The Build Process

of (GNU Tools for ARM Embedded Processors)
2014-03

Table of Contents

Preface **1**

1 Build GNU Tools on Ubuntu 8.10 ..... **2**

1.1 Install Ubuntu 2
1.2 Tune environment and install required softwares 2
1.2.1 Change /bin/sh to bash 2
1.2.2 Change software sources to Main server 3
1.2.3 Install common tools and libraries 4
1.2.4 Install GCC 4.6/MinGW-w64/InstallJammer 4
1.2.5 Make new build tools take effective 7
1.3 Build GNU Tools for ARM Embedded Processors 8

2 Build GNU Tools on Mac OS X **9**

2.1 Prepare a Mac OS X environment 9
2.2 Install the Command Line Tools for Xcode 10
2.3 Install MacTeX to build PDF format documents 11
2.4 Build the tool chain under Mac OS X 11

Appendix A Known Issues **12**

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 certain steps in Section 1.2.4 [Install GCC 4.6/MinGW-w64/InstallJammer], page 4.
- It is possible to have the package built successfully if you skip all steps in Section 1.2.4 [Install GCC 4.6/MinGW-w64/InstallJammer], page 4, but we suggest you not skip them all.

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 Appendix A [Known Issues], page 12 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:

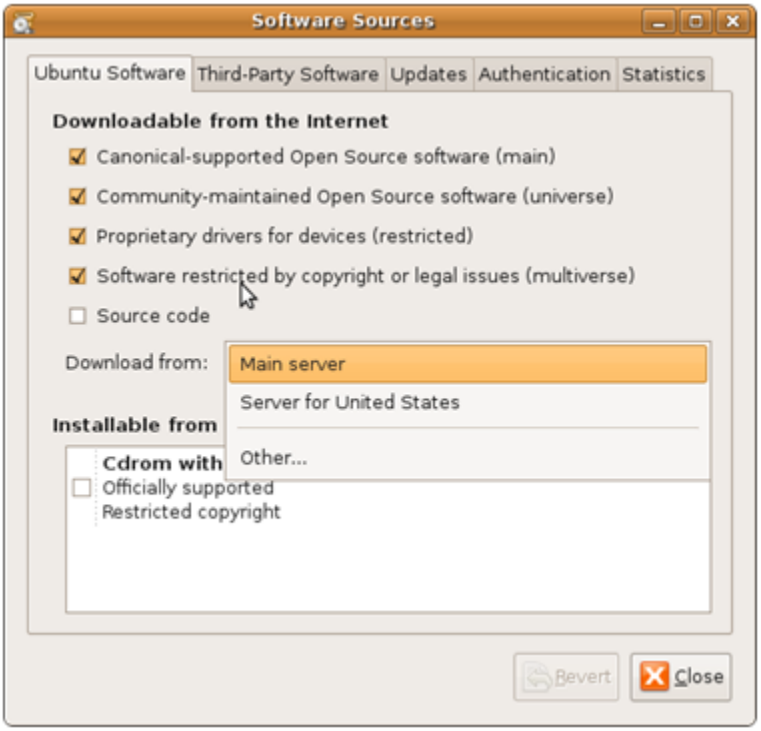
```
$ sudo dpkg-reconfigure -plow dash
```

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 \
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 GCC 4.6/MinGW-w64/InstallJammer

The default native GCC and MinGW build environment in Ubuntu 8.10 are kind of outdated and error prone to build ARM embedded tool chain that is based on GCC 4.8 and newer. This section shows the shell script used to build newer native GCC and MinGW from their own upstream sources.

```
#!/bin/sh

NATIVE_TOOLS_DIR=~/native-tools
NATIVE_TOOLS_SRC=$NATIVE_TOOLS_DIR/src
NATIVE_TOOLS_BLD=$NATIVE_TOOLS_DIR/build
NATIVE_GCC=$NATIVE_TOOLS_DIR/native-gcc
NATIVE_MINGW_W64_GCC=$NATIVE_TOOLS_DIR/native-mingw-w64-gcc
```

```
#Get upstream source packages.
rm -rf $NATIVE_TOOLS_SRC
mkdir -p $NATIVE_TOOLS_SRC
pushd $NATIVE_TOOLS_SRC
wget -c http://ftp.gnu.org/gnu/binutils/binutils-2.23.2.tar.gz
wget -c ftp://ftp.gnu.org/gnu/gcc/gcc-4.6.4/gcc-4.6.4.tar.bz2
wget -c ftp://gcc.gnu.org/pub/gcc/infrastructure/gmp-4.3.2.tar.bz2
wget -c ftp://gcc.gnu.org/pub/gcc/infrastructure/mpfr-2.4.2.tar.bz2
wget -c ftp://gcc.gnu.org/pub/gcc/infrastructure/mpc-0.8.1.tar.gz

wget -c http://downloads.sourceforge.net/project/mingw-w64/mingw-w64/\
mingw-w64-release/mingw-w64-v3.1.0.tar.bz2

find .  -maxdepth 1 -name "*.tar.*" -exec tar xf '{}' \;
cd gcc-4.6.4
ln -s ../gmp-4.3.2 gmp
ln -s ../mpfr-2.4.2 mpfr
ln -s ../mpc-0.8.1 mpc
popd

#Build native gcc
mkdir -p $NATIVE_TOOLS_BLD/native-gcc
pushd $NATIVE_TOOLS_BLD/native-gcc
$NATIVE_TOOLS_SRC/gcc-4.6.4/configure \
          --build=i686-linux-gnu \
          --host=i686-linux-gnu \
          --target=i686-linux-gnu \
          --enable-languages=c,c++ \
          --enable-shared     \
          --enable-threads=posix   \
          --disable-decimal-float \
          --disable-libffi \
          --disable-libgomp \
          --disable-libmudflap   \
          --disable-libssp \
          --disable-libstdcxx-pch \
          --disable-multilib \
          --disable-bootstrap \
          --disable-nls   \
          --with-gnu-as \
          --with-gnu-ld     \
          --enable-libstdcxx-debug   \
          --enable-targets=all \
          --enable-checking=release \
          --prefix=$NATIVE_TOOLS_DIR/native-gcc \
          --with-host-libstdcxx="-static-libgcc \
          -L /usr/lib/gcc/i486-linuxgnu/4.3.2/ -lstdc++ -lsupc++ -lm"

make && make install
popd

export PATH=$NATIVE_GCC/bin:$PATH
```

```
echo "export PATH=$NATIVE_GCC/bin:\$PATH" >> ~/.bashrc

#If no intention to generate ARM embedded toolchain for Windows platform,
#then following steps are unnecessary and we can quit now.

#Start to build native MinGW from building MinGW Binutils.
mkdir -p $NATIVE_TOOLS_BLD/binutils
pushd $NATIVE_TOOLS_BLD/binutils
$NATIVE_TOOLS_SRC/binutils-2.23.2/configure \
          --target=i686-w64-mingw32 --disable-multilib \
          --prefix=$NATIVE_MINGW_W64_GCC
make
make install
popd

export PATH=$NATIVE_MINGW_W64_GCC/bin:$PATH

mkdir -p $NATIVE_TOOLS_BLD/mingw-w64-header
pushd $NATIVE_TOOLS_BLD/mingw-w64-header
$NATIVE_TOOLS_SRC/mingw-w64-v3.1.0/mingw-w64-headers/configure \
          --host=i686-w64-mingw32 \
          --prefix=$NATIVE_MINGW_W64_GCC/i686-w64-mingw32

make install

pushd $NATIVE_MINGW_W64_GCC
ln -s i686-w64-mingw32 mingw32
popd
popd

mkdir -p $NATIVE_TOOLS_BLD/mingw-gcc
pushd $NATIVE_TOOLS_BLD/mingw-gcc
$NATIVE_TOOLS_SRC/gcc-4.6.4/configure \
          --target=i686-w64-mingw32 \
          --prefix=$NATIVE_MINGW_W64_GCC \
          --disable-multilib --enable-languages=c,c++

make all-gcc
make install-gcc
popd

mkdir -p $NATIVE_TOOLS_BLD/mingw-w64-crt
pushd $NATIVE_TOOLS_BLD/mingw-w64-crt
$NATIVE_TOOLS_SRC/mingw-w64-v3.1.0/mingw-w64-crt/configure \
          --host=i686-w64-mingw32 \
          --prefix=$NATIVE_MINGW_W64_GCC/i686-w64-mingw32

make
make install
popd

pushd $NATIVE_TOOLS_BLD/mingw-gcc
```

```
make
make install
popd

#Get and deploy the InstallJammer
pushd $NATIVE_TOOLS_SRC
wget -c http://downloads.sourceforge.net/project/installjammer/\
InstallJammer/1.2.15/installjammer-1.2.15.tar.gz
pushd $NATIVE_TOOLS_DIR
tar xf $NATIVE_TOOLS_SRC/installjammer-1.2.15.tar.gz
popd
popd

#After those two steps, remember to re-source the .bashrc to make
#those new native tools effective before build ARM embedded toolchain.
#There is no need to source .bashrc every time because the reboot or
#re-login will source .bashrc automatically.
echo "export PATH=$NATIVE_MINGW_W64_GCC/bin:\$PATH" >> ~/.bashrc
echo "export PATH=$NATIVE_TOOLS_DIR/installjammer:\$PATH" >> ~/.bashrc
```

1.2.5 Make new build tools take effective

As shown in above setup script, pathes to our new native build tools will be prepended to environment variable PATH through .bashrc file. A further step is requried to active this new PATH variable. This can be done by different ways such as: re-source the .bashrc file in current terminal right after execute the setup script through command '`. ~/.bashrc`', or open a new terminal then do remaining work in this new terminal, or re-login, or reboot the system.

1.3 Build GNU Tools for ARM Embedded Processors

Before proceed, let's run some commands to make sure the new native build tools are taking in charge. Here are commands and expected results:

```
build@shgcc-vbuild01:~$ which gcc
/home/build/native-tools/native-gcc/bin/gcc

build@shgcc-vbuild01:~$ which installjammer
/home/build/native-tools/installjammer/installjammer

build@shgcc-vbuild01:~$ which i686-w64-mingw32-gcc
/home/build/native-tools/native-mingw-w64-gcc/bin/i686-w64-mingw32-gcc

build@shgcc-vbuild01:~$ gcc --version
gcc (GCC) 4.6.4
Copyright (C) 2011 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.

build@shgcc-vbuild01:~$ i686-w64-mingw32-gcc --version
i686-w64-mingw32-gcc (GCC) 4.6.4
Copyright (C) 2011 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.
```

If you can get expected results, you have set up the building environment. You can now start to build the toolchain by yourself with below commands:

```
#Copy the src release package into ~/toolchain/ directory
$ cp gcc-arm-none-eabi-4_8-2014q1-20140314-src.tar.bz2 ~/toolchain
#Prepare source codes
$ cd ~/toolchain
$ tar -xjf gcc-arm-none-eabi-4_8-2014q1-20140314-src.tar.bz2
$ cd ./gcc-arm-none-eabi-4_8-2014q1-20140314/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]
```

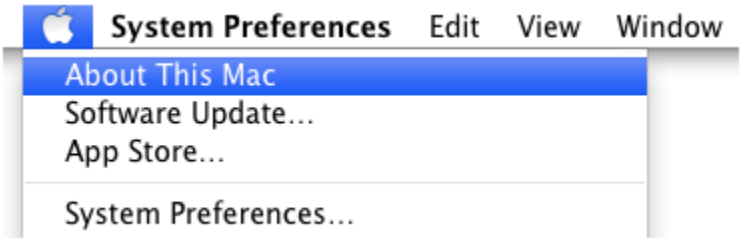
After this, you can '`cd`' into '`~/toolchain/gcc-arm-none-eabi-4_8-2014q1-20140314/pkg`' and find the built toolchain/source code packages and the md5 checksum file.

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 10.7.3 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 10.7.3. 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 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.

2.4 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_8-2014q1-20140314-src.tar.bz2 ~/mac-build

#Prepare source codes
$ cd ~/mac-build
$ tar xjf gcc-arm-none-eabi-4_8-2014q1-20140314-src.tar.bz2
$ cd ./gcc-arm-none-eabi-4_8-2014q1-20140314/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`.
- If you are using different build environment and tools, you might run into problem that binutils can not be successfully built. This is probably caused by binutils bug 13036. 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 1.2.4 [Install GCC 4.6/MinGW-w64/InstallJammer], page 4 to the directory containing '`libstdc++.a`' on your build system.