A SOC DESIGN METHODOLOGY FOR LEON2 ON FPGA

E. Ostúa, J. Juan Chico, J. Viejo, M. J. Bellido, D. Guerrero, A. Millán & P. Ruiz-de-Clavijo

Instituto de Microelectrónica de Sevilla – Centro Nacional de Microelectrónica
Edificio CICA - Av. Reina Mercedes, s/n – 41012 Sevilla (España)
Tel.: +34 955056666 – Fax: +34 955056686 - http://www.imse.cnm.es

Departamento de Tecnología Electrónica – Universidad de Sevilla
E.T.S. Ingeniería Informática - Av. Reina Mercedes, s/n – 41012 Sevilla (España)
Tel.: +34 954556161 - Fax: +34 954552764 - http://www.dte.us.es

{ ostua ; jjchico ; julian ; bellido ; guerre; amillan ; paulino }@dte.us.es

1 ABSTRACT

SoC design methodologies show up as a natural and productive method to implement embedded and/or ubiquitous systems. The authors explore the possibilities of the free LEON2 processor core, originally developed by the European Space Agency, and the Xilinx FPGA family to develop a complete SoC design methodology for both hardware and software. Advantages of the platform and productivity of the proposed methodology are highlighted through an application example, showing the suitability of LEON2 implemented on FPGA for professional-grade applications.

1. INTRODUCTION

The continuous development of the integrated circuit technology is leading to an increasing integration density besides an improvement of the operation speed of electronic systems. In digital systems, it allows for huge performance improvements while the system size shrinks more and more. However, design methodologies are also increasing in complexity, and new challenges are to be faced by designers. Two fundamental problems appear: on one hand, the increment in speed (switching activity) and device density translates in greater power consumption and power and thermal density that may induce system failure unless effective cooling mechanisms are included; on the other hand, as integration density increases, the total number of devices included in the chip increases (up to several million devices these days). Then, it becomes necessary to define good design and verification methodologies in all the levels of abstraction in order to handle effectively the complexity of the whole problem.

One of the more extended methodologies in this sense is the so-called System-on-Chip methodology or, more simply, SoC design. The basic idea under this methodology is to use the current integration possibilities not to design more complex specific systems, like Intel or AMD processors, but to implement whole systems made out of several individual building blocks on a single chip. Figure 1 illustrate the SoC idea. This way, systems currently implemented out of several chips on a PCB (Printed Circuit Board) are being integrated in a single chip while maintaining the overall structure of the system. This methodology provides many advantages without the need to completely re-define the way electronic designers work. Among these advantages are power consumption reduction, increased performance and much more miniaturization, making the SoC design a strong methodology for embedded and ubiquitous applications.

Almost 100% of the SoC's are digital systems built around a microprocessor that acts as a controlling unit. The selected microprocessor will strongly determine the performance and overall characteristics of the system, and the cost of the SoC. It is important to note that most embedded applications do not require high computation capabilities, but usually demand high reliability and low power consumption.

Additionally, if several thousands of million of units are to be produced, the cost becomes a major issue.

There are several processor cores that are commonly used in SoC applications, both commercial cores that requires the acquisition of a license of use, like ARM [FURB00], PowerPC [POWE04], NIOS3 [NIOS04], MicroBlaze [MICR05], and many others; and free or open cores that may be used without the need to acquire a licence, like LEON2 [GAIS04] and OpenRisc [LAMP01]. While privative processors are usually well tested and optimized, the openness of free cores provides valuable advantages. For example, MicroBlaze from Xilinx is well integrated with the development platform from the same foundry, which leads to highly optimized designs at the cost of being bound to a particular technology and set of tools. On the other hand, LEON2 is a powerful processor that can be implemented on Xilinx hardware besides a variety of other FPGA hardware and any standard ASIC process. Porting to a new technology is feasible an depends only on the implementer. A variety of software development tools are available under both free and privative licenses.

This paper is the consequence of the early developments of the author's research team in the SoC design area. At this time, we have set up a SoC development methodology based on the above-mentioned LEON2 processor and the FPGA family from Xilinx, which is already being applied to an industrial project. LEON2 has been chosen because it is one of the more developed free processor cores available and it may be implemented on a variety of hardware solutions like stated before. Besides, Xilinx FPGA's are chosen for its wide implantation both on the industry and in the academia.

A LEON2 design methodology start with the processors source code which is mostly written in VHDL core and is fully synthesizable. The following steps in the design process may follow a variety of alternatives. The main objective of this paper is to share our experience setting up a LEON2 based design methodology, where processor possibilities besides software and hardware design alternatives are explored in order to produce a full SoC. The task of porting the LEON2 interface to the Virtex-II Evaluation Kit from Avnet, previously not supported by LEON2 is of an special interest.

The rest of the paper is organized as follows: in the next section we provide a brief introduction to the LEON2 processor. Section 3 describes the

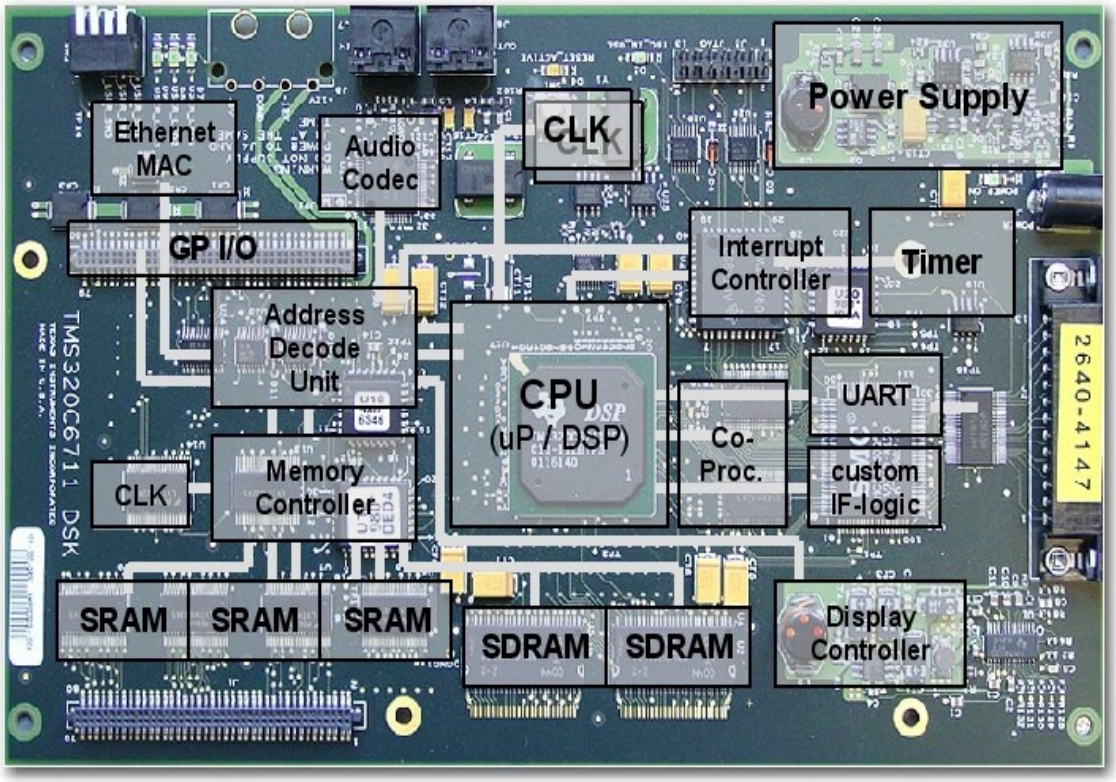

Figure 1: System-on-Chip concept

implemented design methodology including the software and hardware design process and application integration. Section 4 describes an application example. Finally, some conclusions are derived in section 5.

2. LEON2 MICROPROCESSOR

LEON2 is a microprocessor which implements a RISC architecture conforming to the SPARC v8 definition [SPAR92]. It's a synthesizable core written in VHDL and can be implemented both on FPGAs and ASICs. It's distributed under the terms of the GNU LGPL license so it is an open hardware [SEAM02] and it is specifically designed for embedded applications. It was originally developed by the European Space Agency and nowadays it is maintained by Gaisler Research.

The LEON2 32-bit core implements the full SPARC v8 standard, it uses big-endian byte ordering, has 32-bit internal registers, 72 different instructions in 3 different instruction formats and 3 addressing modes (immediate, displacement and indexed). It implements signed and unsigned multiply, divide and MAC operations and has a 5-stage instruction pipeline (Instruction Fetch, Decode, Execute, Memory & Write). It also implements two separate instruction and data cache interfaces, Harvard Architecture [HENN93].

The VHDL model is fully synthesizable with most of the commonly synthesis tools, it is very configurable and it uses the AMBA-2.0 AHB/APB on-chip buses [AMBA99], which makes it easy to extend its functionality. All these features makes LEON2 an ideal microprocessor for System-on-Chip applications.

A block diagram of LEON2 architecture can be seen in figure 2. Many of those blocks are optional and can be removed from the model our concrete application implements.

LEON2 implements the following features:

- *32 bits RISC microprocessor*
- *SPARC v8 compliant*
- *5-stage instruction pipeline*
- *multiply/divide/mac operations on hardware*
- *separated instruction and data caches*
- *memory management unit, MMU*
- *memory interfaces for FLASH, SRAM, SDRAM & PROM*
- *on-chip RAM*
- *interrupt handler*
- *interface for a floating point unit, FPU*
- *debug support unit, DSU*
- *two 24-bit timers*

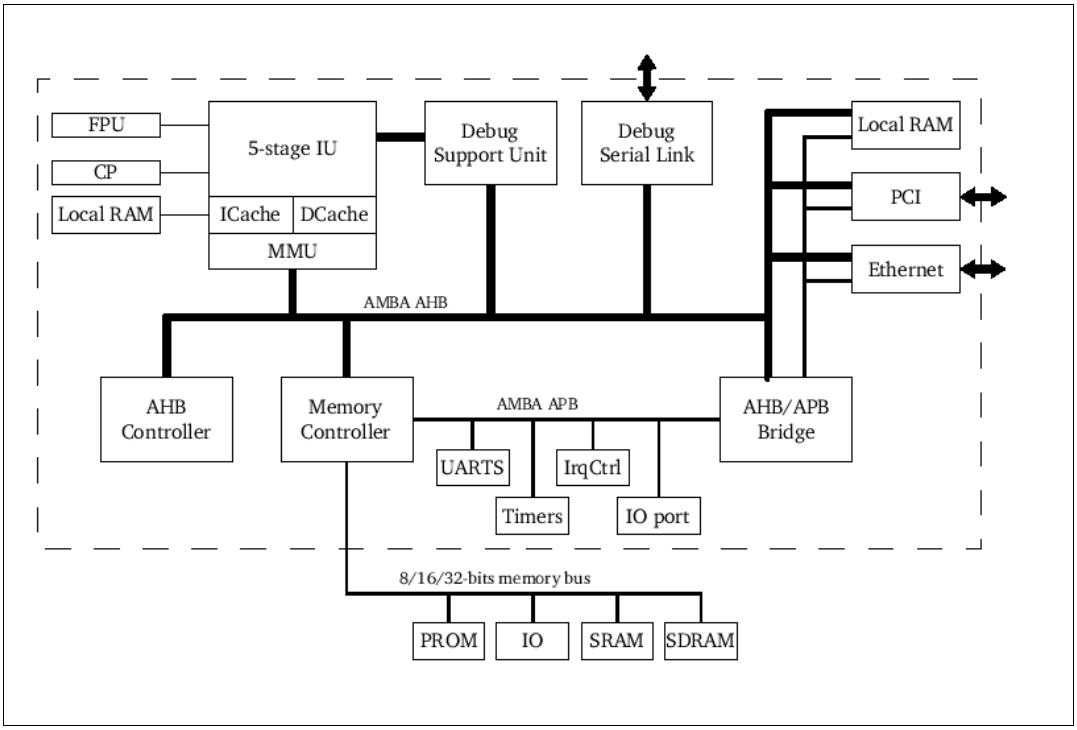

Figure 2: LEON2 microprocessor architecture

- *two serial port controllers, UART*
- *16-bit I/O port mapped on memory*
- *watchdog & power-down*
- *Ethernet controller 10/100 MAC*
- *PCI interface*
- *co-processor interface*

SPARC v8 processor defines three main units, integer unit, floating-point unit and a custom co-processor, each one with its own 32-bit internal registers. The later two units are optional, not mandatory for the processor to be SPARC complaint. LEON2 implements the integer unit completely and the interfaces for the other two units in its core. Gaisler Research also has a commercial high-performance FPU for LEON2 available [CATO03], fully IEEE-754 compliant [STEV81]. LEON2 also can provide a generic interface for a custom user-defined co-processor which will work in parallel with the main processor in order to increase performance.

LEON2 uses the AMBA-2.0 AHB bus to connect the main processor with high-speed controller like cache and memory ones and other optional units like the onchip RAM or PCI or Ethernet interfaces. In the default configuration LEON2 is the only master of the bus.

Another AMBA-2.0 bus is used to access most on-chip peripherals, the APB bus. It's optimized for simple operation and low-power consumption and it's connected to the AHB (and LEON2) via the AHB/APB Bridge, which is the master of that bus.

LEON2 external memory access is provided by a programmable memory controller with interfaces to PROM, SRAM & SDRAM chips, providing also memory mapped I/O operation. The controller can decode a map of up to 2 Gbytes.

3. DESIGN METHODOLOGY

In order to implement a System-on-Chip application based on the LEON2 synthesizable microprocessor on a FPGA board we've defined a design methodology with a dual design flow, so starting with the specifications of our applications we must consider a different (but closely related) flow for the hardware design and for the software design of the SoC. A representation of this methodology is shown in figure 3.

The design flow for the hardware of the SoC application consist of the configuration of the LEON2 model and all its modules, next there is the possibility of adding a new hardware peripheral to any of the onchip buses (usually to the APB) or as a co-processor (in order to improve the overall performance), then arriving to the simulation step and finally coming to the synthesis and implementation process.

The other main flow of this methodology is the software development design which will provide us with the software application which will run on the hardware LEON2 model we obtained in the hardware flow. This is a typical software development flow, with the special features of the LEON2 tools available for the SoC implementation.

Both design flows are closely related but can be covered in parallel and with little or none dependences between them. At the end we can get the hardware model of the SoC and the software application and implement both on a real hardware target to get the SoC running and do the final complete verification and monitor checking.

3.1 Hardware Design Flow

The design flow for the hardware of our LEON2 based SoC application includes the configuration and adaptation of the processor model to our needs and also the inclusion of any additional user-designed peripheral to the core for a particular application.

So, the first step in the design process is to configure the LEON2 model with a selection of the onchip peripherals it includes, according to our needs, and also giving values to the main parameters that defines the behaviour of the core. This can be accomplished by using a graphical configuration tool provided with the LEON2 model. This tool works on both Windows (with CYGWIN) and Unix/Linux and it is build upon the TCL/TK software1, providing some menus that allows the designer to configure how the different units of LEON2 will work and which ones will be implemented in the final hardware.

Once the model is configured to fit the actual SoC requirements it is possible to simulate the model by running a testbench that stimulates the main components of LEON2 in order to check if the results obtained are accurate for our application needs. This testbench is provided with the LEON2 VHDL model

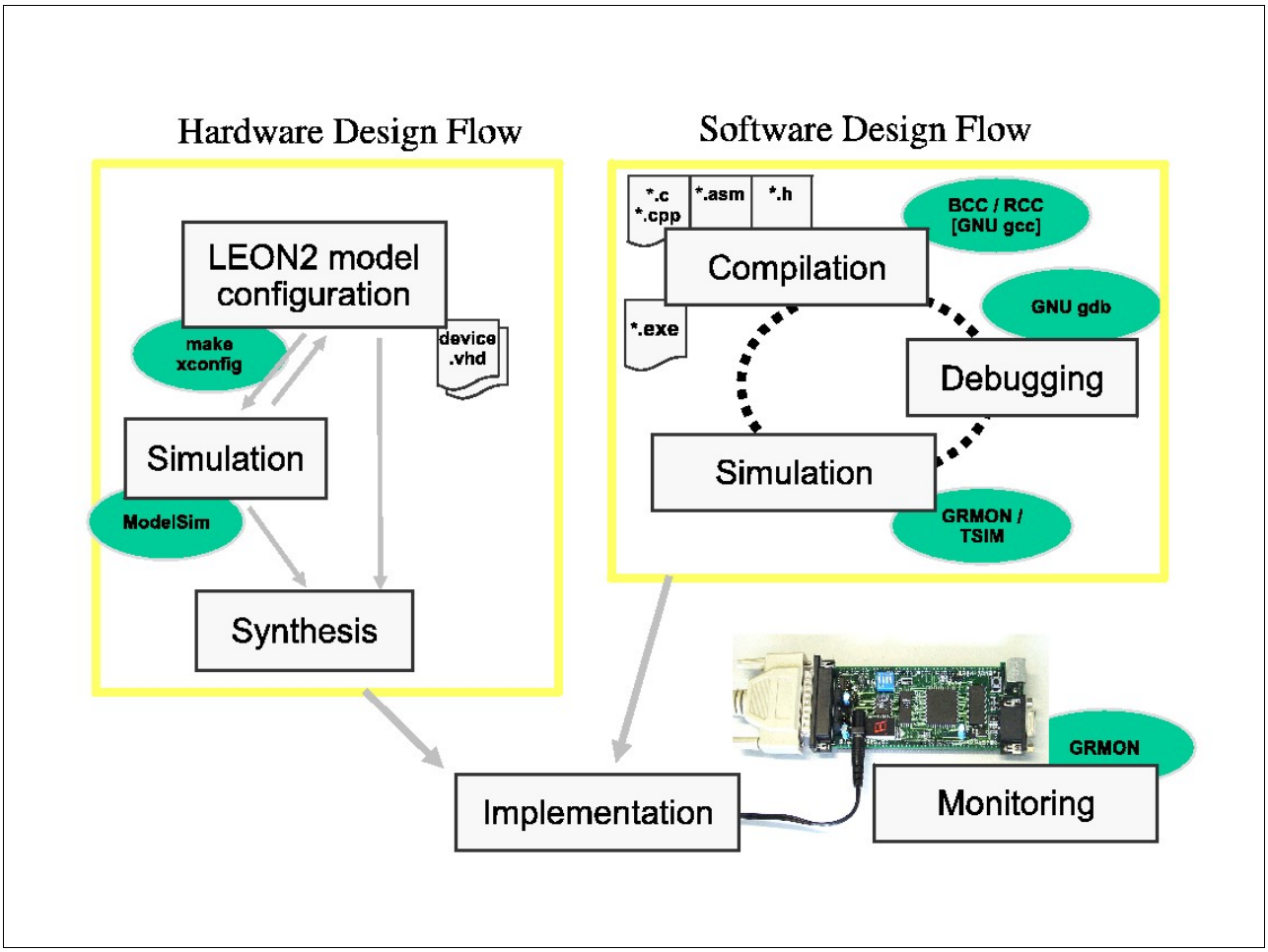

Figure 3: SoC Design Methodology for LEON2 on FPGA

and it works with many major simulation tools, like Modelsim from Mentor1, NCSIM from Cadence2, VSS from Synopsys3 or GHDL from GNU.

One of the most important steps in the hardware design flow is to provide a way to add our own user-designed peripherals to the actual LEON2 core in order to add a new specialized functionality. The most usual way to obtain this is to add the peripheral to the AMBA onchip buses LEON2 works with and do the adaptation on some of the controllers to get the new core working in conjunction with the rest of the microprocessor. In this work we've accomplished this goal by adding a new peripheral to the model attaching it to the APB bus just as most of the controllers of LEON2 are connected too. This is explained in detail on Section 4.

Next step on our design flow is to synthesize the SoC hardware model described in the previous steps. The synthesis tool will compile all the descriptions on the VHDL files and build a netlist of the target FPGA components and it's interconnections. LEON2 is designed with synthesis process in mind so most of the common synthesis tools can be used to accomplish this goal, like XST from Xilinx ISE software package [XISE04], Synplify from Synplicity4, Design Compiler from Synopsys or Leonardo from Mentor.

And finally we reach the implementation step where the proper FPGA tools run the processes of mapping and place & route to get a final bitstream file with the information needed to program the FPGA with the hardware model obtained in the complete flow. To accomplish this goal LEON2 includes support for several FPGA development boards on market, providing configuration files and automated scripts to implement the model onto those boards.

LEON2 comes with several scripts and project files for common synthesis and implementation tools so the designer can use them to work with the complete

project described in VHDL files from the usual tool as with any other hardware project.

3.2 Software design flow

The SoC software design flow usually runs in parallel to the hardware design flow and there are close influences between them, in the sense that the hardware that is going to run the software must be well known by the designer.

The methodology for this design flow is based on the GNU toolchain, with the software development as the first step, next comes the compilation, also the debugging and finally the implementation on the hardware. Also there's a LEON2 simulator that can be used to accelerate the design flow and to check the software on a virtual system made by the simulator. On this chapter we're presenting not only the design flow for software applications but also all the tools the designer can choose to use in his/her work.

When trying to design a big application, with a lot of code, it's always a nice idea to use any of the visual development tools to arrange some files in projects and have one integrated tool so the designer doesn't have to explore all the interrelations between the multiple low-level tools available. LEON2 developers have provided Eclipse1 with some facilities to be able to design software for this microprocessor, by adding a plug-in to the platform, so we can manage most of the project operations into the same environment.

There are two different C/C++ cross compilers available for LEON2, both running on Windows and Unix/Linux and with GPL licenses, and with a similar functionality than the GCC toolchain. They are BCC and RTEMS-CC, both integrated into Eclipse as mentioned above.

BCC, which stands for Bare-C Cross Compiler, is a C/C++ cross compiler for the LEON2 processor based on the GNU toolchain, the binutils and the standard Newlib library, with full math support and simple I/O operations (non-files). It's a simple bare-c runtime system with support for interrupts and single-threaded applications. If also allows to perform source-level symbolic debugging, both on the LEON2 simulator and on real hardware, including the support for hardware multiplier/divider. It doesn't support the floating-point unit.

The RTEMS Cross Compiler, RTEMS-CC, is mostly like BCC but it introduces in the final application the RTEMS kernel [RTEM03] where the designed program will run. It's a multi-task kernel with real-time features and it also includes support for the floating-point unit in hardware.

There are also other kernels which can be the base for the software application being developed. So, furthermore the RTEMS kernel, we can also use eCos and Linux for our LEON2 SoC applications. All of them are free software.

eCos stands for embedded Configurable operating system and it's a kernel for embedded applications with real-time features [MASS02], ported to a lot of different microprocessor architectures. It's highly configurable and it comes with support for the LEON2 hardware implemented floating-point unit and also the Ethernet interface.

Linux is supported in LEON2 by a particular release of the SnapGear Embedded Linux distribution, and it can be run two different kernels, standard Linux 2.6 for cores with memory management unit (MMU) implemented in hardware, and also ucLinux 2.0 [MCCU03], a modified version for embedded processors without the MMU. It includes also some usual libraries and other tools to build embedded systems with Linux. It has support for the hardware multiplier/divider and also the hardware floating point unit.

Debugging the application is one important step in the software development flow, to validate the results obtained with the program in the SoC. Gaisler Research also includes a GDB-like debugger, also from the GNU toolchain, ported to LEON2 and available for both compilers BCC and RTEMS-CC and it can be ran both on real hardware and also on the LEON2 simulator.

Gaisler Research has released TSIM, a complete LEON2 instruction-level simulator. TSIM can run in standalone mode or connected through a network socket to the GDB debugger, acting like a remote target using a common debugging protocol, so it can be used with another interfaces, like the graphical debugging tool DDD. TSIM is a commercial application but it's also available as an evaluation version for non-commercial uses.

3.3 SoC Implementation on Hardware

Once completed the two main goals, both the hardware flow and the software flow, we've obtained a

hardware model of the LEON2 core, with some personalization for our needs, and also the software that the microprocessor will run. To complete the proposed methodology we have to implement the SoC on the target hardware and also run the software on the SoC and check the results.

In this work we established a procedure to implement the SoC on a FPGA evaluation board from Avnet which provides a Xilinx Virtex-II 1000 FPGA with a million equivalent gates.

At the end of the hardware design flow we obtained a bitstream with the information to program the FPGA device with the LEON2 microprocessor core. We have to download it to the FPGA using the Xilinx iMPACT programming tool through a JTAG cable and in a few seconds the SoC core will be implemented on the hardware.

To download the software to get it running on the LEON2 we need to use a tool from Gaisler Research which allows to communicate with the processor for a non-intrusive monitoring and debugging, providing full access to internal registers, memories and all the main peripherals. This tool, the LEON2 monitor, named GRMON, has a commercial license and an evaluation version is also available for non-commercial purposes.

As a final step, debugging of the complete SoC application is possible thanks to GRMON also, so that we can integrate it with GDB and perform a debug of the software running on the real hardware.

In order to use the LEON2 monitor we have to include the debug support unit (DSU) in the SoC model. This unit will provide a link for the communication with GRMON, via a serial cable or a PCI interface. Also, there's a graphical interface for the GRMON tool so it appears integrated in the Eclipse toolkit, to have access to most of the relevant information in a simple view.

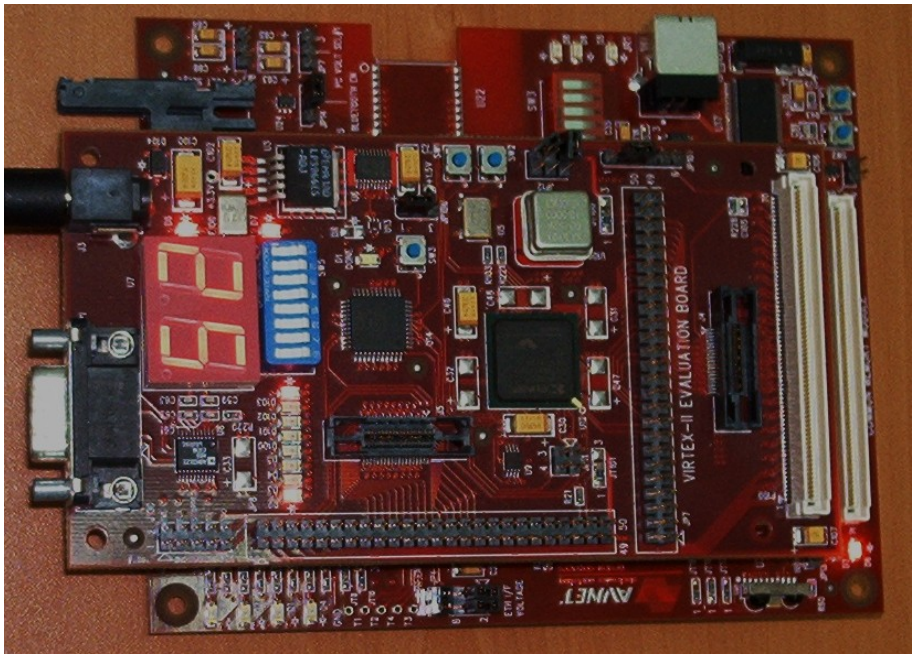

Figure 4: FPGA board with the sample SoC application

4. APPLICATION EXAMPLE

In this final chapter we put together all the steps in the proposed methodology by making an example SoC application based on LEON2 and adding a new onchip peripheral to the core, connected to the AMBA-2.0 APB internal bus. The SoC runs on a FPGA evaluation board from Avnet, as shown in Figure 4, and is monitored from a PC via a serial-cable connection.

4.1 LEON2 Adaptation for the Evaluation Board

LEON2 comes with support for various FPGA evaluation and development boards from different manufacturers, providing several configuration files and user-constraints for each model of board to automatize the synthesis and implementation steps as much as possible. But we have one board not supported by LEON2 so there's a first step which we had to complete prior to implementing the core for the board.

The board used was a “Virtex-II Evaluation Kit” from Avnet, with a Xilinx Virtex-II 1000 FPGA integrated, and also a JTAG interface and serial port, several LEDs, buttons and switches, and with a clock generator of 40 Mhz. As this board doesn't have any RAM memory chips it was also needed to connect this one to a “Communications and Memory Module”,

also from Avnet, with different banks of SRAM (1 MByte) and SDRAM (64 Mbytes) memory and some communications modules like Ethernet, USB and Bluetooth ports.

First we need to choose the internal peripherals in LEON2 according to the peripherals on the board, so we can have a microprocessor model suited for that board.

It was also necessary to change the behaviour of some of the components of LEON2, for example in the memory controller, because the signals of the different memory banks have some differences with the LEON2 usual interface; so a wrapper for LEON2 was created to change the external interface slightly to fit with the RAM banks on this specific board. The wrapper was designed in VHDL and integrated in the LEON2 hardware design flow by modifying the scripts for automatic synthesis and implementation.

And finally we constructed a user constraints file (UCF) for that combined board, to map the FPGA pads (on the Virtex-II Evaluation Kit) to the corresponding pins of the different peripherals and memory banks (on any of the two boards).

4.2 Adding a New Peripheral to LEON2

A new peripheral was designed for LEON2 in order to test the accuracy of the proposed SoC methodology. This peripheral was connected to the AMBA-2.0 APB bus as a slave, like most of the LEON2 included peripherals, which provides a simple interface and is designed for low-power consumption.

The interface for write and read operations for a slave peripheral connected to the APB bus are shown in figures 5 & 6. Each APB peripheral has the following control signals, described in VHDL:

```
-- APB slave inputs
PSEL:     Std_ULogic;                    -- slave select
PENABLE:  Std_ULogic;                    -- strobe
PADDR:    Std_Logic_Vector(PAMAX-1 downto 0);
-- address bus
PWRITE:   Std_ULogic;                    -- write
PWDATA:   Std_Logic_Vector(PDMAX-1 downto
0); -- write data bus
-- APB slave outputs
PRDATA:   Std_Logic_Vector(PDMAX-1 downto 0);
-- read data bus
```

The peripheral is designed completely in VHDL and is a controller for the dual 7-segment displays available on the board, for debugging purposes. The peripheral has two internal registers which are mapped on memory and the contents are shown in decimal on the two displays.

Following the proposed methodology, moreover the design of the APB slave peripheral it's also necessary to change some of the blocks in LEON2 to add the new core to the existing model to run all them together flawlessly. So in order to get a new APB slave working in the LEON2 model these VHDL files need to be modified:

ambacomp.vhd, where there's a declaration of each component connected to the AMBA-2.0 AHB or APB buses. The new component have to be declared.

ambamst.vhd, which describes the behaviour of the APB master (the AHB(APB Bridge). We have to add a entry to select the new component when necessary, mapping its accesses by allocating some space on the memory map.

mcore.vhd, where there's an instance of each peripheral mapped to the memory. The new component has to be instanced and all their interface signals connected to the rest of the logical blocks or external signals.

leon.xst, that's a script for the synthesis tools (XST from Xilinx in this case) to compile and implement all the necessary files automatically. A new entry has to be included to add the new APB core.

To test the peripheral a C application has also been developed that changes the values shown on the displays from 00 to 99, working like a decimal counter.

5. CONCLUSIONS

The importance of SoC design for embedded and ubiquitous applications has been highlighted, besides the versatility and robustness of free cores like LEON2 to establish a professional-level SoC design methodology.

A SoC design methodology for LEON2 on Xilinx FPGAs has been developed exploring the different software and hardware tools available. SoC hardware

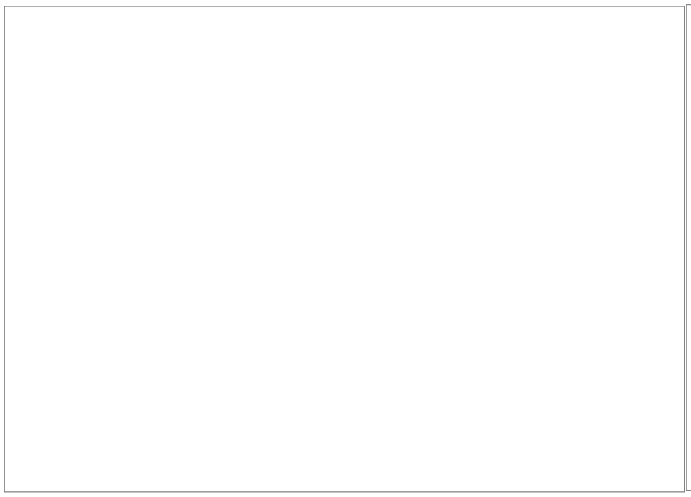

Figure 5: Write operation on a APB slave peripheral

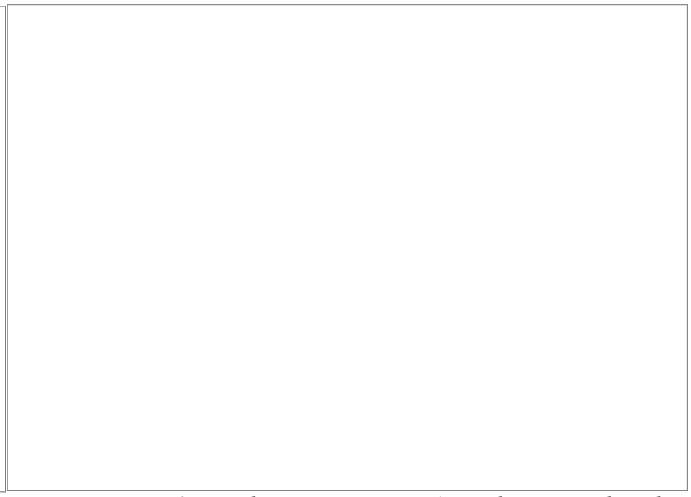

Figure 6: Read operation on a APB slave peripheral

development using LEON2 is highly productive due to the large number of pre-defined and highly configurable devices included with LEON2. The same applies to software, where a variety of platforms are available, ranging from compact libraries to whole operating systems like Linux. As an example, a new home-made device has been added to the basic LEON2 architecture using the standard APB expansion bus and a test software program has been written. The implementation cycle, including APB interfacing logic, is done in a matter of hours showing the productivity of the platform.

Additional infrastructure has been added to the LEON2 standard package in order to support the specific evaluation board used in the experiments. This enhanced functionality will be contributed to the LEON2 project.

The good perspectives derived from this experience with LEON2 has motivated us to start porting some industrial SoC developments in which our group is currently involved to the LEON2 platform, including software and hardware co-design.

6. REFERENCES

[AMBA99] "AMBA (tm) Specification, Rev. 2.0", ARM Limited, 1999. http://www.arm.com/

[CATO03] Edvin Catovic: "GRFPU – High Performance IEEE 754 Floating-Point Unit", Gaisler Research, 2003.

[FURB00] Steve Furber: "ARM system-on-chip architecture, 2nd edition", Ed. Addison-Wesley 2000.

[GAIS04] Jiri Gaisler: "LEON2 Processor User's Manual", Gaisler Research, 2004

[HENN93] John L. Hennessy, David A. Patterson: "Arquitectura de Computadores – Un enfoque Cuantitativo", Ed. Mc-Graw-Hill 1993.

[LAMP01] Damjan Lampret: "OpenRISC 1200 IP Core Specification", 2001

[MASS02] Anthony Massa: "Embedded Software Development with eCos", Ed. Prentice Hall, 2002.

[MCCU03] David McCullough: "Getting started with ucLinux", Cyberguard Technical Bulletin #12, 2003.

[MICR05] "Microblaze Processor Reference Guide", Xilinx Inc. 2005, http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf

[NIOS04] "NIOS 3 CPU Data Sheet", Altera Corp. 2004, http://www.altera.com/literature/ds/ds_nios_cpu.pdf

[POWE04] "IBM PowerPC Quick Reference Guide", IBM Corp. 2004

[RTEM03] "Getting Started with RTEMS", Online Applications Research Corp., 2003.

[SEAM02] Graham Seaman: "Free Hardware: Past, Present & Future", Erste Oekonux Konferenz, 2002

[SPAR92] "The SPARC Architecture Manual, Version 8", SPARC International Inc., 1992.

[STEV81] David Stevenson, et al: "A Proposed Standard for Binary Floating-Point Arithmetic, IEEE Computer Vol. 14, 1981.

[XISE04] "ISE 6.3i Release Notes and Installation Guide", Xilinx Inc. 2004