Opencores based Embedded System on Chip for Network Applications

F. Abid, N. Izeboudjen, L.Sahli, D. Lazib, S.Titri, F. Louiz, M.Bakiri
Centre de Développement des Technologies Avancées
Lotissement 20 Août 1956, Baba Hassan, Alger
Email: fabid@cdta.dz, abidfaroudja@yahoo.fr

Abstract**—** The System-on-Chip (SoC) refers to a mini computer independent system where all essential parts of computing are integrated into a single FPGA or ASIC chip and where the application is executed by a program, which is loaded into an on chip memory or an off shelf component. However the increasing complexity of embedded systems and the staggering costs associated with designing systems-on-chip imposes system designer and companies to seek collaboration on a variety of intellectual property issues. Consequently the cost of building SoC is increasing significantly when the system integrates several parts. This paper presents Hardware/Software development of an embedded system on chip for network application, based on the Opencores and Opensources design concepts, in order to build an embedded system-on-chip for network applications at free cost. This approach is based on the IP (Intellectual Property) reuse strategy which facilitates the rapid creation and verification design process. In this paper we define the methodology adopted to construct our SoC. The system includes a hardware part and a software part which are linked to each other through a µClinux operating system. For the hardware part, an HDL file describing all the cores of the library is created. The SoC architecture is mapped into the virtex5 *XC5VLX50-1FF676* FPGA development board. Results show that the SoC architecture occupies 27% of logic resources and 35% of IOBs (In/Out Blocs). The software development of the embedded network application includes two parts: Configuration and compilation of the µClinux and programming of network application. In this part we have chosen an embedded network TFTP (trivial file transfer protocol) client as a test application. The results of the software part shows that the boot of µClinux and TFTP client test under the Or1ksim which is an instruction set simulator as well as the frame transfers were successfully done.

Keywords**—**Embedded system, MAC/Ethernet, System on Chip (SoC), OpenCores, OpenRISC, Opensource, TFTP.

I. INTRODUCTION

AN embedded system on chip (SoC) refers to a mini computer independent system where all essential parts of computing are integrated into a single FPGA or ASIC chip and where the application is executed by a program, which is loaded into an on chip memory or an off shelf component. The embedded SoC solution aims to realize portable systems for reducing power dissipation, chip interconnects and device size. Nowadays with the advance of the microelectronic technology, it is possible to integrate a whole system into a single FPGA circuit. Thus, a new field which integrates a network application into system on chip based FPGA circuit is emerging. This paper describes the embedded system on chip for network application which is part of the SoC platform based on Opencores and Opensources design concepts for Voice over Internet Protocol (VoIP) application [1], [2]. The benefit of using such methodology is the flexibility; reuse and accessibility of the IP (intellectual property) cores at free cost. Thus the cost of the whole VoIP system is significantly reduced. One of the main components in a network application hardware solution is the MAC/Ethernet circuit (Media Access Controller) [3] which is used to guarantee an internet connection. In traditional solutions, a PCI network card is inserted inside a computer which is based on a general purpose microprocessor. In this situation, the network application competes equally in processing time with other applications causing an overload in the processing. In order to overcome this problem, solutions implemented in dedicated hardware, ASICs or FPGA are available [4], [5] and [6]. These solutions allow that part of the processing, instead of being realised by the microprocessor of general purpose, now can be executed separately by a dedicated hardware. We propose a solution which not only integrates the MAC/Ethernet hardware component into FPGA but also embeds the software of the network application into the system. A case study of TFTP protocol is taken as an example. In section II, the Open System Interconnection Reference Model (OSI) is presented. In section III, a presentation of the FPGA embedded design methodology is given. Section IV deals with the SoC hardware architecture and shows synthesis, hardware simulation and implementation results of the embedded SoC. The Software design is presented in section V. Finally a conclusion in section IV.

II. OSI MODEL

The Open Systems Interconnection (OSI) Model is an abstract description for layer communications and computer network protocol design. It was developed as part of the Open Systems Interconnection (OSI) initiative. Figure 1 shows the OSI and TCP/IP (Transmission Control Protocol/Internet Protocol) models. In its most basic form, OSI model divides network architecture into seven layers which, from top to bottom, are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layers. Regarding the TCP/IP model the network architecture is divided into five

layers: Application, Transport, Internet and Network Access Layer [4].

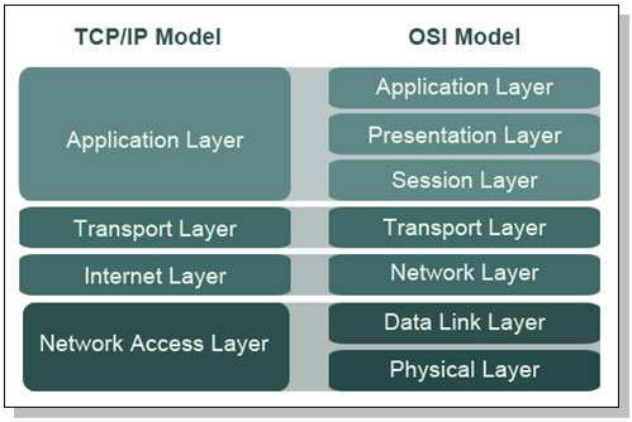

Fig. 1 OSI and TCP/IP Model.

Trivial File Transfer Protocol (TFTP) is a simple protocol to transfer files, with the functionality of a very basic form of File Transfer Protocol (FTP). FTP is a protocol of the application layer. The TFTP has been implemented on the top of the User Datagram Protocol (UDP). TFTP is designed to be small and easy to implement, it can be implemented using a very small amount of memory. It was therefore useful for booting computers such as routers which do not have any data storage devices. It is still used to transfer small amounts of data between hosts on a network, such as an operating system images. Therefore, TFTP only reads and writes files (or mail) from/to a remote server. It can not list directories, and currently has no provisions for user authentication. The process of the file transfer is illustrated in Figure 2.

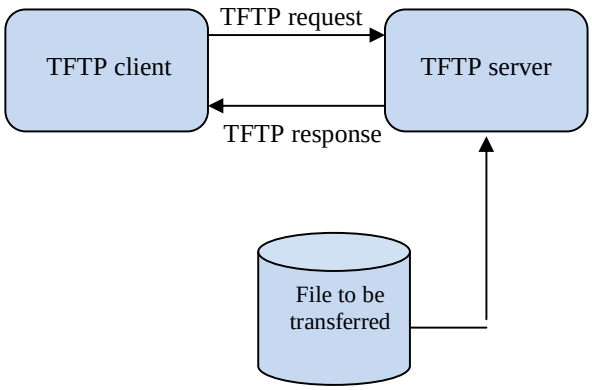

Fig. 2 File transfer process

III. FPGA EMBEDDED DESIGN METHODOLOGY

Figure 3 shows the general architecture of an embedded system. This one is composed of two parts. A hardware part which represents the system architecture which is mainly based on a processor and some peripherals components that communicate with each other through a suitable interconnect bus and a software part which is related to the application. The hardware part and software parts are linked to each other through the OS (operating system).A lot of approaches have emerged from industrial and academic research to design embedded systems into FPGA. Among these approaches, the Xilinx approach which uses the Microblase processor, the Altera approach which is based on the use of the Nios processor, the IBM approach which is based on the IBM processor and the Opencores approach which uses the OpenRisc processor. Each approach tries to promote its processor in the market [4].

From theses approaches we have chosen the Opencores approach.This choice is justified by the following points:

- Availability of the cores and tools at free cost.
- Register Transfer Level (RTL) descriptions are given for all the cores or IPs (Intellectual property) components so that the whole system can be mapped into FPGA or ASIC; this allow flexibility and reusability of the cores.
- We can make an embedded system design reference from scratch.

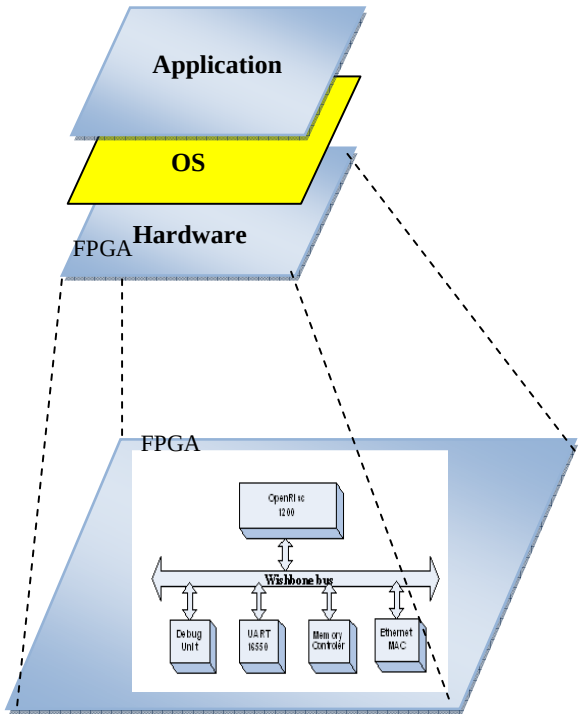

Fig.3 General view of an embedded system

Figure 4 gives an overview of our own platform created for the design, implementation and test of the embedded systems based on the OpenCores and Opensource design approaches. The hardware part is related to synthesis, place & roote, bitstream generation and downloading the generated file into FPGA circuit.

The software part contains a set of tools such as the Or1ksim simulator, a GCC compiler, a GNU debugger that are used to debug and load the application into an on chip FPGA memory or an external memory depending on the application size. For the hardware part, an HDL file describing all the cores of the library is created. . The cores communicate

through the wishbone bus interface. In order to automate the disgn steps, a make file is created for simulation and synthesis.

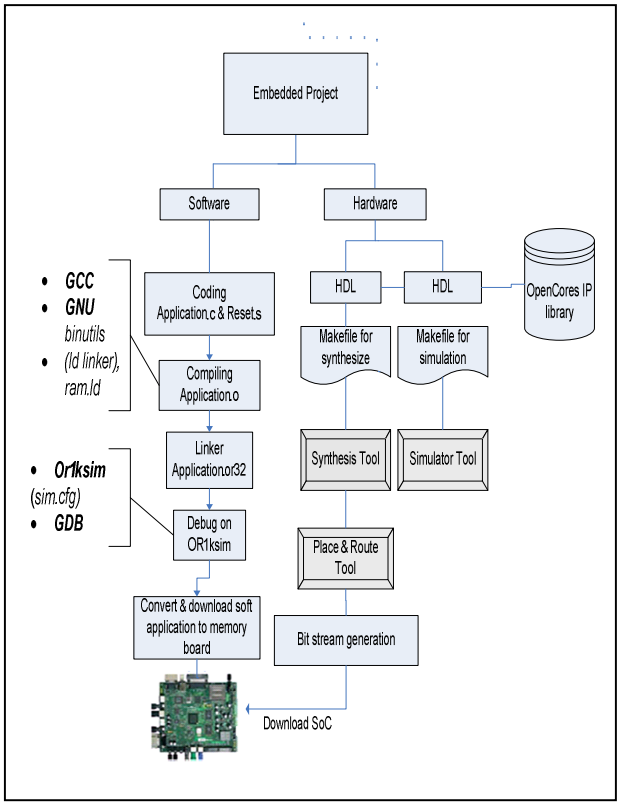

Fig. 4 Design methodology

VI. Hardware Architecture Presentation

Figure 5 shows the block diagram of the network SoC architecture. From the hardware point of view the embedded system on chip must be comprised of an embedded processor, embedded memory and required peripherals like Ethernet controller and input/output interface. In order to assure flexibility for different possible applications the embedded system was implemented on a FPGA platform. Soft-core processors like the Openrisc processor, Xilinx Microblaze, Altera Nios as well as any processor written in HDL languages can be implemented in FPGA devices. This avoids the need of a separate processor chip.

The soft-core processor configuration can be additionally tailored for specific application. Required peripheral devices are connected via wishbone bus. The embedded system must have sufficient memory to contain an operating system with a network application. In this work we have choosing µClinux the open source operating system.

We have developed an OpenRISC-based SoC platform that includes a 32bit Reduced Instruction Set Computer (RISC) processor [8]. The embedded network SoC includes the OR1200 core and a minimum set of elements needed to provide network functionality. These elements are the debug unit for debugging purpose, a memory controller that controls an external memory that carries µClinux and a network application, an Universal Asynchronous Receiver Transmitter (UART), the MAC/Ethernet that transmit voice packets over the Internet. The Internet connection is established by this IP, all the cores communicate through the wishbone bus. The most difficult task in designing SoC hardware is how to write an HDL code that integrates all the SoC components codes and how these components communicate each with other. We have set the OR1200 processor as MASTER for all components. The MAC/Ethernet and Memory controller are configured as slaves compared to the OR1200 and a MASTERs to the UART and debug unit. For integration of all the cores we have created a SoC Verilog description.

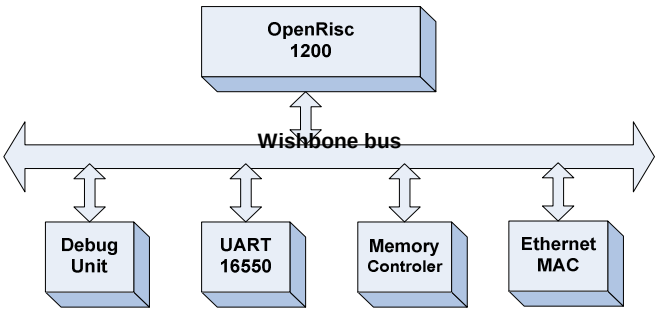

Fig. 5 Network SoC hardware architecture

VI. Presentation of OpenRISC Processor and the MAC/Ethernet

The Openrisc processor and the MAC/Ethernet cores are the basic IPs in our architecture, thus we give some details of their architecture.

A. OpenRISC processor

The OpenRISC 1200 is a synthesizable processor developed and managed by a team of developers at OpenCores [9]. OpenRISC 1200 is a 32-bit RISC processor implementing the 32-bit OpenRISC 1200 architecture. An overview of the OpenRISC 1200 processor architecture can be seen in Figure 6. The processor is intended for embedded, portable and network applications. OpenRISC 1200 is an open source IP-core freely available from the OpenCores website as a Verilog model, licensed under the GNU LGPL license.

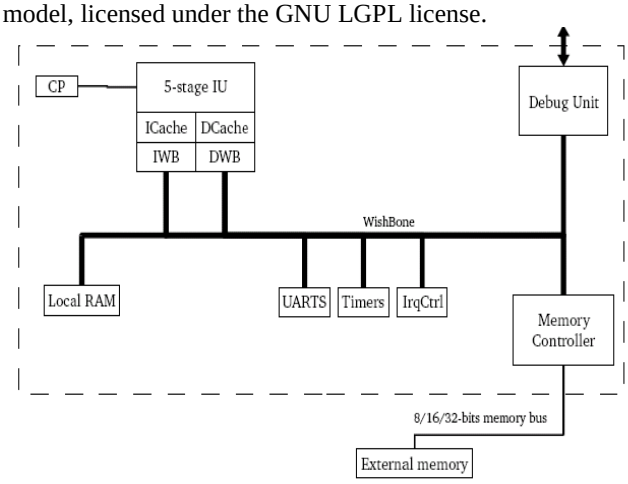

Fig. 6 OR1200 Architecture

B. *Ethernet/MAC*

The Ethernet core [10], [3] is a 10/100 Media Access Controller. It consists of synthesizable Verilog RTL core that provides all features necessary to implement the layer 2 protocol of the standard Ethernet. It is designed to run according to the IEEE 802.3 specification that defines the 10 Mbps and 100Mbps for Ethernet and Fast Ethernet applications respectively. In this work the Ethernet/MAC allows Internet connection.

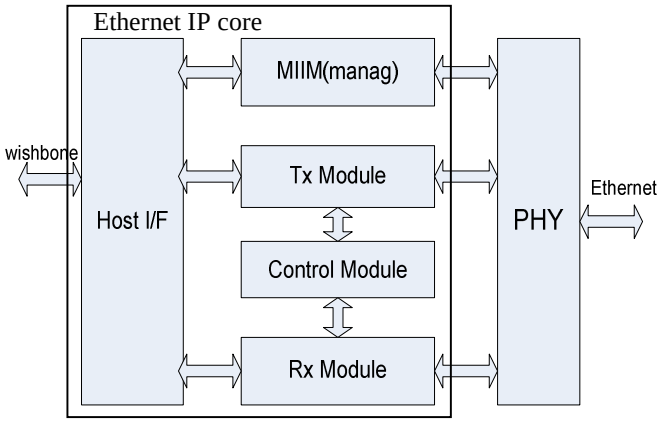

Fig. 7 MAC/Ethernet Architecture

Figure 7 shows the general architecture of the IP. It consists of several building blocks: a TX module an RX module, a control module, a management block and a WISHBONE interface. The TX and RX modules provide full transmission and reception functionality respectively. Cyclic Redundancy Check

(CRC) generators are incorporated in both modules for error detection purposes. The control module provides full duplex flow control. The management module provides the standard IEEE 802.3 Media Independent Interface (MII) that defines the connection between the PHY and the link layer.

Using this interface, the connected device can force PHY to run at 10 Mbps with frequency of 2.5 MHz versus 100 Mbps (25 MHz) or to configure it to run at full or half duplex mode. The wishbone interface connects the Ethernet core to the RISC processor and to external memory. To adapt this IP to our application we have configured it, then tested the transmit and receive process which are the two basic process in our application [3].

VI.2 SYNTHESIS AND SIMULATION RESULTS

A. *Hardware simulation results*

The simulation is done with Mentor Graphics simulator ModelSim. Figure 8 shows the MAC/Ethernet simulation.

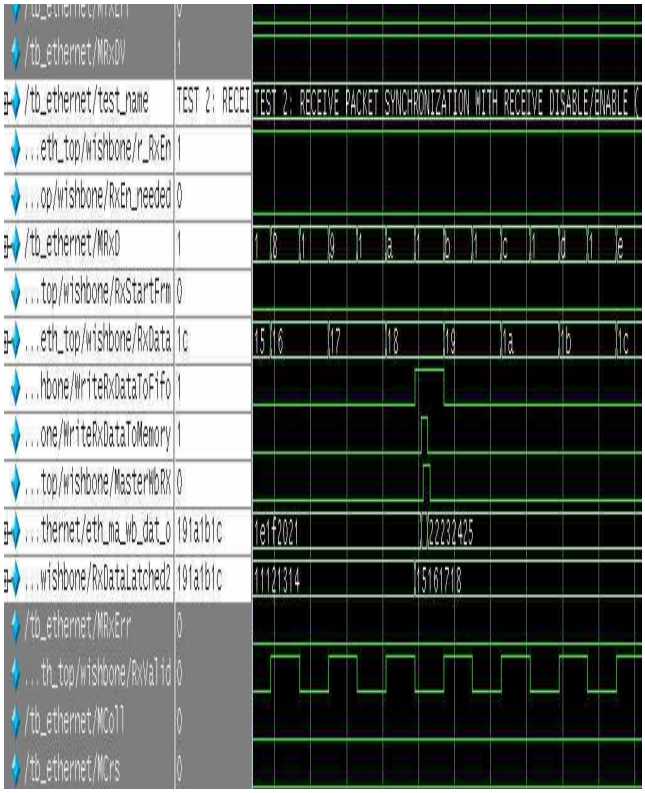

Fig. 8 MAC/Ethernet simulation results

B. *Synthesis results*

We have chosen the virtex5 *XC5VLX50-1FF676* FPGA FPGA as target to implement this system using the ISE 9.2i Xilinx [11] tool. As shown in table1 the network SoC architecture occupies 27 % of logic resources , 35 % of IOBs and 6% of DSP48Es. We synthesized the architecture using the XST (Xilinx Synthesis Tool).

TABLE 1
SYNTHESIS RESULTS

Number of Slices	5705 out of 14336	27%
Number of bonded IOBs	155 out of 484	35%
Number of BRAMs	11 out of 96	12%
Number of DSP48Es	5 out of 16	6%

The figure 9 shows the mapping of the architecture into a virtex5 FPGA using FPGA Editor tool [11].

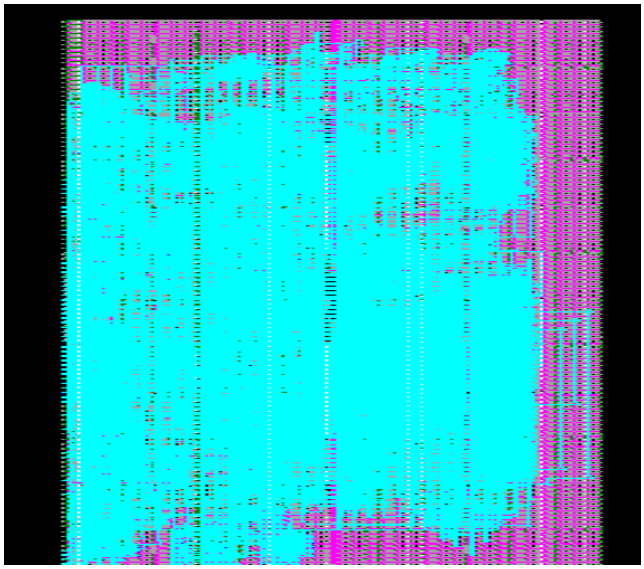

Fig.9 Mapping of the network SoC in Virtex5 FPGA

V. SOFTWARE DESIGN

In the software part, the GNU toolchains [2] are used to compile, link programs and generate the binary file of the application.

The first step consists on developing a C program "**application.c**". This file implements the main function needed for VoIP application. In the next step, the GCC tool is used to compile the program, in this phase the object file "**application.o**" is generated, in order to be used by the linker (*ld linker*). This file is linked with the linker file (*ram.ld*) which is used to map all the instructions, variables, data and stacks to the corresponding address in the memory.

The resulting binary file "**application.or32**" is used with the configuration file "*sim.cfg"* for the debug on Or1ksim step. In simulation step the "*board.h"* file which contains the hardware platform configurations is also used.

The "*sim.cfg"* file contains the default configurations of peripherals and a set of simulations environments which are similar to the actual hardware situation. In this phase the Or1ksim simulator and the GDB tool are invoked. Finally the binary file "**application.or32**" is converted to the memory initialisation file and downloaded into the SDRAM memory.

We envisage to run the network application on µClinux (microcontroller Linux version) operating system. The µClinux (Linux Kernel v2.0.38) is a port of Linux to systems without a Memory Management Unit (MMU) designed for small systems like microcontrollers and small microprocessors which are suitable for implementation in FPGA. The µClinux operating system and the network application were stored in the RAM memory. The software development of the embedded system includes two parts:

1. Configuration and compilation of µClinux operating system. In this part the µClinux kernel is adjusted to the target system and compiled.

2. Programming the network application.

A. µClinux Configuration

Mainly we have configurated µClinux to use the opencores MAC/Etehrnet core as a network controller and include de TCP/IP stack that provides the IP conection.

B. The network application

First we have tested the design through a frame transfer between FPGA board and PC under Linux. The serial port is used to visualise the frame transfer. The result of this test is shown in next section. A case study of TFTP protocol is taken as an example. In this test we have chosen the open source TFTP loader program with a lot of useful utilities and extra functions -- e.g. memory dump, flash loading. It is useful and fast to load in most applications. It is accessible using serial port or VGA terminal. First we have adjusted the "*board.h"* file according to our system platform. The "*board.h"* file contains the hardware platform configurations and the network configuration according to CDTA Local Area Network (LAN) configuration.

C. Software simulation results

At first we have run µClinux on Or1ksim simulator [12] we have configured Or1ksim to be used with µClinux via the configuration file "sim.cfg". Figure 10 shows the results of running µClinux on Or1ksim and the Internet protocols needed for a network application (ICMP (Internet Control Message Protocol) for ping, TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) under µClinux.

Fig. 10 Trace of booting µClinux on Or1ksim

Figure 11 shows the device network parameters set using TFTP loader program under Or1ksim simulator. We aim to test the file transfer protocol in the SoC using the 10/100 MAC/Ethernet network controller and µClinux operating system.

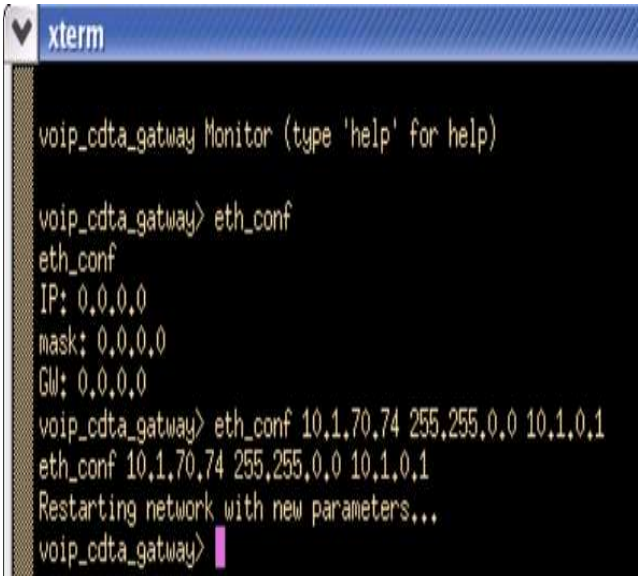

Fig. 11 Device network parameters set

Figure 12 shows the FPGA board-PC frame transfert test result.

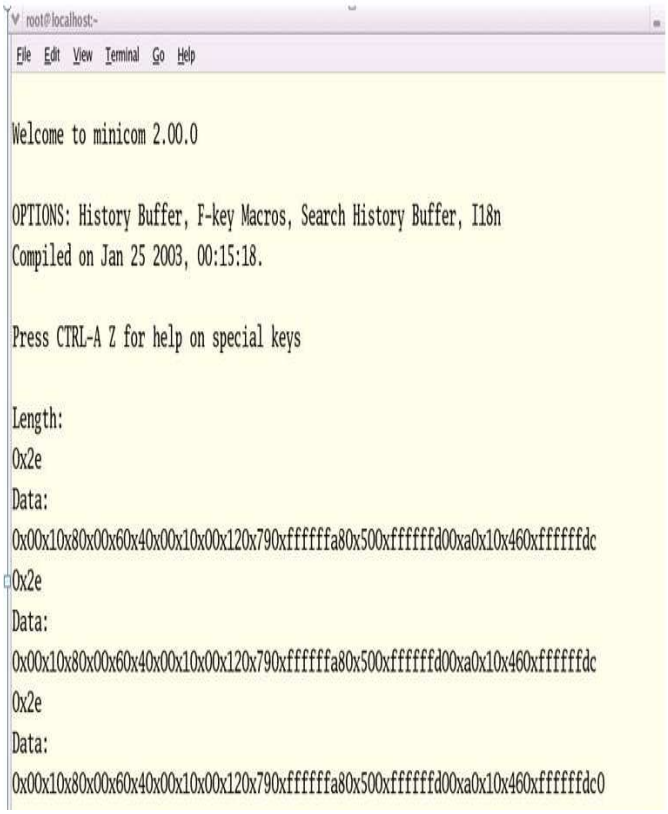

Fig. 12 FPGA board-PC Frame transfer test result

IV. CONCLUSION

In this work an embedded system for network application chip based opencores/opensources approach is developed. The system constitutes a solution for various network devices. The system incorporates software and hardware parts. To bluilt the Hrdware part, we have created a SoC Verilog description. The network SoC architecture is mapped into an *XC5VLX50-1FF676* FPGA development board. Synthesis results show that the SoC architecture occupies 27% of logic resources and 35% of IOBs (In/Out Blocs). Concerning the software part the network application based on the MAC/Ethernet and µClinux operating system is the first test application. The last one constitutes the main step in a VoIP application. Our next objective is to finalise the VoIP gateway integration in order to connect to the public phone system through a gateway and record and archive calls on a computer system. Then test the VoIP gateway performances in Internet Network Area.

REFERENCES

[1] S.Titri, N.Izeboudjen, L.Salhi, D.Lazib "OpenCores Based System on chip Platform for Telecommunication Applications: VOIP".DTIS'07,International Conference on design & Technology of Integrated Systems in Nanoscale Era, Sep. 2-5, 2007., Rabat (Morocco), pp. 253-256.

[2] Abid Faroudja, Nouma. Izeboudjen, Sabrina. Titri, Leila.Salhi, Fatiha.Louiz, Dalila.Lazib "Hardware /Software Development of System on Chip Platform for VoIP Application ", ICM, International Conference on Microelectronics, December 19-22, 2009, Marakech, Morocco, pp 62-65.

[3] F.Abid, N.Izeboudjen, L.Sahli, S.Titri, D.Lazib, F.Louiz "Integration of the Opencores' MAC/Ethernet in a VOIP based system on chip application", ESC, Embedded System Conference, Mai 5-6, 2009. Algers (Algeria).

[4] F.Abid, N.Izeboudjen, L.Sahli, S.Titri, D.Lazib, F.Louiz "Embedded Network SoC Application Based on the OpenRISC Soft Processor ", ICMOSS, Embedded System Conference, Mai 29-31, 2010. Tiaret (Algeria), pp 161-166

[5] F. L. Herrmann, G. Perin and al. "An UDP/IP Network Stack in FPGA",http://gmicro1.ct.ufsm.br/batista/images/stories/Artigos/Sforum_2009.pdf.

[6] K.Morita, K. Abe, "Implementation of UDP/IP Protocol on FPGA and its performance evaluation", IPSJ General Conference. Special5, Pages157-158.

[7] Xilinx, ML501 Evaluation Platform User Guide, version 1.4, August 24, 2009.

[8] Mjan Lampret "OpenRISC 1200 IP Core Specification", Rev. 0.7 Sep 6, 2001, http://www.opencores.org/projects/or1k/

[9] www.opencores.org

[10] Igor Mohor "Ethernet IP Core Specification ", Rev. 0.4 October 29, 2002.

[11] XILINX ISE 9.2 user manual. www.xilinx.com B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[12] Jeremy Bennett "Or1ksim User Guide", Embecosm, 2008.