GCC-MPC8XX Version 1 Tools - Release Notes

(Embedded Software Document)

Document Number C001431

2003/01/10

Document Revision History

C-Number	Date	Explanation
C001431	2002/11/18	Initial release of the cross-compiler build process
C001431		Current version

Table of contents

1. Document Background

This document provides the background technical information about a port of the GNU Compiler Collection (GCC) used as a cross-compiler for the Motorola MPC8xx processor family. The MPC8xx processor family
- is based on the PowerPC architecture,
- falls under the definition of "embedded" processors due to a high level of peripheral logic integration,
- relies on software emulation for floating point computations.

The classical approach to software development for embedded systems is to rely on commercial cross-compiler tools. The alternative described in this document is known as GCC from the Free Software Foundation. It is the compiler behind the Linux operating system and it is increasingly used as a cross-compiler for a number of different target environments.

A first purpose of this document is to support the development of embedded applications using the ABCD-Proto-Kernel™ and the PPCMB/850 circuit board, both originating from CONNOTECH (the PPCMB/850 uses the MPC850 or MPC823 processor). Accordingly, this document focuses on the support of a specific embedded application environment. Nonetheless, the contents of this document can be useful if you wish to achieve related goals, e.g. software development on a MPC860 circuit board without a third-party operating system.

In its first release, this document does not contain a *development tools user guide* section. Such a section would complement the GNU free software documentation, which is itself not systematically kept up to date and/or timely converted to the friendly HTML or PDF formats.

2. The Cross-Compiler Building Process

2.1 Reasons to Proceed with the Building Process

The cross-compiler building process needs to be repeatable so that the run-time libraries can be tailored to the detailed target environment characteristics. Commercial cross-compilers are usually released for a specific CPU architecture, and their run-time libraries can be rebuilt without (cross-)dependencies between the library and the compiler. This is not necessarily the case with the GNU GCC which is released in source code for a number of CPU architecture.

Also, there are many involved technical issues related to the run-time environment for the C++ language, even with a full scale operating system is available for application support. This is even worse in the case of embedded system software, where in practice only a vaguely defined subset of the C++ language is supported in most instances. The recent releases of the GNU GCC package include enhancements to the support of the C++ language, up to gcc-3.2.1 which is the basis for the current port. Since at CONNOTECH we whish to use and support C++ programming of embedded systems, we plan to upgrade the GCC-MPC8XX cross-compiler build process and binaries if we find that C++ language features are better supported in forthcoming official gcc releases (otherwise the rule "don't fix it if it is not broken" should apply).

2.2 Blindly Applying the Building Process

Starting with the following distribution, shown here with their respective sha fingerprints:

binutils-2.13.tar.gz	**595BE33A 5D4469D9 1C4FB72C C909F7B9 4EEC59A5**
gcc-3.2.1.tar.gz	**0F8CF222 07306100 E8C0D3A1 2FF3BFA1 648E1561**
newlib-1.10.0.tar.gz	**CA0747E4 9623CCE3 79672439 28D88324 7A389207**

The following two files are specific to this port of the GNU GCC tools, again with their respective sha fingerprints:

gcc-mpc8xx-patches-1.0.tar.gz	**6FC59CE2 4E65AC15 7FE2AFA8 F3A6611D 6C8DADFD**
port-xgcc.sh	**9E43F01C 0BB995C3 9646028F 8163CA61 73F78B60**

The sha fingerprints can be verified with an utility implementing the SHA-1 algorithm (Secure Hash Standard, FIPS PUB 180-1) available at http://www.connotech.com/misc_tools/sha_fertile.htm.

We use a special account, e.g. the "compiler" account, that is part of group "root", for the compiler build process. We had to fix group directory permissions in /usr/local. We expect to require a few "chown" to "root" after installation as a binaries packaging step.

Unpack and untar the files in the three directories: binutils-2.13, gcc-3.2.1, and newlib-1.10.0:

```
gzip -d -c gnu_dist/binutils-2.13.tar.gz |tar -x
gzip -d -c gnu_dist/gcc-3.2.1.tar.gz |tar -x
gzip -d -c gnu_dist/newlib-1.10.0.tar.gz |tar -x
```

Change a few files for the MPC850 adaptation:

```
gzip -d -c gcc-mpc8xx-patches-1.0.tar.gz |tar -x
```

Run the compiler building script.

2.3 Overview of the Compiler Building Script.

This document section complements the explanations found in the INSTALL directory in the gcc-3.2.1 distribution. The script file port-xgcc.sh is provided as a sample that seemed to work in one environment and needs to be run with caution. It does not stop when a step fails. Furthermore, it contains remove (**rm**) commands that can be damaging to a system if the directory structure is not dedicated to the cross-compiler building process.

The GCC tool set is made of three parts:
- the binutils tools,
- the gcc tools and
- a run-time library (e.g. newlib).

More or less, the binutils tools are the low level stuff like an assembler and linker, and the gcc tools are the compilers for the high level languages. As a library, we use the newlib which is itself an alternative to the usual gcc library, because presumably the newlib is better suited to the embedded environment (perhaps another alternate library will be considered for the ABCD Proto-Kernel).

Each component is prepared in three steps:
- a configure step,
- a "make all" step,
- a "make install" step.

Note that the tools used in these steps are deeply rooted in the Unix/Linux world; they are of little use in the native Windows environment. Thus, the build system (the system on which the cross-compiler is being build) itself requires adequate tools (a compiler, a make utility, ...). The configure step meticulously verifies this requirement.

There are three directory trees involved:

- the source distribution directory,
- the build directory,
- the target directory.

Basically, the configure step carries files from the source distribution directory to the build directory, and the "make install" step carries files from the build directory to the target directory. We use three distinct source distribution directories for the three parts, but we use a common build directory.

The gcc tools are dependent on the run-time library (this dependency occurs through the header files tailored for the run-time library), but the run-time library must be compiled by a cross-compiler. The cross-compiler building script is structured to resolve this apparent paradox.

Note: A recent post on the mailing list crossgcc@sources.redhat.com suggests that a proper use of the configure option like "--with-headers=newlib_dir/newlib/libc/include" might make possible a one-pass compiler build process. We didn't investigate this any further, but we mention it here so that the reader does not take our procedure as the final say.

Step-1.1

First, we make the binutils tools:

a configure step	for the binutils tools,
a "make all" step	for the binutils tools, and
a "make install" step	for the binutils tools.

We now have a GNU cross-assember that takes PowerPC assembler source code and a linker that produces ELF format, the foremost file format for PowerPC embedded software image. These tools should be all right, but we will re-build them again later.

Step-1.2

Then, we make the gcc tools with a special configure option "--without-headers" which means the run-time library headers are absent:

a configure step	for the gcc tools,
a "make all" step	for the gcc tools, and
a "make install" step	for the gcc tools.

Step-1.3

Now, we are ready to compile the run-time library with the cross-compiler:

a configure step	for newlib,
a “make all” step	for newlib, and
a “make install” step	for newlib

(this last step is admittedly of little use due to the next action).

Step-2.1

Now, we erase the target directory.

Step-2.2

We immediately re-install the run-time library:

a “make install” step	for newlib.

We got rid of the gcc tools qualified with the “--without-headers” option, but we kept a version of the run-time library with header files configured properly.

Step-2.3

Now we erase the build directory, and start over again.

Step-3.1

First, we make the binutils tools:

a configure step	for the binutils tools,
a “make all” step	for the binutils tools, and
a “make install” step	for the binutils tools.

Step-3.2

Finally, we make the gcc tools, this time without the special configure option “--without-headers,” so that the run-time library include files are taken into account:

a configure step	for the gcc tools,
a “make all” step	for the gcc tools, and
a “make install” step	for the gcc tools.

That’s it. We have a cross-compiler.

The packaging into a set of compressed binaries involves the **`chown`** command applied to the files in the target directory, so that the files are distributed with the root account as the owner. The file created for the binaries-only installation is `xgcc-mpc8xx-v1.tar.gz`.

2.4 Installation

Decompress the xgcc-mpc8xx-v1.tar.gz file distribution. This installs files in the target directory (default /usr/xgcc-mpc8xx-v1). Expand the linux PATH with the bin sub-directory within this target directory indication, e.g. with the following command:

```
    export PATH=/usr/xgcc-mpc8xx-v1/bin:${PATH}
```

The equivalent function can be embedded in the user profile definitions.

The executable files in the /usr/xgcc-mpc8xx-v1/bin directory have a **`--help`** option that is a starting point for usage assistance.

2.5 Hints for Further Configuration

The main configuration options are superficially documented in the file INSTALL/configure.html in the distribution gcc-3.2.1.tar.gz. Some advanced C++ configuration options are documented in the file libstdc++-v3/docs/html/configopts.html also in the distribution gcc-3.2.1.tar.gz.

We are not aware of *installation* options applicable to the cross-compiler build process. You may want to change the directory in which the cross-compiler is installed. Be careful though, we put some potentially dangerous remove (**`rm`**) commands where a **`make clean`** might be more appropriate.

Also, the list of directories searched for include files is rather confusing. Sorting this installation issue appears difficult. A usual recommendation is to use the **`-print-search-dirs`** option to the powerpc-eabi-gcc command, so you can see what's going on.

The host system on which the cross-compiler build was performed is a Red Hat Linux version 7.3. You may attempt the build and install on other host environments.

The use of another C run-time library may be considered. The cross-compiler build process must be adapted. The selection of a run-time library may be influenced by the relevant licensing issues (see http://www.connotech.com/license/licensing_overview.htm). The C++ run-time library and the C++ compiler are too much dependent on each other for using an alternative library.

2.6 Deliberate Limitations of GNU Tools in the GCC-MPC8xx Configuration

The GNU GCC package is customized for the PPCMB/850 development environment in order to reduce the possibility for development tools configuration errors (above all, we whish to assist our customers in preventing and fixing errors as early as possible in their software development cycle). This customization imposes some limitations:

> the MPC850 processor may be replaced by other members of the Motorola PowerPC 8xx series, but other PowerPC architecture implementations such as the Motorola MPC82xx processors are not currently supported;
>
> the programming specifications required by the ABCD Proto-Kernel implementation are to remain in effect (e.g. continued compliance with the EABI specifications).

Note that this customization applies to the compiler phase. The assembler and linker phases are more forgiving to input (configuration settings, source files and object files) that would be incompatible with the run-time specifications.