esa DOCUMENT

document title/ titre du document

System on Chip development based on Magillem 2.3SE

prepared by/*préparé par* Mattias Carlqvist

reference/*réference*
issue/*édition* 1
revision/*révision* 0
date of issue/*date d'édition* 12-12-2005
status/*état* Final
Document type/*type de document* YGT final report
Distribution/*distribution*

European Space Agency
Agence spatiale européenne

ESTEC
Keplerlaan 1 - 2201 AZ Noordwijk - The Netherlands
Tel. (31) 71 5656565 - Fax (31) 71 5656040

APPROVAL

Title *Titre*	System on Chip development based on Magillem 2.3SE	issue *issue*	1
		revision *revision*	0

Author *Auteur*	Mattias Carlqvist	date *date*	12-12-2005

approved by *approuvé by*	Laurent Hili	date *date*	19-12-2005

CHANGE LOG

reason for change /*raison du changement*	issue/*issue*	revision/*revision*	date/*date*

CHANGE RECORD

Issue: 1 Revision: 0

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*

Abstract

The objective of this work is to design a System on Chip (SoC) using a software tool, Magillem 2.3SE. Answers to questions related to this design activity, IP reuse, IP integration, hardware design flow and software refinement will be found in this report. A general platform (satellite computer) has been implemented using both IPs provided by the tool plus already developed ones related to ESA.

In order to validate both the design methodology and the tool the SoC has been tested on a Xilinx FPGA.

AMBA and OCP standards are used for communication in the SoC during this development. SpaceWire is used for external communication.

FOREWORD

All work has been done at the micro electronic section/data system division, ESTEC, the Netherlands. This is the technology center of the European Space Agency (ESA). This work is a continuation from a previous stagier. Efforts have been focused on using the Magillem tool with its limitation and keep the source code intact. Unfortunately this prolonged the design time and still a lot of manual work was necessary.

I would like to thank all the people that I have worked with during this trainee stage. Especially **Agustin Fernández León** for accepting me for this YGT position. My supervisor **Laurent Hili** for his knowledge in SoC design methodologies, the work proposal and support. **Roland Weigand** for his excellence in VHDL and experience advices during synthesis, place and route and finally hardware debugging.

Another big “thank you” to **Michele Satriani** for his fast support regarding all computer related issues that arose during this year.

Finally a huge thanks to the Prosilog support team with **Antony Vielmon** and **Emmanuel Vaumorin** as main ESA communicators and **Inga-Lena Hannukka** for spelling and grammar correction of this report.

_____

Mattias Carlqvist, Stockholm, December 2005

TABLE OF CONTENTS

1 INTRODUCTION

In order to deal with the complexity during SoC developments, many different aspects must be taken into account. Aiming for higher performance and more functionality will also introduce more risk and longer development time. One way to avoid this is to reuse already proven IPs and integrate them in the new system.

Magillem 2.3SE is a software tool developed by PROSILOG made to ease this task, among several others. This report will give an overview on how a system design can be made using this tool. All IPs used will be described in two sections; one related to the library provided by PROSILOG and the other with ESA related IPs.

A traditional hardware flow has been used to test and validate the design on a Xilinx FPGA. This part of the work is to prove that the complete chain is valid, from schematic to a working circuit.

When mention a SoC development it is important to remember the corresponding software development. For the software refinement several interesting features of Magillem are used. This work has been more focused on hardware so not all the refinement steps have been performed.

The systemC library has not been assessed during this stage and in the beginning it was not clear how to co-simulate VLSI languages and systemC together. The new methodology using systemC and co-simulation has been addressed in a parallel project [1] within the microelectronics section.

2 DESCRIPTION OF THE SYSTEM

2.1 Introduction

This system is a basic platform containing all the necessary building blocks for a small satellite computer. One microprocessor is for control purposes and computation. Two serial data communication IPs compliant with the SpaceWire standard [2] are used for external communication. The backbone of the system is based on AMBA 2.0 standard [3]. Even OCP standard [4] IPs are connected to this backbone using adaptors. All IPs used will be described in detail in chapter 3.

When this platform is done it is easy to adopt it to all different kinds of applications and needs. An example given in [1] showing how straight forward the integration of a new IP is. Complex designs based on several or multi-layer backbone buses are easy to achieve.

Appendix A shows the Magillem schematic of the design.

2.2 Test program

In order to test the design a loop back configuration is used. Both SpaceWires are connected together externally. Both SpaceWires are configured as Direct Memory Access (DMA); they transfer the data direct to the memory as soon as they receive data from the link. The load on the CPU will be kept to a minimum. The test program executes as follow:

- Setup data are sent to both SpaceWires.
- Wait until serial communication is established.
- Start the loop transfer by sending a data packet to SpaceWire one.
- Packet passes the external link and is received by SpaceWire number two.
- Data are stored in the on-chip memory using DMA and when finished, one interrupt is sent to the microprocessor from SpaceWire number two.
- When the processor receives an interrupt from the SpaceWire it fetches the data packet from the on-chip memory and sends it back to the same one.
- SpaceWire one will also store the data packet in the on-chip memory and send one interrupt.

The loop of data packets will continue until the on-chip memory is full and the processor crashes when reading outside the memory boundary.

3 IP INTEGRATION AND REUSE

3.1 Introduction

How IPs can be reused in SoC design is a key element in trying to tackle the complexity and development costs. Today for IP exchange and integration exist; one example is the SPIRIT consortium.

In this section all the IPs used in the design will be described in detail and how Magillem handle them.

3.2 Foreign IP Integration in Magillem

Magillem use XML to describe the interface of an IP. This file must be created the first time an IP is imported into the tool. There are two ways to proceed:

- Use an IP to XML graphical wizard inside Magillem, Figure 1.
- Write the file manually inside a text editor.

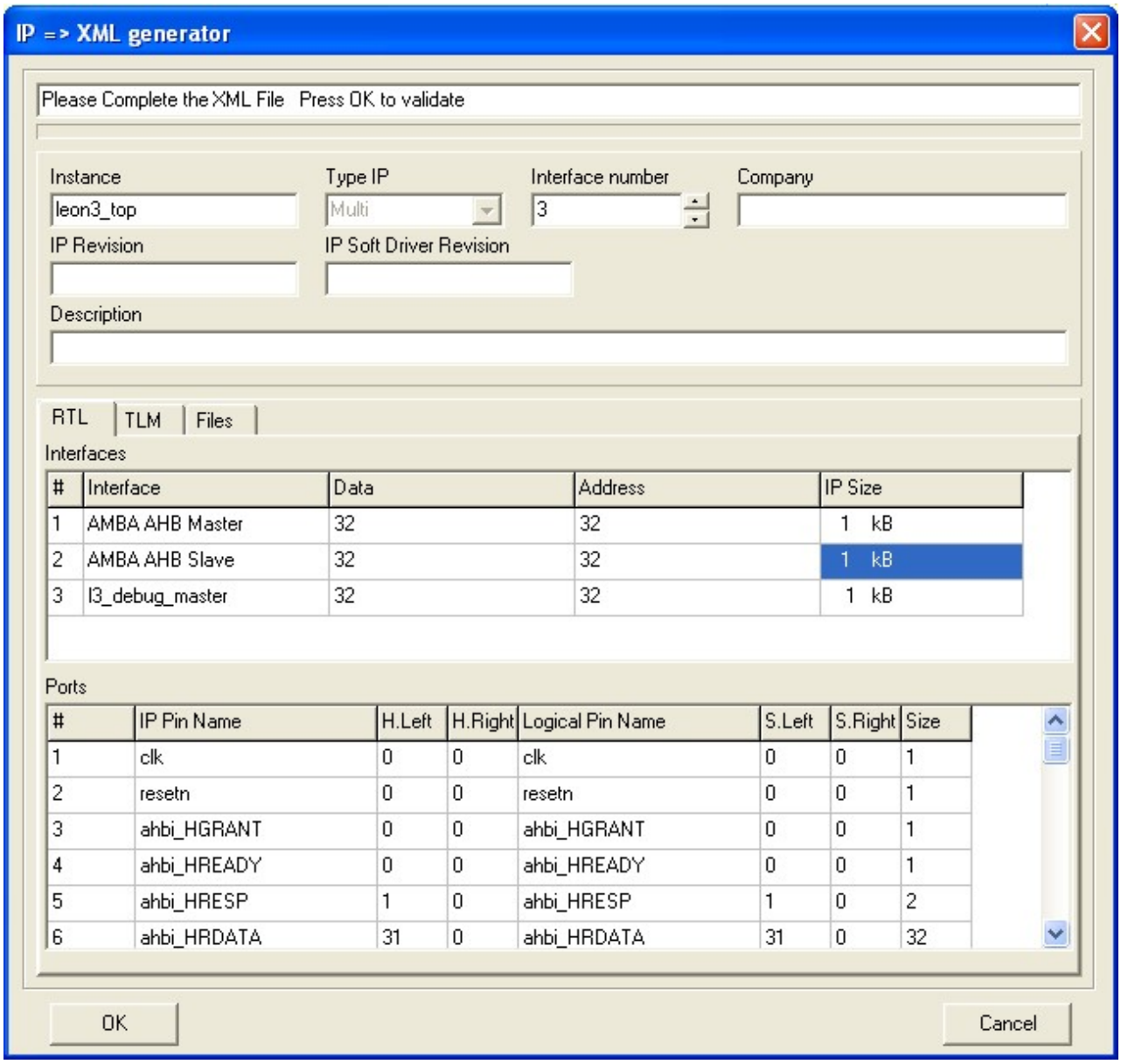


Figure 1 IP to XML generator

Unfortunately the tool accepts only basic VHDL coding style for the IP top file. This is mainly due to two reasons:

- Tool accepts only std_logic and std_logic_vector VHDL types.
- No records or packets are supported.

The solution is to do a Magillem compliant wrapper file around the IP (or modify the original top file). This has been the case for all foreign IPs during this project. Files are found in Appendix B.

3.3 Open Core Protocol (OCP)

The basic concept of OCP is to define a high-performance bus-independent interface between IP cores to reduce design time, design risk and manufacturing costs for SoC designs. Several big companies have taken part in specifying this protocol [4].

In order to communicate over the AMBA AHB bus using the OCP protocol, Magillem generates the necessary bridges. Some performance disadvantage occurs in terms of speed and gates. But the advantage is to have a bus architecture independent IP.

3.4 Import non bus compliant IPs using IP Creator OCP

Magillem provides a tool, OCP IP Creator [5], to encapsulate non compliant IPs with OCP interface. See 3.6.3 for how this tool is used to encapsulate the SpaceWire from Dundee with an OCP interface. Two common interfaces are compatible with the tool; FIFO and memory like. Unfotunately dual port memory interface is not supported.

This is a powerful tool that shortens the IP integration design time.

3.5 IPs provided by Magillem

This section gives a short description of the IPs used from the Magillem library. More detailed information can be found in magillem_docs.pdf/IP portfolio.

3.5.1 AHB DUMMY MASTER

This IP is required when a slave IP can give a split transfer response. If all other masters have received a split the controller must grant the bus to someone; the dummy master.

3.5.2 AMBA AHB CONTROLLER

The AMBA Controller connects all the main IPs together. All controller settings and information regarding IP address and size are shown in Figure 2. There are three different arbitration schemes to choose from:

- Fixed Priority
- Round Robin
- FIFO

On top of that it is possible to enable a second arbitration, TDMA. It will allow allocating a certain percentage of the total bus bandwidth to the different masters. The master order for the arbitration scheme and TDMA does not need to be the same.

It is also possible to specify single (classic) or multi-layer AHB controller. The multi-layer one is more complex but may result in a big performance increase because each master / slave pair can transfer data simultaneously.

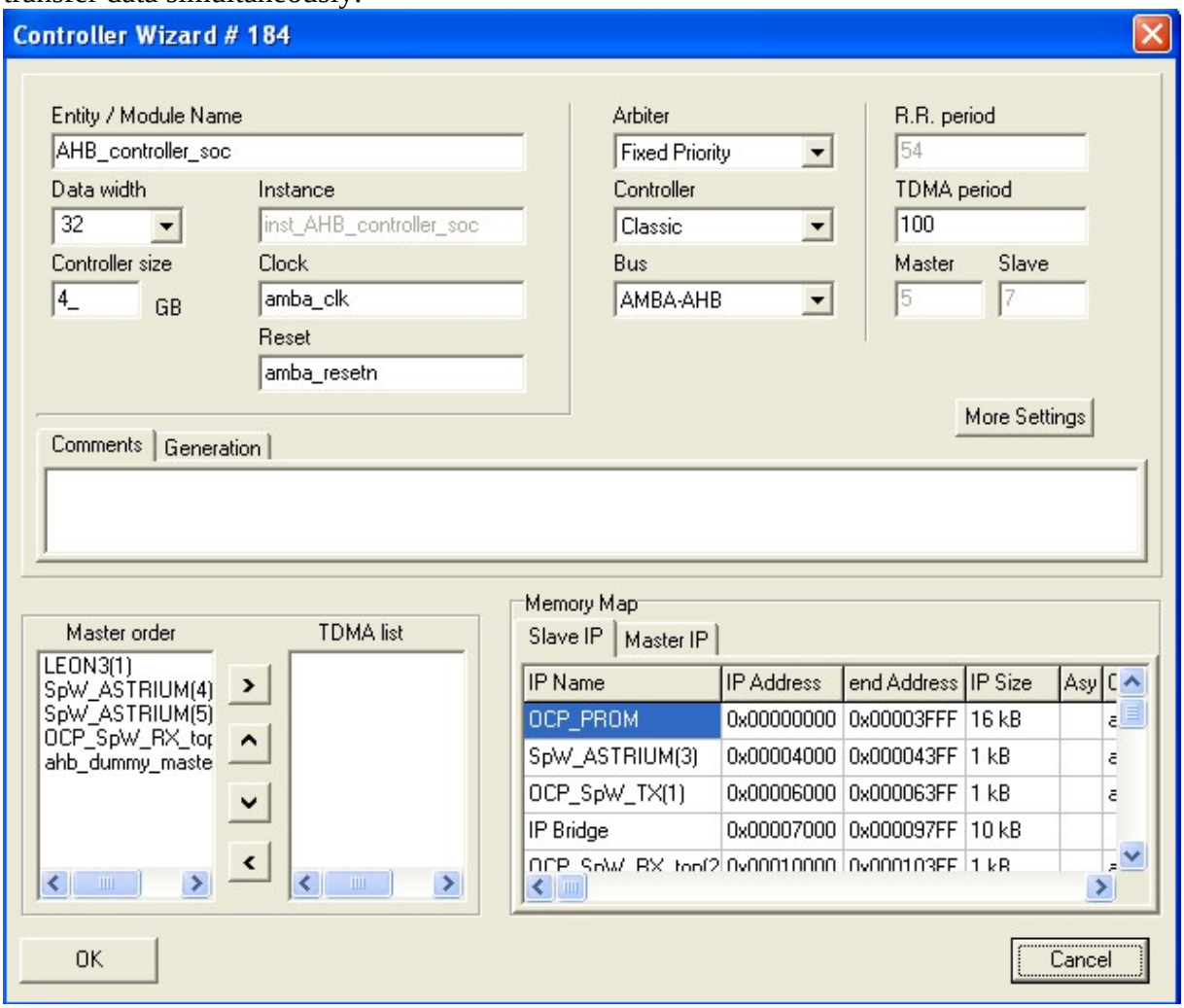


Figure 2 AHB Controller settings

3.5.3 AMBA AHB TO APB BRIDGE / APB CONTROLLER

The APB bus has only one master, the AHB/APB bridge itself. The bridge appears as a slave on the AHB side and as a master on the APB side, Figure 3.

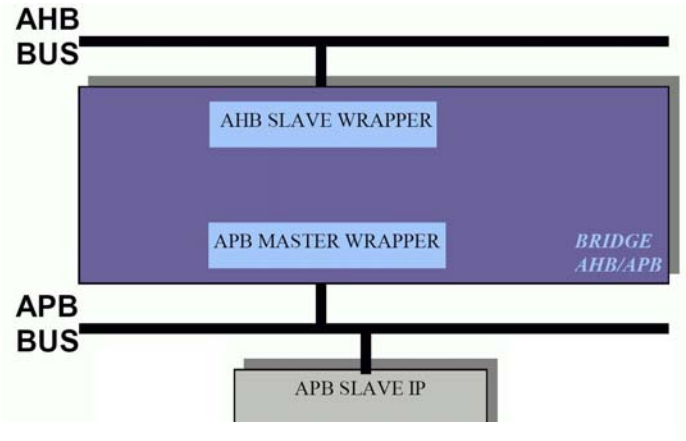


Figure 3 AMBA AHB to APB bridge

A bridge wizard window is used for setting all the bridge properties, Figure 4. The memory size of the APB bus is 10kb and the memory start address is 0x00007000. There is no possibility to divide this 10kb memory area into several smaller areas that are not memory aligned. As a result of this, APB slaves (in this example) can only be mapped between 0x7000 and 0x97FF. If larger memory separation than specified in the slave size between the APB slaves is necessary, the only solution is to use several bridges and APB busses.

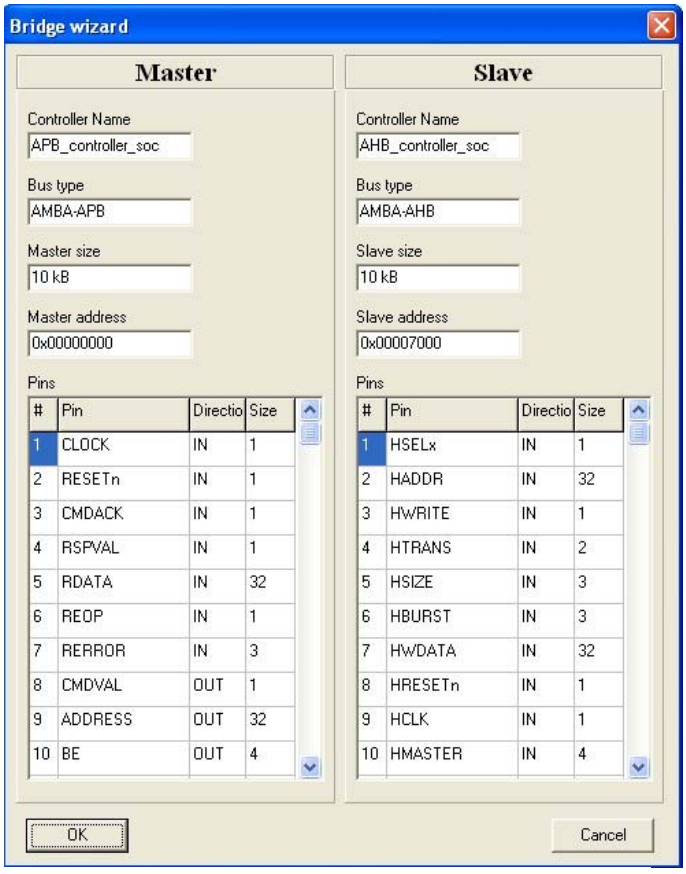


Figure 4 AHB to APB bridge wizard

3.6 ESA related IPs

This section describes the ESA related IPs and how they are adopted to be integrated into the System on Chip. All information and access to these IPs are found in [6] and [7].

3.6.1 LEON3

This is the most recent version of the processor provided by Gaisler Research in Gothenburg, Sweden. It is a highly configurable IP and only the basic features have been implemented during this project. These include the integer unit with 1kb data and instruction cache plus a 16x16 multiplier. More detailed information regarding the processor can be found in [8]. Gaisler Research also provides an IP library, GRLIB, and all the necessary tools to build a SoC.
Note: LEON3 in this report is only the CPU core mentioned above and should not be mistaken for a complete SoC based on LEON/GRLIB such as the AT697 chip [9].

LEON3s interface is based on AMBA 2.0 plus additional sideband signals. One purpose of these signals is to ease the IP integration using GRLIB and not depend on a global source for the memory map of the system. Magillem uses a global source (AHB controller) for this, so LEON3 is not the most suited processor to be used in third party CAD tools. The only option to compile and run C software on LEON3 is to include the sideband signals in all IPs (memory information is located both in the IPs and in the AHB memory controller) or modify the software linker. None of these options have been implemented. All software has been done in Assembler to avoid this problem.

The AMBA interfaces are grouped into records which make the integration harder. See Appendix B for Magillem compliant LEON3 VHDL wrapper.

3.6.2 SPACEWIRE ASTRIUM

This IP is developed by ASTRIUM in Vélizy, France. It refers to the SPW-AMBA ESA core. It implements the SpaceWire standard [2] and has four AMBA interfaces, one APB for configuration and three AHB for the data transfer. On the transmit side two options are available; TX DMA mode and TX slave mode. During this project only the TX slave mode has been used. On the receive side the data is stored in the host memory using DMA transfer. Two different memory areas may be used. **Note: There is no way to manually switch memory area.** Lack of this feature limits the use of this IP as an AHB master during boot/application software downloads to the host CPU. More details about this IP may be found in [10].

Integration into Magillem is very simple but the restrictions mentioned in 3.2 forces the use of a wrapper.

3.6.3 SPACEWIRE DUNDEE

University of Dundee has developed this IP and it is referred to as SPWb. It implements the same protocol as the ASTRIUM one, with one major difference; this is only a SpaceWire codec implementing the function without any compliant bus interface. To integrate an IP like this into the SoC requires a lot of effort. The IP creator OCP tool makes this work easier but far from autonomous. OCP is chosen as target for the interface due to three reasons:

- General interface to increase the IP reuse
- IP Creator tool generate this interface
- Magillem provide wrappers between AHB and OCP

In order to interface the existing SpW IP, developed by University Dundee, in the Magillem environment, a bridge must be used. This bridge makes it possible to connect the SpW to any OCP 2.0 compliant bus. Two different bridges are used, one for the SpW TX side and one for the SpW RX.
Prosilog provides a tool, IP Creator OCP Version [5], for creating bridges between the OCP 2.0 protocol and non-compliant IPs. Two different interfaces are supported to the IP; FIFO and memory like. FIFO interface has been used for generating the two bridges. **Note: TIMECODE CONTROL is not implemented in the two bridges.**

Figure 5gives an overview of how the adaptation is made. The source code for the bridges can be found in Appendix C.

Note: OCP control and status signals are not included in the OCP to AHB Bridge, provided by Magillem. One solution is to implement a simple APB to OCP control/status Bridge, see Appendix D.

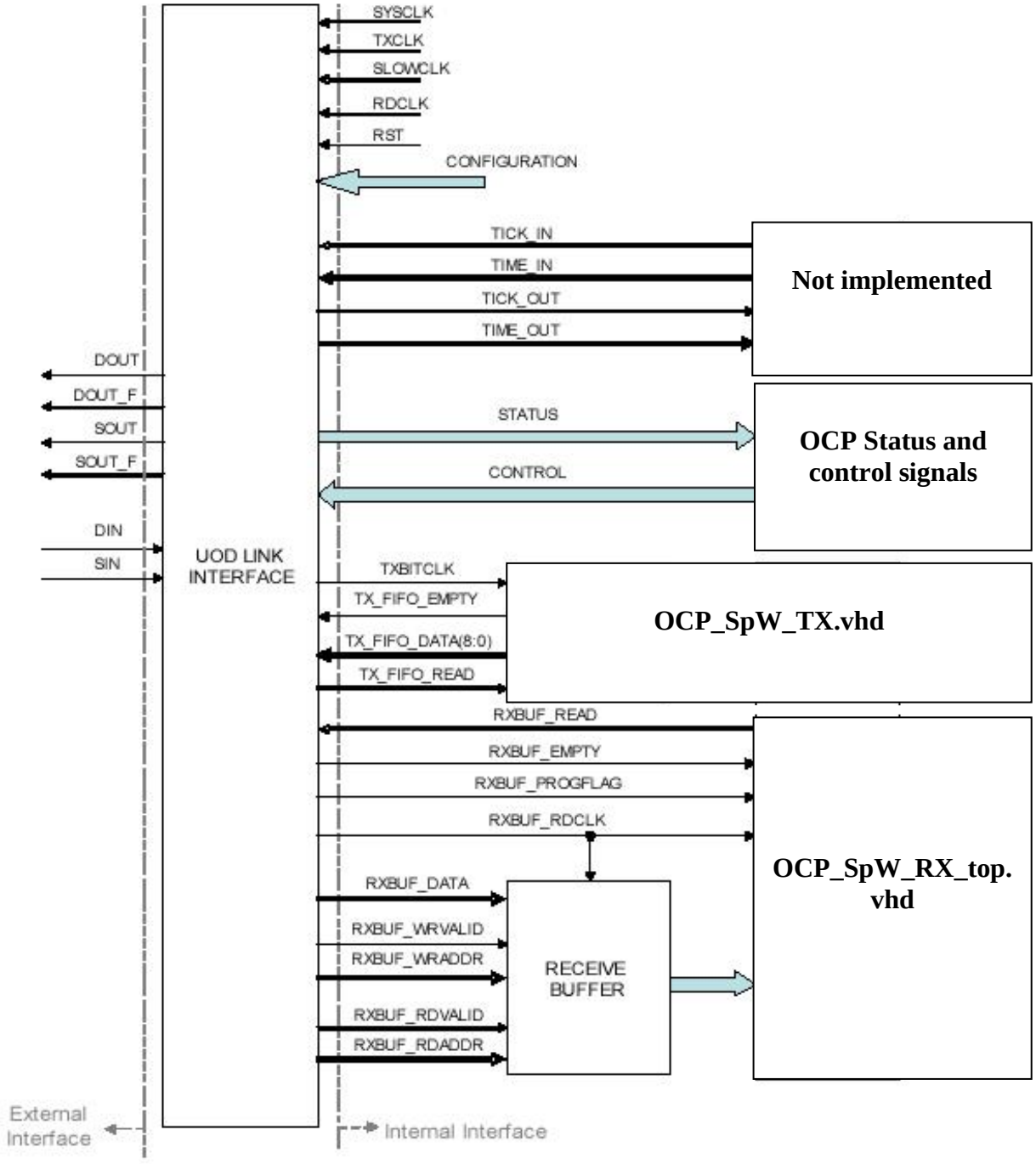


Figure 5 SpaceWire CODEC adapted with OCP interfaces

3.6.3.1 SpaceWire TX Bridge

Since this interface on the CODEC side already is a FIFO the connection is straight forward.

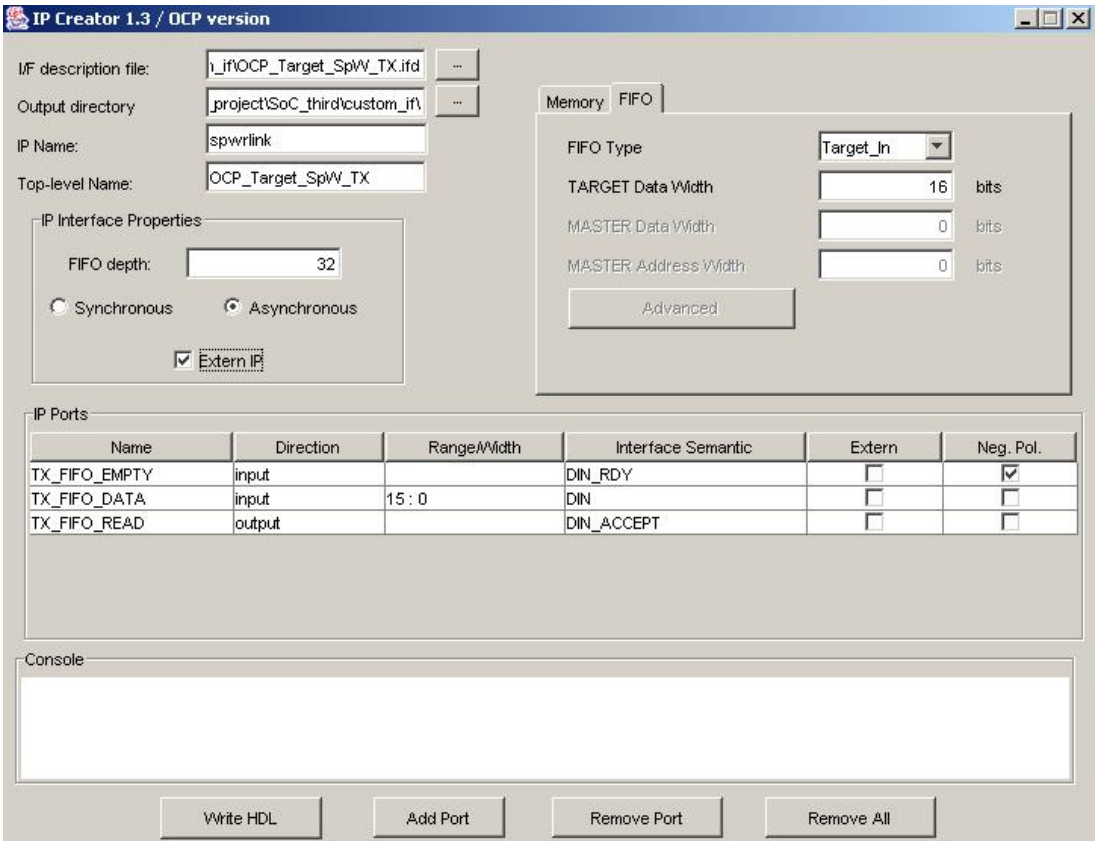


Figure 6 IP Creator settings for SpaceWire TX bridge

The asynchronous FIFO option makes it possible to interface the two different clock domains. The first one uses the AMBA_CLK and the second one the TXBITCLK. These settings are done in the Magillem tool when importing the generated bridge. The timing for data transferred to the SpW UOD Link Interface is shown below.

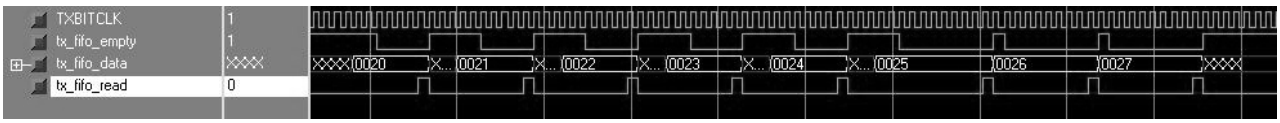


Figure 7 Transfer of 8 bytes from OCP Target SpW TX FIFO to the UOD Link Interface

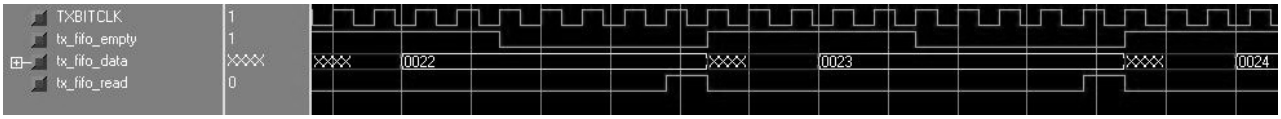


Figure 8 Zoom of the two first bytes transferred in Figure 7

3.6.3.2 SpaceWire RX Bridge

The design of the SpW RX bridge is little more complicated than the TX one. One reason is due to the fact that the SpW is master over the receive buffer. A dual port RAM memory must be used and the IP Creator tool has not a built-in interface for this. The other reason is that the RXBUF_READ signal must be asserted to receive any data at the output of the receive buffer.

Also the width adaptation from the Space Wires 9 to the AHB bus 32 and the End Of Packet (EOP) must be dealt with. The solution is to design a state machine that interface the CODEC receive buffer and provide a FIFO interface to the IP Creator tool. Details are found in Appendix C.

In order to make autonomous data transfers to the system memory from the SpW, the bridge is of OCP Master FIFO output type. To solve the difficulties listed above, a wrapper around the receive buffer is used. A state machine handles the dataflow between the SpW RX side and the OCP Master_Out FIFO interface, Appendix D entity Spw_RX. The IP Creator settings are shown in Figure 9. In order to have the SpW receive side synchronous to the AHB_CLK it is connected to the RDCLK.

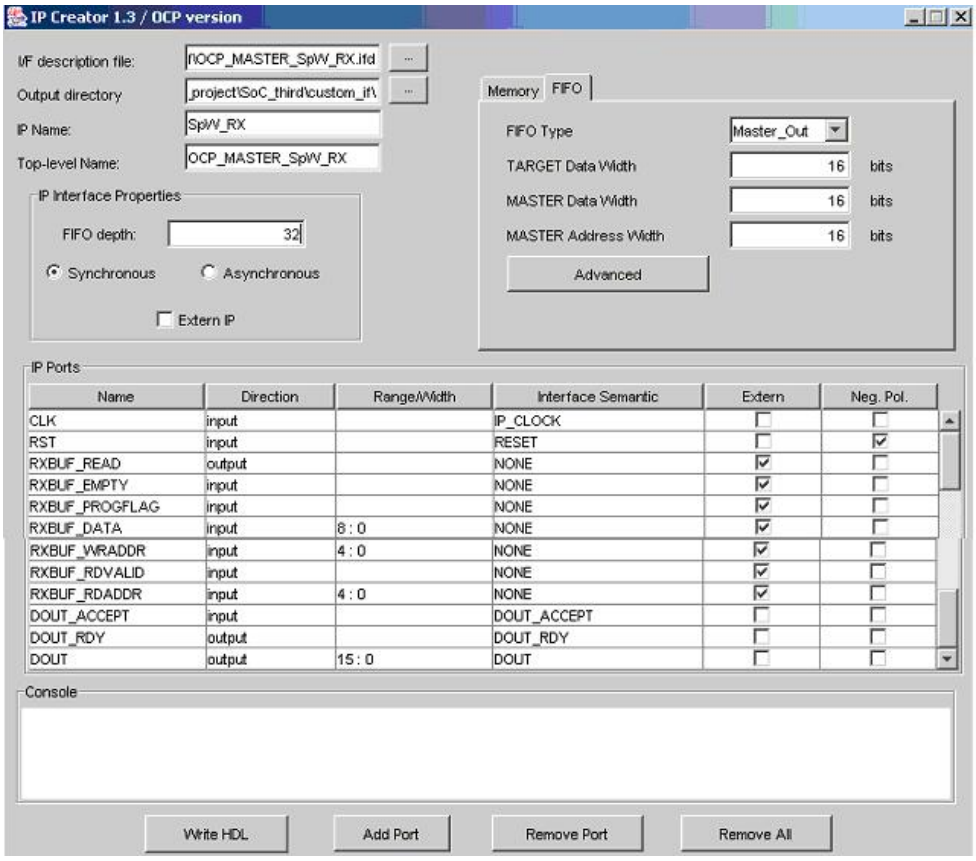


Figure 9 IP Creator settings for SpaceWire RX

In order to setup an OCP Master SpW RX an OCP slave port is used. The OCP Master has 10 different registers that must be properly initiated. Please refer to chapter 5.10.2 in [5] for specific information about these registers.

Configuration and status registers for the SpW Codec are shown below.

Control register 0x0

Bits [31:0]	Name	Function	Reset Value
31-30	Not used	N/A	N/A
29-25	RXBUF_PROGVAL	See [11]	0x0
24-19	TXRATE	See [11]	0x0
18	CFG_SLOW_CE	See [11]	‘0’
17-12	CFG_SLOWRATE_SYSCLK	See [11]	0x0
11-6	CFG_SLOWRATE_TXCLK	See [11]	0x0
5-0	CFG_MAXCREDIT	See [11]	0x0

Control register 0x4

Bits [31:0]	Name	Function	Reset Value
31-8	Not used	N/A	N/A
7-6	SEND_EOP	See [11]	0x0
5	RESET_INTERUPT_RX	See [11]	‘0’
4	START_RX	See [11]	‘0’
3	FLUSH_TX	See [11]	‘0’
2	AUTO_START	See [11]	‘0’
1	LINK_DISABLE	See [11]	‘0’
0	LINK_START	See [11]	‘0’

Status register 0x0

Bits [31:0]	Name	Function	Reset Value
31-20	Not used	N/A	N/A
19	CREDIT_RUN_ERR	See [11]	‘0’
18	ESCAPE_RUN_ERR	See [11]	‘0’
17	PARITY_RUN_ERR	See [11]	‘0’
16	DISC_RUN_ERR	See [11]	‘0’
15-0	STATUS	See [11]	0x0

Status register 0x4

Bits [31:0]	Name	Function	Reset Value
31-18	Not used	N/A	N/A
17-16	status_eop	The type of EOP received; “01” EOP “10”EEP	“00”
15-0	total_bytes_spw	The total number of SpaceWire bytes received	0x0

4 HARDWARE DESIGN FLOW

4.1 Introduction

Getting from a system specification to a working circuit requires several steps and programs. Figure 10 gives an overview of the hardware design flow and each step is described in this chapter. This flow is well known for prototyping on FPGAs.

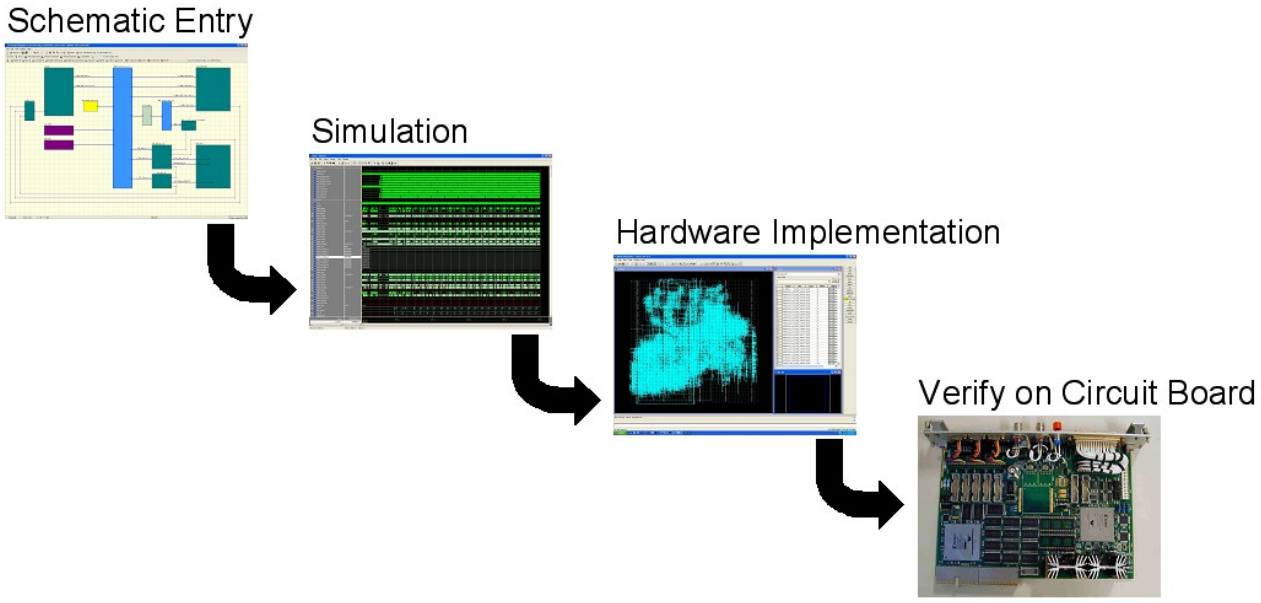


Figure 10 Hardware Design Flow

4.2 Schematic Entry

Magillem is used to define the System on Chip and its entire component. In a traditional flow this part is represented by VHDL/Verilog source code. All the following useful features are accessible through the graphical user interface:

- Use IPs from provided library
- Import custom IPs
- Define interfaces
- Define Bus architecture
- Interconnect all IPs
- Complete memory map of the system

Appendix A shows the schematic of the last design made. After a satisfactory schematic is built, Magillem generates the top VHDL file and bridges (if needed). Unfortunately no compilation script is generated by the tool.

4.3 Simulate the Design

ModelSim 5.8e is used for simulation. During this step all the HDL files are compiled and the design is then simulated. The result is displayed in a waveform window and used as validation (Figure 11).

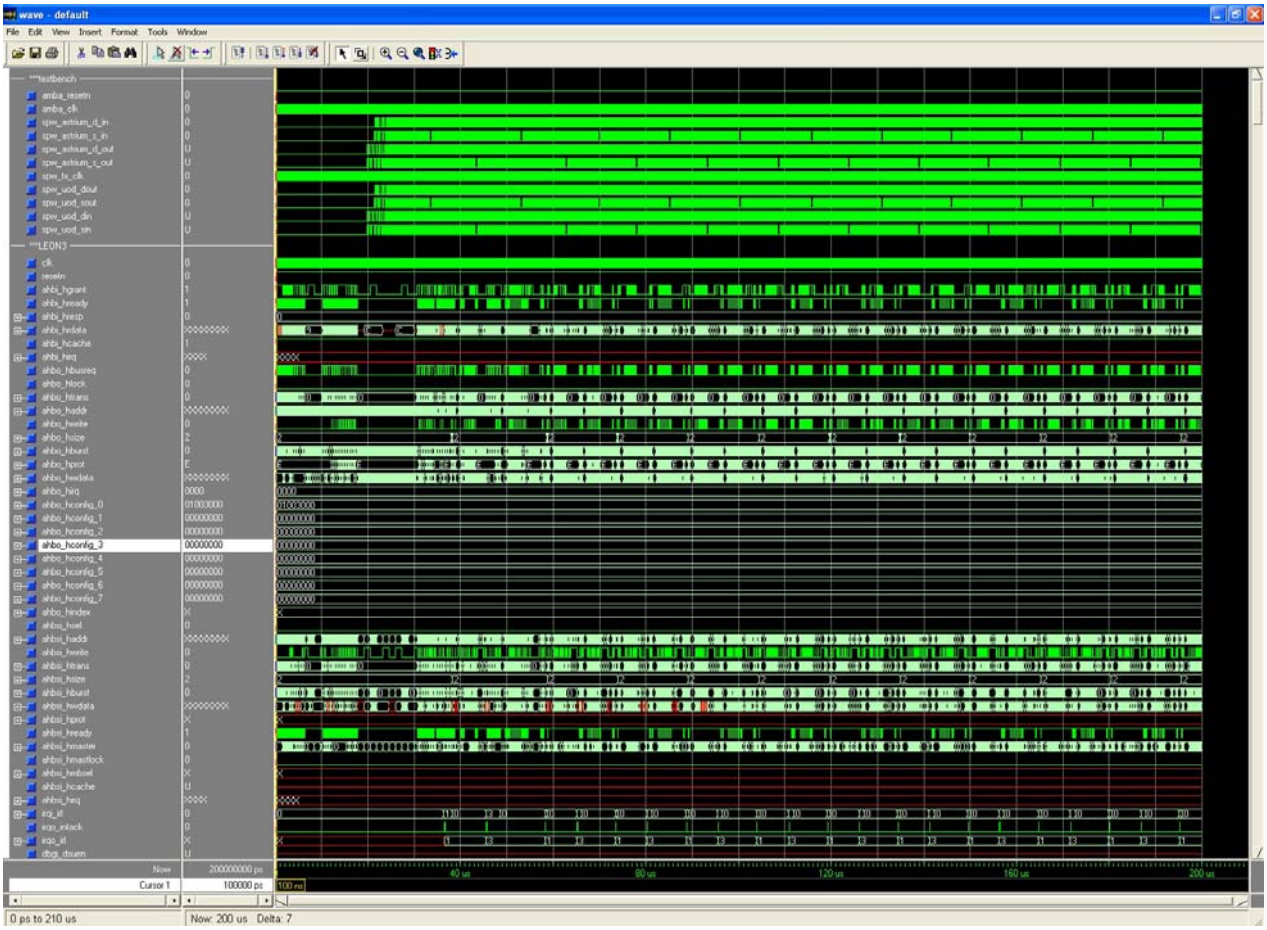


Figure 11 Waveform example

4.4 Synthesis

Synplify is used for synthesis of the SoC. This stage translates the VHDL source code into a net list. A specific target FPGA must be specified.

4.5 Place and Route

Xilinx ISE is used for the final step. Both layout on the chip and programming are done at this stage.

4.6 Validation of the Design

A board from ASTRIUM has been used during hardware testing (Figure 12 ASTRIUM BLADE board). Equipment from 4-Links and STAR-Dundee are used for communication and monitoring on the SpW link. The SpW link is tested up to 100Mbit/s.

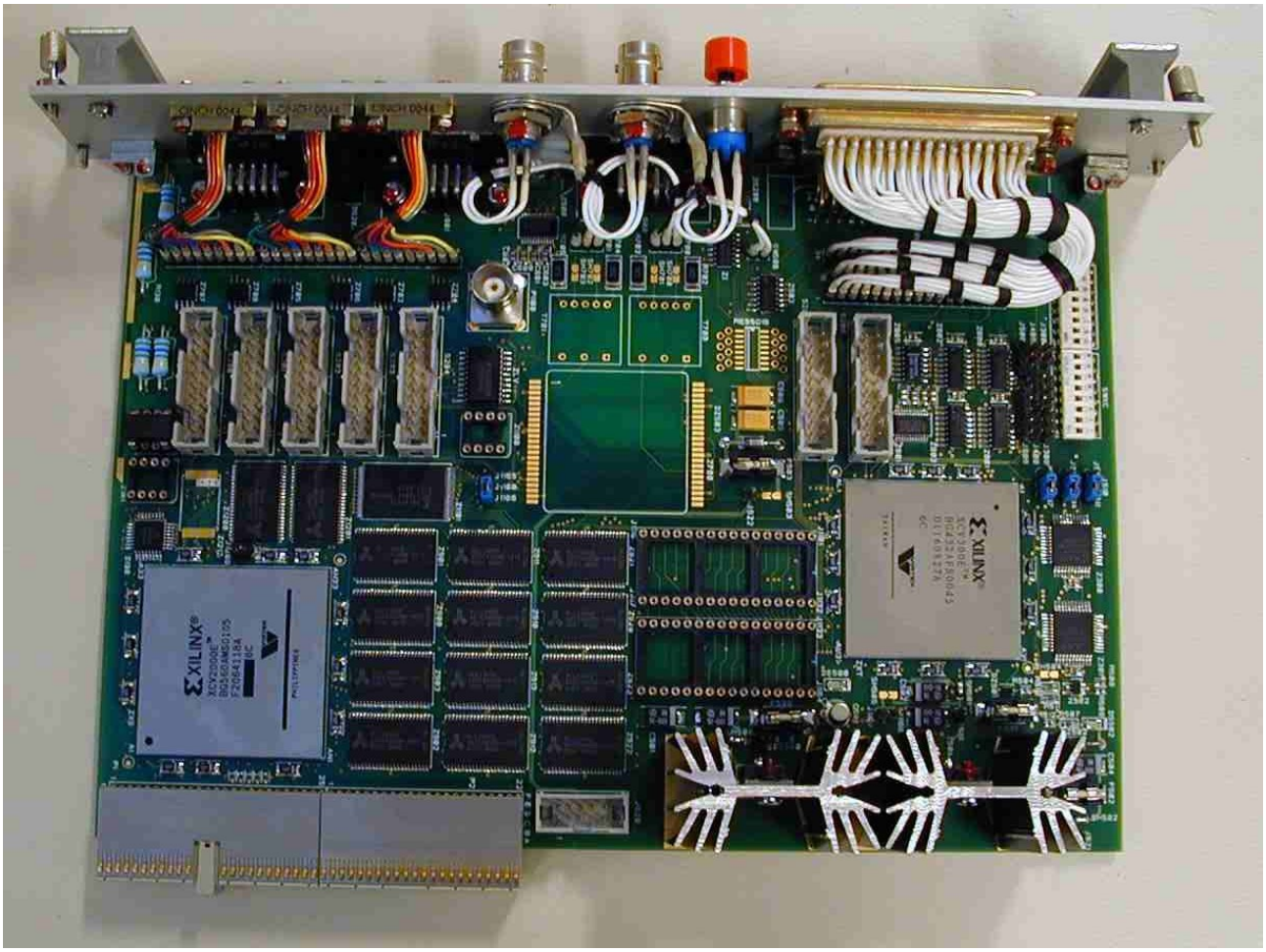


Figure 12 ASTRIUM BLADE board

5 SOFTWARE REFINEMENT

5.1 Introduction

One big part of a SoC development is the software. During traditional developments the software engineer starts when the hardware is almost finished. To reduce this gap Prosilog provides two solutions; Graphical User Interface (GUI) and Application Programmable Interface (API). Both these solutions replace the CPU during early development phases. This is a rather powerful way of sending and receiving data through the SoC without using any CPU. These solutions are compliant with the AMBA-AHB bus and the modelsim simulator. Figure 13 shows the software refinement steps.

Note: Both the GUI and API are only used during the simulation and verification phase of the project and can not be used for hardware implementation.

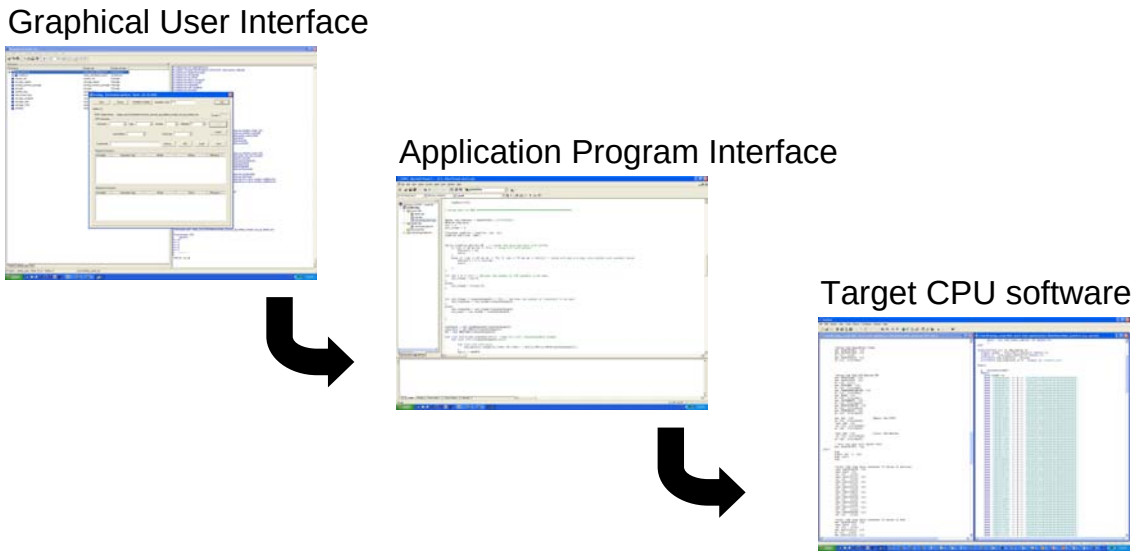


Figure 13 Software refinement steps

5.2 Graphical Software Interface

The GUI makes use of the foreign language interface (FLI) to interact with the modelsim simulator.
When the simulation starts the Prosilog GUI window appears, see Figure 14. From here, commands and different scenario files are loaded into a queue. Pressing the start button will execute the entire queue. It is possible to stop the simulation and load new commands and then start it again. However, the GUI blocks it until the “ModelSim Handle” button is pressed. The GUI provides a fast and easy way to interact with the rest of the SoC. This has been used building test

benches and to test the basic functions of the SoC. Synchronization to the simulation time is not provided and as result the repeatability of different simulations is reduced. Nevertheless this is a very useful tool as a first step for testing different functions.

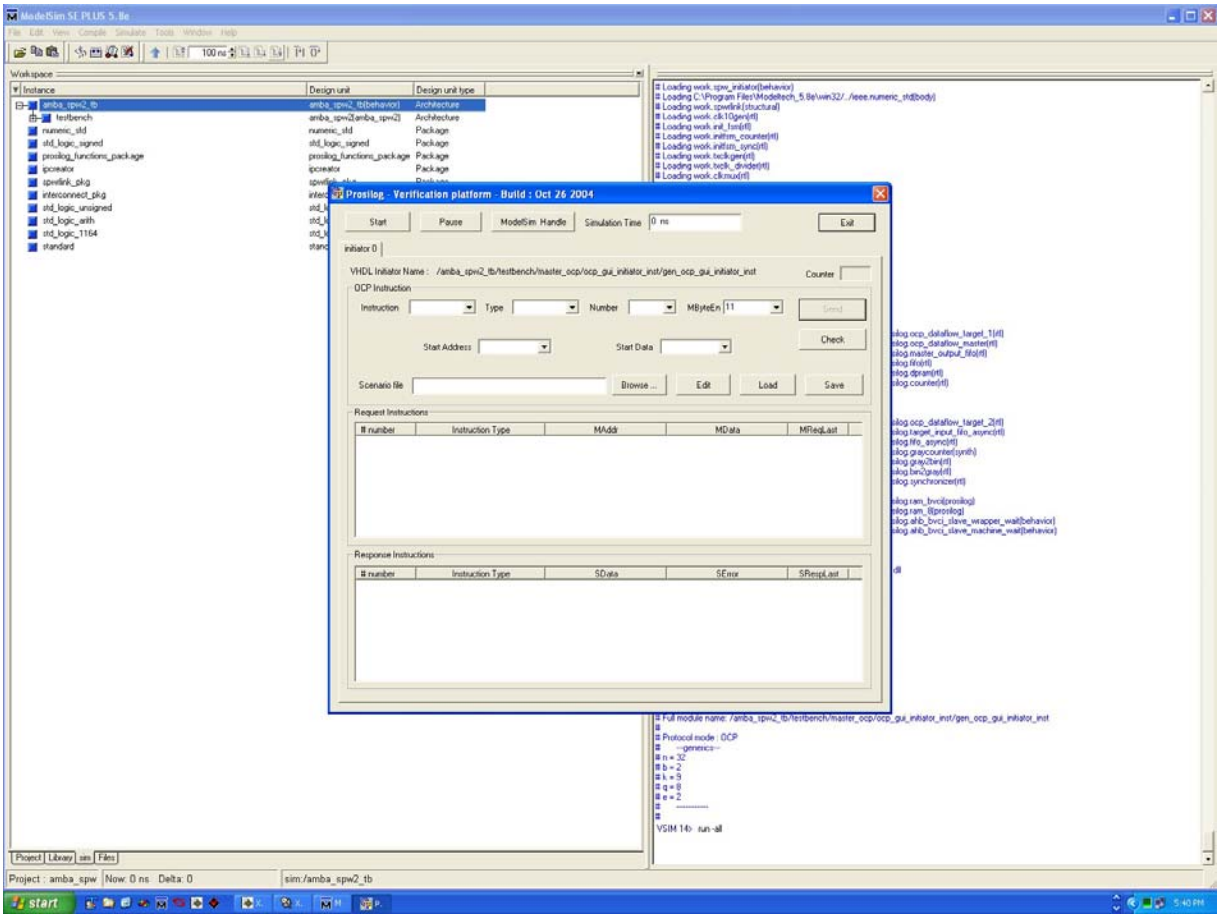


Figure 14 Graphical User Interface

An example of a GUI file(loop_OCP_Master_2_128.sce) to setup one SpaceWire Dundee IP to loop the data over the SpW link is shown below:

```
instruction 1 Write 0x9000 0x4000    0 Default 11 0
instruction 1 Write 0x9002 0x4000    0 Default 11 0
instruction 1 Write 0x9004 0x000F    0 Default 11 0
instruction 1 Write 0x9006 0x0       0 Default 11 0
instruction 1 Write 0x9008 0x0       0 Default 11 0
instruction 1 Write 0x900A 0x8       0 Default 11 0
instruction 1 Write 0x9010 0x1       0 Default 11 0
instruction 1 Write 0x9010 0x0       0 Default 11 0
instruction 1 Write 0x9012 0x1       0 Default 11 0
instruction 1 Write 0x900C 0x0       0 Default 11 0
```

5.3 Application Programmable Interface

When more complex software is to be used and tested the API is more suited than the GUI. In fact the API can be considered as a general CPU. It is built upon the C++ library SystemC. The following functions are supported:

- OCP wait function
- OCP read functions (contiguous, wrap, stream, unknown)
- OCP write functions (contiguous, wrap, stream, unknown)
- OCP idle functions (contiguous, wrap, stream, unknown)
- Interrupts (read and write acknowledge)

A C++ project must be set up in order to use the API. Only Visual Studio 6 is supported. After compilation a dll-file is created. Modelsim will call this dll-file during simulation. It is possible to set breakpoints and perform debugging from Visual Studio when running Modelsim. More detailed information can be found in [12].
This co-simulation between C++ source code and the SoC hardware compiled in Modelsim, is a very powerful tool for testing the software together with the hardware early in the development phase. As the API is based on the OCP protocol, different bus architectures may be evaluated using the application software. This may help the system engineer to perform an accurate trade-off before the final target hardware has been chosen.

The test program in 2.2 is shown below (UserThread_spwb.cpp):

```
#include "UserThread_spwb.h"

#define ARRAY_SIZE 10
#define DATA_SIZE 8

using namespace prosilog_api;

void
UserThread::onClock(){}

void
UserThread::onExit(){}

void
UserThread::run(){

        //Set the common variables
        MAddr address1[ARRAY_SIZE]= {0x8000,0x8002,0x8004,0x8006,0x8008,0x800A,0x8010,0x8010,0x8012,0x800C};  //
OCP_MASTER_SPW_1 setup address
        MAddr address2[ARRAY_SIZE]= {0x9000,0x9002,0x9004,0x9006,0x9008,0x900A,0x9010,0x9010,0x9012,0x900C};  //
OCP_MASTER_SPW_2 setup address

        MAddr stream_address1= 0x4000;  // OCP_TARGET_SPW_1 address
        MAddr stream_address2= 0x6000;  // OCP_TARGET_SPW_1 address

        MByteEn be[ARRAY_SIZE]= {0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};
        MByteEn name_be[DATA_SIZE]={0xff,0xff,0xff,0xff,0x0ff,0xff,0xff,0xff};
        MByteEn stream_be[DATA_SIZE]= {0xff,0xff,0xff,0xff,0xff,0xff,0xff,0xff};

        ocpResponse response1[ARRAY_SIZE];
        ocpResponse response2[ARRAY_SIZE];
```

```
		// *************** Start of Loop program ****************

		MData idata1[ARRAY_SIZE]= {0x6000,0x6000,0x80,0x0,0x0,0x0,0x8,0x1,0x0,0x0,0x1,0x0};
		MData idata2[ARRAY_SIZE]= {0x4000,0x4000,0x80,0x0,0x0,0x0,0x8,0x1,0x0,0x0,0x1,0x0};  // OCP_MASTER_SPW_2 setup data,
compare to GUI file: loop_OCP_MASTER_2_128.sce

		MData stream_data1[DATA_SIZE]= {0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27};  // OCP_TARGET_SPW_1 data

		//set initiator
		ocpSetValtimeCmd(0);
		ocpSetDefaultack();


		//Initiate the two OCP_MASTER_RX_SpW
		ocpWriteUnknown(address1,be,idata1,response1,ARRAY_SIZE);
		mti_PrintMessage("Data sent to OCP_MASTER_1");
		ocpWriteUnknown(address2,be,idata2,response2,ARRAY_SIZE);
		mti_PrintMessage("Data sent to OCP_MASTER_2");

   // Send the data to the OCP_TARGET_SpW_TX
		ocpWriteStream(stream_address1,stream_be,stream_data1,response3,8);

for (;;)
ocpWait(1);

}
		// ******** END of Loop program ************
```

5.4 Target CPU Software

In order to translate the API software to the target CPU drivers for the different OCP read and write functions must be developed. This work is out of the scope of this project. Instead all the software is translated by hand directly in assembler. Off course it is not suitable to work like this with larger software. But in order to compile and run C programs on the LEON3 the sideband signals must be used or the C-linker must be modified to exclude them.
One big drawback not using a C program is the very limit debug capability.

The entire source code that corresponds to the test program in 2.2 can be found in Appendix E. Parts of the code are shown below:

```
        !start the loop data transfer 32 bytes to UoD
        set 0x00006000, %l0
        !set 0x20, %l1
        !st %l1,  [%l0]
        set 0x11111111, %l1
        st %l1,  [%l0]
        set 0x22222222, %l1
        st %l1,  [%l0]
        set 0x33333333, %l1
        st %l1,  [%l0]
        set 0x01234567, %l1
        st %l1,  [%l0]
        set 0x55555555, %l1
        st %l1,  [%l0]
        set 0x66666666, %l1
        st %l1,  [%l0]
        set 0x77777777, %l1
        st %l1,  [%l0]
        set 0x88888888, %l1
        st %l1,  [%l0]

_trap_handler_spw_uod_eop:
        set 0x00008004, %l3         !RX status/control register address
        set 0x00006000, %l4         !UoD TX address
        set 0x0000FFFF, %l5
        set 0x00010000, %l6         !OCP master address
        set 0x00000001, %l7
        st %l7, [%l6+0x18]   !stop the Master
        ld [%l3], %g3               !load spacewire header from RX status register
        st %g3, [%g5]               !write spacewire header in memory
        set 0x00000034, %l7
        st %l7, [%l3]               !reset interrupt = '1'
        set 0x00000014, %l7
        st %l7, [%l3]               !reset interrupt = '0'

        !change order
        !ld [%l3], %g3              !load spacewire header from RX status register
        !st %g3, [%g5]              !write spacewire header in memory

!start of send back program
        and %g3, %l5, %g3           !mask out packet header
        !st %g3, [%g0]              !debug purpose
        udivcc %g3, 4, %g3 !determine the number of words to send back
        ble _send_back2             !don't sub if packet header is less than 4 bytes
        nop
        sub %g3, 1, %g3
```

6 RESULTS AND CONCLUSION

6.1 Overall

First of all results the most important one; A System on Chip has successfully been implemented with this new development tool and design methodology. Non compliant IP cores have been integrated together with IPs from Prosilog and ESA portfolios. One Xilinx FPGA is used for verification in silicon.

On the software side a new development flow is shown and hardware support for downloading software to the LEON3 using the SpaceWire link is demonstrated.

6.2 Magillem 2.3SE

This tool gives the System Engineer the ability to make system trade-offs earlier in the design process. When the entire IP portfolio is integrated into the Magillem environment different SoC can be made very easy. Unfortunately this step in the development process consumed very much time. Manly because the lack of functionality of the tool and also more important, all bugs, both in the IP library and the tool.

Plus	Minus
• Easy to build different SoC when all IPs have been adopted to the tool • Possibility to build higher level of abstraction SoC • Possible to evaluate a SoC design before the bus architecture and CPU are decided • Easy IP interconnection • AMBA-OCP bridge • Multi-Layer AHB controller • OCP IP Creator is a good tool to integrate non-compliant IPs	• Bugs in the generated VHDL code • Bugs in IP library • Generated code still needs modification by hand • Harder to debug due to auto generated signals • Poor monitoring tool • Not mature

Table 1 Magillem 2.3SE Conclusions comparison

Despite interesting features and new concepts, the version 2.3 of Magillem is not at a muturity level which would allow us to use it for an industrial project. The version 3.0 (Q1 2006) shall normally solve most of the yougness problems. Unfortunately the 3.0 version release was delayed and therefore not possible to test within the timeframe of this project.

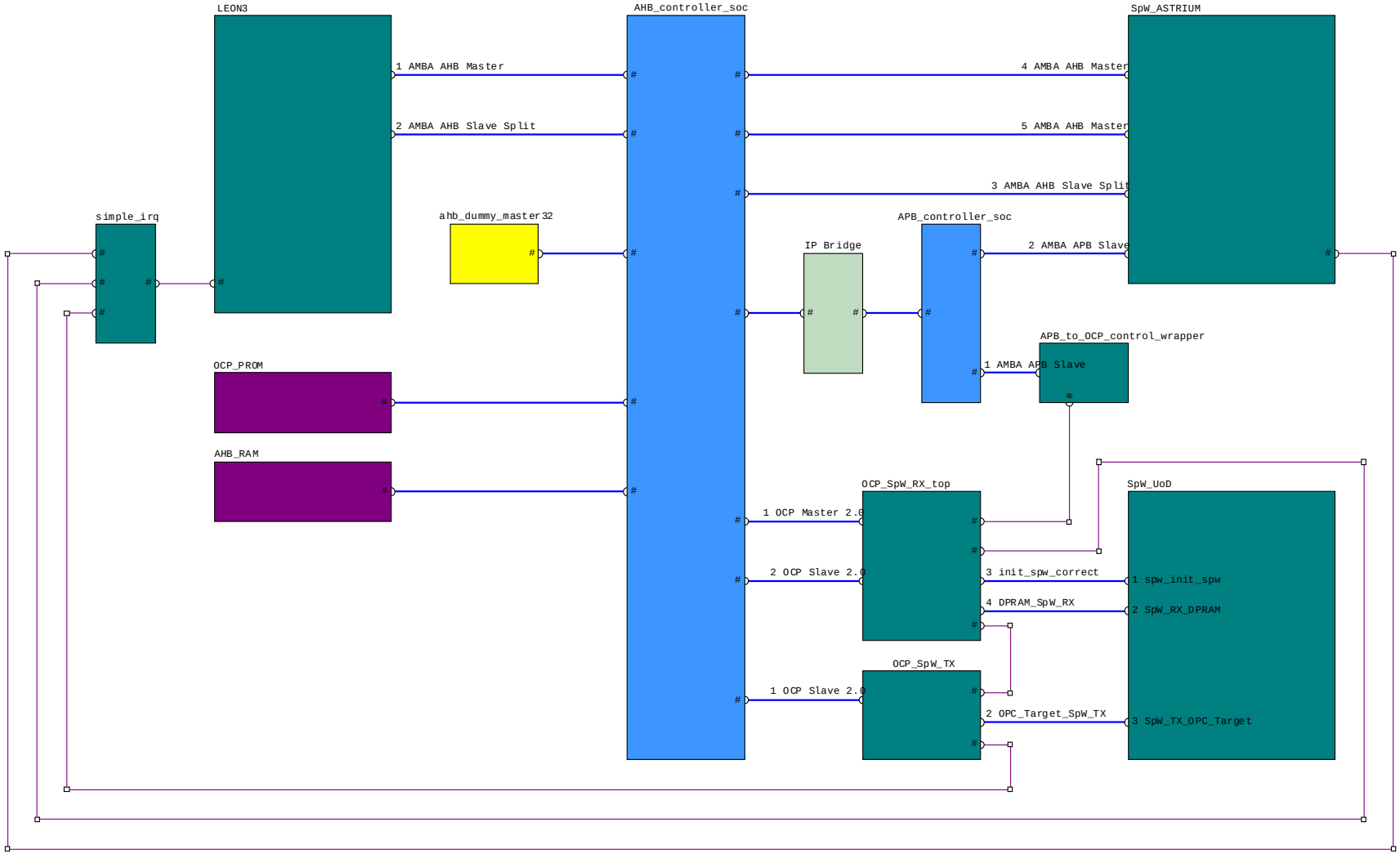
6.3 Other

ESA has a long history using AMBA based microprocessors based on SPARC architecture. This inheritance reduces the interest of OCP and other non hardware specific standards since the target architecture is rather fix from the beginning of the project. Gasiler Research offers a straight forward solution with the GRLIB and LEON3. But this solution is less flexible. A direct comparison between Magillem and GRLIB is not meaningful because these are two completely different tools.
One last remark; the LEON3 is not well suited to be integrated in third party tools. This conclusion has also been made from other projects [13].

7 REFERENCES

[1] Research on new SoC design Methodology using SystemC, Nicolas Laine, Iss 1, Rev 0
[2] ECSS-E-50-12A, 24 January 2003, ESA-ESTEC
[3] AMBA Specification, rev 2.0, ARM IHI 0011A, ARM
[4] Open Core Protocol Specification, Release 2.0, Revision 1.1.1,http://www.ocpip.org/
[5] IP Creator 1.3 (OCP Version), Prosilog
[6] http://www.estec.esa.nl/wsmwww/core/corepage.html
[7] http://www.gaisler.com/
[8] LEON3 Processor User's Manual ver 0.11, October 2004
[9] AT697 LEON microprocessor, http://www.esa.int/techresources/ESTEC-Article-fullArticle_par-28_1120038112963.html
[10] SpaceWire IP Core Specification and Architecture, Ref R&D-SOC-NT-292-V-ASTR, Issue 0, Rev 2, Date 18/06/2003
[11] SpaceWire Codec VHDL User Manual, Ref UoD_Link_User, Rev 1.1, Date 19/04/2004
[12] OCP Initiator API Library, July 2004, ver 2.1c, Prosilog
[13] Development of a Spacecraft Controller on a Chip (SCoC), EADS Vélizy, http://www.estec.esa.nl/wsmwww/core/scoc/scoc_exec_summary_iss_00_01.pdf

APPENDIX A MAGILLEM SCHEMATIC



APPENDIX B LEON3 AND SPW ASTRIUM VHDL WRAPPER FILES

```
----------------------------------------------------------------------
-- Wrapper for LEON3
-- for Prosilog Magillem 2.3SE tool
-- Design by Mattias Carlqvist 20/12/2004 ESA ESTEC
----------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
library grlib;
use grlib.amba.all;
library gaisler;
use gaisler.leon3.all;

entity leon3_top is
  port (
    clk    : in  std_logic;
    resetn   : in  std_logic;

    --ahbi   : in  ahb_mst_in_type;
    ahbi_HGRANT:       in std_logic;                        -- bus grant
    ahbi_HREADY:       in std_logic;                        -- transfer done
    ahbi_HRESP:        in std_logic_vector(1 downto 0); -- response type
    ahbi_HRDATA:       in std_logic_vector(31 downto 0); -- read data bus
    ahbi_HCACHE:       in std_logic;                         -- cacheable data
    ahbi_HIRQ:         in std_logic_vector(15 downto 0); -- interrupt result bus

    --ahbo   : out ahb_mst_out_type;
    ahbo_HBUSREQ:      out std_logic;                         -- bus request
    ahbo_HLOCK:        out std_logic;                         -- lock request
    ahbo_HTRANS:       out std_logic_vector(1 downto 0); -- transfer type
    ahbo_HADDR:        out std_logic_vector(31 downto 0); -- address bus (byte)
    ahbo_HWRITE:       out std_logic;                          -- read/write
    ahbo_HSIZE:        out std_logic_vector(2 downto 0); -- transfer size
    ahbo_HBURST:       out std_logic_vector(2 downto 0); -- burst type
    ahbo_HPROT:        out std_logic_vector(3 downto 0); -- protection control
    ahbo_HWDATA:       out std_logic_vector(31 downto 0); -- write data bus
    ahbo_HIRQ:         out std_logic_vector(15 downto 0);    -- interrupt bus
    ahbo_HCONFIG_0:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_1:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_2:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_3:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_4:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_5:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_6:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HCONFIG_7:    out std_logic_vector(31 downto 0);   -- memory access reg
    ahbo_HINDEX:       out std_logic_vector(2 downto 0);  -- diagnostic use only

    --ahbsi  : in  ahb_slv_in_type;
    ahbsi_HSEL:        in std_logic;                          -- slave select
    ahbsi_HADDR:       in std_logic_vector(31 downto 0); -- address bus (byte)
    ahbsi_HWRITE:      in std_logic;                          -- read/write
    ahbsi_HTRANS:      in std_logic_vector(1 downto 0); -- transfer type
    ahbsi_HSIZE:       in std_logic_vector(2 downto 0); -- transfer size
    ahbsi_HBURST:      in std_logic_vector(2 downto 0); -- burst type
    ahbsi_HWDATA:      in std_logic_vector(31 downto 0); -- write data bus
    ahbsi_HPROT:       in std_logic_vector(3 downto 0); -- protection control
    ahbsi_HREADY:      in std_logic;                          -- transfer done
    ahbsi_HMASTER:     in std_logic_vector(3 downto 0); -- current master
    ahbsi_HMASTLOCK:   in std_logic;                          -- locked access
    ahbsi_HMBSEL:      in std_logic_vector(3 downto 0); -- memory bank select
    ahbsi_HCACHE:      in std_logic;                         -- cacheable
    ahbsi_HIRQ:              in std_logic_vector(15 downto 0); -- interrupt reslt bus
```

```
    --irqi   : in  l3_irq_in_type;
    irqi_irl:          in std_logic_vector(3 downto 0);

    --irqo   : out l3_irq_out_type;
    irqo_intack:               out std_logic;
    irqo_irl:          out std_logic_vector(3 downto 0);

    --dbgi   : in  l3_debug_in_type;
    dbgi_dsuen:        in std_logic;  -- DSU enable
    dbgi_denable:      in std_logic;  -- diagnostic register access enable
    dbgi_dbreak:       in std_logic;  -- debug break-in
    dbgi_step:                 in std_logic;   -- single step
    dbgi_dwrite:       in std_logic;                   -- read/write
    dbgi_daddr:        in std_logic_vector(23 downto 2); -- diagnostic address
    dbgi_ddata:        in std_logic_vector(31 downto 0); -- diagnostic data
    dbgi_btrapa:       in std_logic;     -- break on IU trap
    dbgi_btrape:       in std_logic;  -- break on IU trap
    dbgi_berror:       in std_logic;  -- break on IU error mode
    dbgi_bwatch:       in std_logic;  -- break on IU watchpoint
    dbgi_bsoft:        in std_logic;  -- break on software breakpoint (TA 1)
    dbgi_tenable:      in std_logic;
    dbgi_timer:        in std_logic_vector(30 downto 0);

    --dbgo   : out l3_debug_out_type
    dbgo_data:         out std_logic_vector(31 downto 0);
    dbgo_crdy:         out std_logic;
    dbgo_dsu:                  out std_logic;
    dbgo_dsumode:      out std_logic;
    dbgo_errmode:      out std_logic;
    dbgo_error:                out std_logic
 );
end leon3_top;

architecture rtl of leon3_top is

        component leon3s is
            port (
              clk    : in  std_ulogic;
              rstn   : in  std_ulogic;
              ahbi   : in  ahb_mst_in_type;
              ahbo   : out ahb_mst_out_type;
              ahbsi  : in  ahb_slv_in_type;
              irqi   : in  l3_irq_in_type;
              irqo   : out l3_irq_out_type;
              dbgi   : in  l3_debug_in_type;
              dbgo   : out l3_debug_out_type

            );
        end component;

begin
------------------------------
-- instantiation of leon3s
------------------------------

inst_leon3s : leon3s
  port map(
    clk               =>      std_ulogic(clk),
    rstn              =>      std_ulogic(resetn),

    ahbi.HGRANT(0)            =>      ahbi_HGRANT,
    ahbi.HGRANT(1 to 7) =>    "0000000",
    ahbi.HREADY       =>      std_ulogic(ahbi_HREADY),
    ahbi.HRESP        =>      ahbi_HRESP,
    ahbi.HRDATA       =>      ahbi_HRDATA,
    ahbi.HCACHE       =>      std_ulogic(ahbi_HCACHE),
    ahbi.HIRQ         =>      ahbi_HIRQ,

    std_logic(ahbo.HBUSREQ) =>        ahbo_HBUSREQ,
```

```
    std_logic(ahbo.HLOCK) =>  ahbo_HLOCK,
    ahbo.HTRANS               =>      ahbo_HTRANS,
    ahbo.HADDR         =>     ahbo_HADDR,
    std_logic(ahbo.HWRITE) => ahbo_HWRITE,
    ahbo.HSIZE         =>     ahbo_HSIZE,
    ahbo.HBURST               =>      ahbo_HBURST,
    ahbo.HPROT         =>     ahbo_HPROT,
    ahbo.HWDATA               =>      ahbo_HWDATA,
    ahbo.HIRQ          =>     ahbo_HIRQ,
    ahbo.HCONFIG(0)    =>     ahbo_HCONFIG_0,
    ahbo.HCONFIG(1)    =>     ahbo_HCONFIG_1,
    ahbo.HCONFIG(2)    =>     ahbo_HCONFIG_2,
    ahbo.HCONFIG(3)    =>     ahbo_HCONFIG_3,
    ahbo.HCONFIG(4)    =>     ahbo_HCONFIG_4,
    ahbo.HCONFIG(5)    =>     ahbo_HCONFIG_5,
    ahbo.HCONFIG(6)    =>     ahbo_HCONFIG_6,
    ahbo.HCONFIG(7)    =>     ahbo_HCONFIG_7,
    std_logic_vector(ahbo.HINDEX) => ahbo_HINDEX,

    ahbsi.HSEL(0)      =>     std_ulogic(ahbsi_HSEL),
    ahbsi.HSEL(1 to 7) =>     "0000000",
    ahbsi.HADDR        =>     ahbsi_HADDR,
    ahbsi.HWRITE       =>     std_ulogic(ahbsi_HWRITE),
    ahbsi.HTRANS       =>     ahbsi_HTRANS,
    ahbsi.HSIZE        =>     ahbsi_HSIZE ,
    ahbsi.HBURST       =>     ahbsi_HBURST,
    ahbsi.HWDATA       =>     ahbsi_HWDATA,
    ahbsi.HPROT        =>     ahbsi_HPROT,
    ahbsi.HREADY       =>     std_ulogic(ahbsi_HREADY) ,
    ahbsi.HMASTER      =>     ahbsi_HMASTER,
    ahbsi.HMASTLOCK    =>     std_ulogic(ahbsi_HMASTLOCK),
    ahbsi.HMBSEL       =>     ahbsi_HMBSEL,
    ahbsi.HCACHE       =>     std_ulogic(ahbsi_HCACHE),
    ahbsi.HIRQ         =>     ahbsi_HIRQ,

    irqi.irl           =>     irqi_irl,

    std_ulogic(irqo.intack) =>       irqo_intack,
    irqo.irl           =>     irqo_irl,

    dbgi.dsuen         =>     dbgi_dsuen,
    dbgi.denable       =>     dbgi_denable,
    dbgi.dbreak        =>     dbgi_dbreak,
    dbgi.step          =>     dbgi_step,
    dbgi.dwrite        =>     dbgi_dwrite,
    dbgi.daddr                =>      dbgi_daddr,
    dbgi.ddata                =>      dbgi_ddata,
    dbgi.btrapa        =>     dbgi_btrapa,
    dbgi.btrape        =>     dbgi_btrape,
    dbgi.berror        =>     dbgi_berror,
    dbgi.bwatch        =>     dbgi_bwatch,
    dbgi.bsoft                =>      dbgi_bsoft,
    dbgi.tenable       =>     dbgi_tenable,
    dbgi.timer         =>     dbgi_timer,

    dbgo.data          =>     dbgo_data,
    dbgo.crdy          =>     dbgo_crdy,
    dbgo.dsu           =>     dbgo_dsu,
    dbgo.dsumode       =>     dbgo_dsumode,
    dbgo.errmode       =>     dbgo_errmode,
    dbgo.error         =>     dbgo_error

    );


end rtl;
```

```
------------------------------------------------------------------
-- Wrapper for ESA SpaceWire core version 1.2 developed by ASTRIUM
-- for Prosilog Magillem 2.3SE tool
-- Design by Mattias Carlqvist 30/11/2004 ESA ESTEC
------------------------------------------------------------------
library IEEE ;
use IEEE.Std_Logic_1164.all ;
library amba_lib;
use amba_lib.amba.all;
library spw_v12;
use spw_v12.all;

entity SpW12_top is
port(
  clk_sw         : in Std_Logic;  -- SpaceWire clock
  clk_txin       : in Std_Logic;  -- Tx clock
  clk_txout      : out Std_Logic; -- Tx clock for test
  resetn         : in Std_Logic;  -- asynchronous reset
  tickin_ctm     : in Std_Logic;   -- time code to send
  tickout_ctm    : out Std_Logic;  -- good time code received
  test_mode_hard : in Std_Logic;   -- test mode asserted by hardware

-- link interface --
  d_in           : in Std_Logic;  -- data input
  s_in           : in Std_Logic;  -- strobe input
  d_out          : out Std_Logic; -- data output
  s_out          : out Std_Logic; -- strobe output

-- APB interface --
--  apb_slv_in     : in apb_slv_in_type;
        PSEL:          in    Std_Logic;                        -- slave select
        PENABLE:       in    Std_Logic;                        -- strobe
        PADDR:         in    Std_Logic_Vector(31 downto 0);    -- address bus (byte)
        PWRITE:        in    Std_Logic;                        -- write
        PWDATA:        in    Std_Logic_Vector(31 downto 0);   -- write data bus

--  apb_slv_out    : out apb_slv_out_type;
        PRDATA:        out  Std_Logic_Vector(31 downto 0); -- read data bus

-- AHB interface --
--  tx_ahb_slv_in  : in ahb_slv_in_type;   -- AHB SLAVE
         tx_ahb_slv_in_HSEL:         in Std_Logic;                          -- slave select
         tx_ahb_slv_in_HADDR:        in Std_Logic_Vector(31 downto 0); -- address bus (byte)
         tx_ahb_slv_in_HWRITE:       in Std_Logic;                          -- read/write
         tx_ahb_slv_in_HTRANS:       in Std_Logic_Vector(1 downto 0); -- transfer type
         tx_ahb_slv_in_HSIZE:        in Std_Logic_Vector(2 downto 0); -- transfer size
         tx_ahb_slv_in_HBURST:       in Std_Logic_Vector(2 downto 0); -- burst type
         tx_ahb_slv_in_HWDATA:       in Std_Logic_Vector(31 downto 0); -- write data bus
         tx_ahb_slv_in_HPROT:        in Std_Logic_Vector(3 downto 0); -- protection control
         tx_ahb_slv_in_HREADY:       in Std_Logic;                          -- transfer done
         tx_ahb_slv_in_HMASTER:      in Std_Logic_Vector(3 downto 0); -- current master
         tx_ahb_slv_in_HMASTLOCK:    in Std_Logic;                          -- locked access

--  tx_ahb_slv_out : out ahb_slv_out_type; -- AHB SLAVE
       tx_ahb_slv_out_HREADY:              out Std_Logic;                          -- transfer
done
         tx_ahb_slv_out_HRESP:       out Std_Logic_Vector(1 downto 0); -- response type
         tx_ahb_slv_out_HRDATA:      out Std_Logic_Vector(31 downto 0); -- read data bus
         tx_ahb_slv_out_HSPLIT:      out Std_Logic_Vector(15 downto 0); -- split completion

--  tx_ahb_mst_in  : in ahb_mst_in_type;   -- AHB MASTER
         tx_ahb_mst_in_HGRANT:       in Std_Logic;                          -- bus grant
         tx_ahb_mst_in_HREADY:       in Std_Logic;                          -- transfer done
         tx_ahb_mst_in_HRESP:        in Std_Logic_Vector(1 downto 0); -- response type
         tx_ahb_mst_in_HRDATA:       in Std_Logic_Vector(31 downto 0); -- read data bus
         tx_ahb_mst_in_HCACHE:       in Std_Logic;                          -- cacheable data

--  tx_ahb_mst_out : out ahb_mst_out_type; -- AHB MASTER
         tx_ahb_mst_out_HBUSREQ:     out Std_Logic;                          -- bus request
```

```
          tx_ahb_mst_out_HLOCK:       out Std_Logic;                          -- lock request
          tx_ahb_mst_out_HTRANS:      out Std_Logic_Vector(1 downto 0); -- transfer type
          tx_ahb_mst_out_HADDR:       out Std_Logic_Vector(31 downto 0); -- address bus (byte)
          tx_ahb_mst_out_HWRITE:      out Std_Logic;                           -- read/write
          tx_ahb_mst_out_HSIZE:       out Std_Logic_Vector(2 downto 0); -- transfer size
          tx_ahb_mst_out_HBURST:      out Std_Logic_Vector(2 downto 0); -- burst type
          tx_ahb_mst_out_HPROT:       out Std_Logic_Vector(3 downto 0); -- protection control
          tx_ahb_mst_out_HWDATA:      out Std_Logic_Vector(31 downto 0); -- write data bus

--  rx_ahb_mst_in  : in ahb_mst_in_type;   -- AHB MASTER
          rx_ahb_mst_in_HGRANT:       in Std_Logic;                           -- bus grant
          rx_ahb_mst_in_HREADY:       in Std_Logic;                           -- transfer done
          rx_ahb_mst_in_HRESP:        in Std_Logic_Vector(1 downto 0); -- response type
          rx_ahb_mst_in_HRDATA:       in Std_Logic_Vector(31 downto 0); -- read data bus
          rx_ahb_mst_in_HCACHE:       in Std_Logic;                            -- cacheable data
--  rx_ahb_mst_out : out ahb_mst_out_type; -- AHB MASTER
          rx_ahb_mst_out_HBUSREQ:     out Std_Logic;                           -- bus request
          rx_ahb_mst_out_HLOCK:       out Std_Logic;                           -- lock request
          rx_ahb_mst_out_HTRANS:      out Std_Logic_Vector(1 downto 0); -- transfer type
          rx_ahb_mst_out_HADDR:       out Std_Logic_Vector(31 downto 0); -- address bus (byte)
          rx_ahb_mst_out_HWRITE:      out Std_Logic;                           -- read/write
          rx_ahb_mst_out_HSIZE:       out Std_Logic_Vector(2 downto 0); -- transfer size
          rx_ahb_mst_out_HBURST:      out Std_Logic_Vector(2 downto 0); -- burst type
          rx_ahb_mst_out_HPROT:       out Std_Logic_Vector(3 downto 0); -- protection control
          rx_ahb_mst_out_HWDATA:      out Std_Logic_Vector(31 downto 0); -- write data bus
-- INTERRUPT --
  err_int        : out Std_Logic; -- Error interrupt
  nom_int        : out Std_Logic -- Nominal interrupt
  );
end SpW12_top;

architecture rtl of SpW12_top is

        component spacewire is
                port(
                  clk_sw         : in Std_Logic;
                  clk_txin       : in Std_Logic;
                  clk_txout      : out Std_Logic;
                  resetn         : in Std_Logic;
                  tickin_ctm     : in Std_Logic;
                  tickout_ctm    : out Std_Logic;
                  test_mode_hard : in Std_Logic;

                -- link interface --
                  d_in           : in Std_Logic;
                  s_in           : in Std_Logic;
                  d_out          : out Std_Logic;
                  s_out          : out Std_Logic;

                -- APB interface --
                  apb_slv_in     : in apb_slv_in_type;
                  apb_slv_out    : out apb_slv_out_type;

                -- AHB interface --
                  tx_ahb_slv_in  : in ahb_slv_in_type;
                  tx_ahb_slv_out : out ahb_slv_out_type;
                  tx_ahb_mst_in  : in ahb_mst_in_type;
                  tx_ahb_mst_out : out ahb_mst_out_type;
                  rx_ahb_mst_in  : in ahb_mst_in_type;
                  rx_ahb_mst_out : out ahb_mst_out_type;

                -- INTERRUPT --
                  err_int        : out Std_Logic; -- Error interrupt
                  nom_int        : out Std_Logic -- Nominal interrupt
        );
        end component;


begin
```

```
------------------------------
-- instantiation of spacewire
------------------------------

inst_spacewire : spacewire
port map(
  clk_sw                =>   clk_sw,
  clk_txin              =>   clk_txin,
  clk_txout             =>   clk_txout,
  resetn                =>   resetn,

  tickin_ctm            =>   tickin_ctm,
  tickout_ctm           =>   tickout_ctm,
  test_mode_hard        =>   test_mode_hard,

  d_in                  =>   d_in,
  s_in                  =>   s_in,
  d_out                       =>   d_out,
  s_out                       =>   s_out,

  apb_slv_in.PSEL       =>   Std_ULogic(PSEL),
  apb_slv_in.PENABLE    =>   Std_ULogic(PENABLE),
  apb_slv_in.PADDR      =>   PADDR,
  apb_slv_in.PWRITE     =>   Std_ULogic(PWRITE),
  apb_slv_in.PWDATA     =>   PWDATA,

  apb_slv_out.PRDATA    =>   PRDATA,

  tx_ahb_slv_in.HSEL      =>   Std_ULogic(tx_ahb_slv_in_HSEL),
  tx_ahb_slv_in.HADDR   =>   tx_ahb_slv_in_HADDR,
  tx_ahb_slv_in.HWRITE  =>   Std_ULogic(tx_ahb_slv_in_HWRITE),
  tx_ahb_slv_in.HTRANS  =>   tx_ahb_slv_in_HTRANS,
  tx_ahb_slv_in.HSIZE   =>   tx_ahb_slv_in_HSIZE,
  tx_ahb_slv_in.HBURST  =>   tx_ahb_slv_in_HBURST,
  tx_ahb_slv_in.HWDATA  =>   tx_ahb_slv_in_HWDATA,
  tx_ahb_slv_in.HPROT         =>   tx_ahb_slv_in_HPROT,
  tx_ahb_slv_in.HREADY  =>   Std_ULogic(tx_ahb_slv_in_HREADY),
  tx_ahb_slv_in.HMASTER       =>   tx_ahb_slv_in_HMASTER,
  tx_ahb_slv_in.HMASTLOCK =>   Std_ULogic(tx_ahb_slv_in_HMASTLOCK),

  tx_ahb_slv_out.HREADY   =>   tx_ahb_slv_out_HREADY,
  tx_ahb_slv_out.HRESP    =>   tx_ahb_slv_out_HRESP,
  tx_ahb_slv_out.HRDATA   =>   tx_ahb_slv_out_HRDATA,
  tx_ahb_slv_out.HSPLIT   =>   tx_ahb_slv_out_HSPLIT,

  tx_ahb_mst_in.HGRANT    =>   Std_ULogic(tx_ahb_mst_in_HGRANT),
  tx_ahb_mst_in.HREADY    =>   Std_ULogic(tx_ahb_mst_in_HREADY),
  tx_ahb_mst_in.HRESP     =>   tx_ahb_mst_in_HRESP,
  tx_ahb_mst_in.HRDATA    =>   tx_ahb_mst_in_HRDATA,
  tx_ahb_mst_in.HCACHE    =>   Std_ULogic(tx_ahb_mst_in_HCACHE),

  tx_ahb_mst_out.HBUSREQ  =>   tx_ahb_mst_out_HBUSREQ,
  tx_ahb_mst_out.HLOCK    =>   tx_ahb_mst_out_HLOCK,
  tx_ahb_mst_out.HTRANS   =>   tx_ahb_mst_out_HTRANS,
  tx_ahb_mst_out.HADDR    =>   tx_ahb_mst_out_HADDR ,
  tx_ahb_mst_out.HWRITE   =>   tx_ahb_mst_out_HWRITE,
  tx_ahb_mst_out.HSIZE    =>   tx_ahb_mst_out_HSIZE,
  tx_ahb_mst_out.HBURST   =>   tx_ahb_mst_out_HBURST,
  tx_ahb_mst_out.HPROT    =>   tx_ahb_mst_out_HPROT,
  tx_ahb_mst_out.HWDATA   =>   tx_ahb_mst_out_HWDATA,

  rx_ahb_mst_in.HGRANT    =>   Std_ULogic(rx_ahb_mst_in_HGRANT),
  rx_ahb_mst_in.HREADY    =>   Std_ULogic(rx_ahb_mst_in_HREADY),
  rx_ahb_mst_in.HRESP     =>   rx_ahb_mst_in_HRESP,
  rx_ahb_mst_in.HRDATA    =>   rx_ahb_mst_in_HRDATA,
  rx_ahb_mst_in.HCACHE    =>   Std_ULogic(rx_ahb_mst_in_HCACHE),

  rx_ahb_mst_out.HBUSREQ  =>   rx_ahb_mst_out_HBUSREQ,
```

```
    rx_ahb_mst_out.HLOCK    =>   rx_ahb_mst_out_HLOCK,
    rx_ahb_mst_out.HTRANS   =>   rx_ahb_mst_out_HTRANS,
    rx_ahb_mst_out.HADDR    =>   rx_ahb_mst_out_HADDR ,
    rx_ahb_mst_out.HWRITE   =>   rx_ahb_mst_out_HWRITE,
    rx_ahb_mst_out.HSIZE    =>   rx_ahb_mst_out_HSIZE,
    rx_ahb_mst_out.HBURST   =>   rx_ahb_mst_out_HBURST,
    rx_ahb_mst_out.HPROT    =>   rx_ahb_mst_out_HPROT,
    rx_ahb_mst_out.HWDATA   =>   rx_ahb_mst_out_HWDATA,

    err_int               =>   err_int,
    nom_int               =>   nom_int
    );

end rtl;
```

APPENDIX C OCP BRIDGE VHDL FILES FOR SPACEWIRE DUNDEE IP

OCP_SpW_RX_top.vhd

```
-------------------------------------------------------------------------
---- Generated by PROSILOG IP Creator
---- Date: Thu Jun 30 15:08:06 2005
---- Copyright (c) 2004 Prosilog S.A.
-------------------------------------------------------------------------

-- Modified for 32bit address and wider control/status registers
-- Mattias Carlqvist TEC-EDM 2005-06-29

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

library prosilog;
use prosilog.ipcreator.all;

-------------------------------------------------------------------
-- ENTITY DECLARATION
-------------------------------------------------------------------
entity OCP_SpW_RX_top is
  port (
              Clock          : in    std_logic;

    ---------------------------
    -- OCP Target Ports
    ---------------------------
    MReset_n : in std_logic;
    MCmd        : in std_logic_vector(2 downto 0);
    SCmdAccept  : out std_logic;
--MC
--  MAddress    : in std_logic_vector(5 downto 0);
    MAddress    : in std_logic_vector(31 downto 0);
-- END MC
    MData       : in std_logic_vector(31 downto 0);
    SResp       : out std_logic_vector(1 downto 0);
    SData       : out std_logic_vector(31 downto 0);
    MRespAccept  : in std_logic;
-- MC
--    Status       : out std_logic_vector(32-1 downto 0);
--    Control      : in std_logic_vector(32-1 downto 0);
    Status       : out std_logic_vector(49 downto 0);
    Control      : in std_logic_vector(39 downto 0);
-- END MC
    ControlWr    : in std_logic;
    ---------------------------
    -- DMA Ports (OCP Master)
    ---------------------------
    DMA_MCmd          : out std_logic_vector(2 downto 0);
    DMA_SCmdAccept    : in std_logic;
    DMA_MAddress      : out std_logic_vector(31 downto 0);
    DMA_MReqLast      : out std_logic;
    DMA_MBurstPrecise : out std_logic;
    DMA_MBurstLength  : out std_logic_vector(7 downto 0);
    DMA_MData         : out std_logic_vector(31 downto 0);

    ---------------------------
    -- System Ports (Non-OCP)
    ---------------------------
    Disconnect_It       : out std_logic;
```

```
    Master_error_It : out std_logic;

            SpW_RX_top_RXBUF_READ                   : out   std_logic ;
            SpW_RX_top_RXBUF_EMPTY                  : in    std_logic ;
            SpW_RX_top_RXBUF_PROGFLAG               : in    std_logic ;
            SpW_RX_top_RXBUF_DATA                   : in    std_logic_vector(8 downto 0) ;
            SpW_RX_top_RXBUF_WRVALID                : in    std_logic ;
            SpW_RX_top_RXBUF_WRADDR                 : in    std_logic_vector(4 downto 0) ;
            SpW_RX_top_RXBUF_RDVALID                : in    std_logic ;
            SpW_RX_top_RXBUF_RDADDR                 : in    std_logic_vector(4 downto 0) ;
-- MC       SpW_RX_top_OCP_RX_fifo_empty                    : in    std_logic ;
            SpW_RX_top_CFG_MAXCREDIT                : out   std_logic_vector(5 downto 0) ;
            SpW_RX_top_CFG_SLOWRATE_TXCLK                   : out   std_logic_vector(5 downto 0) ;
            SpW_RX_top_CFG_SLOWRATE_SYSCLK                  : out   std_logic_vector(5 downto 0) ;
            SpW_RX_top_CFG_SLOW_CE                  : out   std_logic ;
            SpW_RX_top_TXRATE               : out   std_logic_vector(5 downto 0) ;
            SpW_RX_top_RXBUF_PROGVAL                : out   std_logic_vector(4 downto 0) ;
            SpW_RX_top_LINK_START                   : out   std_logic ;
            SpW_RX_top_AUTO_START                   : out   std_logic ;
            SpW_RX_top_FLUSH_TX                     : out   std_logic ;
            SpW_RX_top_SEND_EOP                     : out   std_logic_vector(1 downto 0) ;
            SpW_RX_top_STATUS               : in    std_logic_vector(15 downto 0) ;
            SpW_RX_top_DISC_RUN_ERR                 : in    std_logic ;
            SpW_RX_top_PARITY_RUN_ERR               : in    std_logic ;
            SpW_RX_top_ESCAPE_RUN_ERR               : in    std_logic ;
            SpW_RX_top_CREDIT_RUN_ERR               : in    std_logic ;
            SpW_RX_top_SpW_Error_irq                : out   std_logic ;
            SpW_RX_top_EOP_irq              : out   std_logic ;
            SpW_RX_top_LINK_DISABLE                 : out   std_logic
    );

end OCP_SpW_RX_top;

architecture rtl of OCP_SpW_RX_top is

  ---------------------------
  -- User IP component
  ---------------------------
        component SpW_RX_top
        port(
                DOUT_ACCEPT                     : in    std_logic ;
                DOUT                    : out   std_logic_vector(31 downto 0) ;
                DOUT_RDY                        : out   std_logic ;
                CLK                     : in    std_logic ;
                RSTn                    : in    std_logic ;
                RXBUF_READ                      : out   std_logic ;
                RXBUF_EMPTY                     : in    std_logic ;
                RXBUF_PROGFLAG                  : in    std_logic ;
                RXBUF_DATA                      : in    std_logic_vector(8 downto 0) ;
                RXBUF_WRVALID                   : in    std_logic ;
                RXBUF_WRADDR                    : in    std_logic_vector(4 downto 0) ;
                RXBUF_RDVALID                   : in    std_logic ;
                RXBUF_RDADDR                    : in    std_logic_vector(4 downto 0) ;
--MC
--              OCP_status                      : out   std_logic_vector(31 downto 0)
--              OCP_control                     : in    std_logic_vector(31 downto 0) ;
                OCP_status                      : out   std_logic_vector(49 downto 0) ;
                OCP_control                     : in    std_logic_vector(39 downto 0) ;
-- END MC
                OCP_RX_fifo_empty                       : in    std_logic ;
                CFG_MAXCREDIT                   : out   std_logic_vector(5 downto 0) ;
                CFG_SLOWRATE_TXCLK                      : out   std_logic_vector(5 downto 0) ;
                CFG_SLOWRATE_SYSCLK                     : out   std_logic_vector(5 downto 0) ;
                CFG_SLOW_CE                     : out   std_logic ;
                TXRATE                  : out   std_logic_vector(5 downto 0) ;
                RXBUF_PROGVAL                   : out   std_logic_vector(4 downto 0) ;
                LINK_START                      : out   std_logic ;
                AUTO_START                      : out   std_logic ;
                FLUSH_TX                        : out   std_logic ;
```

```
                SEND_EOP                        : out   std_logic_vector(1 downto 0) ;
                STATUS                  : in    std_logic_vector(15 downto 0) ;
                DISC_RUN_ERR                    : in    std_logic ;
                PARITY_RUN_ERR                  : in    std_logic ;
                ESCAPE_RUN_ERR                  : in    std_logic ;
                CREDIT_RUN_ERR                  : in    std_logic ;
                SpW_Error_irq                   : out   std_logic ;
                EOP_irq                 : out   std_logic ;
                LINK_DISABLE                    : out   std_logic
        );
        end component;

-------------------------------------------------------------
-- INTERNAL SIGNAL DECLARATION
-------------------------------------------------------------
        signal  IPsig_DOUT_ACCEPT               : std_logic ;
        signal  IPsig_DOUT              : std_logic_vector(31 downto 0) ;
        signal  IPsig_DOUT_RDY          : std_logic ;
        signal  IPsig_CLK               : std_logic ;
        signal  IPsig_RST               : std_logic ;
        signal  IPsig_RXBUF_READ                : std_logic ;
        signal  IPsig_RXBUF_EMPTY               : std_logic ;
        signal  IPsig_RXBUF_PROGFLAG            : std_logic ;
        signal  IPsig_RXBUF_DATA                : std_logic_vector(8 downto 0) ;
        signal  IPsig_RXBUF_WRVALID             : std_logic ;
        signal  IPsig_RXBUF_WRADDR              : std_logic_vector(4 downto 0) ;
        signal  IPsig_RXBUF_RDVALID             : std_logic ;
        signal  IPsig_RXBUF_RDADDR              : std_logic_vector(4 downto 0) ;
-- MC
--      signal  IPsig_OCP_status                : std_logic_vector(31 downto 0) ;
--      signal  IPsig_OCP_control               : std_logic_vector(31 downto 0) ;
        signal  control_reg_bus_out             : std_logic_vector(39 downto 0);
         signal reg_OCP_status                  : std_logic_vector(49 downto 0) ;
        signal  IPsig_OCP_status                : std_logic_vector(49 downto 0) ;
        signal  IPsig_OCP_control               : std_logic_vector(39 downto 0) ;
-- END MC
        signal  IPsig_OCP_RX_fifo_empty                 : std_logic ;
        signal  IPsig_CFG_MAXCREDIT             : std_logic_vector(5 downto 0) ;
        signal  IPsig_CFG_SLOWRATE_TXCLK                : std_logic_vector(5 downto 0) ;
        signal  IPsig_CFG_SLOWRATE_SYSCLK               : std_logic_vector(5 downto 0) ;
        signal  IPsig_CFG_SLOW_CE               : std_logic ;
        signal  IPsig_TXRATE            : std_logic_vector(5 downto 0) ;
        signal  IPsig_RXBUF_PROGVAL             : std_logic_vector(4 downto 0) ;
        signal  IPsig_LINK_START                : std_logic ;
        signal  IPsig_AUTO_START                : std_logic ;
        signal  IPsig_FLUSH_TX          : std_logic ;
        signal  IPsig_SEND_EOP          : std_logic_vector(1 downto 0) ;
        signal  IPsig_STATUS            : std_logic_vector(15 downto 0) ;
        signal  IPsig_DISC_RUN_ERR              : std_logic ;
        signal  IPsig_PARITY_RUN_ERR            : std_logic ;
        signal  IPsig_ESCAPE_RUN_ERR            : std_logic ;
        signal  IPsig_CREDIT_RUN_ERR            : std_logic ;
        signal  IPsig_SpW_Error_irq             : std_logic ;
        signal  IPsig_EOP_irq           : std_logic ;
        signal  IPsig_LINK_DISABLE              : std_logic ;
-- MC   signal control_reg_bus_out    : std_logic_vector(32-1 downto 0);
        signal control_write         :std_logic;
-- MC   signal  reg_OCP_status : std_logic_vector(31 downto 0) ;
        signal  S_DOUT_RDY              : std_logic;
        signal  S_DOUT          : std_logic_vector(32-1 downto 0);
        signal  S_DOUT_ACCEPT  : std_logic;
        signal  Master_FIFO_out_Dout  : std_logic_vector( 32-1 downto 0);
    signal MAddress_completed     : std_logic_vector( 5 downto 0);
    signal S_Mod_Addr             : std_logic_vector( 3 downto 0 );
    signal S_Mod_Data_Out         : std_logic_vector( 32-1 downto 0    );
    signal S_Mod_Data_In          : std_logic_vector( 32-1 downto 0    );
    signal S_Address_DataOut      : std_logic_vector( 32-1 downto 0 );
    signal S_DataOut              : std_logic_vector( 32-1 downto 0    );
    signal S_OUTPUT_packet_length : std_logic_vector( 8-1 downto 0  );
```

```
    signal S_Address_DataIn       : std_logic_vector( 32-1 downto 0 );
    signal S_DataIn               : std_logic_vector( 32-1 downto 0    );
    signal S_INPUT_packet_length  : std_logic_vector( 8-1 downto 0  );
-- MC    signal status_reg_bus_out     : std_logic_vector( 32-1 downto 0    );
    signal  S_Request_DataOut    : std_logic;
    signal  S_OUTPUT_Request_Ack : std_logic;
    signal  S_OUTPUT_EOP         : std_logic;
    signal  S_Request_DataIn     : std_logic;
    signal  S_INPUT_Request_Ack  : std_logic;
    signal  S_Response_Rdy       : std_logic;
    signal  S_INPUT_EOP          : std_logic;
    signal  S_OCP_clock          : std_logic;
    signal  S_IP_clock           : std_logic;
    signal  S_INPUT_fifo_full    : std_logic;
    signal  S_Read_req           : std_logic;
    signal  S_Write_req          : std_logic;
    signal  S_Mod_Rdy            : std_logic;
-----------------------------------------------------------------
-- ARCHITECTURE begins...
-----------------------------------------------------------------
begin
--MC
--    MAddress_completed <= MAddress;
    MAddress_completed <= MAddress(5 downto 0);
-- END MC
    ---------------------------------------
    ------      OCP TARGET Module    -------
    ---------------------------------------
    OCP_Target : ocp_dataflow_target_1
    generic map (
      ocp_data_width => 32
    )
              port map (
            clk             => S_OCP_clock       ,
            mreset_n        => MReset_n          ,
            mcmd            => MCmd              ,
            scmdaccept      => SCmdAccept        ,
            -- Address LSB bits are unused since addressed
            -- registers are 32-bit wide
            maddr           => MAddress_completed(5 downto 2)        ,
            mdata           => MData             ,
            sresp           => SResp             ,
            sdata           => SData             ,
            mrespaccept     => MRespAccept       ,
            mod_addr        => S_Mod_Addr        ,
            mod_data_out    => S_Mod_Data_Out    ,
            mod_data_in     => S_Mod_Data_In     ,
            read_req        => S_Read_req        ,
            write_req       => S_Write_req       ,
            mod_rdy         => S_Mod_Rdy
            );

    ---------------------------------------
    ------     OCP MASTER Module    -------
    ---------------------------------------
    OCP_Master : ocp_dataflow_master
    generic map (
      master_address_width => 32,
      master_data_width => 32,
      mburstlength_wdth => 8,
      master_be_width => 4
    )
              port map (
        clk                  => S_OCP_clock           ,
        sreset_n             => MReset_n              ,
        scmdaccept           => DMA_SCmdAccept        ,
        maddr                => DMA_MAddress          ,
        mcmd                 => DMA_MCmd              ,
        mreqlast             => DMA_MReqLast          ,
```

```
        mburstlength         => DMA_MBurstLength       ,
        mburstprecise        => DMA_MBurstPrecise      ,
        mdata                => DMA_MData              ,
        sdata                => (others => '0')        ,
        sresp                => "00"                   ,
        sresplast            => '0'                    ,
        mrespaccept          => open,
        merror_it            => Master_error_It        ,
        address_dataout      => S_Address_DataOut      ,
        dataout              => S_DataOut              ,
        request_dataout      => S_Request_DataOut      ,
        output_request_ack   => S_OUTPUT_Request_Ack   ,
        output_eop           => S_OUTPUT_EOP           ,
        output_packet_length => S_OUTPUT_packet_length ,
        address_datain       => S_Address_DataIn       ,
        datain               => S_DataIn               ,
        request_datain       => S_Request_DataIn       ,
        input_request_ack    => S_INPUT_Request_Ack    ,
        response_rdy         => S_Response_Rdy         ,
        input_eop            => S_INPUT_EOP            ,
        input_packet_length  => S_INPUT_packet_length  ,
        input_fifo_full      => S_INPUT_fifo_full
        );

    ---------------------------------------
    ------  MASTER FIFO OUT Module  -------
    ---------------------------------------
--MC
--  Master_FIFO_Out : master_output_fifo
    Master_FIFO_Out : master_output_fifo_modified
-- END MC
    generic map (
      TARGET_ADDRESS_WIDTH  =>  4,
      TARGET_DATA_WIDTH     =>  32,
      MASTER_ADDRESS_WIDTH  =>  32,
      MASTER_DATA_WIDTH     =>  32,
      TRANSFER_LGTH_WIDTH   =>  8,
      INCREMENT_WIDTH       =>  8,
      PACKET_LGTH_WIDTH     =>  8,
      FIFO_LENGTH           =>  32,
      STATUS_WIDTH          =>  6
    )
    port map (
            clock           => S_OCP_clock              ,
            resetn          => MReset_n                 ,
            disconnect_it   => Disconnect_It            ,
            threshold_it    => open,
            sel             => '1'                      ,
            target_address  => S_Mod_Addr(3 downto 0)   ,
            target_read     => S_Read_req               ,
            target_write    => S_Write_req              ,
            target_data_in  => S_Mod_Data_In            ,
            target_data_out => Master_FIFO_out_Dout     ,
            address_dataout => S_Address_DataOut        ,
            dataout         => S_DataOut                ,
            request_dataout => S_Request_DataOut        ,
            request_ack     => S_OUTPUT_Request_Ack     ,
            end_of_packet   => S_OUTPUT_EOP             ,
            packet_length   => S_OUTPUT_packet_length   ,
            module_rdy      => S_Mod_Rdy                ,
            module_error    => open,
            module_done     => open,
            dout_rdy        => S_DOUT_RDY               ,
            dout            => S_DOUT                   ,
            dout_accept     => S_DOUT_ACCEPT         ,
            FIFO_out      => IPsig_OCP_RX_fifo_empty       -- added
            );

    -- This signal has to be tied to zero to ensure a correct behaviour of the
```

```
    --     'ocp_dataflow_master' module
    S_INPUT_fifo_full  <= '0';


                ---------------------------------------------
                ------   SpW_RX_top instance           -------
                ---------------------------------------------
SpW_RX_top_inst: SpW_RX_top
        port map(
        DOUT_ACCEPT             =>      IPsig_DOUT_ACCEPT,
        DOUT            =>      IPsig_DOUT,
        DOUT_RDY                =>      IPsig_DOUT_RDY,
        CLK             =>      IPsig_CLK,
        RSTn            =>      IPsig_RST,
        RXBUF_READ              =>      IPsig_RXBUF_READ,
        RXBUF_EMPTY             =>      IPsig_RXBUF_EMPTY,
        RXBUF_PROGFLAG          =>      IPsig_RXBUF_PROGFLAG,
        RXBUF_DATA              =>      IPsig_RXBUF_DATA,
        RXBUF_WRVALID           =>      IPsig_RXBUF_WRVALID,
        RXBUF_WRADDR            =>      IPsig_RXBUF_WRADDR,
        RXBUF_RDVALID           =>      IPsig_RXBUF_RDVALID,
        RXBUF_RDADDR            =>      IPsig_RXBUF_RDADDR,
        OCP_status              =>      IPsig_OCP_status,
        OCP_control             =>      IPsig_OCP_control,
        OCP_RX_fifo_empty               =>      IPsig_OCP_RX_fifo_empty,
        CFG_MAXCREDIT           =>      IPsig_CFG_MAXCREDIT,
        CFG_SLOWRATE_TXCLK              =>      IPsig_CFG_SLOWRATE_TXCLK,
        CFG_SLOWRATE_SYSCLK             =>      IPsig_CFG_SLOWRATE_SYSCLK,
        CFG_SLOW_CE             =>      IPsig_CFG_SLOW_CE,
        TXRATE          =>      IPsig_TXRATE,
        RXBUF_PROGVAL           =>      IPsig_RXBUF_PROGVAL,
        LINK_START              =>      IPsig_LINK_START,
        AUTO_START              =>      IPsig_AUTO_START,
        FLUSH_TX                =>      IPsig_FLUSH_TX,
        SEND_EOP                =>      IPsig_SEND_EOP,
        STATUS          =>      IPsig_STATUS,
        DISC_RUN_ERR            =>      IPsig_DISC_RUN_ERR,
        PARITY_RUN_ERR          =>      IPsig_PARITY_RUN_ERR,
        ESCAPE_RUN_ERR          =>      IPsig_ESCAPE_RUN_ERR,
        CREDIT_RUN_ERR          =>      IPsig_CREDIT_RUN_ERR,
        SpW_Error_irq           =>      IPsig_SpW_Error_irq,
        EOP_irq         =>      IPsig_EOP_irq,
        LINK_DISABLE            =>      IPsig_LINK_DISABLE
        );
                ---------------------------------------------
                ------  Status Register                      -------
                ---------------------------------------------
        status_register: process (S_IP_clock)
        begin
        if ( S_IP_clock='1' and S_IP_clock'event )
        then
                if ( MReset_n='0' )
                then
                reg_OCP_status          <=      (others => '0');
                else
                reg_OCP_status          <=      IPsig_OCP_status;
                end if;
        end if;
        end process;
        Status <= reg_OCP_status;
                ---------------------------------------------
                ------  Control Register              -------
                ---------------------------------------------
        control_write <= ControlWr;
        control_register: process (S_OCP_clock)
        begin
        if ( S_OCP_clock='1' and S_OCP_clock'event )
        then
                if ( MReset_n='0' )
```

```
                then
                control_reg_bus_out    <=     (others => '0');
                elsif(control_write='1')
                then
                control_reg_bus_out    <=     Control ;
                else
                control_reg_bus_out    <=     control_reg_bus_out ;
                end if;
        end if;
        end process;
--MC
--      IPsig_OCP_control      <=     control_reg_bus_out( 31 downto 0);
        IPsig_OCP_control      <=     control_reg_bus_out( 39 downto 0);
 -- END MC
    ------------------------------------------------
    --------   Misc internal assignments     -------
    ------------------------------------------------

    -- Connect 'S_Mod_Data_Out   ' to Master_Out module
     S_Mod_Data_Out      <= Master_FIFO_out_Dout;

    -- DMA input request is not used here
    -- These signals must be tied to zero to ensure a correct behaviour of the
    -- 'ocp_dataflow_master' module
        S_Address_DataIn       <= (others => '0');
        S_Request_DataIn       <= '0';
        S_INPUT_EOP            <= '0';
        S_INPUT_packet_length  <= (others => '0');

       ------------------------------------------------
       --------        Clock Assignment               -------
       ------------------------------------------------
        S_IP_clock      <= Clock;
        S_OCP_clock     <= Clock;

       ------------------------------------------------
       --------        Semantic signals assignment  -------
       ------------------------------------------------
       -- reset signal
        IPsig_RST       <= MReset_n;
       -- dout_rdy signal
        S_DOUT_RDY      <= IPsig_DOUT_RDY;
       -- dout_accept signal
        IPsig_DOUT_ACCEPT      <= S_DOUT_ACCEPT;
       -- dout signal
        S_DOUT          <= IPsig_DOUT;
       -- clock signal
        IPsig_CLK       <= S_IP_clock;

       ------------------------------------------------
       --------        SpW_RX_top extern signals    -------
       ------------------------------------------------
     SpW_RX_top_RXBUF_READ             <=     IPsig_RXBUF_READ;
        IPsig_RXBUF_EMPTY       <= SpW_RX_top_RXBUF_EMPTY;
        IPsig_RXBUF_PROGFLAG    <= SpW_RX_top_RXBUF_PROGFLAG;
        IPsig_RXBUF_DATA        <= SpW_RX_top_RXBUF_DATA;
        IPsig_RXBUF_WRVALID     <= SpW_RX_top_RXBUF_WRVALID;
        IPsig_RXBUF_WRADDR      <= SpW_RX_top_RXBUF_WRADDR;
        IPsig_RXBUF_RDVALID     <= SpW_RX_top_RXBUF_RDVALID;
        IPsig_RXBUF_RDADDR      <= SpW_RX_top_RXBUF_RDADDR;
--MC     IPsig_OCP_RX_fifo_empty        <= SpW_RX_top_OCP_RX_fifo_empty;
     SpW_RX_top_CFG_MAXCREDIT          <=     IPsig_CFG_MAXCREDIT;
     SpW_RX_top_CFG_SLOWRATE_TXCLK             <=     IPsig_CFG_SLOWRATE_TXCLK;
     SpW_RX_top_CFG_SLOWRATE_SYSCLK            <=     IPsig_CFG_SLOWRATE_SYSCLK;
     SpW_RX_top_CFG_SLOW_CE            <=     IPsig_CFG_SLOW_CE;
     SpW_RX_top_TXRATE         <=     IPsig_TXRATE;
     SpW_RX_top_RXBUF_PROGVAL          <=     IPsig_RXBUF_PROGVAL;
     SpW_RX_top_LINK_START             <=     IPsig_LINK_START;
     SpW_RX_top_AUTO_START             <=     IPsig_AUTO_START;
```

```
     SpW_RX_top_FLUSH_TX              <=     IPsig_FLUSH_TX;
     SpW_RX_top_SEND_EOP              <=     IPsig_SEND_EOP;
         IPsig_STATUS    <= SpW_RX_top_STATUS;
         IPsig_DISC_RUN_ERR     <= SpW_RX_top_DISC_RUN_ERR;
         IPsig_PARITY_RUN_ERR   <= SpW_RX_top_PARITY_RUN_ERR;
         IPsig_ESCAPE_RUN_ERR   <= SpW_RX_top_ESCAPE_RUN_ERR;
         IPsig_CREDIT_RUN_ERR   <= SpW_RX_top_CREDIT_RUN_ERR;
     SpW_RX_top_SpW_Error_irq         <=     IPsig_SpW_Error_irq;
     SpW_RX_top_EOP_irq               <=     IPsig_EOP_irq;
     SpW_RX_top_LINK_DISABLE          <=     IPsig_LINK_DISABLE;

END rtl;
```

SpW_RX_top.vhd

```
--------------------------------------------------------------------------------
-- SpaceWire RX top

-- Including
-- Memory block for RX buffer
-- Statemachine to packet the bytes into word and handle EOP/EEP

-- Mattias Carlqvist TEC-EDM, ESTEC, 27-06-2005

--------------------------------------------------------------------------------
-- libraries used
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

--------------------------------------------------------------------------------
-- entity declaration
--------------------------------------------------------------------------------

entity SpW_RX_top is
        port (
                CLK             : in  STD_LOGIC;        -- sync clock
                RSTn            : in  STD_LOGIC;        -- async acive low reset
                -- signals SpaceWire RX
                RXBUF_READ      : out STD_LOGIC;       -- read signal to SpW
                RXBUF_EMPTY           : in  STD_LOGIC;        -- fifo empty flag
                RXBUF_PROGFLAG : in  STD_LOGIC;       -- not used
                RXBUF_DATA      : in    STD_LOGIC_VECTOR(8 downto 0);
                RXBUF_WRVALID   : in    STD_LOGIC;      -- connected to WE
                RXBUF_WRADDR    : in    STD_LOGIC_VECTOR(4 downto 0);
                RXBUF_RDVALID   : in    STD_LOGIC;      -- connected to EN
                RXBUF_RDADDR    : in    STD_LOGIC_VECTOR(4 downto 0);
                -- control signals SpaceWire
                CFG_MAXCREDIT   : out STD_LOGIC_VECTOR(5 downto 0);
                CFG_SLOWRATE_TXCLK  : out STD_LOGIC_VECTOR(5 downto 0);
                CFG_SLOWRATE_SYSCLK : out STD_LOGIC_VECTOR(5 downto 0);
                CFG_SLOW_CE     : out STD_LOGIC;
                TXRATE          : out STD_LOGIC_VECTOR(5 downto 0);
                RXBUF_PROGVAL          : out  STD_LOGIC_VECTOR(4 downto 0);
                LINK_START             : out  STD_LOGIC;  -- start link
                LINK_DISABLE           : out  STD_LOGIC;  -- disable link
                AUTO_START             : out  STD_LOGIC;  -- set autostart bit
                FLUSH_TX               : out  STD_LOGIC;  -- flush the transmitter independant off
state
                SEND_EOP        : out  STD_LOGIC_VECTOR(1 downto 0); -- feed to SpW_TX
                -- status signals SpaceWire
                STATUS          : in STD_LOGIC_VECTOR(15 downto 0);  -- output status
                DISC_RUN_ERR    : in STD_LOGIC;  -- run errors (in run state)
                PARITY_RUN_ERR : in STD_LOGIC;
                ESCAPE_RUN_ERR : in STD_LOGIC;
                CREDIT_RUN_ERR : in STD_LOGIC;
                -- signals OCP_MASTER
                DOUT_ACCEPT     : in  STD_LOGIC;        -- external read
                DOUT_RDY        : out STD_LOGIC;        -- data is valid active high
```

```
                DOUT           : out   STD_LOGIC_VECTOR(31 downto 0);
                OCP_status     : out STD_LOGIC_VECTOR(49 downto 0);
                OCP_control    : in STD_LOGIC_VECTOR(39 downto 0);
                OCP_RX_fifo_empty: in STD_LOGIC;
                -- interupt signal
                SpW_Error_irq  : out STD_LOGIC;       -- Error interupt to CPU
                EOP_irq        : out STD_LOGIC               -- EOP interupt to CPU
                );

end entity SpW_RX_top;

architecture structural of SpW_RX_top is

        component memblock
                generic(
                        addr_len       :       INTEGER;
                        mem_width      :       INTEGER
                        );
                port(
                        WRCLK   :       in      STD_LOGIC;
                        RDCLK   :       in      STD_LOGIC;
                        RST     :       in      STD_LOGIC;
                        WE      :       in      STD_LOGIC;
                        EN      :       in      STD_LOGIC;
                        RDADDR  :       in      STD_LOGIC_VECTOR(addr_len-1 downto 0);
                        WRADDR  :       in      STD_LOGIC_VECTOR(addr_len-1 downto 0);
                        DI      :       in      STD_LOGIC_VECTOR(mem_width-1 downto 0);
                        DO      :       out     STD_LOGIC_VECTOR(mem_width-1 downto 0)
                        );
        end component;

        component SpW_RX
                port(
                        CLK             : in  STD_LOGIC;        -- sync clock
                        RSTn            : in  STD_LOGIC;        -- async reset
                        RXBUF_READ      : out STD_LOGIC;       -- read signal to SpW
                        RXBUF_EMPTY           : in  STD_LOGIC;        -- spw buffer empty flag
                        RXBUF_RDVALID   : in  STD_LOGIC;       -- read ack from spw buffer
                        DI              : in  STD_LOGIC_VECTOR(8 downto 0);
                        DOUT_ACCEPT     : in  STD_LOGIC;        -- read ack signal from OCP master
                        DOUT_RDY        : out STD_LOGIC;        -- data is valid active high
                        DOUT            : out STD_LOGIC_VECTOR(31 downto 0);
                        OCP_fifo_empty  : in  STD_LOGIC;
                        start           : in  STD_LOGIC;
                        reset_interupt  : in  STD_LOGIC;
                        EOP_interupt    : out STD_LOGIC;
                        total_bytes_spw       : out STD_LOGIC_VECTOR(15 downto 0);
                        status_spw      : out STD_LOGIC_VECTOR(1 downto 0)
                        );
        end component;


signal mem_data: STD_LOGIC_VECTOR(8 downto 0);
signal start_rx: STD_LOGIC;
signal reset_interupt: STD_LOGIC;
signal total_bytes_spw: STD_LOGIC_VECTOR(15 downto 0);
signal status_eop: STD_LOGIC_VECTOR(1 downto 0);

signal OCP_control_dummy: STD_LOGIC_VECTOR(1 downto 0);


begin

        memblock_inst: memblock
        generic map(
                addr_len       =>      5,
                mem_width      =>      9
                )
        port map(
```

```
                WRCLK   =>      CLK,
                RDCLK   =>      CLK,
                RST     =>      RSTn,
                WE      =>      RXBUF_WRVALID,
                EN      =>      RXBUF_RDVALID,
                RDADDR  =>      RXBUF_RDADDR,
                WRADDR  =>      RXBUF_WRADDR,
                DI      =>      RXBUF_DATA,
                DO      =>      mem_data
                );

        SpW_RX_inst: SpW_RX
        port map(
                CLK             => CLK,
                RSTn            => RSTn,
                RXBUF_READ      => RXBUF_READ,
                RXBUF_EMPTY     => RXBUF_EMPTY,
                RXBUF_RDVALID   => RXBUF_RDVALID,
                DI              => mem_data,
                DOUT_ACCEPT     => DOUT_ACCEPT,
                DOUT_RDY        => DOUT_RDY,
                DOUT            => DOUT,
                OCP_fifo_empty  => OCP_RX_fifo_empty,
                start           => start_rx,
                reset_interupt => reset_interupt,
                EOP_interupt    => EOP_irq,
                total_bytes_spw => total_bytes_spw,
                status_spw      => status_eop
                );


-- map of  OCP control register
CFG_MAXCREDIT          <= OCP_control(5 downto 0);
CFG_SLOWRATE_TXCLK     <= OCP_control(11 downto 6);
CFG_SLOWRATE_SYSCLK    <= OCP_control(17 downto 12);
CFG_SLOW_CE            <= OCP_control(18);
TXRATE                         <= OCP_control(24 downto 19);
RXBUF_PROGVAL                  <= OCP_control(29 downto 25);
OCP_control_dummy      <= OCP_control(31 downto 30);

LINK_START                     <= OCP_control(32);
LINK_DISABLE                   <= OCP_control(33);
AUTO_START                     <= OCP_control(34);
FLUSH_TX                       <= OCP_control(35);
START_RX               <= OCP_control(36);
RESET_INTERUPT         <= OCP_control(37);
SEND_EOP               <= OCP_control(39 downto 38);

-- map of OCP status register
OCP_status(15 downto 0) <= STATUS;
OCP_status(16)          <= DISC_RUN_ERR;
OCP_status(17)          <= PARITY_RUN_ERR;
OCP_status(18)          <= ESCAPE_RUN_ERR;
OCP_status(19)          <= CREDIT_RUN_ERR;
OCP_status(31 downto 20)<= (others => '0');
OCP_status(47 downto 32)<= total_bytes_spw;
OCP_status(49 downto 48)<= status_eop;

SpW_Error_irq <= DISC_RUN_ERR or PARITY_RUN_ERR or ESCAPE_RUN_ERR or CREDIT_RUN_ERR;

end structural;
```

SpW_RX.vhd

```
--------------------------------------------------------------------------------
-- SpaceWire RX bridge to OCP_MASTER_FIFO_OUT
-- Mattias Carlqvist TEC-EDM, ESTEC, 27-06-2005
--
-- Updated to package the data and take care of the EOP signal, 32bit data widht
-- 20-06-2005
--------------------------------------------------------------------------------
-- libraries used
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

--------------------------------------------------------------------------------
-- entity declaration
--------------------------------------------------------------------------------
entity Spw_RX is
       port (
              CLK            : in  STD_LOGIC;        -- sync clock
              RSTn           : in  STD_LOGIC;        -- async active low reset
              -- signals SpaceWire RX
              RXBUF_READ     : out STD_LOGIC;       -- read signal to SpW
              RXBUF_EMPTY           : in  STD_LOGIC;        -- spw buffer empty flag
              RXBUF_RDVALID  : in  STD_LOGIC;       -- read ack from spw buffer
              DI             : in  STD_LOGIC_VECTOR(8 downto 0);
              -- signals OCP_MASTER
              DOUT_ACCEPT    : in  STD_LOGIC;        -- read ack signal from OCP master
              DOUT_RDY       : out STD_LOGIC;        -- data is valid active high
              DOUT           : out STD_LOGIC_VECTOR(31 downto 0);
              OCP_fifo_empty  : in  STD_LOGIC;
              start          : in  STD_LOGIC;
              reset_interupt : in  STD_LOGIC;
              EOP_interupt   : out STD_LOGIC;
              total_bytes_spw       : out STD_LOGIC_VECTOR(15 downto 0);
              status_spw     : out STD_LOGIC_VECTOR(1 downto 0)

              );

end entity Spw_RX;

architecture rtl of Spw_RX is

signal start_i:std_logic;
signal status: std_logic_vector(1 downto 0); -- status register to be read by CPU
--signal interupt: std_logic := '0';                -- interupt EOP/EEP signal to cpu
--signal OCP_fifo_empty: std_logic;
--signal start: std_logic;
signal total_bytes: unsigned(15 downto 0);          -- total number of bytes received
signal cnt_bytes: unsigned(1 downto 0);             -- number of bytes packet into the 32 bit word
signal complete_word: std_logic_vector(31 downto 0);        -- the word to be sent to OCP master
fifo

type state is (idle, store , checkEOP, doutValid, doutValidEOP, irq);
signal current_state, next_state: state;

begin


state_reg: process(clk, rstn)
begin
   if rstn = '0' then
      current_state <= idle;
   elsif clk'event and clk = '1' then
      current_state <= next_state;
   end if;
end process state_reg;
```

```
comb_state: process (current_state, rxbuf_empty, rxbuf_rdvalid, dout_accept, di, cnt_bytes,
OCP_fifo_empty, reset_interupt)

begin

               case current_state is

                       when idle =>                  -- wait for received byte
                       if (rxbuf_empty = '0') then
                              next_state <= store;
                       else
                              next_state <= idle;
                       end if;

                       when store =>                 -- packet 4 bytes into one word
                       if rxbuf_rdvalid = '1' then

                              next_state <= checkEOP;
                       else
                              next_state <= store;
                        end if;

                       when checkEOP =>
                       if di(8) = '1' then
                              next_state <= doutValidEOP;
                       elsif cnt_bytes = 3 then
                              next_state <= doutValid;
                       else
                              next_state <= store;
                       end if;

                       when doutValid =>
                       if dout_accept = '1' then
                          next_state <= idle;
                       else
                          next_state <= doutValid;
                       end if;

                       when doutValidEOP =>
                       if OCP_fifo_empty = '1' then
                          next_state <= irq;
                       else
                          next_state <= doutValidEOP;
                       end if;


                       when irq =>                   -- wait for start command set by host
                       if reset_interupt = '1' then
                              next_state <= idle;
                       else
                              next_state <= irq;
                       end if;

               end case;
end process comb_state;

status_r: process(clk,rstn)
begin
        if rstn = '0' then
           status <= "00";
--         interupt <= '0';
        elsif clk'event and clk = '1' then
           if (current_state = checkEOP and next_state = doutValidEOP) then
              if di(0) = '0' then
                 status <= "01";      -- EOP
              else
                 status <= "10";      -- EEP
              end if;
           end if;
```

```
       end if;
 end process status_r;

word: process(clk,rstn)
begin
       if rstn = '0' then
          complete_word <= (others => '0');
       elsif clk'event and clk = '1' then
          if current_state = checkEOP then
                      case cnt_bytes is
                      when "00" =>
                         complete_word(31 downto 24) <= di(7 downto 0);
                         complete_word(23 downto 0) <= (others => '0');
                      when "01" =>
                         complete_word(23 downto 16) <= di(7 downto 0);
                      when "10" =>
                         complete_word(15 downto 8)  <= di(7 downto 0);
                      when "11" =>
                         complete_word(7 downto 0)   <= di(7 downto 0);
                      when others =>
                         null;
                      end case;
          end if;
   end if;
 end process word;

cnt_bytes_r: process(clk,rstn,current_state)
begin
       if clk'event and clk = '1' then
          if (rstn = '0' or current_state = idle) then
             cnt_bytes <= "00";
          elsif current_state = checkEOP and next_state = store then
             cnt_bytes <= cnt_bytes + 1;
          end if;
       end if;
end process cnt_bytes_r;

cnt_total_r: process(clk,rstn)       -- reset the total bytes after CPU irq ack
begin
       if clk'event and clk = '1' then
          if rstn = '0' or (current_state = irq and next_state = idle) then
             total_bytes <= (others => '0');
          elsif current_state = checkEOP and next_state /= doutValidEOP then
             total_bytes <= total_bytes + 1;
          end if;
       end if;
end process cnt_total_r;

       rxbuf_read <= '1' when (current_state = store) else '0';
       dout_rdy <= '1' when (current_state = doutValid) else '0';
       dout <= complete_word;
       total_bytes_spw <= std_logic_vector(total_bytes);
       status_spw <= status;
       eop_interupt <= '1' when current_state = irq else '0';

start_i <= start;
end rtl;
```

OCP_SpW_TX.vhd

```
------------------------------------------------------------------------
---- Generated by PROSILOG IP Creator
---- Date: Tue Jun 28 16:35:03 2005
---- Copyright (c) 2004 Prosilog S.A.
------------------------------------------------------------------------
-- Modified for 32bit address
-- Mattias Carlqvist TEC-EDM 2005-06-29

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
library prosilog;
use prosilog.ipcreator.all;
---------------------------------------------------------------
-- ENTITY DECLARATION
---------------------------------------------------------------
entity OCP_SpW_TX is
  port (
    SpW_TX_Clock  : in std_logic;
    OCP_Clock             : in std_logic;

    ---------------------------
    -- OCP Target Ports
    ---------------------------
    MReset_n : in std_logic;
    MCmd        : in std_logic_vector(2 downto 0);
    SCmdAccept  : out std_logic;
-- MC
--    MAddress    : in std_logic_vector(3 downto 0);
    MAddress    : in std_logic_vector(31 downto 0);
-- END MC
    MData       : in std_logic_vector(31 downto 0);
    SResp       : out std_logic_vector(1 downto 0);
    SData       : out std_logic_vector(31 downto 0);
    MRespAccept  : in std_logic;
    ---------------------------
    -- System Ports (Non-OCP)
    ---------------------------
    Threshold_It              : out std_logic;
           SpW_TX_FIFO_DATA                 : out   std_logic_vector(8 downto 0) ;
           SpW_TX_FIFO_READ                 : in    std_logic ;
           SpW_TX_FIFO_EMPTY                : out   std_logic ;
           SpW_TX_CONTROL                   : in    std_logic_vector(1 downto 0) ; -- send EOP req
           SpW_TX_EOP_sent_interupt                 : out   std_logic
    );

end OCP_SpW_TX;

architecture rtl of OCP_SpW_TX is
  ---------------------------
  -- User IP component
  ---------------------------
       component SpW_TX
       port(
              DIN_ACCEPT                        : out   std_logic ;
              DIN                       : in    std_logic_vector(31 downto 0) ;
              DIN_RDY                   : in    std_logic ;
              CLK                       : in    std_logic ;
              RSTn                      : in    std_logic ;
              FIFO_DATA                         : out   std_logic_vector(8 downto 0) ;
              FIFO_READ                         : in    std_logic ;
              FIFO_EMPTY                        : out   std_logic ;
              CONTROL                   : in    std_logic_vector(1 downto 0) ;
              EOP_sent_interupt                         : out   std_logic
       );
       end component;
```

```
------------------------------------------------------------
-- INTERNAL SIGNAL DECLARATION
------------------------------------------------------------
       signal  IPsig_DIN_ACCEPT              : std_logic ;
       signal  IPsig_DIN             : std_logic_vector(31 downto 0) ;
       signal  IPsig_DIN_RDY         : std_logic ;
       signal  IPsig_CLK             : std_logic ;
       signal  IPsig_RST             : std_logic ;
       signal  IPsig_FIFO_DATA               : std_logic_vector(8 downto 0) ;
       signal  IPsig_FIFO_READ               : std_logic ;
       signal  IPsig_FIFO_EMPTY              : std_logic ;
       signal  IPsig_CONTROL         : std_logic_vector(1 downto 0) ;
       signal  IPsig_EOP_sent_interupt               : std_logic ;
       signal  S_DIN_RDY             : std_logic;
       signal  S_DIN          : std_logic_vector(32-1 downto 0);
       signal  S_DIN_ACCEPT   : std_logic;
       signal   Target_FIFO_In_Dout   : std_logic_vector( 32-1 downto 0);
       signal   S_Target_FIFO_In_Full : std_logic;
-- MC
--    signal MAddress_completed     : std_logic_vector( 5 downto 0);
    signal MAddress_completed     : std_logic_vector( 31 downto 0);
-- END MC
    signal S_Mod_Addr             : std_logic_vector( 3 downto 0 );
    signal S_Mod_Data_Out         : std_logic_vector( 32-1 downto 0    );
    signal S_Mod_Data_In          : std_logic_vector( 32-1 downto 0    );
    signal S_Address_DataOut      : std_logic_vector( 1-1 downto 0 );
    signal S_DataOut              : std_logic_vector( 1-1 downto 0    );
    signal S_OUTPUT_packet_length : std_logic_vector( 1-1 downto 0  );
    signal S_Address_DataIn       : std_logic_vector( 1-1 downto 0 );
    signal S_DataIn               : std_logic_vector( 1-1 downto 0    );
    signal S_INPUT_packet_length  : std_logic_vector( 1-1 downto 0  );
    signal status_reg_bus_out     : std_logic_vector( 32-1 downto 0    );
    signal  S_Request_DataOut    : std_logic;
    signal  S_OUTPUT_Request_Ack : std_logic;
    signal  S_OUTPUT_EOP         : std_logic;
    signal  S_Request_DataIn     : std_logic;
    signal  S_INPUT_Request_Ack  : std_logic;
    signal  S_Response_Rdy       : std_logic;
    signal  S_INPUT_EOP          : std_logic;
    signal  S_OCP_clock          : std_logic;
    signal  S_IP_clock           : std_logic;
    signal  S_INPUT_fifo_full    : std_logic;
    signal  S_Read_req           : std_logic;
    signal  S_Write_req          : std_logic;
    signal  S_Mod_Rdy            : std_logic;
------------------------------------------------------------
-- ARCHITECTURE begins...
------------------------------------------------------------
begin

-- MC
--    MAddress_completed <= "00" & MAddress;
      MAddress_completed <=  MAddress;
-- END MC
    ---------------------------------------
    ------      OCP TARGET Module    -------
    ---------------------------------------
    OCP_Target : ocp_dataflow_target_2
    generic map (
      ocp_data_width => 32
    )
               port map (
            clk            => S_OCP_clock       ,
            mreset_n       => MReset_n          ,
            mcmd           => MCmd              ,
            scmdaccept     => SCmdAccept        ,
            -- Address LSB bits are unused since addressed
            -- registers are 32-bit wide
            maddr          => MAddress_completed(5 downto 2)        ,
```

```
            mdata          => MData            ,
            sresp          => SResp            ,
            sdata          => SData            ,
            mrespaccept    => MRespAccept      ,
            mod_addr       => S_Mod_Addr       ,
            mod_data_out   => S_Mod_Data_Out   ,
            mod_data_in    => S_Mod_Data_In    ,
            read_req       => S_Read_req       ,
            write_req      => S_Write_req      ,
            mod_rdy        => S_Mod_Rdy
            );
    Target_FIFO_In : target_input_fifo_async
    generic map (
      FIFO_WIDTH            =>  32,
      FIFO_LENGTH           =>  32,
      THRESHOLD_HIGH_LEVEL  =>  1,
      THRESHOLD_LOW_LEVEL   =>  1,
      STATUS_WIDTH          =>  6
    )
    port map (
            ip_clock   =>S_IP_clock,
            vci_clock  =>S_OCP_clock,
            resetn           =>MReset_n,
            threshold_low    => open,
            threshold_high   => open,
            threshold_it => Threshold_It            ,
            sel          => '1'                     ,
            address      => S_Mod_Addr(1 downto 0)  ,
            vci_read     => S_Read_req              ,
            vci_write    => S_Write_req             ,
            vci_data_in  => S_Mod_Data_In           ,
            vci_data_out => Target_FIFO_In_Dout     ,
            full         => S_Target_FIFO_In_Full   ,
            module_rdy   => S_Mod_Rdy               ,
            module_error => open,
            module_done  => open,
            din_rdy      => S_DIN_RDY               ,
            din          => S_DIN                   ,
            din_accept   => S_DIN_ACCEPT
            );
    S_DataOut <= (others => '0');

               ------------------------------------------------
               ------   SpW_TX instance               -------
               ------------------------------------------------
SpW_TX_inst: SpW_TX
        port map(
        DIN_ACCEPT             =>     IPsig_DIN_ACCEPT,
        DIN            =>     IPsig_DIN,
        DIN_RDY        =>     IPsig_DIN_RDY,
        CLK            =>     IPsig_CLK,
        RSTn           =>     IPsig_RST,
        FIFO_DATA              =>     IPsig_FIFO_DATA,
        FIFO_READ              =>     IPsig_FIFO_READ,
        FIFO_EMPTY             =>     IPsig_FIFO_EMPTY,
        CONTROL        =>     IPsig_CONTROL,
        EOP_sent_interupt              =>     IPsig_EOP_sent_interupt
        );
    ------------------------------------------------
    --------   Misc internal assignments     -------
    ------------------------------------------------
    -- Connect 'S_Mod_Data_Out   ' to Target_In module
     S_Mod_Data_Out     <= Target_FIFO_In_Dout;

        ------------------------------------------------
        --------        Clock Assignment               -------
        ------------------------------------------------
         S_IP_clock      <= SpW_TX_Clock;
         S_OCP_clock     <= OCP_Clock;
```

```
        ------------------------------------------------
        --------        Semantic signals assignment  -------
        ------------------------------------------------
        -- reset signal
         IPsig_RST       <= MReset_n;
        -- din_rdy signal
         IPsig_DIN_RDY   <= S_DIN_RDY;
        -- din_accept signal
         S_DIN_ACCEPT    <= IPsig_DIN_ACCEPT;
        -- din signal
         IPsig_DIN              <= S_DIN;
        -- clock signal
         IPsig_CLK       <= S_IP_clock;

        ------------------------------------------------
        --------        SpW_TX extern signals    -------
        ------------------------------------------------
     SpW_TX_FIFO_DATA          <=    IPsig_FIFO_DATA;
         IPsig_FIFO_READ        <= SpW_TX_FIFO_READ;
     SpW_TX_FIFO_EMPTY         <=    IPsig_FIFO_EMPTY;
         IPsig_CONTROL   <= SpW_TX_CONTROL;
     SpW_TX_EOP_sent_interupt          <=    IPsig_EOP_sent_interupt;

END rtl;
```

SpW_TX.vhd

```
--------------------------------------------------------------------------------
-- SpaceWire TX bridge to OCP_TARET_FIFO_IN

-- Mattias Carlqvist TEC-EDM, ESTEC, 27-06-2005
--------------------------------------------------------------------------------
-- libraries used
--------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

--------------------------------------------------------------------------------
-- entity declaration
--------------------------------------------------------------------------------
entity Spw_TX is
        port (
                CLK             : in  STD_LOGIC;        -- sync clock
                RSTn            : in  STD_LOGIC;        -- async reset
                --signals SpaceWire TX
                FIFO_DATA       : out  STD_LOGIC_VECTOR(8 downto 0); -- transmit data
                FIFO_READ       : in STD_LOGIC;    -- read from tx
                FIFO_EMPTY      : out  STD_LOGIC;    -- tx fifo empty flag
                -- signals OCP_TARGET
                CONTROL         : in STD_LOGIC_VECTOR(1 DOWNTO 0);
                DIN_ACCEPT      : out STD_LOGIC ;
                DIN             : in    STD_LOGIC_VECTOR(31 downto 0) ;
                DIN_RDY         : in    STD_LOGIC;
                EOP_sent_interupt: out STD_LOGIC

                );

end entity Spw_TX;

architecture rtl of Spw_TX is
signal complete_word: std_logic_vector(31 downto 0);        -- the word read from OCP_TARGET_FIFO
signal cnt_bytes: unsigned(1 downto 0);             -- number of bytes store in the SpW FIFO

--resync of control signals
signal CONTROL_i: STD_LOGIC_VECTOR(1 DOWNTO 0);
signal CONTROL_ii: STD_LOGIC_VECTOR(1 DOWNTO 0);
signal FIFO_EMPTY_i: STD_LOGIC;
```

```
type state is (idle, load_data ,send_data, send_EOP, sent_EOP);
signal current_state, next_state: state;

begin

state_reg: process(clk, rstn)
begin
   if rstn = '0' then
      current_state <= idle;
   elsif clk'event and clk = '1' then
      current_state <= next_state;
   end if;
end process state_reg;

comb_state: process (current_state, DIN_RDY, CONTROL_ii, cnt_bytes, FIFO_READ)

begin

               case current_state is

                      when idle =>                  -- check Control signals and wait DIN_Ready
                      if (CONTROL_ii = "01" or CONTROL_ii = "10") and DIN_RDY = '0' then
                             next_state <= send_EOP;
                      elsif DIN_RDY = '1' then
                             next_state <= load_data;
                      else
                             next_state <= idle;
                      end if;


                      when load_data =>                     -- load word
                             next_state <= send_data;

                      when send_data =>                     -- send bytes to SpW
                      if cnt_bytes = 3 and FIFO_READ = '1' then
                             next_state <= idle;
                      else
                             next_state <= send_data;
                      end if;

                      when send_EOP =>
                      if FIFO_READ = '1' then
                             next_state <= sent_EOP;
                      else
                             next_state <= send_EOP;
                      end if;

                      when sent_EOP =>
                      if CONTROL_ii /= "00" then
                             next_state <= sent_EOP;
                      else
                             next_state <= idle;
                      end if;

               end case;
end process comb_state;


tx_data: process(clk, rstn, complete_word, DIN, current_state, next_state)
begin
       if rstn = '0' then
          complete_word <= (others => '0');
          FIFO_DATA <= "000000000";
       elsif clk'event and clk = '1' then
          if (current_state = idle and next_state = send_EOP) then
             if CONTROL_ii = "01" then
                FIFO_DATA <= "100000000";    -- Send EOP to SpW
             else
                FIFO_DATA <= "100000001";    -- Send EEP to SpW
```

```
            end if;
         end if;
         if (current_state = load_data) then
            complete_word <= DIN;
         else
            complete_word <= complete_word;
         end if;

         if (current_state = send_data) then
                  case cnt_bytes is
                  when "00" =>
                     FIFO_DATA(7 downto 0) <= complete_word(31 downto 24);
                     FIFO_DATA(8) <= '0';
                  when "01" =>
                     FIFO_DATA(7 downto 0) <= complete_word(23 downto 16);
                     FIFO_DATA(8) <= '0';
                  when "10" =>
                     FIFO_DATA(7 downto 0) <= complete_word(15 downto 8);
                     FIFO_DATA(8) <= '0';
                  when "11" =>
                     FIFO_DATA(7 downto 0) <= complete_word(7 downto 0);
                     FIFO_DATA(8) <= '0';
                  when others =>
                     null;
                  end case;
         end if;
   end if;
end process tx_data;

cnt: process(clk,rstn)
begin
        if rstn = '0' then
           cnt_bytes <= "00";
        elsif clk'event and clk = '1' then

           if current_state = idle then
              cnt_bytes <= "00";
           elsif current_state = send_data and FIFO_READ = '1' then
              cnt_bytes <= cnt_bytes + 1;
           end if;
        end if;
end process cnt;

resync:process(clk,rstn,CONTROL)
begin
        if rstn = '0' then
           CONTROL_i <= "00";
           CONTROL_ii <= "00";
        elsif clk'event and clk = '1' then
           CONTROL_i  <= CONTROL;
           CONTROL_ii <= CONTROL_i;
        end if;
end process resync;

delay_fifo:process(clk,rstn,FIFO_EMPTY_i)
begin
        if rstn = '0' then
           FIFO_EMPTY <= '0';
        elsif clk'event and clk = '1' then
           FIFO_EMPTY <= FIFO_EMPTY_i;
        end if;
end process delay_fifo;

FIFO_EMPTY_i   <= '0' when ((current_state = send_data and next_state = send_data) or current_state
= send_eop) else '1';
DIN_ACCEPT     <= '1' when current_state = load_data else '0';
EOP_sent_interupt <= '1' when current_state = sent_EOP else '0';
end rtl;
```

APPENDIX D APB TO OCP STATUS/CONTROL BRIDGE

```
----------------------------------------------------------------------
----- APB to OCP control and status register wrapper
----- Mattias Carqlvist TEC-EDM ESTEC 2005-06-15
----- Support 2 register for control and data
----------------------------------------------------------------------
LIBRARY ieee ;
use ieee.std_logic_1164.all ;
use ieee.std_logic_arith.all ;
use ieee.std_logic_unsigned.all ;

----------------------------------------------------------------------
----- Entity declaration
----------------------------------------------------------------------
entity APB_to_OCP_control_wrapper is
       port (
              pclk : in std_logic;
              presetn : in std_logic;
              pselx : in std_logic;
              penable : in std_logic;
              pwrite : in std_logic;
              paddr : in std_logic_vector(31 downto 0);
              pwdata : in std_logic_vector(31 downto 0);
              prdata : out std_logic_vector(31 downto 0);
              OCP_status : in std_logic_vector(63 downto 0);
              OCP_control : out std_logic_vector(63 downto 0);
              OCP_controlWr: out std_logic
       );
end APB_to_OCP_control_wrapper;

architecture rtl of APB_to_OCP_control_wrapper is

begin

comb: process(pselx, penable, pwrite, paddr)
   begin
       if ((pselx and penable and pwrite) = '1') then
              if paddr(2) = '0' then
                 OCP_control(31 downto 0) <= pwdata;
              else
                 OCP_control(63 downto 32) <= pwdata;
              end if;
          OCP_controlWr <= '1';
       else
          OCP_controlWr <= '0';
       end if;
       if ((pselx and penable) = '1' and pwrite = '0') then
              if paddr(2) = '0' then
                  prdata <= OCP_status(31 downto 0);
              else
                  prdata <= OCP_status(63 downto 32);
              end if;


       end if;
end process comb;
end rtl;
```

APPENDIX E LEON3 TEST PROGRAM

```
!This file test the SoC_two_SpaceWire design
!2005-06-09 Mattias Carlqvist


#define TRAP(H)  mov %psr, %l0; sethi %hi(H), %l4; jmp %l4+%lo(H); nop;
#define BAD_TRAP ta 0; nop; nop; nop;
#define SOFT_TRAP BAD_TRAP
#define PSR_INIT   0x10e0
#define MCFG1 0x233
#define MCFG2 0xe6B86e60
#define MCFG3 0x000ff000
#define ASDCFG 0xfff00100
#define DSDCFG 0xe6B86e60
#define L2MCTRLIO 0x80000000
#define RAMSTART  0x40000000
#define RAMSIZE   0x00003FFF  !16kbyte on chip ram

!SpaceWire astrium setup register values
#define SPW1START  0x40000000
#define SPW1MIDDLE 0x40001F00
#define SPW1END    0x40001FFF

!SpaceWire UoD OCP register values
#define SPW2START      0x40002000
#define SPW2END        0x40003FFF
#define TRANSFERLENGTH 0x00000020
#define MODE           0x00000002
#define INCREMENT      0x00000004
#define BURSTLENGTH    0x00000008
#define THRESHOLD      0x00000008


    .seg       "text"
    .proc      0
    .align     4
    .global    start, _trap_table, _trap_handler_spw1nom, _trap_handler_spw2nom,


start:
_trap_table:
  TRAP(_hardreset);           ! 00 reset trap
  BAD_TRAP;                   ! 01 instruction_access_exception
  BAD_TRAP;                   ! 02 illegal_instruction
  BAD_TRAP;                   ! 03 priveleged_instruction
  BAD_TRAP;                   ! 04 fp_disabled
  BAD_TRAP;                   ! 05 window_overflow
  BAD_TRAP;                   ! 06 window_underflow
  BAD_TRAP;                   ! 07 memory_address_not_aligned
  BAD_TRAP;                   ! 08 fp_exception
  BAD_TRAP;                   ! 09 data_access_exception
  BAD_TRAP;                   ! 0A tag_overflow
  BAD_TRAP;                   ! 0B undefined
  BAD_TRAP;                   ! 0C undefined
  BAD_TRAP;                   ! 0D undefined
  BAD_TRAP;                   ! 0E undefined
  BAD_TRAP;                   ! 0F undefined
  BAD_TRAP;                   ! 10 undefined
  TRAP(_trap_handler_spw1nom);  ! 11 interrupt level 1
  TRAP(_trap_handler_spw2nom);  ! 12 interrupt level 2
  BAD_TRAP;                   ! 13 interrupt level 3
  BAD_TRAP;                   ! 14 interrupt level 4
  BAD_TRAP;                   ! 15 interrupt level 5
```

```
  BAD_TRAP;                   ! 16 interrupt level 6
  BAD_TRAP;                   ! 17 interrupt level 7
  BAD_TRAP;                   ! 18 interrupt level 8
  BAD_TRAP;                   ! 19 interrupt level 9
  BAD_TRAP;                   ! 1A interrupt level 1
  BAD_TRAP;                   ! 1B interrupt level 11
  BAD_TRAP;                   ! 1C interrupt level 12
  BAD_TRAP;                   ! 1D interrupt level 13
  BAD_TRAP;                   ! 1E interrupt level 14
  BAD_TRAP;                   ! 1F interrupt level 15

_hardreset:

        flush
        set PSR_INIT, %g1
        mov %g1, %psr
        mov %g0, %wim
        mov %g0, %tbr
        mov %g0, %y
        nop
        set  0x1000f, %g1
        sta %g1, [%g0] 2    !Cache Control Reg: enable data/inst caches and inst fetch burst

2:
        mov %asr17, %g3
        and %g3, 0x1f, %g3
        mov %g0, %g4
        mov %g0, %g5
        mov %g0, %g6
        mov %g0, %g7
1:
        mov %g0, %l0
        mov %g0, %l1
        mov %g0, %l2
        mov %g0, %l3
        mov %g0, %l4
        mov %g0, %l5
        mov %g0, %l6
        mov %g0, %l7
        mov %g0, %o0
        mov %g0, %o1
        mov %g0, %o2
        mov %g0, %o3
        mov %g0, %o4
        mov %g0, %o5
        mov %g0, %o6
        mov %g0, %o7
        subcc %g3, 1, %g3
        bge 1b
        save

        mov 2, %g1
        mov %g1, %wim
        set 0x10e0, %g1              ! enable traps
        mov %g1, %psr
        nop; nop; nop;


        !send the setup data to SPW_1
        set SPW1START, %l0
        set 0x00007000, %l1
        st %l0, [%l1+0xc] !register offset
        set SPW1MIDDLE, %l0
        st %l0, [%l1+0x14]
        set SPW1END, %l0
        st %l0, [%l1+0x4]
        set 0x07000705, %l0 ! using 80MHz transmit clk
        st %l0, [%l1]
```

```
        !setup the UoD_OCP_Master_RX
        set SPW2START, %l0
        set 0x0010000, %l1
        st %l0, [%l1]
        set SPW2END, %l0
        st %l0, [%l1+0x4]
        set TRANSFERLENGTH, %l0
        st %l0, [%l1+0x8]
        set MODE, %l0
        st %l0, [%l1+0xC]
        set INCREMENT, %l0
        st %l0, [%l1+0x10]
        set BURSTLENGTH, %l0
        st %l0, [%l1+0x14]
        set THRESHOLD, %l0
        st %l0, [%l1+0x24]

        set 0x1, %l0          !Empty the FIFO
        st %l0, [%l1+0x20]
        set 0x0, %l0
        st %l0, [%l1+0x20]

        set 0x0, %l0          !start the Master
        st %l0, [%l1+0x18]


        ! wait for spw init aprox 10us
        set 0x00000055, %g3
_wait:
        nop
        subcc %g3, 1, %g3
        bge _wait
        nop


        !start the loop data transfer
        set 0x4000, %l0
        set 0x20, %l1
        st %l1,  [%l0]
        set 0x11111111, %l1
        st %l1,  [%l0]
        set 0x22222222, %l1
        st %l1,  [%l0]
        set 0x33333333, %l1
        st %l1,  [%l0]
        set 0x44444444, %l1
        st %l1,  [%l0]
        set 0x55555555, %l1
        st %l1,  [%l0]
        set 0x66666666, %l1
        st %l1,  [%l0]
        set 0x77777777, %l1
        st %l1,  [%l0]
        set 0x88888888, %l1
        st %l1,  [%l0]



        set _loop, %g2
_loop:
        nop
        nop
        jmp %g2
        nop

_trap_handler_spw1nom:
```

```
        !add %g1, 1, %g1
        set 0x00007030, %l3
        set 0x40000000, %l4
        set 0x0000FFFF, %l5
        set 0x00004000, %l7
        sth %g0, [%l3]          !reset the interupt
        ld [%l4], %g3
        and %g3, %l5, %g3
        st %g3, [%l7]
        udiv %g3, 4, %g3
        sub %g3, 1, %g3

_send_back1:
        add %l4, 0x4, %l4
        ld [%l4], %l6
        !add %l6, %g1, %l6
        st %l6, [%l7]
        subcc %g3, 1, %g3
        bge _send_back1
        nop

        jmpl %l1, %g0           !return to _loop
        rett %l2
        nop



_trap_handler_spw2nom:
        add %g4, 0x20, %g4      !increment the SPW2START register
        add SPW2START, %g4, %l0
        st %l0, [%l1+0xc]

_send_back2:
        add %l4, 0x4, %l4
        ld [%l4], %l6
        !add %l6, %g1, %l6
        st %l6, [%l7]
        subcc %g3, 1, %g3
        bge _send_back2
        nop

        jmpl %l1, %g0           !return to _loop
        rett %l2



.align  32
```