Toolchains

From eLinux.org
Jump to: navigation, search

A toolchain is a set of distinct software development tools that are linked (or chained) together by specific stages such as GCC, binutils and glibc (a portion of the GNU Toolchain). Optionally, a toolchain may contain other tools such as a debugger or a compiler for a specific programming language, such as C++. Quite often, the toolchain used for embedded development is a cross toolchain, or more commonly known as a cross compiler. All the programs (like GCC) run on a host system of a specific architecture (such as x86), but they produce binary code (executables) to run on a different architecture (for example, ARM). This is called cross compilation and is the typical way of building embedded software. It is possible to compile natively, running GCC on your target. Before searching for a prebuilt toolchain or building your own, it's worth checking to see if one is included with your target hardware's Board Support Package (BSP) if you have one.

Contents

[hide]

Introduction

When talking about toolchains, one must distinguish three different machines:

- The build machine, on which the toolchain is built
- The host machine, on which the toolchain is executed
- The target machine, for which the toolchain generates code

From these three different machines, we distinguish four different types of toolchain building processes:

- A native toolchain, as can be found in normal Linux distributions, has usually been compiled on x86, runs on x86 and generates code for x86.
- A cross-compilation toolchain, which is the most interesting toolchain type for embedded development, is typically compiled on x86, runs on x86 and generates code for the target architecture (be it ARM, MIPS, PowerPC or any other architecture supported by the different toolchain components)
- A cross-native toolchain, is a toolchain that has been built on x86, but runs on your target architecture and generates code for your target architecture. It's typically needed when you want a native GCC on your target platform, without building it on your target platform.
- A Canadian build is the process of building a toolchain on machine A, so that it runs on machine B and generates code for machine C. It's usually not really necessary.

Toolchain components

Binutils

The GNU Binutils is the first component of a toolchain. The GNU Binutils contains two very important tools:

- *as*, the assembler, that turns assembly code (generated by GCC) to binary.
- *ld*, the linker, that links several object code into a library, or an executable.

Binutils also contains a couple of other binary file manipulation or analysis tools, such as objcopy, objdump, nm, readelf, strip, and so on. The Binutils website has some documentation on all these tools.

C, C++, Java, Ada, Fortran, Objective-C compiler

The second major component of a toolchain is the compiler. In the embedded Linux, the only realistic solution today is GCC, the GNU Compiler Collection. Nowadays, as input, it not only supports C, but also C++, Java, Fortran, Objective-C and Ada. As output, it supports a very wide range of architectures.

C library

The C library implements the traditional POSIX API that can be used to develop userspace applications. It interfaces with the kernel through system calls and provides higher-level services.

Realistically, there are nowadays two options for the C Library:

- glibc is the C library from the GNU project. It's the C library used by virtually all desktop and server GNU/Linux systems. It's feature-full, portable, complies to standards, but a bit bloated.
- Embedded GLIBC (EGLIBC) is a variant of the GNU C Library (GLIBC) optimized for embedded systems. Its goals include reduced footprint, support for cross-compiling and cross-testing, while maintaining source and binary compatibility with GLIBC. The project is discontinued.
- uClibc is an alternate C library, which features a much smaller footprint. This library can be an interesting alternative if flash space and/or memory footprint is an issue. However, the space advantages gained using uClibc are becoming less important as the price of memory and flash continues to drop. It is still useful C library for embedded systems without an MMU.
- uClibc-ng is a spin-off of uClibc C library. The main goal of the spin-off is to do regular releases and do a lot of automatic runtime testing.
- musl New standard C library. musl is lightweight, fast, simple, free, and strives to be correct in the sense of standards-conformance and safety.

The C library has a special relation with the C compiler, so the choice of the C library *must* be done when the toolchain is generated. Once the toolchain has been built, it is no longer possible to switch to another library.

Debugger

The debugger is also usually part of the toolchain, as a cross-debugger is needed to debug applications running on your target machine. In the embedded Linux world, the typical debugger is GDB.

Lazarus and Free Pascal

Free Pascal is a professional but free 32 bit / 64 bit compiler for Pascal and Object Pascal. It supports a wide variety of processors and Linux distributions including the Raspberry Pi.

The Free Pascal toolchain is widely independent from GCC and other external tools. Major components are the Free Pascal compiler (FPC), a command-line tool, a text-mode IDE and, as an optional component, Lazarus, a full-featured GUI-based IDE. FPCUnit is a framework allowing for unit-testing.

On most platforms Free Pascal makes use of the GDB debugger.

http://elinux.org/Tiny6410

http://elinux.org/Micro2440

http://elinux.org/Mini210

http://elinux.org/Tiny210

Getting a toolchain

There are several ways to get a toolchain:

- Get a prebuilt toolchain, either from a vendor such as CodeSourcery, or probably inside the Board Support Package shipped with your hardware platform by the vendor. This is the easiest solution, as the toolchain is already built, and has supposedly been tested by the vendor. The drawback is that you don't have flexibility on your toolchain features (which C library? hard-float or soft-float? which ABI?)
- Build a toolchain on your own. However, this can be a real pain. There are version dependency issues, patches required to make something work, etc., etc. Check out this (obsolete) build matrix for crosstool and look at all the red "failed" entries.
- Build a toolchain using an automated tool. The community has built several scripts or more elaborate systems to ease the process of building a toolchain. This way, the recipes and patches needed to build a

toolchain made of particular versions of the various components are shared and easily available.

Prebuilt toolchains

Free Electrons

Since June 2017, Free Electrons is offering free pre-built toolchains on toolchains.free-electrons.com, supporting more than 16 hardware architectures (not counting the variants within each architecture), and three C libraries: GNU libc, uClibc-ng and musl. More details...

CodeSourcery

CodeSourcery used to release free cross-compiling toolchains for the major embedded architectures.

In 2010, CodeSourcery was acquired by Mentor Graphics, who now sells the materials described here as part of their Sourcery Tools Services product line. Indeed, the above link (to codesourcery.com) now takes you to Mentor Graphics' web site.

Now the free toolchains binaries they maintain only support MIPS, NIOS-II, AMD64 and Hexagon. Old versions with ARM support are still available through build systems (Buildroot...), which still know where to find them.

Linaro (ARM)

Linaro releases optimized toolchains for recent ARM CPUs (Cortex A8, A9...). These include Linaro's latest contributions to mainline GCC, but backported to stable GCC versions for immediate use by product developers. Linaro actually hires CodeSourcery people to improve ARM toolchains, so the ARM toolchains that you get with Linaro should be at least as good as the CodeSourcery ones.

Native toolchains are available through the standard GCC toolchain in Ubuntu. Cross toolchains from Linaro are available to Ubuntu users through special packages:

```
sudo add-apt-repository ppa:linaro-maintainers/toolchain
sudo apt-get install gcc-arm-linux-gnueabi
```

Now find out the path and name of the cross-compiler executable by looking at the contents of the package:

```
dpkg -L gcc-arm-linux-gnueabi
```

Arch Linux users can install:

```
yaourt -S gcc-linaro-arm-linux-gnueabihf
```

Linaro also makes source releases which can then be used by any build system (see below).

DENX ELDK

The DENX Embedded Linux Development Kit (ELDK) provides a complete and powerful software development environment for embedded and real-time systems. It is available for ARM, PowerPC and MIPS processors and consists of:

- Cross Development Tools (Compiler, Assembler, Linker etc.) to develop software for the target system.
- Native Tools (Shell, commands and libraries) which provide a standard Linux development environment that runs on the target system.

- Firmware (U-Boot) that can be easily ported to new boards and processors.
- Linux kernel including the complete source-code with all device drivers, board-support functions, etc.
- Xenomai - RTOS Emulation framework for systems requiring hard real-time responses.
- SELF (Simple Embedded Linux Framework) as fundament to build your embedded systems on.

All components of the ELDK are available for free with complete source code under GPL and other Free Software Licenses. Also, detailed instructions to rebuild all the tools and packages from scratch are included.

The ELDK can be downloaded for free from several mirror sites or ordered on CD-ROM for a nominal charge (99 Euro). To order the CD please contact office@denx.de.

Detailed information about the ELDK is available here.

Scratchbox

Scratchbox provides toolchains for ARM and x86 target architectures (with PowerPC, MIPS and CRIS in experimental stages, but they aren't making real progress for the past many years, so Scratchbox should probably be considered ARM and x86 only). Both uClibc and glibc are supported.

Scratchbox simplifies cross compiling software which is built using GNU autotools - Code tests performed by configure are run in an emulator or even on the actual target. The toolchains scratchbox ships with are based on GCC 3.3 and as such are quite old, but stable and well tested. It should be pointed out that scripts to build custom toolchains are also provided with scratchbox allowing more recent GCC versions to be used.

Fedora ARM

Fedora ARM is a try to port Fedora to ARM. It provides some tools as an ARM toolchain packaged in RPM format. Link: Fedora ARM

Debian cross-tools packages

For Debian users, the toolchains problem is fairly reliably solved.

For a Debian-based box, just install pre-built cross toolchains from Debian experimental.

Targets include nearly all Debian-supported architectures. As of this writing the supported compiler is GCC 4.9. You can get older unsupported compilers from emdebian.

You will need to add the target architecture to your list of installable architectures. For example,

```
dpkg --add-architecture armhf
apt-get update
apt-get install gcc-arm-linux-gnueabihf
```

Free Pascal

Free Pascal is available for several processor architectures from http://www.freepascal.org, the Lazarus IDE from http://www.lazarus.freepascal.org.

Toolchain building systems

Buildroot

Buildroot is a complete build system based on the Linux kernel configuration system and supports a wide range of target architectures. It generates root file system images ready to be written to flash. In addition to having a huge number of packages which can be compiled into the image, it also generates a cross toolchain to build those packages from source. Even if you don't want to use buildroot for your root filesystem, it is a useful tool for generating a toolchain. Buildroot supports uClibc-ng, glibc and musl.

OpenADK

OpenADK is a complete build system based on the Linux kernel configuration system and supports a wide range of target architectures. It is similar to buildroot. It generates root file system images ready to be written to flash. In addition to having a huge number of packages which can be compiled into the image, it also generates a cross toolchain to build those packages from source. Even if you don't want to use OpenADK for your root filesystem, it is a useful tool for generating a toolchain. OpenADK supports glibc, uClibc-ng, newlib and musl.

Crossdev (Gentoo)

Crossdev is specific to developers using Gentoo for their development PCs. It is a script which generates a cross toolchain using the portage build scripts for GCC, etc. There are numerous architectures which are supported and both uClibc and glibc toolchains can be built. Link: Gentoo Crossdev info

Crosstool-NG

Crosstool-NG is a well-maintained fork of crosstool, targeted at easier configuration, re-factored code, and a learning base on how toolchains are built, with support for both uClibc and glibc, for debug tools (gdb, strace, dmalloc, etc.), and a wide range of versions for each tools. Different target architectures are supported as well. It offers a kernel-like configuration system to select the different configuration options of the toolchain (component versions, component configuration, etc.). Crosstool-NG has an active and responsive user and developer community.

Crossdev/tsrpm (Timesys)

Crossdev is a project sponsored by Timesys, completely unrelated to the Gentoo cross toolchain generation system. The projects main focus is on a tool called tsrpm which is used to build cross development toolchains and generate cross-compiled software packages. Currently only x86 and select PowerPC architectures are supported. Link: Crossdev

EmbToolkit

EmbToolkit is the first toolchain building system giving the choice to generate GCC or llvm/clang based toolchain.
EmbToolkit supports use of eglibc, glibc or uClibc as C Library and musl C library is also planned at time of writing.
EmbToolkit can be used to generate only a toolchain (usable in a external project), but it is also possible to generate various root filesystems.

OSELAS.Toolchain()

The OSELAS.Toolchain() project aims at supplying a complete build system for recent GNU toolchains. It uses the PTXdist build system, a userland build system based on Kconfig. The current version 1.99.3.1 of OSELAS.Toolchain() contains support for ARM, x86, AVR, MIPS and PowerPC. In addition, there are toolchains for bare metal platforms like Cortex-M3 and AVR-8-Bit.

OSELAS.Toolchain() Feature matrix (v1.99.1, build with PTXdist v1.99.7)

Architeture	CPUtype	gcc	glibc	binutils	kernel header
ARM (eabi)	1136jfs,xscale(-hf),iwmmx,v4t(-hf),v5te,xscale	4.1.2, 4.3.2	2.5, 2.8	2.17, 2.18	2.6.18, 2.6.27
AVR	n/a	3.4.6, 4.1.2, 4.3.2	1.0.5, 1.4.8, 1.6.2	2.17	n/a
x86	i586, i686	4.1.2, 4.3.2	2.5, 2.8	2.17, 2.18	2.6.18, 2.6.27
PowerPC	603e	4.1.2, 4.3.2	2.5, 2.8	2.17, 2.18	2.6.18, 2.6.27
MIPS	mipsel (softfloat)	4.2.3	2.8	2.18	2.6.27

OSELAS.Toolchain() contains some further goodies like gcj support for ARM and MinGW support for x86.
Link: OSELAS.Toolchain() Link: PTXdist

Bitbake

Bitbake is the tool used by OpenEmbedded. The best way to get started is probably by just building an existing distribution that uses openembedded (for example, Ångström, see http://www.angstrom-distribution.org/building-ångström for details).

By platform

ARM

See ARMCompilers

Category:

- Development Tools

Navigation menu

Personal tools

- Log in
- Request account

Namespaces

- Page
- Discussion

Variants

Views