GNU Bare-Metal Targeted Tools for Arm 32-bit Embedded Processors

NXP GCC for Arm Release: 23-January-2025

Release description

This release is based on:

- gcc 10.2.0
- binutils 2.38
- newlib 3.3.0
- gdb 9.2.0

This distribution provides support for newlib and newlib-nano.
The components used to build this release are listed in the NXP_SOURCES text file located here:
<installation_directory>/patches_applied/NXP_SOURCES

Version identification

The version information (emitted via `--version` or `-v`) is based on a build **BLD** identifier as an integer value whose value will always increase with each subsequent release.

```
$ arm-none-eabi-gcc --version
arm-none-eabi-gcc.exe (BLD = 1801) 10.2.0 20200723
(build.sh rev=g95479d5 s=F1020 Earmv7nGCC -W64)
```

```
$ arm-none-eabi-gcc -v
gcc version 10.2.0 20200723 (build.sh rev=g95479d5
s=F1020 Earmv7nGCC -W64) (BLD = 1801)
```

About this Release

This is GCC 10.2 for Arm Linux and embedded (eabi) target processors.

The highlights relevant to NXP processors are:

- Augmented multilib support for 32-bit Arm baremetal builds (Cortex-R52 Arm mode libs added).
- Include the backport fix for GCC Bugzilla Bug 95253 [10/11 Regression] Build failure on MSys. Wrong dependency file escaping on Windows.

  (https://gcc.gnu.org/bugzilla/show_bug.cgi?id=95253)
- Two eabi builds are provided per 32-bit and per 64-bit Arm: one with multi-threaded newlib enablement and one with single thread newlib enablement. This support meets the Zephyr OS multi-threaded requirements.

Using the GNU GCC Compiler System

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, the size optimized libraries have been renamed as:

libc.a --> libc_nano.a

libg.a --> libg_nano.a

To use newlib-nano, user 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 add it to the linker options if compiling and linking separately.

Additional newlib-nano libraries usage

Newlib-nano is different from newlib in addition to naming. Formatted input/output of floating-point number are implemented as weak symbol. So, when using %f the symbol must be pulled in by explicitly specifying the "-u" command option.

```
-u _scanf_float
-u _printf_float
```

Semihosting

Semihosting is enabled using the following options:

```
$ arm-none-eabi-gcc --specs=rdimon.specs ${OTHER_LINK_OPTIONS}
```

Semihosting v2

The semihosting v2 specification introduces new extensions and the HLT encodings. Where possible, have semihosting callers continue to use the previously existing trap instructions to ensure compatibility with legacy semihosting implementations. This requires a mixed implementation of libraries and specs.

Semihosting with v2 specs is enabled using the following options:

```
$ arm-none-eabi-gcc --specs=rdimon-v2m.specs ${OTHER_LINK_OPTIONS}
```

Non-semihosting / retarget

```
$ arm-none-eabi-gcc --specs=nosys.specs ${OTHER_LINK_OPTIONS}
```

Users must provide their own I/O subroutine implementations (*e.g. _read, _write, etc*).

Newlib hooks

Stack pointer initialization

The original newlib stack pointer initialization has several drawbacks:

1. There is no SP initialization for Cortex-R profile cores
2. No SP initialization for different CPU modes (e.g. FIQ, ABT etc) in Thumb state
3. Different SP and SL initialization for semihosting
4. Hardwired SP and SL values

Therefore, existing Arm crt0 code were refactored and SP and SL initialization was moved into a separate _*stack_init* hook. A short description of this hook is below:

```
User mode only:        This routine makes default target specific Stack
  +-----+ <- SL_sys,   Pointer initialization for different processor modes:
  |     |    SL_usr    FIQ, Abort, IRQ, Undefined, Supervisor, System (User)
  | SYS |              and setups a default Stack Limit in-case the code has
  | USR | -=0x10000    been compiled with "-mapcs-stack-check" for FIQ and
  |     |              System (User) modes.
  |     |
  +-----+ <- initial SP,
          becomes SP_sys   Hard-wiring SL value is not ideal, since there is
          and SL_usr     currently no support for checking that the heap and
                         stack have not collided, or that this default 64k
All modes:               is enough for the program being executed. However,
  +-----+ <- SL_sys,     it ensures that this simple crt0 world will not
  |     |    SL_usr      immediately cause an overflow event.
  | SYS |
  | USR | -=0x10000        We go through all execution modes and set up SP
  |     |                for each of them.
  +-----+ <- SP_sys,
  |     |    SP_usr     Note:
  | SVC | -= 0x8000        Mode switch via CPSR is not allowed once in
  |     |                  non-privileged mode, so we take care not to enter
  +-----+ <- SP_svc        "User" to set up its sp, and also skip most
  |     |                  operations if already in that mode.
  | IRQ | -= 0x2000
  |     |               Input parameters:
^ +-----+ <- SP_und      - sp - Initialized SP
s |     |                - r2 - May contain SL value from semihosting
t | UND | -= 0x1000             SYS_HEAPINFO call
a |     |               Scratch registers:
c +-----+ <- SP_und      - r1 - new value of CPSR
k |     |                - r2 - intermediate value (in standalone mode)
  | ABT | -= 0x1000      - r3 - new SP value
g |     |                - r4 - save/restore CPSR on entry/exit
r +-----+ <- SP_abt,
o |     |    SL_fiq       Declared as "weak" so that user can write and use
w | FIQ | -= 0x1000     his own implementation if current doesn't fit.
t |     |
h +-----+ <- initial SP
         becomes SP_fiq
```

GCC Section Pragmas

NXP GCC supports `#pragma GCC section (text|data|bss|rodata) ".name"`

This pragma allows to redefine standard sections by user defined section:

```
#pragma GCC section (text|data|bss|rodata) ".name"
```

This pragma restores the standard section:

```
#pragma GCC section (text|data|bss|rodata) "default"
```

Example:

```
#pragma GCC section bss ".foobss"
#pragma GCC section data ".foodata"
#pragma GCC section rodata ".foorodata"
#pragma GCC section text ".footext"

char c1;
char c2 = 2;
const char c3 = 3;
int f1 (int x) { return x; }

#pragma GCC section bss "default"
#pragma GCC section data "default"
#pragma GCC section rodata "default"
#pragma GCC section text "default"
```

Installing executables on Windows

Some executables built by MinGW require additional libraries. So, in order to support out-of-the-box execution, libwinpthread.dll is included in the package. This library can be placed as needed by the host system. The alternative is to install the MinGW package.

Installing executables on Linux

This release contains only 64-bit Linux executables. 32-bit Linux hosts are no longer supported.

GDB with Python

Windows

To use gdb python build (arm-none-eabi-gdb-py.exe) on Windows, 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/.

Release history

Date	Description
7-June-2021 bld=1706 rev=gd25dbc2	GCC 10.2 Release
23-July-2021 bld=1725 rev=g8a8a426	Fix section pragma to work with static variables
10-September-2021 bld=1728 rev=g5963bc8	Dependency on linux host GLIBC_2.27 library is removed
23-January-2025 bld=1801 rev=g95479d5	GCC 10.2 Release Update: Multilib configuration augmented with Arm mode libraries for Cortex-R52 and included fix for GCC Bugzilla Bug 95253