The Build Process

of (GNU Tools for ARM Embedded Processors)

2013-03

Table of Contents

Preface

This manual provides a step-by-step guide to help you build 'GNU Tools for ARM Embedded Processors' on a newly installed Ubuntu 8.10 operating system. But some of the steps can be ignored,

- If you don't want to build the package running on windows, please ignore Section 1.2.4 [Install MinGW and InstallJammer], page 4.
- It is possible to have the package built successfully if you skip Section 1.4 [Build new native gcc], page 7, but we suggest you not skip the step.

Note that the steps below may most likely also work on an Ubuntu which is not newly installed or version other than 8.10, but it is not guaranteed. In this case please go through ⟨undefined⟩ [Known Issues], page ⟨undefined⟩ before you go, and you need to solve any other problems you may encounter by yourself. We highly appreciate if you could share the problems and solutions with us.

1 Build GNU Tools on Ubuntu 8.10

1.1 Install Ubuntu

Ubuntu 8.10 ISO image is available from `http://old-releases.ubuntu.com/releases/8.10/ubuntu-8.10-desktop-i386.iso`. You can install it as a native system or a virtual machine. The command lines provided in this document are all using user id 'build' as an example, so please create a new user called 'build' in the system. Otherwise, you have to replace user id 'build' with your own one.

1.2 Tune environment and install required softwares

1.2.1 Change /bin/sh to bash

Some shell scripts in gcc and other packages are incompatible with the dash shell, which is the default /bin/sh for Ubuntu 8.10. You must make /bin/sh a symbolic link to one of the supported shells: saying bash. Here on Ubuntu 8.10 system, this can be done by running following command firstly:

```
$ sudo dpkg-reconfigure -plow dash
```

Then choose 'No' in the 'Configuring dash' popup dialog and press enter. You can run following command and check that /bin/sh points to 'bash':

```
$ ls -l /bin/sh
...... /bin/sh -> bash
```

1.2.2 Change software sources to Main server

On Ubuntu 8.10 system, click 'System->Administration->Software Sources' to open 'Software Sources' dialog, choose 'Main server' in 'Download from:' list box, then click 'close'. You will be prompted by a window saying 'The information about available software is out-of-date', please click 'Reload'. And then there will be a warning message box popped up, which can be just ignored by clicking 'Close'.
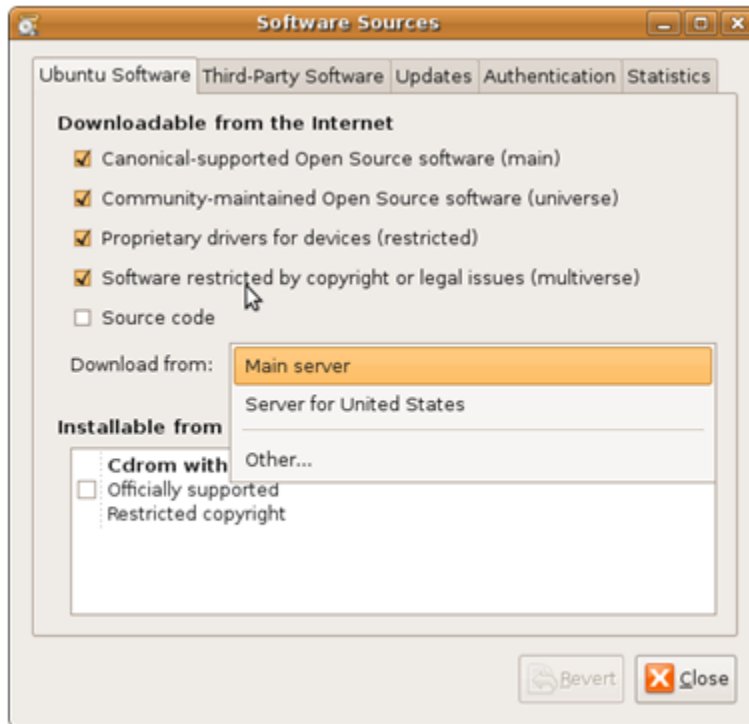
Edit the file using command line:

```
$ sudo vi /etc/apt/sources.list
```

replace all 'http://*.ubuntu.com' with 'http://old-releases.ubuntu.com' in that file, save and exit. Run following command to update package list. It should not fail, or else something has been wrong.

```
$ sudo apt-get update
```

1.2.3 Install common tools and libraries

Install common tools and libraries needed by build process with below command:

```
$ sudo apt-get install apt-src \
gawk \
gzip \
perl \
autoconf \
m4 \
automake \
libtool \
libncurses5-dev \
gettext \
gperf \
dejagnu \
expect \
tcl \
autogen \
guile-1.6 \
flex \
flip \
bison \
tofrodos \
texinfo \
g++ \
gcc-multilib \
mingw32 \
mingw32-binutils \
mingw32-runtime \
libgmp3-dev \
libmpfr-dev \
debhelper \
texlive \
texlive-extra-utils
```

Note that the package management software might complain that several packages cannot be installed properly while installing texlive and texlive-extra-utils. It won't harm our building process, please just ignore it now. Some of those tools might be unnecessary, but it won't hurt if installed.

1.2.4 Install MinGW and InstallJammer

If you don't want to build package running on windows, please skip this step.

1.2.4.1 Download mingw32 and related

Mingw32 source and patch can be downloaded from following links:

```
* mingw32_4.2.1.dfsg-2ubuntu1.diff.gz
```
        https://launchpad.net/ubuntu/maverick/+source/mingw32
        /4.2.1.dfsg-2ubuntu1/+files/mingw32_
        4.2.1.dfsg-2ubuntu1.diff.gz

* `mingw32_4.2.1.dfsg.orig.tar.gz`
  `https://answers.launchpad.net/ubuntu/+archive/primary`
  `/+files/mingw32_4.2.1.dfsg.orig.tar.gz`

* `mingw32_4.2.1.dfsg-2ubuntu1.dsc`
  `https://answers.launchpad.net/ubuntu/+archive/primary`
  `/+files/mingw32_4.2.1.dfsg-2ubuntu1.dsc`

Save these files in '`~/tools/mingw32/`' directory. Download the mingw32-binutils source and patch from following links:

* `mingw32-binutils_2.20-0.1.diff.gz`
  `https://launchpad.net/ubuntu/+archive/primary/+files`
  `/mingw32-binutils_2.20-0.1.diff.gz`

* `mingw32-binutils_2.20-0.1.dsc`
  `https://launchpad.net/ubuntu/+archive/primary/+files`
  `/mingw32-binutils_2.20-0.1.dsc`

* `mingw32-binutils_2.20.orig.tar.gz`
  `https://launchpad.net/ubuntu/+archive/primary/+files`
  `/mingw32-binutils_2.20.orig.tar.gz`

Save these files in '`~/tools/mingw32-binutils/`' directory. Download the mingw32-runtime source and patch from following links:

* `mingw32-runtime_3.15.2-0ubuntu1.diff.gz`
  `https://launchpad.net/ubuntu/+archive/primary/+files`
  `/mingw32-runtime_3.15.2-0ubuntu1.diff.gz`

* `mingw32-runtime_3.15.2-0ubuntu1.dsc`
  `https://launchpad.net/ubuntu/+archive/primary/+files`
  `/mingw32-runtime_3.15.2-0ubuntu1.dsc`

* `mingw32-runtime_3.15.2.orig.tar.gz`
  `https://launchpad.net/ubuntu/+archive/primary/+files`
  `/mingw32-runtime_3.15.2.orig.tar.gz`

Save these files in '`~/tools/mingw32-runtime/`' directory.

> **Note:** In this step, the '`*.diff.gz`' files are compressed patch files. But if you download these packages using internet explorer (such as IE, FireFox, etc.), it might automatically help you unzip these files. Please check the downloaded files using following commands:
>
> ```
> #Here using mingw32_4.2.1.dfsg-2ubuntu1.diff.gz as an example,
> #the same story stands for all *.diff.gz files
>
> $ cd ~/tools/mingw32/
> $ file mingw32_4.2.1.dfsg-2ubuntu1.diff.gz
> mingw32_4.2.1.dfsg-2ubuntu1.diff.gz: ASCII English text
> ```
>
> It says that '`mingw32_4.2.1.dfsg-2ubuntu1.diff.gz`' is an ASCII text file, though file name has '`gz`' suffix. In this case, you could choose either following methods:

- Download the packages using "wget" command;
- Execute following commands to prepare the diff file:

```
$ cd ~/tools/mingw32/
$ mv mingw32_4.2.1.dfsg-2ubuntu1.diff.gz mingw32_4.2.1.dfsg-
2ubuntu1.diff
```

Using this method, there is no need to unzip '`*.diff.gz`' in the next step.

1.2.4.2 Update/Build/Install mingw32 and related

Run the following commands to update mingw32 package

```
$ cd ~/tools/mingw32/
$ ls
mingw32_4.2.1.dfsg-2ubuntu1.diff.gz mingw32_4.2.1.dfsg.orig.tar.gz
mingw32_4.2.1.dfsg-2ubuntu1.dsc
$ tar xzf mingw32_4.2.1.dfsg.orig.tar.gz
$ gzip -d mingw32_4.2.1.dfsg-2ubuntu1.diff.gz
$ mv mingw32-4.2.1.dfsg.orig  mingw32-4.2.1.dfsg
$ cd mingw32-4.2.1.dfsg
$ patch -p1 <../mingw32_4.2.1.dfsg-2ubuntu1.diff
$ chmod a+x ./debian/rules
$ sudo dpkg-buildpackage
```

Everything should be ok and debian package '`mingw32_4.2.1.dfsg-2ubuntu1_i386.deb`' should be generated in '`~/mingw32/`' directory.

Install the generated package:

```
$ cd ~/tools/mingw32/
$ sudo dpkg -i mingw32_4.2.1.dfsg-2ubuntu1_i386.deb
```

Run the following commands to update mingw32-binutils package:

```
$ cd ~/tools/mingw32-binutils/
$ ls
mingw32-binutils_2.20-0.1.diff.gz  mingw32-binutils_2.20-0.1.dsc
mingw32-binutils_2.20.orig.tar.gz
$ tar xzf mingw32-binutils_2.20.orig.tar.gz
$ gzip -d mingw32-binutils_2.20-0.1.diff.gz
$ cd mingw32-binutils-2.20
$ patch -p1 <../mingw32-binutils_2.20-0.1.diff
$ chmod a+x ./debian/rules
$ sudo dpkg-buildpackage
```

Install the generated package:

```
$ cd ~/tools/mingw32-binutils/
$ sudo dpkg -i mingw32-binutils_2.20-0.1_i386.deb
```

Run the following commands to update mingw32-runtime package:

```
$ cd ~/tools/mingw32-runtime/
$ ls
mingw32-runtime_3.15.2-0ubuntu1.diff.gz
mingw32-runtime_3.15.2-0ubuntu1.dsc
mingw32-runtime_3.15.2.orig.tar.gz
```

```
$ tar xzf mingw32-runtime_3.15.2.orig.tar.gz
$ gzip -d mingw32-runtime_3.15.2-0ubuntu1.diff.gz
$ cd mingw32-runtime-3.15
$ patch -p1 <../mingw32-runtime_3.15.2-0ubuntu1.diff
$ chmod a+x ./debian/rules
$ sudo dpkg-buildpackage
```

Install the generated package:

```
$ cd ~/tools/mingw32-runtime/
$ sudo dpkg -i mingw32-runtime_3.15.2-0ubuntu1_all.deb
```

1.3 Install InstallJammer

Download latest InstallJammer installer (installjammer-1.2.15.tar.gz) from website: http://www.installjammer.com/. Save it in '~/tools' directory, unpack it by running following command:

```
$ cd ~/tools
$ tar xzf installjammer-1.2.15.tar.gz
```

After unpacking, find the executable program named installjammer in the destination directory, for example '~/tools/installjammer'. Export the path in PATH environment variable by appending following lines at the end of '~/.bashrc' file:

```
#For example, $YOUR_INSTALLJAMMER_PATH=/home/build/tools/installjammer
export PATH=$YOUR_INSTALLJAMMER_PATH:$PATH
```

Save '.bashrc' and quit, then restart or re-login your system. Check that installjammer is in your '$PATH' by running below command:

```
$ which installjammer
$YOUR_INSTALLJAMMER_PATH/installjammer
#in my system, the output would be:
#/home/build/tools/installjammer/installjammer
```

1.4 Build new native gcc

> **note:** If you don't want to build the package running on the platform which is different from your build machine, please skip this step.

Reconfigure and compile native gcc/g++ on Ubuntu 8.10 system. Gcc distributed along with Ubuntu 8.10 system supports ssp utility by default, which uses symbols of version **GLIBC_2.8/GLIBC_2.7**, etc. In glibc, resulting in programs compiled by this gcc/g++ may not be able to run on system with lower version glibc, like redhat5/redhat4. To provide greater application coverage of our toolchain, we have to reconfigure and compile gcc with libssp disabled.

Download 'gcc-4.3.6.tar.bz2' from website and save it in '~/tools/native-gcc/src' directory. The website is ftp://ftp.gnu.org/gnu/gcc/gcc-4.3.6/gcc-4.3.6.tar.bz2.

Reconfigure and compile gcc with following commands:

```
$ cd ~/tools/native-gcc/src
$ mkdir -p ~/tools/native-gcc/obj/gcc && mkdir ~/tools/native-gcc/target
$ tar -xjf gcc-4.3.6.tar.bz2
$ cd ../obj/gcc
$ ../../src/gcc-4.3.6/configure  \
--build=i686-linux-gnu  \
--host=i686-linux-gnu  \
--target=i686-linux-gnu  \
--enable-languages=c,c++  \
--enable-shared   \
--enable-threads=posix   \
--disable-decimal-float  \
--disable-libffi  \
--disable-libgomp  \
--disable-libmudflap   \
--disable-libssp  \
--disable-libstdcxx-pch  \
--disable-multilib  \
--disable-nls    \
--with-gnu-as   \
--with-gnu-ld     \
--enable-libstdcxx-debug   \
--enable-targets=all  \
--enable-checking=release  \
--prefix=/home/build/tools/native-gcc/target \
--with-host-libstdcxx="-static-libgcc -L /usr/lib/gcc/i486-linuxgnu/4.3.2/ -
lstdc++ -lsupc++ -lm"
$ make
$ make install
```

After this, add path '**/home/build/tools/native-gcc/target/bin**' into
your PATH environment, make sure to use the new gcc/g++ to compile our
arm toolchain. You can do this by appending following line at the end of
'**~/.bashrc**' file:

```
export PATH=/home/build/tools/native-gcc/target/bin:$PATH
```

Then re-start or re-login the system.

1.5 Build GNU Tools for ARM Embedded Processors

You have set up the building environment. You can now build the toolchain by yourself with below commands:

```
#Copy the src release package into ~/toolchain/ directory
$ cp gcc-arm-none-eabi-4_7-2013q1-20130313-src.tar.bz2 ~/toolchain
#Prepare source codes
$ cd ~/toolchain
$ tar -xjf gcc-arm-none-eabi-4_7-2013q1-20130313-src.tar.bz2
$ cd ./gcc-arm-none-eabi-4_7-2013q1-20130313/src
$ find -name '*.tar.*' | xargs -I% tar -xf %
#Since we should not modify zlib package,
#here provides a patch to compile it successfully.
$ cd zlib-1.2.5
$ patch -p1 <../zlib-1.2.5.patch
$ cd ../../
#Start building the toolchain.
#Can specify "--skip_mingw32" option to skip building windows host
#toolchain, and if specify that option when building prerequisites,
#you have to specify it when building toolchain too.
$ ./build-prerequisites.sh [--skip_mingw32]
$ ./build-toolchain.sh [--skip_mingw32]
```

After this, you can 'cd' into
'~/toolchain/gcc-arm-none-eabi-4_7-2013q1-20130313/pkg' and find
the built toolchain/source code packages and the md5 checksum file.

2 Build GNU Tools on Mac OS X

In addition to the build on Ubuntu, the build scripts in same source package can also be used on Mac OS X to natively build a tool chain whose host is Mac OS X and target is arm-none-eabi. In this step we will describe how to install required software components and how to execute the build scripts. After this step you should be able to generate a same tool chain with the one released. Due to resource limit, this build process is only tested against Mac OS X 10.7.4 along with components listed below.

2.1 Prepare a Mac OS X environment

The hardware should be an x86-based Mac machine like iMac. The installed OS should be Mac OS X which is updated to 10.7.4. The way to find out the Mac OS X version information is to click the **Apple** menu and choose **About This Mac**.
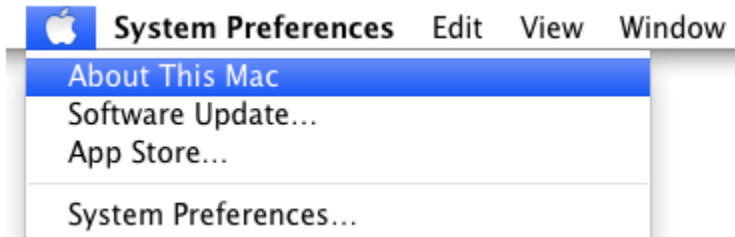
For the environment we are using, it looks as below:

2.2 Install the Command Line Tools for Xcode

This component is originally part of Apple Xcode but can be installed separately without Xcode. It can be freely obtained from Apple official website https://developer.apple.com/downloads/index.action. A valid Apple ID is required to login and download. The one we are using is in the item named 'Command Line Tools for Xcode – June 2012'. After finish the download, just double click the '.dmg' file and follow the instructions to install it.

2.3 Set up the brew to install other required components

In order to handle GNU configure and build system, we have to install some components like automake/autoconf through brew which can be easily set up from its official website http://mxcl.github.com/homebrew/. After set up the brew, use below commands to install required components:

```
brew install automake autoconf libtool coreutils gnu-tar md5sha1sum
```

2.4 Install MacTeX to build PDF format documents

This is an optional step and can be skipped if PDF format documents aren't needed. The build process will use TeX engineer provided by MacTeX-2012 to generate PDF format documents. This component can be freely obtained from its official website http://www.tug.org/mactex/2012/. Its original size is approximately 2.1G. After download, just double click the 'MacTeX.pkg' file and follow the instructions to install it. By default the related TeX executable files won't be installed into default path like '/usr/bin', so we need restart the Terminal before we run the build scripts.

2.5 Build the tool chain under Mac OS X

With all the dependent packages installed, we can start to natively build tool chain on Mac OS. Following are commands and steps we are using:

```
#Copy the src release package into ~/mac-build/ directory
$ cp gcc-arm-none-eabi-4_7-2013q1-20130313-src.tar.bz2 ~/mac-build

#Prepare source codes
$ cd ~/mac-build
$ tar xjf gcc-arm-none-eabi-4_7-2013q1-20130313-src.tar.bz2
$ cd ./gcc-arm-none-eabi-4_7-2013q1-20130313/src
$ find . -name '*.tar.*' | xargs -I% tar -xf %

#Since we should not modify zlib package,
#here provides a patch to compile it successfully.
$ cd zlib-1.2.5
$ patch -p1 <../zlib-1.2.5.patch
$ cd ../../

#Start building the toolchain.
$ ./build-prerequisites.sh
$ ./build-toolchain.sh
```

Appendix A Known Issues

- This document and build scripts sometimes have problem when running on ubuntu 9.10 due to bug in 'tar-1.22-1' package. You can update 'tar' to higher version. For more information, please refer to https://bugs.launchpad.net/ubuntu/+source/tar/+bug/453330.
- Due to binutils bug 13036, native gcc with version higher than 4.5.1 won't build binutils successfully. You should note this issue when trying to build the cross-toolchain in your specific host environment. For more information, please refer to http://sourceware.org/bugzilla/show_bug.cgi?id=13036.
- On system other than Ubuntu 8.10, you should change the directory '/usr/lib/gcc/i486-linuxgnu/4.3.2/' in Section 1.4 [Build new native gcc], page 7 to the directory conaining 'libstdc++.a' on your build system.