tinyMicon MatisseCORE™

mtld ユーザーズガイド

tinyMicon MatisseCORE™用リンカ mtld ユーザーズガイド

© 2025 ROHM Co., Ltd.

No. 68UG028J Rev.003
2025.3

更新履歴

日付	バージョン	内容
2024/06/05	Rev.001	mtld V1.01.00 の内容を記述
2025/02/21	Rev.002	3.4.12.3.1.3(type) の記述を追加
2025/03/13	Rev.003	mtld V2.00.00 と V3.00.00 に対応

© 2025 ROHM Co., Ltd.

目次

1. 概要 5
1.1. 特徴 5
2. 動作環境 6
2.1. 必要なシステム 6
2.2. インストール 6
3. 使用方法 7
3.1. mtld の起動 7
3.2. コマンドラインオプション一覧 7
3.3. コマンドラインオプション詳細 7
3.3.1. -o 7
3.3.2. --program-size= 8
3.3.3. --data-size= 8
3.3.4. --ex-program-size= 8
3.3.5. -L 9
3.3.6. -l 9
3.3.7. -defsym 9
3.3.8. --script 10
3.3.9. -Map= 10
3.3.10. --exclude-mul= 10
3.3.11. --crt 11
3.3.12. --libc 11
3.3.13. --gc-sections 12
3.3.14. -v 12
3.3.15. -h 12
3.4. リンカスクリプト 13
3.4.1. リンカスクリプト概要 13
3.4.2. リンカスクリプトフォーマット 13
3.4.3. コメント 14
3.4.4. トップレベルコマンド 14
3.4.5. ENTRY コマンド 18
3.4.6. PROVIDE キーワード 18
3.4.7. ASSERT コマンド 18
3.4.8. SIZEOF 関数 19
3.4.9. ADDR 関数 19
3.4.10. LOADADDR 関数 20
3.4.11. ALIGN 関数 20
3.4.12. SECTIONS コマンド 22
3.4.13. MEMORY コマンド 31
4. 無保証機能 33
4.1. コマンドラインオプション 33
4.2. リンカスクリプト 33
5. オープンソースソフトウェアについて 34
6. 商標について 34

© 2025 ROHM Co., Ltd.

© 2025 ROHM Co., Ltd.

1. 概要

mtld は ROHM が開発した LSI (Large Scale Integration) 組み込み専用の超小型 8-bitCPU (Central Processing Unit) 、tinyMicon MatisseCORE™向けのリンカです。
リンカはコンパイラが出力したオブジェクトファイルをリンクして一つのファイルにまとめるプログラムです。

1.1. 特徴

mtld は以下の特徴を持っています。

1 ELF 32-bit LSB に対応。
2 オブジェクトファイル(*.o)とスタティックライブラリファイル(*.a)のリンクが可能。
3 様々なコマンドラインオプション。

© 2025 ROHM Co., Ltd.

2. 動作環境

mtld の動作環境を以下に説明します。

2.1. 必要なシステム

Table 1. システム要件

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

2.2. インストール

Matisse 開発環境のインストーラ(matisse-dev-env-installer-XX.XX.XX.exe)を実行すると mtld もインストールされます。
デフォルト設定ではツール一式は"C:¥Program Files¥ROHM¥Matisse¥"以下にインストールされます。

© 2025 ROHM Co., Ltd.

3. 使用方法

3.1. mtld の起動

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

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

mtld を実行するコマンドラインの具体例は以下のようになります。

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

3.2. コマンドラインオプション一覧

リンカ(mtld)で使用可能なコマンドラインオプションの一覧は以下になります。

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

3.3. コマンドラインオプション詳細

3.3.1. -o

3.3.1.1. 説明

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

```
-o filename
```

3.3.1.2. 使用例

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

© 2025 ROHM Co., Ltd.

3.3.2. --program-size=

3.3.2.1. 説明

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

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

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

3.3.2.2. 使用例

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

3.3.3. --data-size=

3.3.3.1. 説明

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

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

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

3.3.3.2. 使用例

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

3.3.4. --ex-program-size=

3.3.4.1. 説明

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

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

3.3.4.2. 使用例

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

© 2025 ROHM Co., Ltd.

3.3.5. -L

3.3.5.1. 説明

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

```
-Ldirname
```

3.3.5.2. 使用例

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

3.3.6. -l

3.3.6.1. 説明

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

```
-llibname
```

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

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

-l オプションのパラメータにはライブラリファイル名から"lib"と".a"を取り除いた文字列を指定します

3.3.6.2. 使用例

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

3.3.7. -defsym

3.3.7.1. 説明

--defsym オプションは以下のフォーマットでシンボルを定義します。このオプションは複数指定可能です。
定義されたシンボルはリンカスクリプト内で有効になります。

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

3.3.7.2. 使用例

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

© 2025 ROHM Co., Ltd.

3.3.8. --script

3.3.8.1. 説明

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

```
--script filename
```

3.3.8.2. 使用例

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

3.3.9. -Map=

3.3.9.1. 説明

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

```
-Map=filename
```

3.3.9.2. 使用例

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

3.3.10. --exclude-mul=

3.3.10.1. 説明

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

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

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

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

3.3.10.2. 使用例

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

© 2025 ROHM Co., Ltd.

3.3.10.3. 詳細

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

3.3.11. --crt

3.3.11.1. 説明

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

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

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

3.3.11.2. 使用例

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

3.3.12. --libc

3.3.12.1. 説明

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

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

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

3.3.12.2. 使用例

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

© 2025 ROHM Co., Ltd.

3.3.13. --gc-sections

3.3.13.1. 説明

--gc-sections オプションはリンカ(mtld)のガベージコレクション機能を有効にします。このオプションを使用すると、未使用のセクションがリンク時に自動的に削除されます。これにより、最終的な実行可能ファイルやライブラリのサイズを小さくすることができます。

注意: 3.4.12.4.2KEEP で指定されたセクションはガベージコレクションの対象になりません。

3.3.13.2. 使用例

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

3.3.14. -v

3.3.14.1. 説明

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

3.3.14.2. 使用例

```
> mtld -v
```

3.3.15. -h

3.3.15.1. 説明

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

3.3.15.2. 使用例

```
> mtld -h
```

© 2025 ROHM Co., Ltd.

3.4. リンカスクリプト

リンカスクリプトはリンク処理時のリンカ(mtld)のふるまいを定義するためのスクリプトファイルです。リンカスクリプトを記述することでプログラムのメモリレイアウトの詳細を指定することができます。
リンカ(mtld)で使用可能なリンカスクリプトの機能をこれ以降の章で説明します。
リンカ(mtld)実行時にはリンカスクリプトはデフォルトでは"C:¥Program Files¥ROHM¥Matisse¥lib¥ldscripts¥matisse.x"が使用されます。
コマンドラインオプションの--script を使用することで適用するリンカスクリプトを変更することができます。

3.4.1. リンカスクリプト概要

リンカ(mtld)はリンカスクリプトの記述に沿って入力ファイルを 1 つの出力ファイルに結合します。各入力ファイルと出力ファイルは ELF と呼ばれるデータフォーマットになっています。入出力ファイルはオブジェクトファイルと呼ばれます。各オブジェクトファイルは複数のセクションを持っています。入力ファイルのセクションは入力セクションと呼ばれ、出力ファイルのセクションは出力セクションと呼ばれます。

3.4.1.1. セクション

オブジェクトファイルの各セクションは、名前とサイズとコンテンツ(内容)を持っています。リンカ(mtld)を実行すると入力セクションのコンテンツがリンカスクリプトで指定されたメモリ領域に配置され、出力ファイルが作成されます。

3.4.1.2. シンボル

各オブジェクトファイルはシンボルを持ちます。各シンボルには名前とアドレスがあります。C 言語プログラムをコンパイルしてオブジェクトファイルに変換すると、プログラム内の関数やグローバル変数、スタティック変数を示すシンボルが作成されます。これらのシンボルはリンカスクリプト内のコマンドや式で使用できます。さらに、リンカスクリプト内での新たなシンボルの定義や、定義済みシンボルの値の変更も可能です。

3.4.1.3. LMA と VMA

出力セクションは 2 つのアドレスを持っています。1 つ目は LMA(ロードメモリアドレス)で、セクションがロードされるアドレスです。2 つ目は VMA(仮想メモリアドレス)で、出力ファイルが実行された時にそのセクションが持つアドレスです。ほとんどの場合、これらの 2 つのアドレスは同じになります。これらの 2 つのアドレスが異なる例としては、データセクションが ROM にロードされ、プログラム起動時に RAM にコピーされる場合があります（この手法は、組み込みシステムでグローバル変数を初期化するときによく使われます）。この場合データセクションの LMA は ROM 領域内のアドレスとなり、VMA は RAM 領域のアドレスとなるようにリンカスクリプトを記述します。

3.4.2. リンカスクリプトフォーマット

リンカ(mtld)が対応しているリンカスクリプトの BNF 表記は以下の通りです。

リンカスクリプト := コメント*|トップレベルコマンド*| MEMORY コマンド?|SECTIONS コマンド? トップレベルコマンド := 代入式|ENTRY コマンド|PROVIDE キーワード|ASSERT コマンド|SIZEOF 関数|ADDR 関数|LOADADDR 関数

© 2025 ROHM Co., Ltd.

3.4.3. コメント

3.4.3.1. 説明

リンカスクリプト内にはコメントを記述できます。/*と*/で囲まれた部分がコメントとなります。コメントの内容はリンカスクリプトの動作に一切影響を及ぼしません。

3.4.3.2. 使用例

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

/* multiple
lines
of
comments.
*/
```

3.4.4. トップレベルコマンド

リンカスクリプト内に記述可能なトップレベルコマンドを以下に説明します。
ただし、3.4.4.1. 代入式、3.4.6. PROVIDE キーワード、3.4.8. SIZEOF 関数、3.4.9. ADDR 関数、3.4.10. LOADADDR 関数は 3.4.12. SECTIONS コマンド内でも使用可能です。

3.4.4.1. 代入式

3.4.4.1.1 構文

代入式の BNF 表記は以下の通りです。

```
代入式 := シンボル名=式;
```

代入式が評価されると左辺のシンボルに右辺の式の評価結果が代入されます。
左辺のシンボルが既に定義済みシンボルの場合はシンボルの値が右辺値で更新され、未定義のシンボルだった場合は新たにシンボルが定義されてから右辺値が代入されます。

3.4.4.1.1.1 使用例

```
symbol1 = 10; /* 新しいシンボル"symbol1"が定義され、10 が代入される */
symbol1 = 20; /* "symbol1"の値が 20 に更新される */
```

3.4.4.2. 式

式には即値、演算式、比較式、があります。各項目を以下に説明します。

3.4.4.2.1 即値

© 2025 ROHM Co., Ltd.

3.4.4.2.1.1 構文

即値の BNF 表記は以下の通りです。即値は整数のみ使用できます。

```
即値 := 10 進数文字+接尾辞?|0x16 進数文字+接尾辞?
10 進数文字 := 0|1|2|3|4|5|6|7|8|9
接尾辞 := K|M
16 進数字 := 0|1|2|3|4|5|6|7|8|9|a|b|c|d|e|f
```

10 進数や 16 進数に接尾辞の K をつけると値が 1024 倍され、接尾辞の M をつけると値が 1048576 倍されます。
使用可能な即値の範囲は 0 から 4294967295(0xFFFFFFFF)までです。4294967295 より大きい即値は評価時に 4294967295 に置き換えられます。
即値に負の数を使用しないでください。リンカスクリプトに負の数を記述したとしても、評価時には符号なし 32bit 整数として扱われるため、演算式や比較式で意図しない結果を招きます。

即値は単体では使用できず、代入式の一部か関数の引数として使用する必要があります。

3.4.4.2.1.2 使用例

```
symbol1 = 10;  /* symbol1 を定義し、即値 10 を代入 */
symbol2 = 5K; /* symbol2 を定義し、 5 * 1024 の結果を代入 */
symbol3 = 10M; /* symbol3 を定義し、10 * 1048576 の結果を代入 */
symbol4 = 0x1234abcd;  /* symbol4 を定義し、16 進数即値 0x1234abcd を代入 */
```

3.4.4.2.2 演算式

3.4.4.2.2.1 構文

演算式の BNF 表記は以下の通りです。

```
演算式 := 式 演算子 式
演算子 := +|-|*
```

演算式が評価される前に演算式の右辺が評価されます。
+演算子を使用すると演算式の評価時に加算処理が実行され、-演算子を使用すると減算処理が実行され、*演算子を使用すると乗算処理が実行されます。

有効な演算結果の範囲は 0 から 4294967295(0xFFFFFFFF)までです。結果がこの範囲外となる演算の評価結果は下 32bit の値のみ採用されます。

演算式は単体では使用できず、代入式の一部か関数の引数として使用する必要があります。

3.4.4.2.2.2 使用例

```
symbol1 = 10 + 2; /* symbol1 が定義され、12 が代入される */
symbol2 = symbol1 - 5; /* symbol2 が定義され、7 が代入される */
symbol3 = 0xFFFFFFFF; /* symbol3 が定義され、0xFFFFFFFF が代入される */
symbol4 = symbol1 + symbol3; /* symbol4 が定義され、11 が代入される (0x10000000B の下 32bit)*/
symbol5 = symbol1 - symbol3; /* symbol5 が定義され、13 が代入される (0x10000000D の下 32bit)*/
```

© 2025 ROHM Co., Ltd.

```
symbol6 = 0x80000000 * 2; /* symbol6 が定義され、0 が代入される (0x100000000 の下 64bit) */
```

3.4.4.2.3 比較式

3.4.4.2.3.1 構文

比較式の BNF 表記は以下の通りです。

```
比較式 := 式 比較演算子 式
比較演算子 := <|<=|>|>=|==|!=
```

比較式が評価される前に比較演算子の左右の項が評価されます。
比較式を評価すると比較演算子のルールに従って左右の項が比較され、比較結果が真の場合は比較式の評価結果は 1 となり、比較結果が偽の場合は比較式の評価結果は 0 となります。

比較式は単体では使用できず、代入式の一部か関数の引数として使用する必要があります。

各比較演算子の説明は以下です。

比較演算子	説明
<	例えば"a < b "と記述した時、a が b より小さい場合に比較結果が真となり、それ以外の場合は偽となる。
<=	例えば"a <= b" と記述した時、a が b 以下の場合に比較結果が真となり、それ以外の場合は偽となる。
>	例えば"a > b" と記述した時、a が b より大きい場合に比較結果が真となり、それ以外の場合は偽となる。
>=	例えば"a >= b" と記述した時、a が b 以上の場合に比較結果が真となり、それ以外の場合は偽となる。
==	例えば"a == b" と記述した時、a と b が等しい場合に比較結果が真となり、それ以外の場合は偽となる。
!=	例えば"a != b" と記述した時、a が b 等しくない場合に比較結果が真となり、それ以外の場合は偽となる。

3.4.4.2.3.2 使用例

```
symbol1 = 10;
symbol2 = 20;
symbol3 = symbol1 < symbol2; /* 比較結果が真なので 1 が symbol3 に代入される */
symbol4 = symbol1 == symbol2; /* 比較結果が偽なので 0 が symbol4 に代入される */
```

3.4.4.2.4 .変数(ドット変数)

3.4.4.2.4.1 説明

.変数(ドット変数)は次に配置されるデータやセクションのアドレスを表す位置カウンタの操作に使用する特殊な変数です。位置カウンタは SECTIONS コマンドの開始時に初期化され、その後、出力セクションにデータが追加されるたびに更新されます。
.変数が代入式の左辺にある場合、代入式を評価すると位置カウンタの値は代入式の右辺値に更新されます。
.変数が式の内部で使用される場合、.変数を評価すると現在の位置カウンタの値が取得できます。

3.4.4.2.4.2 構文

```
. = xxxx;  /* 代入式の左辺として使用 */
xxxx = .; /* 代入式の右辺として使用 */
```

© 2025 ROHM Co., Ltd.

3.4.4.2.4.3 使用例

```
SECTIONS
{
  .text : {
    . = 0x1000;       /* 位置カウンタを 0x1000 に設定 */
    symbol1 = .;      /* 位置カウンタの値を symbol1 に代入 */
    . = . + 0x200;    /* 位置カウンタに 0x200 を加算 */
  }
}
```

© 2025 ROHM Co., Ltd.

3.4.5. ENTRY コマンド

3.4.5.1. 説明

ENTRY コマンドはリンカスクリプトでプログラムのエントリーポイントを設定するために使用されます。エントリーポイントは、プログラムが実行を開始する最初の命令のアドレスです。ENTRY コマンドは、リンカスクリプト内で特定のシンボル名を指定することで、そのシンボルがエントリーポイントであることを示します。

ENTRY コマンドが使用されなかった場合は、.text セクションの先頭アドレスがエントリーポイントとなります。ENTRY コマンドが使用されず.text セクションも存在しない場合はアドレス 0 番地がエントリーポイントとなります。

3.4.5.2. コマンド構文

```
ENTRY(シンボル名)
```

3.4.5.3. 使用例

```
entry_point = 0x0000; /* シンボル entry_point を定義し、0x0000 を代入 */
ENTRY(entry_point) /* シンボル entry_point をエントリーポイントに設定 */
```

注意: リンカスクリプト内に複数の ENTRY コマンドを記述してもエラーにはなりませんが、最後に記述した ENTRY コマンドのみが有効になります。

3.4.6. PROVIDE キーワード

3.4.6.1. 説明

PROVIDE キーワードはリンカスクリプト内で条件付きシンボル定義を行うために使用されます。PROVIDE キーワードは指定されたシンボルが入力オブジェクトファイルまたはリンカスクリプトで既に定義されていない場合にのみ、シンボルを定義します。

3.4.6.2. 構文

```
PROVIDE(シンボル名 = 式);
```

3.4.6.3. 使用例

```
PROVIDE(not_defined = 0x1234); /* not_defined が定義されていない場合のみ not_defined を定義し、0x1234 を代入 */
```

3.4.7. ASSERT コマンド

3.4.7.1. 説明

ASSERT コマンドは指定した条件が満たされない場合にエラーメッセージを表示し、リンク処理を停止させるために使用されます。ASSERT コマンドを使用する事で、リンク時に特定の条件が満たされている事が確認できます。

© 2025 ROHM Co., Ltd.

3.4.7.2. コマンド構文

ASSERT コマンドの構文は以下の通りです。

```
ASSERT(即値|シンボル名|演算式|比較式, エラーメッセージ);
```

3.4.7.3. 使用例

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

/* rom_size と ram_size の合計値が total_memory_size を超えた場合にエラーとなる */
ASSERT(rom_size + ram_size <= total_memory_size, "Memory size over capacity.");
```

3.4.8. SIZEOF 関数

3.4.8.1. 説明

SIZEOF 関数はリンカスクリプト内で出力セクションのサイズを取得するために使用します。SIZEOF 関数の評価結果は指定した出力セクションの byte 単位のサイズになります。

3.4.8.2. 関数構文

```
SIZEOF(出力セクション名)
```

3.4.8.3. 使用例

```
SECTIONS
{
  .text : { *(.text) }
}
symbol1 = SIZEOF(.text); /* symbol1 に.text セクションのサイズを代入 */
```

3.4.9. ADDR 関数

3.4.9.1. 説明

ADDR 関数はリンカスクリプト内で特定の出力セクションの先頭の仮想メモリアドレス(VMA)を取得するために使用されます。ADDR 関数の評価結果は指定した出力セクションの先頭の仮想メモリアドレス(VMA)になります。

3.4.9.2. 関数構文

```
ADDR(出力セクション名)
```

© 2025 ROHM Co., Ltd.

3.4.9.3. 使用例

```
SECTIONS
{
  .text : { *(.text) }
}
symbol1 = ADDR(.text); /* symbol1 に.text セクションの開始アドレスを代入 */
```

3.4.10. LOADADDR 関数

3.4.10.1. 説明

LOADADDR 関数はリンカスクリプト内で特定の出力セクションの先頭のロードメモリアドレス(LMA)を取得するために使用されます。LOADADDR 関数の評価結果は指定した出力セクションの先頭のロードメモリアドレス(LMA)になります。

3.4.10.2. 関数構文

```
LOADADDR(出力セクション名)
```

3.4.10.3. 使用例

```
SECTONS
{
  .text : { *(.text) }
}
symbol1 = LOADADDR(.text); /* symbol1 に.text セクションのロードアドレスを代入 */
```

3.4.11. ALIGN 関数

3.4.11.1. 説明

ALIGN 関数はリンカスクリプト内で特定のアライメントのメモリアドレスを取得するために使用されます。アライメントとは、メモリアドレスが特定の境界を指すことを意味します。ALIGN 関数の評価結果は現在の位置カウンタの前方の最も近いメモリ境界のアドレス値となります。現在の位置カウンタが既に指定したメモリ境界にある場合は ALIGN 関数の評価結果は現在の位置カウンタとなります。
例えば、現在の位置カウンタが 0x1010 の場合は ALIGN(0x1000)の評価結果は 0x2000 になり、現在の位置カウンタが 0x2000 の場合は ALIGN(0x1000)の評価結果は 0x2000 になります。

3.4.11.2. コマンド構文

```
ALIGN(メモリ境界)
```

注意: ALIGN 関数で指定するメモリ境界は 2 のべき乗を示す即値である必要があります。

3.4.11.3. 使用例

```
SECTIONS
{
```

© 2025 ROHM Co., Ltd.

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

symbol1 = ALIGN(0x1000); /* symbol1 に位置カウンタの前方の最も近い 0x1000 境界アドレスを代入 */
```

© 2025 ROHM Co., Ltd.

3.4.12. SECTIONS コマンド

3.4.12.1. 説明

SECTIONS コマンドはリンカスクリプト内で入力ファイルに含まれるセクションを出力ファイルのセクションにどのようにマップするかを定義し、そして出力セクションをターゲットシステムのメモリにどのように配置するかも定義します。

注意: SECTIONS コマンドはリンカスクリプト内で最大一つしか記述できません。

3.4.12.2. コマンド構文

SECTIONS コマンドの構文は以下の通りです。括弧の中に 0 個以上の sections-command を記述できます。

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

3.4.12.3. section-command

section-command は以下のいずれかになります。

- 出力セクションの記述
- 代入式
- PROVIDE キーワード, SIZEOF 関数, ADDR 関数, LOADADDR 関数, ALIGN 関数

section-command の各項目について以下に説明します。

3.4.12.3.1 出力セクションの記述

出力セクションの記述のフォーマットは以下の通りです。出力セクションの各属性を記述していきます。[]内は省略可能です。

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

注意: output-section-command に関しては 3.4.12.4. output-section-command を参照してください。
出力セクションの属性を以下に説明します。

3.4.12.3.1.1 section 名

出力セクションの名前です。

3.4.12.3.1.2 address

出力セクションの開始アドレスを指定します。指定される開始アドレスは仮想メモリアドレス(VMA)です。

© 2025 ROHM Co., Ltd.

注意: 3.4.12.3.1.7>region と同時に使用しないでください。VMA に関しては 3.4.1.3. LMA と VMA を参照してください。

3.4.12.3.1.3 (type)

出力セクションのタイプを指定します。以下のタイプが指定できます。

タイプ名	タイプの説明
NOLOAD	指定されたセクションはメモリ内のアドレスのみ確保され、出力ファイルに含まれなくなる。

3.4.12.3.1.4 AT(lma)

出力セクションのロードメモリアドレス(LMA)を指定します。
注意: 3.4.12.3.1.8AT>region と同時に使用しないでください。LMA に関しては 3.4.1.3. LMA と VMA を参照してください。

3.4.12.3.1.5 ALIGN(boundary)

出力セクションの開始アドレスを特定のアライメントに整列させます。整列される開始アドレスは仮想メモリアドレス(VMA)です。
アライメントとは、メモリ内のアドレスが特定の境界(2 のべき乗)に配置されることを意味します。
注意: 3.4.11. ALIGN 関数を使用する事で位置カウンタやシンボルのアライメントを指定することもできます。

3.4.12.3.1.6 SUBALIGN(boundary)

出力セクション内の入力セクションの開始アドレスを特定のアライメントに整列させます。整列される開始アドレスは仮想メモリアドレス(VMA)です。
アライメントとは、メモリ内のアドレスが特定の境界(2 のべき乗)に配置されることを意味します。

3.4.12.3.1.7 >region

出力セクションの仮想メモリアドレス(VMA)を割り当てるメモリ領域を指定します。
注意: 3.4.12.3.1.2address と同時に使用しないでください。VMA に関しては 3.4.1.3. LMA と VMA を参照してください。メモリ領域は 3.4.13. MEMORY コマンドで記述します。

3.4.12.3.1.8 AT>region

出力セクションのロードメモリアドレス(LMA)を割り当てるメモリ領域を指定します。
注意: 3.4.12.3.1.4AT(lma)と同時に使用しないでください。LMA に関しては 3.4.1.3. LMA と VMA を参照してください。メモリ領域は 3.4.13. MEMORY コマンドで記述します。

3.4.12.3.1.9 =fillexp

指定した式の評価結果がフィルパターンとなります。フィルパターンとは出力セクション内のデータが配置されない領域を埋めるバイナリデータの事です。
使用可能なフィルパターンのサイズは 1byte から 4byte までです。

3.4.12.3.2 代入式

3.4.12.3.2.1 説明

© 2025 ROHM Co., Ltd.

3.4.4.1. 代入式は section-command としても使用可能です。代入式を使用する事で SECTIONS コマンド内でもシンボルの定義やシンボルへの代入が可能です。

3.4.12.3.2.2 使用例

```
SECTIONS
{
  .text : { *(.text) }
  symbol1 = 0x1000;       /* symbol1 に 0x1000 を代入 */
}
```

3.4.12.3.3 PROVIDE キーワード

3.4.12.3.3.1 説明

3.4.6. PROVIDE キーワードは section-command としても使用可能です。PROVIDE キーワードを使用する事で SECTIONS コマンド内でも条件付きシンボル定義が可能です。

3.4.12.3.3.2 使用例

```
SECTIONS
{
  .text : { *(.text) }
  /* not_defined が定義されていない場合のみ not_defined を定義し、0x1234 を代入 */
  PROVIDE(not_defined = 0x1234);
}
```

3.4.12.3.4 SIZEOF 関数

3.4.12.3.4.1 説明

3.4.8. SIZEOF 関数は section-command としても使用可能です。SIZEOF 関数を使用する事で SECTIONS コマンド内でも出力セクションのサイズ取得が可能です。

3.4.12.3.4.2 使用例

```
SECTIONS
{
  .text : { *(.text) }
  symbol1 = SIZEOF(.text); /* symbol1 に.text セクションのサイズを代入 */
}
```

3.4.12.3.5 ADDR 関数

3.4.12.3.5.1 説明

© 2025 ROHM Co., Ltd.

3.4.9. ADDR 関数は section-command としても使用可能です。ADDR 関数を使用する事で SECTIONS コマンド内でも特定の出力セクションの先頭の仮想メモリアドレス(VMA)の取得が可能です。

3.4.12.3.5.2 使用例

```
SECTIONS
{
  .text : { *(.text) }
   symbol1 = ADDR(.text); /* symbol1 に.text セクションの VMA を代入 */
}
```

3.4.12.3.6 LOADADDR 関数

3.4.12.3.6.1 説明

3.4.10. LOADADDR 関数は section-command としても使用可能です。LOADADDR 関数を使用する事で SECTIONS コマンド内でも特定の出力セクションの先頭のロードメモリアドレス(LMA)の取得が可能です。

3.4.12.3.6.2 使用例

```
SECTONS
{
  .text : { *(.text) }
   symbol1 = LOADADDR(.text); /* symbol1 に.text セクションの LMA を代入 */
}
```

3.4.12.3.7 ALIGN 関数

3.4.12.3.7.1 説明

3.4.11. ALIGN 関数は section-command としても使用可能です。ALIGN 関数を使用する事で SECTIONS コマンド内でも指定したアライメントのアドレスの取得が可能です。

3.4.12.3.7.2 使用例

```
SECTIONS
{
  .text : { *(.text) }
  symbol1 = ALIGN(0x1000); /* symbol1 に位置カウンタの前方の最も近い 0x1000 境界アドレスを代入 */
}
```

3.4.12.4. output-section-command

output-section-command は 3.4.12.3.1 出力セクションの記述の内部に記述するコマンドです。
output-section-command は以下のいずれかになります。

© 2025 ROHM Co., Ltd.

- 入力セクションの記述
- 代入式
- PROVIDE キーワード, SIZEOF 関数, ADDR 関数, LOADADDR 関数, ALIGN 関数, KEEP キーワード

output-section-command の各項目について以下に説明します。

3.4.12.4.1 入力セクションの記述

3.4.12.4.1.1 説明

指定した入力ファイルから指定したセクションを取り出し、出力セクションに配置します。

3.4.12.4.1.2 コマンド構文

入力セクションの記述の構文は以下の通りです。

```
入力ファイル名(セクション名)
```

3.4.12.4.1.3 ワイルドカード

入力ファイル名やセクション名にはワイルドカード文字"*"が使用可能です。ワイルドカード文字"*"は全ての文字列を表します。

3.4.12.4.1.4 使用例

```
SECTIONS
{
  .text : {
    file1.o(.text) /* 入力ファイル"file1.o"内の.text セクションをここに配置 */
  }

  .data : {
    *(.data) /* 全ての入力ファイル内の.data セクションをここに配置 */
    *(.data.*) /* 全ての入力ファイル内の.data の全てのサブセクションをここに配置 */
   }
}
```

3.4.12.4.2 代入式

3.4.12.4.2.1 説明

3.4.4.1. 代入式は output-section-command としても使用可能です。代入式を使用する事で SECTIONS コマンド内でもシンボルの定義やシンボルへの代入が可能です。

3.4.12.4.2.2 使用例

```
SECTIONS
{
```

© 2025 ROHM Co., Ltd.

```
  .text : {
    symbol1 = 0x1000;       /* symbol1 に 0x1000 を代入 */
  }
}
```

3.4.12.4.3 PROVIDE キーワード

3.4.12.4.3.1 説明

3.4.6. PROVIDE キーワードは output-section-command としても使用可能です。PROVIDE キーワードを使用する事で SECTIONS コマンド内でも条件付きシンボル定義が可能です。

3.4.12.4.3.2 使用例

```
SECTIONS
{
  .text : {
      *(.text)
      /* not_defined が定義されていない場合のみ not_defined を定義し、0x1234 を代入 */
      PROVIDE(not_defined = 0x1234);
   }
}
```

3.4.12.4.4 SIZEOF 関数

3.4.12.4.4.1 説明

3.4.8. SIZEOF 関数は output-section-command としても使用可能です。SIZEOF 関数を使用する事で SECTIONS コマンド内でも出力セクションのサイズ取得が可能です。

3.4.12.4.4.2 使用例

```
SECTIONS
{
  .text : {
    *(.text)
    symbol1 = SIZEOF(.text); /* symbol1 に.text セクションのサイズを代入 */
  }
}
```

3.4.12.4.5 ADDR 関数

3.4.12.4.5.1 説明

3.4.9. ADDR 関数は output-section-command としても使用可能です。ADDR 関数を使用する事で SECTIONS コマンド内でも特定の出力セクションの先頭の仮想メモリアドレス(VMA)の取得が可能です。

© 2025 ROHM Co., Ltd.

3.4.12.4.5.2 使用例

```
SECTIONS
{
  .text : {
    *(.text)
    symbol1 = ADDR(.text); /* symbol1 に.text セクションの VMA を代入 */
  }
}
```

3.4.12.4.6 LOADADDR 関数

3.4.12.4.6.1 説明

3.4.10. LOADADDR 関数は output-section-command としても使用可能です。LOADADDR 関数を使用する事で SECTIONS コマンド内でも特定の出力セクションの先頭のロードメモリアドレス(LMA)の取得が可能です。

3.4.12.4.6.2 使用例

```
SECTONS
{
  .text : {
    *(.text)
    symbol1 = LOADADDR(.text); /* symbol1 に.text セクションの LMA を代入 */
  }
}
```

3.4.12.4.1 ALIGN 関数

3.4.12.4.1.1 説明

3.4.11. ALIGN 関数は output-section-command としても使用可能です。ALIGN 関数を使用する事で SECTIONS コマンド内でも指定したアライメントのアドレスの取得が可能です。

3.4.12.4.1.2 使用例

```
SECTIONS
{
  .text : {
    *(.text)
    symbol1 = ALIGN(0x1000); /* symbol1 に位置カウンタの前方の最も近い 0x1000 境界アドレスを代入 */
  }
}
```

© 2025 ROHM Co., Ltd.

3.4.12.4.2 KEEP キーワード

3.4.12.4.2.1 説明

KEEP キーワードはセクションがガベージコレクションによって削除されないようにするために使用されます。3.3.13. --gc-sections を使用してリンカ(mtld)が未使用セクションを削除する場合でも、KEEP キーワードで指定されたセクションは保持することができます。

3.4.12.4.2.2 構文

```
KEEP(入力ファイル名(入力セクション名))
```

注意: KEEP キーワードの引数には 3.4.12.4.1.3 ワイルドカードが使用できます。

3.4.12.4.2.3 使用例

```
/* リンカ(mtld)のコマンドラインオプション--gc-sections を使用して実行 */
SECTIONS
{
  .text : {
    *(.text)
    KEEP(*(.text.keep))  /* 全ての入力ファイルの.text.keep セクションをガベージコレクションの対象外にする */
  }
}
```

3.4.12.5. SECTIONS コマンド使用例

これまで説明した section-command や outputu-section-command を使用した SECTIONS コマンドの例は以下の通りです。

```
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); /* 位置カウンタを 0x10 の境界に整列 */
    _data_end = .;
  } >RAM
```

© 2025 ROHM Co., Ltd.

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

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

  .rodata : {
    . = ALIGN(0x20); /* 位置カウンタを 0x20 の境界に整列 */
    *(.rodata)
    SUBALIGN(4) /* これ以降配置される入力セクションを 4 バイトの境界に整列 */
    *(.rodata.*)
    _rodata_end = .;
  } >ROM AT>RAM

  _data_loadaddr = LOADADDR(.data); /* .data セクションのロードアドレスを取得 */
  _bss_size = SIZEOF(.bss); /* .bss セクションのサイズを取得 */
  _rodata_vma = ADDR(.rodata); /* .rodata セクションの仮想アドレスを取得 */
}
```

© 2025 ROHM Co., Ltd.

3.4.13. MEMORY コマンド

3.4.13.1. 説明

MEMORY コマンドはメモリ領域の位置とサイズを記述するために使用します。MEMORY コマンドでターゲットシステムのどのメモリ領域が使用可能で、どのメモリ領域が使用できないかを指定できます。さらに、セクションをこのコマンドで記述したメモリ領域に割り当てることもできます(詳細は 3.4.12. SECTIONS コマンド参照)。
一度メモリ領域を定義すると、3.4.12. SECTIONS コマンドの出力セクション属性 3.4.12.3.1.7>region を使うことで、特定の出力セクションをそのメモリ領域に置くようにリンカ(mtld)に指示することができます。例えば、`mem' という名前のメモリ領域がある場合は、SECTIONS コマンドの出力セクションの定義で `>mem' と指定できます。
出力セクションにアドレスが指定されなかった場合、リンカ(mtld)は MEMORY コマンドで記述されたメモリ領域内の次に利用可能なアドレスに出力セクションを配置します。あるメモリ領域に対する出力セクションの合計がそのメモリ領域に対して大きすぎる場合、リンカ(mtld)はエラーメッセージを表示しリンク処理を中止します。

注意: MEMORY コマンドはリンカスクリプト内で最大一つしか記述できません。

3.4.13.2. コマンド構文

MEMORY コマンドの構文は以下の通りです。括弧内に 0 個以上のメモリ領域を記述できます。

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

3.4.13.2.1 name

リンカスクリプト内の他の部分から MEMORY コマンドで記述したメモリ領域を参照するための名前です。
この名前はリンカスクリプト内でのみ意味を持ち、シンボル名やファイル名やセクション名と競合しません。メモリ領域の名前は一意である必要があります。

3.4.13.2.2 attr

メモリ領域の属性を指定する可変長の文字列です。
リンカスクリプトで明示的にマッピングされていない入力セクションに、特定のメモリ領域を使用するかどうかを指定する属性のオプションリストです。SECTIONS コマンドで説明したように、入力セクションに対して出力セクションを指定しない場合、リンカ(mtld)は入力セクションと同じ名前の出力セクションを自動的に作成します。メモリ領域の属性が定義されていると、リンカ(mtld)はその属性を使用して、作成する出力セクションのメモリ領域を選択します。
使用可能な属性とその説明は以下です。属性の文字は大文字でも小文字でも同じ意味となります。

属性	属性の説明
R	読み取り専用セクション
W	読み書き可能セクション
X	実行可能セクション

3.4.13.2.3 ORIGIN

© 2025 ROHM Co., Ltd.

メモリ領域の開始アドレスを示す式です。この式はメモリ割り当て処理が行われる前に評価され、結果が定数になる必要があります。

3.4.13.2.4 LENGTH

メモリ領域のサイズをバイト単位で表す式です。この式はメモリ割り当て処理が行われる前に評価され、結果が定数になる必要があります。

3.4.13.3. 使用例

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

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

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

© 2025 ROHM Co., Ltd.

4. 無保証機能

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

4.1. コマンドラインオプション

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



4.2. リンカスクリプト

mtld のベースになっているオープンソースソフトウェアのリンカで利用可能なリンカスクリプトの機能一覧は以下の添付ファイルに記載しています。この添付ファイルに記載されている機能の中で、本ドキュメントで説明されていないものは全て無保証機能となり、正常に動作するかは保証しません。利用者の自己判断に基づいて使用してください。



© 2025 ROHM Co., Ltd.

5. オープンソースソフトウェアについて

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

含まれているソフトウェアとそのライセンス条件
・libc(Apache License Version 2.0)
・binutils(GPL Version 2)
・crt(GPL Version 3)

6. 商標について

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

© 2025 ROHM Co., Ltd.

ご注意

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

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

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

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

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

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

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

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

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

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

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

ROHM Customer Support System

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

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