14

Open Development Platform for Embedded Systems

E. Ostúa, A. Muñoz, P. Ruiz-de-Clavijo, M.J. Bellido,
D. Guerrero and A. Millán
Grupo de Investigación y Desarrollo Digital (ID2)
Dpto. Tecnología Electrónica, University of Seville
Spain

1. Introduction

Embedded systems are those that implement a specific function while satisfying a number of restrictions. These restrictions have evolved over time and today, the most common for this type of system are cost and low consumption, and minimum size and weight. Moreover, the demand for embedded systems for different functions, and also the complexity of these functions has increased significantly in recent years. Also, due to the high level of competition, is the need to significantly reduce development times of these systems (Sangiovanni-Vincentelli, 2001).

All these problems in the design of embedded systems have been giving rise to an evolution in design methodology, so it has passed from implementing them as an application specific hardware, to a methodology whose main part is software design.

This change in methodology has been possible thanks to advances in technology that allow a complete system to be included in a single chip (ie, System on Chip, SoC). The hardware architecture of Embedded systems in SoC technology consists of a microprocessor as a core, around which are the components or peripherals necessary to carry out the function of the system. This function is implemented through software running on the microprocessor (Beeckler, 2007; Atitallah, 2008).

For views of both cost reduction as well as development time, design methodologies of embedded systems have consisted of pre-built hardware platforms closed with a wide range of applications. This range of applications is achieved because these platforms have multiple ports that are able to package different types of signals. The most common platforms of this type are microcontrollers (MCUs) and digital signal processors (DSPs). With this type of component, the final application design consists primarily in the development of software running on them. In order to reduce development times as well as being able to use these platforms in highly complex functions, manufacturers have made an effort in software development environments (Salewski, 2005). Thus, in recent years, the new MCUs are characterized by supporting several operating system because they provide a convenient way for software applications development.

However, these type of closed platforms reduce the degree of freedom of the designer of embedded systems. This is so because the designer is not able to manipulate and adapt the

hardware to the specific application. This could lead to an improved final performance of the system.

An alternative to closed hardware platforms is the use of FPGAs (Ostua, 2008; Muñoz, 2008; Eastman, 2005). The high capacity of integration, low cost on short runs, and high performance in terms of operating frequency and power consumption in FPGAs makes it possible to implement an entire microprocessor-based system in one of these programmable devices. This type of FPGA-based hardware platform has several advantages over closed platforms:

- SoC design adapted to the specific needs of embedded systems
- Ability to design and adapt the hardware components to specific application functionality, and include them in the SoC
- Dynamic reconfiguration capability of the SoC (Williams, 2004)

However, the use of FPGAs necessarily implies an effort to design the hardware architecture that makes the total development time of the system can be significantly higher than in the case of closed platforms. To make the design of embedded systems using a FPGA as central core of the system feasible, CAD tools are necessary to facilitate the construction of the hardware architecture quickly and efficiently. Indeed, nowadays, manufacturers are leveraging the SoC design tools on FPGAs and there is a level of competition for the best solution that combines both ease of construction of the complete architecture as the best performance of the final system implemented. Examples of these tools are, for example Xilinx EDK (Xilinx Inc., 2009), Altera ESD (Altera Inc., 2009), etc.

But in addition to facilitating the development of the hardware architecture it is also necessary to facilitate software development. This is the point where there is greater difference between the MCUs and FPGAs. The new MCUs are characterized by supporting operating systems. The introduction of the operating system level in an embedded system allows these to be used in applications of high complexity. In contrast, microprocessors used in FPGAs, are only now beginning to support operating systems (Henkel, 2004) such as the case with petalinux MicroBlaze (Petalogix Inc., 2010), or projects to port Linux to Altera's Nios (Altera Inc., 2010) or Lattice Mico32 (Lattice Inc., 2010).

In this chapter, we present a platform hw / sw open in the sense that it is technologically independent, and that it is implementable in low-cost FPGAs that meets both of the main features mentioned above: CAD tools that facilitate the development of SoC hardware architecture and, above all, a software development methodology comparable to the methods used in standard PCs.

The platform is based on the LEON3 processor (Gaisler, 2010) that implements the SPARC v8 architecture. The main advantage is that on this microprocessor a Debian Linux distribution (Debian, 2010) was implemented. Having not only a Linux operating system, but of all the components of a very mature distribution as Debian, gives rise to a great potential and easiness when developing software applications for embedded systems.

In what follows we describe the platform, starting with the hardware architecture. Then we describe the implementation process of the Debian distribution. In the 4th section, we detail a general development methodology for any application based on this platform. We show the advantages of the platform on a specific example of embedded system consisting of a terminal unit for location and positioning. Finally, are summarized the main points of work.

2. SoC hardware level

As we mentioned before, we chose LEON3 synthesizable microprocessor, that implements a SPARC architecture. The main features of this microprocessor are described as follows:

- The LEON3 32-bit core implements the full SPARC V8 standard
- It uses big-endian byte ordering and has 32- bit internal registers
- It has 72 instructions in 3 different formats and 3 different instruction addressing modes (immediate, displacement and indexed)
- Implements signed and unsigned multiply, divide and MAC operations and has a 7-stage pipeline instructions
- Also implements Harvard Architecture with two separate instruction and data cache interfaces

With final purpose in mind, Debian was chosen as the Linux distribution, since it has been adapted and compiled for many architectures such as x86 or SPARC v8 among other features. The microprocessor we want to use, must have the ability to run any of these sets of instructions and should be capable of running a Linux kernel 2.6 version. For this purpose, it requires some extra functional blocks, such as a memory management unit (MMU) and a floating point unit (FPU), and should be included within the microprocessor.

Moreover, the whole system must provide a minimum number of other functional blocks that enable a proper operating system support. These blocks are an interrupt controller and a timers unit. On the other hand, support for system memory and an alternative to connecting a mass storage medium should be included. The developed SoC includes a mixed memory control block, which allows access to both FLASH memory and SDRAM memory. It's memory map is assigned as follows: the lowest memory addresses point to the non-volatile memory, while the the rest of addresses cover the dynamic memory. In this way, the start of FLASH memory match the microprocessor boot vector, allowing one to store and run a boot loader in these positions.

In order to provide disk space for the operating system, the inclusion of an IDE interface controller was a good option. This controller has been connected to a Compact Flash card and, in addition, this medium is also used to provide swap memory to the system.

Finally, it is necessary to add some peripheral blocks for communication with the outside of the SoC. It is interesting to provide some kind of Internet access to fully exploit the advantages that the operating system as a development platform provides. Therefore, we have implemented both access methods: modem and Ethernet interfaces. It also includes other general-purpose and debug ports, such an UART blocks that provide RS232 communication.

In summary, figure 1 shows the block-level diagram of the system, where all functional blocks that have been included can be appreciated. Its interconnections has been made through open AMBA 2.0 Bus specification.

Most building blocks of this design are part of a set of libraries and IP cores (including LEON3 microprocessor) called Grlib (Gaisler, 2010). However, it is relatively easy to add new logic cores to the system. For example, a 16550 UART core have been added to support a modem GSM / GPRS communications. This core, wich is based on a model written in Verilog

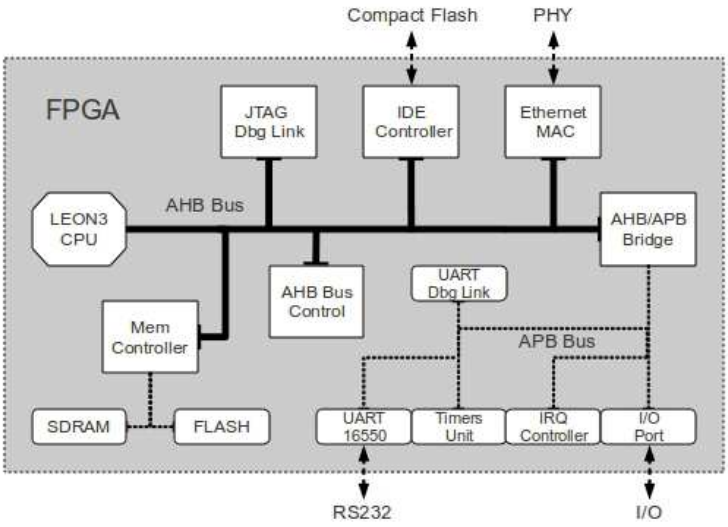


Fig. 1. SoC block-level diagram

language, is obtained from Opencores (OpenCores, 2010). It was necessary do some work to adapt its native intefarce to function properly with the AMBA APB bus.

The synthesis of this design can be accomplished with many software packages and may use different FPGA families and even ASIC as target.

Grlib includes some shell scripts for Linux that greatly facilitate this task. In this chapter, the Xilinx ISE package (Xilinx Inc, 2010) has been used, obtaining the results of compilation and synthesis for Spartan3 XC1500 shown at table 1.

These results are heavily dependent on the choices made when configuring the microprocessor. For example, the size of data and instructions caches, and the number and complexity of the peripherals that have been added to the system, will dramatically change these results.

3. SoC platform operating system

The main objective of this chapter is aimed at obtaining a development platform similar to those available on conventional workstations, but this one will run on top of the SoC platform itself. To accomplish this, and based on the exclusive use of open software, we will install a complete Linux distribution, allowing the use of all development tools, compilers and libraries and also take advantage of using open source code without restrictions.

We can divide our campaign into two main goals, as the SoC platform should first be able to boot a Linux kernel before a complete Linux distribution can be installed. The LEON3 core we

Device Utilization Summary		
Number of BUFGMUXs	4 out of 8	50%
Number of DCMs	2 out of 4	50%
Number of LOCed DCMs	2 out of 2	100%
Number of External IOBs	221 out of 333	66%
Number of LOCed IOBs	192 out of 221	86%
Number of MULT18X18s	1 out of 32	3%
Number of RAMB16s	14 out of 32	43%
Number of Slices	12809 out of 13312	96%
Number of SLICEMs	391 out of 6656	5%
Overall effort level (-ol):	High	
Router effort level (-rl):	High	
Timing summary		
Timing errors:	0	
Score:	0	
Constraints cover	21241138 paths, 0 nets and 106771 connections	
Design statistics		
Minimum period:	24.822ns	
Maximum frequency:	40.287MHz	
Minimum input required time before clock:	6.591ns	
Minimum output required time after clock:	12.193ns	

Table 1. Xilinx ISE compilation and synthesis

planned to synthesize meets all the requirements for the implementation of a modern Linux kernel 2.6, while a few adaptations are necessary to meet the particularities of its architecture.

3.1 SnapGear & boot-loader

Snapgear (SnapGear, 2010) is an open source specific Linux distribution for embedded systems. A fork of the main distribution was adapted by Gaisler Research for LEON3 systems, which have included various kernel patches and a few basic device drivers.

This software package also includes a fundamental and indispensable element in order to load the operating system, which is the boot loader. This small piece of software is usually stored at the beginning of the FLASH memory which must correspond to the memory address $0, so it's then executed by LEON3 processor on the startup process.

The first function of the boot loader is to initialize the basic hardware system, such as debugging console or memory chips. Then it proceeds to uncompress the software package you want to run to the RAM memory system, both a 2.6 Linux kernel and a small romfs filesystem, both Gzipped. On this read-only file system the Linux kernel mounts the root system, which includes all utilities and applications that we have decided to include in the software applications compilation.

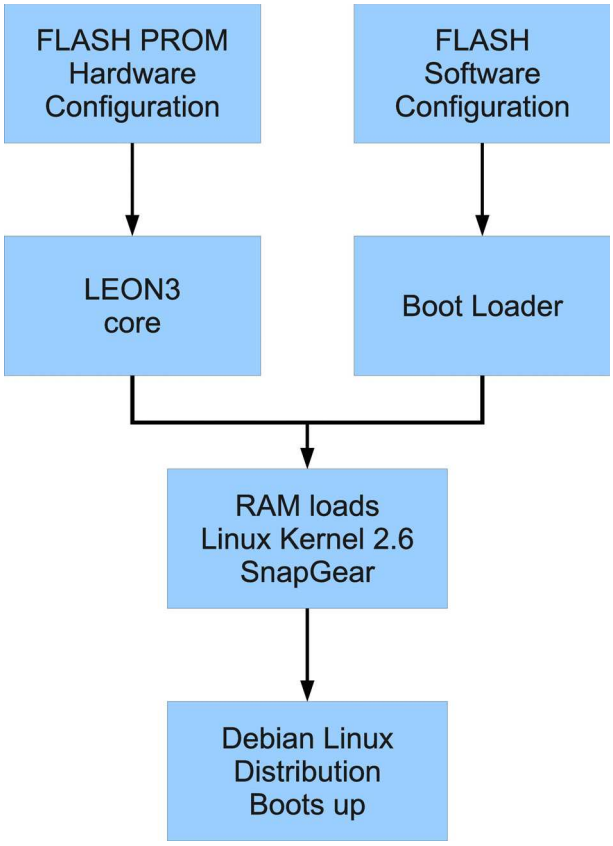


Fig. 2. SoC Boot Up

Another important ability of this small piece of software (boot loader) is to pass arguments to the Linux kernel, which avoids the need to use complex bootloaders such as LILO or GRUB, which are necessary in workstations.

The configuration of the Linux kernel and boot loader and the preparation of the romfs unit are performed by a few scripts. After this process, we proceed to compile the software system and so a set of software images (object files) are obtained.

With this set of object files generated we chose an image prepared for flash memory that contains the boot loader with the Linux kernel and the small romfs file system. Then we can program the flash memory using the Debug-Support Unit (DSU) within LEON3 core by interfacing with a JTAG or Ethernet connection, using the debugging application grmon from Gaisler Research.

Once the process is completed we have a functional Linux kernel with very basic core lightweight applications on top of our SoC platform. The complete programming and booting process of the SoC platform is shown in figure 2.

3.2 Debian

Once Snapgear Linux operating system is running on the system it's time to implement a complete Debian distribution, obtained directly from an official Debian repository. This distribution is precompiled for many platforms, including SPARC V8, which makes it mandatory to usage of Integer Multiplier (MUL) and a Floating Point Unit (FPU) hardware cores, which LEON3 provides.

There are a few ways to install this operating system, however we have chosen the defacto approach to use a small application called debootstrap, which takes care of the download, unzip and install all necessary packages for our target platform. A file system was created that includes such romfs application and its dependencies to other libraries and applications (such as wget and binutils) and they were integrated into the image we created for Snapgear in the previous step.

After the inclusion of the new file system in the romfs Snapgear compilation, it is possible to boot the Linux system and to prepare the installation process of the Debian distribution. In order to do so, having initialized the system and using the Linux shell, it proceeds to boot and configure the Ethernet interface to access the Internet and a DNS server.

Then comes the setup for the drive where you want to install the operating system, which in this case it's a portable solid state memory Compact Flash Card connected through an ATA Controller IP core, but also could have opted for a remote unit NFS hard disk drive IDE interface as a network booting solution.

It is desirable to have at least 1.5 GBytes free on the device to allow easy installation of those software packages and also in order to make use of swap space. Next, two partitions are created in the disk, one for the ext2 filesystem and one swap, with the commands fdisk, mke2fs and mkswap, which are then assembled and activated by the commands mount and swapon.

We must note that romfs is a read-only filesystem, so it has been necessary to have an empty path to mount the unit, in our case we have used the path /mnt/debinst. In this state, the system is ready to install Debian Operating System Cross. Simply run the setup application debootstrap like this:

```
# /usr/sbin/debootstrap –arch sparc stable /mnt/debinst
http://ftp.es.debian.org/debian
```

This command starts downloading packages from the Internet repository. These software packages are stored in the mass storage unit, to be validated and then installed in the system.

Once the process gets the basic installation of Debian on the drive, then it's necessary to configure certain aspects before its first use. At this point the use of chroot command proved quite useful to facilitate the entire process of design and preparation. This command allows the change of the root environment file system and keep running applications of the new distribution in the path that has been installed, allowing one to continue the setup process without restarting the whole system.

Firstly, the process was necessary to explicitly indicate the operating system to invoke a command line (shell) when booting the system (init process). To achieve this we simply

modified the description file load during the boot process of the operating system, the file /etc/inittab, by adding the following line:

```
T0: 234: respawn:/sbin/getty –L ttyS0 38400 vt100
```

Thus, the getty application launched a shell connected to the serial port every time the system restarts.

Secondly, the network interface and the name servers (DNS) have to be configured in order to introduce a suitable network configuration, adjusting the files /etc/network/interfaces and /etc/resolv.conf . Also it's need to tell where the mount points of the units will be, such as the boot, root and swap file systems. Like in other standard distributions, one just has to edit the configuration file /etc/fstab.

Finally the /dev directory is empty in the new distribution installation and so it was necessary to create the block and character devices in order to let the kernel manage the peripherals, a task done using the mknod command.

Once the installation and configuration of the Debian Linux distribution is complete, it's time to recompile the kernel adapted to the LEON3 microprocessor. In this new compilation the romfs partition is removed and the kernel will take the parameters where to locate the actual root file system (the drives) by modifying the boot loader configuration.

This argument was necessary to provide the kernel to configure the system console through the serial port and to tell the kernel where the root filesystem was:

```
# console=ttyS0, 38400 root=/dev/hda1
```

Finally started, the system with the new kernel and the Debian Linux installation that, after loading, presented a console with all the benefits of this operating system.

Importantly, once complete the whole process of installation and configuration, has the tools Debian APT package installation, such as apt-get, which allow you to install many libraries and applications, such as compilers, servers, internet services, daemons, desktop environments, etc., and even full distribution upgrades quickly and conveniently.

4. Developing methodology

The previous sections have been devoted to describing the basic technical characteristics of the platform proposed. This section, however, is devoted to discuss the development methodology of embedded systems based on this platform. The developing methodology of SOCs is divided two main parts: hardware developing and software developing. Although both are interlaced, they are very different. With regard to hardware development, begin identifying the components needed for the specific embedded system will be implemented. These components will be the interface of the SoC with the ports needed by the system.

Once identified, we must make the complete design of SoC (as shown in Figure 1) including all necessary components. As mentioned in section 2, this process will be done with GRLIB library that comes with the development system LEON3 processor. GRLIB includes a large number of components common in embedded systems. One of the advantages of this

platform is the possibility of designing a controller or peripheral specific for a particular application, which can be included in the SoC. It is also possible to reuse IP cores available for other systems. In these cases, it will make a design interface with the AMBA bus, which is the main bus that uses the hardware platform.

The most important improvement on this platform is under software developing since, the biggest effort in SOCs developing falls on the software and specifically in the interaction with peripherals. Our platform includes a operating system installed that solves the peripheral management through the device drivers. Even if a given driver is not avaliable for a new hardware, using Debian distribution we have tools to makes easy the developing. While with a conventional software developing methodology, some external tools are required: cross compilers, emulators, debug connection tools, etc., once installed the Debian distribution some advantages are achieved versus conventional software development methodologies of embedded systems.

We have achived significant improvements in the software tasks such as installing, compiling and developing, also, in the manage of devices drives or in high level applications. All this due some advantages of the using Debian as base software, it can be summarized as following:

- The Debian repositories has a large set of precompiled software that can be easily installed using the distribution tools. All is done inside of the platform without external tools.
- The distribution provides a the development environment and integrated debug tools that may be installed inside the platform. Also, without the need of external tools.
- Several improvements in the process of update firmware/software. The distribution adds tools to update all software installed inside in an automated mode.

Usually, is not an easy task to install any precompiled software in a SoC. Most medium-high complex software applications depend of other software as shared libraries or external tools. Software distributions give tools to solve applications dependences. Specifically, the main feature of Debian software distribution is the Debian package machinery. This resolves the software dependencies and creates a minimal configuration for each installed software. This minimal configuration allows the putting services on just after install processes and also, can be used as example of end software configuration.

Using precompiled software avoids the heavy task of compile software. Compile software for embeded systems is a heavy task due some of following reasons: one, is the dependence between applications and shared libraries making the compilation process complex. Usually one application needs an exact version of each required shared library. If there are several applications installed at the same time, the same library could appear several times, but with a different version number. In this situation install, upgrade and maintain the software is a tedious task.

Other reasons that make the task of compile software for SoCs complex is the use of cross compilers to generate software. Once compiled software, to test it, all necessary files must be transferred and installed: executable files, shared libraries, config files, etc. The steps of compiling and transference are repeated as many times as necessary to achieve the software to be ready. This process is improved when the Debian distribution is installed. It is possible to install all compiler tools in the platform and make the compilation process inside of platform. Also, this task is assisted by Debian tools since, it detects all dependencies between

applications, source code, and shared libraries. Debian tools leave the software developer environment ready inside of platform and the software compilations can be made easily. At this point the software can be developed inside of SoC, the software is also able to be debugged inside using the debug tools included.

Once the developing process has finished, this platform has some advantages during its lifetime. Having the ability of upgrade in the field is a common task today and is an advantage over closed systems. Upgrading the software or fix critical bugs are easy tasks becouse Debian community maintains its repository and, the software upgrades are automated when packages machinary is used. Other embedded SoCs require external tools to update the firmware, instead, this platform has the ability of autoupgrade if its necessary, eg. using network connection.

Another important aspect is the improvement in the process of developing drivers for this platform. By running 2.6 kernels it is possible to load devices drivers in the kernel under demand. With this, we obtain two advantages, one concerning to the resources used and the other, in the devices drivers developing process. Both are now explained in detail.

One of improvements is the capacity to add all avaliable device drivers with the kernel. Although most of the device drivers are not necessary at same time, the capabilities of the platform is not affected by including all. In fact, only the necessary device drivers are loaded into memory on demand, while the rest of them remain stored at disk. With this, the system maintains all functionality and, it saves resources for the rest of the tasks. On the other hand, new device drivers are developed without touching the binary kernel that is running into platform. Since the new drivers are loaded and unloaded dynamically, the driver developer can debug inside the platform. This is an advantage over other SOCs where, the kernel must be fully recompiled and loaded into the SOC for each change in the drivers thus, making the debugging process of drivers difficult.

5. Sample system implemented with this hardware platform

This section presents a specific development of a Remote Terminal Unit (RTU) which implements the standard IEC-60870-5 application-layer protocol for telecontrol messaging stacks. Both the hardware and the operating system are based on the platform already presented in this chapter and we discuss the main tasks required to customize our developments to fit the target system and also describe some of the advantages of using our embedded platform.

In a typical telecontrol scenario one station (primary station, called CC), controls the communication with other stations (secondary stations, also called RTUs), so IEC 60870-5 specification allows online telecontrol applications to take place. In this sense, the IEC defines a set of functions (profiles) that performs standard procedures for telecontrol system. A typical scenario for this kind of communication is shown in figure 3.

The transmission channels available within the design RTU are RF (Radio Frequency), GSM (Global System Mobile) and GPRS (General Packet Radio System). We had to provide the hardware platform with these devices and to integrate everything in the kernel drivers so they were addressable by the user level applications.

For RF transmission an ICOM IC-V82 (VHF transceiver) device was used, with the digital unit UT-118. This device is D-star capable, and can be connected to any RS232 device for data

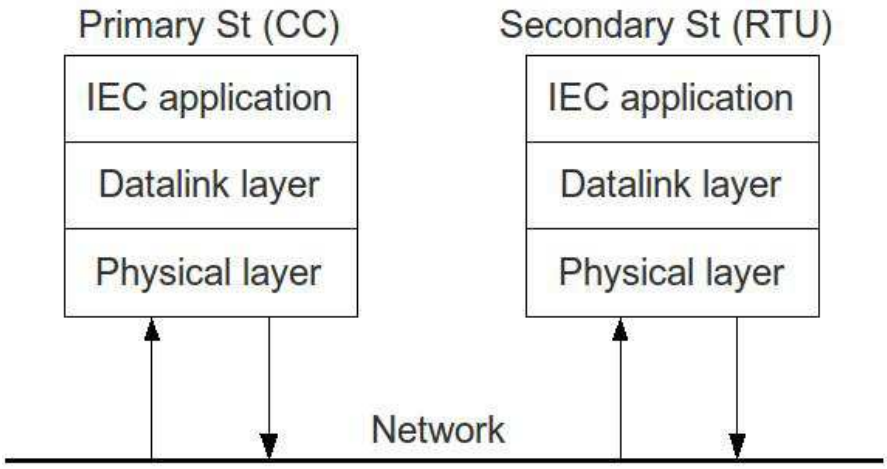


Fig. 3. IEC 60870-5 transmission scenario

transmission, with a data transmission speed of 1200bps. The GSM modem was a Wavecom Fastrack M1306B, which behaves as a standard AT-command modem via a RS232 port, so the modem is connected to the RTU this way. According to device specifications, it allows a data transmission of up to 14.400 bps for GSM but this feature depends on the GSM operator used, so it might not be available (in fact, our tests ran at 9600bps).

In order to get those devices into the design we needed some serial ports and so we integrated a fully capable UART 16550A compatible peripheral, which improves the default UART integrated with LEON microprocessor, by using all the transmit, receive & handshaking lines and also adding two onchip FIFO buffers for improved usability.

We started the development of the UART with one of the free IP cores from OpenCores community (written in Verilog) and we had to interface the Wishbone bus interconnection architecture to the AMBA-2.0 APB (ARM Corp., 1999) peripheral connection LEON manages (the one for relatively slow and simple devices like this). The main issue when interfacing those two protocols is that the AMBA-APB bus does not have a signal for acknowledgement, so every operation always has the same duration, while the Wishbone peripherals usually insert wait states in the communication to the bus master using that ACK line. Finally some Plug & Play information was added to the system so it enable the identification and allocation of the peripheral automatically on the LEON based system startup.

In figure 4 a picture of a prototype of the platform is shown, where you can see the GPS device, the GPRS modem fitted in a custom expansion card with some more serial ports, a compact flash card to hold the Linux local filesystems and the FPGA which holds the embedded design. The prototype has been implemented on the development board XC3S GR-1500. This board includes a Xilinx XC1500 FPGA Spartan3. Also, the board provide others features needed to run the system: 64Mbytes of SDRAM, 64 Mbit of Flash memory, Ethernet PHY transceivers R232, etc. In order to connect both the GPS and GSM modem and Compact Flash card has been necessary to develop adaptive PCBs as shown in figure 4.

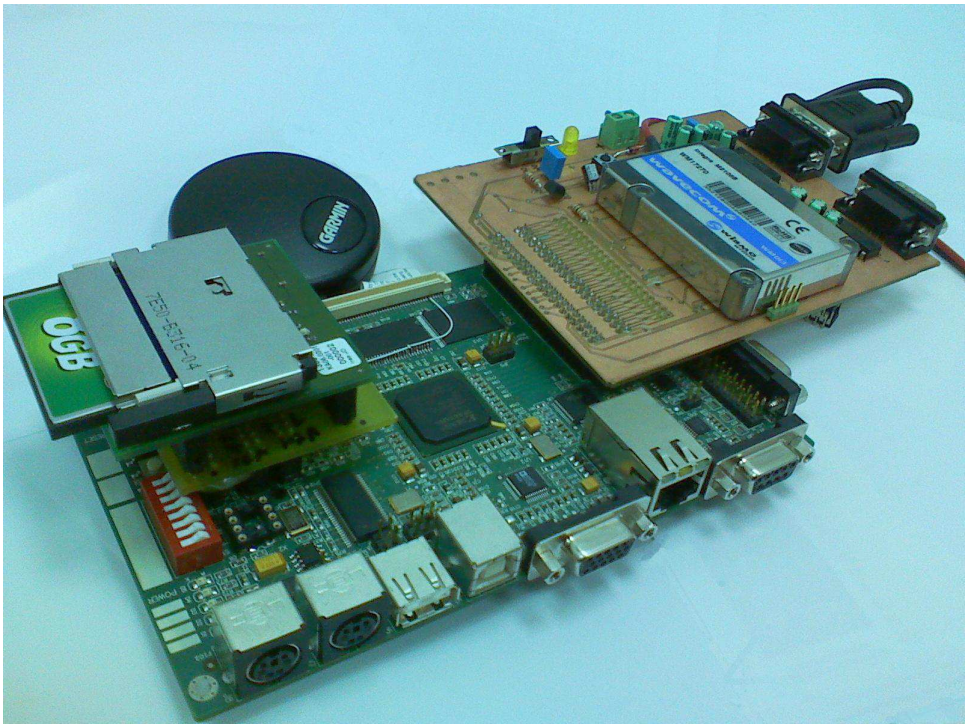


Fig. 4. Hardware Platform Prototype

Regarding the software development, there is no open source applications available for the IEC standard protocols, so both the data-link layer and the application layer have to be implemented from scratch. Anyway, as we're using Linux on our platform we already have a TCP/IP stack and standard development tools and libraries, so the data transport and other concrete IEC specifications were built.

The software project was divided into a few different modules, but the most interesting one is the Remote Module, (RTU module). Modules have been developed in C++ language, under a typical software development IDE on a hardware platform x86 (PC) running Linux operating system. Only standard libraries have been used in the development. After the initial setup and debugging the software toolkit also has been compiled for LEON (SPARC) architecture. As LEON has a standard Linux Debian distribution with standard C++ libraries, getting a binary file running is as simple as compiling the same source code developed in the PC in the embedded Linux by using the standard C compilers for the architecture. A GCC compiler has been used to compile and link the RTU source code inside LEON.

In order to test the full system, two parameters were analyzed, initialization time (IT), mean time required to complete the initialization on a RTU, and Poll Time (PT), mean time required to complete a poll over a RTU. Also two scenarios were tested. In one the CC and RTU were PCs and in the other the RTU was LEON. Full details on the application solution implemented and main results were presented in (Medina, 2009).

6. Conclusions

The breakthrough in the capabilities and performance of FPGAs has made becoming an alternative to the MCU in the design and implementation of embedded systems. The main differences between the two alternatives are in fact that the design with FPGAs need to design not only the SoC software but also hardware. This can be very advantageous in many cases because the hardware can be adapted to the specific needs of an embedded system. However, there are still significant differences in the software design methodology due to the development tools available to the MCU are far more advanced than the software development environment for microprocessors that are implemented in FPGAs.

In this chapter we present a platform hw / sw implementable on FPGA which greatly facilitates the process of software development. This platform has several significant advantages such as it is based on open components (LEON3 microprocessor and operating system Linux, Debian), tools that facilitate the development of hardware platform, etc.. But, fundamentally, the main advantage is based on having implemented the Linux operating system with the Debian distribution. This distribution is prepared to run on a standard PC. This means that all the software available for Debian is easily installed. It also has a large set of libraries for developing software applications that can be implemented and used in the embedded system that is based on this platform.

To demonstrate the efficiency of embedded systems development based on this platform we have designed a terminal of geolocation. Thus, it has completed the design of both hardware platform and software applications running on the terminal unit. Software development has demonstrated the advantage and versatility that means having both a large set of available libraries and development environments equivalent to those of a standard PC.

7. Acknowledgements

This work has been partially supported by the Ministerio de Economía y Competitividad of the Spanish Government through project HIPERSYS (TEC2011-27936) and the European Regional Development Fund (ERDF).

8. References

Sangiovanni-Vincentelli, A. and Martin, G. 2001. "Platform-Based Design and Software Design Methodology for Embedded Systems" *IEEE Design & Test of Computers*, 18, 6 (Nov. 2001), 23-33. DOI:10.1109/54.970421

Beeckler, J.S. , Gross, W.J., 2007. "A Methodology for Prototyping Flexible Embedded Systems" *CCECE: Canadian Conference on Electrical and Computer Engineering*, 2007. pp. 1679-1682

A. Ben Atitallah, P. Kadionik, N. Masmoudi, H. Levi. "FPGA implementation of a HW/SW platform for multimedia embedded systems" *Design Automation for Embedded Systems* (2008) 12: 293âĂŞ311, DOI 10.1007/s10617-008-9030-2

Salewski, F., Wilking, D., and Kowalewski, S. 2005. "Diverse hardware platforms in embedded systems lab courses: a way to teach the differences" *SIGBED* Rev. 2, 4 (Oct. 2005), 70-74. DOI:10.1145/1121812.1121825

E. Ostua, J. Viejo, M. J. Bellido, A. Millan, J. Juan, A. Muñoz, 2008, "Digital Data Processing Peripheral Design for an Embedded Application Based on the Microblaze Soft Core",

4th Southern Conference on Programmable Logic (SPL 2008) ; San Carlos de Bariloche (Argentina), 2008

A. Muñoz, E. Ostua, M. J. Bellido, A. Millan, J. Juan, D. Guerrero, 2008, "Building a SoC for industrial applications based on LEON microprocessor and a GNU/Linux distribution", *IEEE International Symposium on Industrial Electronics (ISIE 2008)* pp. 1727-1732, Cambridge (United Kingdom)

J. Williams and N. Bergmann, "Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip" *Proc. Eng. Reconfig. Syst. Algorithms (ERSA)*, Jun. 2004, pp. 171-176

Salewski, F. and Kowalewski, S. 2007. "Hardware platform design decisions in embedded systems: a systematic teaching approach" *SIGBED* Rev. 4, 1 (Jan. 2007), 27-35. DOI:10.1145/1217809.1217814

Nancy Eastman. 2005. "Moving Embedded Systems onto FPGAs" *00 Embedded Magazine* http://china.xilinx.com/publications/magazines/emb_02/xc_pdf/emb02-altium.pdf

Xilinx Inc. (2009) "EDK Concepts, Tools, and Techniques" http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/edk_ctt.pdf

Altera Inc. (2009) "Nios II Processor Reference Handbook" http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Joachim Henkel, Mark Tins (2004) "Munich/MIT Survey: Development of Embedded Linux" http://pascal.case.unibz.it/retrieve/1535/MunichMIT-Survey_Embedded_Linux_2004.pdf

Petalogix Inc. (2010), "Petalinux User Guide" http://www.petalogix.com/resources/documentation/petalinux/userguide

Altera Inc. (2010) "Nios Forum: Operative Systems" http://www.alteraforum.com/forum/forumdisplay.php?f=38

Lattice Inc. (2010) "uCLinux for LatticeMico32" http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/mico32uclinux.cfm

Jiri Gaisler, Sandi Habinc, 2010 "GRLIB IP Library User's Manual"

Debian (2010) "Debian SPARC Port" http://www.debian.org/ports/sparc/index.en.html

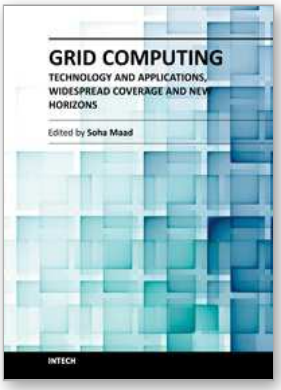
OpenCores.org, equivalent to ORSoC AB, all rights reserved. OpenCoresÂő, registered trademark http://www.OpenCores.org

Xilinx Inc. (2009) "ISE Design Suite Software Manuals and Help" http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/manuals.pdf

Snapgear 2010, "SnapGear Embedded Linux Distribution" http://www.snapgear.org/

ARM Ltd. Corp. 1999, "AMBA(TM) Specification (Rev 2.0)" http://www.arm.com/products/solutions/AMBA_Spec.html

V. Medina, I. Gomez, E. Dorronzoro, D. Oviedo, S. Martin, J. Benjumea, G. Sanchez, 2009, "IEC-60870-5 application layer for an Open and Flexible Remote Unit" *IEEE International Symposium on Industrial Electronics (ISIE 2009)* ; Seoul Olympic Parktel, Seoul, Korea July 5-8, 2009



Grid Computing - Technology and Applications, Widespread Coverage and New Horizons

Edited by Dr. Soha Maad

ISBN 978-953-51-0604-3
Hard cover, 354 pages
Publisher InTech
Published online 16, May, 2012
Published in print edition May, 2012

Grid research, rooted in distributed and high performance computing, started in mid-to-late 1990s. Soon afterwards, national and international research and development authorities realized the importance of the Grid and gave it a primary position on their research and development agenda. The Grid evolved from tackling data and compute-intensive problems, to addressing global-scale scientific projects, connecting businesses across the supply chain, and becoming a World Wide Grid integrated in our daily routine activities. This book tells the story of great potential, continued strength, and widespread international penetration of Grid computing. It overviews latest advances in the field and traces the evolution of selected Grid applications. The book highlights the international widespread coverage and unveils the future potential of the Grid.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821