tinyMicon MatisseCORE™

mtld user's guide

Linker User's Guide for tinyMicon MatisseCORE™

No. 68UG027E Rev.003
2025.3

Update history

Date	Version	Description
2024/06/05	Rev.001	Describe the contents of mtld V1.01.00
2025/02/21	Rev.002	The description of “3.4.12.3.1.3(type)” added
2025/03/13	Rev.003	Updated for mtld V2.00.00 and V3.00.00

Table of Contents

1. Overview

mtld is a linker for tinyMicon MatisseCORE™, an ultra-small 8-bit CPU (Central Processing Unit) developed by ROHM for embedding into LSI (Large Scale Integration).

Linker is a program that links an object file output by the compiler to a single file.

1.1. Characteristics

mtld has the following characteristics:

1 ELF 32-bit LSB is supported.

2 Links between object files (*.o) and static library files (*.a) are possible.

3 Various command-line options.

2. Operating Environment

The operating environment of mtld is described below.

2.1. System Requirements

Table 1. System Requirements

OS	Windows™ Server 2012 (64-bit) Windows™ 7 (32-bit/64-bit) Windows™ 10 (32-bit/64-bit) Windows™ 11 (64-bit)
CPU	Equivalent to Intel™ Core™ series or a CPU with equivalent performance.
Memory	Equipped with 4 GByte or more.
HDD/SSD	More than 200 MB of free space.

2.2. Install

If you run the Matisse developing environment installer (matisse-dev-env-installer-XX.XX.XX.exe), mtld will also be installed.

By default, the set of tools will be installed in "C:\Program Files\ROHM\Matisse\".

3. How to use

3.1. Starting mtld

mtld runs from the command line as follows: Multiple input files can be specified, and an object file (*.o) generated by mtcc can be specified.

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

An example of a command line that runs mtld is shown below.

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

3.2. Command Line Options List

The following list of command-line options is available for linker (mtld):

Command line option	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 for the library file. More than one can be specified.
-l	Specifies the library file to link. More than one can be specified.
--defsym	Defines a global symbol for the linker. More than one can be specified.
--script	Uses this option to change the linker script.
-Map=	Specifies the destination of the map file.
--exclude-mul=	Specifies whether there is a multiplier for the core to be targeted.
--crt	Modifies the run-time library that you want to link.
--libc	Modifies the library file to be linked.
-v	Displays version information.
-h	Displays help messages.

3.3. Command-line option details

3.3.1. -o

3.3.1.1. Description

The -o option specifies the output filename in the following format: The binary file (ROM image) is output. At the same time, the ELF file is automatically generated (the file name of the ELF file changes the output file name extension to .elf).

```
-o filename
```

3.3.1.2. Examples of Use

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

3.3.2. --program-size=

3.3.2.1. Description

The --program-size= option specifies the size of the program area in the following format: The program area is the size of the specified number multiplied by 4 kbytes. If you omit this option, the default value of 8 (32kbytes) applies.

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

If the sum of the values specified by --program-size= and --data-size= is 17 (68kbytes) or greater, a memory capacity error occurs.

3.3.2.2. Examples of Use

```
> mtld --program-size=8 input.o -o output.bin
```

3.3.3. --data-size=

3.3.3.1. Description

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

```
--data-size=n [n=1,2,3,4,5,6,7,8]
```

If the sum of the values specified by --program-size= and --data-size= is 17 (68kbytes) or greater, a memory capacity error occurs.

3.3.3.2. Examples of Use

```
> mtld --data-size=1 input.o -o output.bin
```

3.3.4. --ex-program-size=

3.3.4.1. Description

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

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

3.3.4.2. Examples of Use

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

3.3.5. -L

3.3.5.1. Description

The -L option specifies the directory in the following format and adds the specified directory to the library file search path. This option can be more than one.

```
-Ldirname
```

3.3.5.2. Examples of Use

```
> mtld -L./lib input.o -o output.bin
```

3.3.6. -l

3.3.6.1. Description

The -l option specifies the library file to link in the following format: The specified library file is searched for in the directory that is registered in the library search path with the -L option. This option can be more than one.

```
-llibname
```

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

- Static library
- File name begins with "lib"
- The file extension is ".a"

The optional parameter -l specifies the string from the library filename that "lib" and ".a" are removed.

3.3.6.2. Examples of Use

```
> mtld -ltest input.o -o output.bin // The library file named libtest.a will be linked
```

3.3.7. -defsym

3.3.7.1. Description

The -defsym option defines the symbol in the following format: This option can be more than one.
The defined symbols are enabled in the linker script.

```
-defsym SYMBOL=EXPRESSION
```

3.3.7.2. Examples of Use

```
> mtld --defsym __my_symbol=0xABCD input.o -o output.bin
```

3.3.8. --script

3.3.8.1. Description

The script option specifies the linker script to use for link processing in the following format: If you omit this option, mtld uses the default linker script. The default linker script is "C:\Program Files\ROHM\Matisse\C\lib\ldscripts\matisse.x".

```
--script filename
```

3.3.8.2. Examples of Use

```
> mtld --script my_linker_script.x input.o -o output.bin
```

3.3.9. -Map=

3.3.9.1. Description

The -Map= option specifies the output destination for the map file in the following format: Map files contain the addresses of global variables and functions.

```
-Map=filename
```

3.3.9.2. Examples of Use

```
> mtld -Map=mapfile.map input.o -o output.bin
```

3.3.10. --exclude-mul=

3.3.10.1. Description

The --exclude-mul option specifies whether the target CPU core has a multiplier in the following format: If you omit this option, the default value (true) applies.

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

tinyMicon MatisseCORE™ has several variations, including a CPU core with a multiplier and a CPU core without a multiplier. This option can be used to create machine language files suitable for each core.

Set value	Output result
true	Creates a binary file for a CPU core without a multiplier.
false	Creates a binary file for a CPU core with a multiplier.

3.3.10.2. Examples of Use

```
> mtld --exclude-mul=true input.o -o output.bin
```

3.3.10.3. Details

This option determines the type of compiler runtime library (CRT) that will be linked. There are two types of CRTs: the one uses the mul instructions for the CPU core with multiplier and the one doesn't use the mul instructions for the CPU without the multiplier.
Applying true to the --exclude-mul option links CRTs for CPU cores without a multiplier.
Applying false to the --exclude-mul option links the CRT for the CPU core that has a multiplier.
The multiplication process is implemented as the CRT functions __mulhi3 and __mulsi3. If the application part requires multiplication, these CRT functions are called.
That is, if you apply false to the--exclude-mul option, the mul instruction will not be generated in the application part.
This option does not change the code of the application. Only the CRT being linked changes.

3.3.11. --crt

3.3.11.1. Description

The --crt option replaces the run-time library with the file specified in the following format: If you omit this option, the parameter of --exclude-mul= option determines the run-time library file being linked. If --exclude-mul=true, "C:\Program Files\ROHM\Matisse\C\lib\crt.a" is linked, and if --exclude-mul=false, "C:\Program Files\ROHM\Matisse\C\lib\crt_with_mul.a" is linked.
Runtime library files that are specified in this option must meet all of the following conditions:

- Static library
- Have defined all CRT symbols referenced by the linker script
- Implementing all the runtime libraries required by the compiler (mtcc)

3.3.11.2. Examples of Use

```
> mtld --crt=my_crt.a input.o -o output.bin
```

3.3.12. --libc

3.3.12.1. Description

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

The library file specified by this option must satisfy all the following conditions:

- Static library
- All standard library functions supported by the compiler (mtcc) are implemented

3.3.12.2. Examples of Use

```
> mtld --libc=./libcanother.a input.o -o output.bin
```

3.3.13. --gc-sections

3.3.13.1. Description

The --gc-sections option enables the garbage collection function of the linker (mtld). This option automatically removes unused sections when linked. This enables you to reduce the size of the final executable file or library.

NOTE: The sections specified with KEEP keyword (3.4.12.4.2 KEEP keyword) are not subject to garbage collection.

3.3.13.2. Examples of Use

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

3.3.14. -v

3.3.14.1. Description

The -v option shows the version information for the linker (mtld) to standard output.

3.3.14.2. Examples of Use

```
> mtld -v
```

3.3.15. -h

3.3.15.1. Description

The -h option shows a help message for the linker (mtld) to standard output.

3.3.15.2. Examples of Use

```
> mtld -h
```

3.4. Linker script

A linker script is a script file that defines the linker (mtld) behavior during link processing. You can specify details for the program's memory layout by describing the linker script.

The linker script functions available for linker (mtld) are described in the following chapters.

"C:\Program Files\ROHM\Matisse\C\lib\ldscripts\matchisse.x" is used as the default linker script. You can use the --script command-line option to change the linker script that you want to apply.

3.4.1. Linker script overview

The linker (mtld) combines the input files into one output file according to the linker script description. Each input and output file is in a data format called an ELF. An I/O file is called an object file. Each object file has multiple sections. The section of the input file is called an input section, and the section of the output file is called an output section.

3.4.1.1. Section

Each section of the object file has a name, size, and content. When you run linker (mtld), the contents of the input section are placed in the memory region specified by linker script and an output file is created.

3.4.1.2. Symbol

Each object file has a symbol. Each symbol has a name and address. When you compile a C language program and convert it to an object file, symbols for the functions, global variables, and static variables are created. You can use these symbols as parameters of commands and expressions in the linker script. In addition, you can define a new symbol in the linker script or change the value of defined symbols.

3.4.1.3. LMA and VMA

The output section has two addresses. The first is the Load Memory Address (LMA), the address to which the section is loaded. The second is the Virtual Memory Address (VMA), which is the address of the section when the program is being executed. In most cases, these two addresses are identical. As an example of the difference between these two addresses, the data section is loaded into the ROM and copied to the RAM when the program is started (this method is often used to initialize global variables in embedded systems). In this case, you must write a linker script like the LMA of the section is in the ROM area, and the VMA is in the RAM area.

3.4.2. Linker script format

The linker (mtld) supports the following BNF notation:

```
Linker script := Comment* | Top level command* | MEMORY command? | SECTIONS Command?

Top level command := Assignment | ENTRY command | PROVIDE keyword | ASSERT command | SIZEOF function | ADDR function | LOADADDR function
```

3.4.3. Comments

3.4.3.1. Description

You can write comments in the linker script. The area enclosed in /* and */ is a comment. The contents of the comment do not affect the operation of the linker script.

3.4.3.2. Examples of Use

```
/* This is a comment. */

/* multiple
Lines
Of
```

```
Comments.
*/
```

3.4.4. Top level commands

Top level commands that can be written in the top level of linker script are described below.

However, you can also use top level commands described in chapters 3.4.4.1. Assignment expression, 3.4.6. PROVIDE keyword, 3.4.8. SIZEOF function, 3.4.9. ADDR function, 3.4.10. LOADADDR function, 3.4.12. SECTIONS command.

3.4.4.1. Assignment expression

3.4.4.1.1 Syntax

The BNF representation of the assignment expression is as follows.

```
Assignment expression:= symbol name = expression;
```

When the assignment expression is evaluated, the left-hand symbol is substituted with the value of the right-hand symbol.

If the symbol on the left-hand is already defined, the symbol value is updated with the right-hand value. If the symbol is not defined, the symbol is newly defined, and the right-hand value is substituted.

3.4.4.1.1.1 Examples of Use

```
symbol1 = 10; /* A new symbol "symbol1" is defined and 10 is assigned */
symbol1 = 20; /* The value of "symbol1" is updated to 20 */
```

3.4.4.2. Expressions

Expressions include immediate values, arithmetic expressions, and comparison expressions. Each item is described below.

3.4.4.2.1 Immediate value

3.4.4.2.1.1 Syntax

The BNF notation of immediate value is as follows. Immediate values are available only for integers.

```
Immediate value := decimal character+ Suffix? | 0x hexadecimal characters+ suffixes?
Decimal character := 0|1|2|3|4|5|6|7|8|9
Suffix := K| M
Hexadecimal := 0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f
```

A decimal or hexadecimal number with the suffix K multiplies the value by 1024, and the suffix M multiplies the value by 1048576. The range of immediate values that are available is 0 to 4294967295 (0xFFFFFFFF). The immediate value greater than 4294967295 is replaced with 4294967295 at the time of evaluation.

Do not use negative numbers for immediate values. Even if a negative number is written in the linker script, it is treated as an unsigned 32-bit integer during evaluation, which results in unintended results in arithmetic expressions and comparison expressions.

The immediate value cannot be used alone and must be used as the right-hand part of the assignment expression or as an argument to a function.

3.4.4.2.1.2 Examples of Use

```
symbol1 = 10; /* Defines symbol1 and assigns an immediate value of 10 */
symbol2 = 5K; /* Defines symbol2 and assigns 5 * 1024 results */
symbol3 = 10M; /* Defines symbol3 and assigns the results of 10 * 1048576 */
symbol4 = 0x1234abcd; /* Defines symbol4 and assigns hexadecimal immediate value 0x1234abcd */
```

3.4.4.2.2 Arithmetic expressions

3.4.4.2.2.1 Syntax

The BNF representation of the arithmetic expression is as follows.

```
Arithmetic expression  := Expression Operator's Expression
Operator := +| -| *
```

The right-hand part of the arithmetic expression is evaluated before the arithmetic expression is evaluated.

The + operator performs summation when the expression is evaluated, the - operator performs subtraction when it is evaluated, and the * operator performs multiplication when it is evaluated.

Valid operation results range from 0 to 4294967295 (0xFFFFFFFF). Only the lower 32-bit part of the result is used for evaluation of operations with out-of-range results.

An arithmetic expression cannot be used alone and must be used as right-hand part of the assignment expression or as an argument to a function.

3.4.4.2.2.2 Examples of Use

```
symbol1 = 10 + 2; /* symbol1 is defined and 12 is assigned */
symbol2 = symbol1-5; /* symbol2 is defined and 7 is assigned */
symbol3 = 0xFFFFFFFF; /* symbol3 is defined and 0xFFFFFFFF is assigned */
symbol4 = symbol1 + symbol3; /* symbol4 is defined and 11 is assigned (lower 32-bit part of 0x1000000B)*/
symbol5 = symbol1-symbol3; /* symbol5 is defined and 13 is assigned (lower 32-bit part of 0x10000000D)*/
symbol6 = 0x80000000 * 2; /* symbol6 is defined, and 0 is assigned (lower 32-bit part of 0x10000000) */
```

3.4.4.2.3 Comparison expressions

3.4.4.2.3.1 Syntax

The BNF representation of the comparison expression is as follows.

```
Comparison Expression := Expression Comparison Operator Expression
Comparison operator:= <|<=|>|>=|==|!=
```

The left-hand and right-hand terms of the comparison operator are evaluated before the comparison expression is evaluated.

The left-hand and right-hand terms are compared according to the comparison operator rules to evaluate the comparison expression. If the comparison result is true, the comparison result is 1, and if the comparison result is false, the comparison result is 0.

Comparison expressions cannot be used alone and must be used as right-hand part of an assignment expression or as an argument to a function.

The description of each comparison operator is as follows.

Comparison Operators	Description
<	For example, if "a < b", the result is true if "a" is less than "b", and false otherwise.
<=	For example, if "a <= b", the result is true if "a" is less than or equal to "b", and false if "a" is less than or equal to "b".
>	For example, if "a > b", the result is true if "a" is greater than "b", and false otherwise.
>=	For example, if "a >= b", the result is true if "a" is greater than or equal to b, otherwise it is false.
==	For example, if "a == b", the result is true if a and b are equal, and false otherwise.
!=	For example, when "a != b", the result is true if "a" is not equal to "b", and false otherwise.

3.4.4.2.3.2 Examples of Use

```
symbol1 = 10;
symbol2 = 20;
symbol3 = symbol1 < symbol2; /* Since the comparison is true, 1 is assigned to symbol3 */
symbol4 = symbol1 == symbol2; /* Since the comparison result is false, 0 is assigned to symbol4 */
```

3.4.4.2.4 .variable (dot variable)

3.4.4.2.4.1 Description

A . variable (or dot variable) is a special variable that you use to manipulate location counters that represent the addresses of the next data or section. The location counter is initialized at the start of the SECTIONS command and is then updated each time data is added to the output section.

If the variable is on the left side of the assignment expression, then the value of the location counter is updated to the right side of the assignment expression when the assignment expression is evaluated.

If the variable is used inside the expression, Variables can be evaluated to obtain the value of the current location counter.

3.4.4.2.4.2 Syntax

```
. = xxxx; /* used as the left-hand side of the assignment expression */
xxxx = . ; /* used as the right-hand side of the assignment expression */
```

3.4.4.2.4.3 Examples of Use

```
SECTIONS
{
  .text : {
    . = 0x1000; /* Set the location counter to 0x1000 */
    symbol1 = . ; /* Value of location counter is assigned to symbol1 */
    . = . + 0x200; /* Add 0x200 to the location counter */
  }
}
```

3.4.5. ENTRY command

3.4.5.1. Description

The ENTRY command is used by the linker script to set the entry point of the program. The entry point is the address of the first instruction that the program begins executing. The ENTRY command indicates that the symbol is an entry point by specifying a specific symbol name in the linker script.

If the ENTRY command is not used, the first address in the.text section becomes the entry point. If the ENTRY command is not used and there is no text section, then address 0 is the entry point.

3.4.5.2. Command syntax

```
ENTRY(symbol name)
```

3.4.5.3. Examples of Use

```
entry_point = 0x0000; /* defines symbol entry_point and assigns 0x0000 */
ENTRY(entry_point) /* Set the symbol entry_point to the entry point */
```

Note: Writing multiple ENTRY commands in the linker script does not result in an error, but only the ENTRY command that is listed last takes effect.

3.4.6. PROVIDE keyword

3.4.6.1. Description

The PROVIDE keyword is used to perform conditional symbol definitions in the linker script. The PROVIDE keyword defines a symbol only if the specified symbol is not already defined in the input object file or linker script.

3.4.6.2. Syntax

```
PROVIDE(symbol name = expression);
```

3.4.6.3. Examples of Use

```
PROVIDE (not_defined = 0x1234); /* define symbol not_defined only if not_defined is not defined and then assign 0x1234 */
```

3.4.7. ASSERT command

3.4.7.1. Description

The ASSERT command displays an error message if the specified condition is not met and is used to stop link processing. By using the ASSERT command, you can verify that certain conditions are met when linking.

3.4.7.2. Command syntax

The syntax of the ASSERT command is as follows.

```
ASSERT (immediate value | symbol name | arithmetic expression | comparison expression, error message);
```

3.4.7.3. Examples of Use

```
rom_size = 0xF000;
ram_size = 0xF000;
total_memory_size = 0x10000;

/* Error occurs when the sum of rom_size and ram_size exceeds total_memory_size
ASSERT(rom_size + ram_size <= total_memory_size, "Memory size over capacity.");
```

3.4.8. SIZEOF function

3.4.8.1. Description

Use the SIZEOF function to retrieve the size of the output section in the linker script. The result of the evaluation of the SIZEOF function is the size in bytes of the specified output section.

3.4.8.2. Function syntax

```
SIZEOF(output section name)
```

3.4.8.3. Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
}
symbol1 = SIZEOF(.text); /* assign symbol1 with the size of the text section */
```

3.4.9. ADDR function

3.4.9.1. Description

The ADDR function is used to retrieve the virtual memory address (VMA) at the head of a particular output section in the linker script. The result of the evaluation of the ADDR function is the virtual memory address (VMA) at the beginning of the specified output section.

3.4.9.2. Function syntax

```
ADDR(output section name)
```

3.4.9.3. Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
}
symbol1 = ADDR(.text); /* assign symbol1 with the start address of aVMA in the..text section */
```

3.4.10. LOADADDR function

3.4.10.1. Description

The LOADADDR function is used to obtain the load memory address (LMA) at the beginning of a specific output section in the linker script. The evaluation result of the LOADADDR function is the load memory address (LMA) at the beginning of the specified output section.

3.4.10.2. Function syntax

```
LOADADDR(output section name)
```

3.4.10.3. Examples of Use

```
SECTONS
{
  .text : {*(.text)}
}
symbol1 = LOADADDR(.text); /* assign symbol1 with the start address of LMA of the .text section */
```

3.4.11. ALIGN function

3.4.11.1. Description

The ALIGN function is used to obtain the memory address of a particular alignment in the linker script. Alignment means that the memory address points to a specific boundary. The memory boundary value specified by the ALIGN function must be the power

of 2. The evaluation result of the ALIGN function is the nearest memory boundary address in front of the current location counter. If the current location counter is on the specified memory boundary, the evaluation result of the ALIGN function is the current location counter.

For example, when the current location counter is 0x1010, the evaluation result of ALIGN (0x1000) is 0x2000, and when the current location counter is 0x2000, the evaluation result of ALIGN (0x1000) is 0x2000.

3.4.11.2. Command syntax

```
ALIGN(memory boundaries)
```

NOTE: The memory boundary value specified by the ALIGN function must be the power of 2.

3.4.11.3. Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
}

symbol1 = ALIGN(0x1000); /* assign symbol1 with the nearest 0x1000 boundary address in front of the location counter */
```

3.4.12. SECTIONS command

3.4.12.1. Description

The SECTIONS command defines how the sections in the input file are mapped to the sections of the output file in the linker script, and also defines how the output sections are placed in the memory of the target system.

NOTE: The SECTIONS command can contain up to one description in the linker script.

3.4.12.2. Command syntax

The syntax of the SECTIONS command is as follows. You can write zero or more sections-commands in parentheses.

```
SECTIONS
{
   section-command
   ...
}
```

3.4.12.3. section-command

The section-command can be one of the following:

- Description of the output section
- Assignment expression
- PROVIDE keyword, SIZEOF function, ADDR function, LOADADDR function, ALIGN function

Each section-command item is described below.

3.4.12.3.1 Description of the output section

The format of the description in the output section is as follows. Describe the attributes in the output section this way. The brackets are optional attributes.

```
section name [address] [(type)] : [AT(lma)] [ALIGN(boundary)] [SUBALIGN(boundary)]
{
   output-section-command
   ...
} [>region] [AT>region] [=fillexp]
```

Note: For detailed information about output-section-command, see “3.4.12.4. output-section-command”.

The output section attributes are described below.

3.4.12.3.1.1 section name

The name of the output section.

3.4.12.3.1.2 address

Specifies the start address of the output section. The start address specified is a virtual memory address (VMA).

Note: “3.4.12.3.1.7 >region” should not be used at the same time as this feature. For detailed information about VMA, see “3.4.1.3. LMA and VMA”.

3.4.12.3.1.3 (type)

Specifies the type of output section. The following types can be specified:

Type Name	Description
NOLOAD	Only the memory space of the specified section will be reserved and the data of the section will not be included in the output file.

3.4.12.3.1.4 AT(lma)

Specifies the load memory address (LMA) for the output section.

NOTE: “3.4.12.3.1.8AT>region” should not be use at the same time as this feature. For detailed information about LMA, see “3.4.1.3. LMA and VMA".

3.4.12.3.1.5 ALIGN(boundary)

Aligns the start address of the output section to a specific alignment. The start address to be aligned is a virtual memory address (VMA).

Alignment means that the addresses in memory are placed at a particular boundary (the power of 2).

Note: 3.4.11. You can also use the ALIGN function to specify location counters and symbol alignment.3.4.11. ALIGN function

3.4.12.3.1.6 SUBALIGN(boundary)

Aligns the start address of the input section in the output section to a specific alignment. The start address to be aligned is a virtual memory address (VMA).

Alignment means that the addresses in memory are placed at a particular boundary (the power of 2).

3.4.12.3.1.7 >region

Specifies the memory region to which the virtual memory address (VMA) of the output section is assigned.

NOTE: “3.4.12.3.1.2 address” should not be used at the same time as this feature. For detailed information about VMA, see “3.4.1.3. LMA and VMA”. For detailed information about the memory region, see “3.4.13. MEMORY command”.

3.4.12.3.1.8 AT>region

Specifies the memory region to which the load memory address (LMA) of the output section is allocated.

NOTE: “3.4.12.3.1.4 AT(lma)” should not be used at the same time as this feature. For detailed information about LMA, see “3.4.1.3. LMA and VMA”. For detailed information about the memory region, see “3.4.13. MEMORY command”.

3.4.12.3.1.9 =fillexp

The result of the evaluation of the specified expression is used as a fill pattern. A fill pattern is binary data that fills an area in the output section where the data is not located. The available fill pattern sizes are 1 byte to 4 bytes.

3.4.12.3.2 Assignment expression

3.4.12.3.2.1 Description

“3.4.4.1. Assignment is also available as section-command. By using assignment expressions, you can define symbols and assign value to symbols in SECTIONS command.

3.4.12.3.2.2 Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
  symbol1 = 0x1000; /* assign symbol1 with 0x1000 */
}
```

3.4.12.3.3 PROVIDE keyword

3.4.12.3.3.1 Description

“3.4.6. PROVIDE keyword” is also available as section-command. By using PROVIDE keyword, you can perform conditional symbol definitions in SECTIONS command.

3.4.12.3.3.2 Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
  /* define symbol not_defined only if it is not defined and then assign with 0x1234 */
  PROVIDE(not_defined = 0x1234);
}
```

3.4.12.3.4 SIZEOF function

3.4.12.3.4.1 Description

“3.4.8. SIZEOF function” is also available as section-command. By using SIZEOF function, you can retrieve the size of the output section in SECTIONS command.

3.4.12.3.4.2 Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
  symbol1 = SIZEOF(.text); /* assign symbol1 with the size of the .text section */
}
```

3.4.12.3.5 ADDR function

3.4.12.3.5.1 Description

“3.4.9. ADDR function” is also available as section-command. By using ADDR function, you can retrieve the virtual memory address (VMA) of a particular output section in SECTIONS command.

3.4.12.3.5.2 Examples of Use

```
SECTIONS
{
  .text : {*(.text)}
   symbol1 = ADDR(.text); /* assign symbol1 with the VMA of the .text section */
}
```

3.4.12.3.6 LOADADDR function

3.4.12.3.6.1 Description

“3.4.10. LOADADDR function” is also available as section-command. By using LOADADDR function, you can retrieve the load memory address (LMA) of a particular output section in SECTIONS command.

3.4.12.3.6.2 Examples of Use

```
SECTONS
{
  .text : {*(.text)}
   symbol1 = LOADADDR(.text); /* assign symbol1 with the LMA of the .text section */
}
```

3.4.12.3.7 ALIGN function

3.4.12.3.7.1 Description

“3.4.11. ALIGN function” is also available as section-command. By using ALIGN function, you can obtain the memory address of a particular alignment in SECTIONS command.

3.4.12.3.7.2 Examples of Use

```
SECTIONS
{
```

```
  .text : {*(.text)}
  symbol1 = ALIGN(0x1000); /* assign symbol1 with the nearest 0x1000 boundary address in front of the location counter */
}
```

3.4.12.4. output-section-command

The output-section-command is a command that can be used within “3.4.12.3.1 Description of the output section”.

The output-section-command can be one of the following:

- Description of the input section
- Assignment expression
- PROVIDE keyword, SIZEOF function, ADDR function, LOADADDR Function, ALIGN function, KEEP keyword

Each item of output-section-command is described below.

3.4.12.4.1 Description of the input section

3.4.12.4.1.1 Description

Retrieves the specified section from the specified input file and places it in the output section.

3.4.12.4.1.2 Command syntax

The syntax of the input section description is as follows.

```
Input file name(section name)
```

3.4.12.4.1.3 Wildcards

The wildcard character "*" can be used for the input file name or section name. Wildcard characters "*" represent all strings.

3.4.12.4.1.4 Examples of Use

```
SECTIONS
{
  .text : {
    file1.o(.text) /* Place the .text section in input file "file1.o" here */
  }

  .data : {
    *(.data) /* place the .data section here in all input files */
    *(.data.*) /* place all subsections of .data in all input files here */
   }
}
```

3.4.12.4.2 Assignment expression

3.4.12.4.2.1 Description

“3.4.4.1. Assignment expression” is also available as output-section-command. By using assignment expressions, you can define symbols and assign value to symbols in SECTIONS command.

3.4.12.4.2.2 Examples of Use

```
SECTIONS
{
  .text : {
    symbol1 = 0x1000; /* assign symbol1 with 0x1000 */
  }
}
```

3.4.12.4.3 PROVIDE keyword

3.4.12.4.3.1 Description

“3.4.6. PROVIDE keyword” is also available as output-section-command. By using PROVIDE keyword, you can perform conditional symbol definitions in SECTIONS command.

3.4.12.4.3.2 Examples of Use

```
SECTIONS
{
  .text : {
     *(.text)
      /* define symbol not_defined only if not_defined is not defined and assign 0x1234 */
      PROVIDE(not_defined = 0x1234);
   }
}
```

3.4.12.4.4 SIZEOF function

3.4.12.4.4.1 Description

“3.4.8. SIZEOF function” is also available as output-section-command. By using SIZEOF function, you can retrieve the size of the output section in SECTIONS command.

3.4.12.4.4.2 Examples of Use

```
SECTIONS
{
  .text : {
    *(.text)
    symbol1 = SIZEOF(.text); /* assign symbol1 with the size of the .text section */
  }
}
```

3.4.12.4.5 ADDR function

3.4.12.4.5.1 Description

“3.4.9. ADDR function” is also available as output-section-command. By using ADDR function, you can retrieve the virtual memory address (VMA) of a particular output section in SECTIONS command.

3.4.12.4.5.2 Examples of Use

```
SECTIONS
{
  .text : {
    *(.text)
    symbol1 = ADDR(.text); /* assign symbol1 with the start address of VMA in the .text section */
  }
}
```

3.4.12.4.6 LOADADDR Function

3.4.12.4.6.1 Description

“3.4.10. LOADADDR function” is also available as output-section-command. By using LOADADDR function, you can retrieve the load memory address (LMA) of a particular output section in SECTIONS command.

3.4.12.4.6.2 Examples of Use

```
SECTONS
{
  .text : {
    *(.text)
    symbol1 = LOADADDR(.text); /* assign symbol1 with the start address of LMA in the .text section */
  }
}
```

3.4.12.4.1 ALIGN function

3.4.12.4.1.1 Description

“3.4.11. ALIGN function” is also available as output-section-command. By using ALIGN function, you can obtain the memory address of a particular alignment in SECTIONS command.

3.4.12.4.1.2 Examples of Use

```
SECTIONS
{
  .text : {
    *(.text)
    symbol1 = ALIGN(0x1000); /* assign symbol1 with the nearest 0x1000 boundary address in front of the location counter */
  }
}
```

3.4.12.4.2 KEEP keyword

3.4.12.4.2.1 Description

The KEEP keyword is used to ensure that specified sections are not deleted by the garbage collection. Even if “3.3.13. --gc-sections” is used to remove unused sections by linkers (mtld), the sections specified by the KEEP keyword can still be retained.

3.4.12.4.2.2 Syntax

```
KEEP(input file name (input section name))
```

Note: You can use “3.4.12.4.1.3 Wildcards” for KEEP keyword arguments.

3.4.12.4.2.3 Examples of Use

```
/* Run using the --gc-sections command line option */
SECTIONS
{
  .text : {
    *(.text)
    KEEP(*(.text.keep)) /* protect the .text.keep section of all input files from the garbage collection*/
  }
}
```

3.4.12.5. SECTIONS command example

The following is an example of the SECTIONS command using the section-commands and output-section-commands described above.

```
MEMORY
{
  ROM (rx) : ORIGIN = 0x0000, LENGTH = 4K
  RAM (rwx) : ORIGIN = 0x1000, LENGTH = 4K
}

SECTIONS
{
  .text : AT(ADDR(.text) + SIZEOF(.data)) ALIGN(4) SUBALIGN(8) {
    *(.text)
    KEEP(*(.init))
    _text_end = .;
  } >ROM =0xFF

  .data : AT(LOADADDR(.text) + SIZEOF(.text)) ALIGN(8) {
    _data_start = .;
    *(.data)
    . = ALIGN(0x10); /* Align the location counter to the 0x10 boundary */
    _data_end = .;
  } >RAM
```

```
  .bss : {
    _bss_start = .;
    *(.bss)
    _bss_end = .;
  } >RAM

  .user_stack (NOLOAD) : {
    . += 0x100;
    .user_stack_start = .;
  } >RAM

  .rodata : {
    . = ALIGN(0x20); /* Align the location counter to the 0x20 boundary */
    *(.rodata)
    SUBALIGN(4) /* Alignment of subsequent input sections to a 4-byte boundary */
    *(.rodata.*)
    _rodata_end = . ;
  } >ROM AT>RAM

  _data_loadaddr = LOADADDR(.data); /* gets the load address in the .data section */
  _bss_size = SIZEOF(.bss); /* gets the size of the .bss section */
  _rodata_vma = ADDR(.rodata); /* gets the virtual address of the .rodata section */
}
```

3.4.13. MEMORY command

3.4.13.1. Description

The MEMORY command is used to describe the location and size of the memory region. The MEMORY command enables you to specify which memory region of the target system is available and which memory region is unavailable. In addition, you can assign a section to the memory region defined by this command. For detailed information about sections, see “3.4.12. SECTIONS command”.

Once a memory region is defined, you can use “3.4.12.3.1.7>region” attribute to instruct the linker (mtld) to place a particular output section in that memory region. For example, if you have a memory region named "mem", you can specify “> mem” in the description of the output section in the SECTIONS command.

If an address is not specified in the output section, the linker (mtld) places the output section at the next available address in the memory regoin described in the MEMORY command. If the sum of the output sections for a memory region is too large for that memory region, the linker (mtld) displays an error message and stops processing the link.

NOTE: The MEMORY command can contain up to one description in the linker script.

3.4.13.2. Command syntax

The syntax of the MEMORY command is as follows. You can write zero or more memory regions in parentheses.

```
MEMORY
{
  name [(attr)] : ORIGIN = origin, LENGTH = len
  ...
}
```

3.4.13.2.1 name

The name to refer to the memory region described with the MEMORY command from the rest of the linker script.

The name is meaningful only in the linker script and does not conflict with the symbol name, file name, or section name. The name of the memory region must be unique.

3.4.13.2.2 attr

This is a variable-length string that specifies the attributes of the memory region.

The attr string is an optional list of attributes that specifies whether to use a specific memory region for an input section that is not explicitly mapped in the linker script. As described in the SECTIONS command, if you do not specify an output section for the input section, the linker (mtld) automatically creates an output section with the same name as the input section. When the attributes of a memory region are defined, the linker (mtld) uses those attributes to select the memory region of the output section to be created.

The available attributes and their descriptions are as follows. Characters in an attribute have the same meaning both in upper and lower case.

Attribute	Description of the attribute
R	Read-only section
W	Readable and writable sections
X	Executable section

3.4.13.2.3 ORIGIN

This expression indicates the start address of the memory region. The expression is evaluated before the memory allocation process takes place and the result must be constant value.

3.4.13.2.4 LENGTH

This expression represents the size of the memory region in bytes. The expression is evaluated before the memory allocation

process takes place and the result must be constant value.

3.4.13.3. Examples of Use

```
MEMORY
{
  rom (RX) : ORIGIN = 0x0, LENGTH = 4K
  ram (WX) : ORIGIN = 0x1000, LENGTH = 1M
}

SECTIONS
{
  .text : {
     *(.text)
   } >rom

  .data : {
       *(.data)
   } >ram
}
```

4. No assurance function

This linker (mtld) is based on open source software. The features of the open source software are also diverted to mtld. This means that features not described in this document are available in mtld. These functions are called "no assurance function".
“No assurance function”’s has not been validated and are not assured to work correct. Use these functions on the user's own responsibility.

4.1. Command line option

The following attachment lists the command-line options available for the open source software on which mtld is based. Of the features described in this attachment, all that are not described in this document are “No assurance functions”, and those are not assured to work correct. Use these features on the user's own responsibility.

4.2. linker script

The following attachment lists the linker script features available for open-source software on which mtld is based. Of the features described in this attachment, all that are not described in this document are “No assurance functions”, and those are not assured to work correct. Use these features on the user's own responsibility.

5. Open Source Software

The linker (mtld) includes, in addition to the software in which ROHM is entitled or licensed, open source software (hereinafter referred to as the "Open Source Software Program") provided under the following licensing terms.
In the event of any conflict between the license terms of the open source software program and this material, the license terms of the open source software program will prevail, since the respective license terms will apply to the open source software program.

Software contained and its licensing terms

・libc (Apache License Version 2.0)
・binutils (GPL Version 2)
・crt (GPL Version 3)

6. For trademarks

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

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