Embedded Linux Development with Debian for ARM

Roland Stigge

October 8, 2007

Embedded Linux Development with Debian for ARM
by Roland Stigge

Published 10 October 2007
Copyright © 2007 Philosys Software GmbH

Contents

1 Introduction 2

2 Debian GNU/Linux 2
2.1 Embedded Debian, Emdebian 2

3 Approaches to Embedded Linux Development Environments 3
3.1 Native Debian 3
3.2 Cross compilation of standard Debian packages 3
3.3 Individual cross toolchain use 4
3.4 Emdebian package management for the target 4

4 Emdebian Example Session 4

5 Conclusion 5

6 References 6

Abstract

This article describes several approaches to Embedded Linux development for the ARM platform done purely with software available within the Debian GNU/Linux distribution. One method, done with tools available in the Emdebian subproject inside Debian, is emphasized, while examples for setup and typical Embedded Linux work are given.

1 Introduction

Traditionally, there are several specialized Linux distributions available in the Embedded market. This is founded on several reasons, for example different focuses on commercialization of special market niches (like handhelds, realtime and industrial control units), concentration on certain kinds of devices or special development models. This way, distributors and distributions like Montavista, Denx, GPE and uClinux concentrate on certain subsets of those aspects.

In contrast, traditional GNU/Linux distributions are trying to provide a solution that fits a maximum of purposes, like desktop and server use. While typical embedded devices like handhelds, automotive and communication devices are catching up with common requirements of those general purpose operating systems, Embedded software development is still a more sophisticated process than mostly collecting commonly availabe software packages on the target platform.

This article shows an approach that integrates both the use of a traditional GNU/Linux distribution and Embedded development environment by using mostly the software already provided by the GNU/Linux distribution.

2 Debian GNU/Linux

In contrast to many other distributions, Debian GNU/Linux ["DEBIAN"] doesn't primarily focus on desktop or server use. It is one of only a few distributions that consists of a widely diverse developer base that works on a broad spectrum of aspects to operating systems. Therefore, Debian is often referred to as the "Universal Operating System". In fact, the system is that flexible that other parties consider it a meta distribution and derive from it to create separate GNU/Linux distributions like Knoppix and Ubuntu with special focuses, while Debian itself is a fully usable distribution in its own right.

While in the past, it was still common to use additional third party software on a Debian development host for Embedded Linux development, it is now possible to only use Debian packages for this purpose.

2.1 Embedded Debian, Emdebian

Due to its size (>1000 official developers with even more regular contributors, and about 18000 software packages), Debian consists of several subprojects focusing on different aspects like quality assurance, release management, security, archive maintenance, and packaging, of course. One of those efforts is a working group dedicated to Embedded Linux aspects. Here, everything related to using Debian on the development host and the embedded target is concentrated. While Embedded Debian ["EDEB"] is a general term for all of those embedded aspects, Emdebian is the name for a special bundle of software inside Debian to accomplish certain kinds of embedded software development work, e.g.:

- Installing and setting up different cross development toolchains from Debian
- Building custom toolchains, fully integrated into the Debian package management
- Transforming standard Debian software packages into packages suitable for the embedded target
- (Re-) Building software packages for the target, the Debian way
- Quality Assurance tools for embedded Debian packages

While Debian natively supports 11 different processor architectures (x86, amd64, arm, sparc, mips/el, hppa, ia64, powerpc, s390, alpha) and native development and compilation on those platforms, traditional Embedded Debian work focuses on cross platform development which heavily differs from normal Debian methods. Embedded Debian is currently mostly used by ARM software projects, due to the popularity of this architecture in the embedded world. However, most concepts of Embedded Debian development can easily be applied to other architectures, since tools and source code compatibility are designed in a quite generalized way.

3 Approaches to Embedded Linux Development Environments

Embedded development work with Debian, as described in this article, focuses on the use of certain tools and techniques available within Debian. However, Embedded Linux projects utilizing Debian can be categorized into several classes, as described in the following.

3.1 Native Debian

The original way to deploy Debian GNU/Linux onto different target platforms is to port the whole software stack available inside Debian to the respective platform. This includes the whole toolchain to be available natively on the target so that the target architecture can recompile all Debian packages (which is important for Debian since all software is expected to compile from source). In fact, the build machine network of the Debian projects includes machines of all supported architectures to build daily-current versions of software automatically, without the need for cross compilation.

Unfortunately, this technique is not well suited for embedded projects where the target devices lack ressources (RAM, storage, processing speed). Instead, this method is only suitable for reasonably well equipped machines. However, we need to keep in mind that due to recent developments and new requirements in the hardware sector, even small and mobile devices are getting more and more powerful to fill this gap. In fact, many of the currently available evaluation boards of hardware manufacturers are already powerful enough to run native Debian installations and are actually in use as Debian build daemons.

On the other hand, hardware requirements of software commonly run on embedded Linux targets don't necessarily grow with the same speed. Therefore, native Debian installations from the standard distribution are a realistic option for embedded Linux projects in the future.

3.2 Cross compilation of standard Debian packages

Certain projects already acknowledge the processing power and capacity of today's embedded systems to run nearly native Debian packages. However, the processing speed of embedded hardware is still far from current compile machines (PCs, etc.). Therefore, the package scratchbox, available in standard Debian, provides a cross compilation framework based on target hardware emulation: The development host provides a basic emulator for the target hardware to simulate a target environment, while actual compilation (the majority of work in common project build setups) is transparently done by a cross compiler. This makes the build appear as a native compilation on the target platform. It makes modification of build scripts (Makefiles etc.) unnecessary, since the cross compilation setup just needs to be done once for the scratchbox environment (Debian package). This method was employed for the Nokia 770 project, and is now available to the whole Debian using Linux world.

While this method addresses the need for easy cross compilation without difficult setups, it doesn't solve the problem of adjusting the contents of actual binary packages to be used on the target software (shrinking to acceptable size etc.).

3.3 Individual cross toolchain use

In practice, many typical embedded (Linux) projects building upon the ARM platform employ a sophisticated cross development toolchain from a third party software provider (if not even a self built toolchain). Cross toolchains and the respective development environment are now completely available from Debian by default. There is no need anymore for additional software installation, and everything is integrated into the well respected Debian package management system.

While this approach still includes more setup work (Makefiles etc.) for a target project than the above described native method, it is a fully Debian-integrated way of cross development work on the host machine. However, binary software on the embedded target machine commonly doesn't support the common Debian package management tools.

3.4 Emdebian package management for the target

To address the middle ground between the above described Native and Individual methods that differ greatly in the basic concepts, tools have been developed inside Debian to benefit from the advantages of both approaches: To address the need for cross development for ressource reasons and still have the useful support for the common Debian package management system (apt, dpkg), the software suite emdebian-tools, available in standard Debian, provides tools for cross compiling standard Debian packages and additionally provided software. The result is a set of automatically modified, cross compiled Debian packages suitable for small embedded systems.

The Emdebian approach acknowledges the high quality and reusability of standard Debian packages. Those can relatively easily be modified (patched) in their source code format to adapt to the requirements of typical embedded target machine requirements. Changes to exclude unnecessary documentation, translation files for all languages (sometimes up to 70 languages with each several dozens of kilobytes) and changelog/license texts from binary packages are relatively easy to accomplish. However, some Debian packages are still not completely suitable for cross compilation. This most famously includes packages like self hosting compilers, but also some sophisticated, nonstandard build scripts make cross compilation more difficult than packages using the most widely used Debian build systems like CDBS ["CDBS"] and debhelper ["DEBHLPR"]. The developers of emdebian-tools have already made sure that the most common Debian packages can be "stripped" this way, i.e. several thousands of Debian packages currenty build seamlessly in the Emdebian way. For some Debian packages, patches are necessary to support the Emdebian build process. This is available from a publicly accessible Subversion repository and will be applied automatically in the build process for packages already reworked this way. However, developers should be able to build all available Debian packages in the Emdebian style to benefit from the above mentioned advantages. Work is currently being done for the less common packages that are not yet supported in this way, but in many cases, the neccessary changes to the respective packages are trivial, and guidelines are given how to prepare the packages in cases of doubt ["EDGUIDE"].

Emdebian is the approach this article mainly concentrates on since it appears as the most flexible and promising method for future Embedded Debian work.

4 Emdebian Example Session

This section gives an overview of a typical project setup and development session and discusses some corner cases for Embedded Linux development with emdebian-tools. See also ["EDQS"] and ["EDGUIDE"]. In the following, the prompt sign "$" designates a user prompt and "#" marks a command entered into a root shell, since the command needs to be entered as root.

```
# aptitude install emdebian-tools
```

With installing the package emdebian-tools, necessary packages for a complete cross development environment are automatically added to the system. During installation, a target platform will be asked for. For our purposes, we would choose "arm".

```
# aptitude update
# aptitude upgrade
```

After installing the package, we probably want to get the latest development version of emdebian-tools even if running the stable version of Debian, since emdebian-tools is under constant development, and we need to keep up with the latest changes.

```
$ emsetup --verbose --simulate
$ emsetup --verbose
```

Running **emsetup** (part of emdebian-tools) does the necessary setup in `~/.dpkg-cross` for the respective user account. First, the same run with the option `--simulate` will show what would be done without actually changing the user's environment. The actual command will also try to install necessary cross toolchain packages according to the above specified default target architecture. If this is not possible for the normal user, run the printed **apt** command as root, afterwards.

```
$ emsource coreutils
```

The **emsource** command automatically downloads and unpacks the specified Debian package (here: coreutils). This will also run **em_make** if the respective Debian package wasn't emdebianized before. This means that some necessary changes to the package (patches) are done automatically, but the user will probably need to check the results afterwards and adjust options as necessary. If patches in the Subversion repository of Emdebian can be found via network access, those are automatically applied instead.

```
$ cd coreutils-5.97
$ emdebuild
```

This command builds the package, the (Embedded) Debian way, using the default cross compiler for the artitecture specified during the installation of emdebian-tools. In this case, it is ARM. Afterwards, the Debian package (`coreutils*.deb`) can directly be installed with dpkg on the target.

```
$ arm-linux-gnu-gcc -o hello hello.c
```

This is just an example for using the Debian provided cross compiler to create a target executable. The cross toolchain can be used like any other toolchain as known from other Embedded Linux distributions.

```
$ emchain
```

This command (re-) builds a toolchain for the current package state of Debian. I.e., the latest source versions of toolchain packages are used.

```
$ svn co http://buildd.emdebian.org/repos/current/emdebian/trunk/buildcross/trunk ↩
    buildcross
$ cd buildcross
$ vi conf_vars.sh
$ buildcross
```

This is the canonical way of building all cross toolchain packages for all desired architectures. This script is being run on the Emdebian buildd machine of the Debian project to support several binary cross development toolchains by default.

5 Conclusion

This article summed up several approaches to ARM platform cross development under Debian and focused on the new Emdebian method. As shown, Debian already provides complete and extensive means for the whole development cycle of ARM-based embedded projects. Finally, due to the universal nature of Debian, we just needed to focus on the recently still missing part of available cross toolchains and automatic Debian package handling for this purpose. Other parts of software support for development (UML and other analysis and design programs, testing frameworks ["STIGGE06"], maintenance tools, debuggers, SCM, project management, programs for documentation, productivity, etc. are already available for quite some time now.

6 References

[CDBS] *CDBS, the Common Debian Build System, http://build-common.alioth.debian.org/*

[DEBHLPR] *Debhelper, a still common predecessor of cdbs http://kitenet.net/~joey/code/debhelper/*

[DEBIAN] *Debian GNU/Linux, the Universal Operating System, http://www.debian.org/*

[EDEB] *Embedded Debian Overview, http://wiki.debian.org/Embedded_Debian*

[EDGUIDE] *Embedded Debian Development Guide, http://wiki.debian.org/EmdebianGuide*

[EDQS] *Embedded Debian Quick Start, http://wiki.debian.org/EmdebianQuickStart*

[STIGGE06] *Embedded Linux Quality Assurance: Unit Testing with Open Source, Roland Stigge, Electronica 2006, Embedded Conference*