

# Operating Systems And Applications For Embedded Systems

Toolchains

# Plan

## Toolchain

Toolchain

Main component of GNU toolchain

C library

Finding a toolchain

## cross-tool-NG

cross-tool-NG

Installing

Anatomy of a toolchain

Information about cross-compiler

Configuration

Most interesting features

Sysroot

Other tools

POSIX functions AP

# Toolchain

A toolchain is the set of tools that compiles source code into executables that can run on your target device, and includes a compiler, a linker, and run-time libraries.

# Main component of GNU toolchain

- ▶ Binutils: A set of binary utilities including the assembler, and the linker, ld. It is available at <http://www.gnu.org/software/binutils/>.
- ▶ GNU Compiler Collection (GCC): These are the compilers for C and other languages which, depending on the version of GCC, include C++, Objective-C, Objective-C++, Java, Fortran, Ada, and Go. They all use a common back-end which produces assembler code which is fed to the GNU assembler. It is available at <http://gcc.gnu.org/>.
- ▶ C library: A standardized API based on the POSIX specification which is the principle interface to the operating system kernel from applications. There are several C libraries to consider, see the following section.

# C library

- ▶ glibc: Available at <http://www.gnu.org/software/libc>. It is the standard GNU C library. It is big and, until recently, not very configurable, but it is the most complete implementation of the POSIX API.
- ▶ eglibc: Available at <http://www.eglibc.org/home>. This is the embedded GLIBC. It was a series of patches to glibc which added configuration options and support for architectures not covered by glibc (specifically, the PowerPC e500). The split between eglibc and glibc was always rather artificial and, fortunately, the code base from eglibc has been merged back into glibc as of version 2.20, leaving us with one improved library. eglibc is no longer maintained.
- ▶ uClibc: Available at <http://www.uclibc.org>. The 'u' is really a Greek 'mu' character, indicating that this is the micro controller C library. It was first developed to work with uClinux (Linux for CPUs without memory management units), but has since been adapted to be used with full Linux. There is a configuration utility which allows you to fine-tune its features to your needs. Even a full configuration is smaller than glibc but it is not as complete an implementation of the POSIX standards.
- ▶ musl libc: Available at <http://www.musl-libc.org>. It is a new C library designed for embedded systems.

# Finding a toolchain

- ▶ SoC or board vendor. Most vendors offer a Linux toolchain.
- ▶ A consortium dedicated to providing system-level support for a given architecture. For example, Linaro, (<https://www.linaro.org>) have pre-built toolchains for the ARM architecture.
- ▶ Third-party Linux tool vendors such as Mentor Graphics, TimeSys, or MontaVista.
- ▶ Cross tool packages for your desktop Linux distribution, for example, Debian-based distributions have packages for cross compiling for ARM, MIPS, and PowerPC targets.
- ▶ A binary SDK produced by one of the integrated embedded build tools, the Yocto Project has some examples at <http://autobuilder.yoctoproject.org/pub/releases/CURRENT/toolchain> and there is also the Denx Embedded Linux Development Kit at <ftp://ftp.denx.de/pub/eldk/>.
- ▶ A link from a forum that you can't find any more.

# crosstool-NG

I am going to begin with crosstool-NG because it allows you to see the process of creating the toolchain and to create several different sorts.

Some years ago, Dan Kegel wrote a set of scripts and makefiles for generating cross development toolchains and called it crosstool ([kegel.com/crosstool](http://kegel.com/crosstool)). In 2007, Yann E. Morin used that base to create the next generation of crosstool, crosstool-NG ([crosstool-ng.org](http://crosstool-ng.org)).

Today it is, by far, the most convenient way to create a stand alone cross toolchain from source.

# Installing

```
sudo apt-get install automake bison chrpath flex g++ git gperf gawk libexpat1-dev  
libncurses5-dev libsdl1.2-dev libtool python2.7-dev texinfo  
tar xf crosstool-ng-1.20.0.tar.bz2  
cd crosstool-ng-1.20.0  
./configure --enable-local  
make  
make install
```



# Anatomy of a toolchain

```
PATH= /x-tools/arm-cortex_a8-linux-gnueabi/bin:$PATH
arm-cortex_a8-linux-gnueabi-gcc helloworld.c -o helloworld
file helloworld
helloworld: ELF 32-bit LSB executable, ARM, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 3.15.4, not
stripped
```

# Information about cross-compiler

```
arm-cortex_a8-linux-gnueabi-gcc --version
```

```
arm-cortex_a8-linux-gnueabi-gcc (crosstool-NG 1.20.0) 4.9.1
```

```
Copyright (C) 2014 Free Software Foundation, Inc.
```

```
This is free software; see the source for copying conditions. There
```

```
is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR  
PURPOSE.
```

# Configuration I

```
arm-cortex_a8-linux-gnueabi-gcc -v
```

Using built-in specs.

```
COLLECT_GCC=arm-cortex_a8-linux-gnueabi-gcc
```

```
COLLECT_LTO_WRAPPER=/home/chris/x-tools/arm-cortex_a8-  
linux-gnueabi-gcc/libexec/gcc/arm-cortex_a8-linux-gnueabi-gcc/4.9.1/lto-wrapper
```

```
Target: arm-cortex_a8-linux-gnueabi
```

Configured with:

```
/home/chris/hd/home/chris/build/MELP/build/crosstool-ng-  
1.20.0/.build/src/gcc-4.9.1/configure --build=x86_64-build_unknownlinux-gnu  
--host=x86_64-build_unknownlinux-gnu --target=armcortex_a8-linux-gnueabi-gcc  
--prefix=/home/chris/x-tools/arm-cortex_a8-linux-gnueabi-gcc  
--with-sysroot=/home/chris/x-tools/arm-cortex_a8-  
linux-gnueabi-gcc/arm-cortex_a8-linux-gnueabi-gcc/sysroot --enable-languages=c,c++  
--with-arch=armv7-a --with-cpu=cortex-a8 --with-tune=cortex-a8 --with-float=hard  
--with-pkgversion='crosstool-NG 1.20.0' --enable-__cxa_atexit --disable-libmudflap
```

## Configuration II

```
-disable-libgomp -disable-libssp -disable-libquadmath -disable-libquadmath-support  
-disable-lsanitizer -withgmp=/home/chris/hd/home/chris/build/MELP/build/crostoool-ng-  
1.20.0/.build/arm-cortex_a8-linux-gnueabi/f/buildtools  
-withmpfr=/home/chris/hd/home/chris/build/MELP/build/crostoool-ng-  
1.20.0/.build/arm-cortex_a8-linux-gnueabi/f/buildtools  
-withmpc=/home/chris/hd/home/chris/build/MELP/build/crostoool-ng-  
1.20.0/.build/arm-cortex_a8-linux-gnueabi/f/buildtools  
-withisl=/home/chris/hd/home/chris/build/MELP/build/crostoool-ng-  
1.20.0/.build/arm-cortex_a8-linux-gnueabi/f/buildtools  
-withclog=/home/chris/hd/home/chris/build/MELP/build/crostoool-ng-  
1.20.0/.build/arm-cortex_a8-linux-gnueabi/f/buildtools  
-withlibelf=/home/chris/hd/home/chris/build/MELP/build/crostoool-ng-  
1.20.0/.build/arm-cortex_a8-linux-gnueabi/f/buildtools -with-hostlibstdcxx='-static-libgcc  
-Wl,-Bstatic,-lstdc++,-Bdynamic -lm' -enable-threads=posix -enable-target-ospace
```

# Configuration III

```
-enable-plugin - enable-gold -disable-nls -disable-multilib -with-localprefix=/home/chris/x-  
tools/arm-cortex_a8-linux-gnueabi/hf/armcortex_a8-linux-gnueabi/hf/sysroot -enable-c99  
-enable-long-long  
Thread model: posix  
gcc version 4.9.1 (crosstool-NG 1.20.0)
```

# Most interesting features

1. `-with-sysroot=/home/chris/x-tools/arm-cortex_a8-linuxgnueabi/hf/arm-cortex_a8-linux-gnueabi/hf/sysroot`: This is the default sysroot directory, see the following section for an explanation
2. `-enable-languages=c,c++`: Using this we have both C and C++ languages enabled  
`-with-arch=armv7-a`: The code is generated using the ARM v7a instruction set
3. `-with-cpu=cortex-a8` and `-with-tune=cortex-a8`: The the code is further tweaked for a Cortex A8 core
4. `-with-float=hard`: Generate opcodes for the floating point unit and uses the VFP registers for parameters
5. `-enable-threads=posix`: Enable POSIX threads

# Sysroot

- ▶ lib: Contains the shared objects for the C library and the dynamic linker/ loader, ld-linux
- ▶ usr/lib: the static library archives for the C library and any other libraries that may be installed subsequently
- ▶ usr/include: Contains the headers for all the libraries
- ▶ usr/bin: Contains the utility programs that run on the target, such as the ldd command
- ▶ /usr/share: Used for localization and internationalization
- ▶ sbin: Provides the ldconfig utility, used to optimize library loading paths

# Other tools I

Tablica: Tablica

Command	Description
addr2line	Converts program addresses into filenames and numbers by reading the debug symbol tables in an executable file. It is very useful when decoding addresses printed out in a system crash report.
ar	The archive utility is used to create static libraries.
as	This is the GNU assembler.
c++filt	This is used to demangle C++ and Java symbols.
cpp	This is the C preprocessor, and is used to expand #define, #include, and other similar directives. You seldom need to use this by itself. elfedit This is used to update the ELF header of ELF files.
g++	This is the GNU C++ front-end, which assumes source files contain C++ code.
gcc	This is the GNU C front-end, which assumes source files contain C code.



## Other tools II

gcov	This is a code coverage tool.
gdb	This is the GNU debugger.
gprof	This is a program profiling tool.
ld	This is the GNU linker.
nm	This lists symbols from object files.
objcopy	This is used to copy and translate object files.
objdump	This is used to display information from object files.
ranlib	This creates or modifies an index in a static library, making the linking stage faster.
readelf	This displays information about files in ELF object format.
size	This lists section sizes and the total size.
strings	This display strings of printable characters in files.
strip	This is used to strip an object file of debug symbol tables, thus making it smaller. Typically, you would strip all the executable code that is put onto the target.

## Other tools III

# POSIX functions AP

- ▶ `libc`: The main C library that contains the well-known POSIX functions such as `printf`, `open`, `close`, `read`, `write`, and so on
- ▶ `libm`: Maths functions such as `cos`, `exp`, and `log`
- ▶ `libpthread`: All the POSIX thread functions with names beginning with `pthread_`
- ▶ `librt`: The real-time extensions to POSIX, including shared memory and asynchronous I/O

```
arm-cortex_a8-linux-gnueabihf-readelf -a myprog | grep "Shared library"  
0x00000001 (NEEDED) Shared library: [libm.so.6]  
0x00000001 (NEEDED) Shared library: [libc.so.6]
```

# References



C. Simmonds.

*Mastering Embedded Linux Programming.*

Packt Publishing, 2015.

# The End