# MIPS Toolchain

# Small C Library Reference Manual

Copyright © Imagination Technologies Limited. All Rights Reserved.

This publication contains proprietary information which is subject to change without notice and is supplied 'as is' without warranty of any kind. Imagination Technologies, the Imagination logo, PowerVR, MIPS, Meta, Ensigma and Codescape are trademarks or registered trademarks of Imagination Technologies Limited. All other logos, products, trademarks and registered trademarks are the property of their respective owners.

| | | |
|---|---|---|
| Filename | : | MIPS_Toolchain.Small C Library Reference Manual.docx |
| Version | : | 1.3.10 External Issue |
| Issue Date | : | 21 Jul 2015 |
| Author | : | Imagination Technologies Limited |

# Contents

# 1. Introduction

This document discusses design principles of size-optimized variants of the C standard library, referred to as SmallCLib which provided as part of the MIPS bare metal toolchain and also serves as a technical reference.

SmallCLib is derived from Newlib v2.0 with small influences from Musl v0.9.14.. The document enumerates all functional differences in the library as compared with Newlib 2.0. It is intended to be used in conjunction with Newlib reference documentation provided with the MIPS Bare Metal toolchain or available on-line at https://sourceware.org/newlib

**Applicable toolchain version**

This document describes a version of SmallCLib compatible with toolchain version 2015.01-5 and higher.

# 2. SmallCLib

The goal of SmallCLib is to provide as much functionality as possible in a small amount of space, and it is intended primarily for embedded use. The standalone library containing size optimized versions of functions is usable from the MIPS bare metal toolchain.

The SmallCLib comes in two variants:

## 2.1. ISO conforming

The ISO C-99 conforming version of SmallCLib, referred to as the small variant, does not omit functionality that's required by the standards in order to achieve a smaller library. Most of the space saving in this version comes from aggressive refactoring of the code to eliminate redundancy and some of the space saving comes at the cost of performance. This version provides an IEEE 754 compliant software floating point Math library.

## 2.2. Non-conforming

This version, referred to as the tiny-variant, omits some ISO features in order to achieve higher code density as compared to the compliant version. Following sections describe differences between ISO conforming and non-conforming versions.

### 2.2.1. File and standard IO

The file and standard IO functions provided by this version operate only on standard streams (stdin, stdout and stderr). All the supported streams are un-buffered.

### 2.2.2. Locales

Locale support in this version is limited to UTF-8 encoding only, which allows direct code paths and better performance. The only locale supported by this version is the default C.UTF-8 locale.

### 2.2.3. Floating point support

The floating point support in this version differs from IEEE 754 standard in the following ways,

- NaN, INF and de-normal input values are not handled. Operations involving such inputs generate unpredictable results.
- Sign of zero is ignored
- No IEEE exceptions are flagged

### 2.2.4. Reentrancy

This version of SmallCLib does not support reentrancy.

## 2.3. Integration with the GCC-based toolchain

Small C library support in GCC is provided via new multi-lib configurations. Pre-built small/tiny libraries variants are provided with the toolchain for select multi-lib combinations. In order to see the multi-lib configurations available with your MIPS toolchain, invoke gcc with the option '–print-multi-lib'.

The Small C library is provided within the toolchain as a set of three static library files, for both small and tiny variants. At the least, the toolchain installation will contain the files mentioned below:

| File | Directory in installed toolchain | Description |
|---|---|---|
| libc.a | small | ISO conforming size optimized string, memory, IO, time and other functions. |
| libm.a | small | ISO conforming size optimized Math functions. |
| libg.a | small | ISO conforming size optimized string, memory, IO, time and other functions, compiled with debugging enabled |
| libc.a | tiny | Non-conforming size optimized string, memory, IO, time and other functions |
| libm.a | tiny | Non-conforming size optimized Math functions. |
| libg.a | tiny | Non-conforming size optimized string, memory, IO, time and other functions, compiled with debugging enabled |

Based on the actual combinations of multilib that the toolchain is built with, there may be multiple versions of these libraries – each having been compiled using a unique combination of compiler options and located within a directory structure determined by the corresponding multi-lib configuration. The libraries listed above correspond to big-endian, hard-float build for mips32r2 architecture.

## 2.4. Linking with and using the small C Library

Selection of standard C library is controlled by a new command-line option to the compiler driver

```
gcc -mclib=[newlib|small|tiny] ...
```

If no -mclib option is specified, the default newlib implementation will be used. At compilation stage, mclib option automatically defines macros necessary for correct linkage. At the link stage, this option controls the library search path so that the desired version of the standard C library gets linked against. If specified, this option must be consistent across both compile and link stages.

*Note: If mclib=[small|tiny] is specified in conjunction with an unsupported combination of other multi-lib options, the compiler will link with the default C library(newlib), unless an alternate library search path is explicitly supplied.*

### 2.4.1. Example application

A variant of the simple hello world application is available in <toolchain-root>/share/examples/helloclib. This example allows the user to build with any one of the 3 C-library variants and prints the sizes of the resulting binary for comparison.

Command to build this example is:

```
# make [MIPS_INST_ROOT=path] [LITEND=1] [CLIB=[newlib|small|tiny]]
```

# 3. Specification Differences with Newlib

This table summarizes the differences between newlib and the size-optimized variants. The following sections provide details about specific function behaviour.

| | Newlib | SmallClib | TinyClib |
|---|---|---|---|
| Complex arithmetic | Yes | No | No |
| Character handling and Localization | Full locale support | Full locale support | Only 7-bit ASCII(default C) locale support |
| Math | IEEE754 compliant,<br>ISO C99 compliant | IEEE754 compliant,<br>ISOC99 compliant | No support for denormals<br>Inifinity, NaN arguments, |
| Extended multi-byte and wide character utilities | Yes | Yes | No |
| Re-entrancy | Yes | Not supported | Not supported |

## 3.1. Character handling and localization

Newlib and the small variant support the default “C”, “POSIX”, “C-JIS", "C-SJIS", "C-EUCJP" and “C-KOI8-R” locales with full wide and multi-byte functionality. The support for locale in tiny variant is limited to default C locale only.

## 3.2. Math functions

Changes in the small-variant of the library are oriented towards better compliance with the ISO C standard in places where Newlib implementation differs from the standard. Key differences relate to the setting of *errno* and correct generation of floating point exceptions. In general, the small-variant tries to do the minimum as required by the ISO C standard – behaviour that is not required or optional under the ISO standard is curtailed to favour a smaller code size.

The tiny-variant departs from IEEE754 and ISO C99 standards by removing support for special values such as de-normalized numbers, infinity and NaN inputs. Special values are correctly supported only by the predicate functions necessary to detect them, so code that uses the math-library can catch these input conditions before invoking library functions. Error-handling based on errno is kept true to the standard, subject to the exclusion of special values, mentioned earlier. Exception generation is entirely disregarded – the library may or may not generate exceptions in any given operation – the only predictable mode of use is with all floating-point exceptions disabled.

For elementary trigonometric functions(sin/cos/tan), the tiny-variant differs from Newlib in terms of arithmetic correctness – it is accurate only for a limited range of inputs. See details below.

### 3.2.1. expm1 – exponential minus 1

```
#include <math.h>
double expm1 (double x);
float expm1 (float x);
```

**Error-handling for large x**

ISO standard requires double-precision expm1 to signal a range-error for large values of x. Newlib does not set the errno, whereas both small and tiny variants of the C-library set errno to ERANGE.

### 3.2.2. fma – fused multiply add

```
#include <math.h>
double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
```

**Arithmetic accuracy**

Double precision variant provided by Newlib performs a non-fused multiply-add. Both small and tiny variants of the C library provide the accuracy of fused operation by performing the intermediate multiplication in parts. Resultant accuracy is close to 80-bit extended precision operation, with very few ULP errors.

The single precision function behaves the same way for all library variants, by approximating fused operation using non-fused operation in double precision.

### 3.2.3. remquo, remquof -- remainder and quotient

**Synopsis**

```
#include <math.h>
double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
```

**Boundary condition: y=0**

The implementation may either set errno to signal a domain error or return 0.0 for this condition, as per Section 7.12.10.3 of ISO-C99. Newlib implementation returns NaN and does not set errno. Both small and tiny variants of the C library set errno to EDOM and return NaN.

### 3.2.4. sin, cos, sinf, cosf -- sine or cosine

**Synopsis**

```
#include <math.h>
double sin (double x);
double cos (double x);
float sinf (float x);
float cosf (float x);
```

**Arithmetic accuracy (only for tiny variant)**

The tiny variant uses an alternate range-reduction strategy which limits accuracy for large arguments. For the double precision function, accuracy varies with absolute input range as follows:

- For $|x| < 2^{53}$ : equal to or better than newlib implementation
- For $2^{53} <= |x| < 2^{64}$ : occasionally worse than newlib, but relatively correct.
- For $|x| > 2^{64}$ : completely unreliable

For single precision function, accuracy varies with absolute input range as follows:

- For $|x| < 2^{23}$ : equal to or better than newlib implementation
- For $2^{23} <= |x| < 2^{32}$ : occasionally worse than newlib, but relatively correct.
- For $|x| > 2^{32}$ : completely unreliable

This variant is significantly smaller and faster than both the small variant and the default newlib implementation.

**Inexact exception for small x ( < $2^{-28}$)**

Newlib implementation generates an inexact exception for small values of x. Both small and tiny variants of the C library do not generate this exception. It is implementation defined as per Appendix F.9, point 8 of ISO-C99.

### 3.2.5. tan, tanf -- tangent

**Synopsis**

```
#include <math.h>
double tan (double x);
float tanf (float x);
```

**Arithmetic accuracy (only for tiny variant)**

Same as sin/cos (see Section 3.2.1)

**Inexact exception for small x ( < $2^{-28}$)**

Same as sin/cos (see Section 3.2.1)

### 3.2.6. nextafter, nextafterf

**Synopsis**

```
#include <math.h>
double nextafter(double x, double y);
float nextafterf(float x, float y);
```

**Arithmetic accuracy**

The nextafter functions return y if x equals y, as per Section 7.12.11.3 of ISO-C99 standard. Newlib implementation returns x. Both small and tiny variants of the C library return y.

## 3.3. Formatted IO functions

Following two macros are available for restricted version of formatted IO. These macros can be used along with small or tiny version of SmallCLib.

### 3.3.1. Integer and character only

__mips_fio_int__ : This macro can be used to disable the support for floating point and wide character format specifiers. It enables the use of integer and character format specifiers (d, i , u, o, n, p, x, X, c, s).

### 3.3.2. Floating point, Integer and character only

__mips_fio_float__ : This macro can be used to enable the support for floating point format specifier (only %f). It enables the use of integer, character and floating point format specifiers (d, i , u, o, n, p, x, X, c, s and f).

The default provides support for all format specifiers.