Embedding Linux with Ability to Analyze Network Traffic on a Development Board based on FPGA

Andres M. Leiva-Cochachin
Instituto Nacional de Investigación y Capacitación de Telecomunicaciones de la Universidad Nacional de Ingeniería INICTEL-UNI, Lima

Fredy Chalco-Mendoza
Instituto Nacional de Investigación y Capacitación de Telecomunicaciones de la Universidad Nacional de Ingeniería INICTEL-UNI, Lima

ABSTRACT
In recent years, there has been an increased usage of embedded systems in different areas such as automation, telecommunications and medicine. These systems are widely used in these areas due to their portability, low cost and excellent performance. For embedded systems that require hardware and software integration, the system designer needs to know the general procedures that are necessary to reduce the development time. This paper provides the key guidelines for the entire design and implementation processes of an FPGA-based embedded system running Linux. The process of embedding a Linux operating system in a Xilinx ML505 development board is described in detail. Furthermore, as an example, it is presented the key guidelines for creating a basic network traffic capture application using the open source programming library libpcap.

General Terms
Embedded systems, Operating system, Networking.

Keywords
FPGA, Linux, cross-compilation, libpcap.

1. INTRODUCTION
The Field Programmable Gate Array (FPGA) devices enable the development of custom digital hardware. These devices are preferred by many embedded system designers since FPGAs have less development time than Application Specific Integrated Circuits (ASICs), are more economical, flexible, and are capable of concurrently execute multiple hardware tasks, offering higher performance compared to traditional single core microprocessors that execute the same tasks implemented in software. Using languages such as Very High Speed Integrated Circuit Hardware Description Language (VHDL) or Verilog, the system designer may be able to implement simple designs such as small combinational/sequential circuits and controllers, up to a complete System on a Programmable Chip (SoPC) which includes a microprocessor, embedded memory, and different peripherals [1]. FPGA vendors such as Xilinx, Altera, and Lattice, provide soft-core processors such as MicroBlaze, Nios II, and Mico 32, respectively, which are built using the programmable logic resources of a FPGA. These soft-core processors have the advantage of being flexible so each internal component, such as the Arithmetic Logic Unit (ALU), peripherals, memory address space and so on, are highly configurable [1].

In this work, it was exploited the aforementioned advantages of the FPGAs and was chosen the Xilinx MicroBlaze soft-core processor for the embedded system. The MicroBlaze was configured to support virtual addressing using a Memory Management Unit (MMU) in order to be able to run an operating system (OS). For this purpose, Linux was chosen as the OS. Additionally, Xilinx Intellectual Property (IP) cores are included for specific functionalities, as part of the entire embedded system.

Linux is an open source OS which can be supported by multiple processor architectures, either in 32 or 64 bits. Besides, the Linux OS can be ported into several hardcore or soft-core processors in FPGA devices. The Linux standard kernel for MicroBlaze was selected for the embedded system of this work. In this article, the process to embed the selected Linux OS in a Virtex-5 LX50T FPGA on a Xilinx ML505 development board is described. An example of an embedded Linux on a Xilinx FPGA to enable hardware multitasking is given in [2].

The necessity of solving many problems that occur in current high speed networks, forces the designers to use specialized systems that must have the ability of analyzing and processing data network packets quickly. On the other hand, a Linux kernel has a network subsystem which is responsible for collecting, identifying, and dispatching data packets that arrive in an asynchronous mode to the network interface of a given computing system. The operating system is in charge of delivering data packets across programs and network interfaces [3]. Specific application programs make use of a network interface to extract information from the data packets, such as the type of protocol, source and destination addresses, etc. Programs such as tcpdump, snort, and wireshark are based upon the popular open source programming library libpcap to provide a high level interface to packet capture [4]. In this work, it is used the libpcap library for programming a basic application tested in the target board, involving the MicroBlaze soft-core processor and the Linux OS.

In this paper, apart from describing the process for embedding the Linux OS on a FPGA-based system, it is exposed the procedure for testing a basic network sniffer in the embedded system. A customized Eclipse Integrated Development Environment (IDE) is used to enable the direct compilation of applications that are based on the libpcap library. The key steps related to this process are also outlined.

2. PROCEDURE FOR BUILDING THE EMEDDED SYSTEM
For building the entire embedded system, it is necessary four applications [5]: the Xilinx development tools, a C/C++ cross-compiler for MicroBlaze, the Linux kernel source and a root file system. The building process of the embedded system is divided in four tasks: Hardware Development, Device Tree Generation, Kernel Configuration and Compilation, and ACE File Generation. Some files are used as input and in turn each ask generate other files for the next tasks. The logical order and

the files that are employed in each task are portrayed in the scheme of the Fig. 1.

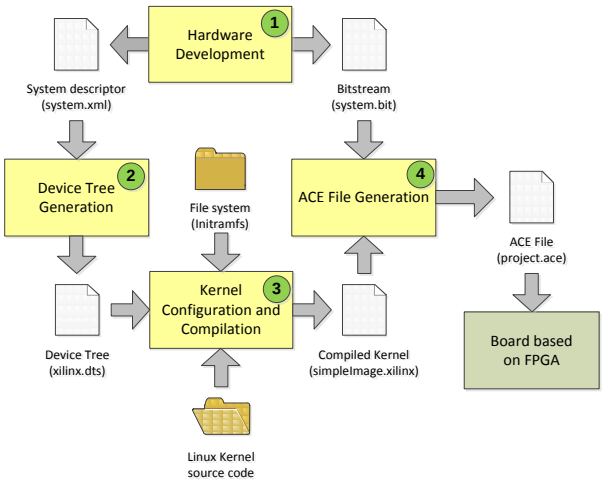

Fig. 1: Tasks and files involved in building the embedded system

The first task is the Hardware development where the structure of the hardware system inside the FPGA is designed. Then, a file that describes the hardware design is generated in the Device Tree Generation task. Afterwards, the respective Linux kernel image is built in the Kernel Configuration and Compilation task. In the end, an unique file is generated as from the hardware design and the Kernel image. This file configures the FPGA and executes the kernel in the board.

3. HARDWARE DEVELOPMENT

The Xilinx ML505 development board is isud which has a Virtex-5 LX50T FPGA device. Since this device does not include a hardcore processor such as PowerPC, a MicroBlaze soft-core processor is used instead. As a software tools, it Is used the Xilinx Integrated Software Environment (ISE) 12.1 and the Xilinx Embedded Development Kit (EDK) 12.1 tools which includes both the Xilinx Platform Studio (XPS) and the Software Development Kit (SDK) tools. ISE and EDK were installed in a PC workstation running Ubuntu 12.04 Linux distribution. Both Xilinx tools helped significantly during the design process. The EDK provided highly configurable hardware blocks called IP cores to implement different peripherals and interfaces connected to the MicroBlaze using the Processor Local Bus (PLB) as shown in Fig. 2. The ISE provided the programs for synthetizing and implementing the embedded system. ISE and EDK tools are configured using the default installation wizard and some environment variables were created and modified.

```
# xil_home=/opt/Xilinx/12.1/ISE_DS
# source $xil_home/ISE/settings32.sh
# source $xil_home/EDK/settings32.sh
# export PATH="$PATH:$xil_home/EDK/gnu/microblaze/lin/bin"
# export XILINX_EDK="$xil_home/EDK"
# export XILINX="$xil_home/ISE"
```

In order to automatically add the MicroBlaze processor as the main IP core, the Base System Builder (BSB) from Xilinx Platform Studio (XPS) is used. Likewise, other basic blocks such as the processor debugger, local memory, local memory controllers, system clock generator, and other peripherals were generated. The embedded system included the following hardware blocks (see Fig. 2):

- A controller for the RAM memory (mpmc).
- An interrupt controller (xps_intc).
- A timer block (xps_timer) with two timers. Each timer is configured to provide hardware interrupts.
- An UART interface (xps_uarlite) connected to a serial port and is configured to provide hardware interrupts.
- An Ethernet core 10/100/1000 Mbps (xps_ll_temac) with direct memory access DMA and configured to provide hardware interrupts.

For executing Linux OS seamlessly, the MicroBlaze processor must be configured properly. The MicroBlaze must be configured with a MMU to be able to provide virtual memory addressing using two memory protection zones [6].

After finishing the connection and parameterization of all components in the system as depicted in Fig. 2, the netlist is generated in XPS. Next, the design is implemented in ISE. Finally, the implemented design is converted to a bitstream that can be downloaded to the FPGA [7]. The bitstream is created as "system.bit" which is located within the directory "/implementation" which in turn is located inside the project directory.

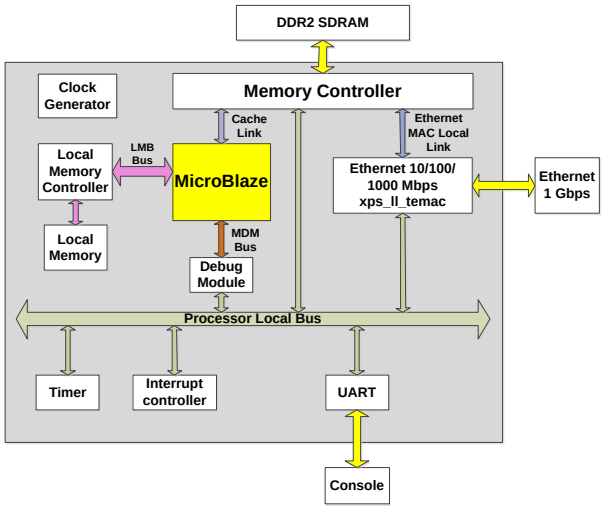

Fig. 2: Hardware blocks inside the Virtex-5 FGPA

4. DEVICE TREE GENERATION

By default, XPS does not have any option for using Linux as the OS in the development board that was used. Therefore, it is not possible to generate the Board Support Package directly. Nonetheless, the Linux kernel that is used is able to read data structure format called device tree [8] which describes the hardware system as generated in the XPS, such that the kernel can configure itself during the boot process [6].

The Xilinx Software Development Kit (SDK) is incorporated inside the EDK and it is used together with the device tree generator script for generating the device tree file. A hardware descriptor file in XML format called "system.xml" should be exported from XPS and imported to SDK to specify the particular hardware that is used. The device tree generator script should be added as a new repository in SDK. Next, it was necessary to configure the kernel boot arguments with the options that device tree offers. The serial interface (ttyUL0) is configured as the main console for the embedded system. Likewise, it is necessary to configure the place where the root file system is going to be stored and from where is going to boot. In this particular case, the root file system was located in

the partition 1 of the Compact Flash (CF) disk (/dev/xsysace/disc0/part1) [9] as shown in Fig. 3. Afterwards, the device tree file called "xilinx.dts" is automatically generated.

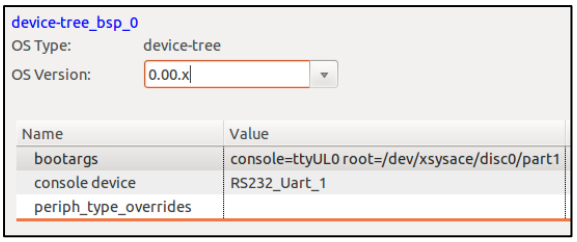

Fig. 3: Device tree configuration in SDK environment

5. KERNEL CONFIGURATION AND COMPILATION

This task is performed in a similar way for the creation of a Linux kernel using the PowerPC hardcore processor [6]. The Linux kernel should be configured according to the hardware configuration of the entire embedded system defined in XPS, including the MicroBlaze processor, the peripherals, and the connections of all modules. In this case, it was not used a bootloader to uncompress the kernel, since it was not use a compressed kernel. Besides, in this case, the operating system load is carried out before the development of the basic network application.

5.1 Installing the necessary tools

To facilitate the development, it was defined a directory denoted as "[tools_directory]" where the necessary software tools were stored. These tools were the cross-compilation tool chain (microblaze-gnu), the kernel source code (linux-xlnx) and the file system (inictramfs_minimal.cpio.gz). For using the cross-compilation tool chain from the command line, it is necessary to configure the environment variable PATH in the Ubuntu Linux of the PC workstation by adding the location of some directories where the binary executable files are. Also, the variable CROSS_COMPILE was set.

 # Export PATH="$PATH:$xil_home/EDK/gnu/microblaze/lin/libexec/gcc/microblaze-xilinx-elf/4.1.2"
 # export PATH=$PATH:[tools_directory]/microblaze-gnu/binaries/lin32-microblaze-unknown-linux-gnu_14.3_early/bin
 # export CROSS_COMPILE=microblaze-unknown-linux-gnu-

The generated device tree file "xilinx.dts" need to be copied to the path "[kernel_directory]/arch/microblaze/boot/dts" so that the embedded Linux kernel can be generated considering the hardware configuration of the embedded system during the compilation process.

5.2 Embedded Linux kernel configuration

To configure the embedded Linux kernel source, the menuconfig tool was used (see Fig. 4) which is part of the Linux kernel. First, a default configuration "mmu_defconfig" was used and then the "make menuconfig" command targeting the processor architecture.

 # cd [tools_directory]/linux-xlnx
 # make ARCH=microblaze mmu_defconfig
 # make ARCH=microblaze menuconfig

The configuration should be selected according to the board that are using. In the "General Setup" option shown in Fig. 4, the path where the file system is located was selected, which in this case is "[tools_directory]/initramfs_minimal.cpio.gz". In "Platform options", each option should match with the MicroBlaze processor configuration and defined peripherals that were included in XPS. In "Processor type and features" option, the option "Default kernel string" should match with the string in the device tree, that is "console=ttyUL0 root=/dev/xsysace/disc0/part1". In "Device Drivers" option, it is necessary to specify which drivers were necessary according to the hardware that was defined. In this particular case, the appropriate driver for the network interface in the system (Xilinx LLTEMAC 10/100/1000 Ethernet MAC driver) was configured entering to the Network Device Support option, Ethernet Driver support option.

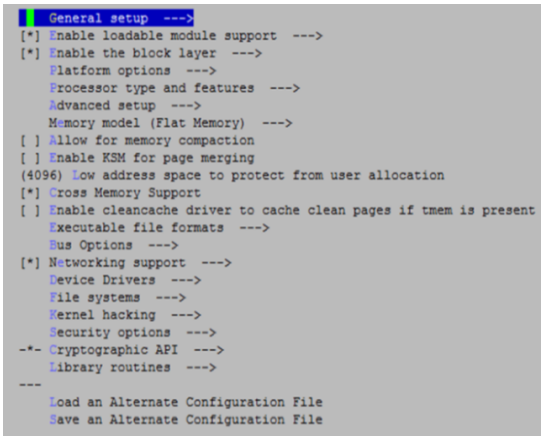

Fig. 4: Kernel menu configuration options

5.3 Kernel compilation

The embedded Linux kernel should be generated in a development machine like Linux Workstation instead of being compiled in the target embedded system. After that, the compiled kernel image is downloaded to the board. The "make" command is typed with the name of the kernel, which in this case is "simpleImage" followed by a period and the string "xilinx" which should be the same name of the device tree file.

 # make ARCH=microblaze simpleImage.xilinx

This process took about 30 minutes. This duration could be different based on the amount of options that were enabled during the configuration, the functionality that the embedded Linux kernel would include, the configuration of the PC workstation (number and speed of CPU cores, amount of RAM, and current load of programs). In this case, the kernel image is located in this path ([tools_directory/Linux-xlnx/arch/microblaze/boot]). This image file already includes the root file system.

6. ACE FILE GENERATION AND DOWNLOAD

There are many configuration modes for the FPGA. In this project, a CF memory was used to store the Xilinx System ACE™ Technology Configuration File which is generated from the FGPA bitstream and the Executable Linked Format File (ELF) [10] which is the format of the compiled kernel image. To create this file, the Xilinx Microprocessor Debugger (XMD) tool is used which is included in the EDK software and

provides a Tool Command Language (Tcl) Interface [10]. In this interface, the System ACE File Generator (GenACE) script named "genace.tcl" was used as well as an options file "myproject.opt" which should contain the following lines:

-jprog
-hw system.bit
-ace project.ace
-board ml505
-target mdm
-elf simpleImage.xilinx
-start_address 0x00000000

The ACE file "project.ace" was generated using the following command:

 # xmd -tcl genace.tcl -opt myfile.opt

The ACE file that is generated after the execution of the last command, should be copied into the CF memory. According to [11], by default, the board ML505 has a System ACE Controller that access the data stored in the card to allow the card to program the FPGA through the JTAG port. This controller supports up to eight configuration images on a single CompactFlash card which is already formatted in FAT16 file system with eight available partitions. The partition 1 was chosen for storing the kernel image. Then, the CF memory is inserted in the board, and connected the board serial interface to the PC workstation serial port with the same communications parameters (baud rate: 9600; data: 8 bits; not parity nor flow control just for this time). The board was configured using the three configuration address switches so that the FPGA can read the ACE file from the proper location (partition 1) in the CF memory. Turn on the board and wait for around 35 seconds. During this time, the FPGA is being configured and the OS starts booting (see Fig. 5).

Fig. 5: Linux loading in the FPGA based board

7. DEVELOPING A NETWORK SNIFFER

Once the embedded Linux OS is running on the board, it is possible to execute applications. Usually, it is necessary to use previously compiled libraries and include header files depending on the type of application. These two elements facilitate and speed up the programming process and in turn avoid writing unnecessary code. An example application was tested on the target board to show the procedure that was followed. The application is a network sniffer that runs on the embedded system that was designed previously (Fig. 2). The procedure for generating the application is shown in Fig. 6.

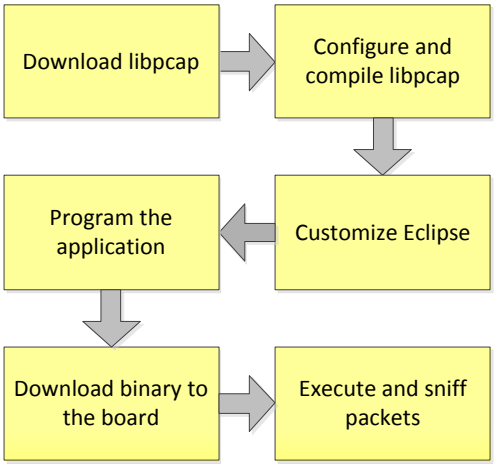

Fig 6: Procedure for generating the sniffer program

7.1 Configuring and compiling libpcap

In this task, the libpcap library is compiled using cross compile tools for MicroBlaze processor. The PATH environment variable should remain the same as before the creation of the kernel. The libpcap source code (libpcap-1.4.0.tar.gz) was downloaded from the libpcap public repository (http://www.tcpdump.org/). After uncompressed the packet, a new directory was created which path was denoted as "[libpcap-source-directory]". A new directory called "build" was created inside this directory. The libpcap source code was configured and compiled using the compiler name and relevant options as shown here:

 # tar xvzf libpcap-1.4.0.tar.gz
 # cd libpcap-1.4.0
 # mkdir build
 # cd build
 # CC=microblaze-unknown-linux-gnu-gcc ../configure
--host=microblaze-unknown-linux-gnu
--with-pcap=linux
ac_cv_linux_vers=2
 # make

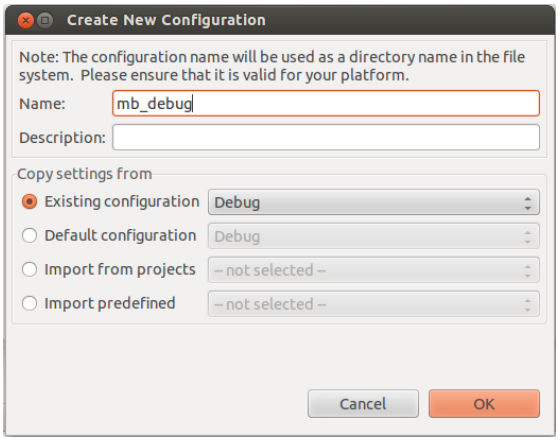

Fig. 7: Creation of the new build configuration in Eclipse

7.2 Customizing Eclipse

The Eclipse software is an integrated development environment (IDE) that provides a sophisticated, featured-rich framework to host embedded development tools [12]. Eclipse was used to construct a functional cross-development environment using the cross-compilation tool chain (microblaze-gnu) that was used for compiling the kernel. The example application was built through integrating the libpcap library and the cross-compilation tool chain in the Eclipse environment. To do that, some default build parameters were modified in Eclipse. Eclipse and C/C++ Development Tools (CDT) plug-in were downloaded and installed based on the guide [13]. Then, a new project with a new build configuration was created. This configuration was called "mb_debug" (see Fig. 7).

The settings option was modified in the C/C++ Build Configuration option of the project properties. The command path for the compiler, linker and the assembler (see Table 1) as well as the paths for directories of headers (see Table 2), and the paths for libraries (see Table 3) [14] needed to be configured. The path for the cross-compilation tools is denoted as "[crosstool-directory]" and is equivalent to ([tools_directory]/microblaze-gnu/binaries/lin32-microblaze-unknown-linux-gnu_14.3_early).

Table 1. Eclipse "Commands" configuration

GCC C Compiler	[crosstool-directory]/bin/microblaze-unknown-linux-gnu-gcc
GCC Linker	[crosstool-directory]/bin/microblaze-unknown-linux-gnu-gcc
GCC Assembler	[crosstool-directory]/bin/microblaze-unknown-linux-gnu-as

Table 2. Eclipse "Compiler includes" configuration

GCC C Compiler, Includes, Include paths (-l)
[crosstool-directory]/microblaze-unknown-linux-gnu/sys-root/usr/include
[libpcap-source-directory]

Table 3. Eclipse "Linker libraries" configuration

GCC Linker, General	
No shared libraries (-static)	
GCC Linker, Libraries	
Libraries (-l)	pcap
Library search path (-L)	[crosstool-directory]/microblaze-unknown-linux-gnu/sys-root/lib
	[libpcap-source-directory]/build

7.3 Programming the application

The application that was used for testing the target board is a simpler version of "sniffex.c" found in [15]. The application captured a group of 100 network packets that travel over an Ethernet interface and the main information of each packet such as the source and destination IP address, ports, and transport protocol if any, was shown in the standard output. The application was created using Eclipse with the new "mb_debug" build configuration. After that, it was possible to compile the project for the target board as any other application. The executable file called "microblaze-pcap" is located in the directory ([workspace]/microblaze-pcap/mb_debug). Fig.7 shows the general process of creating an application for the target board.

7.4 Downloading and executing

To perform this task, it is necessary an Ethernet network connection between the PC workstation and the target board, using the network interface (eth0) available on the board. This interface was activated and then, an IP address was assigned. Then, the binary file was transferred using the FTP client embedded in the file system of the target board and a simple FTP server installed in the PC workstation.

 # ifconfig eth0 up
 # ifconfig eth0 [IP address] netmask [netmask] broadcast [broadcast address]
 # ftpget -u [user] -p [password] [board IP address] [board binary file] [workstation binary file]

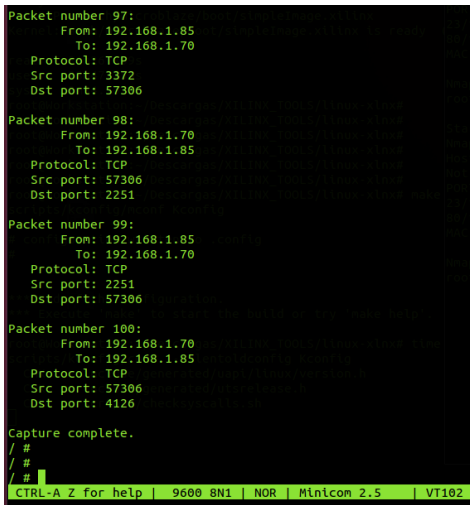

Fig. 8 Output of the sniffer program in Minicom

The binary file called "microblaze-pcap" was executed pointing the network interface "eth0" as an input parameter.

 # chmod +x microblaze-pcap
 # ./microblaze-pcap eth0

Then, the system will sniff this interface. Fig. 8 shows a partial view of the capturing process of data packets in the serial interface in the Workstation using Minicom software in Linux.

8. CONCLUSIONS

All the information provided in this paper is useful for system designers involved in starting up new projects that require high performance and use of embedded OS such as Linux for FPGA-based systems. Three key topics were covered here: the requirements for obtaining a hardware design based on FPGA that supports Linux, the steps for embedding Linux in this hardware and the procedure to configure Eclipse for testing a network sniffer application in the embedded system.

9. FUTURE WORK

Future work will include the modification of the Linux kernel to add a filter functionality to drop packets that are not important for a particular application. Furthermore, we are planning to design a hardware version of data packet sniffer using FPGA resources by designing custom IP cores that will improve the process speed. After that, we will develop the custom Linux drivers for handling these new cores.

10. ACKNOWLEDGEMENT

This project was made possible thanks to Mr. MSc. Aurelio Morales-Villanueva, professor at National University of Engineering (UNI), Perú, who always contributed with wise advices during each stage of the development of the embedded system. Also, the authors gratefully acknowledge the support of INICTEL-UNI for all the given facilities with respect to infrastructure and equipment.

11. REFERENCES

[1] J.O. Hamblen, T. S. Hall; "Using System-on-a-Programmable-Chip Technology to Design Embedded Systems"; International Journal of Computer Applications, Vol. 13, No. 3, Sept. 2006.

[2] A. Morales-Villanueva; A. Gordon-Ross; "On-chip Context Save and Restore of Hardware Tasks on Partially Reconfigurable FPGAs"; Proceedings of the IEEE 21st Annual Symposium on Field-Programmable Custom Computing Machines (FCCM'13), pp. 61-64.

[3] Corbet, Jonathan; Rubini, Alessandro; Kroah-Hartman, Greg. Linux Device Drivers. 3rd Edition. O'Reilly. February 2005. ISBN: 978-0-596-00590-0. Pag. 5.

[4] Luca Deri; "Improving Passive Packet Capture: Beyond Device Polling"; In Proceedings of the 4th International System Administration and Network Engineering Conference (SANE), October 2004.

[5] Rita Nagar; Ravi Kiran Jadi; Prabir Saha; "Porting Linux Kernel on FPGA based Development Boards". International Conference on Computing, Communication and Sensor Network (CCSN) 2012.

[6] Xilinx. Microblaze Linux (General). [Online]. Available from: http://xilinx.wikidot.com/microblaze-linux [Accessed: March 19, 2013].

[7] Xilinx, "EDK Concepts, Tools, and Techniques. A Hands-On Guide to Effective Embedded System Design". Xilinx Online Documents. UG683 (v13.2), July 6, 2011. Page 24. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_2/edk_ctt.pdf

[8] David Gibson; Benjamin Herrenschmidt; "Device trees everywhere"; February 13, 2006. http://ozlabs.org/~dgibson/papers/dtc-paper.pdf

[9] Xilinx. Device Tree Generator. [Online]. Available from: http://xilinx.wikidot.com/device-tree-generator [Accessed: March 25, 2013]

[10] Xilinx, "Embedded System Tools Reference Guide. EDK 12.1". Xilinx Online Documents. UG111. April 19, 2010. pp. 197, 143. http://www.xilinx.com/support/documentation/sw_manuals/xilinx12_1/est_rm.pdf

[11] Xilinx, "ML505/ML506/ML507 Evaluation Platform. User Guide". Xilinx Online Documents. UG347 (v3.1) November 10, 2008. Page 28. http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf

[12] Brian Handley, Senior Engineer Macraigor Systems LLC; "Use Eclipse for embedded cross-development"; EETimes-Asia, June 1-15, 2007.

[13] Jan Axelson. Lake view research. Using Eclipse to Cross-compile Applications for Embedded Systems. Part 2: Install Eclipse and C/C++ Development Tools. Available from: http://www.lvr.com/eclipse2.htm. [Accessed: June 10, 2013].

[14] Jan Axelson. Lake view research. Using Eclipse to Cross-compile Applications for Embedded Systems. Part 3: Create and Configure a Project. Available from: http://www.lvr.com/eclipse3.htm. [Accessed: June 19, 2013].

[15] Tim Carstens. Programming with pcap. [Online]. Available from: http://www.tcpdump.org/pcap.html [Accessed July 4, 2013].