GNU Tools for ARM Embedded Processors
Version: 4.9

Table of Contents

* Installing executables on Linux *
Unpack the tarball to the install directory, like this:
$ cd $install_dir && tar xjf gcc-arm-none-eabi-*-yyyymmdd-linux.tar.bz2

For 64 bit system, 32 bit libc and libncurses are required to run the tools.
In addition, if you want to use gdb python build (arm-none-eabi-gdb-py), you'd
install 32 bit python2.7. Please refer to
https://answers.launchpad.net/gcc-arm-embedded/+faq/2601

For some Ubuntu releases, the toolchain can also be installed via
Launchpad PPA at https://launchpad.net/~terry.guo/+archive/gcc-arm-embedded.

* Installing executables on Mac OS X *
Unpack the tarball to the install directory, like this:
$ cd $install_dir && tar xjf gcc-arm-none-eabi-*-yyyymmdd-mac.tar.bz2

* Installing executables on Windows *
Run the installer (gcc-arm-none-eabi-*-yyyymmdd-win32.exe) and follow the
instructions.

The toolchain in windows zip package is an alternate solution to windows
installer for those who cannot run the installer. The zip package needs to
be decompressed in a proper place and then invoked, following instructions
in next section.

To use gdb python build (arm-none-eabi-gdb-py), you need to install 32 bit
python2.7 no matter 32 or 64 bit Windows. Please get the package from
https://www.python.org/download/.

* Invoking GCC *
On Linux and Mac OS X, either invoke with the complete path like this:
$ $install_dir/gcc-arm-none-eabi-*/bin/arm-none-eabi-gcc

or set the path like this:
$ export PATH=$PATH:$install_dir/gcc-arm-none-eabi-*/bin
and run it with:
$ arm-none-eabi-gcc

On Windows (although the above approaches also work), it can be more
convenient to either have the installer register environment variables, or run
INSTALL_DIR\bin\gccvar.bat to set environment variables for the current cmd.

For windows zip package, after decompression the toolchain can be invoked
either with complete path like this:
TOOLCHAIN_UNZIP_DIR\bin\arm-none-eabi-gcc
or by running TOOLCHAIN_UNZIP_DIR\bin\gccvar.bat to set environment variables
for the current cmd.

* Architecture options usage *

This toolchain is built and optimized for Cortex-A/R/M bare metal development.
the following table shows how to invoke GCC/G++ with correct command line
options for variants of Cortex-A/R and Cortex-M architectures.

+------------+--------------------------------------+--------------+

ARM Core	Command Line Options	multilib
Cortex-M0+ Cortex-M0 Cortex-M1	-mthumb -mcpu=cortex-m0plus -mthumb -mcpu=cortex-m0 -mthumb -mcpu=cortex-m1 -mthumb -march=armv6-m	armv6-m
Cortex-M3	-mthumb -mcpu=cortex-m3 -mthumb -march=armv7-m	armv7-m
Cortex-M4 (No FP)	-mthumb -mcpu=cortex-m4 -mthumb -march=armv7e-m	armv7e-m
Cortex-M4 (Soft FP)	-mthumb -mcpu=cortex-m4 -mfloat-abi=softfp -mfpu=fpv4-sp-d16 -mthumb -march=armv7e-m -mfloat-abi=softfp -mfpu=fpv4-sp-d16	armv7e-m /softfp
Cortex-M4 (Hard FP)	-mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16 -mthumb -march=armv7e-m -mfloat-abi=hard-mfpu=fpv4-sp-d16	armv7e-m /fpu
Cortex-M7 (No FP)	-mthumb -mcpu=cortex-m7	cortex-m7
Cortex-M7 (Soft FP)	-mthumb -mcpu=cortex-m7 -mfloat-abi=softfp -mfpu=fpv5-sp-d16 -mthumb -mcpu=cortex-m7 -mfloat-abi=softfp -mfpu=fpv5-d16	cortex-m7 /softfp /fpv5-sp-d16 cortex-m7 /softfp /fpv5-d16
Cortex-M7 (Hard FP)	-mthumb -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-sp-d16 -mthumb -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16	cortex-m7 /fpu /fpv5-sp-d16 cortex-m7 /fpu /fpv5-d16
Cortex-R* (No FP)	[-mthumb] -march=armv7-r	armv7-ar /thumb
Cortex-R* (Soft FP)	[-mthumb] -march=armv7-r -mfloat-abi=softfp -mfpu=vfpv3-d16	armv7-ar /thumb /softfp
Cortex-R* (Hard FP)	[-mthumb] -march=armv7-r -mfloat-abi=hard -mfpu=vfpv3-d16	armv7-ar /thumb /fpu
Cortex-A* (No FP)	[-mthumb] -march=armv7-a	armv7-ar /thumb
Cortex-A* (Soft FP)	[-mthumb] -march=armv7-a -mfloat-abi=softfp -mfpu=vfpv3-d16	armv7-ar /thumb /softfp
Cortex-A* (Hard FP)	[-mthumb] -march=armv7-a -mfloat-abi=hard -mfpu=vfpv3-d16	armv7-ar /thumb

```
|            |                                       | /fpu         |
+------------+---------------------------------------+--------------+

* C Libraries usage *

This toolchain is released with two prebuilt C libraries based on newlib:
one is the standard newlib and the other is newlib-nano for code size.
To distinguish them, we rename the size optimized libraries as:

  libc.a --> libc_nano.a
  libg.a --> libg_nano.a

To use newlib-nano, users should provide additional gcc link time option:
 --specs=nano.specs

Nano.specs also handles two additional gcc libraries: libstdc++_s.a and
libsupc++_s.a, which are optimized for code size.

For example:
$ arm-none-eabi-gcc src.c --specs=nano.specs $(OTHER_OPTIONS)

This option can also work together with other specs options like
--specs=rdimon.specs

Please note that --specs=nano.specs is a linker option. Be sure
to include it in linker options if compiling and linking separately.

** additional newlib-nano libraries usage

Newlib-nano is different from newlib in addition to the libraries' name.
Formatted input/output of floating-point number are implemented as weak symbol.
If you want to use %f, you have to pull in the symbol by explicitly specifying
"-u" command option.

  -u _scanf_float
  -u _printf_float

e.g. to output a float, the command line is like:
$ arm-none-eabi-gcc --specs=nano.specs -u _printf_float $(OTHER_LINK_OPTIONS)

For more about the difference and usage, please refer the README.nano in the
source package.

Users can choose to use or not use semihosting by following instructions.
** semihosting
If you need semihosting, linking like:
$ arm-none-eabi-gcc --specs=rdimon.specs $(OTHER_LINK_OPTIONS)

** non-semihosting/retarget
If you are using retarget, linking like:
$ arm-none-eabi-gcc --specs=nosys.specs $(OTHER_LINK_OPTIONS)

* Linker scripts & startup code *

Latest update of linker scripts template and startup code is available on
http://www.arm.com/cmsis

* Samples *
Examples of all above usages are available at:
$install_dir/gcc-arm-none-eabi-*/share/gcc-arm-none-eabi/samples

Read readme.txt under it for further information.

* GDB Server for CMSIS-DAP based hardware debugger *
CMSIS-DAP is the interface firmware for a Debug Unit that connects
the Debug Port to USB.  More detailed information can be found at
http://www.keil.com/support/man/docs/dapdebug/.

A software GDB server is required for GDB to communicate with CMSIS-DAP based
```

hardware debugger. The pyOCD is an implementation of such GDB server that is written in Python and under Apache License.

For those who are using this toolchain and have board with CMSIS-DAP based debugger, the pyOCD is our recommended gdb server. The pyOCD binary is released at https://launchpad.net/gcc-arm-embedded-misc/pyocd-binary. More information can be found at https://github.com/mbedmicro/pyOCD.