Building GCC as a cross compiler for Raspberry Pi

Posted on May 6, 2018 by Sol

In this article, I will show you how to build GCC 8 as a cross compiler for Raspberry Pi. A cross compiler is a compiler that runs on an operating system and produces executables for another. This is really useful when you want to use your beefy computer to build a library or other large piece of code for Raspberry Pi. As a practical example, at the end of the article, I will show you how to use the cross compiler to build GCC itself as a native Raspberry Pi application.

Part of this article is a compilation of what I've learned reading other people posts. Here is a list of the sources I've used:

- http://preshing.com/20141119/how-to-build-a-gcc-cross-compiler/
- https://www.raspberrypi.org/documentation/linux/kernel/building.md
- https://wiki.osdev.org/Why_do_I_need_a_Cross_Compiler%3F
- https://wiki.osdev.org/GCC_Cross-Compiler
- https://wiki.osdev.org/Building_GCC
- http://www.ifp.illinois.edu/~nakazato/tips/xgcc.html

From the above list, the first article is the one that is the most complete and, if you follow it, you end up with a cross compiler that partially works. To be fair, the article wasn't written for Raspberry Pi. I recommend that you read it if you want to see a more in depth explanation of certain steps of the process.

To build and host the cross compiler, I've used Ubuntu 18.04, but a similar procedure should work on other Linux distributions.

First, make sure your system is updated and install the required prerequisites:

```
1 sudo apt update
2 sudo apt upgrade
3 sudo apt install build-essential gawk git texinfo bison
```

At the time of this writing, Raspbian comes with *GCC 6.3.0*, *Binutils 2.28* and *Glibc 2.24*. It is really important that we build our cross compiler using the same *Glibc* version as the one from Raspbian. This will allow us to integrate nicely with the OS. If you are from the future and read this article, you can check the versions of the above software with these commands:

```
1 gcc --version
2 ld -v
3 ldd -v
```

This is what I see on my Raspberry Pi:

```
 1 pi@raspberrypi:~ $ gcc --version
 2 gcc (Raspbian 6.3.0-18+rpi1+deb9u1) 6.3.0 20170516
 3 Copyright (C) 2016 Free Software Foundation, Inc.
 4 This is free software; see the source for copying conditions.  There is NO
 5 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 6
 7 pi@raspberrypi:~ $ ld -v
 8 GNU ld (GNU Binutils for Raspbian) 2.28
 9
10 pi@raspberrypi:~ $ ldd --version
11 ldd (Debian GLIBC 2.24-11+deb9u3) 2.24
12 Copyright (C) 2016 Free Software Foundation, Inc.
13 This is free software; see the source for copying conditions.  There is NO
14 warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
15 Written by Roland McGrath and Ulrich Drepper.
```

If you are trying to build *Glibc 2.24* with a more modern *GCC*, like 7.x or 8.x, you are going to get a lot of errors. Easiest approach is to build *Glibc* with *GCC 6.3.0* and later use it with the latest and greatest *GCC*, which at this time is 8.1.

In the next instructions I'll assume that you are doing all the steps in a separate folder and that you keep the same Terminal session until everything is done. For example, you can create a working folder in your home:

```
1 cd ~
2 mkdir gcc_all && cd gcc_all
```

Let's download the software that we'll use for building the cross compiler:

```
1 wget https://ftpmirror.gnu.org/binutils/binutils-2.28.tar.bz2
2 wget https://ftpmirror.gnu.org/gcc/gcc-6.3.0/gcc-6.3.0.tar.gz
3 wget https://ftpmirror.gnu.org/glibc/glibc-2.24.tar.bz2
4 wget https://ftpmirror.gnu.org/gcc/gcc-8.1.0/gcc-8.1.0.tar.gz
5 git clone --depth=1 https://github.com/raspberrypi/linux
```

Next, extract the archives and erase them:

```
1 tar xf binutils-2.28.tar.bz2
2 tar xf glibc-2.24.tar.bz2
3 tar xf gcc-6.3.0.tar.gz
```

```
4 tar xf gcc-8.1.0.tar.gz
5 rm *.tar.*
```

GCC also needs some prerequisites which we can download inside the source folder:

```
1 cd ..
2 cd gcc-6.3.0
3 contrib/download_prerequisites
4 rm *.tar.*
5 cd ..
6 cd gcc-8.1.0
7 contrib/download_prerequisites
8 rm *.tar.*
```

Next, create a folder in which we'll put the cross compiler and add it to the path:

```
1 cd ..
2 sudo mkdir -p /opt/cross-pi-gcc
3 sudo chown $USER /opt/cross-pi-gcc
4 export PATH=/opt/cross-pi-gcc/bin:$PATH
```

Copy the kernel headers in the above folder, see Raspbian documentation for more info about the kernel:

```
1 cd ..
2 cd linux
3 KERNEL=kernel7
4 make ARCH=arm INSTALL_HDR_PATH=/opt/cross-pi-gcc/arm-linux-gnueabihf headers_install
```

Next, let's build *Binutils*:

```
1 cd ..
2 mkdir build-binutils && cd build-binutils
3 ../binutils-2.28/configure --prefix=/opt/cross-pi-gcc --target=arm-linux-gnueabihf --with-arch=armv6 --with-fpu=vfp --with-float=hard --disable-multilib
4 make -j 8
5 make install
```

Open in a text editor *ubsan.c* from *gcc-6.3.0/gcc/*, find line 1474:

```
1       || xloc.file == '\0' || xloc.file[0] == '\xff'
```

and change it to:

```
1       || xloc.file[0] == '\0' || xloc.file[0] == '\xff'
```

save and close the file.

GCC and *Glibc* are interdependent, you can't fully build one without the other, so we are going to do a partial build of *GCC*, a partial build of *Glibc* and finally build *GCC* and *Glibc*. You can read more about this in Preshing's article.

```
1 cd ..
2 mkdir build-gcc && cd build-gcc
3 ../gcc-6.3.0/configure --prefix=/opt/cross-pi-gcc --target=arm-linux-gnueabihf --enable-languages=c,c++,fortran --with-arch=armv6 --with-fpu=vfp --with-f
4 make -j8 all-gcc
5 make install-gcc
```

Now, let's partially build *Glibc*:

```
1 cd ..
2 mkdir build-glibc && cd build-glibc
3 ../glibc-2.24/configure --prefix=/opt/cross-pi-gcc/arm-linux-gnueabihf --build=$MACHTYPE --host=arm-linux-gnueabihf --target=arm-linux-gnueabihf --with-a
4 make install-bootstrap-headers=yes install-headers
5 make -j8 csu/subdir_lib
6 install csu/crt1.o csu/crti.o csu/crtn.o /opt/cross-pi-gcc/arm-linux-gnueabihf/lib
7 arm-linux-gnueabihf-gcc -nostdlib -nostartfiles -shared -x c /dev/null -o /opt/cross-pi-gcc/arm-linux-gnueabihf/lib/libc.so
8 touch /opt/cross-pi-gcc/arm-linux-gnueabihf/include/gnu/stubs.h
```

Back to *GCC*:

```
1 cd ..
2 cd build-gcc
3 make -j8 all-target-libgcc
4 make install-target-libgcc
```

Finish building *Glibc*:

```
1 cd ..
2 cd build-glibc
3 make -j8
4 make install
```

Finish building *GCC 6.3.0*:

```
1 cd ..
2 cd build-gcc
3 make -j8
4 make install
5 cd ..
```

Optionally, write a small C or C++ test program, you can build the code with:

```
1 arm-linux-gnueabihf-g++ test.cpp -o test
```

The resulting executable, *test*, from above was build with our first cross compiler and will run on your Pi. You can check that using the *file* command:

```
1 $ file test
2 test: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, not stripped
```

At this point, you have a full cross compiler toolchain with GCC 6.3.0. Make a backup before proceeding with the next step:

```
1 sudo cp -r /opt/cross-pi-gcc /opt/cross-pi-gcc-6.3.0
```

Next, we are going to use the above built *Glibc* to build a more modern cross compiler that will overwrite 6.3:

```
1 cd ..
2 mkdir build-gcc8 && cd build-gcc8
3 ../gcc-8.1.0/configure --prefix=/opt/cross-pi-gcc --target=arm-linux-gnueabihf --enable-languages=c,c++,fortran --with-arch=armv6 --with-fpu=vfp --with-f
4 make -j8
5 make install
```

At this point, you can use *GCC 8.1* to cross compile any C, C++ or Fortran code for your Raspberry Pi. You can invoke any of the cross compilers by using the prefix:

```
1 arm-linux-gnueabihf-
```

examples: *arm-linux-gnueabihf-gcc*, a*rm-linux-gnueabihf-g++*, *arm-linux-gnueabihf-gfortran*.

In order to stress test our cross compiler, let's use it to cross compile itself for the Pi:

```
1 sudo mkdir -p /opt/gcc-8.1.0
2 sudo chown $USER /opt/gcc-8.1.0
3
4 cd ..
5 mkdir build-native-gcc8 && cd build-native-gcc8
6 ../gcc-8.1.0/configure --prefix=/opt/gcc-8.1.0 --build=$MACHTYPE --host=arm-linux-gnueabihf --target=arm-linux-gnueabihf --enable-languages=c,c++,fortran
7 make -j 8
8 make install-strip
```

You should end up with a a native ARM GCC in your */opt/gcc-8.1.0* folder.

As a side note, building *GCC 8.1* with the above cross compiler took about 12 minutes on my Ubuntu machine. Compare this with the 5 hours I needed to build *GCC 8.1* directly on my Pi 3 and you will see the advantage of having a cross compiler on your main machine. Someone told me that compiling *GCC 7* on a Raspberry Pi Zero took about 5 days!

If you want to permanently add the cross compiler to your path, use something like:

```
1 cd ~
2 echo 'export PATH=/opt/cross-pi-gcc/bin:$PATH' >> .bashrc
```

You can now, optionally, safely erase the build folder. Assuming you've followed my advice, from your home folder use the next command:

```
1 cd ~
2 rm -rf gcc_all
```

Let's archive the compiler and save it to our home folder:

```
1 cd /opt
2 tar -cjvf ~/gcc-8.1.0.tar.bz2 gcc-8.1.0
3 cd ~
```

Copy *gcc-8.1.0.tar.bz2* to your RPi. In the next paragraphs I'll assume you are on your RPi and that the above archive is in your home folder:

```
1 cd ~
2 tar xf gcc-8.1.0.tar.bz2
3 rm gcc-8.1.0.tar.bz2
4 sudo mv gcc-8.1.0 /opt
```

Next, we are going to add the new compilers to the path and create a few symbolic links:

```
 1 echo 'export PATH=/opt/gcc-8.1.0/bin:$PATH' >> .bashrc
 2 echo 'export LD_LIBRARY_PATH=/opt/gcc-8.1.0/lib:$LD_LIBRARY_PATH' >> .bashrc
 3 source .bashrc
 4 sudo ln -s /usr/include/arm-linux-gnueabihf/sys /usr/include/sys
 5 sudo ln -s /usr/include/arm-linux-gnueabihf/bits /usr/include/bits
 6 sudo ln -s /usr/include/arm-linux-gnueabihf/gnu /usr/include/gnu
 7 sudo ln -s /usr/include/arm-linux-gnueabihf/asm /usr/include/asm
 8 sudo ln -s /usr/lib/arm-linux-gnueabihf/crti.o /usr/lib/crti.o
 9 sudo ln -s /usr/lib/arm-linux-gnueabihf/crt1.o /usr/lib/crt1.o
10 sudo ln -s /usr/lib/arm-linux-gnueabihf/crtn.o /usr/lib/crtn.o
```

At this point, you should be able to invoke the compilers with *gcc-8.1.0*, *g++-8.1.0* or *gfortran-8.1.0*.

Let's try to compile and run a C++17 code that uses an if block with init-statement (the example is a bit silly, but it will show you how to compile C++17 programs):

```
 1 #include <iostream>
 2
 3 int main() {
 4     // if block with init-statement:
 5     if(int a = 5; a < 8) {
 6         std::cout << "Local variable a is < 8\n";
 7     } else {
 8         std::cout << "Local variable a is >= 8\n";
 9     }
10     return 0;
11 }
```

Save the above code in a file named *if_test.cpp* and compile it with:

```
1 g++-8.1.0 -std=c++17 -Wall -pedantic if_test.cpp -o if_test
```

This is what I see on my RPi:

```
1 pi@raspberrypi:~ $ g++-8.1.0 -std=c++17 -Wall -pedantic if_test.cpp -o if_test
2 pi@raspberrypi:~ $ ./if_test
3 Local variable a is < 8
4 pi@raspberrypi:~ $
```

If you want to learn more about programming on the Raspberry Pi, a very good book is Exploring Raspberry Pi by Derek Molloy:

If you are interested to learn more about modern C++, I recommend A Tour of C++ by Bjarne Stroustroup:

Show Comments

Categories

x86-64 WSL Scheme Posix C++ Clang C99 macOS OSX Python Roguelike C++17 Multithreading CUDA C++11 ChromeOS Raspberry Pi Fractals Charts Chromebook GCC Books Regex NumPy iOS Arduino Videos Node.js Cling OpenCV Swift Linux Perlin Matplotlib Fortran C# C++14 JavaScript Windows C11 C SciPy Objective-C Armadillo OpenMP Productivity OpenGL

My RSS
My Twitter

Recent posts

- Building GCC as a cross compiler for Raspberry Pi
- Install NumPy, SciPy, Matplotlib and OpenCV for Python 3 on Ubuntu 18.04
- Raspberry Pi - Install Clang 6 and compile C++17 programs
- Python OpenCV - show a video in a Tkinter window
- Python OpenCV - show an image in a Tkinter window
- Writing a minimal x86-64 JIT compiler in C++ - Part 2
- Writing a minimal x86-64 JIT compiler in C++ - Part 1
- C++17 constexpr everything (or as much as the compiler can)
- Clang 6 in a Docker container for C++17 development
- Linux and WSL - Install Clang with libc++ and compile C++17 programs