Building a SoC for industrial applications based on LEON microprocessor and a GNU/Linux distribution

A.Muñoz, E.Ostúa, M.J.Bellido, A.Millán, J.Juan, D.Guerrero
Grupo de Investigación y Desarrollo Digital (ID2)
Departamento de Tecnología Electrónica - Universidad de Sevilla
Av. Reina Mercedes, s/n (E. T. S. Ingeniería Informática) - 41012 Sevilla (Spain)
Tel.: +34 954556160 - Fax: +34 954552764

email: { amrivera; ostua; bellido; amillan; jjchico; guerre; } @ dte.us.es

***Abstract*- This paper presents the design of a complete RTU (*Remote Terminal Unit*) with a System-on-Chip solution based completely on both open hardware and software platforms, and developed in conjunction with two industrial companies. The target implementation of the embedded system is a Spartan family FPGA from Xilinx. The article presents the main features of the base system, which consists of the LEON microprocessor and a Linux operating system distribution (Debian) running on it. Moreover, it shows a complete example of how to add new peripherals to the system. The peripheral that has been added is the UART 16550 compatible peripheral available in OpenCores. Given that the design has been prepared for the WishBone bus, it was necessary to adapt it to the APB bus within the LEON core. Furthermore, it has been adapted to work with the Linux driver for that UART to get a full coupling of peripheral with the system. The experimental results confirm the work done.**

I. Introduction

Latest technological researches give rise to the development of chips for high density integration and as a result it is possible to include an entire system on a single chip. This is known as SoC methodology. There are two technology options for SoC designs: FPGAs and ASICs. Implementing a SoC as an ASIC has the advantage of a better performance that can be achieved in terms of speed and power consumption. But from an industrial point of view the development and manufacture cost of ASICs is very high and it's only offset by large numbers of manufacturing. Instead, implementing a SoC on an FPGA, while it can decrease the performance, has a very similar cost per unit of the system manufactured. In addition, FPGAs have the advantage of being reconfigurable in real time, something that it's not possible with an ASIC, which allows the reuse of the same device for different tasks when each of those tasks do not require constant attention. Thus, taking into account that most of the industrial applications do not require a large number of units, the FPGA becomes a very appropriate solution for system implementation in these environments.

Moreover, nearly 100% of the SoC are built around a microprocessor that acts as the central nucleus of the entire system. The choice of the microprocessor determines heavily it's performance. In most industrial applications is not necessary to achieve a high processing capacity but mainly a good reliability. This means that the microprocessors that are used in such systems do not have to be very complex. There are a number of microprocessors that has been routinely used in the design of SoC solutions, both commercial cores, namely those that require the purchase of a license for its use, such as ARM [1], PowerPC [2], NIOS3 [3], MicroBlaze [4] or others, as well as free open source cores such as LEON [5] or OpenRisc [6]. Commercial microprocessors are generally well tested and optimized, and have integrated tools that facilitate the application development process. However, for certain businesses or industries the high cost to acquire the entire development system can be unfordable. In contrast, open systems do not have the disadvantage of the cost. However, it is a widespread idea that it's very complex to work with these systems.

In our research group we have been working for several years with open systems both from a hardware and a software perspective. Currently, we have a development platform for hardware and software SoC design for FPGAs completely based on open systems. This enables us to ensure that it is feasible to use such systems on industrial applications, assuring that the system have a development time and reliability more than acceptable on its final implementation.

Thus, in this study we pretend to show the work done on a project that aims to develop an embedded system to operate as a remote terminal unit (RTU) for a network of industrials control application.

This project is supported by the Andalusian Regional Government and is monitored by companies of our local environment as Telvent [7] and Guadaltel [8]. Thus, this platform is being implemented as a RTU part of a comprehensive system of tracking and positioning, developed by Guadaltel. The RTU is intended to control the positioning through a GPS built into the system. For data communication both GSM and radio are used. Both GPS and communication systems will communicate with the microprocessor core of the RTU via a RS-232 port.

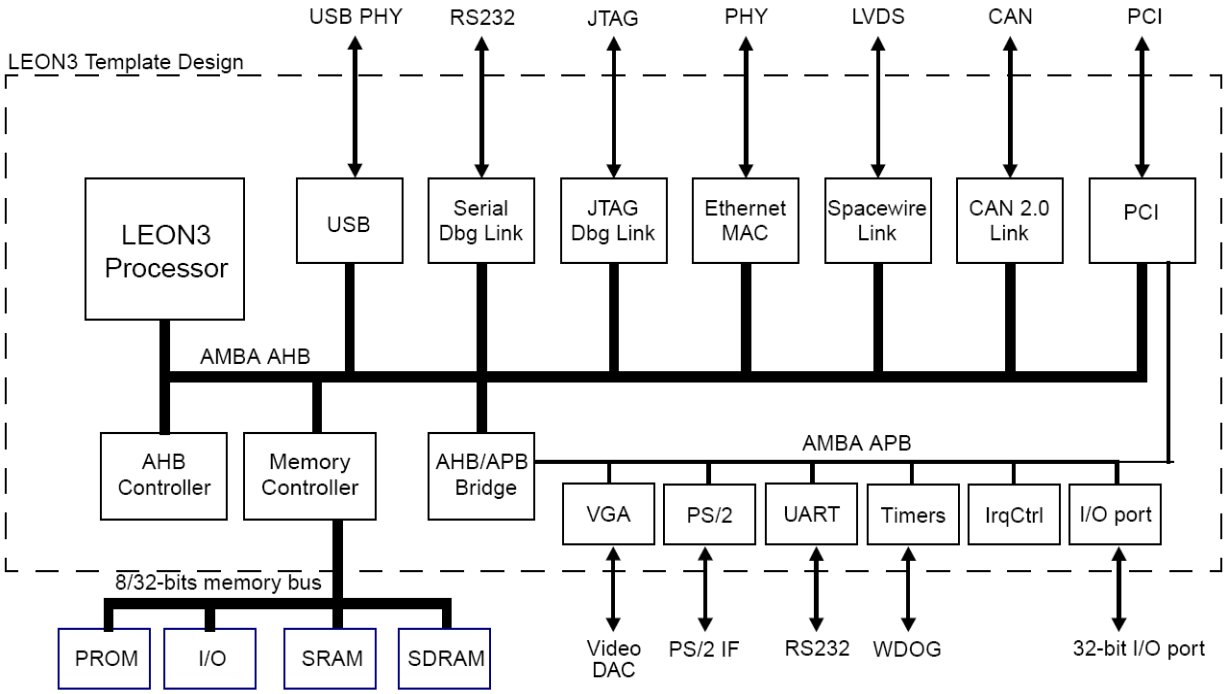

Figure 1: LEON microprocessor system architecture (taken from [5])

The development platform that is used to build this RTU is based on the LEON microprocessor. On top of this microprocessor is running a Linux operating system. The LEON architecture it fully SPARC v8 compatible. This allow the implementation of a regular Linux distribution, like Debian [9], on an embedded system based on the LEON core, as described on some of our previous works [10]. This is a huge advantage from the point of view of software development because Debian Linux has a lot of software already developed and conforms a very easily deployable system.

With this platform basis (LEON microprocessor + Linux-Debian OS), the development of new peripherals for the system requires not only to have the hardware design, but also to build the kernel driver so the OS can actually use the peripheral. Indeed, during the process of designing this RTU within this project, there was a need to incorporate a new driver for a standard RS-232 serial port, because the one that comes with the own LEON's based Linux kernel is not fully compatible and can not be used with physical devices for enhanced serial communications. In order to achieve this, it was decided to port an open design of the controller RS-232, specifically, one UART 16550 controller available in from OpenCores[11]. In order to include this design within our base platform, it was necessary to perform some work on the adaptation, both hardware and software, which is discussed in this paper. Taking this into consideration, the organization of this paper is as follows: The next section is devoted to presenting the main features of the development platform based on LEON + Linux-Debian. The third section shows the design methodology to be followed in order to adapt the UART 16550 IP core from OpenCores into our development platform. In the fourth section, experimental results of the system implemented on the FPGA are shown. The final section summarizes the main conclusions of this work.

II. Development with Leon Microprocessor Core

LEON is a 32-bit microprocessor core which implements a RISC architecture conforming to the SPARC v8 definition [12]. 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 GPL license so it is an open hardware [13] 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. Because of the viral nature of the GPL licensing scheme, Gaisler recently offers also an option to get a commercial license for LEON3 so the source code developed within a project with this core doesn't have to be distributed.

The LEON3 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 7-stage instructions pipeline. It also implements two separate instruction and data cache interfaces, known as Harvard Architecture [14].

A typical LEON3 configuration block diagram for a SoC application is shown in Figure 1. Many of those blocks are optional and can be removed from the model our concrete application implements.

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, which makes it easy to extend its functionality. All this features makes LEON3 an ideal microprocessor for System-on-Chip applications.

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. LEON3 implements the integer unit completely and the interfaces for the other two units in its core. LEON3 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.

LEON3 uses the AMBA-2.0 AHB [15] 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.

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 via the AHB/APB Bridge, which is the master of that bus. This bus is what we pretend to connect the UART 16550 core with.

LEON3 external memory access is provided by a programmable memory controller with interfaces to PROM, SRAM, SSRAM, DDR & SDRAM chips, providing also memory mapped I/O operation. The controller can decode a map of up to 2 Gbytes.

Linux is supported in LEON by a particular release of the SnapGear Embedded Linux distribution, and it can be run two different kernels, regular Linux 2.6 for cores with Memory Management Unit (MMU) implemented in hardware, and also ucLinux 2.0, a modified version targeted 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.

On top of our core system we have implemented an open-source Debian distribution with a Linux 2.6 kernel, with all the necessary drivers to interact with each single peripheral for the microprocessor. Finally, user-space applications running on the OS will interact with this system through the system calls the kernel and base libraries it provides.

In order to get a full Linux distribution like Debian running on top of the FPGA development board, we have interfaced a Compact Flash card reader to the FPGA and used it via the standard IDE protocol as a regular hard disk, so the installation process worked the same as on a PC platform, and after that, also the regular boot process for everyday use during the development of the whole application. The complete details for the process of integration of both Linux and LEON platforms is described in [10].

On Figure 2 a full system overview is shown, from the silicon pieces in the lower level up to the user applications on the top.

III. Adding an UART 16550 to Leon Platform

First of all, after some researching we decided to integrate an UART 16550 to LEON microprocessor to get a fully capable UART instead of the single one included in the core, and we planned to use one of freely available cores on the OpenCores community. In the first place, it should be necessary to adapt the UART 16550 from OpenCores to our SoC internal bus.

Most peripherals available on OpenCores implement the WishBone interconnection architecture, a open hardware bus solution, created and specified by OpenCores and commonly used on many embedded systems applications. But LEON microprocessor, as mention before, uses the AMBA-2.0 AHB and APB buses internally.

In particular, we pretend to use the AMBA-APB bus for the new UART 16550, as it is a specific bus for this kind of relatively slow and simple peripherals. The architecture and behavior of those buses are quite different in general, being the most complete and functional the WishBone, but in particular the writing and reading mechanism of the internal registers are very similar on AMBA-APB and WishBone. Both buses have a

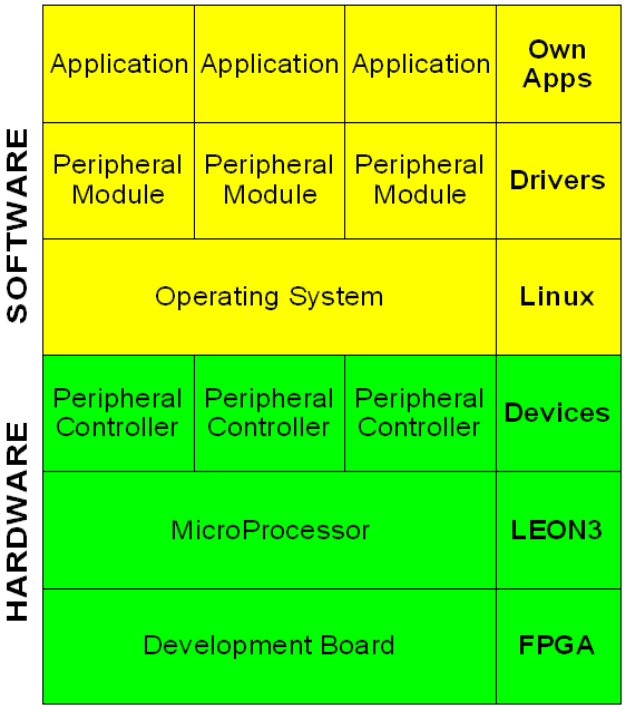

Figure 2: Hardware and Software global architecture

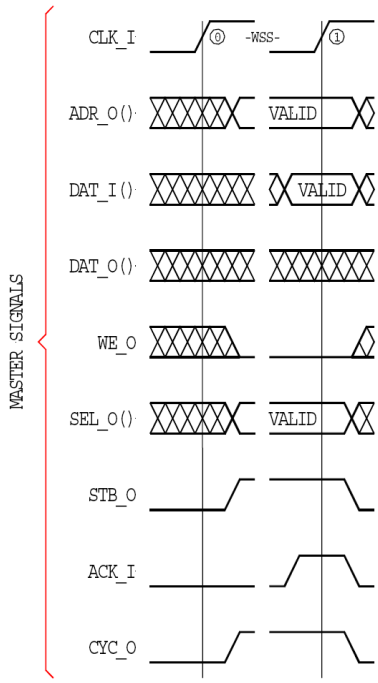

Figure 3: WishBone Single-Read operation (taken from [16])

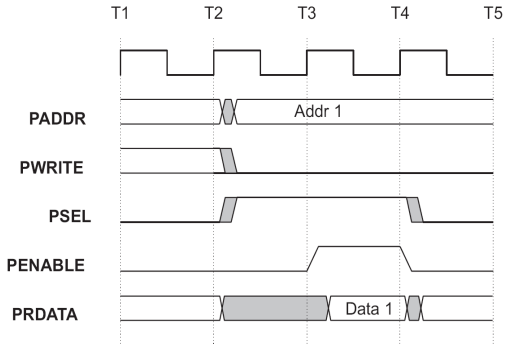

Figure 4: AMBA-APB Read operation (taken from [15])

32-bit address bus and two 32-bit independent data buses independent for reading and writing operations with the target peripheral.

In order to create an interface between the buses it is important to study the single transfer operations to peripherals on both buses. So, in example, a single read operation is shown in Figure 3 and Figure 4, for Wishbone and AMBA-APB respectively.

After a preliminary analysis, it is clear that the main difference between the two buses architecture lies in the signal *ack_o* from the WishBone bus. This signal enables the slave bus to insert waiting-cycles in the operation, i.e. keep the master device waiting for the acknowledgement from the slave peripheral until the operation ends.

However, the AMBA-APB bus does not have this signal acknowledge, so every operation have always the same duration. This implies that the master device never have to wait for the slave peripheral, which must end the operation within the clock cycles limitation without exception. This difference is essential when interfacing the communication between the two buses. Thus, the UART 16550 from OpenCores behaves as a bus slave peripheral on its operations, with the peculiarity that makes use of the signal acknowledge readings in the transfers, i.e. inserting wait states to the master. This UART behaviour prevents us from doing a simple signal connection between both buses, only supported by simple combinational logic. Therefore, in order to fulfil the AMBA-APB bus specifications, it is necessary to make some major changes in the implementation of the UART interfacing to Wishbone bus, so we can remove the need for the insertion of the waiting states cycles.

Reviewing the implementation of the Wishbone interface module in the UART core, we noticed that the main cause of these undesired wait states falls on the sampling of all the input signals. This sampling and subsequent storage in intermediate registers, causes a single clock cycle delay at least. So we have to change this behaviour by eliminating this sampling, and taking immediately the data received when the control signals ask the peripheral to do so.

It is also necessary to do a second modification, because of the disparity between the size of the actual data bus (32-bits) and the size of the UART control registers (8-bits). So, in our design the byte address of each word is aligned with the least significant part of data bus. Thus, addressing for these registers will be made by transferring 32 bits in each access and so address for the next registered will be 4 bytes above. Therefore, on each access to our peripheral registers we have to ignore the two least significant bits of the address bus that arrives from the AMBA-APB.

From the standpoint of hardware description language, It is necessary to modify the peripheral core to include a higher level wrapper which supply this functionality. This entity will help us make sense of alignment between the signals of both buses and to generate information for Plug & Play system that LEON introduces. The majority of connections are conducted in a direct way, except for some control signals such as reset signal to be inverted or the generation of signals *stb_i* and *cyc_i*, which control the transfer cycle and the cycle bus respectively and will be produced by the device signal selection *apb_psel.*

Plug & Play information added to the system will enable the identification and allocation map directions peripherals automatically.

Finally we must not forget connecting the interrupt signal of the UART with a free line of the interrupt controller system.

In the software side, this UART model is supported by the regular Linux kernel driver called *8250.c*. This driver is ready for various architectures like x86, but not for the SPARC architecture, implemented by the Leon microprocessor.

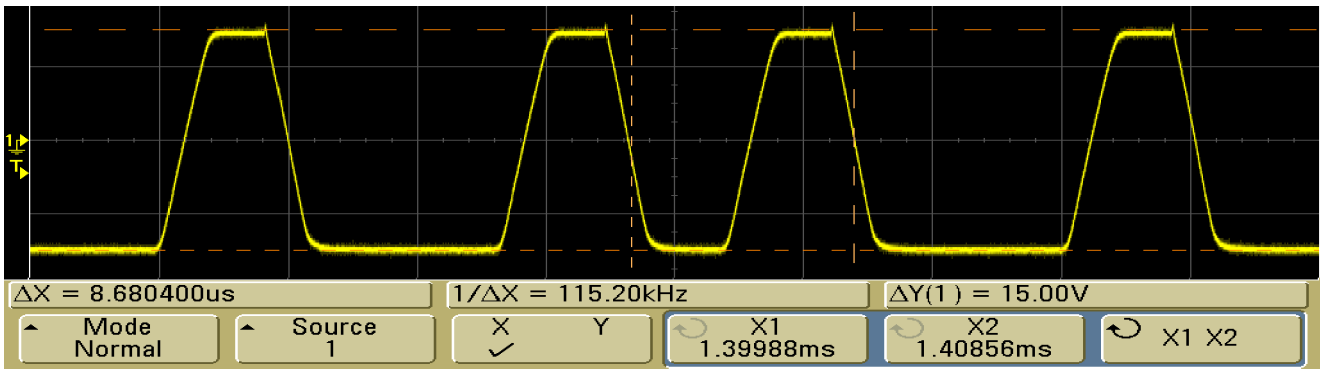

Figure 5: Tx signal on character transmission through the UART 16550.

Therefore it is necessary to modify and adapt the driver for an usable operation.

First, we must modify the building scripts within the kernel sources to allow the inclusion of this driver in the configuration. Once that is completed, we have to adapt the driver source files, starting from the file of register definitions, called *serial_reg.h.* We must change de UART register offsets so it because the AMBA-APB to WishBone address adapter is routed with 2 bits displacement, as discused above, we have to update it for this purpose (multiplying each register address by four).

Moreover, the behaviour of this driver is to try to locate the UART peripheral by probing the addressing where they are usually found for each system architecture. This information is contained in the file *serial.h*, in the *asm* directory of the specific architecture. For example, for x86 it contains four addresses corresponding to the ports from COM1 to COM4 (called /dev/ttyS0 to S3 in Linux), with map addresses ranging from 0x3F8 for COM1 to 0x2E8 for COM4.

At this point we should choose between using the search functions for system devices provided by the Plug & Play mechanism, or by allocating a lookup table with fixed addressing for our new UART 16550 peripheral. For simplicity the second option was implemented, although it would be relatively simple to use the first option.

The last source file to amend is the main source driver, the file *8250.c*, where, first, we had to change the functions for reading and writing in the system bus. These C language functions we are changing are *outb* and *inb*, used for writing (and reading) bytes to (and from) I/O addressing. The changes are needed to fulfil the writing procedure on the AMBA-APB bus, built within the Leon core. These functions are called *leon_bypass_load_pa* and *leon_bypass_store_pa* which are located in the *leon.h* source file. Its main task is to perform non-cached 32-bit readings and writings directly on the AMBA-APB bus.

A remarkable aspect is to modify the parameter that indicates the clock frequency that govern the UART, as it is typically fitted with 1.8432 MHz clock (limiting the top speed of the UART to 115200 bauds), so that the driver assumes that it is set to that value. In our design it has been modified to 40 MHz, which is the main clock speed used by the SOC.

Finally, and perhaps the deepest change, was the bypass of routine for UART type auto detection, which is linked to the specific input-output addressing. This is done so because the UART16550 of OpenCores does not include a *scratch* register that is used by the driver solely for the purpose of determining the exact model UART available on the system. Thus, we forced the driver to detect the UART as a NS16550A compatible one.

IV. Experimental Results

After a successful compilation and installation, the Linux kernel have automatically detected and configured the new device. As shown on Figure 6, where a snapshot of debug information is included. information relating to the UART shown by the Linux kernel in.

The results are quite satisfactory, taking all available features and characteristics of UARTs from desktops, such as flow control lines and modem interaction.

Sending and receiving is done in a proper manner on the full range of standard baudrates. On Figure 5 we show a screen capture from the oscilloscope on the TX signal while a byte

```
Leon3Debian:~# dmesg | grep  "NS16550A" -C 5
grlib apbUART: system frequency: 40000 khz, baud rates: 38400 38400
ttyS0 at MMIO 0x80000100 (irq = 2) is a Leon
ttyS1 at MMIO 0x80000900 (irq = 3) is a Leon
Serial driver 16550 Opencore/APB, by D.T.E/U.S.
Looking for UART 2 as ttyS2: I/O address 0x80000c00
serial8250: ttyS2 at I/O 0x80000c00 (irq = 11) is a NS16550A
Looking for UART 3 as ttyS3: I/O address: 0x80000d00
serial8250: ttyS3 at I/O 0x80000d00 (irq = 13) is a NS16550A
loop: loaded (max 8 devices)
Probing GRETH Ethernet Core at 0x80000b00
10/100 GRETH Ethermac at [0x80000b00] irq 12. Running 100 Mbps full
duplex
PHY: Vendor 4de   Device e   Revision 2
Uniform Multi-Platform E-IDE driver Revision: 7.00alpha2
Leon3Debian:~#
```

Figure 6: Debugging information shown on system boot

was sent by the UART with a 115200 bauds configuration, 8 data bits, no parity and 1 stop bit (that's 115200@8N1).

This speed can indeed be easily increasedby writing a smaller value in the register divider clock inside the UART.

V. Conclusions

This paper has presented the development of a hardware platform implemented on FPGA. The main feature of this platform is that is based on open designs. Specifically, the central part of the system is the soft core microprocessor LEON. On the software level, a Linux kernel it's running with the typical software installation of a full Debian distribution system. This adds another great feature to this platform, as you can get a lot of functional software in a simple way, by using the complete, but yet very easy to setup, installation procedure of these types of Linux distributions.

Moreover, it has been shown the method to be followed to add new peripherals on this platform introducing the example of the UART 16550. For this UART, we have employed an open hardware design, and we have presented the changes in the hardware and software that has been necessary to make it fully functional, including the operating system interfacing.

The main conclusion to highlight from this work is that, indeed, it is feasible to use open designs to implement industrial systems with the advantages that comes from the standpoint of the cost of development and extended functionality without prejudice to the performance.

Acknowledgment

This work has been partially supported by the Ministry of Education and Culture of the Spanish Government through the TEC2007-61802/MIC (HIPER) project and the Andalusian Regional Government's EXC-2005-TIC-1023 project.

References

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

[2] “IBM PowerPC Quick Reference Guide”, IBM Corp. 2005.

[3] “NIOS 3.0 CPU Data Sheet”, Altera Corporation, 2004.

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

[5] Jiri Gaisler, Sandi Habinc, Edvin Catovic: “GRLIB IP Library User's Manual”, Gaisler Research, 2006, http://www.gaisler.com/products/grlib/grlib.pdf

[6] Damjan Lampret: “OpenRISC 1200 IP Core Specification” , 2001, http://www.OpenCores.org/cvsget.cgi/or1k/or1200/ doc/or1200_spec.pdf

[7] Telvent E.M.A.S.A., http://www.telvent.com/

[8] Guadaltel S.A., http://www.guadaltel.com/

[9] Debian GNU/Linux distribution, http://www.debian.org/ intro/about

[10] A. Muñoz, E. Ostua, etc al.: “Un ejemplo de implantación de una distribución Linux en un SoC basado en hardware libre”, III JCRA Workshop on Reconfigurable Computing and Applications, pp. 85-92, Zaragoza (Spain), 2007.

[11] OpenCores, http://www.OpenCores.org/

[12] “The SPARC Architecture Manual, Version 8”, SPARC International Inc., 1992.

[13] Graham Seaman: “Free Hardware: Past, Present & Future”, Erste Oekonux Konferenz, 2002.

[14] John L. Hennessy, David A. Patterson: “Computer Architecture: A Quantitative Approach”, Morgan Kaufmann Publishers, Inc., 1990

[15] “AMBATM Specification (Rev 2.0)”. ARM Ltd corporation,1999. http://www.arm.com/products/ solutions/AMBA_Spec.html

[16] “WISHBONE SoC Architecture Specification, Rev. B.3” OpenCores Organization, http://www.OpenCores.org