Implementation of Open Source IP based Embedded System

Y.Divya reddy
Department of EIE, CVR College of Engineering, Hyderabad, India
Email: divya.reddy037@gmail.com

***Abstract*—Embedded system plays an important role in various industry applications. An embedded system is consisting of software and hardware. The hardware platform of conventional embedded system is typically based on IC chips that have fixed resources. Besides, with the development of FPGA, an emerging approach for designing embedded system is implementing soft IP cores on FPGAs. Soft IP cores are synthesizable hardware blocks described in HDL language. Their source code can be either open or close to public. For example, Open RISC 1200, is an open source 32-bit RISC microprocessor. In addition, the increasing complexity of embedded system forces software developers to consider operating system support to reduce their workload. Thus, in this paper, a prototype of open source IP based embedded system with Linux is implemented on Atlys (Xilinx Spartan-6) FPGA board and the goal is to evaluate if the system is appropriate for industrial applications. The hardware platform is ORPSOC, which is a reference SoC design based on Open RISC 1200 processor. For software, Linux operating system is installed. Furthermore, an application executes on Linux is developed that reads the output of an I2C compass sensor-LSM303DLM. With the success of the application and the investigation of license issues, the conclusion is drawn that open source IP based embedded system with Linux is usable for industry. Although comparing to conventional embedded system, the open source IP based embedded system with Linux has following cons, such as high product cost, basic-supported development environment and more difficult software development if Linux driver doesn't support the hardware. However, its pros are high flexibility and scalability, high software portability, low software development difficulty and high reusability that make it more suitable for industry usage.**

***Index Terms* - embedded system, open source hardware, Open RISC 1200, Linux**

I. INTRODUCTION

The embedded system plays a more and more important role in the world. It can be found in many applications for industrial use, such as process control and monitoring systems. The task for an embedded system is typically to control the machine to function correctly. It can be said that the industrial productivity is improved with introducing embedded system. Differs from general purpose computer, an embedded system is designed for a dedicated task, which requires both hardware and software operate appropriately. Normally, the embedded system designed in conventional way is consisting of 3 layers as shown in Figure 1. The hardware platform is a PCB board containing a microcontroller and other peripheral ICs such as on-board memory, Ethernet module and etc. The software is the application code of the desired task including hardware drivers.

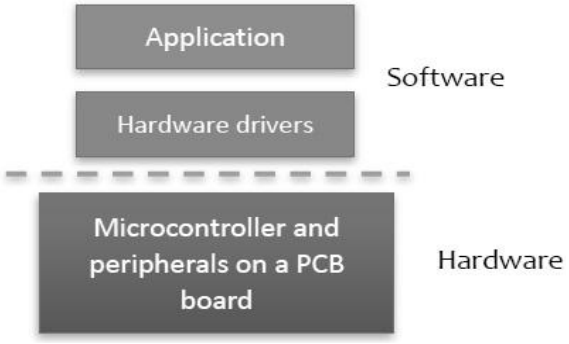

Figure 1: Architecture of conventional embedded system

With the rapid development of semiconductor technology, the capability of embedded system is increasing over time to allow more complex task to be performed. Thus, the embedded operating system support becomes necessary. Figure 2 illustrates the architecture of a conventional embedded system with Linux. The benefits of introducing Linux are providing various hardware drivers, communication protocols and management of system resources that reduce the workload of software developer significantly.

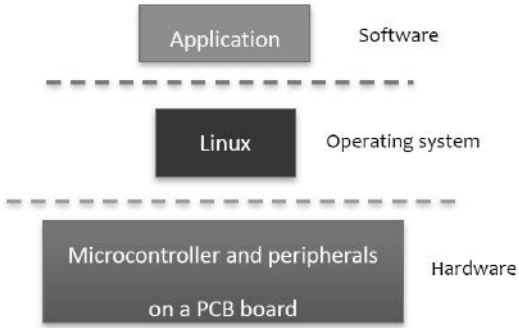

Figure 2: Architecture of conventional embedded system with Linux.

In addition, FPGA (Field Programmable Gate Array) is an alternative for embedded design. It is well known as its hardware re-programmable feature. With this advantage, an emerging technology for achieving embedded system's hardware platform is using soft IP cores. The term IP refers to intellectual property. Soft IP cores are the synthesizable hardware blocks for FPGAs and described in RTL level. A soft IP can be either a processor or other hardware modules such as UART, ETHERNET and I2C controller. Designing embedded system with soft IP cores is flexible and

configurable. At present, there are two leading-edge commercial soft processors, Microblaze and Nios II, supported by Xilinx and ALTERA. Besides, there are other open source processors, such as Open RISC and LEON3. Open IP cores inherit the advantages of soft IPs and are open source to public, which means they are free, and FPGA independent. Therefore, it is meaningful to implement an open source IP based embedded system with Linux to evaluate if it is compatible for industrial applications. In this paper, the hardware platform is ORPSOC (Open RISC Reference Platform System-On-Chip), which is a reference, embedded system design based on Open RISC soft processor. Then the Linux operating system is installed and an application running on Linux that reads the acceleration raw data from an I2C compass sensor, LSM303DLM.

II. Open Source Hardware

The commercial soft IP cores such as Microblaze and Nios II are leading a revolution of embedded system design; they are high performance, well supported and flexible. However, they are not open source and FPGA dependent that means the designers only has the right to use them in their embedded design and the implementation of soft IP cores is only possible on specific vendor's FPGAs. In software world, open source software is becoming popular. The "open source" means "freedom" that everyone has the right to modify it and derived works are allowed under the condition that the developers should pass on the freedom to others. This causes a great success of open source software such as Linux operating system. In hardware world, modern silicon chip is typically built from silicon "intellectual property" (IP), written in a hardware description language. Fabless design houses may never produce a chip themselves—one of the largest and best known is ARM in Cambridge, whose processor IP is built by other companies into one billion chips ever month. That IP costs the same amount to produce, whether it goes into one chip or one billion.

A. Open RISC processor and ORPSOC

Open RISC CPU architecture, one of the flagship projects of opnecores.org is a well-known open source processor. Open RISC 1200 processor is an implementation of Open RISC 1000 processor family. Figure 3 shows the architecture of Open RISC 1200 CPU.

The Open RISC 1200 CPU is a 32-bit scalar RISC with Harvard micro architecture, 5 stage integer pipeline, virtual memory support (MMU) and basic DSP capabilities. The MMU enables the capability of running an operating system. Supplemental facilities include debug unit for real-time debugging, high-resolution tick timer, programmable interrupt controller and power management support[2]. This processor can be synthesized and downloaded onto Altera and Xilinx FPGAs and supports embedded real time operating systems such as Linux, µLinux and OAR RTEMS real time operating system. For software development, tools are available that allow developers to compile programs written in C/C++, Java and Fortran to run on the Open RISC processor [1].

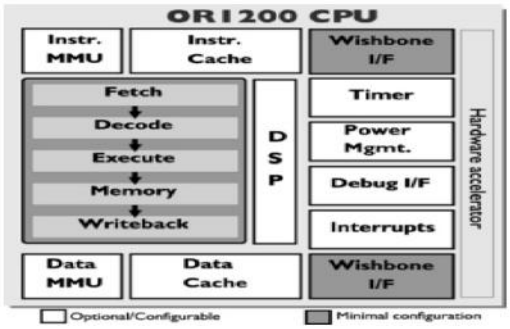

Figure 3: Open RISC 1200 CPU architecture

ORPSOC (Open RISC Reference Platform System-on-Chip) is a reference system-on-chip design that is primarily for Open RISC based embedded system testing and development. And it is also the hardware platform of this thesis that contains an Open RISC 1200 processor and several peripherals. The interconnections between CPU and peripherals are using Wishbone interface, which have been configured already. In ORPSOC's source code, there are make file scripts to generate FPGA configuration bit stream for particular FPGA boards, such as selected Atlys FPGA board in this paper.

B. Linux user space and kernel space

Linux is a successful open source operating system that is widely used in embedded systems as a platform for executing applications. It manages the resources of an embedded system that the software developer can focus on application code on a high-level abstraction view without being involved in hardware driver development if the embedded system performs a complex task. In Linux, memory is divided into two spaces, one is user space, and the other one is kernel space. Figure 4 shows the relationship between user space and kernel space.

The top is user space where applications execute, while the kernel space is an exclusive space only for kernel running. The kernel has the highest authority to access all resources in an embedded system, such as memory and devices. And it should be as stable as possible to prevent any undesired errors happening and coordinates processes. By contrast, user space has less authority that it can't access the data in kernel space directly. The data transition between user space and kernel space is via system call interface.

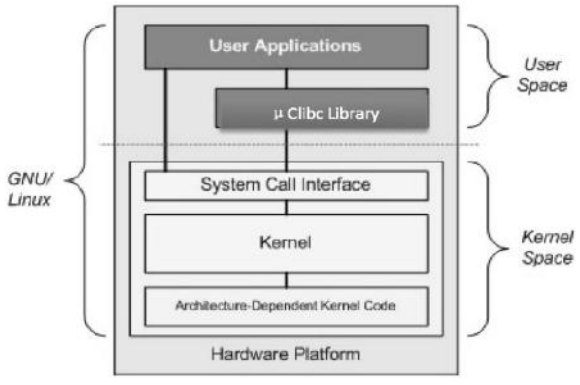

Figure 4: user space and kernel space

 DOI:10.32377/cvrjst0809

Once a user application invokes a system call to access a particular hardware, the kernel handles it with corresponding device driver. A C library is responsible for implementing system calls. Usually it is GNU C Library (glibc). However, GNU C Library is not compatible for embedded Linux due to it is a large library. To quote from Ulrich Drepper, the maintainer of GNU libc: "...glibc is not the right thing for [an embedded OS]. It is designed as a native library (as opposed to embedded). Many functions (e.g., printf) contain functionality which is not wanted in embedded systems." 24 May 1999[3]. Thus, another C library, µClibc is introduced as shown in Figure 4. µ means micro while C stands for controller. And µClibc is the short of "the microcontroller C library". As the name it indicates, the µClibc is intended to support embedded systems that provide as much functionality as possible in a small amount of space.

III. SYSTEM IMPLEMENTATION

In this paper, the usability, license issues, pros and cons of open source IP based embedded system with Linux are investigated via a case study. The case is to develop an application runs on Linux that reads acceleration raw data from an I2C compass sensor-LSM303DLM. The hardware platform of this case is ORPSOC with I2C controller enabled.

The work flow for implementing this case is shown as following:

1. Set up the development environment includes Xilinx ISE for hardware synthesis and GNU tool-chain for software compilation.
2. Generate FPGA configuration bit stream of ORPSOC with I2C controller enabled.
3. Install u-boot.
4. Setup TFTP server and NFS server.
5. Build Linux image and install it through TFTP protocol.
6. Load the application software via NFS protocol to Linux to test the usability of whole system.

If the application is working, then the usability is proved. License issue is surveyed with interpreting license of each component involved in this thesis. And then check out if the usage of each component offends the rules. Pros and cons are obtained by comparing with the previous knowledge of conventional embedded system.

A. Prototype of open source IP based embedded system with Linux

1) The whole system is implemented on Atlys FPGA board including a Xilinx Spartan-6 FPGA chip[7].
2) The USB to JTAG interface is used to program SPI FLASH. The SPI FLASH image contains ORPSOC bit stream for configuring FPGA and binary image of the program that will be executed. In this thesis, the program is u-boot, which is a bootloader for further Linux installation.
3) The interaction interface between user and the board is via UART console.
4) Another host computer acts as TFTP and NFS server. The TFTP server is for transferring Linux image to Atlys board and NFS server is for loading applications to Linux.
5) I2C slave device is a compass sensor, LSM303DLM.

B. Generate ORPSOC with I2C controller enabled

In the source code of ORPSOC, there is a specific folder contains RTL source code and makefile scripts for Atlys board. The I2C controller is a soft IP core that should be added into ORPSOC by connecting it to Wishbone interface. However, it has been connected already and disabled as default in a top-define file "orpsoc-defines.v", which is for configuring functions of ORPSOC. Thus, uncommenting the code "`define I2C0" will enable the I2C controller in ORPSOC[8]. Then the next step is assigning the signals of I2C controller to PMOD socket developed by Digilent for peripheral connection[4]. The I2C interface requires 4 wires to communicate properly; they are Vcc, GND, clock line SCL and data line SDA. In ORPSOC, the names of SCL and SDA signals are "i2c0_scl_io" and "i2c0_sda_io" gained from top-design file "orpsoc_top.v". Figure 5 shows the PMOD interface on Atlys board and signal assignment[4].

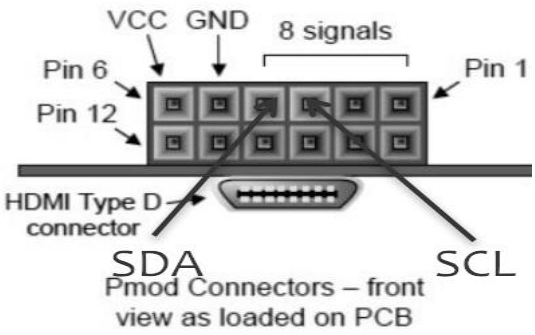

Figure 5: PMOD and I2C signal assignment.

SCL and SDA signals are assigned to Pin3 and Pin4 by editing user constraint file (ucf). The carrier board of I2C slave LSM303DLM has a level-shifter circuit that makes it compatible for PMOD 3.3V system. At last, the hardware platform ORPSOC is generated with the help of makefile scripts[5]. A bit stream for configuring FPGA can be achieved after the synthesis, mapping, place and route steps are finished. Figure 6 shows the block diagram of ORPSOC. The blue blocks are default settings, and the red I2C controller is the one enabled in top-define file.

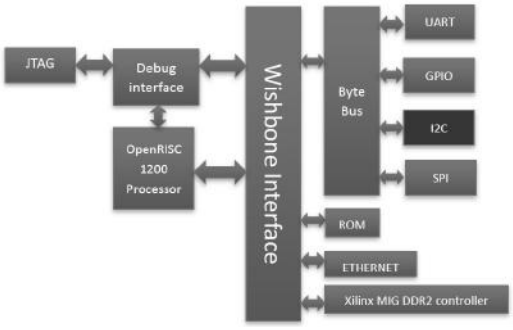

Figure 6: Block diagram of ORPSOC with I2C controller enabled

C. U-boot

Generally, GDB (GNU Project Debugger) is taking the responsibility of loading and debugging the software. However, the Xinlinx USB Platform Cable couldn't be detected by GDB. Thus, an alternative should be found to load the software onto the platform. The approach is u-boot. U-boot refers to universal bootloader, is a primary and powerful bootloader used in embedded systems that eases the procedure of loading Linux image or other application images. The u-boot can only fix the problem of loading software, whereas it can't be a perfect replacement of GDB because software can hardly debug with u-boot. The installation of u-boot is to generate a SPI FLASH image that contains the FPGA configuration bit stream and the u-boot binary image.

In ORPSOC, there is a ROM module written in Verilog language, which acts like a read-only memory. The ROM includes a bootloader program to load the software that is located in SPI flash. Therefore, the ROM and the bootloader are combined to be a bootrom. After the Atlys board is powered on, the FPGA will be configured and the u-boot is copied from SPI flash to external DDR2 RAM to start execution with the help of bootrom[6]. In addition, a block of 4 bytes should be added to the head of u-boot binary image that informs the bootloader in ROM how many bytes should be copied to RAM. In order to load Linux image through TFTP (Trivial File Transfer Protocol), the network parameters of u-boot and host computer should be set properly to build a LAN (Local area network) as shown in Table 1. The Ethernet MAC (Media Access Control) address can be random as long as it is valid if the system is only for experimental use. Moreover, the network speed of host computer should be configured to 100Mbps or below it because 100Mbps is the maximum connection speed supported by Ethernet MAC IP core in ORPSOC[4].

Table 1: Network parameters of u-boot and host computer

	U-boot	**Host computer**
Ethernet MAC address	00:12:34:56:78:9a	Assigned by manufacture
Local IP address	192.168.2.6	192.168.2.2
Network mask	255.255.255.0	255.255.255.0
Gateway IP address	192.168.2.1	192.168.2.1
Server IP address	192.168.2.2	-

D. TFTP and NFS server

TFTP (Trivial File Transfer Protocol) is a simple file transfer protocol without any authentication. Because of its simplicity, it is implemented with less consumption of memory. TFTP server helps transfer Linux images from host computer to the RAM of Atlys board.

The setup procedure of TFTP server is shown below:

1. Install the required packages: tftp, xinetd, tftpd
2. Create a folder for storing images that will be transferred to u-boot. And more important, the folder and the files in it should be configured as everyone can access it.
3. Create a file named "tftp" under "/etc/xinetd.d" path, which is the configuration file for tftp server. Start the service. NFS is defined as Network File System. A NFS server can be mounted on any device supports NFS protocol. Using NFS server during testing avoids copying applications to target board, thus the debugging procedure becomes convenience.

The steps for configuring NFS server is shown below:

1. Install the necessary components: portmap, nfs-kernel-server
2. Edit the file "/etc/exports" to add the entry will be shared.
3. The same as TFTP server, the shared folder and the files it contains should be configured accessible for everyone.
4. Start NFS server and portmap.

E. Building Linux kernel

Before building Linux kernel, a DTS (device tree) file should be created at first. When Linux is booting up, it is necessary for Linux to know what hardware resources the board has. Thus, the device tree plays this role. Device tree is a data structure that describes the hardware and it is passed to Linux kernel by the bootloader. Then the Linux kernel can configure the hardware and work appropriately. The device tree for Atlys board is derived from the one for OpenRISC simulator located in Linux kernel source code. Firstly, the clock frequency and memory size should be corrected to 50MHz and 128MB. And then the I2C controller node can be added according to an example of I2C controller description for device tree gainedfrom Linux's opencores I2C driver. Appendix C shows a complete device tree for this case. For Linux kernel, the I2C device interface support and I2C hardware driver for opencores are attached via "menuconfig", which is a tool for selecting the features of Linux kernel. Then the binary image of Linux kernel can be generated by the GNU tool-chain for Linux development. More than that, the image should be modified to a bootable image for u-boot with the tool "mkimage" provided by u-boot. U-boot refers to universal bootloader, is a primary and powerful bootloader used in embedded systems that eases the procedure of loading Linux image or other application images.

IV. RESULTS

A. Usability

The application reads sensor output successfully that proves the Open source IP based embedded system with Linux is usable for industry. Although the application in this thesis is just to read output from an I2C sensor, but its success demonstrates that the open source IP based embedded system with Linux is usable. With the help of various open source IP cores and Linux drivers, it is possible to develop more complex embedded system for industry usage.

 DOI:10.32377/cvrjst0809

B. Pros and cons

Table 2 is a comparison between open source IP based embedded system with Linux and conventional embedded systems. The open source IP based embedded system with Linux refers to the one whose hardware platform is ORPSOC having Linux installed.

Table 2: A comparison between open source IP based embedded system with Linux and conventional embedded systems

	Open IP based embedded system with Linux	Conventional embedded system without Linux	Conventional embedded system with Linux
Product cost	High	Low	Low
Development environment	Basic-supported	Well-supported	Well-supported
Flexibility and scalability	High	-	Only Linux has scalability
Software Portability	High	Low	High
Software development difficulty	Low (except the case if hardware is not supported by Linux driver)	Normally is high, depends on the vendor's support	Low (except the case if hardware is not supported by Linux driver)
Reusability	High	-	-

The conventional embedded system refers to the one whose hardware platform is based on IC chips with fixed resources, such as AVR32 and ARM architecture. Developers can choose to install Linux on it or not.Obviously, the pros of open source IP based embedded system with Linux are high flexibility and scalability, high software portability, low software development difficulty and high reusability. Whereas the cons are high product cost due to the higher price of FPGA chips, basic-supported development environment and more difficult software development if Linux driver doesn't support the hardware.

V. CONCLUSIONS

In this paper, a prototype of open source IP based embedded system with Linux is presented. A successful application reading the output of acceleration raw data from an I2C compass sensor-LSM303DLM is developed as well, which proves the usability of such an embedded system. Moreover, the investigation of licenses indicates the implementation doesn't offend the rules of GPL and LGPL and developing proprietary software on the system is valid. Hence, companies are allowed to develop and sell the embedded systems without providing their source code to users or public. Comparing to conventional embedded systems, although the open source IP based embedded system with Linux has some disadvantages such as: high product cost, only basic-supported development environment, and more difficult software development if Linux driver doesn't support the hardware. However, it has: high flexibility and scalability, high software portability, low software development difficulty and high reusability.

Open source IP based embedded system with Linux is hard for beginners to start with and the application development for Linux takes time to learn. However, once the developer handles them well, the time to develop an embedded system will be decreased significantly because the lower software development difficulty of Linux user space. Moreover, when the function of such an embedded system is changed, a total re-design of the embedded system can be avoided due to it is flexible, scalable and reusable. In addition to function change, the work of function migration can be reduced because of the high software portability and reusability. Hence, the open source IP based embedded system with Linux is more suitable in industrial usage.

REFERENCES

[1] Jason G. Tong, Ian D. L. Anderson and Mohammed A. S. Khalid, "Soft-Core Processors for Embedded Systems", 2006 International Conference on Microelectronics, Dec. 2006, pp.170-173

[2] Younghoon Bin, Kwangmyong Kang, Hyungjun Kim, Hongkyun Jung, Kwangki Ryoo, "The Development of SoC Platform for Embedded System Applications", 2007 International Conference on Convergence Information Technology, Nov. 2007, pp.2286-2291

[3] Lihong Lian, Xiaochao Li, Fen Xiao, Donghui Guo, "Design and implementation of a debugging system for OpenRISC processor", 2008 2nd International Conference on Anti-counterfeiting, Security and Identification, Aug. 2008, pp.368-371.

[4] Jiesheng Wei, Ling Wang, Feng Wu, Yibo Chen, Long Ju, "Design and implementation of wireless sensor node based on open core", 2009 IEEE Youth Conference on Informtion, Computing and Telecommunication, Sept. 2009, pp.102-105.

[5] Faroudja, A. ; Izeboudjen, N. ; Titri, S. ; Sahli, L. ; Louiz, F. ; Lazib, D, "Hardware/Software development of a System on Chip platform for VoIP application", Microelectronics, Dec. 2009, pp.62-65

[6] Mehdizadeh, N. ; Shokrolah-Shirazi, M. ; Miremadi, S.G, "Analyzing fault effects in the 32-bit OpenRISC 1200 microprocessor", 2008 Third International Conference on Availability, Reliability and Security, March 2008, pp.648-652

[7] Castillo, J. ; Huerta, P. ; Lopez, V. ; Martinez, J.I, "A secure self-reconfiguring architecture based on open-source hardware", 2005 International Conference on Reconfigurable Computing and FPGAs, Sept. 2005, pp.7 pp.-10

[8] Younjin Jung, Ok Kim, Byoungyup Lee, Hongkyun Jung, Kwangki Ryoo, "SoC platform design with multi-channel bus architecture", 2008 International SoC Design Conference, Nov. 2008, Vol.03, pp.III-48-III-49