion command.

- Spacewire 1 & 2: interrupts clearing
- Spacewire :1 descriptor initialization
- Spacewire 2: memory area 1 validation
- Spacewire 1: abort_packet flag activated
- Spacewire 2: verify the EEP_REC IT activation
- Spacewire 2: check the status "01" in the last packet header

4.3.1.10 Test of Area1_valid and Area2_valid flags

These flags cannot be reset by the user except in test mode.

- Spacewire 1: write and read of areax_valid in test mode
- Spacewire 1: write and read of areax_valid in normal mode

4.3.1.11 TX AHB master interface test 3: new transfer after packet abortion

- Spacewire 1 descriptor initialization
- Spacewire 1: check the end_list IT activation
- Compare the data received by the spacewire 2 with the reference

4.3.1.12 Test of the interrupts (IT)

- Spacewire 1: force ITmask to 0000 then check the value
- Spacewire 1: force ITmask to FFFF then check the value
- Spacewire 1: force all the internal IT to 0 then activate only one IT each time and check each IT by reading the IT register
- Spacewire 1: check the nominal or error IT output with the emu_test_sig emulator

4.3.1.13 Test of time code

The spacewire 1 sends time codes during the data transmission. The spacewire 2 is checked for time code reception.

- Spacewire 2: force all internal IT to 0
- Spacewire 1: descriptor initialization
- Spacewire 1: Tickin_ctm activation
- Spacewire 2: Check the tickout bit activation in the IT register. Check the tickout_ctm output activation.
- Spacewire 2: force all internal IT to 0
- Spacewire 2: Check the received time code value
- Spacewire 1: Activate the tickin bit of the management register
- Spacewire 2: Check the tickout bit activation in the IT register. Check the tickout_ctm output activation.
- Spacewire 2: Check the received time code value

4.3.1.14 TX AHB slave interface test 1: ERROR response

Verify the generation of error response to unfit request.

- Spacewire 2: tx_mode=master
- Spacewire 2: TX AHB slave write access requested
- Spacewire 2: check the wrong_mode IT activation
- Spacewire 2: TX AHB slave read access requested
- Spacewire 2: check the rd_access_error IT activation

4.3.1.15 TX AHB master interface test 4

Check the reception of RETRY, SPLIT or ERROR response when the FSM reads the packet size.

- Spacewire 1: descriptor initialization
- Emu_ahbslv generates RETRY, SPLIT then ERROR response when the TX AHB master FSM reads the packet size
- Spacewire 1: verify the amba_error IT activation

4.3.1.16 TX AHB master interface test 5

Check the reception of RETRY, SPLIT or ERROR response when the FSM reads the data address.

- Spacewire 1: descriptor initialization, IT reset
- Emu_ahbslv generates RETRY, SPLIT then ERROR response when the TX AHB master FSM reads the data address
- Spacewire 1: verify the amba_error IT activation

4.3.1.17 TX AHB master interface test 6

Check the reception of RETRY, SPLIT or ERROR response when the FSM reads the data value.

- Spacewire 1: descriptor initialization, IT reset
- Emu_ahbslv generates RETRY, SPLIT then ERROR response when the TX AHB master FSM reads the data value
- Spacewire 1: verify the amba_error IT activation

4.3.1.18 TX AHB master interface test 7

Check the reception of RETRY, SPLIT or ERROR response when the FSM reads the next packet address.

- Spacewire 1: descriptor initialization, IT reset
- Emu_ahbslv generates RETRY, SPLIT then ERROR response when the TX AHB master FSM reads the next packet address
- Spacewire 1: verify the amba_error IT activation

4.3.1.19 Test of LINK_NOT_ENABLED IT

- Spacewire 1: autostart=0, link_start=0, link_disabled=0
- Spacewire 1: check the link_not_enabled IT activation
- Spacewire 1: IT reset
- Spacewire 1: autostart=1, link_start=1, link_disabled=1
- Spacewire 1: check the link_not_enabled IT activation

4.3.1.20 TX AHB slave interface test 2: nominal operation

- Spacewire 1: IT reset, packet abortion
- Spacewire 1: Reception of data packet through the AHB slave interface (single & burst transfers)

The data is transmitted to the spacewire 2 then stored in the softmem. A data check is performed in the softmem by the emu_mem_spy.

4.3.1.21 RX AHB master interface test 7

Check the reception of RETRY, SPLIT or ERROR response when the FSM is in WR_DATA state.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- Emu_ahbslv: responses with RETRY,SPLIT or ERROR when the RX AHB master FSM is in the WR_DATA state
- Check the amba_error IT and the memory

4.3.1.22 RX AHB master interface test 8

Check the reception of RETRY, SPLIT or ERROR response when the FSM is in WR_HEADER state.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write data packets into Spacewire 1
- Data packets are transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- Emu_ahbslv: responses with RETRY,SPLIT or ERROR when the RX AHB master FSM is in the WR_HEADER state
- Check the amba_error IT and the memory

4.3.1.23 RX AHB master interface test 9

Check the reception of RETRY, SPLIT or ERROR response when the FSM is in WR_NULL state.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write data packets into Spacewire 1
- Data packets are transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- Emu_ahbslv: responses with RETRY,SPLIT or ERROR when the RX AHB master FSM is in the WR_NULL state
- Check the amba_error IT and the memory

4.3.1.24 RX AHB master interface test 10

Verify the recovery of the data once an error response appears (quantity of data to be recovered=2).

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- Emu_ahbslv: responses with RETRY,SPLIT or ERROR when the RX AHB master FSM is in the WR_DATA state
- Check the amba_error IT and the memory

4.3.1.25 RX AHB master interface test 11

Check the recovery of the data once an error response appears (quantity of data to be recovered=1).

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- Emu_ahbslv: responses with RETRY,SPLIT or ERROR when the RX AHB master FSM is in the WR_DATA state
- Check the amba_error IT and the memory

4.3.1.26 RX AHB master interface test 12

Start a new area to store one data left from the current packet.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- The current area is full and there is one data left
- Spacewire 2: storage of the left data into a new area
- Check the amba_error IT and the memory

4.3.1.27 RX AHB master interface test 13

Start a new area to store 2 data left from the current packet.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- The current area is full and there are 2 data left
- Spacewire 2: storage of the left data into a new area
- Check the amba_error IT and the memory

4.3.1.28 RX AHB master interface test 14

Start a new area to store 4 data left from the current packet.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- The current area is full and there are 4 data left
- Spacewire 2: storage of the left data into a new area
- Check the amba_error IT and the memory

4.3.1.29 RX AHB master interface test 15

Start a new area to store a new packet.

- Spacewire 2: in TX AHB slave mode, initialization of the areas
- Emu_ahbmst: write a data packet into Spacewire 1
- The data packet is transmitted from the Spacewire 1 to the Spacewire 2
- Spacewire 2: storage of the received data by writing into the emu_ahbslv
- The current area is full and there is no data left
- Spacewire 2: storage of the left data into a new area
- Check the amba_error IT and the memory

4.3.1.30 RX AHB master interface test 16

Packet contains less than 4 data.

- Spacewire 2: reception of a packet containing 3 data
- Spacewire 2: storage, the area becomes full

- Spacewire 2: loads a new area then stores next packet
- Check the memory

4.3.1.31 RX AHB master interface test 17

Packet contains less than 4 data.

- Spacewire 2: reception of a packet containing 3 data
- Spacewire 2: storage, the area becomes full
- Spacewire 2: loads a new area then stores a new packet containing less than 4 data
- Check the memory

4.3.1.32 RX AHB master interface test 18: Area2_valid flag clearing

- Spacewire 2: area2 initialization
- Spacewire 2: storage, the area 2 becomes full
- Check the flag

4.3.1.33 RX AHB master interface test 19: Check the last part of the packet

- Spacewire 2: area2 initialization
- Spacewire 2: storage, the area 2 becomes full but the packet is not entirely stored
- Spacewire 2: storage of the last part of the packet into the new area
- Check the memory for the last packet part

4.3.1.34 TX AHB master interface test 8: the FSM reads a packet size = 0

- Spacewire 1: TX AHB master mode, descriptor programmation
- Spacewire 1: the FSM reads a packet size = 0
- Spacewire 1: the FSM reads the next packet address
- Spacewire 1: the FSM reads the next packet then transmits it
- Check in the memory for the next packet

4.3.1.35 Test 1 of the interface between the AHB FIFO and the TX FIFO

Test the abort_packet signal in WR_SIZE state.

- Spacewire 1: TX AHB slave mode
- Spacewire 2: initialization of the memory area
- Emu_ahbmst: writes the packet size
- The Abort_packet flag is asserted while the FSM of the ahb_tx_int block is in the WR_SIZE state

4.3.1.36 Test 2 of the interface between the AHB FIFO and the TX FIFO

Test the abort_packet signal in WR_DAT state.

The same scenario as in 4.3.1.35, but the abort_packet flag is asserted in the WR_DAT state.

4.3.1.37 Test of the TX FIFO flush when the link is disabled

- Fill the TX FIFO with data
- Disables the link while the TX FIFO is not empty

- Observe the TX FIFO flush

4.3.1.38 Test 1 to increase the test coverage: test of the sw_reg block

- RD/WR accesses to the TIMEOUT register
- RD access to the CUR_BUF_END register
- WR access to the TIMECODE register
- RD/WR accesses to invalid register address

4.3.1.39 Test 2 to increase the test coverage: adds an EEP to the RX FIFO

- Spacewire 1: transmission of data to the Spacewire 2
- Spacewire 2: link disabled while it receives data
- Spacewire 2: an EEP is added into the RX FIFO

4.3.1.40 Test of the initialization protocol to increase the test coverage

- Go from ERRORWAIT state to ERRORRESET state
- Go from READY state to ERRORRESET state
- Go from STARTED state to ERRORRESET state
- Go from CONNECTING state to ERRORRESET state
- Go from RUN state to ERRORRESET state
- Generation of ESC error, character sequence error, outstanding error, credit error, disconnection error

4.4 SIMULATION DESCRIPTION

To compile the VHDL code for simulation, go into the /your_path/simenv1 directory then type "compile". Do the same in the /your_path/simenv2 directory.

4.4.1 Simulation without the host interface

The Austrian Aerospace testbench is in the /your_path/tbref directory. The script files are in the /your_path/codrf directory. The spacewire emulator is in the /your_path/spwref directory.

To execute these simulations, go to the /your_path/simenv2 directory, compile the test bench by typing "compile" then type "RUN" to launch the simulation.

Then, check the transcript window of Modelsim.

4.4.2 Simulation with the host interface

The test bench *tb5.vhd* is in the /your_path/tb directory.

The sofmem block uses a file named *ram.dat* that contains the elements of a linked list. The data in this file are randomly generated by using the script *generate.csh.* The script files *sw.tb* and *constante.tb* contain commands for the emulators. The user must type "tbpp sw.tb . . $PROJ/archi/spec_emu" to update testbench, each time the *sw.tb* or *constante.tb* file is modified.

To execute the simulation, go to the /your_path/simenv1 directory, compile the test bench if necessary by typing "compile" then type "run_simu5" to launch the simulation.

The transcript file shall not contain any error message.

4.5 SIMULATION REPORT

The VHDL simulation has been performed. All the functions have been checked. No functional error has been detected.

5 XILINX SYNTHESIS

Synthesis scripts are in your_path/synth

5.1 SET-UP AND DEVICE

Synplify is used for the synthesis.

The Xilinx component must be specified, the following component was chosen for our breadboard :

- Technology VIRTEX-E
- Part XCV2000E
- Package BG560
- speed_grade -6

5.2 COMPILATION AND MAPPING OPTIONS

The following options are set in Synplify :

- default_enum_encoding = default
- symbolic_fsm_compiler = true
- resource_sharing = true
- top_module = “spacewire”
- fanout_limit = 100

5.3 CONSTRAINTS

The constraints are:

- clk_txin = 110 MHz
- clk_tx (internal) = 110 MHz
- clk_sw (system clock) = 25 MHz
- clk_rx (internal) = 55 MHz

5.4 RESULTS

Performance Summary

```
                    **********************


                    Requested     Estimated     Requested     Estimated
Clock               Frequency     Frequency     Period        Period
Slack
------------------------------------------------------------------------------
--
clk_tx              109.9 MHz     106.7 MHz     9.1           9.4           -
0.3
c_rx.clk_rx         55.0 MHz      84.0 MHz      18.2          11.9          6.3
clk_sw              25.0 MHz      40.7 MHz      40.0          24.6
15.5
clk_txin            109.9 MHz     119.0 MHz     9.1           8.4           0.7
==============================================================================
==
```

```
Resource Usage Report
Mapping to part: xcv2000ebg560-6
Cell usage:
FDCE                533 uses
FD                  37 uses
FDC                 302 uses
FDC_1               10 uses
FDCE_1              1 use
FDP                 10 uses
FDE                 25 uses
FDPE                36 uses
GND                 1 use
VCC                 1 use
```

```
FDR                 36 uses
MUXF5               15 uses
MUXF6               1 use
XORCY               250 uses
MUXCY_L             337 uses
FDRE                36 uses
FDS                 16 uses
MULT_AND            51 uses
keepbuf             2 uses
MUXCY               4 uses
```

```
I/O primitives:
OBUF            247 uses
IBUF            124 uses

BUFGP           2 uses

I/O Register bits:                  1
Register bits not including I/Os:   1041 (2%)

Internal tri-state buffer usage summary
BUFTs + BUFEs: 36 of 19200 (0%)

RAM/ROM usage summary
Dual Port Rams (RAM16X1D): 77

Global buffer usage summary
BUFGs + BUFGPs: 2 of 4 (50%)

Mapping Summary:
Total  LUTs: 1973 (5%)

Found clock clk_tx with period 9.09091ns
Found clock clk_sw with period 40ns
Found clock clk_txin with period 9.09091ns
Found clock clk_rx with period 18.1818ns
```

5.5 SYNTHESIS CONCLUSION

The synthesis is successful. The expected transmission rate is about 100 Mbits/s, so the timing report is satisfactory.

6 XILINX PLACE AND ROUTE

6.1 PRESENTATION

The place and route task is performed in the /your_path/routage directory.

The input netlist file is *spacewire.edf.*

Note: The AMBA interface is connected to the ports. So the timings on the AMBA bus are not taken into account.

6.2 CONSTRAINTS

Input TX clock (clk_txin) : 100 MHz

System clock (clk_sw) : 20 MHz

TX clock obtained after divide of clk_txin (clk_tx) : 100 MHz

RX clock (clk_rx) : 50 MHz

```
NET "clk_tx" TNM_NET = "clk_tx";
TIMESPEC "TS_clk_tx" = PERIOD "clk_tx" 10 ns HIGH 50 %;
NET "clk_txin" TNM_NET = "clk_txin";
TIMESPEC "TS_clk_txin" = PERIOD "clk_txin" 10 ns HIGH 50 %;
NET "c_rx/clk_rx" TNM_NET = "c_rx/clk_rx";
TIMESPEC "TS_c_rx_clk_rx" = PERIOD "c_rx/clk_rx" 20 ns HIGH 50 %;
NET "clk_sw" TNM_NET = "clk_sw";
TIMESPEC "TS_clk_sw" = PERIOD "clk_sw" 50 ns HIGH 50 %;
```

The two clocks clk_tx and clk_rx must be routed by using "backbones" of the Virtex structures. The following constraints must be added:

```
NET "clk_tx" USELOWSKEWLINES;

NET "c_rx/clk_rx" USELOWSKEWLINES;
```

6.3 RESULT

6.3.1 Device utilization summary

```
  Number of External GCLKIOBs         2 out of 4      50%
  Number of External IOBs           371 out of 404    91%
    Number of LOCed External IOBs    0 out of 371     0%


  Number of SLICEs                1332 out of 19200   6%


  Number of GCLKs                   2 out of 4      50%
  Number of TBUFs                  36 out of 19520   1%
```

6.3.2 Constraint report

```
 Constraint                                | Requested  | Actual     | Logic
                                           |            |            | Levels
--------------------------------------------------------------------------------
```

```
  TS_clk_tx = PERIOD TIMEGRP "clk_tx"  10 n |            |            |
  S   HIGH 50.000000 %                      |            |            |
--------------------------------------------------------------------------------
  TS_clk_txin = PERIOD TIMEGRP "clk_txin"   | 10.000ns   | 9.172ns    | 5
  10 nS   HIGH 50.000000 %                  |            |            |
--------------------------------------------------------------------------------
  TS_c_rx_clk_rx = PERIOD TIMEGRP "c_rx/clk | 20.000ns   | 10.122ns   | 6
  _rx"  20 nS   HIGH 50.000000 %            |            |            |
--------------------------------------------------------------------------------
  TS_clk_sw = PERIOD TIMEGRP "clk_sw"  50 n | 50.000ns   | 19.761ns   | 7
  S   HIGH 50.000000 %                      |            |            |
--------------------------------------------------------------------------------

All constraints were met.
Dumping design to file spacewire.ncd.

All signals are completely routed.

Total REAL time to PAR completion: 4 mins 2 secs
Total CPU time to PAR completion: 4 mins 1 secs

Placement: Completed - No errors found.
Routing: Completed - No errors found.
Timing: Completed - No errors found.
Timing summary:
Timing errors: 0  Score: 0
Constraints cover 35204 paths, 0 nets, and 6738 connections (75.5%
coverage)
Design statistics:
   Minimum period:  19.761ns (Maximum frequency:  50.605MHz)
```

6.4 LAYOUT CONCLUSION

The place and route is successful. The timings are met.

6.5 CAO TOOLS CONFIGURATION

The configuration of the tools used by Astrium to develop the spacewire core is the following:

- VHDL simulator : Mentor Modelsim version 5.5a
- Synthesis tool : Synplify version 6.0
- place and route tool : Xilinx Design Manager 4.1i

-----%-----%-----%-----%-----