tinyMicon MatisseCORE™

mtcc ユーザーズガイド

tinyMicon MatisseCORE™用 C 言語コンパイラ mtcc ユーザーズガイド

© 2021 ROHM Co., Ltd.

No. 67UG024J Rev.008
2025.3

更新履歴

日付	バージョン	内容
2021/10/28	Rev.001	mtcc V1.01.00 の内容を記述
2022/03/31	Rev.002	6.8. pragma, 6.9. attribute, 6.10. 事前定義済みマクロの記述を追記
2022/10/21	Rev.003	8. 無保証機能の内容を記述
2023/05/24	Rev.004	3.3. 注意事項の内容を記述
2024/01/15	Rev.005	3.1. 処理系定義の記述を修正 3.2. 翻訳限界を追加
2024/04/04	Rev.006	4.4. C99 規格に厳密に準拠したソースコード診断機能を追加 5.3. ライブラリ関数詳細の記述を修正(stddef.h の内容などを追記) 5.4. 型定義詳細を追加 5.5. マクロ詳細記述を修正(errno.h の内容などを追記)
2024/05/08	Rev.007	5.4.4. stddef.h で ptrdiff_t を修正
2025/03/13	Rev.008	mtcc V2.00.00 と V3.00.00 に対応

© 2021 ROHM Co., Ltd.

目次

1. 概要 7
1.1. 特徴 7
2. 動作環境 8
2.1. 必要なシステム 8
2.2. インストール 8
3. 言語仕様 9
3.1. 処理系定義 9
3.1.1. J.3.1 Translation 9
3.1.2. J.3.2 Environment 10
3.1.3. J.3.3 Identifiers 11
3.1.4. J.3.4 Characters 11
3.1.5. J.3.5 Integers 13
3.1.6. J.3.6 Floating point 13
3.1.7. J.3.7 Arrays and pointers 15
3.1.8. J.3.8 Hints 15
3.1.9. J.3.9 Structures, unions, enumerations, and bit-fields 15
3.1.10. J.3.10 Qualifiers 15
3.1.11. J.3.11 Preprocessing directives 16
3.1.12. J.3.12 Library functions 17
3.1.13. J.3.13 Architecture 23
3.1.14. J.4 Locale-specific behavior 24
3.2. 翻訳限界 26
3.3. 注意事項 27
3.3.1. 無限ループ 27
4. 使用方法 28
4.1. コンパイラ(mtcc)の起動 28
4.2. コンパイラコマンドラインオプション一覧 28
4.3. コンパイラコマンドラインオプション詳細 29
4.3.1. -O 29
4.3.2. -o 29
4.3.3. -I 29
4.3.4. -E 30
4.3.5. -S 30
4.3.6. -c 30
4.3.7. -g 30
4.3.8. -D 31
4.3.9. -Werror 31
4.3.10. -Werror= 31
4.3.11. -Wno- 31
4.3.12. -Weverything 32
4.3.13. -pedantic-errors 32
4.3.14. "-Wl," 32
4.3.15. -v 33

© 2021 ROHM Co., Ltd.

4.3.16. -h 33
4.4. C99 規格に厳密に準拠したソースコード診断機能 33
4.4.1. コマンドラインオプションの説明 33
4.5. リンカ(mtld)の起動 33
4.6. リンカコマンドラインオプション一覧 34
4.7. リンカコマンドラインオプション詳細 35
4.7.1. -o 35
4.7.2. --program-size= 35
4.7.3. --data-size= 35
4.7.4. --ex-program-size= 36
4.7.5. -L 36
4.7.6. -l 36
4.7.7. --defsym 37
4.7.8. --script 37
4.7.9. -Map= 37
4.7.10. --exclude-mul= 38
4.7.11. --crt 38
4.7.12. --libc 39
4.7.13. -v 39
4.7.14. -h 39
5. ライブラリ関数 40
5.1. ライブラリファイル 40
5.2. ヘッダファイル 40
5.3. ライブラリ関数詳細 41
5.3.1. assert.h 41
5.3.2. ctype.h 41
5.3.3. inttypes.h 41
5.3.4. setjmp.h 41
5.3.5. stdarg.h 42
5.3.6. stddef.h 42
5.3.7. stdint.h 42
5.3.8. stdio.h 42
5.3.9. stdlib.h 43
5.3.10. string.h 43
5.3.11. matisse/cpufunc.h 44
5.3.12. matisse/interrupt.h 44
5.3.13. matisse/sections.h 44
5.3.14. util/atomic.h 44
5.4. 型定義詳細 46
5.4.1. inttypes.h 46
5.4.2. setjmp.h 46
5.4.3. stdarg.h 46
5.4.4. stddef.h 46
5.4.5. stdint.h 46
5.4.6. stdio.h 47

© 2021 ROHM Co., Ltd.

5.4.7. stdlib.h 47
5.5. マクロ詳細 48
5.5.1. errno.h 48
5.5.2. iso646.h 49
5.5.3. limits.h 50
5.5.4. setjmp.h 50
5.5.5. stdbool.h 50
5.5.6. stddef.h 50
5.5.7. stdint.h 51
5.5.8. matisse/extmem.h 52
5.5.9. matisse/signature.h 52
5.5.10. matisse/version.h 52
6. 固有の機能 53
6.1. 拡張プログラム領域 53
6.1.1. 使用方法 53
6.1.2. ソースファイル例 53
6.1.3. コンパイルオプション例 54
6.2. 割り込みハンドラ 55
6.2.1. 使用方法 55
6.2.2. ソースファイル例 55
6.2.3. 注意事項 56
6.3. 標準出力 57
6.3.1. 使用方法 57
6.3.2. ソースファイル例 57
6.3.3. デバッガスクリプトファイル例 58
6.3.4. デバッガ実行 58
6.4. SystemInit 関数 59
6.4.1. ソースファイル例 59
6.5. noinit セクション 59
6.5.1. 使用方法 59
6.5.2. ソースファイル例 59
6.5.3. 注意事項 60
6.6. シグネチャデータ 60
6.6.1. 使用方法 60
6.6.2. ソースファイル例 60
6.6.3. 注意事項 60
6.7. インラインアセンブラ 60
6.7.1. 文法 61
6.7.2. ソースファイル例 61
6.7.3. 注意事項 61
6.8. pragma 62
6.8.1. 使用方法 62
6.8.2. ソースファイル例 62
6.8.3. 注意事項 62
6.9. attribute 62

© 2021 ROHM Co., Ltd.

6.9.1. 使用方法 62
6.9.2. ソースファイル例 63
6.9.3. 注意事項 63
6.10. 事前定義済みマクロ 64
6.10.1. 使用方法 64
6.10.2. ソースファイル例 64
7. 呼び出し規約 65
7.1. レジスタの種類 65
7.2. 引数割り当ての呼び出し規約 65
7.2.1. 引数のレジスタへの割り当て規約 65
7.2.2. 引数のスタックへの割り当て規約 65
7.3. 戻り値の呼び出し規約 65
8. 無保証機能 66
8.1. コンパイラコマンドラインオプション 66
8.2. Attribute 66
8.3. リンカコマンドラインオプション 66
オープンソースソフトウェアについて 67
商標について 67

© 2021 ROHM Co., Ltd.

1. 概要

mtcc は ROHM が開発した LSI 組み込み専用の超小型 8-bitCPU、tinyMicon MatisseCORE™向けの C 言語コンパイラです。
C 言語コンパイラは C 言語で記述されたプログラムソースファイルを機械語ファイルに変換します。

1.1. 特徴

mtcc は以下の特徴を持っています。

1 C 言語規格(C99)に準拠。
2 優れた最適化性能。
3 様々なコマンドラインオプション。

© 2021 ROHM Co., Ltd.

2. 動作環境

mtcc の動作環境を以下に説明します。

2.1. 必要なシステム

Table 1. システム要件

OS	Windows Server 2012 (64-bit) Windows 7(32-bit/64-bit) Windows 10(32-bit/64-bit) Windows 11(64-bit)
CPU	Intel Core シリーズ、もしくは同等の性能の CPU。
メモリ	4GByte 以上搭載。
HDD/SSD	200MByte 以上の空き容量。

2.2. インストール

mtcc のインストーラ(MatisseCCompiler-XX.XX.XX.exe)を実行すると、必要なツールが一式インストールされます。
デフォルト設定ではツール一式は"C:¥Program Files¥ROHM¥Matisse¥"以下にインストールされます。

© 2021 ROHM Co., Ltd.

3. 言語仕様

mtcc がサポートする言語仕様は以下の通りです。

言語規格: C99(ISO/IEC 9899:1999)

Table 2. 使用可能な基本データ型のサイズとアライメント

型名	データサイズ	データアライメント
char	1 byte	1 byte
short	2 byte	1 byte
int	2 byte	1 byte
long	4 byte	1 byte
ポインタ型	2 byte	1 byte

Table 3. 使用不可能な基本データ型

型名
float
double
long double
long long

Table 4. 使用不可能な処理

処理
動的メモリ確保
可変長配列

2byte 以上のデータ型はリトルエンディアン形式でメモリに格納されます。
浮動小数点型と 64bit 整数型は組込みプログラムでは殆ど使用されないので mtcc ではサポートしません。
実行環境にはフリースタンディング環境のみをサポートします。

3.1. 処理系定義

C 言語の規格(ISO/IEC 9899:1999 Annex J.3,J.4)で、コンパイラの実装が処理系定義の動作をどのように処理するかについてのドキュメント作成が要求されています。mtcc の処理系定義動作を以下の表に示します。

3.1.1. J.3.1 Translation

Table 5. Translation の要求項目と処理系定義動作

要求項目	処理系定義動作
How a diagnostic is identified (3.10, 5.1.1.3).	[file name]:[line number]:[column number]: [warning/error/fatal error]: [message]
Whether each nonempty sequence of white-space characters other than new-line is retained or replaced by one space character in translation phase 3 (5.1.1.2).	改行文字を除く空白文字は保持されます。

© 2021 ROHM Co., Ltd.

3.1.2. J.3.2 Environment

Table 6. Environment の要求項目と処理系定義動作

要求項目	処理系定義動作
The mapping between physical source file multibyte characters and the source character set in translation phase 1 (5.1.1.2).	識別子、文字列リテラル、文字定数にマルチバイト文字は使用できません。マルチバイト文字からソース文字集合へのマッピングはありません。マルチバイト文字はコメント中でのみ使用可能です。 コメント以外の箇所に使用したマルチバイト文字はコンパイルエラーとなり、例として以下のようなエラーメッセージが表示されます。 `error: non-ASCII characters are not allowed outside of literals and identifiers`

The name and type of the function called at program startup in a freestanding environment (5.1.2.1).	プログラムスタートアップ時に呼び出される関数は int main(void) のみです。
	
	戻り値が int 型以外の main 関数を定義したソースコードはコンパイルエラーとなり、以下のようなエラーメッセージが表示されます。
	`error: 'main' must return 'int'`
	
	引数が 1 個以上ある main 関数を定義したソースコードはコンパイルエラーとなり、以下のようなエラーメッセージが表示されます。
	`error: supported entry point is 'int main(void)' only.`

The effect of program termination in a freestanding environment (5.1.2.1).	main 関数の終了、abort 関数の呼び出し、exit 関数の呼び出しで CPU が HALT 状態になります。
An alternative manner in which the main function may be defined (5.1.2.2.1).	main 関数を定義する他の方法はありません。プログラム内で main 関数を定義しなかった場合はコンパイルエラーとなり、例として以下のようなエラーメッセージが表示されます。 `error: linker error: (.init0+0xe): undefined reference to `main'`
The values given to the strings pointed to by the argv argument to main (5.1.2.2.1).	main 関数の argc 引数と argv 引数はサポートしません。 main 関数は"int main(void)"のみサポートします。 例えば"int main(int argc, char **argv)"を定義したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `error: supported entry point is 'int main(void)' only.`
What constitutes an interactive device (5.1.2.3).	対話型デバイスは特に規定しません。プログラムは外部との対話にハードウェアペリフェラルを使用する事も可能です。
The set of signals, their semantics, and their default handling (7.14).	シグナル機能はサポートしません。 標準ライブラリ内にはシグナル機能用の関数の実装やヘッダファイルが一切存在しません。 例えばシグナル機能を使用しようとして'signal.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'signal.h' file not found`
Signal values other than SIGFPE, SIGILL, and SIGSEGV that correspond to a computational exception (7.14.1.1).	シグナル機能はサポートしません。 標準ライブラリ内にはシグナル機能用の関数の実装やヘッダファイルが一切存在しません。 例えばシグナル機能を使用しようとして'signal.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。

© 2021 ROHM Co., Ltd.

	fatal error: 'signal.h' file not found
Signals for which the equivalent of signal(sig, SIG_IGN); is executed at program startup (7.14.1.1).	シグナル機能はサポートしません。 標準ライブラリ内にはシグナル機能用の関数の実装やヘッダファイルが一切存在しません。 例えばシグナル機能を使用しようとして'signal.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'signal.h' file not found
The set of environment names and the method for altering the environment list used by the getenv function (7.20.4.5).	getenv 関数はサポートしません。 標準ライブラリファイル内に getenv 関数の実装が存在せず、ヘッダファイル'stdlib.h'には getenv 関数の宣言が存在しません。 例えば getenv 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'getenv' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
The manner of execution of the string by the system function (7.20.4.6).	system 関数はサポートしません。 標準ライブラリファイル内に system 関数の実装が存在せず、ヘッダファイル'stdlib.h'には system 関数の宣言が存在しません。 例えば system 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'system' is invalid in C99 [-Werror,-Wimplicit-function-declaration]

3.1.3. J.3.3 Identifiers

Table 7. Identifiers の要求項目と処理系定義動作

要求項目	処理系定義動作
Which additional multibyte characters may appear in identifiers and their correspondence to universal character names (6.4.2).	マルチバイト文字は識別子に使用できません。 マルチバイト文字を識別子に使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: non-ASCII characters are not allowed outside of literals and identifiers
The number of significant initial characters in an identifier (5.2.4.1, 6.4.2).	先頭の 63 文字に意味があります。

3.1.4. J.3.4 Characters

Table 8. Characters の要求項目と処理系定義動作

要求項目	処理系定義動作
The number of bits in a byte (3.6).	1byte は 8bit です。
The values of the members of the execution character set (5.2.1).	実行文字セットの値は US ASCII コードです。
The unique value of the member of the execution character set produced for each of the standard alphabetic escape sequences (5.2.2).	'¥a': 0x07, '¥b': 0x08, '¥f': 0x0C, '¥n': 0x0A, '¥r': 0x0D, '¥t': 0x09, '¥v': 0x0B

© 2021 ROHM Co., Ltd.

The value of a char object into which has been stored any character other than a member of the basic execution character set (6.2.5).	char 型に変換された値です。
Which of signed char or unsigned char has the same range, representation, and behavior as "plain" char (6.2.5, 6.3.1.1).	signed char 型が char 型と同じです。
The mapping of members of the source character set (in character constants and string literals) to members of the execution character set (6.4.4.4, 5.1.1.2).	一対一の対応になります。
The value of an integer character constant containing more than one character or containing a character or escape sequence that does not map to a single-byte execution character (6.4.4.4).	エスケープシーケンス以外は ASCII コードの値になります。 エスケープシーケンスの場合は以下の値が使用されます。 '¥a': 0x07, '¥b': 0x08, '¥f': 0x0C, '¥n': 0x0A, '¥r': 0x0D, '¥t': 0x09, '¥v': 0x0B
The value of a wide character constant containing more than one multibyte character, or containing a multibyte character or escape sequence not represented in the extended execution character set (6.4.4.4).	ワイド文字はサポートしません。 標準ライブラリ内にはワイド文字用の関数の実装やヘッダファイルが一切存在しません。 例えばワイド文字を使用しようとして'wchar.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'wchar.h' file not found`
The current locale used to convert a wide character constant consisting of a single multibyte character that maps to a member of the extended execution character set into a corresponding wide character code (6.4.4.4).	ロケールはサポートしません。 標準ライブラリ内にはロケール用の関数の実装やヘッダファイルが一切存在しません。 例えばロケールを使用しようとして'locale.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'locale.h' file not found`
The current locale used to convert a wide string literal into corresponding wide character codes (6.4.5).	ロケールはサポートしません。 標準ライブラリ内にはロケール用の関数の実装やヘッダファイルが一切存在しません。 例えばロケールを使用しようとして'locale.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'locale.h' file not found`
The value of a string literal containing a multibyte character or escape sequence not represented in the execution character set (6.4.5).	マルチバイト文字はサポートしません。 標準ライブラリ内にはマルチバイト文字用の関数の実装やヘッダファイルが一切存在しません。 例えばマルチバイト文字列の文字数を取得しようとして mblen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `error: implicit declaration of function 'mblen' is invalid in C99 [-Werror,-Wimplicit-function-declaration]` エスケープシーケンスはただのバイナリデータとして文字列リテラルに格納されます。

© 2021 ROHM Co., Ltd.

3.1.5. J.3.5 Integers

Table 9. Integers の要求項目と処理系定義動作

要求項目	処理系定義動作
Any extended integer types that exist in the implementation (6.2.5).	拡張整数型はありません。
Whether signed integer types are represented using sign and magnitude, two's complement, or one's complement, and whether the extraordinary value is a trap representation or an ordinary value (6.2.6.2).	符号付き整数型は 2 の補数で符号と大きさを表現します。トラップ表現はありません。
The rank of any extended integer type relative to another extended integer type with the same precision (6.3.1.1).	拡張整数型はありません。
The result of, or the signal raised by, converting an integer to a signed integer type when the value cannot be represented in an object of that type (6.3.1.3).	変換先の型に格納できないビットは捨てられ、それ以外の下位ビットは変更されません。どのような変数型を型変換した場合でもシグナルは発生しません。
The results of some bitwise operations on signed integers (6.5).	シフト演算の場合は算術シフト演算を行います。それ以外のビット演算は符号なし整数型と同じ演算を行います。

3.1.6. J.3.6 Floating point

Table 10. Floating point の要求項目と処理系定義動作

要求項目	処理系定義動作
The accuracy of the floating-point operations and of the library functions in <math.h> and <complex.h> that return floating-point results (5.2.4.2.2).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'math.h'と'complex.h'は存在しません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `error: floating point type is not supported.`
The rounding behaviors characterized by non-standard values of FLT_ROUNDS (5.2.4.2.2).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'float.h'は存在せず、マクロ FLT_ROUNDS は定義されていません。 例えば FLT_ROUNDS を使用しようとして'float.h'を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'float.h' file not found`
The evaluation methods characterized by non-standard negative values of FLT_EVAL_METHOD (5.2.4.2.2).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'float.h'は存在せず、マクロ FLT_EVAL_METHOD は定義されていません。 例えば FLT_EVAL_METHOD を使用しようとして'float.h'を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'float.h' file not found`
The direction of rounding when an integer is converted to a floating-point number that cannot exactly represent the original value (6.3.1.4).	浮動小数点はサポートしません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。

© 2021 ROHM Co., Ltd.

	error: floating point type is not supported.
The direction of rounding when a floating-point number is converted to a narrower floating-point number (6.3.1.5).	浮動小数点はサポートしません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: floating point type is not supported.
How the nearest representable value or the larger or smaller representable value immediately adjacent to the nearest representable value is chosen for certain floating constants (6.4.4.2).	浮動小数点はサポートしません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: floating point type is not supported.
Whether and how floating expressions are contracted when not disallowed by the FP_CONTRACT pragma (6.5).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'math.h'は存在せずプラグマ FP_CONTRACT は定義されていません。 例えばプラグマ FP_CONTRACT を使用しようとして'math.h'を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
The default state for the FENV_ACCESS pragma (7.6.1).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'fenv.h'は存在せず、プラグマ FENV_ACCESS は定義されていません。 例えばプラグマ FENV_ACCESS を使用しようとして'fenv.h'を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'fenv.h' file not found
Additional floating-point exceptions, rounding modes, environments, and classifications, and their macro names (7.6, 7.12).	浮動小数点はサポートしません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: floating point type is not supported.
The default state for the FP_CONTRACT pragma (7.12.2).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'math.h'は存在せずプラグマ FP_CONTRACT は定義されていません。 例えばプラグマ FP_CONTRACT を使用しようとして'math.h'を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
Whether the "inexact" floating-point exception can be raised when the rounded result actually does equal the mathematical result in an IEC 60559 conformant implementation (F.9).	浮動小数点はサポートしません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: floating point type is not supported.
Whether the "underflow" (and "inexact") floating-point exception can be raised when a result is tiny but not inexact in an IEC 60559 conformant implementation (F.9).	浮動小数点はサポートしません。 例えば浮動小数点型(float, double, long double)の変数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: floating point type is not supported.

© 2021 ROHM Co., Ltd.

3.1.7. J.3.7 Arrays and pointers

Table 11. Arrays and pointers の要求項目と処理系定義動作

要求項目	処理系定義動作
The result of converting a pointer to an integer or vice versa (6.3.2.3).	ポインタから整数型に変換した場合は unsigned int 型の値を変換先の型にキャストした場合と同じ結果になります。整数型からポインタに変換した場合は、変換元の型を unsigned int 型にキャストした場合と同じ結果になります。
The size of the result of subtracting two pointers to elements of the same array (6.5.6).	演算結果は ptrdiff_t 型になります。ptrdiff_t 型は<stddef.h>で定義されていてサイズは 2byte です。

3.1.8. J.3.8 Hints

Table 12. Hints の要求項目と処理系定義動作

要求項目	処理系定義動作
The extent to which suggestions made by using the register storage-class specifier are effective (6.7.1).	register 指定子は無視されます。
The extent to which suggestions made by using the inline function specifier are effective (6.7.4).	inline キーワードは最適化レベルが 1 以上で、コンパイラがインライン展開による効果があると判断した時だけ有効になります。

3.1.9. J.3.9 Structures, unions, enumerations, and bit-fields

Table 13. Structures, unions, enumerations, and bit-fields の要求項目と処理系定義動作

要求項目	処理系定義動作
Whether a "plain" int bit-field is treated as a signed int bit-field or as an unsigned int bit-field (6.7.2, 6.7.2.1).	signed int 型のビットフィールドとして扱われます。
Allowable bit-field types other than _Bool, signed int, and unsigned int (6.7.2.1).	全ての整数型が使用可能です。
Whether a bit-field can straddle a storage-unit boundary (6.7.2.1).	ビットフィールドは記憶単位の境界を跨ぎません。
The order of allocation of bit-fields within a unit (6.7.2.1).	ビットフィールドは下位 bit から割り当てられます。
The alignment of non-bit-field members of structures. This should present no problem unless binary data written by one implementation is read by another. (6.7.2.1).	全てのデータ型のアライメントは 1 バイトです。3. 言語仕様を参照してください。
The integer type compatible with each enumerated type (6.7.2.2).	enum の値に応じて以下の中から最適な型が選択されます。 ● signed int ● unsigned int ● signed long ● unsigned long

3.1.10. J.3.10 Qualifiers

Table 14. Qualifiers の要求項目と処理系定義動作

要求項目	処理系定義動作
What constitutes an access to an object that has volatile-qualified type (6.7.3).	volatile 修飾されたオブジェクトは、非同期的な割り込み機構によってアクセスされる可能性があるとコンパイラに判断され、最適化の対

© 2021 ROHM Co., Ltd.

	象外となり常にメモリにマッピングされます(レジスタには保持されません)。 該当オブジェクトへの読み込み処理は毎回マッピングされたメモリからデータを読み出され、書き込み処理は毎回マッピングされたメモリに反映されます。 該当オブジェクトへの読み書き処理は 1byte ずつ行われます。該当オブジェクトのサイズが 2byte 以上の場合、読み書きアクセスされる byte の順番は不定です。 該当オブジェクトがビットフィールドを含む場合、ビットフィールドへの書き込みは Read-modify-write 方式でアクセスします。

3.1.11. J.3.11 Preprocessing directives

Table 15. Preprocessing directives の要求項目と処理系定義動作

要求項目	処理系定義動作
How sequences in both forms of header names are mapped to headers or external source file names (6.4.7).	ヘッダー名の中のプリプロセッサシーケンスはどちらの形式でも US ASCII コードとして解釈され、ファイル名にマッピングされます。
Whether the value of a character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set (6.10.1).	条件付きインクルードを制御する定数式の文字定数は実行文字セットの文字定数と同じ値を持ちます。
Whether the value of a single-character character constant in a constant expression that controls conditional inclusion may have a negative value (6.10.1).	条件付きインクルードを制御する定数式の単一文字定数は負の値を取る場合があります。
The places that are searched for an included < > delimited header, and how the places are specified or the header is identified (6.10.2).	コンパイラは指定されたファイルを以下の場所から順番で探します。 1. -I オプションで指定されたインクルードパス 2. システムヘッダファイルディレクトリ(デフォルトでは "C:¥Program Files¥ROHM¥Matisse¥C¥include")
How the named source file is searched for in an included " " delimited header (6.10.2).	コンパイラは指定されたファイルを以下の場所から順番で探します。 1. カレントディレクトリ、つまりコンパイル対象のソースファイルがあるディレクトリ 2. -I オプションで指定されたインクルードパス 3. システムヘッダファイルディレクトリ(デフォルトでは "C:¥Program Files¥ROHM¥Matisse¥C¥include")
The method by which preprocessing tokens (possibly resulting from macro expansion) in a #include directive are combined into a header name (6.10.2).	#include ディレクティブ内のトークンは#if ディレクティブ内のプリプロセッサトークンと同じ方法で結合されます。
The nesting limit for #include processing (6.10.2).	15 までサポートしています。
Whether the # operator inserts a ¥ character before the ¥ character that begins a universal character name in a character constant or string literal (6.10.3.2).	#演算子は文字定数または文字列リテラル内の汎用文字名で始まる¥文字の前に¥文字を挿入しません。
The behavior on each recognized non-STDC #pragma directive (6.10.6).	6.8. pragma を参照してください。
The definitions for __DATE__ and __TIME__ when respectively, the date and time of translation are not available (6.10.8).	それらのマクロは常に有効です。

© 2021 ROHM Co., Ltd.

3.1.12. J.3.12 Library functions

Table 16. Library functions の要求項目と処理系定義動作

要求項目	処理系定義動作
Any library facilities available to a freestanding program, other than the minimal set required by clause 4 (5.1.2.1).	5. ライブラリ関数で説明されているすべてのライブラリ関数はフリースタンディング環境で使用可能です。
The format of the diagnostic printed by the assert macro (7.2.1.1).	Assertion failed: ([assertion argument]), function [function name], file [file name], line [line number].
The representation of the floating-point status flags stored by the fegetexceptflag function (7.6.2.2).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'fenv.h'も fegetexceptflag 関数の実装も存在しません。 例えば fegetexceptflag 関数を使用しようとして'fenv.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'fenv.h' file not found`
Whether the feraiseexcept function raises the "inexact" floating-point exception in addition to the "overflow" or "underflow" floating-point exception (7.6.2.3).	浮動小数点はサポートしません。 標準ライブラリ内にヘッダファイル'fenv.h'も feraiseexcept 関数の実装も存在しません。 例えば feraiseexcept 関数を使用しようとして"fenv.h"をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'fenv.h' file not found`
Strings other than "C" and "" that may be passed as the second argument to the setlocale function (7.11.1.1).	ロケールはサポートしません。 標準ライブラリ内にはロケール用の関数の実装やヘッダファイルが一切存在しません。 例えばsetlocale 関数を使用しようとして'locale.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'locale.h' file not found`

The types defined for float_t and double_t when the value of the FLT_EVAL_METHOD macro is less than 0 or greater than 2 (7.12).	数学関数と浮動小数点はサポートしません。
	標準ライブラリ内にヘッダファイル'math.h'と'float.h'は存在せず、float_t 型や double_t 型やマクロ FLT_EVAL_METHOD も定義されていません。
	例えばfloat_tやdouble_tを使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。
	`fatal error: 'math.h' file not found`
	
	また、例えばマクロ FLT_EVAL_MATHOD を使用しようとして'float.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。
	`fatal error: 'float.h' file not found`

Domain errors for the mathematics functions, other than those required by this International Standard (7.12.1).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えば数学関数を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。

© 2021 ROHM Co., Ltd.

	fatal error: 'math.h' file not found
The values returned by the mathematics functions on domain errors (7.12.1).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えば数学関数を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
The values returned by the mathematics functions on underflow range errors, whether errno is set to the value of the macro ERANGE when the integer expression math_errhandling & MATH_ERRNO is nonzero, and whether the "underflow" floating-point exception is raised when the integer expression math_errhandling & MATH_ERREXCEPT is nonzero. (7.12.1).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えばマクロ math_errhandling やマクロ MATH_ERRNO やマクロ MATH_ERREXCEPT を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
Whether a domain error occurs or zero is returned when an fmod function has a second argument of zero (7.12.10.1).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えば fmod 関数を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
The base-2 logarithm of the modulus used by the remquo functions in reducing the quotient (7.12.10.3).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えば remquo 関数を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
Whether the equivalent of signal(sig, SIG_DFL); is executed prior to the call of a signal handler, and, if not, the blocking of signals that is performed (7.14.1.1).	シグナル機能はサポートしません。 標準ライブラリ内にはシグナル機能用の関数の実装やヘッダファイルが一切存在しません。 例えば signal 関数を使用しようとして'signal.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'signal.h' file not found
The null pointer constant to which the macro NULL expands (7.17).	NULL マクロは'((void *)0)'に展開されます。
Whether the last line of a text stream requires a terminating new-line character (7.19.2).	テキストストリームの最終行に改行文字は必要ありません。
Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.19.2).	入力ストリームはサポートしません。 標準ライブラリファイル内に入力ストリーム用関数の実装が存在せず、ヘッダファイル'stdio.h'には入力ストリーム用の変数や関数の宣言が存在しません。 例えば getc 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'getc' is invalid

© 2021 ROHM Co., Ltd.

	in C99 [-Werror,-Wimplicit-function-declaration]
The number of null characters that may be appended to data written to a binary stream (7.19.2).	バイナリストリームに追加される NULL 文字は 0 個です。
Whether the file position indicator of an append-mode stream is initially positioned at the beginning or end of the file (7.19.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
Whether a write on a text stream causes the associated file to be truncated beyond that point (7.19.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
The characteristics of file buffering (7.19.3).	ファイルストリームのバッファリングは行われません。
Whether a zero-length file actually exists (7.19.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
The rules for composing valid file names (7.19.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
Whether the same file can be simultaneously open multiple times (7.19.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。

© 2021 ROHM Co., Ltd.

	例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
The nature and choice of encodings used for multibyte characters in files (7.19.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
The effect of the remove function on an open file (7.19.4.1).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば remove 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'remove' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
The effect if a file with the new name exists prior to a call to the rename function (7.19.4.2).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば rename 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'rename' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
Whether an open temporary file is removed upon abnormal program termination (7.19.4.3).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば tmpfile 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'tmpfile' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
Which changes of mode are permitted (if any), and under what circumstances (7.19.5.4).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fopen 関数を使用したソースコードはコンパイルエラーとなり、

© 2021 ROHM Co., Ltd.

	以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fopen' with type 'FILE *(const char *, const char *)' (aka 'struct __file *(const char *, const char *)') [-Werror,-Wimplicit-function-declaration]
The style used to print an infinity or NaN, and the meaning of any n-char or n-wchar sequence printed for a NaN (7.19.6.1, 7.24.2.1).	無限や NaN はサポートしません。 標準ライブラリ内には無限や NaN 処理用関数の実装やヘッダファイルが一切存在しません。 例えばマクロ INFINITY やマクロ NAN を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'math.h' file not found
The output for %p conversion in the fprintf or fwprintf function (7.19.6.1, 7.24.2.1).	4 桁の 16 進数が出力されます。
The interpretation of a - character that is neither the first nor the last character, nor the second where a ^ character is the first, in the scanlist for %[conversion in the fscanf or fwscanf function (7.19.6.2, 7.24.2.1).	入力ストリームはサポートしません。 標準ライブラリファイル内に入力ストリーム用関数の実装が存在せず、ヘッダファイル'stdio.h'には入力ストリーム用の変数や関数の宣言が存在しません。 例えばfscanf関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fscanf' with type 'int (FILE *restrict, const char *restrict, ...)' (aka 'int (struct __file *restrict, const char *restrict, ...)') [-Werror,-Wimplicit-function-declaration]
The set of sequences matched by a %p conversion and the interpretation of the corresponding input item in the fscanf or fwscanf function (7.19.6.2, 7.24.2.2).	入力ストリームはサポートしません。 標準ライブラリファイル内に入力ストリーム用関数の実装が存在せず、ヘッダファイル'stdio.h'には入力ストリーム用の変数や関数の宣言が存在しません。 例えばfscanf関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'fscanf' with type 'int (FILE *restrict, const char *restrict, ...)' (aka 'int (struct __file *restrict, const char *restrict, ...)') [-Werror,-Wimplicit-function-declaration]
The value to which the macro errno is set by the fgetpos, fsetpos, or ftell functions on failure (7.19.9.1, 7.19.9.3, 7.19.9.4).	ファイル操作はサポートしません。 標準ライブラリファイル内にファイル操作関数の実装が存在せず、ヘッダファイル'stdio.h'にはファイル操作の変数や関数の宣言が存在しません。 例えば fgetpos 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'fgetpos' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
The meaning of any n-char or n-wchar sequence in a string representing a NaN that is converted by the strtod, strtof, strtold, wcstod, wcstof, or wcstold function	それらの関数はサポートしません。 標準ライブラリファイル内にそれらの関数の実装が存在せず、ヘッダファイル'stdlib.h'にはそれらの関数の宣言が存在しません。

© 2021 ROHM Co., Ltd.

(7.20.1.3, 7.24.4.1.1).	例えばstrtod 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'strtod' with type 'double (const char *, char **)' [-Werror,-Wimplicit-function-declaration]
Whether or not the strtod, strtof, strtold, wcstod, wcstof, or wcstold function sets errno to ERANGE when underflow occurs (7.20.1.3, 7.24.4.1.1).	それらの関数はサポートしません。 標準ライブラリファイル内にそれらの関数の実装が存在せず、ヘッダファイル'stdlib.h'にはそれらの関数の宣言が存在しません。 例えばstrtod 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'strtod' with type 'double (const char *, char **)' [-Werror,-Wimplicit-function-declaration]
Whether the calloc, malloc, and realloc functions return a null pointer or a pointer to an allocated object when the size requested is zero (7.20.3).	それらの関数はサポートしません。 標準ライブラリファイル内にそれらの関数の実装が存在せず、ヘッダファイル'stdlib.h'にはそれらの関数の宣言が存在しません。 例えば malloc 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'malloc' with type 'void *(unsigned int)' [-Werror,-Wimplicit-function-declaration]
Whether open streams with unwritten buffered data are flushed, open streams are closed, or temporary files are removed when the abort or _Exit function is called (7.20.4.1, 7.20.4.4).	ストリームはバッファされません、一時ファイルはサポートしません。 標準ライブラリファイル内に一時ファイル処理用関数の実装が存在せず、ヘッダファイル'stdio.h'には一時ファイル処理用の変数や関数の宣言が存在しません。 例えば tmpfile 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'tmpfile' is invalid in C99 [-Werror,-Wimplicit-function-declaration] オープンしたストリームは abort 関数や_Exit 関数の呼び出しでクローズされません。
The termination status returned to the host environment by the abort, exit, or _Exit function (7.20.4.1, 7.20.4.3, 7.20.4.4).	abort 関数を呼び出した場合は終了状態は 1 になります。exit 関数の場合は引数が終了状態になります。 終了状態はレジスタ R14 と R15 に格納されます。7.3. 戻り値の呼び出し規約を参照してください。
The value returned by the system function when its argument is not a null pointer (7.20.4.6).	system 関数はサポートしません。 標準ライブラリファイル内に system 関数の実装が存在せず、ヘッダファイル'stdlib.h'には system 関数の宣言が存在しません。 例えば system 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'system' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
The local time zone and Daylight Saving Time (7.23.1).	時間機能はサポートしません。 標準ライブラリ内に時間機能用の関数の実装やヘッダファイルが一切存在しません。

© 2021 ROHM Co., Ltd.

	例えば時間機能を使用しようとして'time.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'time.h' file not found`
The range and precision of times representable in clock_t and time_t (7.23).	時間機能はサポートしません。 標準ライブラリ内に時間機能用の関数の実装やヘッダファイルが一切存在しません。 例えば clock_t や time_t を使用しようとして'time.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'time.h' file not found`
The era for the clock function (7.23.2.1).	時間機能はサポートしません。 標準ライブラリ内に時間機能用の関数の実装やヘッダファイルが一切存在しません。 例えば clock 関数を使用しようとして'time.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'time.h' file not found`
The replacement string for the %Z specifier to the strftime, and wcsftime functions in the "C" locale (7.23.3.5, 7.24.5.1).	時間機能はサポートしません。 標準ライブラリ内に時間機能用の関数の実装やヘッダファイルが一切存在しません。 例えば strftime 関数を使用しようとして'time.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'time.h' file not found`
Whether or when the trigonometric, hyperbolic, base-e exponential, base-e logarithmic, error, and log gamma functions raise the "inexact" floating-point exception in an IEC 60559 conformant implementation (F.9).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えば数学関数を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'math.h' file not found`
Whether the functions in <math.h> honor the rounding direction mode in an IEC 60559 conformant implementation (F.9).	数学関数はサポートしません。 標準ライブラリ内には数学関数の実装やヘッダファイルが一切存在しません。 例えば数学関数を使用しようとして'math.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 `fatal error: 'math.h' file not found`

3.1.13. J.3.13 Architecture

Table 17. Architecture の要求項目と処理系定義動作

要求項目	処理系定義動作
The values or expressions assigned to the macros specified in the headers <float.h>, <limits.h>, and <stdint.h> (5.2.4.2, 7.18.2, 7.18.3).	<stdint.h>と<limits.h>に関しては 5.5. マクロ詳細を参照してください。 <float.h>はサポートしません。

© 2021 ROHM Co., Ltd.

	'float.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'float.h' file not found
The number, order, and encoding of bytes in any object (when not explicitly specified in this International Standard) (6.2.6.1).	どのオブジェクトもアライメントは 1byte です。エンディアンはリトルエンディアンです。3. 言語仕様を参照してください。
The value of the result of the sizeof operator (6.5.3.4).	3. 言語仕様を参照してください。

3.1.14. J.4 Locale-specific behavior

Table 18. Locale-specific behavior の要求項目と処理系定義動作

要求項目	処理系定義動作
Additional members of the source and execution character sets beyond the basic character set (5.2.1).	ソース文字セットと実行文字セットに追加のメンバはありません。
The presence, meaning, and representation of additional multibyte characters in the execution character set beyond the basic character set (5.2.1.2).	実行文字セットに追加のマルチバイト文字はありません。
The shift states used for the encoding of multibyte characters (5.2.1.2).	マルチバイト文字はサポートしません。 標準ライブラリ内にはマルチバイト文字用の関数の実装やヘッダファイルが一切存在しません。 例えばマルチバイト文字列の文字数を取得しようとして mblen 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'mblen' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
The direction of writing of successive printing characters (5.2.2).	常に左から右です。
The decimal-point character (7.1.1).	小数点は常に"."です。
The set of printing characters (7.4, 7.25.2).	US ASCII コードの 0x20 から 0x7E までです。
The set of control characters (7.4, 7.25.2).	US ASCII コードの 0x00 から 0x1F までと 0x7F です。
The sets of characters tested for by the isalpha, isblank, islower, ispunct, isspace, isupper, iswalpha, iswblank, iswlower, iswpunct, iswspace, or iswupper functions (7.4.1.2, 7.4.1.3, 7.4.1.7, 7.4.1.9, 7.4.1.10, 7.4.1.11, 7.25.2.1.2, 7.25.2.1.3, 7.25.2.1.7, 7.25.2.1.9, 7.25.2.1.10, 7.25.2.1.11).	iswalpha 関数, iswblank 関数, iswlower 関数, iswpunct 関数, iswspace 関数と iswupper 関数はサポートしません。 他の関数については以下の US ASCII コードの文字がテストされます。 isalpha: 0x41 から 0x5A までと 0x61 から 0x7A まで。 isblank: 0x09 と 0x20。 islower: 0x61 から 0x7A まで。 ispunct: 0x21 から 0x2F までと 0x3A から 0x40 までと 0x5B から 0x60 までと 0x7B から 0x7E まで。 isspace: 0x09 から 0x0D までと 0x20。 isupper: 0x41 から 0x5A まで。
The native environment (7.11.1.1).	ロケールはサポートしません。 標準ライブラリ内にはロケール用の関数の実装やヘッダファイルが一切存在しません。 例えばロケールを使用しようとして'locale.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されま

© 2021 ROHM Co., Ltd.

	す。 fatal error: 'locale.h' file not found
Additional subject sequences accepted by the numeric conversion functions (7.20.1, 7.24.4.1).	数値変換関数が受け付ける追加の変換対象シーケンスはありません。
The collation sequence of the execution character set (7.21.4.3, 7.24.4.4.2).	照合関数(strcoll, strxfrm)はサポートしません。 標準ライブラリ内には照合関数の実装が存在せず、ヘッダファイル'string.h'には照合関数の宣言が存在しません。 例えばstrcoll 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicit declaration of function 'strcoll' is invalid in C99 [-Werror,-Wimplicit-function-declaration]
The contents of the error message strings set up by the strerror function (7.21.6.2).	strerror 関数はサポートしません。 標準ライブラリファイル内に strerror 関数の実装が存在せず、ヘッダファイル'string.h'には strerror 関数の宣言が存在しません。 例えば strerror 関数を使用したソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 error: implicitly declaring library function 'strerror' with type 'char *(int)' [-Werror,-Wimplicit-function-declaration]
The formats for time and date (7.23.3.5, 7.24.5.1).	時間機能はサポートしません。 標準ライブラリ内に時間機能用の関数の実装やヘッダファイルが一切存在しません。 例えば strftime 関数を使用しようとして'time.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'time.h' file not found
Character mappings that are supported by the towctrans function (7.25.1).	towctrans 関数はサポートしません。 標準ライブラリ内には towctrans 関数の実装やヘッダファイルが一切存在しません。 例えば towctrans 関数を使用しようとして'wctype.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'wctype.h' file not found
Character classifications that are supported by the iswctype function (7.25.1).	iswctype 関数はサポートしません。 標準ライブラリ内には iswctype 関数の実装やヘッダファイルが一切存在しません。 例えばiswctype 関数を使用しようとして'wctype.h'をインクルードしたソースコードはコンパイルエラーとなり、以下のエラーメッセージが表示されます。 fatal error: 'wctype.h' file not found

© 2021 ROHM Co., Ltd.

3.2. 翻訳限界

C99(ISO/IEC 9899:1999)で定義されている翻訳限界の各項目について、mtcc が動作を保証する値を以下の表に記述します。

Table 19. 翻訳限界における mtcc の動作保証値

項目	C99 規格値	mtcc が動作保証する値
ブロックの入れ子のレベル数	127	127
条件付き取込みにおける入れ子のレベル数	63	63
一つの宣言中の一つの算術型，構造体型，共用体型又は不完全型を修飾するポインタ，配列及び関数宣言子（の任意の組合せ）の個数	12	12
一つの完結宣言子における括弧で囲まれた宣言子の入れ子のレベル数	63	63
一つの完結式における括弧で囲まれた式の入れ子のレベル数	63	63
内部識別子又はマクロ名において意味がある先頭の文字数	63	63
外部識別子において意味がある先頭の文字数	31	31
一つの翻訳単位中における外部識別子数	4095	4095
一つのブロックで宣言されるブロック有効範囲をもつ識別子数	511	511
一つの前処理翻訳単位中で同時に定義されるマクロ識別子数	4095	4095
一つの関数定義における仮引数の個数	127	127
一つの関数呼出しにおける実引数の個数	127	127
一つのマクロ定義における仮引数の個数	127	127
一つのマクロ呼出しにおける実引数の個数	127	127
一つの論理ソース行における文字数	4095	4095
（連結後の）単純文字列リテラル又はワイド文字列リテラル中における文字数	4095	4095(*1)
（ホスト環境の場合）一つのオブジェクトのバイト数	65535	－(*2)
#include で取り込まれるファイルの入れ子のレベル数	15	15
一つの switch 文（入れ子になった switch 文を除く）中における case ラベルの個数	1023	1023
一つの構造体又は共用体のメンバ数	1023	1023
一つの列挙体における列挙定数の個数	1023	1023
一つのメンバ宣言並びにおける構造体又は共用体定義の入れ子のレベル数	63	63

(*1): この項目の最大値の使用はソースコード上では可能ですが、ターゲットとなる Matisse 機種のハードウェア構成によっては文字列オブジェクトが RAM に収まらない場合があります。つまりコンパイル時に構文解析エラーにはなりませんが、リンク時に RAM メモリサイズエラーが発生する場合があります。RAM サイズを変更する方法に関しては 4.7.3. --data-size=を参照してください。
(*2): mtcc はフリースタンディング環境のみサポートしていてホスト環境はサポートしていないので、この項目は適用範囲外です。

© 2021 ROHM Co., Ltd.

3.3. 注意事項

上記の通り mtcc は C99(ISO/IEC 9899:1999)に準拠していますが、この規格の中には注意が必要な内容もあります。
そのため mtcc は規格通りの動作をしているにもかかわらず、一見不具合のような挙動を見せる場合があります。
このセクションではそれらの注意事項の内容と回避策について説明します。

3.3.1. 無限ループ

3.3.1.1. 説明

mtcc は副作用がない無限ループを未定義処理と解釈し、削除する場合があります。それによって、無限ループを含むプログラムが無限ループせずに終了したり、意図しない動作を引き起こしたりする場合があります。

副作用のない無限ループの例は以下です。このような記述は未定義動作となるので避けてください。

```
while(1) {
  // infinite loop with no side-effects
}
```

3.3.1.2. 回避策

以下のように volatile キーワードを使用すると無限ループを安全に記述できます。

```
volatile bool loopFlag = true;
while(loopFlag) {
}
```

volatile キーワードはこの無限ループが副作用を持つ可能性があると mtcc に伝えます。その結果、mtcc はこの無限ループを未定義処理と解釈せず、削除しなくなります。

© 2021 ROHM Co., Ltd.

4. 使用方法

4.1. コンパイラ(mtcc)の起動

mtcc は以下のようにコマンドラインから実行します。入力ファイルは基本的に複数指定可能で、C 言語ソースファイル(*.c)、アセンブラファイル(*.S)、オブジェクトファイル(*.o)が指定できます。

```
> mtcc [options] input_file -o output_file
```

mtcc を実行するコマンドラインの具体例は以下のようになります。

```
> mtcc -O2 -I./include source.c -o output.bin
```

4.2. コンパイラコマンドラインオプション一覧

コンパイラ(mtcc)の主なコマンドラインオプションの一覧は以下になります。

Table 20. コンパイラ(mtcc)の主なコマンドライン一覧

コマンドラインオプション	説明
-O	最適化レベルを指定します。
-o	出力ファイル名を指定します。
-I	インクルードファイルのディレクトリを指定します。複数指定可。
-E	プリプロセッサ実行結果を標準出力に出力します。
-S	C 言語ソースファイルからアセンブラファイルを出力します。
-c	C 言語ソースファイルからオブジェクトファイルを出力します。
-g	出力ファイルにデバッグ情報を追加します。 デバッグ情報のフォーマットは DWARF4 です。
-D	プリプロセッサマクロを定義します。複数指定可。
-Werror	ワーニングを全てエラーとして扱います。
-Werror=	指定したワーニングをエラーとして扱います。
-Wno-	指定したワーニングの発生を抑制します。
-Weverything	全てのワーニングを有効にします。
-pedantic-errors	C99 規格に準拠してエラーを発生させます。
-Wl,	コンマ区切りで指定されたオプションをリンカ(mtld)に渡します。
-v	バージョン情報を表示します。
-help	ヘルプメッセージを表示します。

© 2021 ROHM Co., Ltd.

4.3. コンパイラコマンドラインオプション詳細

4.3.1. -O

4.3.1.1. 説明

-O オプションは以下のフォーマットでコンパイラの最適化レベルを指定します。このオプションを省略した場合はデフォルト値の(-O2)が適用されます。

```
-Ox [x=0,1,2,s]
```

最適化レベルによってプログラムの実行速度や ROM サイズ、デバッグのしやすさが異なります。
各最適化レベルの違いは以下になります。

Table 21. 各最適化レベルの違い

オプション	説明	プログラム実行速度	ROM サイズ	デバッグしやすさ
-O0	最適化なし	遅い	大	しやすい
-O1	最適化レベル 1	普通	中	普通
-O2	最適化レベル最大	最速	小	しづらい
-Os	省 ROM サイズ	速い	最小	しづらい

4.3.1.2. 使用例

```
> mtcc -O0 input.c -o output.bin
```

4.3.2. -o

4.3.2.1. 説明

-o オプションは以下のフォーマットで出力ファイル名を指定します。バイナリ形式のファイル(ROM イメージ)が出力されます。同時に ELF 形式のファイルも自動的に出力されます(ELF 形式のファイルのファイル名は、出力ファイル名の拡張子を.elf に変えたものになります)。
ただし-E オプションが使用された場合は-o オプションは無効化され、-S オプションが使用された場合はアセンブラファイルのみが出力されます。

```
-o filename
```

4.3.2.2. 使用例

```
> mtcc input.c -o output.bin
```

4.3.3. -I

4.3.3.1. 説明

-I オプションは以下のフォーマットでディレクトリを指定し、指定したディレクトリをインクルードファイル探索パスに追加します。-I オプションは複数指定可能です。

```
-Idirname
```

© 2021 ROHM Co., Ltd.

4.3.3.2. 使用例

```
> mtcc -I./include input.c -o output.bin
> mtcc -I./include1 -I./include2 input.c -o output.bin
```

4.3.4. -E

4.3.4.1. 説明

-E オプションはプリプロセッサのみ実行した結果を標準出力に出力します。-E オプション使用時は-o オプションは無効化されます。-E オプションの実行結果をファイルに保存する場合は-o オプションではなくリダイレクトを使用してください。

4.3.4.2. 使用例

```
> mtcc -E input.c > output.pp
```

4.3.5. -S

4.3.5.1. 説明

-S オプションはアセンブラファイルを出力します。-S オプション使用時には入力ファイル数は 1 つに制限されます。出力するアセンブラファイルの拡張子は慣例により(*.S)が推奨されます。

4.3.5.2. 使用例

```
> mtcc -S input.c -o output.S
```

4.3.6. -c

4.3.6.1. 説明

-c オプションはオブジェクトファイルを出力します。-c オプション使用時には入力ファイル数は 1 つに制限されます。出力するオブジェクトファイルの拡張子は慣例により(*.o)が推奨されます。

4.3.6.2. 使用例

```
> mtcc -c input.c -o output.o
```

4.3.7. -g

4.3.7.1. 説明

-g オプションは出力ファイル内にデバッグ情報を生成します。デバッグ情報のフォーマットは DWARF バージョン 4 です。

© 2021 ROHM Co., Ltd.

4.3.7.2. 使用例

```
> mtcc -g input.c -o output.bin
```

4.3.8. -D

4.3.8.1. 説明

-D オプションは以下のフォーマットでプリプロセッサマクロを定義します。-D オプションは複数指定が可能です。

```
-Dname
-Dname=def
```

4.3.8.2. 使用例

```
> mtcc -DDEBUG -DCOUNT=100 input.c -o output.bin
```

4.3.9. -Werror

4.3.9.1. 説明

-Werror オプションはコンパイル時に発生したワーニングを全てエラーとして扱います。

4.3.9.2. 使用例

```
>  mtcc -Werror input.c -o output.bin
```

4.3.10. -Werror=

4.3.10.1. 説明

―Werror=オプションは指定したワーニングをエラーとして扱います。例えば、-Werror=constant-conversion は桁あふれが発生する代入へのワーニングをエラーにします。

```
-Werror=constant-conversion
```

4.3.10.2. 使用例

```
> mtcc -Werror=constant-conversion input.c -o output.bin
```

4.3.11. -Wno-

4.3.11.1. 説明

-Wno-オプションは以下のように指定したワーニングを抑制します。例えば、-Wno- constant-conversion は桁あふれが発生する代入へのワーニングが発生しなくなります。

© 2021 ROHM Co., Ltd.

4.3.11.2. 使用例

```
> mtcc -Wno-constant-conversion input.c -o output.bin
```

4.3.12. -Weverything

4.3.12.1. 説明

-Weverything オプションは全てのワーニングの発生を有効にします。デフォルトでは mtcc は clang コンパイラの-Wall オプションと同等の動作になり、細かいワーニングの発生は抑制されていますが、このオプションを使う事で全てのワーニングが有効になります。

4.3.12.2. 使用例

```
> mtcc -Weverything input.c -o output.bin
```

4.3.13. -pedantic-errors

4.3.13.1. 説明

-pedantic-errors オプションを使用すると、mtcc は C99 規格に準拠したソースコード診断を行いエラーを発生させます。ソースコードが C99 規格通りに書かれているかチェックしたい場合に有用です。ただし、-pedantic-errors オプションを使用した場合でも mtcc のソースコード診断は厳密に C99 に準拠するわけではないので注意が必要です。
厳密に C99 規格に準拠したコード診断を行いたい場合は 4.4. C99 規格に厳密に準拠したソースコード診断機能に記載されているコマンドラインオプションを使用してください。

4.3.13.2. 使用例

```
> mtcc -pedantic-errors input.c -o output.bin
```

4.3.14. "-Wl,"

4.3.14.1. 説明

"-Wl,"オプションは以下のフォーマットでリンカ(mtld)に渡すコマンドラインオプションを指定します。このオプションは全体を 2 重引用符で囲む必要があります。

```
"-Wl,opt1,opt2,opt3,...,optX"
```

4.3.14.2. 使用例

```
> mtcc "-Wl,-L./lib,-Map=mapfile.map" input.c -o output.bin
```

© 2021 ROHM Co., Ltd.

4.3.15. -v

4.3.15.1. 説明

-v オプションはコンパイラ(mtcc)のバージョン情報を標準出力に出力します。

4.3.15.2. 使用例

```
> mtcc -v
```

4.3.16. -h

4.3.16.1. 説明

-h オプションはコマンドラインオプションのヘルプメッセージを標準出力に出力します。

4.3.16.2. 使用例

```
> mtcc -h
```

4.4. C99 規格に厳密に準拠したソースコード診断機能

mtcc のソースコード診断機能を C99 規格に厳密に準拠させたい場合はコンパイル時に以下のコマンドラインオプションを全て追加します。

```
-Wall -pedantic-errors -Werror=undefined-internal -Werror=implicit-function-declaration -Werror=invalid-noreturn -Werror=static-local-in-inline -Wno-keyword-macro -Wno-comment -Wno-trigraphs -Wno-shift-op-parentheses -Wno-initializer-overrides -Wno-tautological-constant-out-of-range-compare -Wno-constexpr-not-const -Wno-format-security -Wno-ignored-qualifiers -Wno-unknown-attributes -Wno-braced-scalar-init -Wno-null-conversion -Wno-ambiguous-ellipsis -Wno-inaccessible-base -Wno-static-self-init -Wno-deprecated-increment-bool -Wno-missing-exception-spec -Wno-deprecated-register -Wno-vexing-parse -Wno-c++11-compat-deprecated-writable-strings -Wno-unknown-pragmas -Wno-absolute-value -Wno-enum-conversion -Wno-deprecated-declarations -Wno-extern-initializer -Wno-constant-logical-operand -Wno-switch -Wno-literal-conversion -Wno-unused-volatile-lvalue -Wno-enum-compare -Wno-parentheses -Wno-array-bounds -Wno-sizeof-array-decay -Wno-pointer-bool-conversion -Wno-visibility -Wno-bitfield-constant-conversion -Wno-unused-value -Wno-missing-braces -Wno-missing-braces -Wno-int-to-pointer-cast -Wno-empty-body -Wno-tautological-compare
```

4.4.1. コマンドラインオプションの説明

-pedantic-errors オプションを使用する事で mtcc のソースコード診断機能が C99 準拠した動作に切り替わります。ただし、-pedantic-errors オプションを使用しただけでは診断機能は規格に厳密に準拠した動作にはなりません。規格では許容されている書き方をエラーにしてしまうケースがあるので-Wno-オプションを使用してそういったケースのエラーを抑制しています。また、規格違反の書き方をエラーではなくワーニングにしてしまうケースもあるので-Werror=オプションを使用してそのようなワーニングをエラーに変更しています。

4.5. リンカ(mtld)の起動

mtld は以下のようにコマンドラインから実行します。入力ファイルは複数指定可能で、オブジェクトファイル(*.o)が指定できます。

```
> mtld [options] input_file -o output_file
```

© 2021 ROHM Co., Ltd.

mtld を実行するコマンドラインの具体例は以下のようになります。

```
> mtld --program-size=10 -Map=output.map input.o -o output.bin
```

また、以下のようにコンパイラ(mtcc)のコマンドラインオプション"-Wl,"を使用する事でリンカ(mtld)のコマンドラインオプションが指定できます。

```
> mtcc -Wl,--program-size=10,-Map=output.map input.c -o output.bin
```

4.6. リンカコマンドラインオプション一覧

リンカ(mtld)の主なコマンドラインオプションの一覧は以下になります。

Table 22. リンカ(mtld)の主なコマンドラインオプション一覧

コマンドラインオプション	説明
--program-size=	プログラム領域のサイズを指定します。
--data-size=	データ領域のサイズを指定します。
--ex-program-size=	拡張プログラム領域のサイズを指定します。
-L	ライブラリファイルのディレクトリを指定します。複数指定可。
-l	リンクするライブラリファイルを指定します。複数指定可。
--defsym	リンカ用のグローバルシンボルを定義します。複数指定可。
--script	リンカスクリプトを変更する場合に使用します。
-Map=	マップファイル出力先を指定します。
--exclude-mul=	ターゲットにするコアの乗算器の有無を指定します。
--crt	リンクするランタイムライブラリを変更します。
--libc	リンクするライブラリファイルを変更します。
-v	バージョン情報を表示します。
-h	ヘルプメッセージを表示します。

© 2021 ROHM Co., Ltd.

4.7. リンカコマンドラインオプション詳細

4.7.1. -o

4.7.1.1. 説明

-o オプションは以下のフォーマットで出力ファイル名を指定します。バイナリ形式のファイル(ROM イメージ)が出力されます。同時に ELF 形式のファイルも自動的に出力されます(ELF 形式のファイルのファイル名は、出力ファイル名の拡張子を.elf に変えたものになります)。

```
-o filename
```

4.7.1.2. 使用例

```
> mtld input.o -o output.bin
```

4.7.2. --program-size=

4.7.2.1. 説明

--program-size=オプションは以下のフォーマットでプログラム領域のサイズを指定します。プログラム領域は指定した数字に 4kbyte を掛けたサイズになります。このオプションを省略した場合はデフォルト値の 8(32kbyte)が適用されます。

```
--program-size=n [n=8,9,10,11,12,13,14,15]
```

--program-size=で指定した値と--data-size=で指定した値の合計が 17(68kbyte)以上になるとメモリ容量エラーになります。

4.7.2.2. 使用例

```
> mtcc "-Wl,--program-size=8" input.c -o output.bin
> mtld --program-size=8 input.o -o output.bin
```

4.7.3. --data-size=

4.7.3.1. 説明

--data-size=オプションは以下のフォーマットでデータ領域のサイズを指定します。データ領域は指定した数字に 4kbyte を掛けたサイズになります。このオプションを省略した場合はデフォルト値の 1(4kbyte)が適用されます。

```
--data-size=n [n=1,2,3,4,5,6,7,7]
```

--program-size=で指定した値と--data-size=で指定した値の合計が 17(68kbyte)以上になるとメモリ容量エラーになります。

4.7.3.2. 使用例

```
> mtcc "-Wl,--data-size=1" input.c -o output.bin
> mtld --data-size=1 input.o -o output.bin
```

© 2021 ROHM Co., Ltd.

4.7.4. --ex-program-size=

4.7.4.1. 説明

--ex-program-size=オプションは以下のフォーマットで拡張プログラム領域のサイズを指定します。拡張プログラム領域は指定した数字に2kbyte を掛けたサイズになります。このオプションを省略した場合はデフォルト値の 0(0kbyte)が適用されます。

```
--ex-program-size=n [n=0..32]
```

4.7.4.2. 使用例

```
> mtcc "-Wl,--ex-program-size=10" input.c -o output.bin
> mtld --ex-program-size=10 input.o -o output.bin
```

4.7.5. -L

4.7.5.1. 説明

-L オプションは以下のフォーマットでディレクトリを指定し、指定したディレクトリをライブラリファイル探索パスに追加します。このオプションは複数指定可能です。

```
-Ldirname
```

4.7.5.2. 使用例

```
> mtcc "-Wl,-L./lib" input.c -o output.bin
> mtld -L./lib input.o -o output.bin
```

4.7.6. -l

4.7.6.1. 説明

-l オプションは以下のフォーマットでリンクするライブラリファイルを指定します。指定したライブラリファイルは-L オプションでライブラリ探索パスに登録されたディレクトリ内から探索されます。このオプションは複数指定可能です。

```
-llibname
```

このオプションで指定するライブラリファイルは以下の条件を全て満たす必要があります。

- スタティックライブラリ
- ファイル名が"lib"で始まる
- 拡張子が".a"

-l オプションではライブラリファイル名から"lib"と".a"を取り除いた文字列を指定します

© 2021 ROHM Co., Ltd.

4.7.6.2. 使用例

```
> mtcc "-Wl,-ltest" input.c -o output.bin  // libtest.a というライブラリファイルがリンクされる
> mtld -ltest input.o -o output.bin        // libtest.a というライブラリファイルがリンクされる
```

4.7.7. --defsym

4.7.7.1. 説明

--defsym オプションは以下のフォーマットでシンボルを定義します。このオプションは複数指定可能です。

```
--defsym SYMBOL=EXPRESSION
```

4.7.7.2. 使用例

```
> mtcc "-Wl,--defsym __my_symbol=0xABCD" input.c -o output.bin
> mtld --defsym __my_symbol=0xABCD input.o -o output.bin
```

4.7.8. --script

4.7.8.1. 説明

--script オプションは以下のフォーマットでリンク処理に使用するリンカスクリプトを指定します。このオプションを省略した場合はデフォルトのリンカスクリプトが使用されます。デフォルトのリンカスクリプトは"C:¥Program Files¥ROHM¥Matisse¥C¥lib¥ldscripts¥matisse.x"です。

```
--script filename
```

4.7.8.2. 使用例

```
> mtcc "-Wl,--script my_linker_script.x" input.c -o output.bin
> mtld --script my_linker_script.x input.o -o output.bin
```

4.7.9. -Map=

4.7.9.1. 説明

-Map=オプションは以下のフォーマットでマップファイルの出力先を指定します。マップファイルにはグローバル変数や関数のアドレスが記載されます。

```
-Map=filename
```

4.7.9.2. 使用例

```
> mtcc "-Wl,-Map=mapfile.map" input.c -o output.bin
> mtld -Map=mapfile.map input.o -o output.bin
```

© 2021 ROHM Co., Ltd.

4.7.10. --exclude-mul=

4.7.10.1. 説明

--exclude-mul オプションは以下のフォーマットでターゲットにする CPU コアが乗算器を持っているかどうかを指定します。このオプションを省略した場合はデフォルト値の(true)が適用されます。

```
--exclude-mul=[true/false]
```

tinyMicon MatisseCORE™にはいくつかのバリエーションがあり、乗算器が有る CPU コアと無い CPU コアがあります。このオプションを使用する事で各コアに適した機械語ファイルを生成する事ができます。

Table 23. --exclude-mul の設定値と出力結果

設定値	出力結果
true	乗算器が無い CPU コア用の機械語ファイルを生成する。
false	乗算器が有る CPU コア用の機械語ファイルを生成する。

4.7.10.2. 使用例

```
> mtcc "-Wl,--exclude-mul=true" input.c -o output.bin
> mtld --exclude-mul=true input.o -o output.bin
```

4.7.10.3. 詳細

このオプションはリンクするコンパイラランタイムライブラリ(CRT)の種類を決めます。CRT は乗算器がある CPU コア用の mul 命令を使用するものと、乗算器が無い CPU コア向けの mul 命令を使用しないもの 2 種類あります。
--exclude-mul オプションに true を適用すると乗算器が無い CPU コア向けの CRT がリンクされます。
--exclude-mul オプションに false を適用すると乗算器が有る CPU コア向けの CRT がリンクされます。
乗算処理は CRT 関数__mulhi3 と__mulsi3 として実装されていて、アプリケーション部分で乗算処理が必要な場合はこれらの CRT 関数が呼び出されます。
つまり--exclude-mul オプションに false を適用した場合でもアプリケーション部分で mul 命令は出力されません。このオプションの設定によってアプリケーション部分のコードは変更されません。

4.7.11. --crt

4.7.11.1. 説明

--crt オプションは以下のフォーマットで指定したファイルでランタイムライブラリを差し替えます。このオプションを省略した場合は--exclude-mul=オプションの設定値に応じたランタイムライブラリファイルが使用され、乗算器が無いコアには"C:¥Program Files¥ROHM¥Matisse¥C¥lib¥crt.a"が、乗算器が有るコアには"C:¥Program Files¥ROHM¥Matisse¥C¥lib¥crt_with_mul.a"がリンクされます。

このオプションで指定するランタイムライブラリファイルは以下の条件をすべて満たす必要があります。

- スタティックライブラリ
- リンカスクリプトから参照されるシンボルを全て定義している
- コンパイラ(mtcc)に必要なランタイムライブラリを全て実装している

© 2021 ROHM Co., Ltd.

4.7.11.2. 使用例

```
> mtcc "-Wl,--crt=my_crt.a" input.c -o output.bin
> mtld --crt=my_crt.a input.o -o output.bin
```

4.7.12. --libc

4.7.12.1. 説明

--libc オプションは以下のフォーマットでライブラリファイルを差し替えます。このオプションを省略した場合はデフォルトのライブラリファイルがリンクされます。デフォルトのライブラリファイルは"C:¥Program Files¥ROHM¥Matisse¥C¥lib¥libc.a"です。

このオプションで指定するライブラリファイルは以下の条件をすべて満たす必要があります。

- スタティックライブラリ
- コンパイラ(mtcc)がサポートするライブラリ関数を全て実装している

4.7.12.2. 使用例

```
> mtcc "-Wl,--libc=./libcanother.a" input.c -o output.bin
> mtld --libc=./libcanother.a input.o -o output.bin
```

4.7.13. -v

4.7.13.1. 説明

-v オプションはリンカ(mtld)のバージョン情報を標準出力に出力します。

4.7.13.2. 使用例

```
> mtld -v
```

4.7.14. -h

4.7.14.1. 説明

-h オプションはリンカ(mtld)のヘルプメッセージを標準出力に出力します。

4.7.14.2. 使用例

```
> mtld -h
```

© 2021 ROHM Co., Ltd.

5. ライブラリ関数

コンパイラ(mtcc)が提供するライブラリ関数について以下で説明します。

5.1. ライブラリファイル

ライブラリファイルにはコンパイラ(mtcc)がサポートする全てのライブラリ関数がまとめられています。デフォルトのライブラリファイルは"C:¥Program Files¥ROHM¥Matisse¥C¥lib¥libc.a"に置かれています。
リンカ(mtld)はライブラリファイルを読み込み、機械語命令ファイルに静的リンクします。リンカ(mtld)はプログラム実行に必要なライブラリ関数のみリンクします。
リンカ(mtld)の--libc オプションでリンクするライブラリファイルを変更する事も可能です。

5.2. ヘッダファイル

ライブラリ関数を使用するにはヘッダファイルをインクルードする必要があります。
コンパイラ(mtcc)が提供するライブラリのヘッダファイル一覧は以下になります。

Table 24. コンパイラ(mtcc)が提供するライブラリのヘッダファイル一覧

ヘッダファイル名	説明
assert.h	診断機能のためのヘッダファイル。
ctype.h	文字データの判別や変換のためのヘッダファイル。
errno.h	ライブラリ関数が使用するエラー番号のためのヘッダファイル。
inttypes.h	stdint.h を拡張し機能を追加するヘッダファイル。
iso646.h	演算子の代替を提供するヘッダファイル。
limits.h	各整数型の限界値を定義するヘッダファイル。
setjmp.h	大域ジャンプのためのヘッダファイル。
stdarg.h	可変長引数を持つ関数のためのヘッダファイル。
stdbool.h	bool 型を提供するヘッダファイル。
stddef.h	共通に使用されるマクロや型を提供するヘッダファイル。
stdint.h	サイズが標準化された整数型を提供するヘッダファイル。
stdio.h	標準入出力のためのヘッダファイル。
stdlib.h	標準ユーティリティに関するマクロや型や関数を提供するヘッダファイル。
string.h	文字列操作のためのヘッダファイル。
matisse/cpufunc.h	CPU を操作する特別なマクロを提供するヘッダファイル。
matisse/extmem.h	拡張プログラム領域に関するマクロを提供するヘッダファイル。
matisse/interrupt.h	割り込み処理に関するマクロを提供するヘッダファイル。
matisse/sections.h	セクションを指定するマクロを提供するヘッダファイル。
matisse/signature.h	出力ファイルにシグネチャを保存するためのヘッダファイル。
matisse/version.h	コンパイラやライブラリのバージョン情報を提供するヘッダファイル。
util/atomic.h	アトミックブロックに関するマクロや関数を提供するヘッダファイル。

© 2021 ROHM Co., Ltd.

5.3. ライブラリ関数詳細

各ヘッダファイルで使用できるライブラリ関数(または関数マクロ)を以下に説明します。
各関数が再入可能(リエントラント)かどうかについても記述しています。再入可能な関数は複数の処理箇所から同時に呼び出せます。
再入可能でない関数は複数個所から同時に呼び出されると意図しない結果を引き起こします。その場合はアトミックブロックを使用して、複数個所からの同時呼び出しを防いでください。

5.3.1. assert.h

Table 25. assert.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
assert	プログラムに診断機能を追加。	再入不可能

5.3.2. ctype.h

Table 26. ctype.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
isalnum	ASCII 英数字かどうかの判定。	再入可能
isalpha	ASCII 英字かどうかの判定。	再入可能
isascii	ASCII コードかどうかの判定。	再入可能
isblank	ブランク文字かどうかの判定。	再入可能
iscntrl	制御文字かどうかの判定。	再入可能
isdigit	10 進数字かどうかの判定。	再入可能
isgraph	標準空白文字以外の印字可能な文字かどうかの判定。	再入可能
islower	ASCII 英小文字かどうかの判定。	再入可能
isprint	標準空白文字を含む印字可能な文字かどうかの判定。	再入可能
ispunct	標準空白文字や ASCII 英数字以外の印字可能な文字かどうかの判定。	再入可能
isspace	標準空白文字かどうかの判定。	再入可能
isupper	ASCII 英大文字かどうかの判定。	再入可能
isxdigit	16 進数字かどうかの判定。	再入可能
toascii	ASCII 文字に変換。	再入可能
tolower	ASCII 英大文字を ASCII 英小文字に変換。	再入可能
toupper	ASCII 英小文字を ASCII 英大文字に変換。	再入可能

5.3.3. inttypes.h

Table 27. inttypes.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
imaxabs	符号あり 32bit 整数型の絶対値を計算。	再入可能
imaxdiv	符号あり 32bit 整数型同志の割り算を行い、商と剰余を返す。	再入可能

5.3.4. setjmp.h

Table 28. setjmp.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性

© 2021 ROHM Co., Ltd.

longjmp	setjmp 関数が呼び出された環境を復元。	再入可能
setjmp	後で longjmp 関数が使用できるように関数呼び出し時の環境を保存。	再入可能

5.3.5. stdarg.h

Table 29. stdarg.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
va_arg	可変長引数を取得。	再入可能
va_copy	現在の可変長引数リストをコピー。	再入可能
va_end	可変長引数の取得を終了。	再入可能
va_start	可変長引数の取得を開始。	再入可能

5.3.6. stddef.h

Table 30. stddef.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
offsetof	構造体メンバのオフセットを取得。	再入可能

5.3.7. stdint.h

Table 31. stdint.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
__CONCATenate	引数の文字を連結。	再入可能
__CONCAT	引数の文字を連結。	再入可能
INT8_C	int8_t にキャスト。	再入可能
UINT8_C	uint8_t にキャスト。	再入可能
INT16_C	int16_t にキャスト。	再入可能
UINT16_C	uint16_t にキャスト。	再入可能
INT32_C	int32_t にキャスト。	再入可能
UINT32_C	uint32_t にキャスト。	再入可能
INTMAX_C	intmax_t にキャスト。	再入可能
UINTMAX_C	uintmax_t にキャスト。	再入可能

5.3.8. stdio.h

Table 32. stdio.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
FDEV_SETUP_STREAM	ストリームのオープン。	再入不可能
fdev_close	ストリームのクローズ。	再入可能
fprintf	ストリームへの書式付き文字列出力。	再入可能
fputc	ストリームへの文字出力。	再入可能
fputs	ストリームへの文字列出力。	再入可能
fwrite	ストリームへの書き込み。	再入可能
printf	標準出力への書式付き文字列出力。	再入不可能
putc	標準出力への文字出力。	再入不可能
putchar	標準出力への文字出力。	再入不可能

© 2021 ROHM Co., Ltd.

puts	標準出力への文字列出力。	再入不可能
snprintf	文字列への書式付き文字列出力。最大出力サイズを指定可能。	再入可能
sprintf	文字列への書式付き文字列出力。	再入可能
vfprintf	ストリームへの書式付き文字列出力。	再入可能
vprintf	標準出力への書式付き文字列出力。	再入可能
vsnprintf	文字列への書式付き文字列出力。最大出力サイズを指定可能。	再入可能
vsprintf	文字列への書式付き文字列出力。	再入可能

5.3.9. stdlib.h

Table 33. stdlib.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
abort	プログラムを異常終了させる。	再入可能
abs	int 型の絶対値を計算。	再入可能
atoi	10 進数文字列を int 型に変換。	再入可能
atol	10 進数文字列を long 型に変換。	再入可能
bsearch	二分探索処理。	再入可能
div	int 型同志の割り算を行い、商と剰余を返す。	再入可能
exit	プログラムを終了させる。	再入可能
labs	long 型の絶対値を計算。	再入可能
ldiv	long 型同志の割り算を行い、商と剰余を返す。	再入可能
qsort	クイックソート処理。	再入可能
rand	疑似乱数を生成。	再入不可能
rand_r	疑似乱数を生成。	再入可能
srand	疑似乱数を初期化。	再入可能
strtol	文字列を long 型に変換。	再入不可能
strtoul	文字列を unsigned long 型に変換。	再入不可能

5.3.10. string.h

Table 34. string.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
memccpy	メモリをコピー。最大コピーサイズを指定可能。	再入可能
memchr	メモリの先頭から文字を検索。	再入可能
memcmp	メモリを比較。	再入可能
memcpy	メモリをコピー。	再入可能
memmove	メモリを移動。	再入可能
memrchr	メモリの末尾から文字を検索。	再入可能
memset	メモリに指定したデータをセット。	再入可能
strcat	文字列を連結。	再入可能
strchr	文字列の先頭から文字を検索。	再入可能
strcmp	文字列を比較。	再入可能
strcpy	文字列をコピー。	再入可能
strcspn	文字列先頭から、指定した文字列のどの文字でもない部分が何文字続くか調べる。	再入可能
strlcat	文字列を連結。最大文字列サイズを指定可能。	再入可能

© 2021 ROHM Co., Ltd.

strlcpy	文字列をコピー。最大文字列サイズを指定可能。	再入可能
strlen	文字列長を取得。	再入可能
strncat	文字列を連結。連結する文字数を指定可能。	再入可能
strncmp	文字列を比較。比較する文字数を指定可能。	再入可能
strncpy	文字列をコピー。コピーする文字数を指定可能。	再入可能
strnlen	文字列長を取得。文字列最大長を指定可能。	再入可能
strpbrk	文字列先頭から、指定した文字列に含まれる文字が見つかった最初の位置を返す。	再入可能
strrchr	文字列の末尾から文字を検索。	再入可能
strspn	文字列先頭から、指定した文字列に含まれる文字が何文字続くか調べる。	再入可能
strstr	文字列先頭から指定した文字列を検索。	再入可能
strtok	文字列をトークンに分割。	再入不可能
strtok_r	文字列をトークンに分割。	再入可能

5.3.11. matisse/cpufunc.h

Table 35. matisse/cpufunc.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
_NOP	nop 命令を出力。	再入可能
_HLT	hlt 命令を出力。	再入可能

5.3.12. matisse/interrupt.h

Table 36. matisse/interrupt.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
ei	割り込みフラグを有効化。	再入可能
isei	割り込みフラグが有効かどうかを取得。	再入可能
di	割り込みフラグを無効化。	再入可能
swi	ソフトウェア割り込みでのマスカブル割り込みを発生。	再入可能
swnmi	ソフトウェア割り込みでのノンマスカブル割り込みを発生。	再入可能

5.3.13. matisse/sections.h

Table 37. matisse/sections.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
SECTION_TEXT	.text 領域に配置。	再入可能
SECTION_RODATA	.rodata 領域に配置。	再入可能
SECTION_DATA	.data 領域に配置。	再入可能
SECTION_BSS	.bss 領域に配置。	再入可能
SECTION_NOINIT	.noinit 領域に配置。	再入可能

5.3.14. util/atomic.h

Table 38. util/atomic.h で使用できるライブラリ関数

関数名/関数マクロ名	説明	再入可能性
ATOMIC_BLOCK	割り込み禁止ブロックを作成する。	再入可能

© 2021 ROHM Co., Ltd.

NONATOMIC_BLOCK	割り込み可能ブロックを作成する。	再入可能
ATOMIC_RESTORESTATE	ATOMIC_BLOCK の引数として使用され、ブロック終了後に割り込みフラグをブロック開始以前の状態に戻す。	再入可能
ATOMIC_FORCEON	ATOMIC_BLOCK の引数として使用され、ブロック終了後に割り込みフラグを有効にする。	再入可能
NONATOMIC_RESTORESTATE	NONATOMIC_BLOCK の引数として使用され、ブロック終了後に割り込みフラグをブロック開始以前の状態に戻す。	再入可能
NONATOMIC_FORCEOFF	NONATOMIC_BLOCK の引数として使用され、ブロック終了後に割り込みフラグを無効にする。	再入可能

© 2021 ROHM Co., Ltd.

5.4. 型定義詳細

各ヘッダファイルで定義されている型を以下に説明します。

5.4.1. inttypes.h

Table 39. inttypes.h で定義されている型

型名	定義

_Lldiv_t	struct {
	intmax_t quot;
	intmax_t rem;
	};

imaxdiv_t	_Lldiv_t

5.4.2. setjmp.h

Table 40. setjmp.h で定義されている型

型名	定義

jmp_buf[1]	struct _jmp_buf {
	unsigned char _jb[_JBLEN];
	};

5.4.3. stdarg.h

Table 41. stdarg.h で定義されている型

型名	定義
__gnuc_va_list	__builtin_va_list
va_list	__gnuc_va_list

5.4.4. stddef.h

Table 42. stddef.h で定義されている型

型名	定義
__PTRDIFF_TYPE__	int
ptrdiff_t	__PTRDIFF_TYPE__
__SIZE_TYPE__	unsigned int
size_t	__SIZE_TYPE__

5.4.5. stdint.h

Table 43. stdint.h で定義されている型

型名	定義
int8_t	signed char
uint8_t	unsigned char
int16_t	signed int
uint16_t	unsigned int

© 2021 ROHM Co., Ltd.

int32_t	signed long
uint32_t	unsigned long
intptr_t	int16_t
uintptr_t	uint16_t
int_least8_t	int8_t
uint_least8_t	uint8_t
int_least16_t	int16_t
uint_least16_t	uint16_t
int_least32_t	int32_t
uint_least32_t	uint32_t
int_fast8_t	int8_t
uint_fast8_t	uint8_t
int_fast16_t	int16_t
uint_fast16_t	uint16_t
int_fast32_t	int32_t
uint_fast32_t	uint32_t
intmax_t	int32_t
uintmax_t	uint32_t

5.4.6. stdio.h

Table 44. stdio.h で定義されている型

型名	定義

__file	struct {
	char *buf;
	uint8_t flags;
	int size;
	int len;
	int (*put)(char, struct __file *);
	int (*get)(struct __file *);
	void *updata;
	};

FILE	__file

5.4.7. stdlib.h

Table 45. stdlib.h で定義されている型

型名	定義

div_t	struct {
	int quot;
	int rem;
	};

ldiv_t	struct {
	long quot;
	long rem;
	};

__compare_fn_t	int (*)(const void *, const void *)

© 2021 ROHM Co., Ltd.

5.5. マクロ詳細

各ヘッダファイルで使用できるマクロを以下に説明します。

5.5.1. errno.h

Table 46. errno.h で使用できるマクロ

マクロ名	定義	値
EDOM	33	33 (0x21)
ERANGE	34	34 (0x22)
ENOSYS	((int)(66081697 & 0x7fff))	21409 (0x53a1)
EINTR	((int)(2453066 & 0x7fff))	28234 (0x6e4a)
E2BIG	ENOERR	11762 (0x2df2)
EACCES	ENOERR	11762 (0x2df2)
EADDRINUSE	ENOERR	11762 (0x2df2)
EADDRNOTAVAIL	ENOERR	11762 (0x2df2)
EAFNOSUPPORT	ENOERR	11762 (0x2df2)
EAGAIN	ENOERR	11762 (0x2df2)
EALREADY	ENOERR	11762 (0x2df2)
EBADF	ENOERR	11762 (0x2df2)
EBUSY	ENOERR	11762 (0x2df2)
ECHILD	ENOERR	11762 (0x2df2)
ECONNABORTED	ENOERR	11762 (0x2df2)
ECONNREFUSED	ENOERR	11762 (0x2df2)
ECONNRESET	ENOERR	11762 (0x2df2)
EDEADLK	ENOERR	11762 (0x2df2)
EDESTADDRREQ	ENOERR	11762 (0x2df2)
EEXIST	ENOERR	11762 (0x2df2)
EFAULT	ENOERR	11762 (0x2df2)
EFBIG	ENOERR	11762 (0x2df2)
EHOSTUNREACH	ENOERR	11762 (0x2df2)
EILSEQ	ENOERR	11762 (0x2df2)
EINPROGRESS	ENOERR	11762 (0x2df2)
EINVAL	ENOERR	11762 (0x2df2)
EIO	ENOERR	11762 (0x2df2)
EISCONN	ENOERR	11762 (0x2df2)
EISDIR	ENOERR	11762 (0x2df2)
ELOOP	ENOERR	11762 (0x2df2)
EMFILE	ENOERR	11762 (0x2df2)
EMLINK	ENOERR	11762 (0x2df2)
EMSGSIZE	ENOERR	11762 (0x2df2)
ENAMETOOLONG	ENOERR	11762 (0x2df2)
ENETDOWN	ENOERR	11762 (0x2df2)
ENETRESET	ENOERR	11762 (0x2df2)
ENETUNREACH	ENOERR	11762 (0x2df2)
ENFILE	ENOERR	11762 (0x2df2)
ENOBUFS	ENOERR	11762 (0x2df2)

© 2021 ROHM Co., Ltd.

ENODEV	ENOERR	11762 (0x2df2)
ENOENT	ENOERR	11762 (0x2df2)
ENOEXEC	ENOERR	11762 (0x2df2)
ENOLCK	ENOERR	11762 (0x2df2)
ENOMEM	ENOERR	11762 (0x2df2)
ENOMSG	ENOERR	11762 (0x2df2)
ENOPROTOOPT	ENOERR	11762 (0x2df2)
ENOSPC	ENOERR	11762 (0x2df2)
ENOTCONN	ENOERR	11762 (0x2df2)
ENOTDIR	ENOERR	11762 (0x2df2)
ENOTEMPTY	ENOERR	11762 (0x2df2)
ENOTSOCK	ENOERR	11762 (0x2df2)
ENOTTY	ENOERR	11762 (0x2df2)
ENXIO	ENOERR	11762 (0x2df2)
EOPNOTSUPP	ENOERR	11762 (0x2df2)
EPERM	ENOERR	11762 (0x2df2)
EPIPE	ENOERR	11762 (0x2df2)
EPROTONOSUPPORT	ENOERR	11762 (0x2df2)
EPROTOTYPE	ENOERR	11762 (0x2df2)
EROFS	ENOERR	11762 (0x2df2)
ESPIPE	ENOERR	11762 (0x2df2)
ESRCH	ENOERR	11762 (0x2df2)
ETIMEDOUT	ENOERR	11762 (0x2df2)
EWOULDBLOCK	ENOERR	11762 (0x2df2)
EXDEV	ENOERR	11762 (0x2df2)
ENOERR	((int)(66072050 & 0xffff))	11762 (0x2df2)

5.5.2. iso646.h

Table 47. iso646.h で使用できるマクロ

マクロ名	定義	値
and	&&	&&
and_eq	&=	&=
bitand	&	&
bitor	|	|
compl	~	~
not	!	!
not_eq	!=	!=
or	|	|
or_eq	|=	|=
xor	^	^
xor_eq	^=	^=

© 2021 ROHM Co., Ltd.

5.5.3. limits.h

Table 48. limits.h で使用できるマクロ

マクロ名	定義	値
CHAR_BIT	__CHAR_BIT__	8 (0x08)
MB_LEN_MAX	1	1 (0x01)
SCHAR_MIN	(-SCHAR_MAX - 1)	-128 (0x80)
SCHAR_MAX	__SCHAR_MAX__	127 (0x7F)
UCHAR_MAX	(SCHAR_MAX * 2 + 1)	255 (0xFF)
CHAR_MIN	SCHAR_MIN	-128 (0x80)
CHAR_MAX	SCHAR_MAX	127 (0x7F)
SHRT_MIN	(-SHRT_MAX - 1)	-32768 (0x8000)
SHRT_MAX	__SHRT_MAX__	32767 (0x7FFF)
USHRT_MAX	(SHRT_MAX * 2U + 1U)	65535 (0xFFFF)
INT_MIN	(-INT_MAX - 1)	-32768 (0x8000)
INT_MAX	__INT_MAX__	32767 (0x7FFF)
UINT_MAX	(INT_MAX * 2U + 1U)	65535 (0xFFFF)
LONG_MIN	(-LONG_MAX - 1L)	-2147483648 (0x80000000)
LONG_MAX	__LONG_MAX__	2147483647 (0x7FFFFFFF)
ULONG_MAX	(LONG_MAX * 2UL + 1UL)	4294967295 (0xFFFFFFFF)

5.5.4. setjmp.h

Table 49. setjmp.h で使用できるマクロ

マクロ名	定義	値
JBLEN	11	11 (0xb)
__ATTR__NORETURN__	__attribute__((__noreturn__))	__attribute__((__noreturn__))

5.5.5. stdbool.h

Table 50. stdbool.h で使用できるマクロ

マクロ名	定義	値
bool	_Bool	_Bool
true	1	1 (0x1)
false	0	0 (0x0)
__bool_true_false_are_defined	1	1 (0x1)

5.5.6. stddef.h

Table 51. stddef.h で使用できるマクロ

マクロ名	定義	値
NULL	((void *)0)	((void *)0)

© 2021 ROHM Co., Ltd.

5.5.7. stdint.h

Table 52. stdint.h で使用できるマクロ

マクロ名	定義	値
INT8_MAX	0x7F	127 (0x7F)
INT8_MIN	(-INT8_MAX - 1)	-128 (0x80)
UINT8_MAX	(INT8_MAX * 2 + 1)	255 (0xFFFF)
INT16_MAX	0x7fff	32767 (0x7FFF)
INT16_MIN	(-INT16_MAX - 1)	-32768 (0x8000)
UINT16_MAX	(__CONCAT(INT16_MAX, U) * 2U + 1U)	65535 (0xFFFF)
INT32_MAX	0x7fffffffL	2147483647 (0x7FFFFFFF)
INT32_MIN	(-INT32_MAX - 1L)	-2147483648 (0x80000000)
UINT32_MAX	(__CONCAT(INT32_MAX, U) * 2UL + 1UL)	4294967295 (0xFFFFFFFF)
INT_LEAST8_MAX	INT8_MAX	127 (0x7F)
INT_LEAST8_MIN	INT8_MIN	-128 (0x80)
UINT_LEAST8_MAX	UINT8_MAX	255 (0xFF)
INT_LEAST16_MAX	INT16_MAX	32767 (0x7FFF)
INT_LEAST16_MIN	INT16_MIN	-32768 (0x8000)
UINT_LEAST16_MAX	UINT16_MAX	65535 (0xFFFF)
INT_LEAST32_MAX	INT32_MAX	2147483647 (0x7FFFFFFF)
INT_LEAST32_MIN	INT32_MIN	-2147483648 (0x80000000)
UINT_LEAST32_MAX	UINT32_MAX	4294967295 (0xFFFFFFFF)
INT_FAST8_MAX	INT8_MAX	127 (0x7F)
INT_FAST8_MIN	INT8_MIN	-128 (0x80)
UINT_FAST8_MAX	UINT8_MAX	255 (0xFF)
INT_FAST16_MAX	INT16_MAX	32767 (0x7FFF)
INT_FAST16_MIN	INT16_MIN	-32768 (0x8000)
UINT_FAST16_MAX	UINT16_MAX	65535 (0xFFFF)
INT_FAST32_MAX	INT32_MAX	2147483647 (0x7FFFFFFF)
INT_FAST32_MIN	INT32_MIN	-2147483648 (0x80000000)
UINT_FAST32_MAX	UINT32_MAX	4294967295 (0xFFFFFFFF)
INTPTR_MAX	INT16_MAX	32767 (0x7FFF)
INTPTR_MIN	INT16_MIN	-32768 (0x8000)
UINTPTR_MAX	UINT16_MAX	65535 (0xFFFF)
INTMAX_MAX	INT32_MAX	2147483647 (0x7FFFFFFF)
INTMAX_MIN	INT32_MIN	-2147483648 (0x80000000)
UINTMAX_MAX	UINT32_MAX	4294967295 (0xFFFFFFFF)
PTRDIFF_MAX	INT16_MAX	32767 (0x7FFF)
PTRDIFF_MIN	INT16_MIN	-32768 (0x8000)
SIG_ATOMIC_MAX	INT8_MAX	127 (0x7F)
SIG_ATOMIC_MIN	INT8_MIN	-128 (0x80)
SIZE_MAX	UINT16_MAX	65535 (0xFFFF)

© 2021 ROHM Co., Ltd.

5.5.8. matisse/extmem.h

Table 53. matisse/extmem.h で使用できるマクロ

マクロ名	定義	値
EXTMEM_FUNCTIONS_BEGIN_HERE	_Pragma("clang section text=¥".extext¥")	_Pragma("clang section text=¥".extext¥")
EXTMEM_FUNCTIONS_END_HERE	_Pragma("clang section text=¥"¥")	_Pragma("clang section text=¥".extext¥")
EXTMEM_THIS_FUNCTION	SECTION_MACRO(extext)	__atrribute__ ((section(".extext"))
EXTMEM_EXCLUDE_THIS_FUNCTION	SECTION_TEXT	__attribute ((section(".text"))

5.5.9. matisse/signature.h

Table 54. matisse/signature.h で使用できるマクロ

マクロ名	定義	値
SIGNATURE_HEADER	0x6D	109 (0x6D)

5.5.10. matisse/version.h

Table 55. matisse/version.h で使用できるマクロ

マクロ名	定義	値
__MTCC_MAJOR__	mtcc V2.00.00 の場合: 2 mtcc V3.00.00 の場合: 3	mtcc V2.00.00 の場合: 2 (0x2) mtcc V3.00.00 の場合: 3 (0x3)
__MTCC_MINOR__	mtcc V2.00.00 の場合: 0 mtcc V3.00.00 の場合: 0	mtcc V2.00.00 の場合: 0 (0x0) mtcc V3.00.00 の場合: 0 (0x0)
__MTCC_REVISION__	mtcc V2.00.00 の場合: 0 mtcc V3.00.00 の場合: 0	mtcc V2.00.00 の場合: 0 (0x0) mtcc V3.00.00 の場合: 0 (0x0)
__MATISSE_LIBC_VERSION_STRING__	"1.1.1"	"1.1.1"
__MATISSE_LIBC_VERSION__	10101UL	10101 (0x2775)
__MATISSE_LIBC_DATE_STRING__	"20220322"	"20220322"
__MATISSE_LIBC_DATE__	20220322UL	20220322 (0x13489A2)
__MATISSE_LIBC_MAJOR__	1	1 (0x1)
__MATISSE_LIBC_MINOR__	1	1 (0x1)
__MATISSE_LIBC_REVISION__	1	1 (0x1)

© 2021 ROHM Co., Ltd.

6. 固有の機能

コンパイラ(mtcc)に固有の機能を説明します。

6.1. 拡張プログラム領域

tinyMicon MatisseCORE™にはいくつかのバリエーションがあり、通常の 16bit メモリ領域(0x0000 - 0xFFFF)以外に拡張プログラム領域(0x10000 - 0x1FFFF)が使用できるコアもあります。
通常メモリ領域と拡張プログラム領域の違いは以下です。

Table 56. 通常メモリ領域と拡張プログラム領域の違い

メモリ種別	プログラムデータの配置	定数データの配置	データ領域(.data, .bss、スタック)として使用
通常メモリ領域	配置可能	配置可能	使用可能
拡張プログラム領域	配置可能	配置不可能	使用不可能

6.1.1. 使用方法

拡張プログラム領域を使用するには以下の 2 つの手順を行う必要があります。

1. C 言語ソースファイルにヘッダファイル<matisse/extmem.h>をインクルードして、拡張プログラム領域用のマクロを使用する。
2. 拡張プログラム領域を有効にするコマンドラインオプションをコンパイル時に指定する。

6.1.2. ソースファイル例

以下のようにソースファイルを記述します。拡張プログラム領域用マクロの詳細な説明は<matisse/extmem.h>のファイル内に記載されたコメントを参照してください。

```
#include <matisse/extmem.h>

// All the functions defined after this macro will be placed in the extended program area.
EXTMEM_FUNCTIONS_BEGIN_HERE

void func1(){
   // this function will be placed in the extended program area.
}

void func2(){
   // this function will be placed in the extended program area.
}

void func3(){
   // this function will be placed in the extended program area.
}

void func4(){
   // this function will be placed in the extended program area.
}

int main(){
```

© 2021 ROHM Co., Ltd.

```
    // this function will be placed in the extended program area.
    return 0;
}
```

6.1.3. コンパイルオプション例

拡張プログラム領域を使用するソースファイルをコンパイルするには以下のようにコマンドラインオプションで--ex-program-size=を指定します。

```
> mtcc "-Wl,--ex-program-size=10,-Map=mapfile.map" input.c -o output.bin
```

--ex-program-size=オプションで拡張プログラム領域のサイズを指定しています。拡張プログラム領域のサイズが足りない場合はコンパイルエラーが発生します。
-Map=オプションでマップファイルを指定しています。マップファイルを見れば関数が拡張プログラム領域に配置されているかどうかを確認できます。

© 2021 ROHM Co., Ltd.

6.2. 割り込みハンドラ

tinyMicon MatisseCORE™はマスカブル割り込みとノンマスカブル割り込みの 2 種類の割り込みをサポートしています。
コンパイラ(mtcc)では割り込みが発生した時に実行される割り込みハンドラを C 言語の関数として定義できます。

6.2.1. 使用方法

割り込みハンドラを使用するには以下の 2 つの手順を行う必要があります。

1. C 言語ソースファイルにヘッダファイル<matisse/interrupt.h>をインクルードする。
2. マクロ ISR() を使用して割り込みハンドラを定義する。ISR(IRQ_vect)がマスカブル割り込みの割り込みハンドラ、ISR(NMI_vect)がノンマスカブル割り込みの割り込みハンドラになる。

6.2.2. ソースファイル例

6.2.2.1. 割り込みハンドラ

マクロ ISR(IRQ_vect)でマスカブル割り込みハンドラが、マクロ ISR(NMI_vect)でノンマスカブル割り込みハンドラが作成できます。

```
#include <matisse/interrupt.h>

volatile int irqCount = 0;
volatile int nmiCount = 0;

ISR(IRQ_vect){
   // this function is the handler of the maskable interrupt.
   irqCount++;
}

ISR(NMI_vect){
   // this function is the handler of the non-maskable interrupt.
   nmiCount++;
}
```

6.2.2.2. 割り込みフラグのオン/オフ

ヘッダファイル<matisse/cpufunc.h>で定義されているマクロを使用して割り込みフラグのオン/オフもできます。

```
#include <matisse/interrupt.h>
#include <matisse/cpufunc.h>

volatile int irqCount = 0;

ISR(IRQ_vect){
   // this function is the handler of the maskable interrupt.
   irqCount++;
}

int main(){
   ei(); // the interrupt flag turns on.
```

© 2021 ROHM Co., Ltd.

```
    timer_start();
    while(irqCount < 100){
        do_some_event_loop();
      }
      di(); // the interrupt flag turns off.
}
```

6.2.2.3. アトミックブロック

ヘッダファイル<util/atomic.h>で定義されているマクロを使用してアトミックブロックを作成する事ができます。アトミックブロックとはマスカブル割込みが発生しないコードブロックの事です。アトミックブロック用マクロの詳細な説明は<util/atomic.h>のファイル内に記載されたコメントを参照してください。

```
#include <matisse/interrupt.h>
#include <matisse/cpufunc.h>
#include <util/atomic.h>

volatile int irqCount = 0;

ISR(IRQ_vect){
    // this function is the handler of maskable interrupt.
    irqCount++;
}

int main(){
    volatile int irqCountCopy = 0;
    ei(); // the interrupt flag turns on.
    timer_start();
    while(irqCount < 100){
          ATOMIC_BLOCK(ATOMIC_RESTORESTATE){
              // Maskable interrupts don't occur in this block, so copying counter is always performed successfully.
              irqCountCopy = irqCount;
          }
        do_some_event_loop();
    }
      di(); // the interrupt flag turns off.
}
```

6.2.3. 注意事項

割り込みハンドラにはサイズ制限があり、256byte 以下にする必要があります。割り込みハンドラ内に多数の処理を記述する事は避けてください。どうしても必要な場合は別の関数に切り出して、割り込みハンドラから呼び出すようにしてください。
割り込みハンドラのサイズ制限を超過するとコンパイル時にエラーが発生します。

© 2021 ROHM Co., Ltd.

6.3. 標準出力

tinyMicon MatisseCORE™は LSI 組み込みシステム専用の CPU なので標準入出力(stdin, stdout, stderr)をサポートしません。
ただしコマンドラインデバッガ(mtsim)によるデバッグ実行時のみ疑似的な標準出力(stdout)が使用できます。

6.3.1. 使用方法

コマンドラインデバッガ(mtsim)による疑似的な標準出力を使用するには以下の手順を行う必要があります。

1. コマンドラインデバッガ(mtsim)をインストールする。
2. ソースファイルに標準出力機能を有効にする処理を記述する。
3. コマンドラインデバッガ(mtsim)の標準出力機能を有効にするスクリプトファイルを記述する。
4. ソースファイルをビルドしてバイナリファイルを生成する。
5. 作成したスクリプトファイルとバイナリファイルを使用して msim を実行する。

6.3.2. ソースファイル例

以下のようにソースファイルを記述します。
FDEV_SETUP_STREAM マクロが作成する特殊なストリームについてはヘッダファイル<stdio.h>のファイル内に記載されたコメントを参照してください。

```
#include <stdio.h>
#include <matisse/interrupt.h>

int myPutChar(char c, FILE* file);

// Create a special output stream.
FILE myStdout = FDEV_SETUP_STREAM(myPutChar, NULL, _FDEV_SETUP_WRITE);

// This is the callback function for processing every char data in the output stream.
int myPutChar(char c, FILE* file)
{
   // Invoke interrupt to access the hook address.
   swnmi();
   return 0;
}

int main (void)
{
   // Overwrite stdout with a special output stream.
   stdout = &myStdout;

   // Each output data is processed by the callback (myPutChar), and then passed to the hook function.
   // The hook function outputs characters to the standard output of mtsim.
   printf("Hello, world!¥n);

   return 0;
}
```

© 2021 ROHM Co., Ltd.

6.3.3. デバッガスクリプトファイル例

以下のようにデバッガを制御するスクリプトファイルを記述し、script.txt というファイル名で保存します。
デバッガ(mtsim)の hook コマンドを使用して、フックアドレスとフック DLL 内の関数を結びつけます。hook コマンドによってデバッガ(mtsim)がフックアドレスにアクセスした時に DLL 内のフック関数が実行されるようになります。
フック DLL はデフォルトでは"C:¥Program Files¥ROHM¥Matisse¥mtsim¥bin¥StdoutHook.dll"にあります。
0x02 はノンマスカブル割り込みの割り込みベクタアドレスです。ソースコードで swnmi()が実行されるとノンマスカブル割り込みが発生して 0x02 に実行が移り、フック関数が実行されます。

```
set echo off
b 0x02
b 0x1a
hook 0x2 C:¥Program Files¥ROHM¥Matisse¥mtsim¥bin¥StdoutHook.dll
c
q
```

6.3.4. デバッガ実行

以下のようなコマンドでデバッガを実行すると mtsim の標準出力に文字列が出力されます。
-sim オプションはシミュレーションモードでの実行、-s オプションはスクリプトファイルを指定しています。コマンドラインの詳細は mtsim のドキュメントを参照してください。
コマンドラインオプションは用途に応じて変更可能です。

```
> mtsim -sim -sscript.txt -q output.bin
Hello, world!
>
```

© 2021 ROHM Co., Ltd.

6.4. SystemInit 関数

プログラム起動時にシステムの初期化処理が必要な場合は SystemInit 関数を作成してください。SystemInit 関数は CPU 動作周波数の変更、ペリフェラルの設定、メモリマップの変更、スタックの変更等に使用されます。
SystemInit 関数を作成した場合は、SystemInit 関数がスタートアップルーチンから自動的に呼び出されます。
SystemInit 関数が呼び出されるタイミングは以下になります。

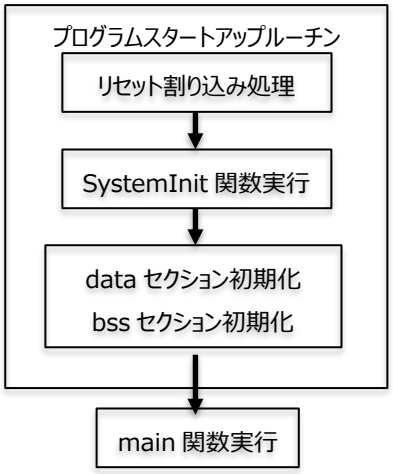

Figure 1. SystemInit 関数の呼び出しシーケンス

6.4.1. ソースファイル例

以下のように SystemInit 関数を記述すると起動時に自動的に実行されます。

```
void SystemInit()
{
// Do some initialization here.
}
```

6.5. noinit セクション

bss セクションの変数はスタートアップルーチンで 0 に初期化されますが、初期化されたくない場合は noinit セクションを使用してください。
noinit セクションに配置された変数は初期化されません。

6.5.1. 使用方法

C 言語ソースファイルにヘッダファイル<matisse/sections.h>をインクルードして、マクロ SECTION_NOINIT を対象の変数に使用してください。

6.5.2. ソースファイル例

```
#include <matisse/sections.h>

SECTION_NOINIT int notInitializedVariable;

SECTION_NOINIT int notInitializedArray[10];
```

© 2021 ROHM Co., Ltd.

6.5.3. 注意事項

noinit セクションに配置される変数には初期値を指定する事はできません。
static 変数を noinit セクションに配置する事はできません。

6.6. シグネチャデータ

C 言語ソースファイルにヘッダファイル<matisse/signature.h>をインクルードする事でコンパイラ(mtcc)が出力するファイル(バイナリ形式ファイルと ELF 形式ファイルの両方)に識別用のシグネチャデータを埋め込むことができます。

シグネチャデータは 7byte のデータで構成され、フォーマットは以下の通りです。

Table 57. シグネチャデータ のフォーマット

データインデックス	説明
0	0x6D(ASCII コードの'M'。Matisse の頭文字。)
1	mtcc major version number
2	mtcc minor version number
3	mtcc revision number
4	matisse libc major version number
5	matisse libc minor version number
6	matisse libc revision number

シグネチャデータが埋め込まれる場所は以下の通りです。

Table 58. シグネチャデータが埋め込まれる場所

ファイル種別	場所
バイナリ形式ファイル	ROM データの末尾
ELF 形式ファイル	signature セクション

6.6.1. 使用方法

C 言語ソースファイルにヘッダファイル<matisse/signature.h>をインクルードするだけでシグネチャデータが出力ファイルに埋め込まれます。

6.6.2. ソースファイル例

```
#include <matisse/signature.h>
```

6.6.3. 注意事項

ヘッダファイル<matisse/signature.h>はプログラム全体のソースファイル内の 1 ファイルでだけインクルード可能です。複数ファイルでインクルードした場合はコンパイル時にエラーが発生します。

6.7. インラインアセンブラ

コンパイラ(mtcc)は簡易なインラインアセンブラ機能をサポートしています。

© 2021 ROHM Co., Ltd.

6.7.1. 文法

インラインアセンブラ文は以下のフォーマットで記述します。

```
__asm__ __volatile__ (assembler_code);
```

コンパイラの最適化で削除されないように必ず__volatile__を指定してください。

assembler_code 部分にはアセンブラ命令がそのまま記述できます。

6.7.2. ソースファイル例

インラインアセンブラは C 言語関数の中に以下のように記述します。

```
void sample_func() {
  // nop instruction
   __asm__ __volatile__ ("nop¥n¥t");

   // load values to r10 and r11
   __asm__ __volatile__ ("ldr r10, 0x10¥n¥t"
                         "ldr r11, 0x20¥n¥t);

   // call other_func using r8 and r9
   __asm__ __volatile__ ("ldr r8, lo8(other_func)¥n¥t"
                         "ldr r9, hi8(other_func)¥n¥t"
                         "call er8¥n¥t");
}
```

6.7.3. 注意事項

インラインアセンブラで使用したレジスタが C 言語ソースファイル部分で使用しているレジスタと衝突しないように注意してください。

レジスタが衝突していると値が書き換えられ、意図しない結果を引き起こします。

© 2021 ROHM Co., Ltd.

6.8. pragma

コンパイラ(mtcc)は pragma 機能をサポートしています。ソースコードに pragma を記述する事でコンパイラに追加の情報を与えられます。

6.8.1. 使用方法

動作を保証している#pragmra 指令は以下になります。

Table 59. 動作を保証している#pragma 指令

pragma	説明
#pragma clang section [section_type="name"]	グローバルオブジェクトのセクション名を name に変更する。 section_type には以下が使用できる。 ● text ● rodata ● data ● bss

6.8.2. ソースファイル例

```
#pragma clang section text=".text.subsection" // All global objects that were supposed to be placed in .text section after this line will be placed into .text.subsection..

// This function will be placed into .text.subsection .
void func1(){
}

// This function will also be placed into .text.subsection .
void func2(){
}

#pragma clang section text="" // Restore the settings of .text section to the default.
```

6.8.3. 注意事項

pragma 機能を理解せずに使用した場合は意図しないコンパイルエラーや不具合を招きます。必ず内容を理解した上で使用してください。

6.9. attribute

コンパイラ(mtcc)は atribute 機能をサポートしています。ソースコードに attribute を記述する事でプログラム内のオブジェクトの属性を指定する事ができます。

6.9.1. 使用方法

動作を保証している attribute 指令は以下になります。

© 2021 ROHM Co., Ltd.

Table 60. 動作を保証している attribute 指令

attribute	説明
__attribute__ ((__const__))	関数に指定した場合に、グローバル変数や static 変数を変更しない関数(const function)だとコンパイラに通知する。
__attribute__((__interrupt__))	関数に指定した場合に、割り込みハンドラだとコンパイラに通知する。
__attribute__((__nodebug__))	指定したオブジェクトに関するデバッグ情報が生成されなくなる。
__attribute__((__noreturn__))	関数に指定した場合に、呼び出し元に帰らない関数になる。
__attribute__((__pure__))	関数に指定した場合に、同じ引数で呼び出した場合は必ず同じ値が返ってくる関数(pure function)だとコンパイラに通知する。
__attribute__ ((section ("[section_name]")))	指定したオブジェクトを section_name に配置する。
__attribute__((unused))	変数・関数・引数に指定できる。対象の未使用警告を抑制する。

6.9.2. ソースファイル例

```
__attribute__((__const__)) void func1() {
// This function will be given the const attribute.
}

__attribute__((__nodebug__)) __attribute__((__pure__)) void func2() {
// This function will be given the nodebug and the pure attributes.
}

void func3() __attribute__(__pure__); // Adding the attribute on function declaration.

int var1 __attribute__((__nodebug__)); // This variable will be given the nodebug attribute.
int var2 __attribute__((__nodebug__)) = 10; // This variable will also be given the nodebug attribute.
```

6.9.3. 注意事項

attribute 機能を理解せずに使用した場合は意図しないコンパイルエラーや不具合を招きます。必ず内容を理解した上で使用してください。

© 2021 ROHM Co., Ltd.

6.10. 事前定義済みマクロ

コンパイラ(mtcc)は事前定義済みマクロを持っています。これらのマクロはコンパイラ(mtcc)の内部に予め定義されていて、コンパイラの種類やコンパイラのバージョンを取得するために使用されます。

6.10.1. 使用方法

コンパイラ(mtcc)で使用できる事前定義済みマクロは以下になります。

Table 61. コンパイラ(mtcc)で使用できる事前定義済みマクロ

マクロ名	値
MATISSE	1
__MATISSE	1
__MATISSE__	1

6.10.2. ソースファイル例

```
#ifdef __MATISSE__
    // Matisse 用のソースコードをここに記述する。
#else
    // 他のターゲット向けのソースコードをここに記述する。
#endif
```

© 2021 ROHM Co., Ltd.

7. 呼び出し規約

コンパイラ(mtcc)が出力する機械語コードでは一定の規約に従って C 言語の関数に引数を渡したり、C 言語の関数呼び出しから値を受け取ったりしています。この規約の事を呼び出し規約と言います。

7.1. レジスタの種類

Table 62. レジスタの種類

レジスタ名	レジスタ種類	説明
R0	特殊レジスタ	16-bit の PC(Program Counter)の下位 byte として使用される。 LDR 命令でソースオペランドとして使用するとフラグレジスタの値が取得できる。 算術演算やビット演算で使用するとゼロレジスタとして使用できる。
R1	特殊レジスタ	16-bit の PC(Program Counter)の上位 byte として使用される。
R2, R3	特殊レジスタ	16-bit の SP(Stack Pointer)として使用される。R2 が下位 byte、R3 が上位 byte。
R4, R5	特殊レジスタ Callee-Saved レジスタ	16-bit の FP(Frame Pointer)として使用される。R4 が下位 byte、R5 が上位 byte。 FP を持たない関数内では Callee-Saved 汎用レジスタとしても使用される。
R6, R7, R8, R9	Callee-Saved レジスタ	関数呼び出しによって値が変更されない汎用レジスタ。
R10, R11	テンポラリレジスタ Caller-Saved レジスタ	様々な一時的な処理のために使用されるレジスタ。 関数呼び出しによって値が変更される可能性がある。
R12, R13, R14, R15	Caller-Saved レジスタ	関数呼び出しによって値が変更される可能性がある汎用レジスタ。

7.2. 引数割り当ての呼び出し規約

引数は順番にレジスタに割り当てられていきます。レジスタが足りなくなった場合は残りの引数はスタックに割り当てられます。

7.2.1. 引数のレジスタへの割り当て規約

- 引数は宣言された順番(一番左から順番)に R15,R14,R13,R12 に割り当てる。
- 1byte の引数は 2byte に拡張してレジスタに割り当てる。

7.2.2. 引数のスタックへの割り当て規約

- 引数は一番右から順番にスタックに push する。

7.3. 戻り値の呼び出し規約

- 戻り値が構造体か共用体の場合は最後のメンバから順番に上位バイトからスタックにプッシュする。
- 戻り値が 1byte の場合はレジスタ R14 に割り当てる。
- 戻り値が 2byte の場合は下位バイトをレジスタ R14 に、上位バイトをレジスタ R15 に割り当てる。
- 戻り値が 4byte の場合は下位バイトから順番にレジスタ R12,R13,R14,R15 に割り当てる。

© 2021 ROHM Co., Ltd.

8. 無保証機能

mtcc と mtld はオープンソースソフトウェアをベースに開発しています。それらのオープンソースソフトウェアが持っている機能は mtcc と mtld にも流用されています。つまり、本ドキュメントで説明されていない機能が mtcc と mtld では利用可能です。これらの機能の事を「無保証機能」と言います。
無保証機能は動作確認しておらず正常に動作するかは保証しません。利用者の自己判断に基づいて使用してください。

8.1. コンパイラコマンドラインオプション

mtcc のベースになっているオープンソースソフトウェアのコンパイラで利用可能なコマンドラインオプション一覧は以下の添付ファイルに記載しています。この添付ファイルに記載されている機能の中で、本ドキュメントで説明されていないものは全て無保証機能となり、正常に動作するかは保証しません。利用者の自己判断に基づいて使用してください。


Adobe Acrobat Document

8.2. Attribute

mtcc のベースになっているオープンソースソフトウェアのコンパイラで利用可能な attribute の一覧は以下の添付ファイルに記載しています。この添付ファイルに記載されている機能の中で、本ドキュメントで説明されていないものは全て無保証機能となり、正常に動作するかは保証しません。利用者の自己判断に基づいて使用してください。


Adobe Acrobat Document

8.3. リンカコマンドラインオプション

mtld のベースになっているオープンソースソフトウェアのリンカで利用可能なコマンドラインオプションの一覧は以下の添付ファイルに記載しています。この添付ファイルに記載されている機能の中で、本ドキュメントで説明されていないものは全て無保証機能となり、正常に動作するかは保証しません。利用者の自己判断に基づいて使用してください。


Adobe Acrobat Document

以上

© 2021 ROHM Co., Ltd.

オープンソースソフトウェアについて

本ソフトウェアは、ロームが権利を保有もしくは権利許諾を受けているソフトウェアに加えて、下記のライセンス条件の下で提供されているオープンソースソフトウェア（以下「オープンソースソフトウェアプログラム」といいます）を含んでいます。
オープンソースソフトウェアプログラムに関しては、 それぞれの使用許諾条件が適用されますので、万一、オープンソースソフトウェアプログラムのライセンス条件と本資料の間で矛盾が生じた場合は、オープンソースソフトウェアプログラムのライセンス条件が優先されます。

含まれているソフトウェアとそのライセンス条件
・LLVM/clang(3-clause BSD license)
・Mono.Options(The MIT License)
・ninja(Apache License Version 2.0)
・libc(Apache License Version 2.0)
・binutils(GPL Version 2)
・crt(GPL Version 3)

商標について

「Windows」は、マイクロソフト グループの企業の商標です。
「Intel」は、Intel Corporation またはその子会社の商標です。
「Core™」は、Intel Corporation またはその子会社の商標または登録商標です。
「tinyMicon MatisseCORE™」は、ローム株式会社の商標または登録商標です。

© 2021 ROHM Co., Ltd.

ご注意

1）本資料に記載されているソフトウェア及びシステム（以下あわせて「本ソフトウェア」といいます）に関する情報、及び本資料の記載内容は、本資料発行時点のものであり、予告なく変更することがあります。

2）本製品又は本ソフトウェアを、極めて高度な信頼性が要求され、その故障や誤動作が人の生命、身体への危険若しくは損害、又はその他の重大な損害の発生に関わるような機器又は装置（医療機器、輸送機器、交通機器、航空宇宙機器、原子力制御装置、燃料制御、カーアクセサリを含む車載機器、各種安全装置等）（以下「特定用途」といいます）へのご使用を検討される際は事前にローム営業窓口までご相談くださいますようお願い致します。ロームの文書による事前の承諾を得ることなく、特定用途に使用したことによりお客様又は第三者に生じた損害等に関し、ロームは一切その責任を負いません。

3）ロームは常に品質・信頼性の向上に取り組んでいますが、半導体製品は種々の要因で故障・誤作動する可能性があります。万一、本ソフトウェアと使用するローム製品（以下「本製品」といいます）が故障・誤作動した場合であっても、その影響により人身事故、火災損害等が起こらないようご使用機器でのディレーティング、冗長設計、延焼防止、バックアップ、フェイルセーフ等の安全確保をお願いします。定格を超えたご使用や使用上の注意事項が守られていない場合、ロームは一切その責任を負いません。

4）ロームは、本ソフトウェアの使用または使用不能によってお客様に生じるいかなる直接的、間接的損害(情報、データ、プログラム等の無体財産の損失、利益の喪失、中断などによる損害を含む)に関して、一切の責任を負いません。

5）本ソフトウェア、本資料及び本製品には、ロームに帰属する機密情報、技術情報又はトレードシークレットが記載されておりますので、ロームの文書による事前の承諾を得ることなく、その全部又は一部について以下の行為をすることは固くお断り致します。
(ア) 第三者へ開示すること
(イ) 逆アセンブル、リバースエンジニアリングその他の解析をすること
(ウ) 転載、複写、複製すること
(エ) 本ソフトウェアに含まれている著作権表示を取り除くこと

6）本製品、本ソフトウェア及び本資料に記載の技術を輸出または国外へ提供する際には、「外国為替及び外国貿易法」、「米国輸出管理規則」など適用される輸出関連法令を遵守し、それらの定めにしたがって必要な手続きを行ってください。

7）ロームは、本ソフトウェア及び本資料に記載された情報に関し第三者の知的財産権及びその他の権利について権利侵害がないことを保証するものではありません。従いまして、上記権利侵害に関し、ロームは一切その責任を負いません。

8）ロームは、本資料に記載された情報についてローム若しくは第三者が所有又は管理している知的財産権その他の権利の実施又は利用を、本資料によって、明示的にも黙示的にも、お客様に許諾するものではありません。

9）お客様は、（１）お客様による第三者の権利の侵害（オープンソースソフトウェアプログラムのライセンス条件への違反を含むがこれに限定されない）や法令違反により生じる、または、（２）お客様が本資料に違反することやこれに関連して生じる賠償金等の請求による損失、罰金や費用（弁護士費用を含むがこれに限定されない）から、ローム及びロームの役員や従業員を免責することを保証し、損害を与えないようにし、また保護するものとします。

１０）本資料は、正確を期して慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載された内容の誤りに起因してお客様又は第三者に損害が生じた場合であっても、ロームは一切その責任を負いません。

ローム製品のご利用ありがとうございます。
製品に関するお問い合わせは、下記より願い致します。

ROHM Customer Support System

http://www.rohm.co.jp/contact/

© 2024 ROHM Co., Ltd. All rights reserved.