Scratchbox toolchains

Ricardo Kekki

rkekki@movial.fi

Scratchbox toolchains

by Ricardo Kekki

Revision history

Version:	Author:	Description:
2005-03-17	Kekki	Scratchbox 0.9.8 toolchain build system update
2005-02-17	Savola	Fixed build option example
2005-02-07	Savola	Scratchbox 1.0 updates
2004-09-07	Kekki	Initial version

Table of Contents

Chapter 1. Introduction

Toolchain is a collection of tools used to develop software for a certain hardware target. Toolchains are based on particular versions of compiler, libraries, special headers and other tools. A cross-toolchain is a toolchain for compiling binaries for different CPU architecture than the host CPU.

Scratchbox is a cross-compilation toolkit for embedded Linux application development. It is designed for compiling software for different target CPU architectures. Scratchbox allows creating several target environments. Each target is a separate environment that has a selected toolchain, target CPU and an own file system.

Building toolchains is not always trivial. Scratchbox uses scripts for building predefined toolchains. After building a toolchain it should be tested to know that it works properly. Testing toolchains is harder than building toolchains.

Chapter 2. Scratchbox toolchains

2.1. General

Scratchbox toolchains provide the cross compilation tools for compiling binaries for the target environment. Each toolchain is built for a certain CPU target and they are based on certain gcc and C-library sources.

Scratchbox toolchains can be used both inside and outside Scratchbox. In Scratchbox each target uses a certain toolchain with a specific target CPU. Scratchbox uses wrappers to make the toolchains appear as if they were native toolchains. Outside Scratchbox the toolchains are used as any normal cross compilation toolchains.

All the installed toolchains can be found in the `/scratchbox/compilers` directory.

2.2. Toolchain design

The toolchains use either glibc or uClibc C-libraries. The glibc toolchain is based on the Debian patched glibc-2.3.2. Other packages used for building the toolchains are binutils, gcc and linux kernel headers. Several patches are used when building the toolchains. The patching depends of the used toolchains configurations and used packages. eg. when using uClibc then binutils and gcc are patched with uClibc patches.

Scratchbox.org [1] offers prebuilt toolchains for x86 and ARM targets. Only these targets are currently supported because too much work to support all different configurations. With the Scratchbox toolchain sources you can build your own custom toolchains.

Scratchbox toolchain building scripts allow you to build easily predefined toolchains and give the possibility to build custom toolchains for different targets. Changing the toolchain binutils, compiler or C-library packages can be a more demanding task.

2.3. Wrappers

Scratchbox uses a *gcc wrapper* for wrapping most of the toolchain command binaries. In the `/scratchbox/compilers/bin/` directory you can see the linked binaries. The wrapper knows how to handle each command and depending on the command it might change some of the command parameters. Then it runs the actual command from the correct path that depends from the selected target.

The gcc wrapper reads all the target specific information from the target configuration files that can be found in the `/scratchbox/users/username/targets/` directory.

Scratchbox has also an *ld wrapper* for linking the binaries properly. When compiling inside Scratchbox a *fake-native* ld is used. Outside Scratchbox a normally behaving ld is used. For example inside Scratchbox the dynamically linked binaries are linked against the libraries that are in standard library paths, not in the toolchain's library path.

The gcc wrapper uses ccache by default. The default cache directory is `/scratchbox/ccache/`. Ccache can be disabled by setting the environment variable SBOX_USE_CCACHE to "no". The cache directory can be changed with the CCACHE_DIR environment variable.

2.4. Toolchain installation

There are several binary packages available. The original toolchains:

- scratchbox-toolchain-arm--glibc
- scratchbox-toolchain-arm-uclibc
- scratchbox-toolchain-i686-glibc
- scratchbox-toolchain-i386-uclibc

Additional toolchains:

- scratchbox-toolchain-arm-cs-glibc
- scratchbox-toolchain-arm-cs-glibc-nofpu
- scratchbox-toolchain-arm-cs-uclibc
- scratchbox-toolchain-arm-cs-uclibc-nofpu

For each toolchain that you want to use you need to create a Scratchbox target. For installation and target creation instructions see *Installing Scratchbox* [2].

Chapter 3. Building custom toolchains

3.1. Toolchain build system

Scratchbox toolchain build scripts are available in the *sb-toolchains* source package that is available at the Scratchbox download page [1] for each Scratchbox version.

Scratchbox toolchains are built with the GAR system [3]. It is a mechanism for automating the compilation and installation of third-party source code. It appears in the form of a tree of directories containing Makefiles and other ancillary bookkeeping files (such as installation manifests and checksum lists).

The toolchains can currently be built with two different systems. The old one uses the `sb-toolchains/meta/target-kit` directory. When the new system uses the `sb-toolchains/meta/gcc-glibc` directory.

The old system allows only selecting the target architecture, compiler name and the build directory. It doesn't select the used source packages or patched that are applied. The new toolchain build system is far more flexible. It uses configuration files for building toolchains. The configuration file is passed to the meta makefile as a variable.

The new build system uses separate build directories for separate toolchain build phases. This makes it's more flexible to change build components in the toolchain build. eg. changing gcc's configuration arguments without breaking the the build system for other toolchains. This would be done by creating a new build directory under the `sb-toolchains/cc` directory and changing the CC_DIR variable in the config file.

The sb-toolchains source package is used also to build *arch tools* and *device tools* which are target dependent. The build instructions in this document automatically build all of them; there is currently no documented way to disable that.

3.2. Environment

Toolchains have to be built with an already existing compiler. Scratchbox has the HOST target for this. HOST's host-gcc toolchain is for compiling binaries for the host. It's configured to make the compiled binaries use the Scratchbox's host libraries.

When building toolchains you need write privileges to the `/scratchbox/compilers` and `/scratchbox/device_tools` directories. The *scratchbox* source package contains the

`scripts/permhack` script for changing the privileges, but you can also do it like this:

```
# chown -R username:groupname /scratchbox/compilers/
# chown -R username:groupname /scratchbox/device_tools/
```

Building Debian patched toolchains needs the Debian devkit package to be installed; the build script uses the dpatch and dpkg commands.

3.3. Compiling toolchains

The following will explain toolchain building with the new system.

The toolchain configuration file includes information like compiler name, target architechture, softfloat support and all source package and patch information. For example see the existing configuration files in the `sb-toolchains/meta/gcc-glibc` directory.

For each toolchain component there are the several variables. eg. for the C-library:

`LIBC_VER`
: version number of the C-library

`LIBC_HEADERS_DIR`
: the build directory of the C-library headers

`LIBC_DIR`
: the build directory of the C-library

`LIBC`
: the C-library source tar

`LIBC_PATCHES`
: the patches that are applied to the source tar

`LIBC_PATCH_SCRIPT`
: a special script that can be used to apply complex patches

`LIBC_SCRIPT_FILE`
: the patch file for the patch script

example of arm-gcc3.4.cs-glibc.conf:

```
LIBC_VER=2.3.2
```

```
LIBC_HEADERS_DIR=glibc-headers
LIBC_DIR=glibc-2.3
LIBC=glibc_2.3.2.ds1.orig.tar.gz
LIBC_PATCHES=glibc-2.3.2-debian.patch glibc-arm-mcount_internal.patch
LIBC_PATCH_SCRIPT=undeb_glibc.sh
LIBC_SCRIPT_FILE=glibc_2.3.2.ds1-20.diff.gz
```

The actual building:

1. Make sure you have the scratchbox-devkit-debian package installed on your system.
2. Download the sb-toolchains source package (see Section 3.1). Extract it into a directory that is visible inside Scratchbox (such as `/scratchbox/users/username/home/username`).
3. Change the permissions of the Scratchbox installation directories (see Section 3.2).
4. Login to Scratchbox.
5. export the debian devkit directory to path

   ```
   [sbox-HOST: ~] >  export PATH=/scratchbox/devkits/debian/bin/:$PATH
   ```

6. Change to the sb-toolchains directory

   ```
   [sbox-HOST: ~] > cd sb-toolchains
   ```

7. Create a new configuration file for your toolchain. See the existing ones for example.
8. If you are using sources packages that are not available in http://scratchbox.org/download/files/sbox-files/ then you should create the packages directory:

   ```
   [sbox-HOST: ~] > mkdir packages
   ```

   And copy your new source packages there.
9. Change to the meta directory.

   ```
   [sbox-HOST: ~/sb-toolchains] > cd meta/gcc-glibc
   ```

10. For all new source packages the checksums have to be added to the existing checksum files. Each build directory has it's own checksum file. eg. libc/glibc-2.3/checksums. The checksum is a md5 sum.
11. Build the toolchain

    ```
    [sbox-HOST: ~/sb-toolchains/meta/gcc-glibc] > make CONFIG=<your-config-file> clean
    [sbox-HOST: ~/sb-toolchains/meta/gcc-glibc] > make CONFIG=<your-config-file>
    ```

12. For buildin a toolchain with an existing configuration file

    ```
    [sbox-HOST: ~/sb-toolchains/meta/gcc-glibc] > make CONFIG=meta/gcc-glibc/arm-gcc3.4.cs-g
    [sbox-HOST: ~/sb-toolchains/meta/gcc-glibc] > make CONFIG=meta/gcc-glibc/arm-gcc3.4.cs-g
    ```

13. After the toolchain is built also the device tools and arch tools need to be built.

    ```
    [sbox-HOST: ~/sb-toolchains/meta/gcc-glibc] > make CONFIG=meta/gcc-glibc/arm-gcc3.4.cs-g
    ```

3.4. Building binary packages

The sb-toolchains/packaging/create_packages script generates deb and rpm packages from the compiled toolchains. If you have made changes to the packages then fix the version numbers in the sb-toolchains/packaging/common.var file. At the moment the script supports only creating arm, ppc and x86 toolchain packages.

For packaging one toolchain use the ./build_one_toolchain script. Run it outside Scratchbox so it can also create RPMs.

Chapter 4. Testing toolchains

4.1. Initial tests

All created compilers should be at least tested to compile, link and run a simple test program. The test program should be tested to link both statically and dynamically. These tests should be done with both a C and a C++ program.

The `sb-toolchains/test_tools/test_scripts/reg_tests.sh` script can be used for these initial tests.

4.2. Regression tests

These tests are a collection of test for the C and C++ frontends of gcc. The tests can be found in gcc's directory gcc-VERSION/gcc/testsuite.

The result of running the testsuite are various *.sum and *.log files in the testsuite subdirectories. The *.log files contain a detailed log of the compiler invocations and the corresponding results, the *.sum files summarise the results. These summaries contain status codes for all tests:

- PASS: the test passed as expected
- XPASS: the test unexpectedly passed
- FAIL: the test unexpectedly failed
- XFAIL: the test failed as expected
- UNSUPPORTED: the test is not supported on this platform
- ERROR: the testsuite detected an error
- WARNING: the testsuite detected a possible problem

The Scratchbox toolchains should be tested with these tests. These tests can be run with reg_tests.sh (see Section 4.1) or manually as described in *GCC Testing* [5].

References

[1] *Scratchbox download page (http://scratchbox.org/download/)* .

[2] *Installing Scratchbox (http://scratchbox.org/documentation/docbook/installdoc.html)* , Valtteri Rahkonen.

[3] *GAR Architecture (http://www.lnx-bbc.org/garchitecture.html)* .

[4] *Scratchbox development team contacts (http://scratchbox.org/contact/)* .

[5] *GCC Testing (http://gcc.gnu.org/install/test.html)* .