By consultant Søren Hansen (soren.hansen@teknologisk.dk)

Abstract

As the development of Linux progresses and an increasing number of hardware platforms are being supported, it becomes increasingly interesting to utilize this particular OS in embedded products. At the same time, Linux is becoming more "responsive" in an RTOS context because of an increasing its usefulness in embedded environments.

In this article we shall discuss why Linux is an ideal alternative to the otherwise customized OS as well as real RTOS. We shall present the advantages of having the same OS on both target and development platforms. We shall also briefly present our views on how the development process can benefit from an OpenSource model. Furthermore, we argue that Linux has recently matured in some cases.

I want an OS!

- Using Linux in Embedded Environments

Traditionally, it is possible to divide embedded systems into two different categories; one which contains a customized OS with few or many features, and another which contains a traditional RTOS such as VxWorks, QNX or MQX.

The emergence of Linux has displayed a new possible direction concerning an OS choice for these platforms. This is possible because the embedded platforms are turning more powerful and contain more memory. Linux itself is becoming more and more responsive[1] *and as a result the two categories - which are otherwise far from each other - are increasingly overlapping each other.*

.

What makes Embedded Linux an ideal alternative?

Linux in embedded environments is an ideal alternative because of several very important factors.

A Complete OS – Linux

An important benefit by using Linux is that you acquire a complete OS and thereby a system designed for user application development and usage. It improves the level of abstraction as to what is "OS work" and what is "User work", and again it results in a more lose coupling between the two. Porting between platforms and target technologies thus become easier.

The above issue regarding "A Complete OS" also includes extremely important issues concerning hardware drivers, networking etc. In Linux these issues are addressed quite impressively as you do not have to worry about having or acquiring important functionality

[1] According to [NEWSCOM] the Linux kernel version 2.6.x is capable of having a response time of 30us. This means that they are rapidly becoming a usable alternative to a full-fledged RTOS in those areas where such latencies are acceptable.

such as IP stacks and support for popular file systems. They are already an integrated part of the kernel. As regards hardware support, the list of hardware currently supported is increasing every day. If a new driver is required for a specific hardware, suitable literature is readily available with the necessary guidelines for developing customized drivers.

Linux *is* Linux

When comparing an embedded Linux platform to a desktop version you will obviously find some differences. They are mainly due to two different aspects, namely which hardware and library functionality/applications (the latter as a result of memory constraints) they actually support.

The library functionality support is the interesting part here; however the most basic functionality will *almost* always be available. As a consequence hereof, the development of the embedded application can be written and tested in a normal Linux desktop environment. It would normally exclude the drivers controlled by the application.

There are several substantial benefits gained in this context:

- The Turn-Around-Time (TAT) for development tests are reduced considerably.
- Reducing TAT excessive testing is an order of magnitude easier, which results in fewer errors on the target platform.
- The Time-To-Market can thus be reduced and create higher productivity.

Using Linux and pursuing a design philosophy - where device drivers are kept small and compact and where the main application is large and resides in user space - a partitioning is achieved which is characterized by being much less sensitive towards hardware upgrades. Consequently hardware upgrades may result in *only* a new device driver.

The section name also points to another advantage namely that both the development and target platforms are more or less identical. As a result, the developer can concentrate on maintaining his or her knowledge of a single OS instead of multiple ones.

OpenSource

Since the term OpenSource is applicable in this context, you can start to include or utilize the numerous applications that are already written for Linux. If for example you need to:

- Publish internal data on http server.
- Connect to the system to alter configuration using the ssh server.
- Acquire an IP address via the DHCP client.
- Configure the target or show status via SNMP using the snmpd server.
- Use scripting such as bash, perl, python, ruby etc.
- Etc.

The conclusion is therefore that using OpenSource 3rd party software can potentially boost productivity whereby development can focus on the company’s main area of production.

The CGL initiative

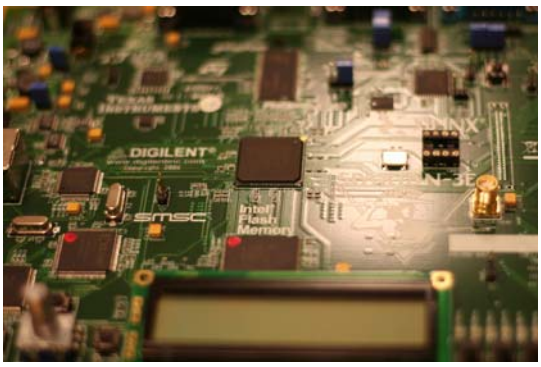
An eternally critical issue is whether or not the system of choice can achieve a satisfactory uptime. In particular in the telecommunication industry, an uptime of 5x9[2] is often an indisputable demand requiring the system to have a high degree of maturity. To meet such demands, the CGL or Carrier Grade Linux [CGL] initiative was formed and as a result hereof a number of Linux distributions have been certified to meet these requirements.
In short, if your product must fulfil high demands regarding uptime, Linux being too immature should not be considered a valid reason for discarding it as a target platform.

Up and running with Linux on a FPGA

The Spartan 3E Starter Kit

Kick off

In the Spartan 3E FPGA chip, a so-called soft-core[3] processor named Microblaze can be pre-programmed. The Microblaze processor is capable of running a Linux kernel. At this time the uClinux Linux distribution has been updated to include support for this processor. uClinux is a collection of libraries and applications together with a number of kernels[4] which can all be tailored to suit individual needs.



As a starting point we choose to get the board booting with a simple classic “Hello world” programme simply to demonstrate that it could be done and that it was not too difficult. A detailed description of how to do this can be found in the Howto [HOWTO-1]. The system created can be controlled through a serial console from which it is possible to logon to once the kernel has booted. At this point it is possible to logon to the system, traverse the file system and run the available commands.

Perspectives

With the basic system up and running, the next step is to enable networking support such that real usage is applicable. The next task will therefore be for the system to acquire an IP and start a web server. Status information should be displayable on this web server system. This demonstrates the easily obtained power using OpenSource. In our case, using uClinux gives us a free IP stack (the kernel), a DHCP client and finally a simple web server enabling us to rapidly create an application which can collect statistics on the embedded platform through 3rd party software. A new Howto will be available when networking support has been attained.

---

[2] 5x9 is an expression describing the amount of time which a system within a certain time frame must be running. If we assume that the duration is a year then the down time for this particular device may not exceed 315 seconds.
[3] The processor core is an IP (Intellectual Property) Core by Xilinx called Microblaze. This processor core is simple and does not contain a MMU (Memory Management Unit).
[4] By a number of kernels is meant kernel minor versions 2.2.x, 2.4.x and 2.6.x.

To come

Virtex 4 Starter Kit

The first thing to be tested and verified is whether the distribution setup that works for the Spartan 3E also works for the Virtex 4, with the soft-core processor used still being Microblaze. This test includes networking, a web server etc. to be up and running as with the Spartan 3E Starter Kit.

Looking ahead, the next important step is to "upgrade" system from using a soft-core processor to a using the hard-core PowerPC instead. This requires more extensive work. However, a port to the Linux 2.6.x kernel should exist. Amongst the improvements is MMU support which results in improved system stability.

Summary

In this article we have covered Linux in embedded environments as a feasible alternative to traditional solutions. The reasons include but are not limited to:

- Reduced Time-To-Market.
- Reduced Turn-Around-Time in the development phase.
- Easier testing.
- Reduced number of errors in the final product.
- Easier porting to new hardware.

This is achieved as development and testing to a great extent are performed on a workstation rather than on target.

References

[LJ-1] Linux Journal - "SMP and Embedded Real Time":
http://www.linuxjournal.com/article/9361

[LJ-2] Linux Journal - "Choosing a GUI Library for Your Embedded Device":
http://www.linuxjournal.com/article/9403 (Needs admittance to LJ)

[LJ-3] Linux Journal - "How to Port Linux When the Hardware Turns Soft":
http://www.linuxjournal.com/article/9362 (Needs admittance to LJ)

[CGL] Carrier Grade Linux: http://www.osdl.org/lab_activities/carrier_grade_linux/

[NEWSCOM] http://news.com.com/Novell+to+launch+quick-response+Linux/2100-7344_3-6117479.html

[HOWTO-1] "Howto create a project for a simple uClinux ready MicroBlaze 4.0 design on XPS (Xilinx Platform Studio) for Spartan-3E"
http://www.teknologisk.dk/20356