GNU Tools for Arm Embedded Processors
Version: 7

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
```

If you want to use gdb python build (arm-none-eabi-gdb-py), you'd install python2.7.

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

* 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 installer can also be run on the command line. When run on the command-line, the following options can be set:
  - /S Run in silent mode
  - /P Adds the installation bin directory to the system PATH
  - /R Adds an InstallFolder registry entry for the install.

For example, to install the tools silently, amend users PATH and add registry entry:
```
> gcc-arm-none-eabi-*-yyyymmdd-win32.exe /S /P /R
```

The toolchain in windows zip package is a backup to windows installer for those who cannot run installer. We need decompress the zip package in a proper place and then invoke it following instructions in next section.

To use gdb python build (arm-none-eabi-gdb-py), you need 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 path like this:
$ export PATH=$PATH:$install_dir/gcc-arm-none-eabi-*/bin
$ 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 we can invoke toolchain either with complete path like this:
TOOLCHAIN_UNZIP_DIR\bin\arm-none-eabi-gcc
or run 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	thumb /v6-m
	-mthumb -march=armv6-m	
Cortex-M3	-mthumb -mcpu=cortex-m3	thumb /v7-m
	-mthumb -march=armv7-m	
Cortex-M4 (No FP)	-mthumb -mcpu=cortex-m4	thumb /v7e-m
	-mthumb -march=armv7e-m	
Cortex-M4 (Soft FP)	-mthumb -mcpu=cortex-m4 -mfloat-abi=softfp -mfpu=fpv4-sp-d16	thumb /v7e-m /fpv4-sp /softfp
	-mthumb -march=armv7e-m -mfloat-abi=softfp -mfpu=fpv4-sp-d16	
Cortex-M4 (Hard FP)	-mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16	thumb /v7e-m /fpv4-sp /hard
	-mthumb -march=armv7e-m -mfloat-abi=hard -mfpu=fpv4-sp-d16	
Cortex-M7 (No FP)	-mthumb -mcpu=cortex-m7	thumb /v7e-m

	-mthumb -march=armv7e-m	
Cortex-M7 (Soft FP)	-mthumb -mcpu=cortex-m7 -mfloat-abi=softfp -mfpu=fpv5-sp-d16	thumb /v7e-m /fpv4-sp /softfp
	-mthumb -march=armv7e-m -mfloat-abi=softfp -mfpu=fpv5-sp-d16	
	-mthumb -mcpu=cortex-m7 -mfloat-abi=softfp -mfpu=fpv5-d16	thumb /v7e-m /fpv5 /softfp
	-mthumb -march=armv7e-m -mfloat-abi=softfp -mfpu=fpv5-d16	
Cortex-M7 (Hard FP)	-mthumb -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-sp-d16	thumb /v7e-m /fpv4-sp /hard
	-mthumb -march=armv7e-m -mfloat-abi=hard -mfpu=fpv5-sp-d16	
	-mthumb -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16	thumb /v7e-m /fpv5 /hard
	-mthumb -march=armv7e-m -mfloat-abi=hard -mfpu=fpv5-d16	
Cortex-M23	-mthumb -mcpu=cortex-m23	thumb /v8-m.base
	-mthumb -march=armv8-m.base	
Cortex-M33 (No FP)	-mthumb -mcpu=cortex-m33	thumb /v8-m.main
	-mthumb -march=armv8-m.main	
Cortex-M33 (Soft FP)	-mthumb -mcpu-cortex-m33 -mfloat-abi=softfp -mfpu=fpv5-sp-d16	thumb /v8-m.main /fpv5-sp /softfp
	-mthumb -march=armv8-m.main -mfloat-abi=softfp -mfpu=fpv5-sp-d16	
	-mthumb -march=armv8-m.main -mfloat-abi=softfp -mfpu=fpv5-d16	thumb /v8-m.main /fpv5 /softfp
Cortex-M33 (Hard FP)	-mthumb -mcpu=cortex-m33 -mfloat-abi=hard -mfpu=fpv5-sp-d16	thumb /v8-m.main /fpv5-sp /hard
	-mthumb -march=armv8-m.main -mfloat-abi=hard -mfpu=fpv5-sp-d16	
	-mthumb -march=armv8-m.main	thumb

	-mfloat-abi=hard -mfpu=fpv5-d16	/v8-m.main /fpv5 /hard
Cortex-R4 Cortex-R5 Cortex-R7 Cortex-R8 (No FP)	[-mthumb] -march=armv7-r	thumb /v7-ar
Cortex-R4 Cortex-R5 Cortex-R7 Cortex-R8 (Soft FP)	[-mthumb] -march=armv7-r -mfloat-abi=softfp -mfpu=vfpv3-d16	thumb /v7-ar /fpv3 /softfp
Cortex-R4 Cortex-R5 Cortex-R7 Cortex-R8 (Hard FP)	[-mthumb] -march=armv7-r -mfloat-abi=hard -mfpu=vfpv3-d16	thumb /v7-ar /fpv3 /hard
Cortex-R52 (No FP)	[-mthumb] -mcpu=cortex-r52 -- [-mthumb] -march=armv8-r+crc	thumb /v7-ar
Cortex-R52 (Soft FP)	[-mthumb] -mcpu=cortex-r52 -mfloat-abi=softfp -mfpu=neon-fp-armv8 -- [-mthumb] -march=armv8-r+crc -mfloat-abi=hard -mfpu=neon-fp-armv8	thumb /v7-ar /fpv3 /softfp
Cortex-R52 (Hard FP)	[-mthumb] -mcpu=cortex-r52 -mfloat-abi=hard -mfpu=neon-fp-armv8 -- [-mthumb] -march=armv8-r+crc -mfloat-abi=hard -mfpu=neon-fp-armv8	thumb /v7-ar /fpv3 /hard
Cortex-A* (No FP)	[-mthumb] -march=armv7-a	thumb /v7-ar
Cortex-A* (Soft FP)	[-mthumb] -march=armv7-a -mfloat-abi=softfp -mfpu=vfpv3-d16	thumb /v7-ar /fpv3 /softfp
Cortex-A* (Hard FP)	[-mthumb] -march=armv7-a -mfloat-abi=hard -mfpu=vfpv3-d16	thumb /v7-ar /fpv3 /hard

* 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 compile and link time option:

```
 --specs=nano.specs
```

At compile time, a 'newlib.h' header file especially configured for newlib-nano will be used if --specs=nano.specs is passed to the compiler.

Nano.specs also handles two additional gcc libraries: libstdc++_nano.a and libsupc++_nano.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 both a compiler and linker option. Be sure to include in both compiler and linker options if compiling and linking are separated.

** 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 https://developer.arm.com/embedded/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. More information can be found at https://github.com/mbedmicro/pyOCD.