tinyMicon MatisseCORE™

mtcc User's Guide

C compiler for tinyMicon MatisseCORE™ mtcc User's Guide

No. 67UG025E Rev.007
December 2023

Update History

Date	Version	Contents
2021/10/28	Rev.001	Describes the contents of mtcc V1.01.00
2022/03/31	Rev.002	Added description of 6.8pragma, 6.9attribute and 6.10predefined macros.
2022/10/21	Rev.003	Describes the contents of 8Unsupported features
2023/5/24	Rev.004	Added description of 3.3　Precautions.
2024/01/15	Rev.005	Modified 3.1Implementation-defined behavior Added 3.2Translation limits
2024/04/04	Rev.006	Added 4.4How to make diagnostic functions strictly compliant with C99 standard Modified 5.3Library Function Details Added 5.4Type definition details Modified 5.5Macro definition details
2024/05/08	Rev.007	Modified 5.4.4stddef.h

Table of Contents

1. Overview

mtcc is a C language compiler for tinyMicon MatisseCORE™.
A C language compiler is a program that converts a program source file written in C language into a machine language file.
tinyMicon MatisseCORE™ is an ultra-compact 8-bit microcontroller developed by ROHM specifically for LSI embedded applications.

1.1. Features

mtcc has the following features.

1 Compliant with the C language standard (C99).
2 Excellent optimization performance.
3 Various command line options.

2. Operating environment

The operating environment of mtcc is described below.

2.1. System Requirements

Table 1. System Requirements

OS	Windows 7 (32-bit/64-bit) Windows 10 (32-bit/64-bit)
CPU	Intel Core series or equivalent performance CPU.
memory	4GByte or more installed
HDD/SSD	At least 200 MByte of free space.

2.2. Installation

Run the mtcc installer (MatisseCCompiler-XX.XX.XX.XX.exe) to install a set of necessary tools.

In the default configuration, the tool set is installed under "C:\Program Files\ROHM\Matisse\".

3. Language specification

The language specifications supported by mtcc are as follows.

Language standard: C99 (ISO/IEC 9899:1999)

Table 2. Data sizes and data alignments of available basic data types

type name	Data size	Data alignment
char	1 byte	1 byte
short	2 byte	1 byte
int	2 byte	1 byte
long	4 byte	1 byte
pointer type	2 byte	1 byte

Table 3. Unavailable basic data types

type name
float
double
long double
long long

Table 4. Unavailable features

feature
dynamic memory allocation
variable length array

Data types larger than 2 bytes will be stored in memory in little-endian format.

Floating-point types and 64-bit integer types are not supported by mtcc because they are rarely used in embedded programs.

Only a freestanding environment is supported for the execution environment.

3.1. Implementation-defined behavior

The C language standard (ISO/IEC 9899:1999 Annex J.3, J.4) requires documenting how the compiler implementation handles the implementation-defined behavior.

The following section shows the implementation-defined behavior of mtcc.

3.1.1. J.3.1 Translation

Table 5. Requirements and implementation-defined behavior of "Translation"

Requirements	Implementation-defined behavior
How a diagnostic is identified (3.10, 5.1.1.3).	[file name]:[line number]:[column number]: [warning/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).	White-space characters except new-line characters are retained.

3.1.2. J.3.2 Environment

Table 6. Requirements and implementation-defined behavior of "Environment"

Requirements	Implementation-defined behavior
The mapping between physical source file multibyte characters and the source character set in translation phase 1 (5.1.1.2).	Multibyte characters are not allowed in identifiers, string literals, or character constants. There is no mapping from multibyte characters to the source character set. Multibyte characters can only be used in comments. Multibyte characters used outside of comments will result in compile errors, for example, the following error message will be displayed. `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).	The only function called at program startup is 'int main(void)'.
	
	Source code that defines a main function whose return value is not of type int will result in compile errors and the following error message will be displayed.
	`error: 'main' must return 'int'`
	
	Source code that defines a main function with more than one argument will result in compile errors and the following error message will be displayed.
	`error: supported entry point is 'int main(void)' only.`

The effect of program termination in a freestanding environment (5.1.2.1).	The CPU will enter the halt state when the main function exits, the abort function is called, or the exit function is called.
An alternative manner in which the main function may be defined (5.1.2.2.1).	There is no other way to define the main function. If the main function is not defined in the program, the source code will result in compile errors, for example, the following error message will be displayed. ``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).	The argc and the argv arguments to main are not supported. The supported main function type is 'int main(void)' only. For example, a source code that defines 'int main(int argc, char **argv)' will result in compile errors and the following error message will be displayed. `error: supported entry point is 'int main(void)' only.`
What constitutes an interactive device (5.1.2.3).	Not specified. The program may use hardware peripherals to interact with external environment.
The set of signals, their semantics, and their default handling (7.14).	Signals are not supported. The function implementation or header file for signal functions do not exist in the standard library. For example, source code that includes 'signal.h' to use signal functions will result in compile errors and the following error message will be displayed. `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).	Signals are not supported. The function implementation or header file for signal functions do not exist in the standard library. For example, source code that includes 'signal.h' to use signal functions will result in compile errors and the following error message will be displayed. `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).	Signals are not supported. The function implementation or header file for signal functions do not exist in the standard library. For example, source code that includes 'signal.h' to use signal functions will result in compile errors and the following error message will be displayed. `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).	The getenv function is not supported. The implementation of the getenv function do not exist in the standard library and the declaration of the getenv function also do not exist in the header file 'stdlib.h'.

	For example, source code using the getenv function will result in compile errors and the following error message will be displayed. `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).	The system function is not supported. The implementation of the system function do not exist in the standard library and the declaration of the system function also do not exist in the header file 'stdlib.h'. For example, source code using the system function will result in compile errors and the following error message will be displayed. `error: implicit declaration of function 'system' is invalid in C99 [-Werror,-Wimplicit-function-declaration]`

3.1.3. J.3.3 Identifiers

Table 7. Requirements and implementation-defined behavior of "Identifiers"

Requirements	Implementation-defined behavior
Which additional multibyte characters may appear in identifiers and their correspondence to universal character names (6.4.2).	Multibyte characters cannot be used as identifiers. For example, source code using multibyte characters as identifiers will result in compile errors and the following error message will be displayed. `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).	Initial 63 characters are significant.

3.1.4. J.3.4 Characters

Table 8. Requirements and implementation-defined behavior of "Characters"

Requirements	Implementation-defined behavior
The number of bits in a byte (3.6).	The number of bits in a byte is 8.
The values of the members of the execution character set (5.2.1).	The values of the members of the execution character set are US ASCII characters.
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
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).	The value is type-converted to the char type.
Which of signed char or unsigned char has the same range, representation, and behavior as "plain" char (6.2.5, 6.3.1.1).	Plain char is the same as signed char.
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).	The mapping of the source character set to the execution character set is one-to-one.
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).	ASCII code values are used except for escape sequences. For escape sequences, the following values are used. '\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).	Wide characters are not supported. The function implementation or header file for wide character functions do not exist in the standard library. For example, source code that includes 'wchar.h' to use wide character functions will result in compile errors and the following error message will be displayed. `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).	Locales are not supported. The function implementation or header file for locale functions do not exist in the standard library. For example, source code that includes 'locale.h' to use locale functions will result in compile errors and the following error message will be displayed. `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).	Locales are not supported.

	The function implementation or header file for locale functions do not exist in the standard library. For example, source code that includes 'locale.h' to use locale functions will result in compile errors and the following error message will be displayed. `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).	Multibyte characters are not supported. The function implementation or header file for multibyte character functions do not exist in the standard library. For example, source code using the mblen function to count the length of multibyte character string will result in compile errors and the following error message will be displayed. `error: implicit declaration of function 'mblen' is invalid in C99 [-Werror,-Wimplicit-function-declaration]` Escape sequence will be stored as a normal binary data in a string literal.

3.1.5. J.3.5 Integers

Table 9. Requirements and implementation-defined behavior of “Integers”

Requirements	Implementation-defined behavior
Any extended integer types that exist in the implementation (6.2.5).	There are no extended integer types.
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).	Signed integer types are represented by two's complement. There is no trap representation.
The rank of any extended integer type relative to another extended integer type with the same precision (6.3.1.1).	There are no extended integer types.
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).	The value is truncated, discarding bits that cannot be stored in the output type, and the least significant bit is left unchanged. Signals are not supported.
The results of some bitwise operations on signed integers (6.5).	For shift operations, an arithmetic shift is performed. For other operations, the value is calculated as an unsigned value.

3.1.6. J.3.6 Floating point

Table 10. Requirements and implementation-defined behavior of “Floating point”

Requirements	Implementation-defined behavior
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).	Floating-points are not supported. The header files 'math.h' and ‘complex.h’ do not exist in the standard library. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `error: floating point type is not supported.`
The rounding behaviors characterized by non-standard values of FLT_ROUNDS (5.2.4.2.2).	Floating-points are not supported. The header file 'float.h’ and ‘FLT_ROUNDS’ macro do not exist in the standard library. For example, source code that includes 'float.h' to use ‘FLT_ROUNDS’ macro will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. The header file 'float.h’ and ‘FLT_EVAL_METHOD’ macro do not exist in the standard library. For example, source code that includes 'float.h' to use ‘FLT_EVAL_METHOD’ macro will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `error: floating point type is not supported.`
Whether and how floating expressions are contracted when not disallowed by the FP_CONTRACT pragma (6.5).	Floating-points are not supported. The header file 'math.h' and 'FP_CONTRACT' pragma do not exist in the standard library. For example, source code that includes 'math.h' to use 'FP_CONTRACT' pragma will result in compile errors and the following error message will be displayed. `fatal error: 'math.h' file not found`
The default state for the FENV_ACCESS pragma (7.6.1).	Floating-points are not supported. The header file 'fenv.h' and 'FENV_ACCESS' pragma do not exist in the standard library. For example, source code that includes 'fenv.h' to use 'FENV_ACCESS' pragma will result in compile errors and the following error message will be displayed. `fatal error: 'fenv.h' file not found`
Additional floating-point exceptions, rounding modes, environments, and classifications, and their macro names (7.6, 7.12).	Floating-points are not supported. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `error: floating point type is not supported.`
The default state for the FP_CONTRACT pragma (7.12.2).	Floating-points are not supported. The header file 'math.h' and 'FP_CONTRACT' pragma do not exist in the standard library. For example, source code that includes 'math.h' to use 'FP_CONTRACT' pragma will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. For example, source code using floating-point variables (float, double, long double) will result in compile errors and the following error message will be displayed. `error: floating point type is not supported.`

3.1.7. J.3.7 Arrays and pointers

Table 11. Requirements and implementation-defined behavior of "Arrays and pointers"

Requirements	Implementation-defined behavior
The result of converting a pointer to an integer or vice versa (6.3.2.3).	The result of converting a pointer to an integer is the result of converting a value of type unsigned int to the destination type. The result of converting an integer to a pointer is the result of converting the source type to an unsigned int.
The size of the result of subtracting two pointers to elements of the same array (6.5.6).	The result of the operation will be of type ptrdiff_t, which is defined in <stddef.h> and the size is 2 bytes.

3.1.8. J.3.8 Hints

Table 12. Requirements and implementation-defined behavior of “Hints”

Requirements	Implementation-defined behavior
The extent to which suggestions made by using the register storage-class specifier are effective (6.7.1).	The register storage-class specifier is ignored
The extent to which suggestions made by using the inline function specifier are effective (6.7.4).	The inline keyword is valid only when the optimization level is 1 or higher and the optimizer determines that there is a benefit to inlining.

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

Table 13. Requirements and implementation-defined behavior of “Structures, unions, enumerations, and bit-fields”

Requirements	Implementation-defined behavior
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).	A "plain" int bit-field is treated as a signed int bit-field.
Allowable bit-field types other than _Bool, signed int, and unsigned int (6.7.2.1).	All integer types are allowed.
Whether a bit-field can straddle a storage-unit boundary (6.7.2.1).	A bit-field cannot straddle a storage-unit boundary.
The order of allocation of bit-fields within a unit (6.7.2.1).	Bit-fields are allocated in order from lowest-bit to highest-bit.
The alignment of non-bit-field members of structures (6.7.2.1).	The alignment of all data types is 1 byte. See 3Language specification.
This should present no problem unless binary data written by one implementation is read by another. The integer type compatible with each enumerated type (6.7.2.2).	Depending on the enumeration value, the most appropriate one of the following types will be selected. ● signed int ● unsigned int ● signed long ● unsigned long

3.1.10. J.3.10 Qualifiers

Table 14. Requirements and implementation-defined behavior of “Qualifiers”

Requirements	Implementation-defined behavior
What constitutes an access to an object that has volatile-qualified type (6.7.3).	A volatile-qualified object is recognized by the compiler as potentially being accessed by asynchronous interrupts, so they are not optimized and are always stored in memory (not in registers). Read and write access to these objects are done directly from and to this memory. Each read and write operation processes data one byte at a time. If an object is larger than two bytes, the sequence of the bytes accessed is undefined. For objects that include bit fields, any modification is handled through a read-modify-write process.

3.1.11. J.3.11 Preprocessing directives

Table 15. Requirements and implementation-defined behavior of “Preprocessing directives”

Requirements	Implementation-defined behavior
How sequences in both forms of header names are mapped to headers or external source file names (6.4.7).	The sequences in both forms of header names will be interpreted as US ASCII code and will be mapped to the source file names.
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).	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.
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).	The value of a single-character character constant in a constant expression that controls conditional inclusion may have a negative value.
The places that are searched for an included < > delimited header, and how the places are specified, or the header is identified (6.10.2).	The compiler will search for the specified files in the following order. 1. From the directories specified by -I option.

	2. From the system header file directory. "C:\Program Files\ROHM\Matisse\include" in default.
How the named source file is searched for in an included " " delimited header (6.10.2).	The compiler will search for the specified files in the following order. 1. From the current directory, that is, the directory containing the source file to be compiled. 2. From the directories specified by -I option. 3. From the system header file directory. "C:\Program Files\ROHM\Matisse\include" in default.
The method by which preprocessing tokens (possibly resulting from macro expansion) in a #include directive are combined into a header name (6.10.2).	The preprocessing tokens in a #include directive are combined by the same method as other preprocessing tokens.
The nesting limit for #include processing (6.10.2).	Up to 15 nests are supported.
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).	The # operator will not insert a \ character before the \ character that begins a universal character name in a character constant or string literal.
The behavior on each recognized non-STDC #pragma directive (6.10.6).	See section 6.8pragma.
The definitions for __DATE__ and __TIME__ when respectively, the date and time of translation are not available (6.10.8).	These macros are always available.

3.1.12. J.3.12 Library functions

Table 16. Requirements and implementation-defined behavior of “Library functions”

Requirements	Implementation-defined behavior
Any library facilities available to a freestanding program, other than the minimal set required by clause 4 (5.1.2.1).	All library functions described in section 5Library function are available to a freestanding program.
The format of the diagnostic printed by the assert macro (7.2.1.1).	Assertion failed: ([assertion argument]), function [functio n name], file [file name], line [line number].
The representation of the floating-point status flags stored by the fegetexceptflag function (7.6.2.2).	Floating-points are not supported. The header file 'fenv.h’ and fegetexceptflag function do not exist in the standard library. For example, source code that includes 'fenv.h' to use fegetexceptflag function will result in compile errors and the following error message will be displayed. `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).	Floating-points are not supported. The header file 'fenv.h’ and feraiseexcept function do not exist in the standard library. For example, source code that includes 'fenv.h' to use feraiseexcept function will result in compile errors and the following error message will be displayed. `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).	The setlocale function is not supported. The function implementation or header file for locale functions do not exist in the standard library. For example, source code that includes 'locale.h' to use setlocale functions will result in compile errors and the following error message will be displayed. `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).	Mathematics functions and floating-points are note supported.
	The header files 'math.h' and ‘float.h’ do not exist in the standard library. And the definition of float_t and double_t and FLT_EVAL_METHOD also do not exist in the standard library.
	For example, source code that includes 'math.h' to use float_t and double_t will result in compile errors and the following error message will be displayed.
	`fatal error: 'math.h' file not found`
	
	Also, for example, source code that includes 'float.h' to use FLT_EVAL_METHOD macro will result in compile errors and the following error message will be displayed.
	`fatal error: 'float.h' file not found`

Domain errors for the mathematics functions, other than those required by this International Standard (7.12.1).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use mathematics functions will result in compile errors and the following error message will be displayed. `fatal error: 'math.h' file not found`
The values returned by the mathematics functions on domain errors (7.12.1).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use mathematics functions will result in compile errors and the following error message will be displayed. `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).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use macro math_errhandling or macro MATH_ERRNO or macro MATH_ERREXCEPT will result in compile errors and the following error message will be displayed. `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).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use fmod function will result in compile errors and the following error message will be displayed. `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).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use remquo functions will result in compile errors and the following error message will be displayed. `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).	Signals are not supported. The function implementation or header file for signal functions do not exist in the standard library. For example, source code that includes 'signal.h' to use signal functions will result in compile errors and the following error message will be displayed. `fatal error: 'signal.h' file not found`
The null pointer constant to which the macro NULL expands (7.17).	The macro NULL will be expanded to ‘((void *)0)’.
Whether the last line of a text stream requires a terminating new-line character (7.19.2).	The last line of a text stream doesn't require a terminating new-line character.
Whether space characters that are written out to a text stream immediately before a new-line character appear when read in (7.19.2).	Input stream is not supported. The implementation of functions for the input stream do not exist in the standard library, and the declaration of variables or functions for the input stream also do not exist in the header file 'stdio.h'. For example, source code using the getc function will result in compile errors and the following error message will be displayed. `error: implicit declaration of function 'getc' is invalid 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).	The number of null characters that may be appended to data written to a binary stream is 0.
Whether the file position indicator of an append-mode stream is initially positioned at the beginning or end of the file (7.19.3).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	File buffering is not performed.
Whether a zero-length file actually exists (7.19.3).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'.

	For example, source code using the remove function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the rename function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the tmpfile function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fopen function will result in compile errors and the following error message will be displayed. `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).	Infinity and NaN are not supported. The function implementation or header file for an infinity or NaN do not exist in the standard library. For example, source code that includes 'math.h' to use ‘INFINITY’ macro of ‘NAN’ macro will result in compile errors and the following error message will be displayed. `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).	A 4-digit hexadecimal number will be output.
The interpretation of a - character that is neither the first nor the last character, nor the second where a ˆ character is the first, in the scanlist for %[conversion in the fscanf or fwscanf function (7.19.6.2, 7.24.2.1).	Input stream is not supported. The implementation of functions for the input stream do not exist in the standard library, and the declaration of variables or functions for the input stream also do not exist in the header file 'stdio.h'. For example, source code using the fscanf function will result in compile errors and the following error message will be displayed. `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).	Input stream is not supported. The implementation of functions for the input stream do not exist in the standard library, and the declaration of variables or functions for the input stream also do not exist in the header file 'stdio.h'. For example, source code using the fscanf function will result in compile errors and the following error message will be displayed. `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).	File operation is not supported. The implementation of functions for the file operation do not exist in the standard library, and the declaration of variables or functions for the file operation also do not exist in the header file 'stdio.h'. For example, source code using the fgetpos function will result in compile errors and the following error message will be displayed. 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 (7.20.1.3, 7.24.4.1.1).	Those functions are not supported. The implementation of those functions do not exist in the standard library, and the declaration of those functions also do not exist in the header file 'stdlib.h'. For example, source code using the strtod function will result in compile errors and the following error message will be displayed. 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).	Those functions are not supported. The implementation of those functions do not exist in the standard library, and the declaration of those functions also do not exist in the header file 'stdlib.h'. For example, source code using the strtod function will result in compile errors and the following error message will be displayed. 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).	Those functions are not supported. The implementation of those functions do not exist in the standard library, and the declaration of those functions also do not exist in the header file 'stdlib.h'. For example, source code using the malloc function will result in compile errors and the following error message will be displayed. 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).	Stream buffer and temporary files are not supported. The implementation of the functions of temporary files do not exist in the standard library, and the declaration of the functions of temporary files also do not exist in the header file ‘stdio.h’. For example, source code using the tmpfile function will result in compile errors and the following error message will be displayed. error: implicit declaration of function 'tmpfile' is invalid in C99 [-Werror,-Wimplicit-function-declaration] Opened streams are not closed when the abort or _Exit function are called.
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).	In the case of the abort function, the termination status will be 1. In the case of the exit function, the argument of the function will be the termination status. The termination code is stored in the register R14 and R15. See section 7.3Function return value calling conventions.
The value returned by the system function when its argument is not a null pointer (7.20.4.6).	The system function is not supported. The implementation of the system function does not exist in the standard library, and the declaration of the system function also does not exist in the header file ‘stdlib.h’ For example, source code using the system function will result in compile errors and the following error message will be displayed.

	`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).	The date and time functions are not supported. The function implementation or header file for date and time functions do not exist in the standard library. For example, source code that includes 'time.h' to use time functions will result in compile errors and the following error message will be displayed. `fatal error: 'time.h' file not found`
The range and precision of times representable in clock_t and time_t (7.23).	The date and time functions are not supported. The function implementation or header file for date and time functions do not exist in the standard library. For example, source code that includes 'time.h' to use clock_t type or time_t type will result in compile errors and the following error message will be displayed. `fatal error: 'time.h' file not found`
The era for the clock function (7.23.2.1).	The date and time functions are not supported. The function implementation or header file for date and time functions do not exist in the standard library. For example, source code that includes 'time.h' to use the clock function will result in compile errors and the following error message will be displayed. `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).	The date and time functions are not supported. The function implementation or header file for date and time functions do not exist in the standard library. For example, source code that includes 'time.h' to use the strftime function will result in compile errors and the following error message will be displayed. `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).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use mathematics functions will result in compile errors and the following error message will be displayed. `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).	Mathematics functions are not supported. The function implementation or header file for mathematics functions do not exist in the standard library. For example, source code that includes 'math.h' to use mathematics functions will result in compile errors and the following error message will be displayed. `fatal error: 'math.h' file not found`

3.1.13. J.3.13 Architecture

Table 17. Requirements and implementation-defined behavior of “Architecture”

Requirements	Implementation-defined behavior
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).	For <stdint.h> and <limits.h>, see section 5.5Macro definition details. <floath.h> is not supported. Source code that includes ‘float.h’ will result in compile errors and the following error message will be displayed. `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).	The alignment of all object is 1 byte. Endianness is little-endian.
The value of the result of the sizeof operator (6.5.3.4).	See section 3Language specification.

3.1.14. J.4 Locale-specific behavior

Table 18. Requirements and implementation-defined behavior of “Locale-specific behavior”

Requirements	Implementation-defined behavior
Additional members of the source and execution character sets beyond the basic character set (5.2.1).	There are no additional members of the source and execution character sets.

The presence, meaning, and representation of additional multibyte characters in the execution character set beyond the basic character set (5.2.1.2).	There are no additional multibyte characters in the execution character set.
The shift states used for the encoding of multibyte characters (5.2.1.2).	Multibyte characters are not supported. The function implementation or header file for multibyte character functions do not exist in the standard library. For example, source code using the mblen function will result in compile errors and the following error message will be displayed. `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).	Always printed from left to right.
The decimal-point character (7.1.1).	The decimal-point character is always ".".
The set of printing characters (7.4, 7.25.2).	Characters from 0x20 to 0x7E in US ASCII code.
The set of control characters (7.4, 7.25.2).	Characters from 0x00 to 0x1F and 0x7F in US ASCII code.
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).	The iswalpha, iswblank, iswlower, iswpunct, iswspace and iswupper functions are not supported. For other functions, tested characters in US ASCII code are as follows. isalpha: From 0x41 to 0x5A and from 0x61 to 0x7A. isblank: 0x09 and 0x20. islower: From 0x61 to 0x7A. ispunct: From 0x21 to 0x2F and from 0x3A to 0x40 and from 0x5B to 0x60 and from 0x7B to 0x7E. isspace: From 0x09 to 0x0D and 0x20. isupper: From 0x41 to 0x5A.
The native environment (7.11.1.1).	Locale is not supported. The function implementation or header file for locale functions do not exist in the standard library. For example, source code that includes 'locale.h' to use locale functions will result in compile errors and the following error message will be displayed. `fatal error: 'locale.h' file not found`
Additional subject sequences accepted by the numeric conversion functions (7.20.1, 7.24.4.1).	No additional subject sequences accepted by the numeric conversion functions.
The collation sequence of the execution character set (7.21.4.3, 7.24.4.4.2).	The collation functions are not supported. The function implementation of the collation functions do not exist in the standard library, and the declaration of the collation functions also do not exist in the header file ‘string.h’. For example, source code using the strcoll function will result in compile errors and the following error message will be displayed. `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).	The strerror function is not supported. The implementation of the strerror function does not exist in the standard library, and the declaration of the strerror function also does not exist in the header file ‘string.h’. For example, source code using the strerror function will result in compile errors and the following error message will be displayed. `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).	Time and date are not supported. The function implementation or header file for date and time functions do not exist in the standard library. For example, source code that includes 'time.h' to use the strftime function will result in compile errors and the following error message will be displayed. `fatal error: 'time.h' file not found`
Character mappings that are supported by the towctrans function (7.25.1).	The towctrans function is not supported. The implementation or header file for the towctrans function do not exist in the standard library.

	For example, source code that includes ‘wctype.h’ to use the towctrans function will result in compile errors and the following error message will be displayed. `fatal error: 'wctype.h' file not found`
Character classifications that are supported by the iswctype function (7.25.1).	The iswctype function is not supported. The implementation or header file for the iswctype function do not exist in the standard library. For example, source code that includes ‘wctype.h’ to use the iswctype function will result in compile errors and the following error message will be displayed. `fatal error: 'wctype.h' file not found`

3.2. Translation limits

For each of the translation limits defined in C99 (ISO/IEC 9899:1999), the values for which mtcc guarantees operation are described in the table below.

Table 19. The translation limits values guaranteed by mtcc

Item	C99 Spec	mtcc guarantees
Nesting levels of blocks	127	127
Nesting levels of conditional inclusion	63	63
Pointer, array, and function declarators (in any combinations) modifying an arithmetic, structure, union, or incomplete type in a declaration	12	12
Nesting levels of parenthesized declarators within a full declarator	63	63
Nesting levels of parenthesized expressions within a full expression	63	63
Significant initial characters in an internal identifier or a macro name	63	63
Significant initial characters in an external identifier	31	31
External identifiers in one translation unit	4095	4095
Identifiers with block scope declared in one block	511	511
Macro identifiers simultaneously defined in one preprocessing translation unit	4095	4095
Parameters in one function definition	127	127
Arguments in one function call	127	127
Parameters in one macro definition	127	127
Arguments in one macro invocation	127	127
Characters in a logical source line	4095	4095
Characters in a character string literal or wide string literal (after concatenation)	4095	4095(*1)
Bytes in an object (in a hosted environment only)	65535	– (*2)
Nesting levels for #included files	15	15
Case labels for a switch statement	1023	1023
Members in a single structure or union	1023	1023
Enumeration constants in a single enumeration	1023	1023
Level of nested structure or union definitions in a single struct-declaration-list	63	63

(*1) The code using the maximum value of this item can be compiled by mtcc, but in some hardware configurations, the string object may exceed the RAM capacity. In other words, it will not cause errors at compile time, but may cause RAM size error at link time. Refer to 4.7.3--data-size= for changing the capacity of RAM.

(*2) This field is out of scope because mtcc supports only the freestanding environment and does not support the hosted environment.

3.3. Precautions

As mentioned above, mtcc complies with C99 (ISO/IEC 9899:1999), but there are some statements that requires caution in that standard.
Therefore, mtcc may behave in a seemingly faulty manner even though it is operating according to the standard.
This section describes the contents and workarounds of these precautions.

3.3.1. Infinite loop

3.3.1.1. Description

mtcc may interpret infinite loops with no side-effects as undefined behavior and remove it. This may cause programs containing infinite loops to terminate or cause unexpected behavior.

Here is an example of an infinite loop without side-effects. Please avoid such code as it would result in undefined behavior.

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

3.3.1.2. Workaround

By using volatile keyword, infinite loop can be safely described as follows.

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

The volatile keyword tells mtcc that this infinite loop may have side-effects. As a result, mtcc will not interpret this infinite loop as undefined behavior and will not remove it.

4. How to use

4.1. Start the compiler (mtcc)

mtcc is executed from the command line as shown below. Basically, multiple input files can be specified. C source files (*.c), assembler files (*.S), and object files (*.o) are supported as input files.

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

A concrete example of the command line to run mtcc is shown below.

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

4.2. List of compiler command line options

The main command line options of the compiler (mtcc) are listed below.

Table 20. The main command line options of mtcc

Command line options	Description
-O	Specifies the optimization level.
-o	Specifies the output file name.
-I	Specifies the directory of the include file. Can be specified multiple times.
-E	Outputs the results of preprocessor execution to standard output.
-S	Outputs an assembler file converted from C language source files.
-c	Outputs an object file converted from a C language source file.
-g	Adds debugging information to the output file. The format of the debug information is DWARF4.
-D	Defines a preprocessor macro. Can be specified multiple times.
-Werror	Turn warnings into errors.
-Werror=	Turn specified warning into an error.
-Wno-	Disable specified warning.
-Weverything	Enable all warnings.
-Wl,	Passes the comma-separated options to the linker (mtld).
-v	Displays the version information.
-h	Displays a help message.

4.3. Details of compiler command line options

4.3.1. -O

4.3.1.1. Description

The -O option specifies the optimization level of the compiler in the following format. If this option is omitted, the default value (-O2) is applied.

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

Depending on the optimization level, the performance of the program, ROM size, and debuggability vary.
The differences between each optimization level are as follows.

Table 21. The difference between each optimization level

Option	Description	program execution speed	ROM size	Debuggability
-O0	No optimization	slow	big	high
-O1	Optimization level 1	general	medium	medium
-O2	Maximum optimization level	fastest	small	very low
-Os	ROM size reduction	fast	smallest	very low

4.3.1.2. Example of use

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

4.3.2. -o

4.3.2.1. Description

The -o option specifies the output file name in the following format. A binary format file (ROM image) is output. At the same time, the ELF format file will be automatically generated (the file name of the ELF format file will be the output file name with its extension changed to .elf).
However, if the -E option is used, the -o option will be disabled, and if the -S option is used, Only the assembler file will be output.

```
-o filename
```

4.3.2.2. Example of use

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

4.3.3. -I

4.3.3.1. Description

The -I option specifies a directory in the following format and adds the specified directory to the include file search path. Multiple -I options can be specified.

```
-Idirname
```

4.3.3.2. Example of use

```
> 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. Description

The -E option prints the results of the preprocessor-only execution to standard output. When the -E option is used, the -o option is disabled. If you want to save the results of the -E option to a file, use the redirection instead of the -o option.

4.3.4.2. Example of use

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

4.3.5. -S

4.3.5.1. Description

The -S option outputs an assembler file. When -S option is used, the number of input files is limited to one. The extension of the output assembler file is recommended to be (*.S) by convention.

4.3.5.2. Example of use

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

4.3.6. -c

4.3.6.1. Description

The -c option outputs an object file. When the -c option is used, the number of input files is limited to one. The extension of the output object file is recommended to be (*.o) by convention.

4.3.6.2. Example of use

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

4.3.7. -g

4.3.7.1. Description

The -g option generates debugging information in the output file. The format of the debugging information is DWARF version 4.

4.3.7.2. Example of use

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

4.3.8. -D

4.3.8.1. Description

The -D option defines a preprocessor macro in the following format. Multiple -D options can be specified.

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

4.3.8.2. Example of use

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

4.3.9. -Werror

4.3.9.1. Description

The -Werror option treats all warnings as errors.

4.3.9.2. Example of use

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

4.3.10. -Werror=

4.3.10.1. Description

The -Werror= option treats the specified warning as an error as follows. For example, -Werror=constant-conversion will treat the warning for an assignment that results in an overflow as an error.

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

4.3.10.2. Example of use

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

4.3.11. -Wno

4.3.11.1. Description

The -Wno- option suppresses the specified warning as follows. For example, -Wno-constant-conversion will suppress warnings for assignments that result in overflow.

```
-Wno-constant-conversion
```

4.3.11.2. Example of use

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

4.3.12. -Weverything

4.3.12.1. Description

The -Weverything option allows all warnings to be generated. By default, mtcc behaves the same as the clang’s -Wall option, suppressing minor warnings. But by using this option, all warnings will be enabled.

4.3.12.2. Example of use

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

4.3.13. “-Wl,”

4.3.13.1. Description

The "-Wl," option specifies command line options to be passed to the linker (mtld) in the following format. This option must be enclosed entirely in double quotes.

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

4.3.13.2. Example of use

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

4.3.14. -v

4.3.14.1. Description

The -v option outputs the version information of the compiler (mtcc) to the standard output.

4.3.14.2. Example of use

```
> mtcc -v
```

4.3.15. -h

4.3.15.1. Description

The -h option prints help messages for command line options to standard output.

4.3.15.2. Example of use

```
> mtcc -h
```

4.4. How to make diagnostic functions strictly compliant with C99 standard

If you want mtcc's source code diagnostics to strictly compliant with the C99 standard, add all of the following command line options at compile time.

```
-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. Descriptions of command line options

The ‘-pedantic-errors’ option ensures that mtcc’s diagnostic functions comply with the C99 standard. However, using the ‘-pedantic-errors’ option alone does not guarantee full compliance with the standard. There are cases where code allowed by the standard is incorrectly marked as errors, so the '-Wno-xxx' options are used to prevent these errors. Also, there are cases where non-standard code is only given warnings instead of errors. So the '-Werror=xxx' options are used to turn those warnings into errors.

4.5. Start the linker (mtld)

mtld can be run from the command line as follows. Multiple input files can be specified, object file (*.o) is supported as input files.

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

A concrete example of the command line to run mtld is shown below.

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

You can also specify the command line option of the linker (mtld) by using the compiler (mtcc) command line option “-Wl,” as shown below.

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

4.6. Linker command line options list

The main command line options of the linker (mtld) are listed below.

Table 22. The main command line options of the linker (mtld)

Command line options	Description
--program-size=	Specifies the size of the program area.
--data-size=	Specifies the size of the data area.
--ex-program-size=	Specifies the size of the extended program area.
-L	Specifies the directory of the library file. Can be specified multiple times.
-l	Specify the library file to be linked. Can be specified multiple times.
--defsym	Defines a global symbol for the linker. Can be specified multiple times.
--script	Used to modify the linker script.
-Map=	Specify the map file output destination.
--exclude-mul=	Specifies whether the target core has a multiplier or not.
--crt	Changes the runtime library to be linked.
--libc	Changes the library file to be linked.
-v	Displays the version information.

-h	Displays a help message.

4.7. Details of linker command line options

4.7.1. -o

4.7.1.1. Description

The -o option specifies the output file name in the following format. A binary format file (ROM image) is output. At the same time, the ELF format file will be automatically generated (the file name of the ELF format file will be the output file name with its extension changed to .elf).

```
-o filename
```

4.7.1.2. Example of use

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

4.7.2. --program-size=

4.7.2.1. Description

The --program-size= option specifies the size of the program area in the following format. The program area will be the size of the specified number multiplied by 4kbyte. If this option is omitted, the default value of 8 (32kbyte) will be applied.

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

If the sum of the value specified by --program-size= and the value specified by --data-size= exceeds 17 (68kbyte), a memory capacity error will occur.

4.7.2.2. Example of use

```
> 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. Description

The --data-size= option specifies the size of the data area in the following format. The data area will be the size of the specified number multiplied by 4kbyte. If this option is omitted, the default value of 1 (4kbyte) will be applied.

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

If the sum of the value specified by --program-size= and the value specified by --data-size= exceeds 17 (68kbyte), a memory capacity error will occur.

4.7.3.2. Example of use

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

4.7.4. --ex-program-size=

4.7.4.1. Description

The --ex-program-size= option specifies the size of the extended program area in the following format. The size of the extended program area is the specified number multiplied by 2kbyte. If this option is omitted, the default value of 0 (0kbyte) will be applied.

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

4.7.4.2. Example of use

```
> 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. Description

The -L option specifies a directory in the following format and adds the specified directory to the library file search path. This option can be specified multiple times.

```
-Ldirname
```

4.7.5.2. Example of use

```
> 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. Description

The -l option specifies the library file to be linked in the following format. The specified library files will be searched in the directory registered in the library search path with the -L option. This option can be specified multiple times.

```
-llibname
```

The library file specified by this option must meet all of the following conditions

- The file is a static library.
- The file name starts with "lib".
- The file extension is ".a".

The -l option specifies the library file name as a string without "lib" and ".a".

4.7.6.2. Example of use

```
> mtcc "-Wl,-ltest" input.c -o output.bin       // “libtest.a” will be linked
> mtld -ltest input.o -o output.bin              // “libtest.a” will be linked
```

4.7.7. --defsym

4.7.7.1. Description

The --defsym option defines a symbol in the following format. This option can be specified multiple times.

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

4.7.7.2. Example of use

```
> 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. Description

The --script option specifies the linker script to be used for link processing in the following format. If this option is omitted, the default linker script will be used.
The default linker script is "C:\Program Files\ROHM\Matisse\lib\ldscripts\matisse.x".

```
--script filename
```

4.7.8.2. Example of use

```
> 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. Description

The -Map= option specifies the output destination of the map file in the following format. The map file will show the addresses of global variables and functions.

```
-Map=filename
```

4.7.9.2. Example of use

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

4.7.10. --exclude-mul=

4.7.10.1. Description

The --exclude-mul option specifies whether the CPU core to be targeted has a multiplier in the following format. If this option is omitted, the default value of (true) will be applied.

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

There are several variants of tinyMicon MatisseCORE™, some with multipliers and some without. This option can be used to generate the appropriate machine language file for CPU core to be targeted.

Table 23. The values and the output results of “--exclude-mul=” option

Value	Output results
true	Generate machine language files for CPU cores that do not have multipliers.
false	Generate a machine language file for CPU cores with a multiplier.

4.7.10.2. Example of use

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

4.7.10.3. detailed information

This option determines the type of compiler runtime library (CRT) to link. There are two types of CRTs: one that uses mul instructions for CPU cores with multipliers, and one that does not use mul instructions for CPU cores without multipliers.
When true is applied to the --exclude-mul option, the CRT for CPU cores without multiplier is linked.
When false is applied to the --exclude-mul option, CRT for CPU cores with multiplier is linked.
The multiplication process is implemented as CRT functions __mulhi3 and __mulsi3, and these CRT functions are called when the multiplication process is needed in the user application.
In other words, even when false is applied to the --exclude-mul option, the mul instruction will not be generated in the user application. The generated code of the user application will not be changed by setting of this option.

4.7.11. --crt

4.7.11.1. Description

The --crt option replaces the runtime library with a file specified in the following format. If this option is omitted, the runtime library file corresponding to the value set in the --exclude-mul= option will be used.
For the CPU cores without a multiplier, "C:\Program Files\ROHM\Matisse\lib\crt.a" will be linked.
For the CPU cores with a multiplier, "C:\Program Files\ROHM\Matisse\lib\crt_with_mul.a" will be linked.

The runtime library file specified by this option must meet all of the following conditions

- The file is a static library.
- Define all symbols referenced by the linker script.
- Implements all the necessary runtime libraries for the compiler (mtcc)

4.7.11.2. Example of use

```
> 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. Description

The --libc option replaces the library files in the following format. If this option is omitted, the default library file will be linked. The default library file is "C:\Program Files\ROHM\Matisse\lib\libc.a".

The library file specified by this option must meet all of the following conditions

- static library
- Implement all library functions supported by the compiler (mtcc)

4.7.12.2. Example of use

```
> 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. Description

The -v option outputs the version information of the linker (mtld) to the standard output.

4.7.13.2. Example of use

```
> mtld -v
```

4.7.14. -h

4.7.14.1. Description

The -h option prints the help messages of the linker (mtld) to the standard output.

4.7.14.2. Example of use

```
> mtld -h
```

5. Library function

The library functions provided by the compiler (mtcc) are described below.

5.1. Library file

The library file contains all the library functions supported by the compiler (mtcc). The default library file is located at "C:\Program Files\ROHM\Matisse\lib\libc.a".
The linker (mtld) reads the library file and statically links it to the machine language instruction file. The linker (mtld) links only the library functions necessary for program execution, so that unnecessary functions are not linked.
It is also possible to change the library file to be linked using the --libc option of the linker (mtld).

5.2. Header file

To use the library functions, you need to include the header file.
The following is a list of header files for the libraries provided by the compiler (mtcc).

Table 24. The list of library functions provided by the compiler (mtcc)

Header file name	Description
assert.h	A header file for diagnostic functions.
ctype.h	A header file for character data identification and conversion.
errno.h	A header file for the error numbers used by library functions.
inttypes.h	A header file that extends stdint.h and adds functionality.
iso646.h	A header file that provides an alternative to the operator.
limits.h	A header file that defines the limits for each integer type.
setjmp.h	A header file for global jump.
stdarg.h	A header file for functions with variable length arguments.
stdbool.h	A header file that provides the bool type.
stddef.h	A header file that provides commonly used macros and types.
stdint.h	A header file that provides an integer type with a standardized size.
stdio.h	A header file for standard I/O.
stdlib.h	A header file that provides macros, types, and functions related to standard utilities.
string.h	A header file for string operations.
matisse/cpufunc.h	A header file that provides special macros to manipulate the CPU.
matisse/extmem.h	A header file that provides macros about the extended program area.
matisse/interrupt.h	A header file that provides macros for interrupt handling.
matisse/sections.h	A header file that provides macros to specify sections.
matisse/signature.h	A header file for storing signatures in the output file.
matisse/version.h	A header file that provides version information for compilers and libraries.
util/atomic.h	A header file that provides macros and functions related to atomic blocks.

5.3. Library Function Details

The library functions (or function macros) that are available in each header file are described below.
It also describes whether each function is reentrant or not. Reentrant functions can be called simultaneously from multiple processing locations.
Non-reentrant functions can cause unintended results if they are called simultaneously from multiple locations. In such cases, use an Atomic block to prevent simultaneous calls from multiple locations.

5.3.1. assert.h

Table 25. The library functions available in assert.ht

Function/Macro Name	Description	Reentrancy
assert	Added diagnostic functions to the program.	non-reentrant

5.3.2. ctype.h

Table 26. The library functions available in ctype.h

Function/Macro Name	Description	Reentrancy
isalnum	Determine if it is ASCII alphanumeric.	reentrant
isalpha	Determines if the character is ASCII alphabet.	reentrant
isascii	Determine if it is an ASCII code.	reentrant
isblank	Determines if a character is a blank character.	reentrant
iscntrl	Determine if it is a control character.	reentrant
isdigit	Determine if it is a decimal number.	reentrant
isgraph	Determine if a character is a printable character other than a standard blank character.	reentrant
islower	Determines if a character is lower ASCII alphabet.	reentrant
isprint	Determine if a character is printable, including standard whitespace characters.	reentrant
ispunct	Determine if a character is a printable character other than standard whitespace or ASCII alphanumeric characters.	reentrant
isspace	Determines if a character is a standard whitespace character.	reentrant
isupper	Determines if a character is upper ASCII alphabet.	reentrant
isxdigit	Determines if the number is a hexadecimal number.	reentrant
toascii	Convert to ASCII characters.	reentrant
tolower	Converts an upper ASCII alphabet character to a lower ASCII alphabet character.	reentrant
toupper	Converts lower ASCII alphabet character to upper ASCII alphabet character.	reentrant

5.3.3. inttypes.h

Table 27. The library functions available in inttype.h

Function/Macro Name	Description	Reentrancy
imaxabs	Calculates the absolute value of a signed 32-bit integer type.	reentrant
imaxdiv	Performs division between signed 32-bit integer types and returns the quotient and remainder.	reentrant

5.3.4. setjump.h

Table 28. The library functions available in setjmp.h

Function/Macro Name	Description	Reentrancy
longjump	Restore the environment in which the setjump function was called.	reentrant
setjump	Save the environment at the time of the function call so that the longjump function can be used later.	reentrant

5.3.5. stdarg.h

Table 29. The library functions available in stdarg.h

Function/Macro Name	Description	Reentrancy
va_arg	Get the variable length argument.	reentrant
va_copy	Copy the current variable length argument list.	reentrant
va_end	End the acquisition of variable length arguments.	reentrant
va_start	Start getting variable length arguments.	reentrant

5.3.6. stddef.h

Table 30. The library functions available in stddef.h

Function/Macro Name	Description	Reentrancy
offsetof	Get the offset value of the member of the strcut type.	reentrant

5.3.7. stdint.h

Table 31. The library functions available in stdint.h

Function/Macro Name	Description	Reentrancy
__CONCATenate	Concatenate the specified characters.	reentrant
__CONCAT	Concatenate the specified characters.	reentrant
INT8_C	Cast specified value to int8_t.	reentrant
UINT8_C	Cast specified value to uint8_t.	reentrant
INT16_C	Cast specified value to int16_t.	reentrant
UINT16_C	Cast specified value to uint16_t.	reentrant
INT32_C	Cast specified value to int32_t.	reentrant
UINT32_C	Cast specified value to uint32_t.	reentrant
INTMAX_C	Cast specified value to intmax_t.	reentrant
UINTMAX_C	Cast specified value to uintmax_t.	reentrant

5.3.8. stdio.h

Table 32. The library functions available in stdio.h

Function/Macro Name	Description	Reentrancy
FDEV_SETUP_STREAM	Stream open.	non-reentrant
fdev_close	Close the stream.	reentrant
fprintf	Output a formatted string to the stream.	reentrant
fputc	Output a character to the stream.	reentrant
fputs	Output string to the stream.	reentrant
fwrite	Write to the stream.	reentrant
printf	Output a formatted string to standard output.	non-reentrant
putc	Output a character to standard output.	non-reentrant
putchar	Output a character to standard output.	non-reentrant
puts	Output string to standard output.	non-reentrant
snprintf	Formatted string output to string. Maximum output size can be specified.	reentrant
sprintf	Output formatted string to string.	reentrant
vfprintf	Output a formatted string to the stream.	reentrant
vprintf	Output a formatted string to standard output.	reentrant
vsnprintf	Formatted string output to string. Maximum output size can be specified.	reentrant
vsprintf	Output formatted string to string.	reentrant

5.3.9. stdlib.h

Table 33. The library functions available in stdlib.h

Function/Macro Name	Description	Reentrancy
abort	Causes the program to terminate abnormally.	reentrant
abs	Calculates the absolute value of an int type.	reentrant
atoi	Converts decimal strings to an int type.	reentrant
atol	Converts a decimal string to a long type.	reentrant
bsearch	Binary search.	reentrant
div	Performs division between signed int types and returns the quotient and remainder.	reentrant
exit	Terminate the program.	reentrant
labs	Calculates absolute values of long type.	reentrant
ldiv	Performs division between signed long types and returns the quotient and remainder.	reentrant
qsort	Quick sort.	reentrant
rand	Generate pseudo-random numbers.	non-reentrant
rand_r	Generate pseudo-random numbers.	reentrant
srand	Initialize pseudo-random numbers.	reentrant
strtol	Converts a string to a long type.	non-reentrant
strtoul	Converts a string to unsigned long type.	non-reentrant

5.3.10. string.h

Table 34. The library functions available in string.h

Function/Macro Name	Description	Reentrancy
memccpy	Copy the memory data. Maximum copy size can be specified.	reentrant
memchr	Search for a character from the beginning of the memory.	reentrant
memcmp	Compare the memory data.	reentrant
memcpy	Copy the memory data.	reentrant
memmove	Move the memory data.	reentrant
memrchr	Search for a character from the end of the memory.	reentrant
memset	Set the specified data in memory.	reentrant
strcat	Concatenate strings.	reentrant
strchr	Search for a character from the beginning of a string.	reentrant
strcmp	Compare strings.	reentrant
strcpy	Copy the string.	reentrant
strcspn	Find out how many characters from the beginning of the string are not any of the characters in the specified string.	reentrant
strlcat	Concatenate strings. Maximum string size can be specified.	reentrant
strlcpy	Copies a string. Maximum string size can be specified.	reentrant
strlen	Get the string length.	reentrant
strncat	Concatenate strings. The number of characters to be concatenated can be specified.	reentrant
strncmp	Compare strings. The number of characters to compare can be specified.	reentrant
strncpy	Copies a string of characters. The number of characters to be copied can be specified.	reentrant
strnlen	Get string length. Maximum string length can be specified.	reentrant
strpbrk	Returns the first position from the beginning of the string at which a character in the specified string is found.	reentrant
strrchr	Search for characters from the end of a string.	reentrant
strspn	Find out how many characters from the beginning of the string are contained in the specified string.	reentrant
strstr	Searches for a specified string from the beginning of the string.	reentrant
strtok	Split a string into tokens.	non-reentrant
strtok_r	Split a string into tokens.	reentrant

5.3.11. matisse/cpufunc.h

Table 35. The library functions available in matisse/cpufunc.h

Function/Macro Name	Description	Reentrancy
_NOP	Output the nop instruction.	reentrant
_HLT	Output the hlt instruction.	reentrant

5.3.12. matisse/interrupt.h

Table 36. The library functions available in matisse/interrupt.h

Function/Macro Name	Description	Reentrancy
ei	Enable interrupt flag.	reentrant
isei	Get whether the interrupt flag is enabled or not.	reentrant
di	Disable the interrupt flag.	reentrant
swi	Generated maskable interrupt in software interrupt.	reentrant
swnmi	Generated non-maskable interrupt in software interrupt.	reentrant

5.3.13. matisse/sections.h

Table 37. The library functions available in matisse/sections.h

Function/Macro Name	Description	Reentrancy
SECTION_TEXT	Place data or functions into .text section.	reentrant
SECTION_RODATA	Place data or functions into .rodata section.	reentrant
SECTION_DATA	Place data or functions into .data section.	reentrant
SECTION_BSS	Place data or functions into .bss section.	reentrant
SECTION_NOINIT	Place data or functions into .noinit section.	reentrant

5.3.14. util/atomic.h

Table 38. The library functions available in util/atomic.h

Function/Macro Name	Description	Reentrancy
ATOMIC_BLOCK	Create an interrupt-disabled block, whitch is called an atomic block. The interrupt flag is disabled in the block.	reentrant
NONATOMIC_BLOCK	Create an interrupt-enabled block, which is called a non-atomic block. The interrupt flag is disabled in the block.	reentrant
ATOMIC_RESTORESTATE	Used as a parameter to ATOMIC_BLOCK. Restore interrupt flag after the end of the atomic block.	reentrant
ATOMIC_FORCEON	Used as a parameter to ATOMIC_BLOCK. Enable interrupt flag after the end of the atomic block.	reentrant
NONATOMIC_RESTORESTATE	Used as a parameter to NONATOMIC_BLOCK. Restore interrupt flag after the end of the non-atomic block.	reentrant
NONATOMIC_FORCEOFF	Used as a parameter to NONATOMIC_BLOCK. Disable interrupt flag after the end of the non-atomic block.	reentrant

5.4. Type definition details

The types that are defined in each header file are described below.

5.4.1. inttypes.h

Table 39. The types defined in inttypes.h

Type Name	Definition

_Lldiv_t	struct {
	    intmax_t quot;
	    intmax_t rem;
	};

imaxdiv_t	_Lldiv_t

5.4.2. setjmp.h

Table 40. The types defined in setjmp.h

Type Name	Definition

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

5.4.3. stdarg.h

Table 41. The types defined in stdarg.h

Type Name	Definition
__gnuc_va_list	__builtin_va_list
va_list	__gnuc_va_list

5.4.4. stddef.h

Table 42. The types defined in stddef.h

Type Name	Definition
__PTRDIFF_TYPE__	int
ptrdiff_t	__PTRDIFF_TYPE__
__SIZE_TYPE__	unsigned int
size_t	__SIZE_TYPE__

5.4.5. stdint.h

Table 43. The types defined in stdint.h

Type Name	Definition
int8_t	signed char
uint8_t	unsigned char
int16_t	signed int
uint16_t	unsigned int
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. The types defined in stdio.h

Type Name	Definition

__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. The types defined in stdlib.h

Type Name	Definition

div_t	struct {
	int quot;
	int rem;
	};

ldiv_t	struct {
	long quot;
	long rem;
	};

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

5.5. Macro definition details

The macros that are defined in each header file are described below.

5.5.1. errno.h

Table 46. The macros defined in errno.h

Macro name	Definition	Value
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)
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. The macros defined in iso646.h

Macro name	Definition	Value
and	&&	&&
and_eq	&=	&=
bitand	&	&
bitor	|	|
compl	~	~
not	!	!
not_eq	!=	!=
or	|	|
or_eq	|=	|=
xor	^	^
xor_eq	^=	^=

5.5.3. limits.h

Table 48. The macros defined in limits.h

Macro name	Definition	Value
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. setmjp.h

Table 49. The macros defined in setjmp.h

Macro name	Definition	Value
JBLEN	11	11 (0xb)
__ATTR__NORETURN__	__attribute__((__noreturn__))	__attribute__((__noreturn__))

5.5.5. stdboo.h

Table 50. The macros defined in stdbool.h

Macro name	Definition	Value
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. The macros defined in stddef.h

Macro name	Definition	Value
NULL	((void *)0)	((void *)0)

5.5.7. stdint.h

Table 52. The macros defined in stdint.h

Macro name	Definition	Value
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)

5.5.8. matisse/extmem.h

Table 53. The macros defined in matisse/extmem.h

Macro name	Definition	Value
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. The macros defined in matisse/signature.h

Macro name	Definition	Value
SIGNATURE_HEADER	0x6D	109 (0x6D)

5.5.10. matisse/version.h

Table 55. The macros defined in matisse/version.h

Macro name	Definition	Value
__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)

6. Specific functions

This section describes the features specific to the compiler (mtcc).

6.1. Extended program area

There are several variations of tinyMicon MatisseCORE™, and some CPU cores can use an extended program area (0x10000 - 0x1FFFF) in addition to the normal 16-bit memory area (0x0000 - 0xFFFF).
The differences between normal memory area and the extended program area are as follows.

Table 56. The differences between normal memory area and extended program area

Memory type	Placing program data	Placing constant data	Used as data area (.data, .bss or stack)
Normal memory area	Placeable	Placeable	Usable
Extended program area	Placeable	Non-placeable	Unusable

6.1.1. How to use

To use the extended program area, you need to perform the following two steps.

1. Include the header file <matisse/extmem.h> in the C source file and use the macros for the extended program area.
2. Specify the command line option to enable the extended program area at compile time.

6.1.2. Example source file

Write the source file as follows. For a detailed description of the macros, refer to the comments in the file <matisse/extmem.h>.

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

6.1.3. Example of compilation options

To compile a source file that uses the extended program area, specify --ex-program-size= in the command line option as shown below.

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

--ex-program-size= option specifies the size of the extended program area. If the size of the extended program area is not enough, a compile error will occur.
-Map= option specifies the map file. By looking at the map file, you can check whether the function is placed in the extended program area or not.

6.2. Interrupt handler

tinyMicon MatisseCORE™ supports two types of interrupts: maskable and non-maskable interrupts.

The compiler (mtcc) can define interrupt handlers as C functions to be executed when an interrupt occurs.

6.2.1. How to use

To use the interrupt handler, you need to perform the following two steps.

1. Include the header file <matisse/interrupt.h> in the C source file.
2. Use the macro ISR() to define interrupt handlers, where ISR(IRQ_vect) is the interrupt handler for maskable interrupts and ISR(NMI_vect) is the interrupt handler for non-maskable interrupts.

6.2.2. Example source file

6.2.2.1. Interrupt handler

The macro ISR(IRQ_vect) can be used to create a maskable interrupt handler, and the macro ISR(NMI_vect) can be used to create a non-maskable interrupt handler.

```
#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. Interrupt flag on/off

You can also turn on/off the interrupt flag using the macros defined in the header file <matisse/cpufunc.h>.

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

6.2.2.3. Atomic block

The macros defined in the <util/atomic.h> header file can be used to create an atomic block. An atomic block is a code block where maskable interrupts do not occur. Please refer to the comments in the <util/atomic.h> file for a detailed description of the macros for atomic blocks.

```
#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. Precautions

Interrupt handlers have a size limit and must be 256 bytes or less. Avoid writing many processes in the interrupt handler. If it is necessary, cut out the processes to a separate function and call the function from the interrupt handler.
Exceeding the size limit of the interrupt handler will result in a compile-time error.

6.3. Standard output

tinyMicon MatisseCORE™ does not support standard I/O (stdin, stdout, stderr) because the CPU is dedicated for LSI embedded systems.
However, the pseudo standard output (stdout) can be used only when debugging with the command line debugger (mtsim).

6.3.1. How to use

To use the pseudo standard output by the command line debugger (mtsim), you need to do the following steps

1. Install the command line debugger (mtsim).
2. Write some special codes to enable the standard output feature in the source file.
3. Write a script file to enable the standard output function of the command line debugger (mtsim).
4. Build the source file and generate the binary file.
5. Run msim using the script file and binary file you created.

6.3.2. Example source file

Write the source file as follows.
Refer to the comments in the header file <stdio.h> file for the special stream created by the FDEV_SETUP_STREAM macro.

```
#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;
}
```

6.3.3. Debugger script file example

Write a script file to control the debugger (mtsim) as shown below and save it as script.txt.
Use the debugger's hook command to link the hook address to a function in the hook DLL. The hook command will execute the hook function in the DLL when the debugger (mtsim) accesses the hook address.
The hook DLL is in "C:\Program Files\ROHM\Matisse\mtsim\bin\StdoutHook.dll" by default.
0x02 is the interrupt vector address for non-maskable interrupts. When swnmi() is executed in the source code, a non-maskable interrupt occurs and the execution is moved to 0x02 then the hook function is called.

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

6.3.4. Debugger execution

If you run debugger with a command like the following, you will see the string on the standard output of mtsim.

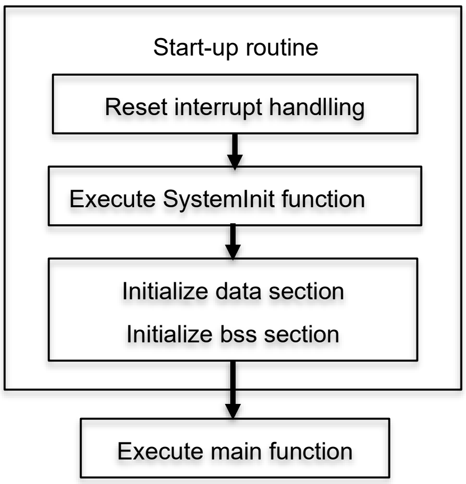
The -sim option specifies execution in simulation mode, and the -s option specifies a script file. Please refer to the mtsim documentation for more details on the command line.

The command line options can be changed according to your usage.

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

6.4. SystemInit function

Create a SystemInit function if you need to initialize the system when the program starts. For example, SystemInit function is used to change the CPU clock frequency, set the peripherals, change the memory map, change the stack, etc.
If you have created a SystemInit function, the SystemInit function will be automatically called from the start-up routine.
The timing when the SystemInit function is called is as follows.



6.4.1. Example source file

If you write the SystemInit function as follows, it will be executed automatically at startup.

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

6.5. Noinit section

Variables in the bss section will be initialized to 0 in the start-up routine, but if you do not want them to be initialized, use the noinit section.
Variables placed in the noinit section will not be initialized.

6.5.1. How to use

Include the header file <matisse/sections.h> in your C source file and use the macro SECTION_NOINIT for the target variable.

6.5.2. Example source file

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

SECTION_NOINIT int notInitializedVariable;

SECTION_NOINIT int notInitializedArray[10];
```

6.5.3. Precautions

Initial values cannot be specified for variables placed in the noinit section.
Static variables cannot be placed in the noinit section.

6.6. Signature data

By including the header file <matisse/signature.h> in the C source file, the compiler (mtcc) can embed signature data for identification in the output files (both binary format file and ELF format file).

The signature data consists of 7 bytes of data, and the format is as follows.

Table 57. The format of the signature data

Data index	Description
0	0x6D ('M' in ASCII code. The initial character of Matisse.)
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

The locations where signature data are embedded are as follows.

Table 58. The locations where the signature data are embedded

File types	Locations
Binary format file	The end of RAM data.
ELF format file	In the signature section.

6.6.1. How to use

Simply include the header file <matisse/signature.h> in the C source file to embed the signature data in the output file.

6.6.2. Example source file

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

6.6.3. Precautions

The header file <matisse/signature.h> can only be included in one file of the entire program. If you include it in multiple files, a compile-time error will occur.

6.7. Inline assembler

The compiler (mtcc) supports the feature of simple inline assembler.

6.7.1. Statement format

The inline assembler statement is written in the following format.

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

Please be sure to specify __volatile__ so that it is not deleted by the optimization of the compiler.

Assembler instructions can be written directly in the assembler_code part.

6.7.2. Example source file

The inline assembler is written in the C language function as follows.

```
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. Precautions

Be careful that the registers used in the inline assembler do not conflict with the registers used in the C source file part.

If the registers conflict, the values will be overwritten, causing unintended results.

6.8. pragma

The compiler (mtcc) supports the pragmas. By writing pragma in the source code, you can provide additional information to the compiler.

6.8.1. How to use

The supported #pragmra directives are as follows.

Table 59. The supported #pragmra directives

pragma	Description
#pragma clang section [section_type="name"]	Change the section name of global objects to "name". The following can be used for "section_type" ● text ● rodata ● data ● bss

6.8.2. Example source file

```
#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. Precautions

If you do not understand the pragmas, they may cause unintended compilation errors or bugs. Please make sure you understand what you are doing before using them.

6.9. attribute

The compiler (mtcc) supports the attributes. By writing attributes in the source code, you can specify the attributes of the objects in the program.

6.9.1. How to use

The supported attribute directives are as follows.

Table 60. The supported attribute directives

attribute	Description
__attribute__ ((__const__))	When specified for a function, it notifies the compiler that the function is a const function, which does not change global or static variables.
__attribute__((__interrupt__))	When specified for a function, it notifies the compiler that the function is an interrupt handler.
__attribute__((__nodebug__))	Debug information about the specified object will not be generated.

__attribute__((__noreturn__))	When specified for a function, the function becomes will not return to the caller.
__attribute__((__pure__))	When specified for a function, it notifies the compiler that it is a pure function, which always returns the same value when called with the same arguments.
__attribute__ ((section ("[section_name]")))	The specified object will be placed in section_name.
__attribute__((unused))	Can be specified for variables, functions, and arguments. Suppresses unused warning.

6.9.2. Example source file

```
__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. Precautions

If you do not understand the attributes, they may cause unintended compilation errors or bugs. Please make sure you understand what you are doing before using them.

6.10. Predefined Macros

The compiler (mtcc) has predefined macros. These macros are predefined inside the compiler (mtcc) and are used to get information about the compiler type and the compiler version.

6.10.1. How to use

The predefined macros available in the compiler (mtcc) are as follows.

Table 61. The predefined macros available in the compiler (mtcc)

Macro name	Value
MATISSE	1
__MATISSE	1
__MATISSE__	1
__MTCC_MAJOR__	1
__MTCC_MINOR__	1
__MTCC_REVISION__	0

6.10.2. Example source file

```
#ifdef __MATISSE__
	// Write the source code for Matisse here.
#else
	// Write the source code for other architecture here.
#endif

#if (__MTCC_MAJOR__ == 1) && (__MTCC_MINOR__ == 1) && (__MTCC_REVISION__ == 0)
	// Write the source code for mtcc V1.01.00 here.
#endif
```

7. Calling convention

The machine language code output by the compiler (mtcc) follows certain convention for passing values to C functions and receiving the return value from C function calls. This convention is called the calling convention.

7.1. Register types

Table 62. Register types

Register name	Register type	Description
R0	Special register	This register is used as the lower byte of a 16-bit PC (Program Counter). When used as the source operand in the LDR instruction, the value of the flag register can be obtained. When used in arithmetic operations or bitwise operations, it can be used as a zero register.
R1	Special register	This register is used as the upper byte of the 16-bit PC (Program Counter).
R2, R3	Special register	These registers are used as a 16-bit SP (Stack Pointer), where R2 is the lower byte and R3 is the upper byte.
R4, R5	Special Register Callee-Saved Register	These registers are used as a 16-bit FP (Frame Pointer), where R4 is the lower byte and R5 is the upper byte. They are also used as Callee-Saved general purpose registers in functions that do not have FP.
R6, R7, R8, R9	Callee-Saved Register	General purpose registers whose values are not changed by function calling.
R10, R11	Temporary register Caller-Saved register	Registers used for various temporary operations. The values of these registers may be changed by function calling.
R12, R13, R14, R15	Caller-Saved register	General purpose registers whose values may be changed by function calling.

7.2. Function parameter calling conventions

The arguments are allocated to the registers in order. If there are not enough registers, the remaining arguments are allocated to the stack.

7.2.1. Conventions for allocating arguments to registers

- Allocate the arguments to R15, R14, R13, and R12 in the order in which they are declared (leftmost to rightmost).
- A 1-byte argument is extended to 2-bytes and allocated to registers.

7.2.2. Conventions for allocating arguments to stack

- The arguments are pushed onto the stack in order from the rightmost one.

7.3. Function return value calling conventions

- If the return value is a structure or a union, the return value will be pushed onto the stack from the upper byte of the last member.
- If the size of the return value is 1-byte, the return value will be allocated to the register R14.
- If the size of the return value is 2-bytes, the lower byte will be allocated to the register R14 and the upper byte will be allocated to the register R15.
- If the size of the return value is 4-bytes, the return value will be allocated to the register R12, R13, R14, and R15 from the lower byte.

8. Unsupported features

mtcc and mtld are developed based on open-source software. The functionality of those open-source software is inherited by mtcc and mtld. In other words, there are additional features available in mtcc and mtld that are not described in this document. These features are called "unsupported features".
We have not tested unsupported features. And we do not guarantee that they will work properly. Please use them based on your own judgment.

8.1. Compiler command line options

A list of command line options available for the OSS compiler on which mtcc is based is provided in the attachment below. All functionalities listed in this attachment that are not described in this document are unsupported features and are not guaranteed to work properly. Please use them based on your own judgment.

8.2. Attributes

A list of attributes available for the OSS compiler on which mtcc is based is provided in the attachment below. All functionalities listed in this attachment that are not described in this document are unsupported features and are not guaranteed to work properly. Please use them based on your own judgment.

8.3. Linker command line options

A list of command line options available for the OSS linker on which mtld is based is provided in the attachment below. All functionalities listed in this attachment that are not described in this document are unsupported features and are not guaranteed to work properly. Please use them based on your own judgment.

9. Open-source software licenses

This software includes open-source software (hereinafter referred to as "open-source software program") provided under the following license conditions, in addition to software for which ROHM owns or is licensed.
Open-source software programs are subject to their respective license terms, so in the event of a conflict between the license terms of an open-source software program and this material, the license terms of the open source software program shall prevail.

Included open-source software and their license terms
・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)

10. Trademark notices

"Windows" is a trademark of Microsoft Group companies.
"Intel" is a trademark of Intel Corporation or its subsidiaries.
"Core™" is a trademark or registered trademark of Intel Corporation or its subsidiaries.
“tinyMicon MatisseCORE™” and “matiseye™” are a trademark or registered trademark of ROHM Corporation.

End.

Caution

1. The information written in these materials regarding the software and system (hereinafter collectively “Software”) and the contents of the materials are current as of the date of the material’s issuance, and may be changed by ROHM, at any time and for any reason, without prior notice.

2. If you plan to use the Software in connection with any equipment or device (such as the medical equipment, transportation equipment, traffic equipment, aerospace equipment, nuclear power control equipment, vehicle equipment including the fuel control system and/or car accessories, and/or various kinds of safety devices etc.) which require extremely high reliability, and whose breakdown or malfunction relate to the risk of personal injury or death, or any other serious damage (such usage is hereinafter called “Special Usage”), you must first consult with the ROHM’s sales representative. ROHM is not responsible for any loss, injury, or damage etc. incurred by you or any other third party caused by any Special Usage without ROHM’s prior written approval.

3. Semiconductor products may break or malfunction due to various factors. You are responsible for designing, testing, and implementing safety measures in connection with your use of any ROHM products using the Software (such ROHM products are hereinafter called “Product”) Such safety measures include, but are not limited to, derating, reductant design, fire spread prevention, backup, and/or fail safe etc. in order to prevent the accident resulting in injury or death and/or fire damage etc.. ROHM is not responsible and hereby disclaims liability for any damage in relation to your use beyond the rated value, or the non-compliance with any precaution for use.

4. ROHM is not responsible for any direct and/or indirect damage to you, or any third parties, (including the damage caused by loss of intangible asset such as information, data, or program etc., loss and/or interruption of profit) which is caused by the use or impossibility to use of the Software.

5. Since the Software, these materials, and/or the Product contain confidential information of ROHM’, including technical information, and/or trade secrets, you are prohibited from engaging in any of the following acts in whole or part, without ROHM’s prior written approval:
   (i) disclosing any ROHM confidential information to a third party;
   (ii) disassembling, reverse engineering, and/or any other analysis;
   (iii) reprinting, copy, and/or reproduction; or
   (iv) removing the copyright notice included in the Software.

6. When exporting the Software, or the technology and/or confidential information written in these materials, you are required to follow the applicable export control laws and regulations such as “Foreign Exchange and Foreign Tarade Act” and/or “Export Administration Regulations (EAR).”.

7. ROHM disclaims all warranties, statutory or otherwise, and ROHM hereby disclaims any warranty for non-infringement for the Software and/or the information written in these materials. Accordingly, ROHM is not liable to you for any direct or third-party claims of infringement of rights.

8. No license, whether expressly or implied, is granted hereby under any intellectual property rights or other rights of ROHM or any third parties with respect to the Software or Products or the information contained in these materials.

9. You agree to indemnify, defend and hold harmless ROHM and ROHM’s officers and/or employees from responsibility, and hold them harmless, and defend them from any damage, loss, penalty, or cost caused by any claim of liability (including but not limited to the attorney fees) resulting from, or incurred relating to the following acts:
   (1) any alleged infringement of a third party’s rights or the violation of laws caused by reading, download, encryption, summarization, copy, or transfer etc.; or
   (2) violation of these materials.

10. ROHM does not guarantee that these materials or the Software is error free. ROHM shall not be in any way responsible or liable for any damages, expenses, or losses incurred by you or third parties resulting from errors contained in these materials.

Thank you for using ROHM products.
For inquiries about our products, please contact us.

ROHM Customer Support System

https://www.rohm.co.jp/contactus