gcc-4_8_2 (/gcc-4_8_2)

How To Build GCC 4.8.2 ARM Cross-Compiler

Intro

In this How To we are going to build an ARM cross-compiler based upon GCC 4.8.2. Before you begin you might want to take a look at my build machine specs to get an idea of what I'm running on compared to your machine. In the least please make sure you have enough memory. Do not expect a flawless build, especially if you are only running with 1 to 2 GB of memory 8P If you run into issues try to figure them out. Just for some perspective, the ~/workbench/gcc-4.8.2 directory on my machine is approx. 4 GB when all is done.

Tar Balls

Here is a list of source packages that we'll need for the build. You can either download them now or wait 'til later in the How To.

- binutils-2.23.2.tar.bz2
- glibc-2.18.tar.gz
- gcc-4.8.2.tar.bz2
- gmp-4.3.2.tar.bz2
- mpfr-2.4.2.tar.bz2
- mpc-0.8.1.tar.gz
- linux-2.6.38.tar.bz2

Create a Workspace

I recommend creating a workspace under your `/home/<your user>/` directory that is dedicated to this build. So let's fire up your terminal and run the following:

```
$ export SRCDIR=~/workbench/gcc-4.8.2/xtools/src
$ mkdir -pv ~/workbench/gcc-4.8.2/xtools
$ mkdir $SRCDIR
$ cd $SRCDIR
```

Gather the Sources

Now that we have a workspace created and we are currently in the `src` directory we can begin bringing down the sources and extracting them.

binutils

```
$ wget http://ftp.gnu.org/gnu/binutils/binutils-2.23.2.tar.bz2
$ tar -pxjf binutils-2.23.2.tar.bz2
```

glibc

```
$ wget http://ftp.gnu.org/gnu/glibc/glibc-2.18.tar.gz
$ tar -pxzf glibc-2.18.tar.gz
```

gcc

Refer to gcc-4.8.2/contrib/download_prerequisites for rcommended gmp, mpfr, and mpc

```
$ wget http://ftp.gnu.org/gnu/gcc/gcc-4.8.2/gcc-4.8.2.tar.bz2
$ wget http://ftp.gnu.org/gnu/gmp/gmp-4.3.2.tar.bz2
$ wget http://ftp.gnu.org/gnu/mpfr/mpfr-2.4.2.tar.bz2
$ wget ftp://gcc.gnu.org/pub/gcc/infrastructure/mpc-0.8.1.tar.gz
$ tar -pxjf gcc-4.8.2.tar.bz2
$ cd gcc-4.8.2/
$ tar -pxjf ../gmp-4.3.2.tar.bz2
$ tar -pxjf ../mpfr-2.4.2.tar.bz2
$ tar -pxzf ../mpc-0.8.1.tar.gz
$ mv gmp-4.3.2/ gmp
$ mv mpfr-2.4.2/ mpfr
$ mv mpc-0.8.1/ mpc
$ cd $SRCDIR
```

kernel

```
$ wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.38.tar.bz2
$ tar -pxjf linux-2.6.38.tar.bz2
```

Build Environment

To make things a little smoother let's setup some environment variables:

```
$ export BINUTILS_SRC=$SRCDIR/binutils-2.23.2
$ export KERNEL_SRC=$SRCDIR/linux-2.6.38
```

```
$ export GCC_SRC=$SRCDIR/gcc-4.8.2
$ export GLIBC_SRC=$SRCDIR/glibc-2.18
$ export BUILDDIR=~/workbench/gcc-4.8.2/xtools/build
$ export TARGETMACH=arm-none-linux-gnueabi
$ export BUILDMACH=i686-pc-linux-gnu
$ export INSTALLDIR=~/workbench/gcc-4.8.2/arm
$ export SYSROOTDIR=$INSTALLDIR/sysroot
```

Next up is to begin building.

Build binutils

```
$ mkdir $BUILDDIR
$ mkdir $BUILDDIR/binutils
$ cd $BUILDDIR/binutils
$ $BINUTILS_SRC/configure --disable-werror --build=$BUILDMACH --target=$
$ make
$ make install
```

Hopefully that went without any errors ;-) If not, try to figure them out. When you are ready to go and/or fixed any errors move to the next step.

Kernel Headers

I'm building this cross-compiler for the PandaBoard thus for `ARCH` I'm using `arm omap2plus_defconfig`. Substitute your own board here. In any case, make sure you're specify the correct architecture, i.e. `arm`.

```
$ cd $KERNEL_SRC
$ make mrproper
$ make ARCH=arm omap2plus_defconfig
$ mkdir -pv $INSTALLDIR/sysroot/usr
$ make ARCH=arm headers_check
$ make ARCH=arm INSTALL_HDR_PATH=$INSTALLDIR/sysroot/usr headers_install
$ cd $SRCDIR
```

Easy peasy. Next is our first go around with gcc.

Bootstrap gcc

```
$ mkdir $BUILDDIR/bootstrap-gcc
$ cd $BUILDDIR/bootstrap-gcc
```

```
$ $GCC_SRC/configure --build=$BUILDMACH --host=$BUILDMACH --target=$TARG
$ make all-gcc install-gcc
$ make all-target-libgcc install-target-libgcc
$ ln -s $INSTALLDIR/lib/gcc/arm-none-linux-gnueabi/4.8.2/libgcc.a $INSTA
```

Not too bad I hope :o)

glibc Headers - If binutils is not compatible with glibc this is where it will fail

NOTE: Make sure you have gawk installed (else you get an error similar to this).

```
$ mkdir -pv $BUILDDIR/libc
$ cd $BUILDDIR/libc
$ echo "libc_cv_forced_unwind=yes" > config.cache
$ echo "libc_cv_c_cleanup=yes" >> config.cache
$ export PATH=$INSTALLDIR/bin:$PATH
$ export CROSS=arm-none-linux-gnueabi
$ export CC=${CROSS}-gcc
$ export LD=${CROSS}-ld
$ export AS=${CROSS}-as
$ $GLIBC_SRC/configure --build=$BUILDMACH --host=$TARGETMACH --prefix=$S
$ make -k install-headers cross_compiling=yes install_root=$SYSROOTDIR

*** We need to move some files ***
$ pushd $SYSROOTDIR/$INSTALLDIR/sysroot/usr/include
$ cp -rv * $SYSROOTDIR/usr/include/
$ popd

$ ln -s $INSTALLDIR/lib/gcc/arm-none-linux-gnueabi/4.8.2/libgcc.a $INSTA
$ cd $SRCDIR
```

Building glibc

```
$ rm -rf $BUILDDIR/libc
$ mkdir -pv $BUILDDIR/libc
$ cd $BUILDDIR/libc
$ echo "libc_cv_forced_unwind=yes" > config.cache
$ echo "libc_cv_c_cleanup=yes" >> config.cache

*** check to make sure these are still set, they should be ***
$ echo $PATH
$ echo $CROSS
```

```
$ echo $CC

$ $GLIBC_SRC/configure --build=$BUILDMACH --host=$TARGETMACH --prefix=/u
$ make -k install-headers cross_compiling=yes install_root=$SYSROOTDIR
$ ln -s $INSTALLDIR/lib/gcc/arm-none-linux-gnueabi/4.8.2/libgcc.a $INSTA
$ make
$ make install_root=$SYSROOTDIR install
```

Building The Next gcc

NOTE: Here is where we would enable more languages, i.e. c++

```
*** unset CC, LD, and AS. We do not want to xcompile the xcompiler :-) *
$ unset CC
$ unset LD
$ unset AS

*** delete gcc-x.x.x and re-install it ***
$ cd $SRCDIR
$ rm -rf gcc-4.8.2
$ tar -pxjf gcc-4.8.2.tar.bz2
$ cd gcc-4.8.2/
$ tar -pxjf ../gmp-4.3.2.tar.bz2
$ tar -pxjf ../mpfr-2.4.2.tar.bz2
$ tar -pxzf ../mpc-0.8.1.tar.gz
$ mv gmp-4.3.2/ gmp
$ mv mpfr-2.4.2/ mpfr
$ mv mpc-0.8.1/ mpc
$ mkdir -pv $BUILDDIR/final-gcc
$ cd $BUILDDIR/final-gcc
$ echo "libc_cv_forced_unwind=yes" > config.cache
$ echo "libc_cv_c_cleanup=yes" >> config.cache
$ BUILD_CC=gcc
$ $GCC_SRC/configure --build=$BUILDMACH --target=$TARGETMACH --prefix=$I
$ make all-gcc
$ make install-gcc
```

If you made it this far, take a break and give yourself an attaboy :-D

Building The Final gcc

```
*** make sure these are still unset ***
$ echo $CC
```

```
$ echo $LD
$ echo $AS

*** delete gcc-x.x.x and re-install it ***
$ cd $SRCDIR
$ rm -rf gcc-4.8.2
$ tar -pxjf gcc-4.8.2.tar.bz2
$ cd gcc-4.8.2/
$ tar -pxjf ../gmp-4.3.2.tar.bz2
$ tar -pxjf ../mpfr-2.4.2.tar.bz2
$ tar -pxzf ../mpc-0.8.1.tar.gz
$ mv gmp-4.3.2/ gmp
$ mv mpfr-2.4.2/ mpfr
$ mv mpc-0.8.1/ mpc
$ mkdir -pv $BUILDDIR/final-gcc-2
$ cd $BUILDDIR/final-gcc-2
$ echo "libc_cv_forced_unwind=yes" > config.cache
$ echo "libc_cv_c_cleanup=yes" >> config.cache
$ $GCC_SRC/configure --build=$BUILDMACH --target=$TARGETMACH --prefix=$I
$ make
$ make install
```

Your done! :) Now go compile a test program with your new toy and see if it works. In a *new* terminal run:

```
$ export INSTALLDIR=~/workbench/gcc-4.8.2/arm
$ export PATH=$INSTALLDIR/bin:$PATH
$ export TARGETMACH=arm-none-linux-gnueabi
$ export BUILDMACH=i686-pc-linux-gnu
$ export CROSS=arm-none-linux-gnueabi
$ export CC=${CROSS}-gcc
$ export LD=${CROSS}-ld
$ export AS=${CROSS}-as
```

Now compile your test program:

```
$ $CC -Wall -Wextra <your test>.c -o <your test>
```

Check to see if your test program was successfully cross-compiled for ARM:

```
$ file <your test>
bash: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically link
```

If you see something similar to the output above, then you have successfully cross-compiled your test program.

Usage

To use the cross-compiler all you need to do is set the following in a new terminal/tab:

```
$ export INSTALLDIR=~/workbench/gcc-4.8.2/arm
$ export PATH=$INSTALLDIR/bin:$PATH
$ export TARGETMACH=arm-none-linux-gnueabi
$ export BUILDMACH=i686-pc-linux-gnu
$ export CROSS=arm-none-linux-gnueabi
$ export CC=${CROSS}-gcc
$ export LD=${CROSS}-ld
$ export AS=${CROSS}-as
```

It may be cumbersome but I highly recommend doing the above for *every* build. That is to say, each time you cross-compile software for ARM do it in a new terminal/tab and begin by setting the environment variables above. In doing so you ensure that your build starts clean. I have had many occasions where my build would fail at odd times and some of the times I was able to fix it by closing the terminal/tab and starting fresh.

To run the programs that have been compiled with this cross-compiler you'll need to move some of the libraries and binaries to the file system on your dev board or your custom Linux file system. These are located in sysroot: `~/workbench/gcc-4.8.2/arm/sysroot`.