



μC3/Compact ユーザーズガイド

プロセッサ依存部 Cortex-M33 編

Rev. 1.4

はじめに

μC3 (マイクロ・シー・キューブ) とは、社団法人トロン協会がオープンリアルタイムカーネルとして策定した μ ITRON4.0 仕様に準拠したカーネルをコアに持った RTOS (リアルタイム・オペレーティング・システム) です。

μ C3 の C3 とは、**Compact (省メモリ)**、**Connectivity (接続性)**、**Capability (性能)** の 3 つのコンセプトを表しています。また、キューブの名称は、これらによる三乗の効果をも生み出す可能性を表しています。

本書の位置づけ

本書は、 μ C3/Compact のプロセッサ依存の Cortex-M33 編とし、 μ C3/Compact のカーネル機能の共通マニュアルを補足します。

TRON は、"The Real-time Operation system Nucleus"の略称です。

μ ITRON は、"Micro Industrial TRON"の略称です。

μ ITRON4.0 仕様の仕様書は、トロン協会のホームページ (<http://www.assoc.tron.org/>) から入手することができます。

μ C3 は、イー・フォース株式会社の登録商標です。

本書で記載されている内容は、予告無く変更される場合があります。

改訂履歴

Rev.	発行日	改訂内容	
		ページ	内容
1.0	2020.01.09	—	First Release
1.1	2021.09.21	12	GCC 版カーネルライブラリの説明を追加
		39-51	ロードモジュールの生成に、e2studio の説明を追加
1.2	2023.02.03	10	CSTACK, HSTACK のレジスタ待避に使われるバイト数を修正
1.3	2024.01.24	53-60	ロードモジュールの生成に、MDK-ARM の説明を追加
1.4	2026.02.03	11	1.4.1 の説明を周期ハンドラからタイムイベントハンドラに修正

目次

はじめに.....	2
改訂履歴.....	3
目次.....	4
第 1 章 プロセッサに依存する状態.....	6
1. 1 Armv8-M Security Extension	6
1. 1. 1 Stack Limit	6
1. 1. 2 カーネルの実行ステート.....	6
1. 1. 3 AIRCR.PRIS	7
1. 2 カーネルの管理外の割込み.....	7
1. 3 割込みレベル.....	8
1. 3. 1 割込みマスク.....	10
1. 4 システムスタック.....	11
1. 4. 1 アイドル・タイムイベントハンドラ用スタック (CSTACK)	11
1. 4. 2 ハンドラ用スタック (HSTACK)	11
1. 5 CPU ロック状態	12
1. 6 ディスパッチ保留状態.....	12
1. 7 カーネルライブラリ.....	13
1. 7. 1 GCC 版カーネルライブラリ	13
第 2 章 プロセッサに依存する機能.....	14
2. 1 データ型.....	14
2. 2 割込みの禁止／許可.....	15
2. 3 全割込みの禁止／許可.....	18
2. 4 コンテキスト ID の取得.....	21
2. 5 低消費電力.....	23
第 3 章 ロードモジュールの生成.....	24
3. 1 IAR Embedded Workbench for Arm.....	24
3. 1. 1 新規プロジェクトの生成.....	24
3. 1. 2 オプションの設定.....	26
3. 1. 3 ファイルの登録.....	38
3. 1. 4 メイク.....	39
3. 2 Renesas e2studio	40
3. 2. 1 新規プロジェクトの生成.....	40
3. 2. 2 オプションの設定.....	45
3. 2. 3 ファイルの登録.....	51

3. 2. 4	ビルド	52
3. 3	MDK-ARM	52
3. 3. 1	新規プロジェクトの生成	53
3. 3. 2	オプションの設定	55
3. 3. 3	ファイルの登録	58
3. 3. 4	ビルド	60

第1章 プロセッサに依存する状態

μ C3/Compact において Cortex-M33 プロセッサに依存するカーネルの状態について説明します。

1. 1 Armv8-M Security Extension

Armv8-M から新たに追加された Security Extension 機能への対応¹を説明します。

1. 1. 1 Stack Limit

カーネルでは以下に示す、レジスタの操作は行いませんので、スタックがオーバーフローしても例外は発生しません。

Secure Main Stack	MSPLIM_S
Secure Process Stack	PSPLIM_S
Non-secure Main Stack	MSPLIM_NS
Non-secure Process Stack	PSPLIM_NS

1. 1. 2 カーネルの実行ステート

- カーネルはセキュア・非セキュアの何れかのステートでのみ実行可能です。両ステートで同時に実行することは出来ません。
- ドメインをまたがったシステムコール呼び出しは出来ません。
 カーネルを**セキュアステート**で実行時、**非セキュアステート**プログラムから μ C3 システムコールは呼び出し不可
 カーネルを**非セキュアステート**で実行時、**セキュアステート**プログラムから μ C3 システムコールは呼び出し不可
- 各カーネルリソース(タスク、セマフォ等)とカーネルライブラリは同一ドメインに配置する必要があります。例えば、カーネルライブラリをセキュア領域に配置し、タスクを非セキュア領域に配置するといった利用方法は出来ません。

¹ 2020 年 1 月 9 日現在

1. 1. 3 AIRCR.PRIS

このビットを有効にすると、非セキュア例外の優先レベルを下位半分マップされるようにすることができますが、カーネルではこのレベル操作を考慮しておりませんので、このビットを有効にしないでください。

(Armv8-M Architecture Reference Manual 説明抜粋)

AIRCR, Application Interrupt and Reset Control Register

PRIS, bit [14]

Prioritize Secure exceptions. The value of this bit defines whether Secure exception priority boosting is enabled.

0 Priority ranges of Secure and Non-secure exceptions are identical.

1 Non-secure exceptions are de-prioritized.

1. 2 カーネルの管理外の割込み

カーネルのオーバーヘッドに影響されない最速の応答性を持つ割込みとして、カーネルの管理外の割込みをサポートしています。カーネルが排他制御に用いる割込みレベルをカーネルレベルと呼び、これを下げることにより常に許可状態にある割込みレベルを設けて、これにカーネルの管理外の割込みを割り当てます。

このカーネルの管理外の割込みは最速の応答性を持つ代わりに、例外を除き、システムコールを呼び出すことはできません。

1. 3 割込みレベル

Cortex-M33 では、割込みレベルを 8 ビットで表現しており、プリエンプト優先レベルとサブ優先レベルを任意のビット幅で分割できます。また、ハード的に実装されている割込みレベルのビット数はデバイスに異なり、アプリケーションが使用できる割込みレベルとの関係を次の表で示します。

優先レベルグループとビット数の関係

優先レベル グループ	プリエンプト 優先レベルビット数	サブ優先 レベルビット数
0	7 ... Bit[7:1]	1 ... Bit[0]
1	6 ... Bit[7:2]	2 ... Bit[1:0]
2	5 ... Bit[7:3]	3 ... Bit[2:0]
3	4 ... Bit[7:4]	4 ... Bit[3:0]
4	3 ... Bit[7:5]	5 ... Bit[4:0]
5	2 ... Bit[7:6]	6 ... Bit[5:0]
6	1 ... Bit[7]	7 ... Bit[6:0]
7	0 ... None	8 ... Bit[7:0]

次にアプリケーションで使用可能なレベルとデバイスに実装された割込みレベルのビット数の関係を次の表で示します。

優先レベル グループ	デバイスに実装された割込みレベルのビット数/ アプリケーションで使用可能なレベル						
	2	3	4	5	6	7	8
0	—	—	—	—	—	0x02 刻み	0x01 刻み
1	—	—	—	—	0x04 刻み	0x02 刻み	0x01 刻み
2	—	—	—	0x08 刻み	0x04 刻み	0x02 刻み	0x01 刻み
3	—	—	0x10 刻み	0x08 刻み	0x04 刻み	0x02 刻み	0x01 刻み
4	—	0x20 刻み	0x10 刻み	0x08 刻み	0x04 刻み	0x02 刻み	0x01 刻み
5	0x40 刻み	0x20 刻み	0x10 刻み	0x08 刻み	0x04 刻み	0x02 刻み	0x01 刻み
6	0x40 刻み	0x20 刻み	0x10 刻み	0x08 刻み	0x04 刻み	0x02 刻み	0x01 刻み
7	—	—	—	—	—	—	—

※ — は使用不可

μ C3/Compact の実装では、1 ビット以上のプリエンプト優先レベルが必要とされ、デフォルトではプリエンプト優先レベルの 0 (最も高い優先レベル) はカーネルレベルとしてカーネルが占有します。このため、カーネルの管理外の割込みで使用可能な割込みレベルと、割込みサービスルーチンで使用可能な割込みレベルは次の関係になります。

0 ≦ カーネルの管理外の割込みで使用可能な割込みレベル ≦ カーネルレベル
カーネルレベル < 割込みサービスルーチンで使用可能な割込みレベル ≦ 255

カーネルレベルと同レベルのカーネルの管理外の割込みの割込みレベルを定義することはできますが、カーネルと排他的に使用するため、応答性に若干の影響が出ます。

1. 3. 1 割込みマスク

`chg_ims` はマスクするプリエンプト優先レベルを指定し、`get_ims` はマスクしているプリエンプト優先レベルを返します。例えば、割込みレベル `0x20~0xFF` をマスクする場合には、`0x20` を指定します。割込みマスクを元に戻す場合は、`0`（タスクレベル）を指定します。ただし、カーネルレベルと同じ、或いはカーネルレベルよりも高いレベルを設定することはできません。

1. 4 システムスタック

タスクのスタック領域以外にも、周期ハンドラ、割込みサービスルーチンなどで使用されるスタック領域が必要になります。 μ C3/Compact では、タスクのスタック以外のスタックを総称してシステムスタックと呼んでいます。Cortex-M33 用カーネルでは、2 種類のシステムスタックを使い分けます。

1. 4. 1 アイドル・タイムイベントハンドラ用スタック (CSTACK)

アイドルとタイムイベントハンドラで使用されるスタック領域です。必要とされるサイズは、各ハンドラで使われるスタックサイズの中で一番大きいサイズに割込み発生時にレジスタ待避に使われる 32 バイト (FPU 付きの場合は最大で 104 バイト) を加算したバイト数になります。

1. 4. 2 ハンドラ用スタック (HSTACK)

例外発生時に使用されるスタックで、カーネルの管理外の割込みを含め、割込みで使用されるスタック領域です。必要とされるサイズは、同一割込みレベルの割込みサービスルーチンの中で一番大きいサイズに割込み発生時にレジスタ待避に使われる 48 バイト (FPU 付きの場合は最大で 120 バイト) を加算し、さらに使用する割込みレベル分を加算したバイト数になります。

1. 5 CPU ロック状態

CPU ロック状態とは、全割込みサービスルーチンの割込みをマスクした状態とし、割込みマスクを操作しています。そのため、`loc_cpu`, `unl_cpu` による CPU ロック状態で、`chg_ims` による割込みマスクの操作はできません。

1. 6 ディスパッチ保留状態

非タスクコンテキストでは、常に、ディスパッチ保留状態です。一方、タスクレベルでは、以下のいずれかの条件に当てはまれば、ディスパッチ保留状態です。

- CPU ロック状態
- ディスパッチ禁止状態
- 割込みレベルをタスクレベルより高くしている

ここで、「割込みレベルをタスクレベルより高くしている」とは、`chg_ims` により割込みマスクを 0 より大きくしている状態です。

1. 7 カーネルライブラリ

カーネルライブラリは、実行ステートと FPU の有無の違いにより 4 種類が用意されています。

【IAR Systems Embedded Workbench for ARM】

TrustZone mode	FPU	カーネルライブラリ
Secure	無し	uC3cortexm33nl_s.a
Secure	VFPv5-SP(VFPv5-DP)	uC3cortexm33fl_s.a
Non-Secure	無し	uC3cortexm33nl_ns.a
Non-Secure	VFPv5-SP(VFPv5-DP)	uC3cortexm33fl_ns.a

【Arm Compiler Version 6】

TrustZone mode	FPU	カーネルライブラリ
Secure	無し	uC3cortexm33nl_s.lib
Secure	VFPv5-SP(VFPv5-DP)	uC3cortexm33fl_s.lib
Non-Secure	無し	uC3cortexm33nl_ns.lib
Non-Secure	VFPv5-SP(VFPv5-DP)	uC3cortexm33fl_ns.lib

【GCC (GNU Arm Embedded Toolchain)】

TrustZone mode	FPU	カーネルライブラリ
Secure	無し	libuC3cortexm33nl_s.a
Secure	VFPv5-SP(VFPv5-DP)	libuC3cortexm33fl_s.a
Non-Secure	無し	libuC3cortexm33nl_ns.a
Non-Secure	VFPv5-SP(VFPv5-DP)	libuC3cortexm33fl_ns.a

1. 7. 1 GCC 版カーネルライブラリ

FPU 対応カーネルライブラリをアプリケーションに組み込みリンクする場合は、アプリケーションのコンパイルオプションを次のように設定します。

Floating point hardware オプション：-mfpu=fpv5-sp-d16

Floating-point ABI オプション：-mfloat-abi=hard²

² 2021 年 9 月 17 日時点、GCC 系統合開発 IDE の Floating-point ABI オプションのデフォルト設定は hard が主流なため。カーネルライブラリも hard 設定でビルドして提供しています。

第2章 プロセッサに依存する機能

2. 1 データ型

Cortex-M33 依存の μ ITRON4.0 仕様で規定しているデータ型は次の通りです。

INT	符号付き 32 ビット整数
UINT	符号無し 32 ビット整数
VP_INT	データタイプが定まらないものへのポインタまたは符号付き 32 ビット整数
FLGPTN	符号無し 32 ビット整数

2. 2 割込みの禁止／許可

プロセッサ依存のシステムコールとして、割込み禁止／許可（dis_int,ena_int）、独自仕様の割込みレベルの設定（vset_ipl）を実装しています。

dis_int

割込みの禁止

【書式】

```
ER ercd = dis_int(INTNO intno);
```

【パラメータ】

INTNO	intno	割込みを禁止する割込み番号
-------	-------	---------------

【戻り値】

ER	ercd	正常終了（E_OK）またはエラーコード
----	------	---------------------

【エラーコード】

E_PAR	パラメータエラー（intno が不正）
-------	---------------------

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	可

【解説】

割込み番号 intno を持つ割込み要因の受け付けを禁止します。

割込み番号の定義は、「デバイス依存部マニュアル」を参照してください。

ena_int

割込みの許可

【書式】

```
ER ercd = ena_int(INTNO intno);
```

【パラメータ】

INTNO	intno	割込みを許可する割込み番号
-------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	パラメータエラー (intno が不正)
-------	----------------------

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	可

【解説】

割込み番号 intno を持つ割込み要因の受け付けを許可します。

割込み番号の定義は、「デバイス依存部マニュアル」を参照してください。

vset_ipl**割込みレベルの設定****【書式】**

```
ER ercd = vset_ipl(INTNO, IPL ipl);
```

【パラメータ】

INTNO	intno	割込みを許可する割込み番号
IPL	ipl	割込みレベル

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	パラメータエラー (intno, ipl が不正)
-------	---------------------------

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	可

【解説】

割込み番号 intno を持つ割込み要因に割込みレベル ipl を設定します。

割込み番号の定義は、「デバイス依存部マニュアル」を参照してください。また、割込みレベルの設定範囲は、コンフィグレーションにより異なり、詳しくは「1. 3 割込みレベル」を参照してください。

2. 3 全割込みの禁止／許可

独自仕様のカーネルのシステムコールとして、管理外の割込みを含めた全割込みの禁止/許可（`vloc_cpu`, `vunl_cpu`）を実装しています。このシステムコールは、割込み以外の例外やカーネルの管理外の割込みで呼び出すことができます。ただし、全割込みの禁止中に全割込みの許可（`vunl_cpu`）以外のシステムコールを呼び出すことはできません。

vloc_cpu

全割込みの禁止

【書式】

ER ercd = vloc_cpu(void);

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了 (E_OK) のみ
----	------	----------------

【エラーコード】

なし

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	可

【解説】

割込み以外の例外やカーネルの管理外の割込みでも呼び出すことができます。

vunl_cpu

全割込みの許可

【書式】

```
ER ercd = vunl_cpu(void);
```

【パラメータ】

なし

【戻り値】

ER	ercd	正常終了 (E_OK) のみ
----	------	----------------

【エラーコード】

なし

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	可

【解説】

割込み以外の例外やカーネルの管理外の割込みでも呼び出すことができます。

2. 4 コンテキスト ID の取得

独自仕様のカーネルのシステムコールとして、現在実行中のコンテキスト ID の取得 (`vget_ctx`) を実装しています。このシステムコールは、割込み以外の例外やカーネルの管理外の割込みで呼び出すことができます。ただし、これらから呼び出された場合は、カーネルが最後に認識したタスク、周期ハンドラ、割込みのコンテキスト ID を返します。つまり、異常系の例外処理から呼び出した場合は、それまでにどのコンテキストを実行していたのかを知ることができます。

vget_ctx

コンテキストの ID 取得

【書式】

```
UW ercd = vget_ctx(void);
```

【パラメータ】

なし

【戻り値】

UW	ercd	コンテキスト ID
コンテキスト ID の構造 (32 ビット長)		
Bit[31-24]	未使用(0)	
Bit[23-16]	コンテキスト属性	
	(0x00) タスク	Bit[15-0] タスク ID
	(0x10) アイドル	Bit[15-0] 0
	(0x20) 割込み	Bit[15-0] 例外番号
	(0x30) 初期化処理	Bit[15-0] 0
	(0x40) 周期ハンドラ	Bit[15-0] 周期ハンドラ ID

【エラーコード】

なし

【呼び出しコンテキスト】

タスク	可
タイムイベントハンドラ	可
割込みサービスルーチン	可

【解説】

割込み以外の例外やカーネルの管理外の割込みでも呼び出すことができます。

2. 5 低消費電力

Cortex-M33 は、電力管理の機能としてスリープ・モードを備えています。この機能を利用することにより消費電力を低減することができます。スリープ・モードにはスリープ、ディープ・スリープの二種類があり、 μ C3/Compact はその内のスリープをサポートしています。

「デバイス依存部マニュアル」の Cortex-M33 コンフィグレーションにおいて低消費電力モードを有効にすると、 μ C3/Compact は実行状態のタスクが無く、割込みが発生していない（アイドル状態）時、自動的にスリープに入ります。

【補足】

μ C3/Compact はユーザ定義のアイドル関数がある場合、それが実行された後にスリープに遷移します。そのため、アイドル関数内で永久ループがあればスリープには遷移しないのをご注意ください。

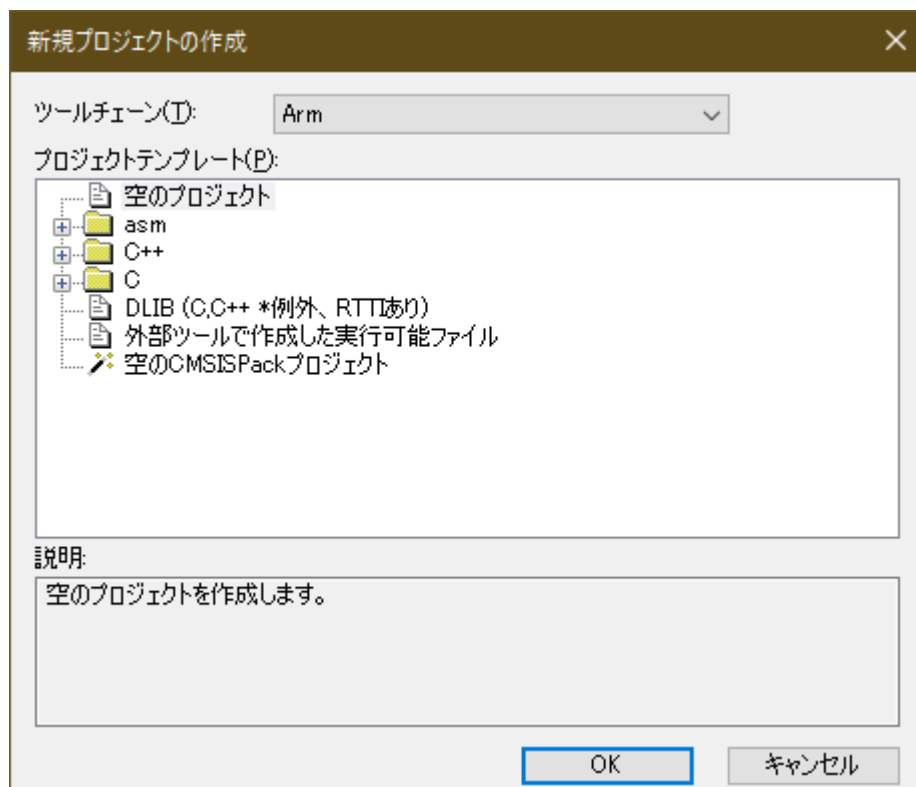
第3章 ロードモジュールの生成

コンフィグレータによりコードを生成し、スケルトンコードを元にして、アプリケーションコードを追加・編集したものとして、ロードモジュール生成までを説明します。

3. 1 IAR Embedded Workbench for Arm

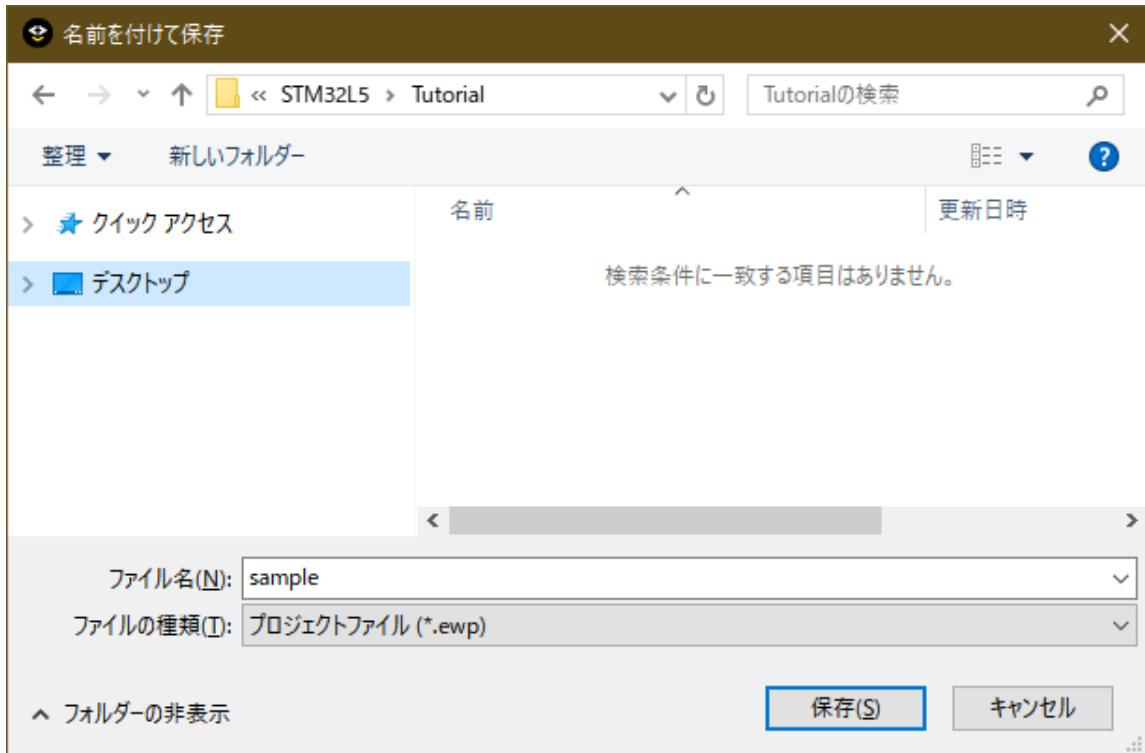
3. 1. 1 新規プロジェクトの生成

「IAR Embedded Workbench for Arm」³を起動します。最初に、「プロジェクト」→「新規プロジェクトの作成」によりプロジェクトを作成します。この時、「空のプロジェクト」を選択し、「OK」をクリックして空のプロジェクトを作成します。



³ IAR Embedded Workbench for ARM version 8.32.3 を使用して説明しています。

プロジェクトには、任意の名前を付け保存します。ここでは、コンフィグレータが生成したファイルのあるフォルダに、プロジェクト名を「sample」にし「保存」をクリックします。

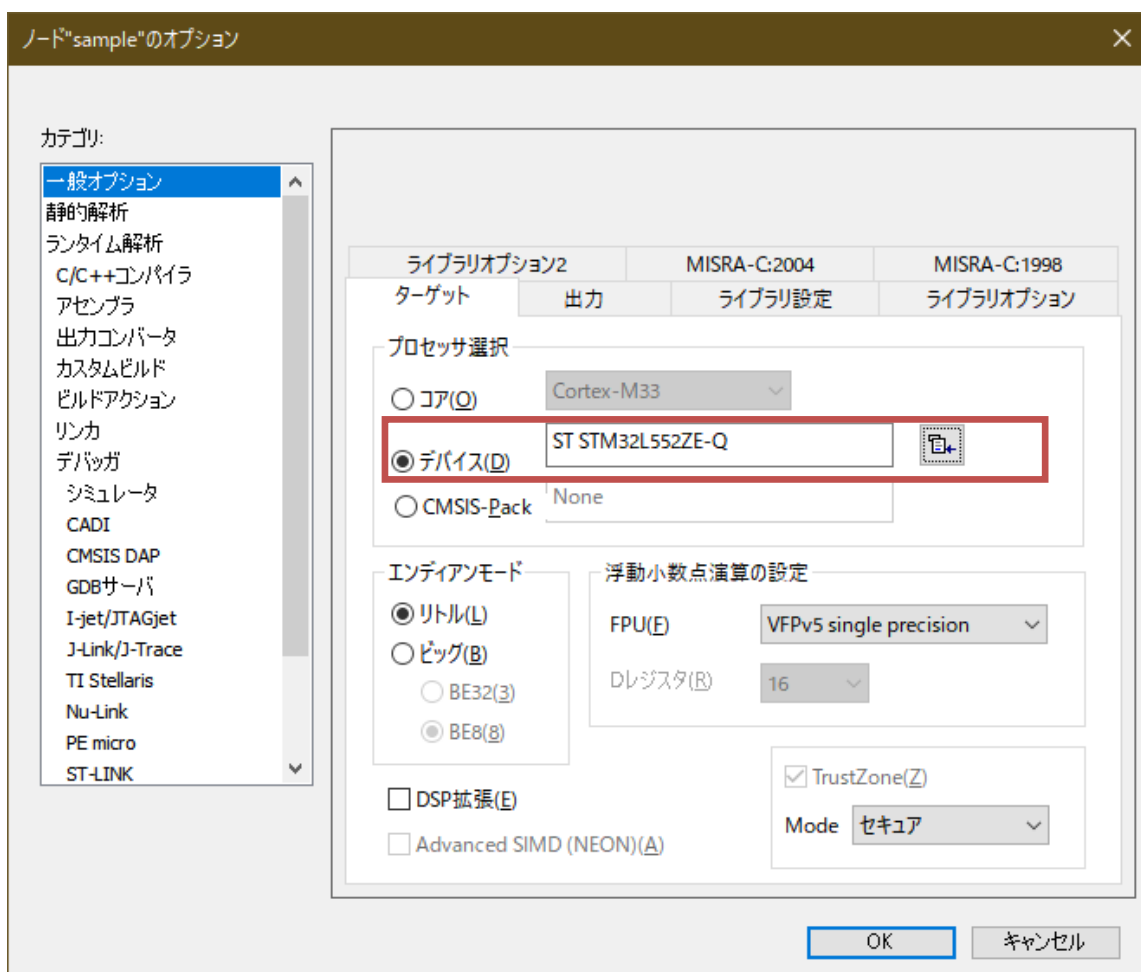


3. 1. 2 オプションの設定

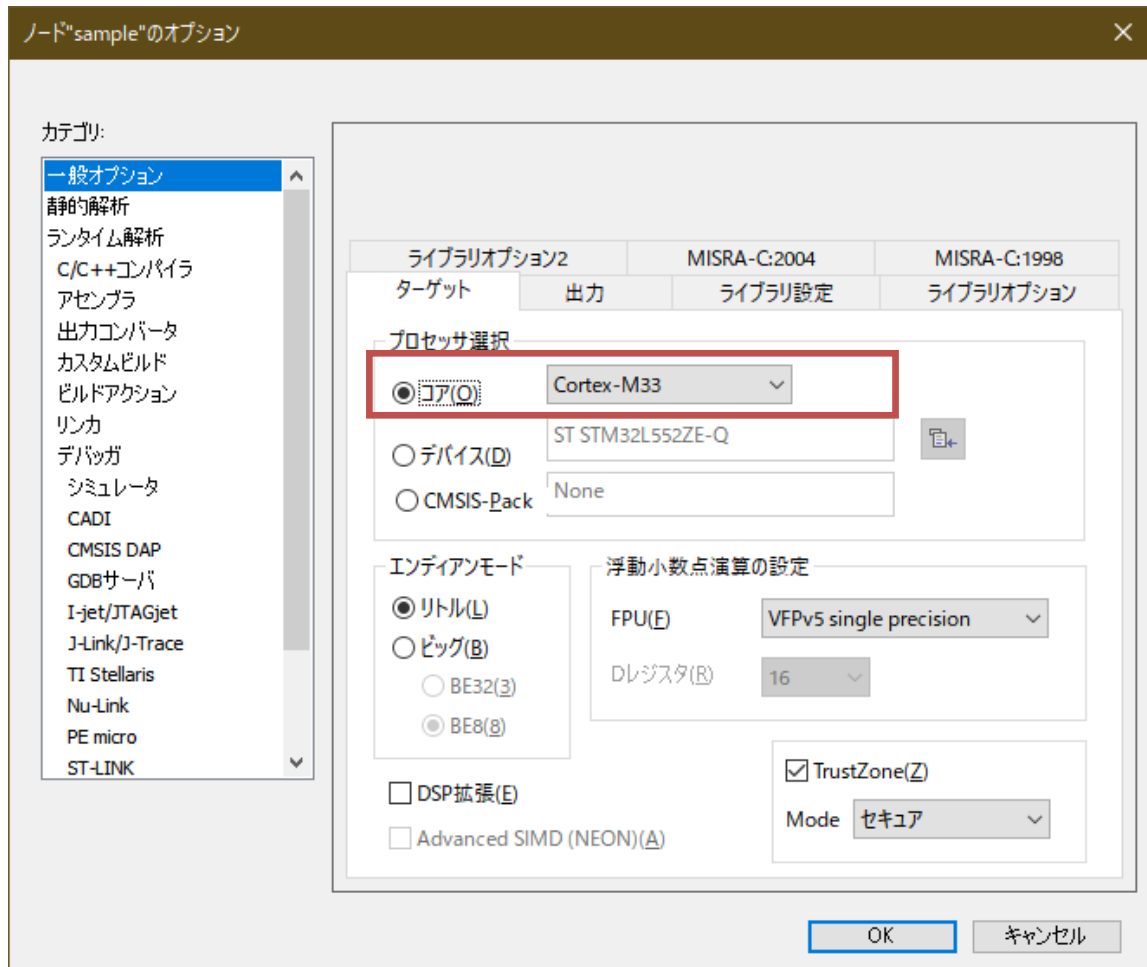
「プロジェクト」→「オプション」によりオプション画面を開きます。

A. 一般オプション

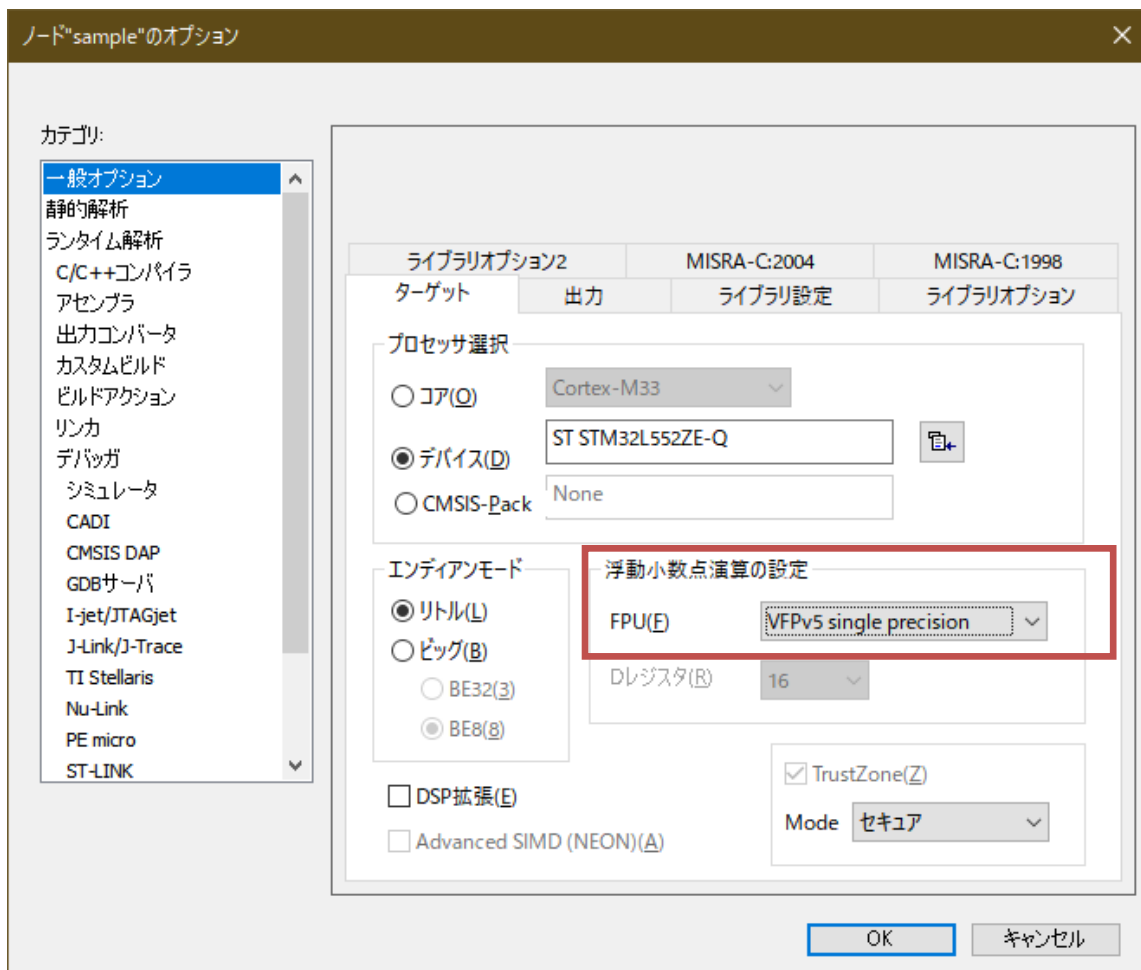
カテゴリ「一般オプション」の「ターゲット」タブでは、「デバイス」をクリックし、さらにその右にあるボタンをクリックしてデバイスリストを表示させます。該当するデバイスが見つければ、それを選択します。



該当するデバイスが見つからない場合は、「コア」をクリックし、「Cortex-M33」を選択します。

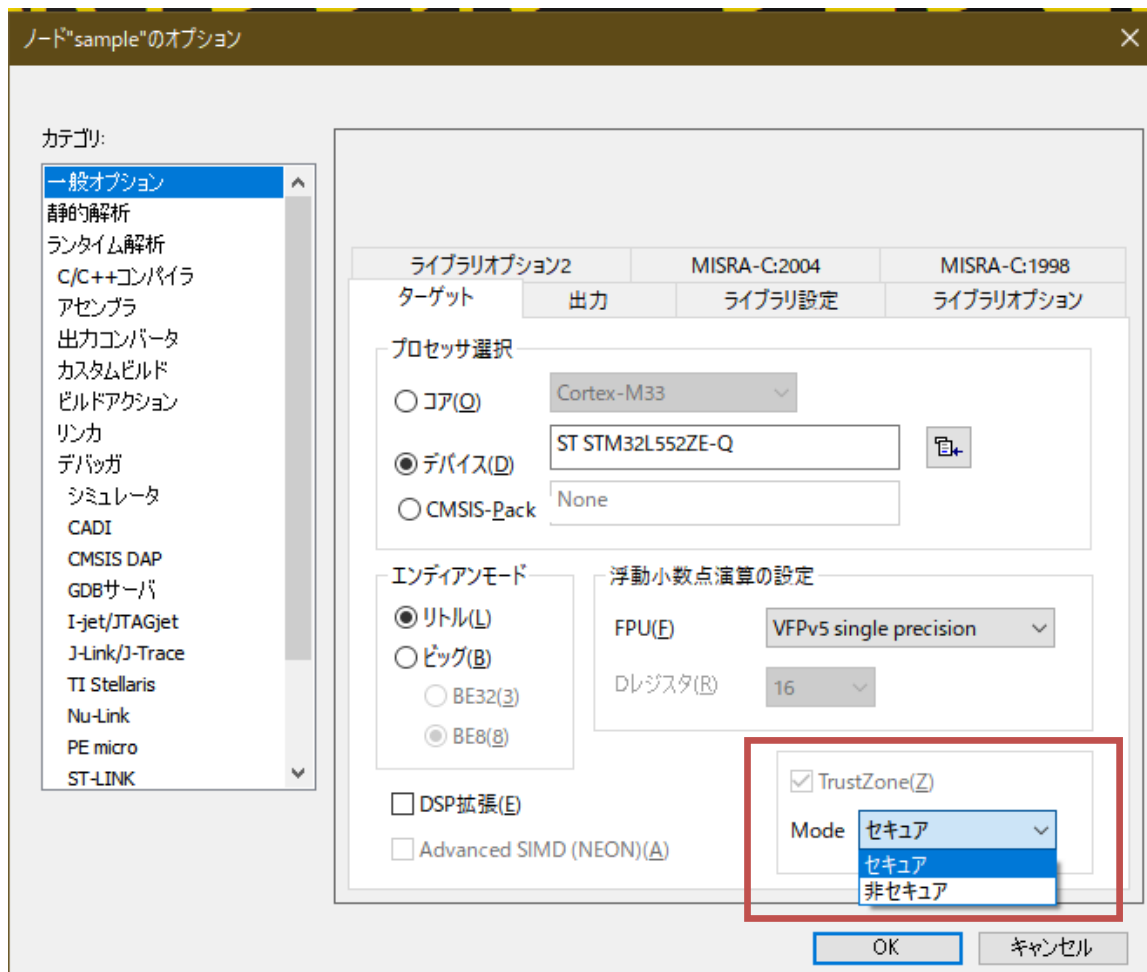


デバイスが FPU⁴をサポートしている場合は、プルダウンより FPU の使用有無を選択します。



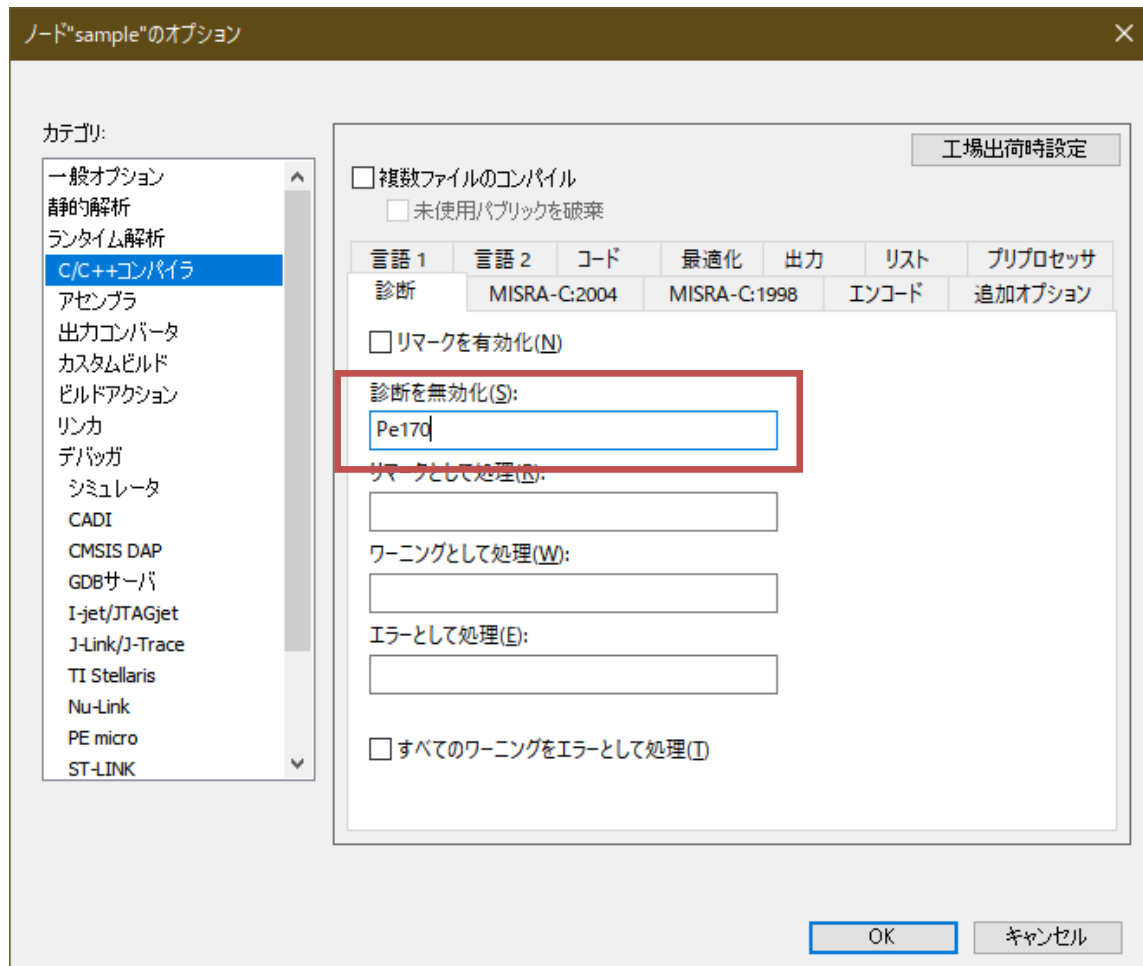
⁴ デバイスが VFPv5 double-precision をサポートしてないケースがありますので、デバイスのマニュアルを参照のうえ、選択してください。

デバイスが Security Extension をサポートしている場合は、実行ステートをセキュア・非セキュアの何れかを選択します。



B. C/C++ コンパイラ

μC3/Compact では、回避できないワーニングが発生します。これを抑止するため、カテゴリ「C/C++ コンパイラ」の「診断」タブでは、「診断を無効化」に「Pe170」と入力します。

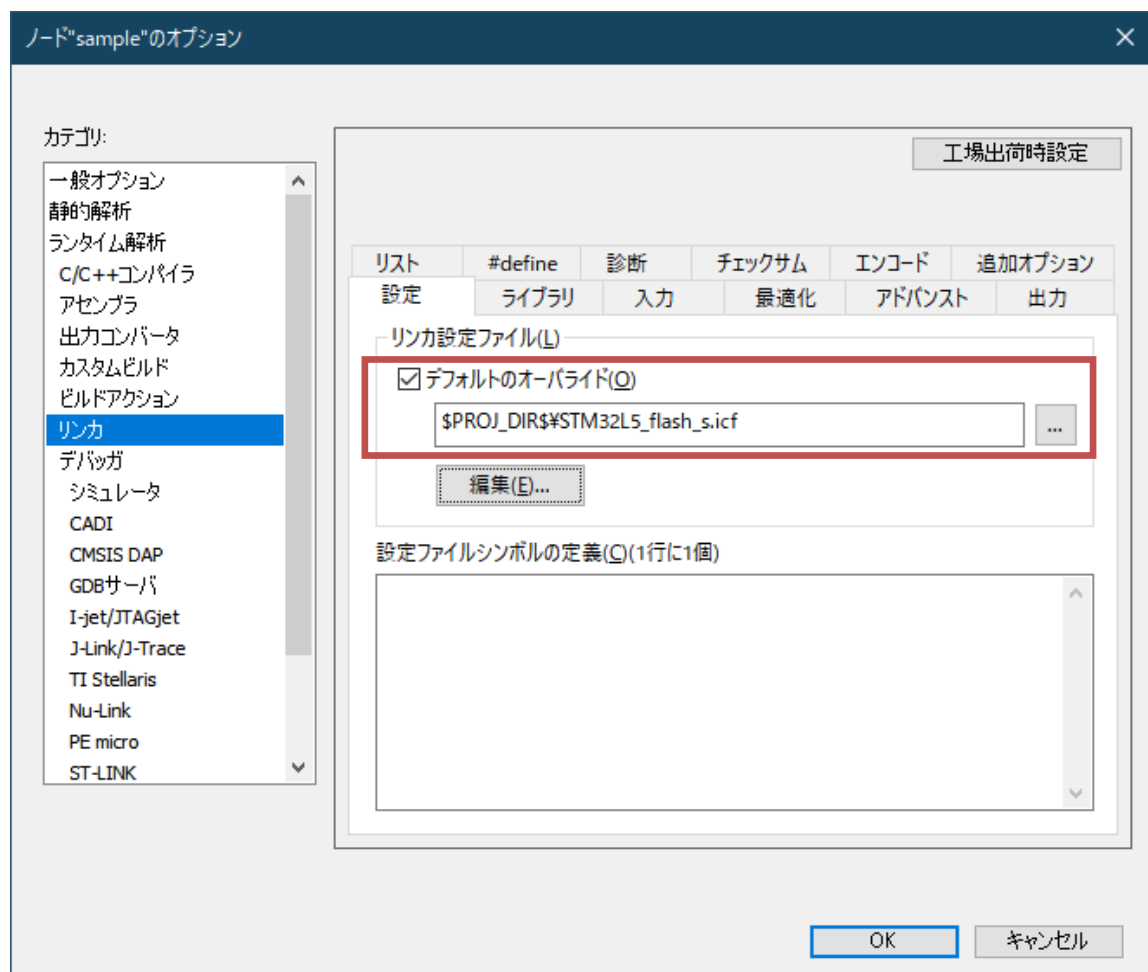


C. リンカ

カテゴリ「リンカ」の「設定」タブでは、「リンカ設定ファイル」の「デフォルトのオーバーライド」をチェックし、デフォルトのリンカ設定ファイルを置き換えます。

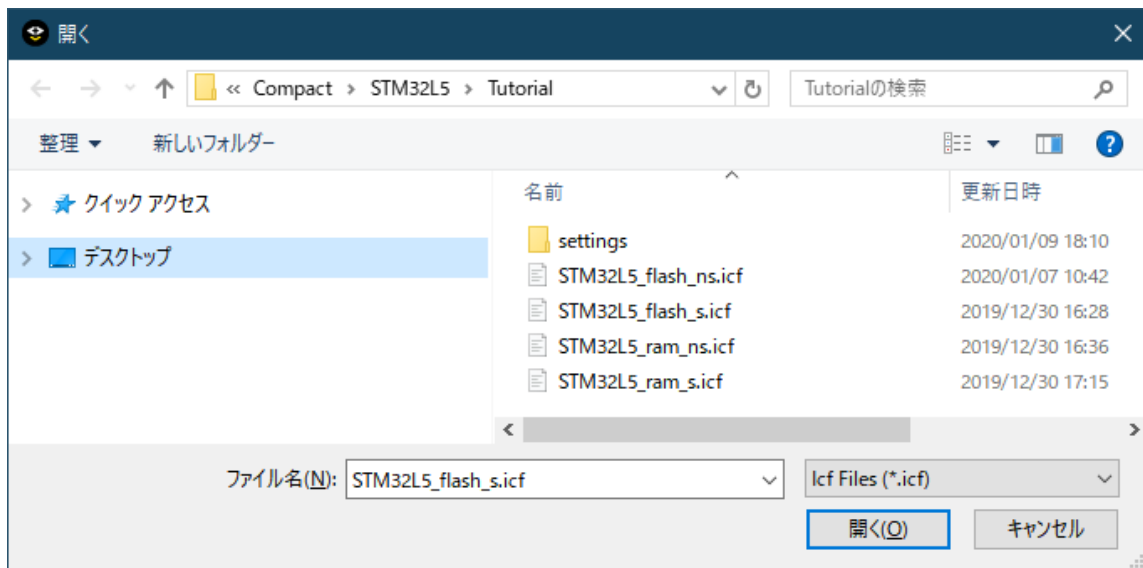
リンカ設定ファイルはパッケージに含まれているサンプルから現在のプロジェクトがあるフォルダにコピーして利用します。⁵プロジェクトファイルのあるフォルダに相対パスで、「\$PROJ_DIR\$¥xxxxxxx.icf」のように指定します。

リンカ設定ファイルのファイル名はデバイスにより異なります。通常は、Flash メモリに書き込む場合のメモリマップの「xxxxxx_flash_s.icf or xxxxxx_flash_ns.icf」と、RAM にロードしてデバッグする場合のメモリマップの「xxxxxx_ram_s.icf or xxxxxx_ram_ns.icf」を用意しています。



⁵ リンカ設定ファイルはコンフィグレータから生成されません。

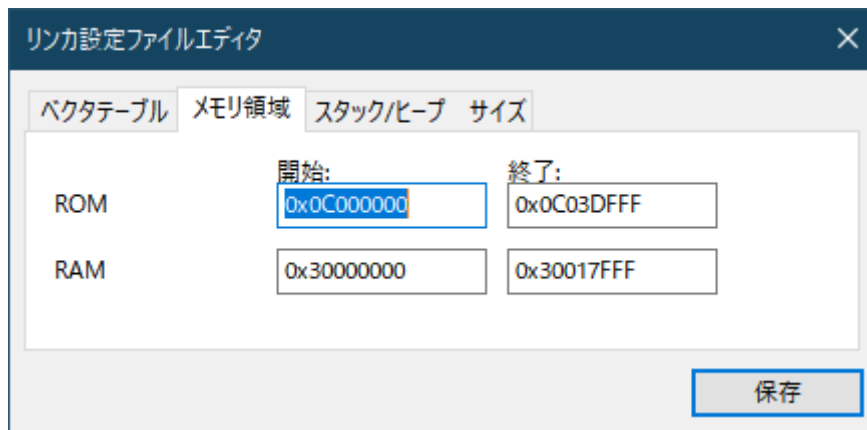
または、右側の「...」をクリックして、リンカ設定ファイル（拡張子 icf）を直接指定することもできます。



リンカ設定ファイルの修正

リンカ設定ファイルには、一般的なメモリマップで定義されていますが、Flash メモリのサイズや内蔵 RAM のサイズはお使いになるデバイスに合わせる修正が必要になります。編集ボタンを押し、「Memory Regions」タブの、ROM の End アドレスと RAM の End アドレスには、デバイスのデータシートを参照して修正します。

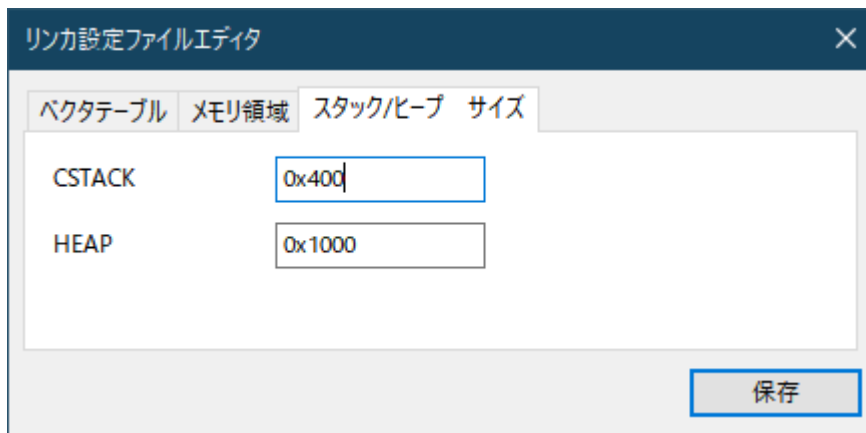
また、RAM にロードするメモリマップの場合では、ROM と RAM の設定は同じにし、End アドレスを内蔵 RAM の最終アドレスを指定します。



The screenshot shows a dialog box titled 'リンカ設定ファイルエディタ' (Linker Settings File Editor) with a close button (X) in the top right corner. Below the title bar are four tabs: 'バクタテーブル' (Bakuta Table), 'メモリ領域' (Memory Regions), 'スタック/ヒープ' (Stack/Heap), and 'サイズ' (Size). The 'メモリ領域' tab is selected. Inside the dialog, there are two rows of memory regions: 'ROM' and 'RAM'. Each row has two input fields: '開始:' (Start) and '終了:' (End). For ROM, the start address is '0x0C000000' and the end address is '0x0C03DFFF'. For RAM, the start address is '0x30000000' and the end address is '0x30017FFF'. A '保存' (Save) button is located at the bottom right of the dialog.

Memory Region	Start Address	End Address
ROM	0x0C000000	0x0C03DFFF
RAM	0x30000000	0x30017FFF

「Stack/Heap Sizes」タブの「CSTACK」には、コンフィグレータで指定したタイムイベントハンドラ用スタックサイズが設定されています。



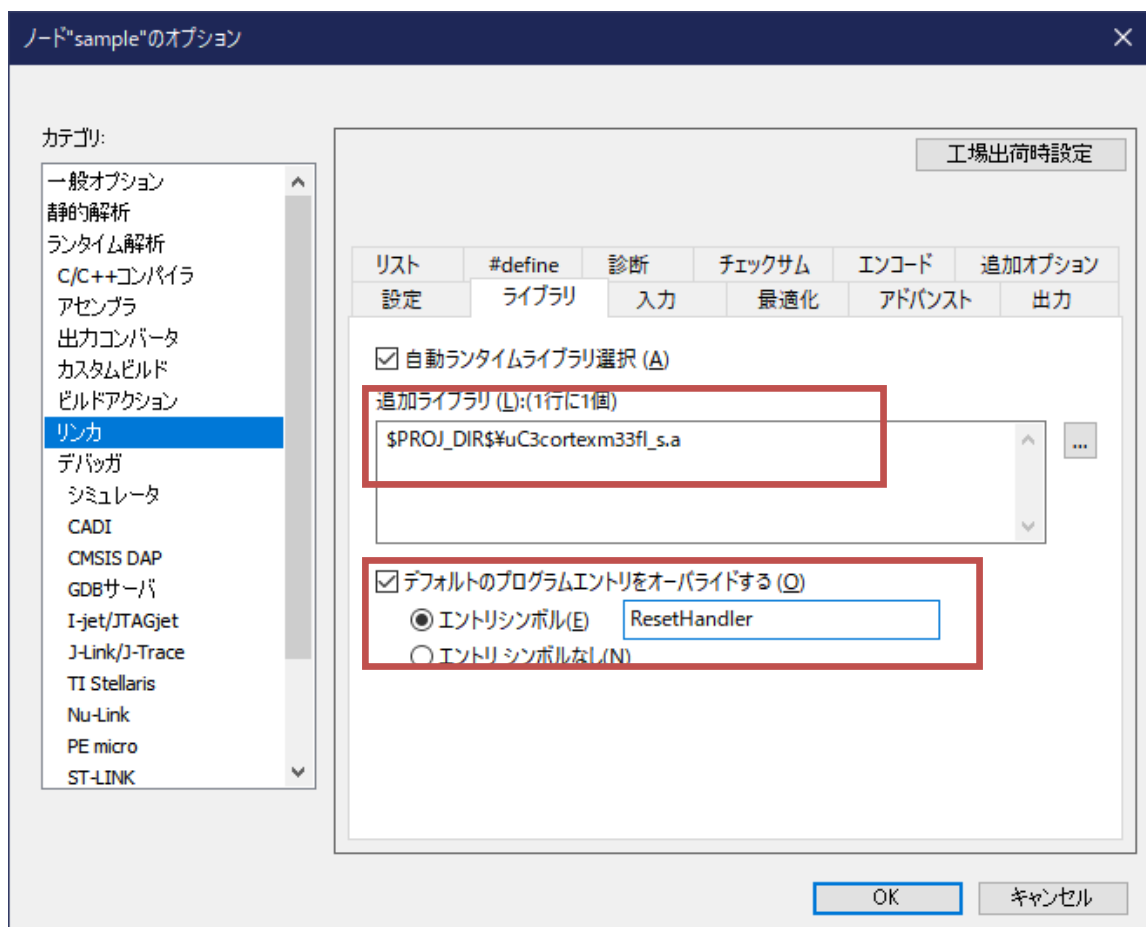
なお、uC3 動作に必要なハンドラ用スタック(HSTACK)はリンカ設定ファイルエディタからは入力できませんので、直接 Windows のメモ帳等で設定してください。

設定例の抜粋

```
define symbol __size_hstack__ = 0x400;
...
define memory mem with size = 4G;
...
define block HSTACK    with alignment = 8, size = __size_hstack__    { };
...
place at end of RAM_region { block RamBottom with fixed order { block HSTACK, block
CSTACK, block HEAP}};
```

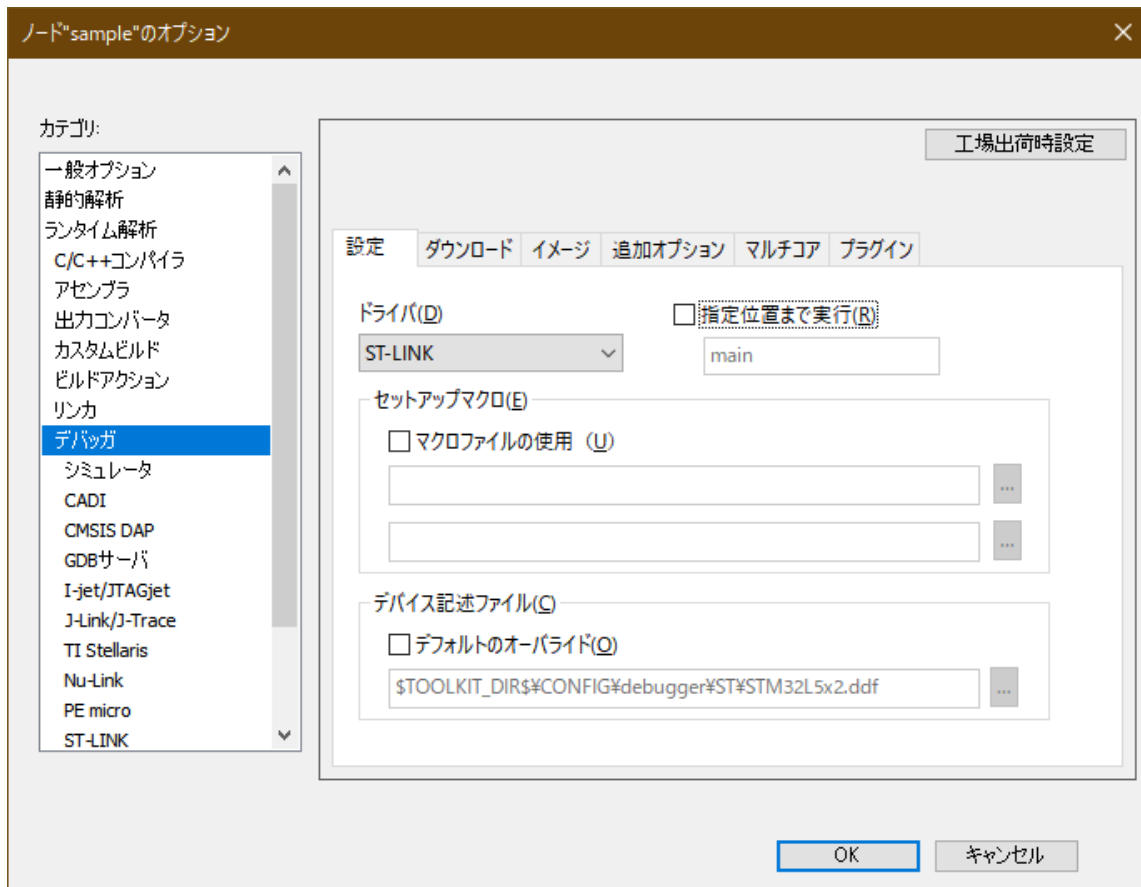
カテゴリ「リンカ」の「ライブラリ」タブでは、「追加ライブラリ」にカーネルライブラリ名を入力します。コンフィグレータは、カーネルライブラリも生成しており、プロジェクトファイルのあるフォルダに相対パスで、「\$PROJ_DIR\$\u00C3cortexm33fl_s.a」のように指定します。なお、「一般オプション」にて設定した、FPU 有無と TrustZone モードに対応したライブラリを指定するようにしてください。使用可能カーネルライブラリに関しては「1. 7 カーネルライブラリ」を参照してください。

プログラムのエントリを「デフォルトのプログラムエントリをオーバーライドする」をチェックし、デフォルトのプログラムのエントリから「ResetHandler」に置き換えます。

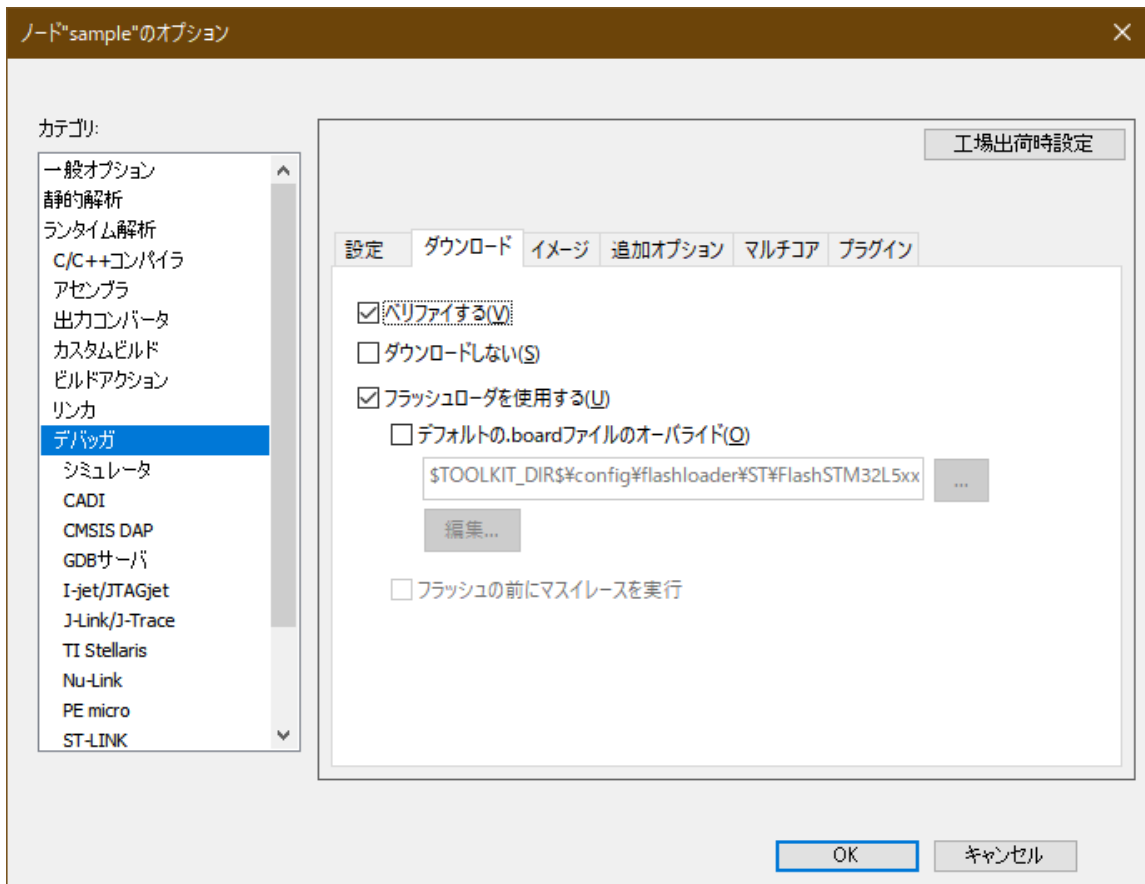


D. デバッガ

使用するデバッガを選択します。本書では「ST-LINK」を選択しています。



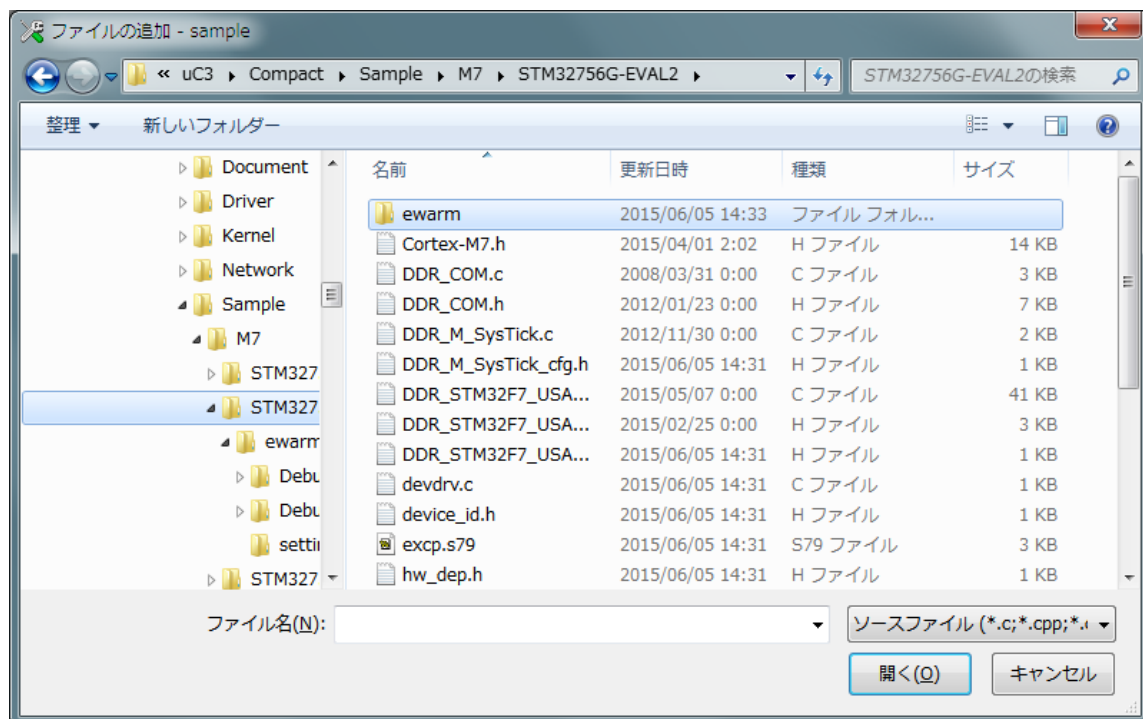
Flash メモリにロードする場合には、カテゴリ「デバッガ」の「ダウンロード」タブでは、「フラッシュローダを使用する」をチェックします。ただし、デフォルトで用意されている Flash ロードは、デバイスリストの登録された内蔵 Flash メモリのみです。その他の Flash メモリを使用する場合には、「IAR Embedded Workbench」のマニュアルを参照してください。



3. 1. 3 ファイルの登録

「プロジェクト」→「ファイルの追加」より、ファイル選択画面を開きます。

選択するファイルは、コンフィグレータが生成した C 言語ソースファイル（拡張子 c）と、アセンブラ言語ソースファイル（拡張子 s79）と、アプリケーションとして作成したファイルです。この時、スケルトンコード（main.c）をテンプレートとして用い、新たなアプリケーションファイルとして作成した場合には、スケルトンコードを指定しないよう注意してください。スケルトンコードで定義したシンボルが、重複定義でエラーが発生します。



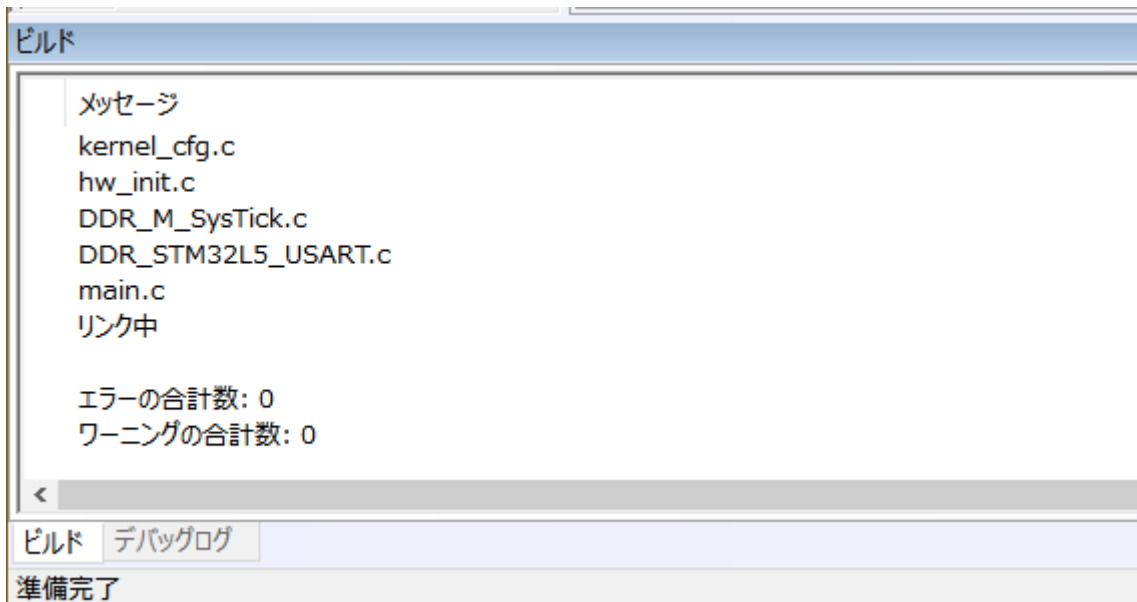
アセンブラ言語ソースファイルは実行ステート毎に生成されますので、「A 一般オプション」で選択したモードのファイルを登録します。

	セキュアステート用	非セキュアステート用
ベクタテーブル	vector_s.s79	vector_ns.s79
リセットハンドラ	reset_handler_s.s79	reset_handler_ns.s79
例外ハンドラ	exception_s.s79	exception_ns.s79

3. 1. 4 メイク

「プロジェクト」→「メイク」により、選択したファイルをコンパイル/アセンブル後にリンクし、最終的なロードモジュールを生成します。また、すべてのファイルを強制的にコンパイル/アセンブルさせる場合には、「プロジェクト」→「すべてを再ビルド」を行います。

これらが、エラーもワーニングもなく、すべて正常に終われば、次のようなメッセージが表示されます。



ロードモジュールは、プロジェクトファイルの下、..`¥Debug¥Exe¥sample.out` か、あるいは..`¥Release¥Exe¥sample.out` が生成されます。

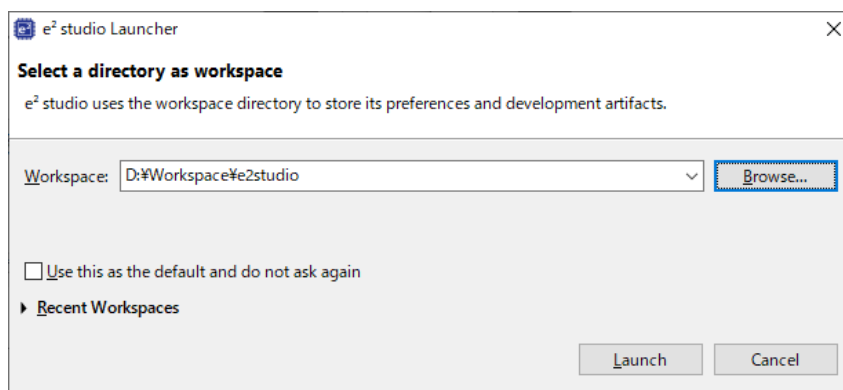
3. 2 Renesas e2studio

Renesas Electronics 社の開発ツール e2studio を使用して、ロードモジュールを作成する手順を説明します。

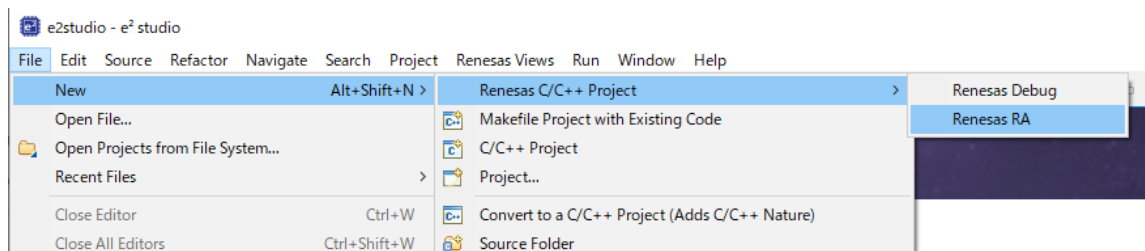
Renesas RA ファミリ向けの e2studio に付属の Flexible Software Package ⁶は使用しない手順です。FSP を使用する場合は、「 μ C3/Compact と Renesas RA FSP の共存ガイド」も合わせて参照ください。

3. 2. 1 新規プロジェクトの生成

- 1) 「e2studio」⁷を起動します。最初に、ワークスペースに任意のフォルダを指定し、「Launch」をクリックします。



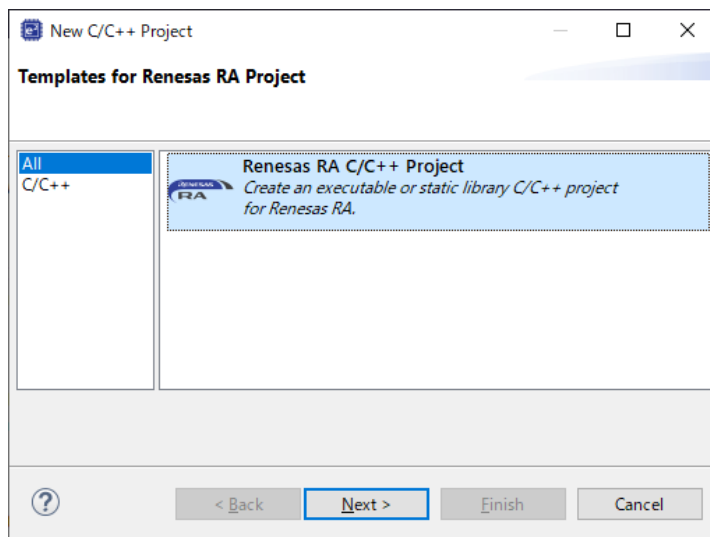
- 2) 「File」→「New」→「Renesas C/C++ Project」→「Renesas RA」を選択します。



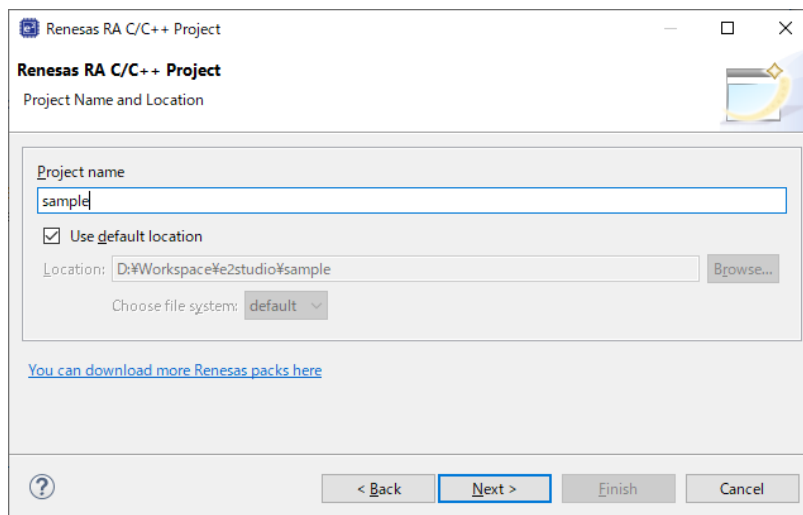
⁶ 以下、FSP と略します。

⁷ e2studio v2021-07 を使用して説明しています。

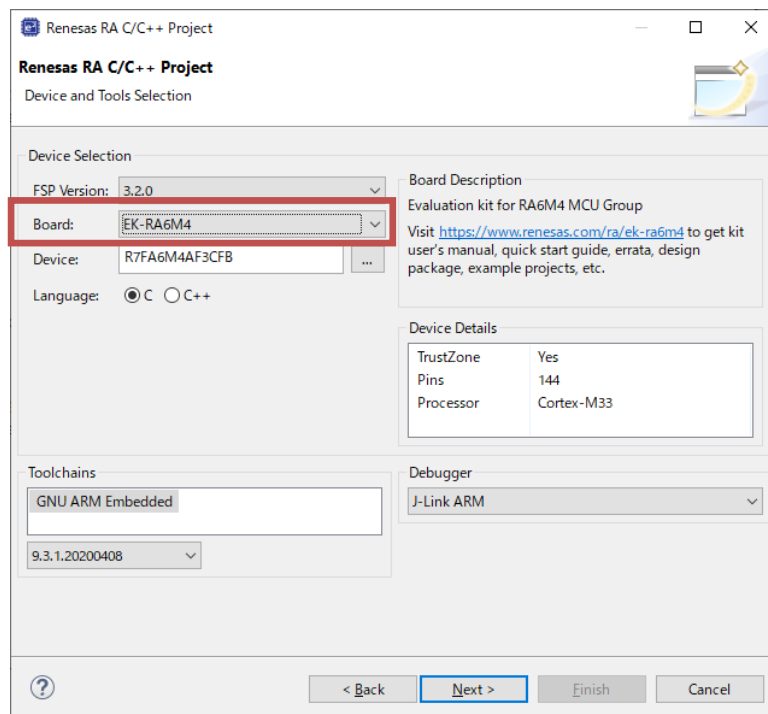
- 3) 「Renesas RA C/C++ Project」を選択し、「Next」をクリックします。



- 4) 「Project name」に任意のプロジェクト名を入力し、「Next」をクリックします。
※ここでは「sample」とします。

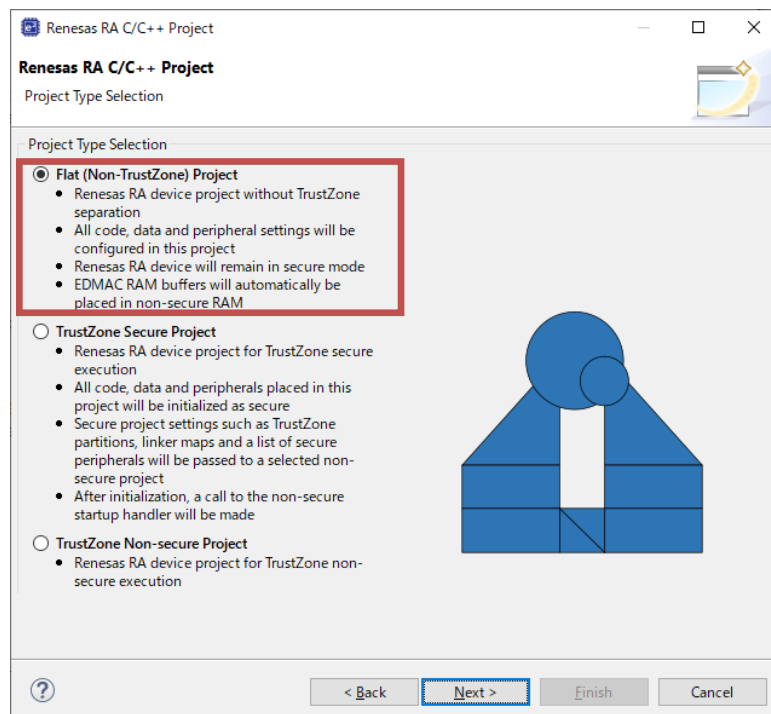


5) 任意のボードを選択して、「Next」をクリックします。

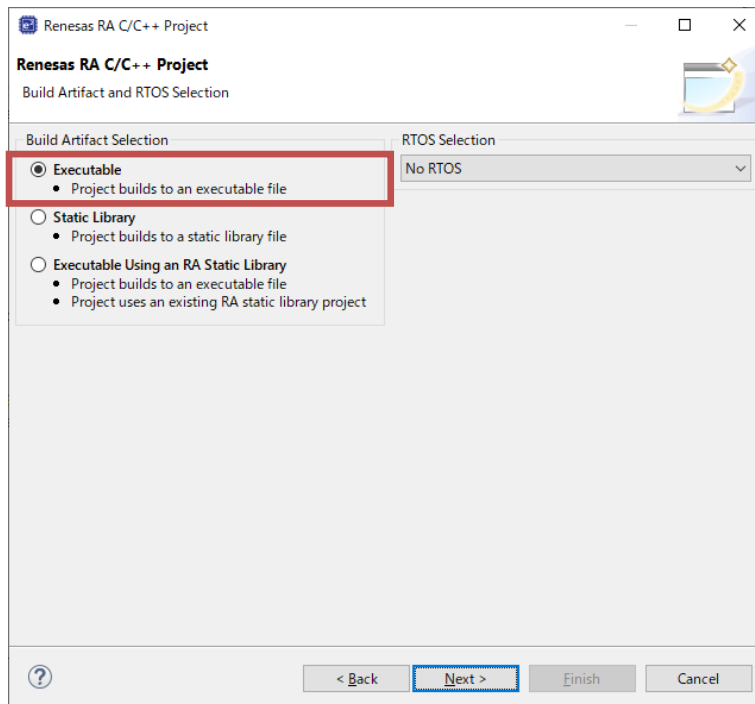


6) 「Flat (Non-TrustZone) Project」を選択して、「Next」をクリックします。

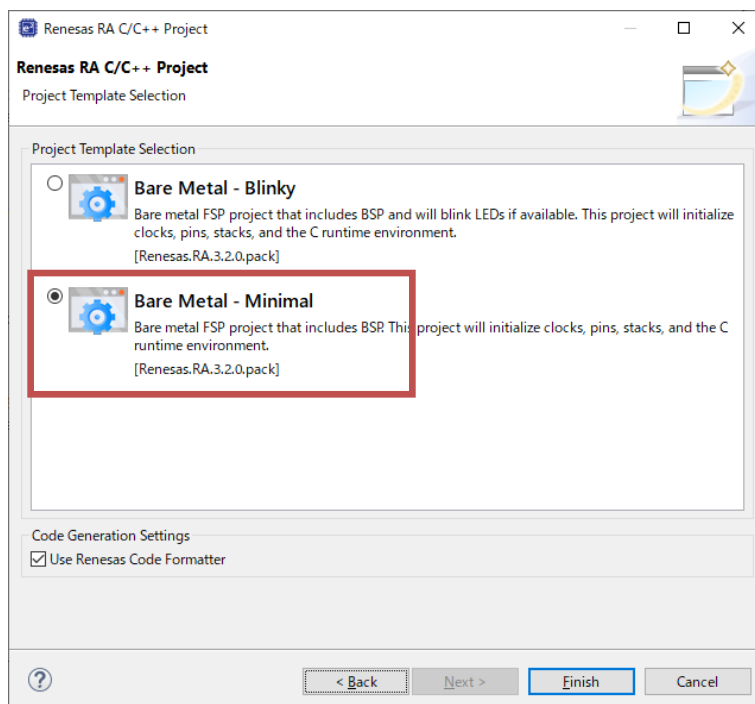
TrustZone を使用して、セキュア・非セキュアのプロジェクトを作成する場合は、適切なプロジェクトを選択してください。



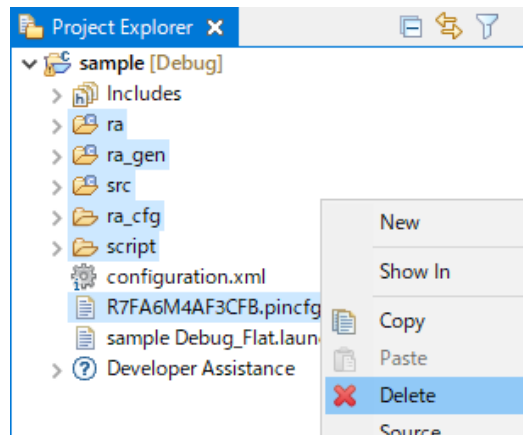
- 7) 「Executable」を選択して、「Next」をクリックします。



- 8) 「Bare Metal – Minimal」を選択して、「Finish」をクリックします。



- 9) プロジェクト作成が完了すると、デフォルトで FSP の関連ファイルが生成されます。今回は FSP の生成ファイルを使用しないため、「Project Explorer」から選択し、「Delete」してください。



削除するフォルダ：ra、ra_gen、src、ra_cfg、script

削除するファイル：*.pincfg (選択したボードによりファイル名が異なります)

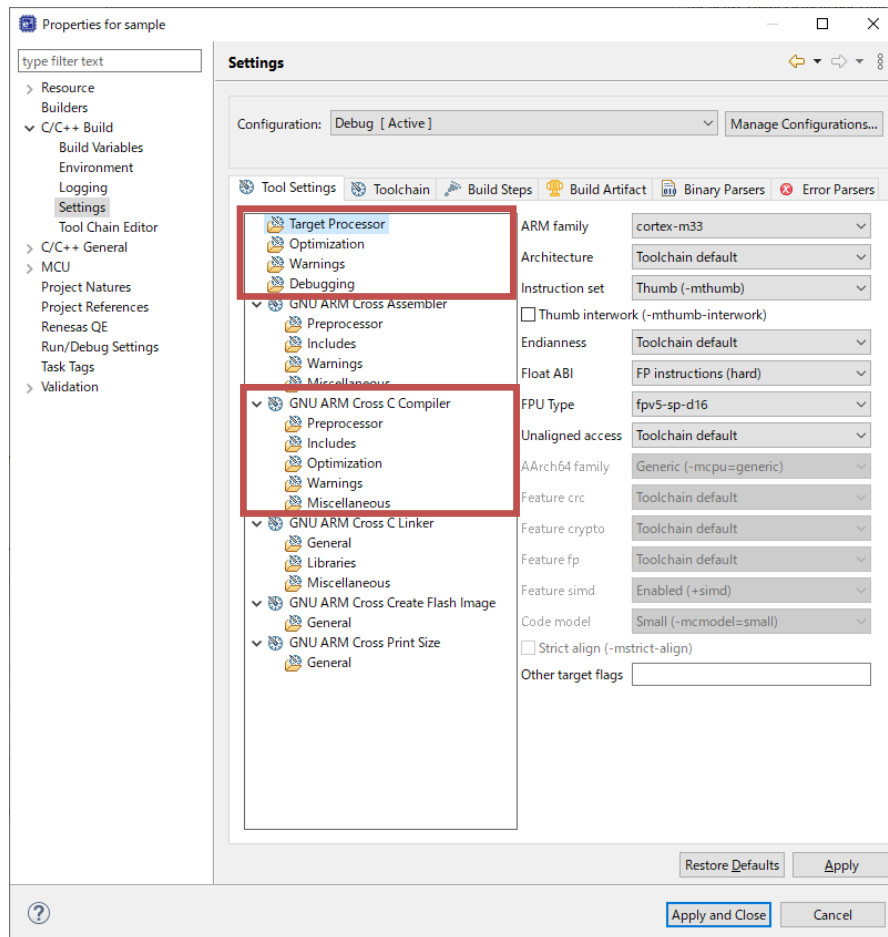
3. 2. 2 オプションの設定

メニュー「Project」→「Properties」より、プロパティ画面を開きます。

A. コンパイラオプション

左ペイン「C/C++ Build」→「Settings」の「Tool settings」タブを開きます。

「Target Processor」、「Optimization」、「Warnings」、「Debugging」、「GNU ARM Cross C Compiler」の項で、C コンパイラオプションを設定します。

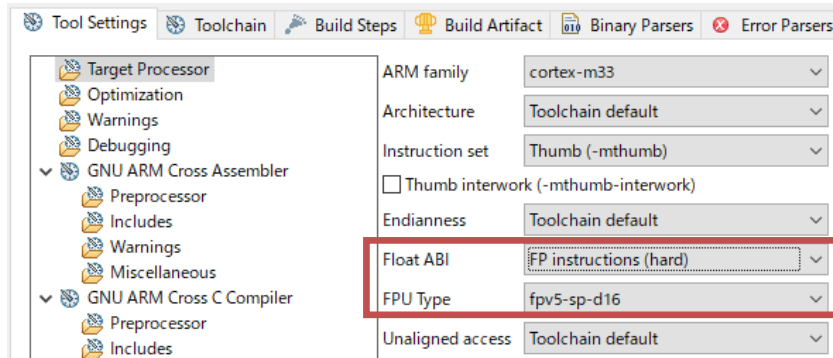


FPU の設定

デフォルトでは、FPU 対応カーネルライブラリを使用する場合の設定になっています。

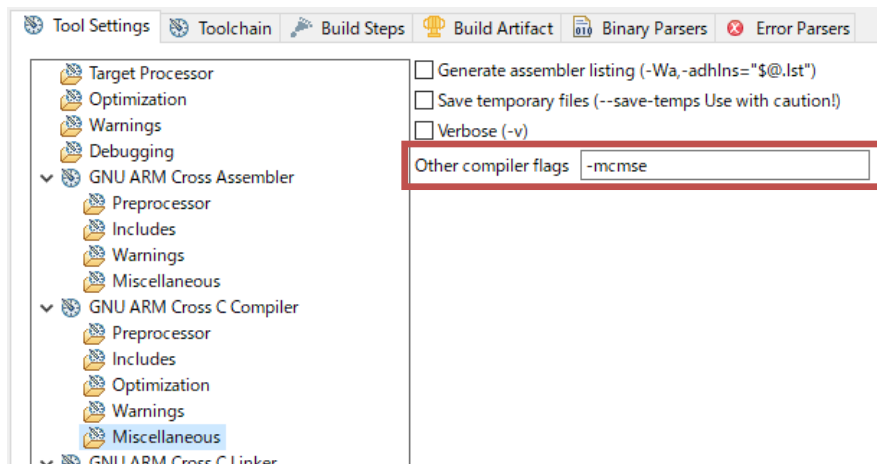
FPU 無しのカーネルライブラリを使用する場合は、「Target Processor」→「Float ABI」で

「Library (soft)」を設定してください。なお、カーネルライブラリに関しては「1.7 カーネルライブラリ」を参照してください。⁸



実行ステートの設定

デバイスが Security Extension をサポートしている場合は、実行ステートをセキュアまたは非セキュアに設定できます。セキュアに設定する場合は、「Other compiler flags」に「-mcmse」を設定します。



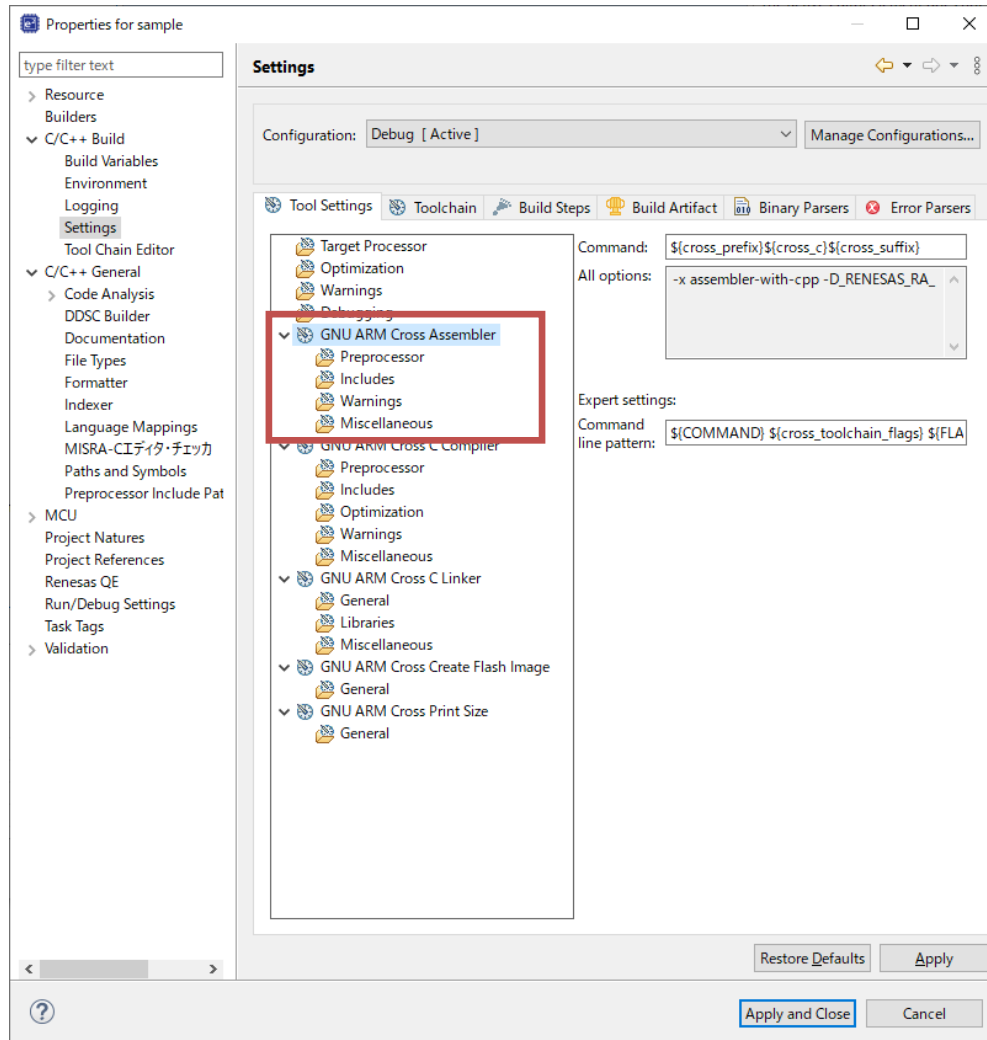
⁸ デバイスが VFPv5 double-precision をサポートしていない場合がありますので、デバイスのマニュアルを参照のうえ、選択してください。

B. アセンブラオプション

左ペイン「C/C++ Build」→「Settings」の「Tool settings」タブを開きます。

「GNU ARM Cross Assembler」の項で、アセンブラオプションを設定します。

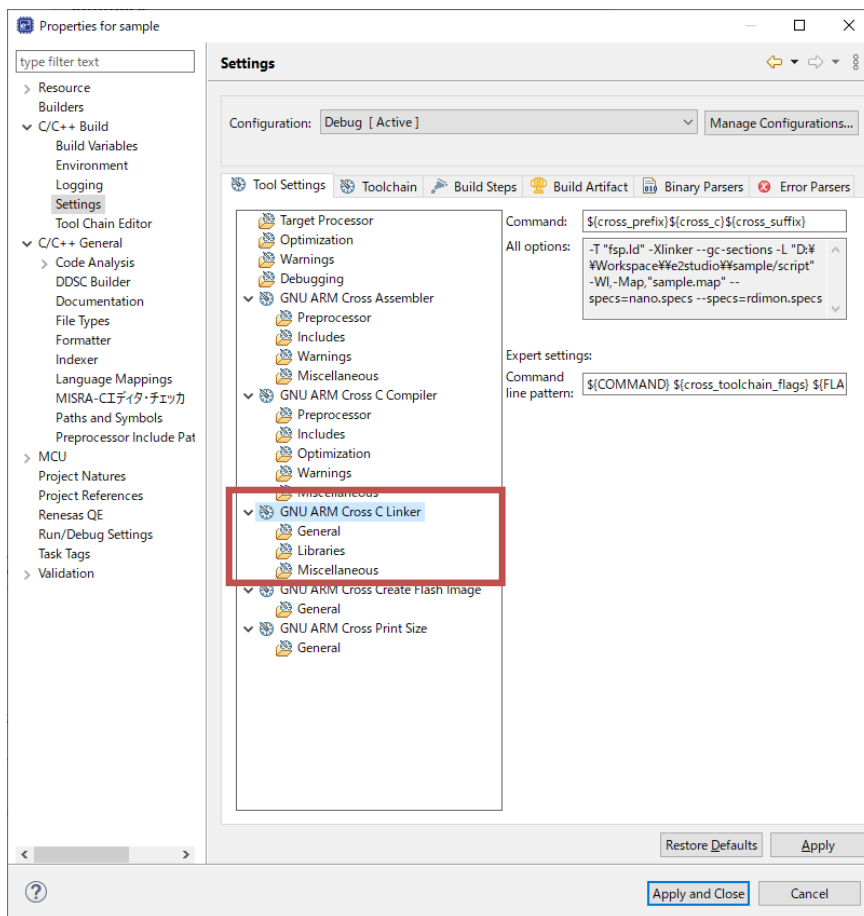
※今回は、特別な設定はありませんのでデフォルトのままとします。



C. リンカオプション

左ペイン「C/C++ Build」→「Settings」の「Tool settings」タブを開きます。

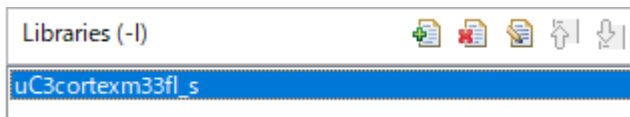
「GNU ARM Cross C Linker」の項で、リンカオプションを設定します。



カーネルライブラリの設定

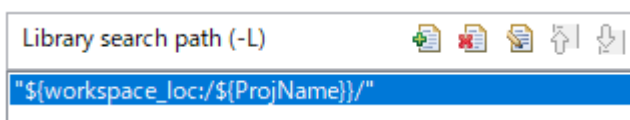
「Libraries (-l)」に、カーネルライブラリ名を入力します。

コンフィグレータで出力されるカーネルライブラリ名(lib*.a)の先頭「lib」と拡張子「.a」を外した名前を指定します。たとえば、libuC3cortexm33fl_s.a の場合には、「uC3cortexm33fl_s」を設定します。



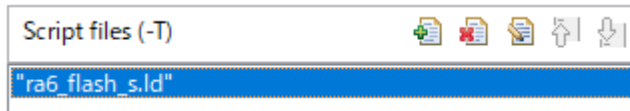
「Library search path (-L)」には、

プロジェクトフォルダ直下のパス "\${workspace_loc:\${ProjName}}/" を設定します。



リンカスクリプトの設定

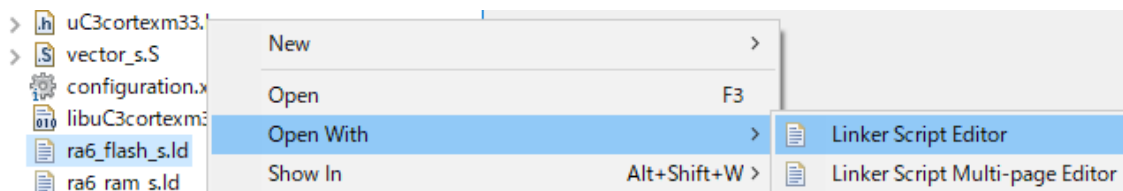
Linker Script(-T): "ra6_flash_s.ld"



リンカスクリプトは、パッケージに含まれているサンプル内のファイルをコピーして利用します。⁹リンカスクリプト名はデバイスにより異なります。通常は、Flash メモリに書き込む場合のメモリマップの「xxxxxxx_flash_s.ld or xxxxxx_flash_ns.ld」と、RAM にロードしてデバッグする場合のメモリマップの「xxxxxxx_ram_s.ld or xxxxxx_ram_ns.ld」を用意しています。

リンカスクリプトの修正

リンカスクリプトファイルには、一般的なメモリマップが定義されていますが、Flash メモリのサイズや内蔵 RAM のサイズはお使いになるデバイスに合わせる修正が必要になります。

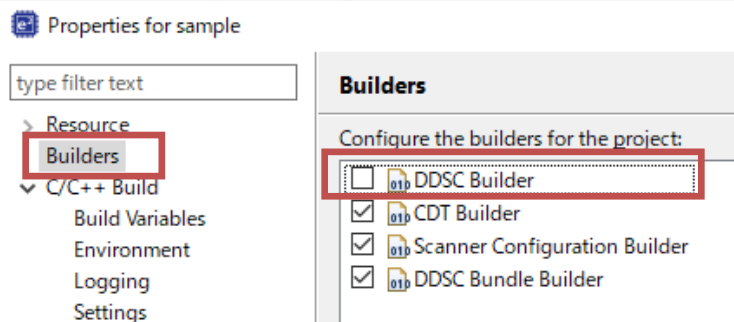


リンカスクリプトファイルを右クリックし、「Linker Script Editor」で開いて編集してください。

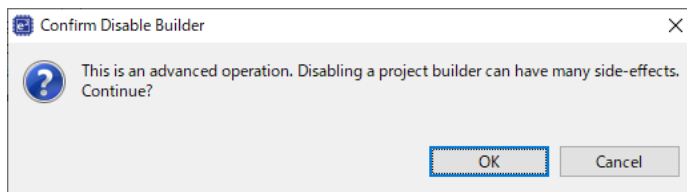
⁹ リンカスクリプトはコンフィグレータから生成されません。

D. ビルダー

左ペイン「Builders」を選択し、「DDSC Builder」のチェックをはずします。



確認のダイアログが表示された場合は、「OK」をクリックします。



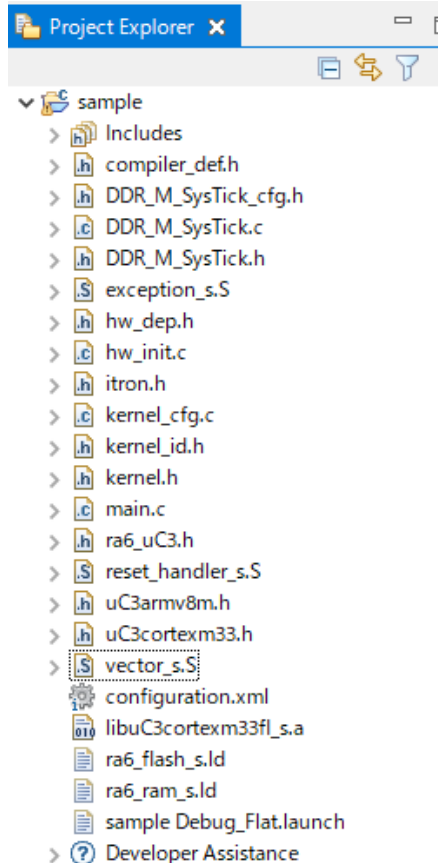
※ 「3. 2. 1 新規プロジェクトの生成」の手順 (9) のように、今回は FSP の生成ファイルを使用しないため、FSP のファイルが生成されないように「DDSC Builder」のチェックをはずしています。

FSP を使用する場合には、本手順は必要ありません。

3. 2. 3 ファイルの登録

コンフィグレータで生成したファイルを「3. 2. 1 新規プロジェクトの生成」で作成したプロジェクトフォルダ配下に移動します。e2studio では、プロジェクトフォルダ配下にあるソースコードが自動登録されます。

スケルトンコード (main.c) をテンプレートとして用い、新たなアプリケーションファイルとして作成した場合には、スケルトンコードは含めないように注意してください。スケルトンコードで定義したシンボルが、重複定義でエラーが発生します。



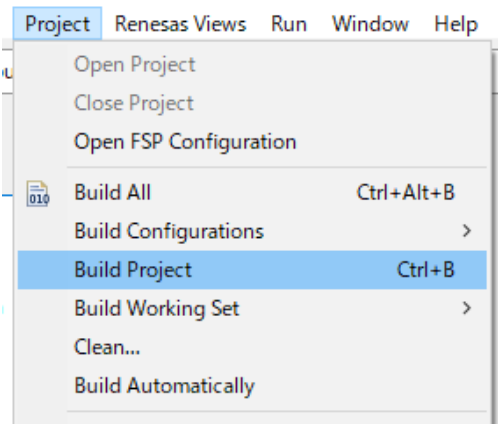
アセンブラ言語ソースファイルは実行ステート毎に生成されますので、「3. 2. 2 オプションの設定」の「A. コンパイラオプション」で選択したモードのファイルを登録します。

	セキュアステート用	非セキュアステート用
ベクタテーブル	vector_s.S	vector_ns.S
リセットハンドラ	reset_handler_s.S	reset_handler_ns.S
例外ハンドラ	exception_s.S	exception_ns.S

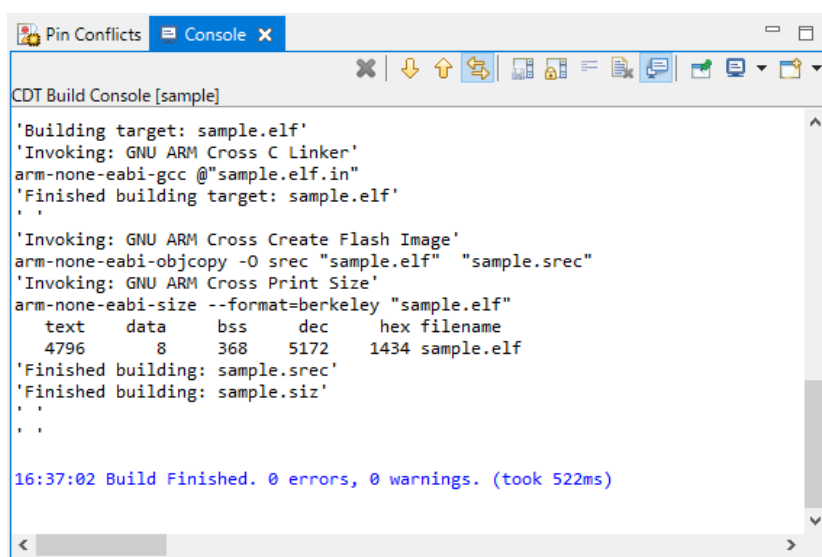
3. 2. 4 ビルド

メニューの「Project」→「Build Project」により、選択したファイルをコンパイル/アセンブル後にリンクし、最終的なロードモジュールを生成します。

また、すべてのファイルを強制的にコンパイル/アセンブルさせる場合には、一度「Project」→「Clean...」を実行後、再度「Project」→「Build Project」を行います。



これらが、エラーもワーニングもなく、すべて正常に終われば、次のようなメッセージが表示されます。



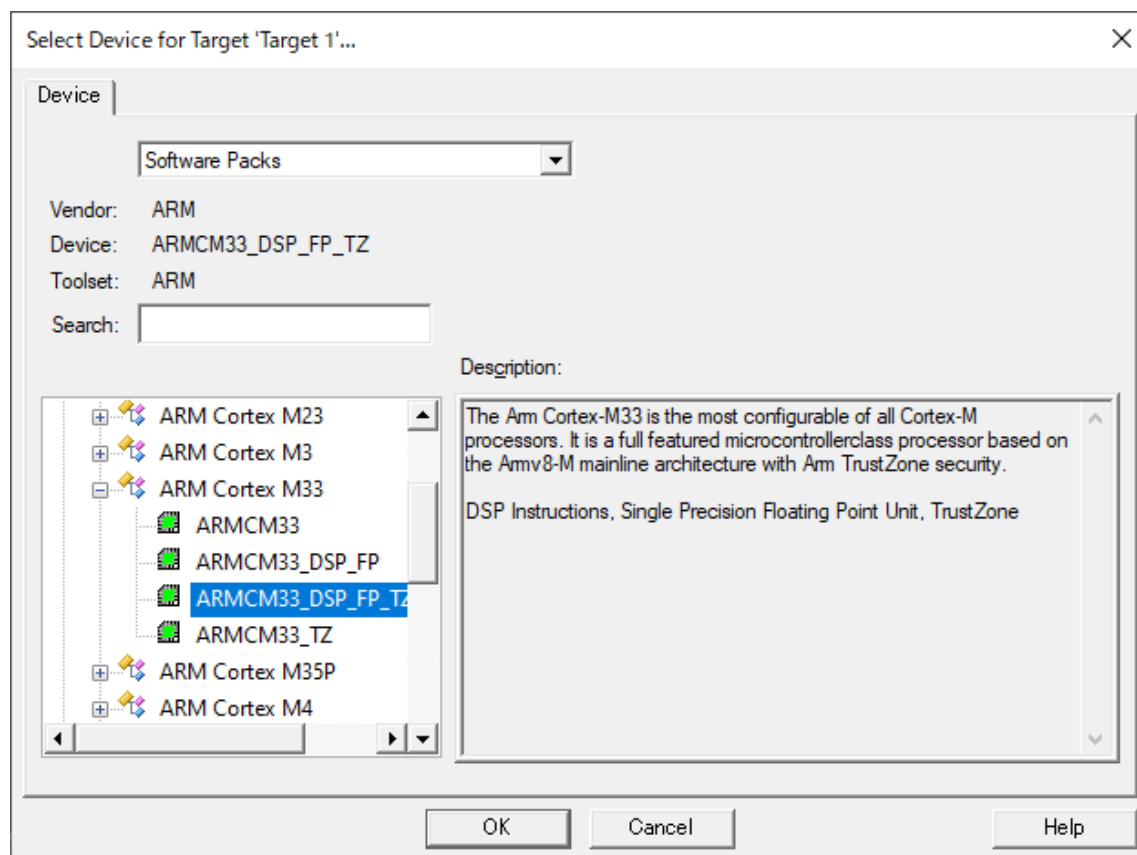
ロードモジュールは、プロジェクトファイルの下、`..¥Debug¥sample.elf` か、あるいは `..¥Release¥sample.elf` が生成されます。

3. 3 MDK-ARM

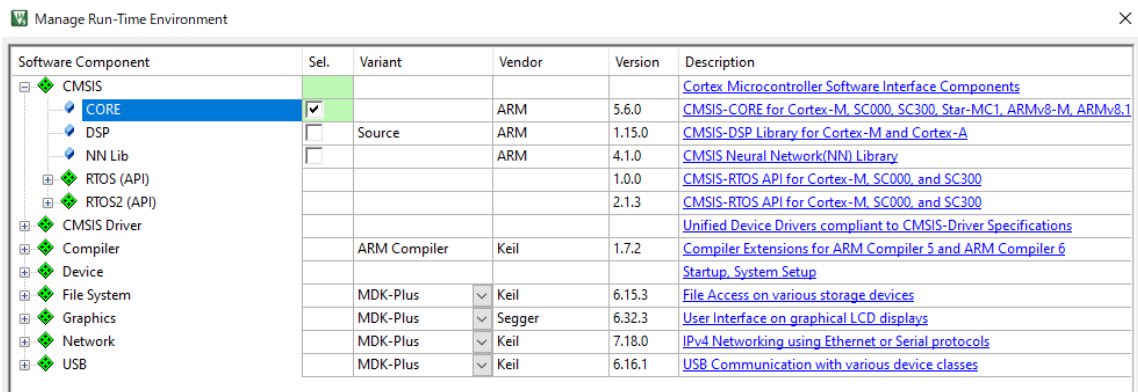
Keil 社の開発ツール RealView Microcontroller Development Kit (MDK-ARM) の IDE である μ Vision を使用して、ロードモジュールを作成する手順を説明します。

3. 3. 1 新規プロジェクトの生成

μ Vision を起動します。最初に、「Project」→「New μ Vision Project ...」を選択し、新規にプロジェクトを作成します。「Select Device for Target...」のダイアログでは使用するマイコンを指定します。

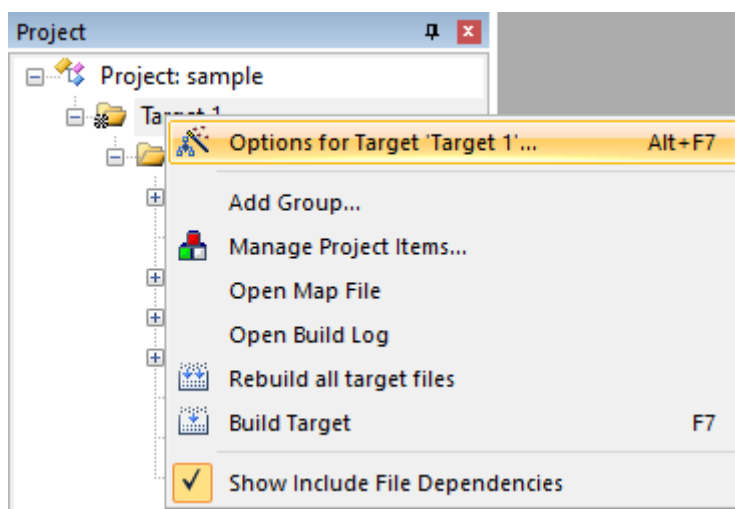


「Manage Run-Time…」のダイアログでは使用するコンポーネントを設定します。



3. 3. 2 オプションの設定

Project ウィンドウで「Target 1」を選択し、右メニューで「Option for Target…」を指定します。



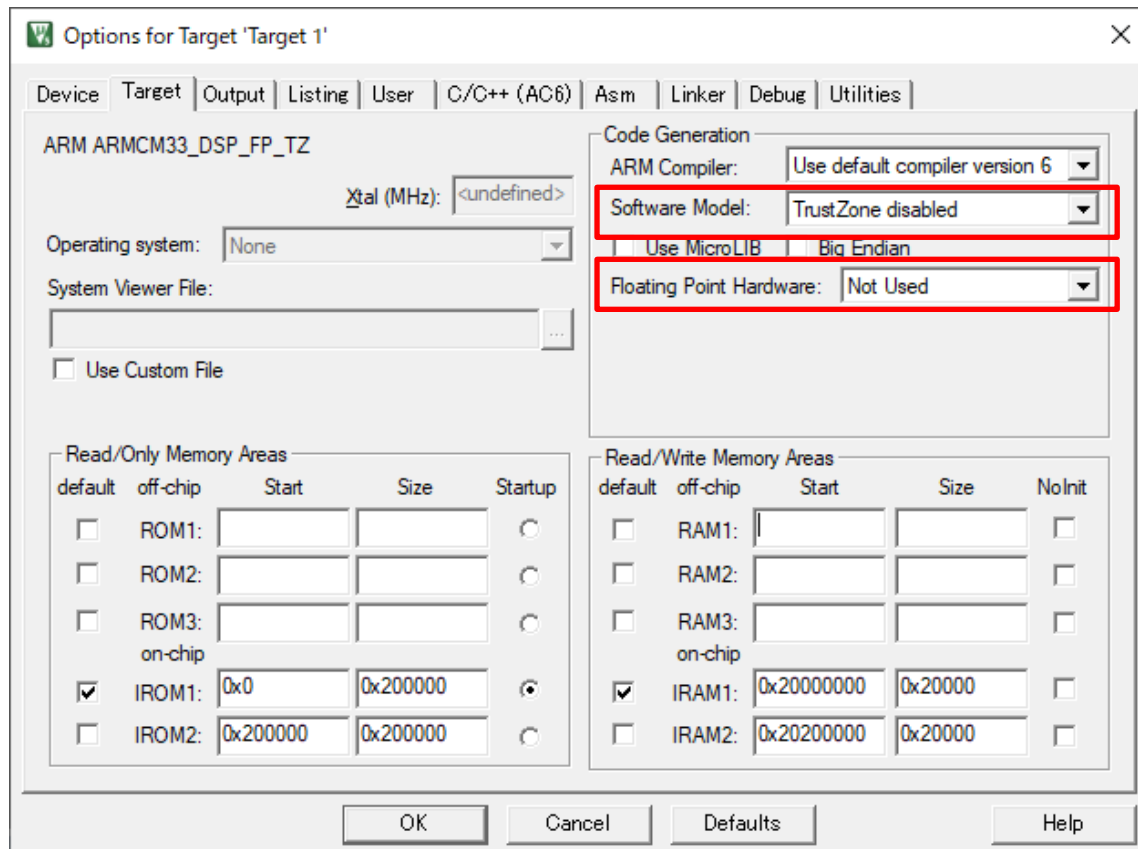
A. ターゲット設定

「Target」タブを開きます。

「Software Model」設定で、セキュアモードを選択します。

「Floating Point Hardware」設定で、FPU を選択します。

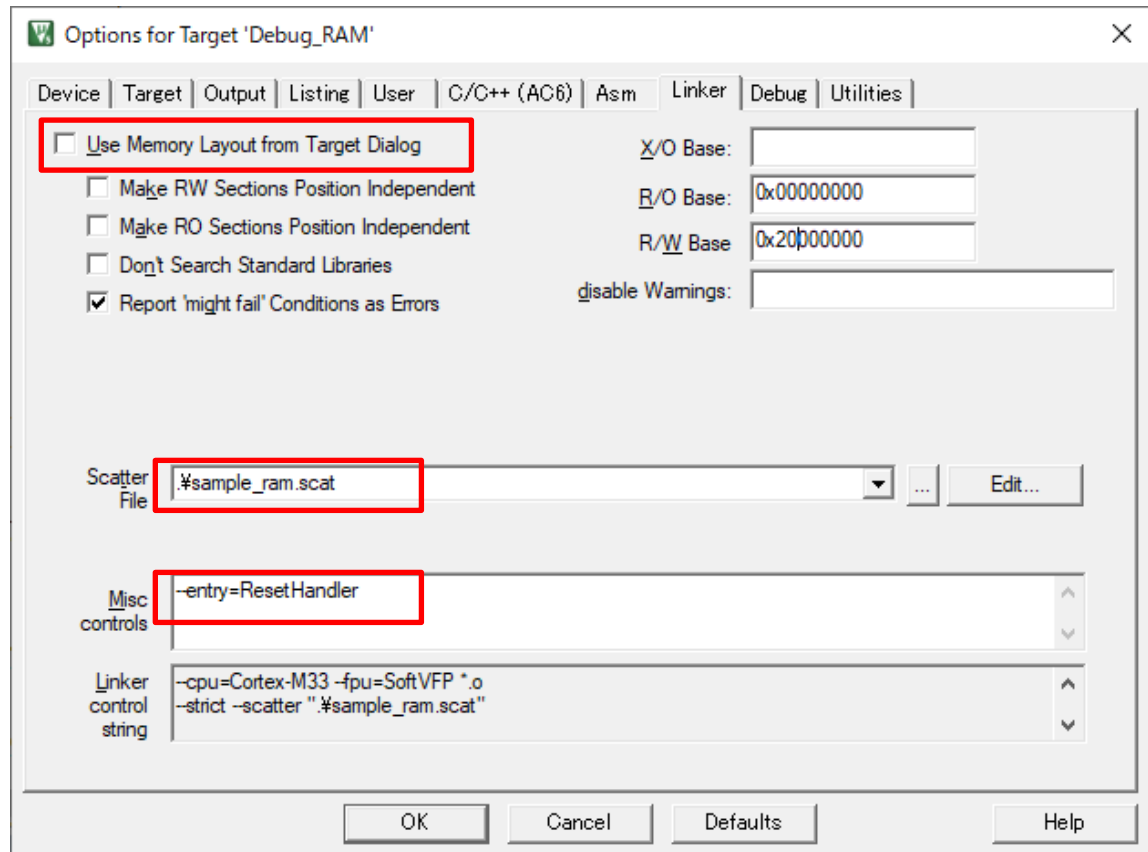
使用する uC3 のカーネルライブラリの設定と合わせる必要があります。



B. リンカオプション

「Linker」タブを開きます。

「Use Memory Layout…」のチェックをはずし、「Scatter File」でコンフィギュレータが出力したスキッタファイルを指定します。また、エントリポイントを設定するため、「Misc controls」に「--entry=ResetHandler」を追加します。

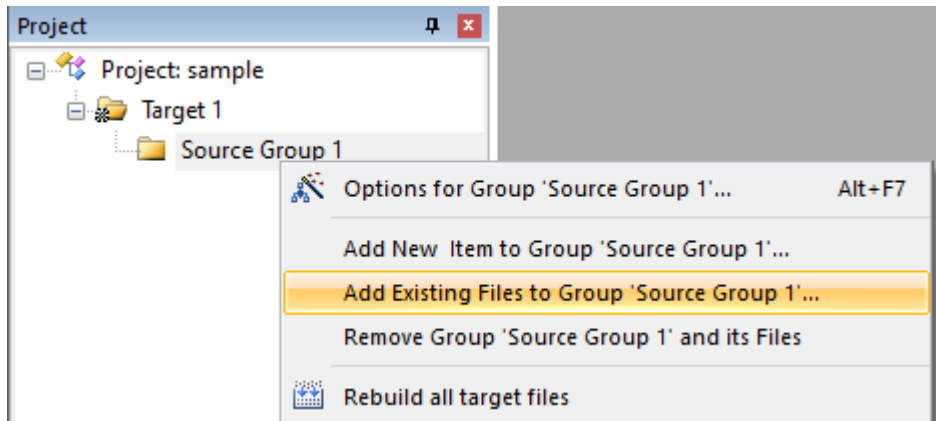


C. その他のオプション

その他のオプションは使用するターゲットや環境に合わせて設定してください。

3. 3. 3 ファイルの登録

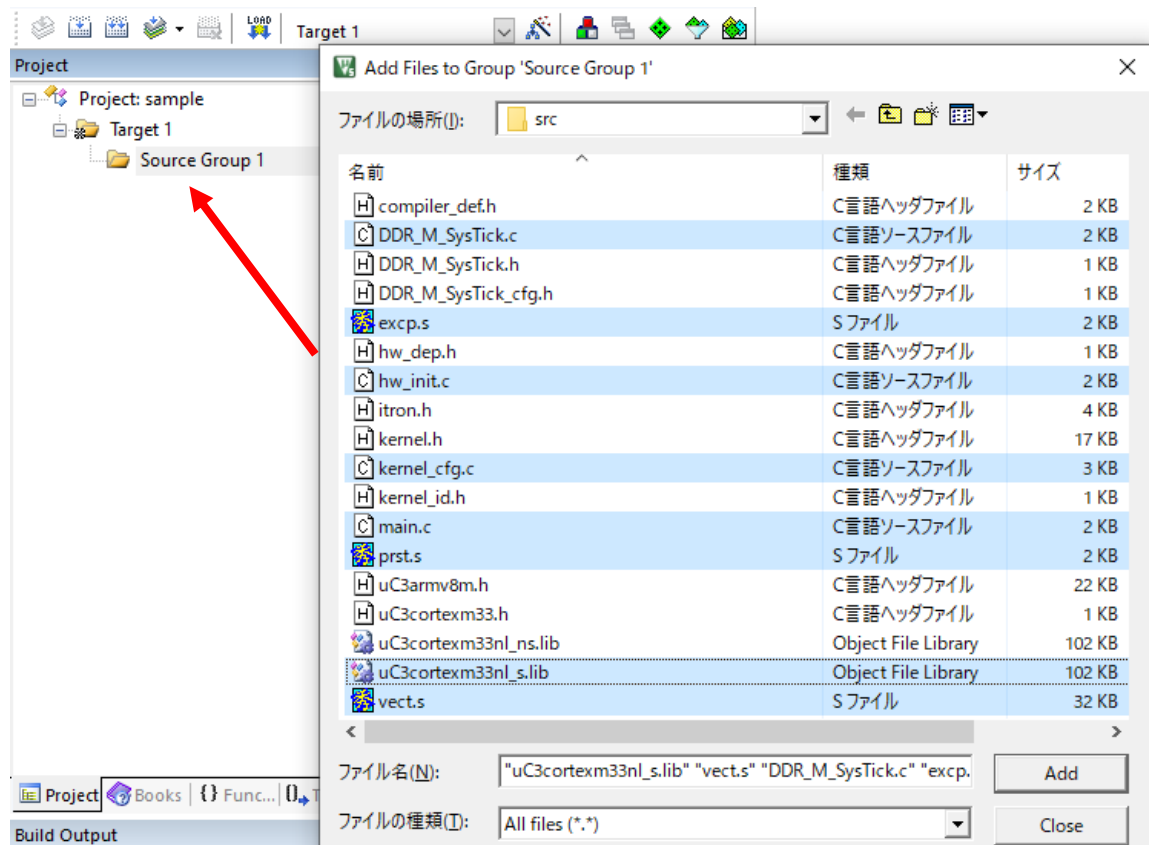
コンフィギュレータが出力した OS のライブラリとソースファイルをプロジェクトに追加します。Project ウィンドウの「Source Group 1」を選択し、右メニューで「Add Existing File to Group…」を指定して各ファイルを追加します。



選択するファイルは、コンフィグレータが生成した C 言語ソースファイル（拡張子 c）、アセンブラ言語ソースファイル（拡張子 s）、カーネルライブラリとアプリケーションとして作成したファイルです。

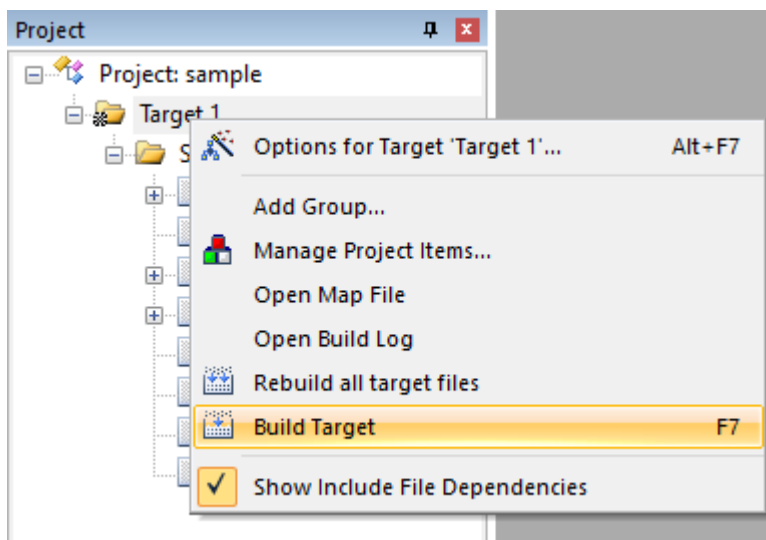
なお、使用可能カーネルライブラリに関しては「1. 6 カーネルライブラリ」を参照してください。

この時、スケルトンコードをテンプレートとして用い、新たなアプリケーションファイルとして作成した場合には、スケルトンコードを指定しないよう注意してください。スケルトンコードで定義したシンボルが、重複定義でエラーが発生します。



3. 3. 4 ビルド

Project ウィンドウで「Target 1」を選択し、右メニューで「Build Target」を指定すると、選択したファイルをコンパイル/アセンブル後にリンクし、最終的なロードモジュールを生成します。また、すべてのファイルを強制的にコンパイル/アセンブルさせる場合には、「Rebuild all target files」を指定します。



これらが、エラーもワーニングもなく、すべて正常に終われば、次のようなメッセージが表示されます。

```
Build Output
compiling hw_init.c...
compiling main.c...
linking...
Program Size: Code=3408 RO-data=2408 RW-data=0 ZI-data=2508
".\Objects\sample.axf" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:00
```

μC3/Compact ユーザーズガイド プロセッサ依存部 Cortex-M33 編

イー・フォース株式会社 <https://www.eforce.co.jp/>

お問い合わせ support@eforce.co.jp

Copyright (C) 2019-2026 eForce Co.,Ltd. All Rights Reserved.
