

# RX開発環境移行ガイド

V850からRXへの移行

(統合開発環境編)

(PM+ / CubeSuite / CubeSuite+ / CS+ → CS+)

2017/04/20

R20UT2609JJ0101

ソフトウェア事業部ソフトウェア技術部  
ルネサス システムデザイン株式会社

# はじめに

---

- 本資料は、V850ファミリ用CコンパイラCA850およびCX のプロジェクトを RXファミリ用Cコンパイラ CC-RXのプロジェクトへ移行する際の、CS+のプロジェクトの操作方法について記述しています。
- 本資料では、統合開発環境CS+、V850ファミリ用Cコンパイラ およびRXファミリ用Cコンパイラ CC-RXを対象に説明しています。  
対象バージョンは以下の通りです。
  - CS+ V5.00.00
  - CX V1.31
  - CA V3.50
  - CC-RX V2.06.00

# アジェンダ

---

- 既存環境からRX用CS+への移行 ページ 04
  - プロジェクト新規作成によるプロジェクト移行 ページ 06
  - 既存プロジェクトの流用1 ページ 07
  - 既存プロジェクトの流用2 ページ 08
- V850用プロジェクトとの違い ページ 10
  - スタートアップファイル ページ 11
  - オプション設定 ページ 15
  - 最適化オプションの設定 ページ 24

# 既存環境からRX用CS+への移行

# 既存環境からRX用CS+への移行

---

- 作成済みのPM+、CubeSuite、CubeSuite+、CS+のV850用プロジェクトをRX用のCS+環境へ移行するには以下の方法があります。

## (1) CS+で新規にプロジェクトを作成

作成済みのお客様のソースファイルをRX用CS+のプロジェクトを作成し登録

## (2) 既存プロジェクトの流用1

V850用CubeSuite+、CS+のプロジェクトを流用してRX用CS+用の新規プロジェクトを作成

## (3) 既存プロジェクトの流用2

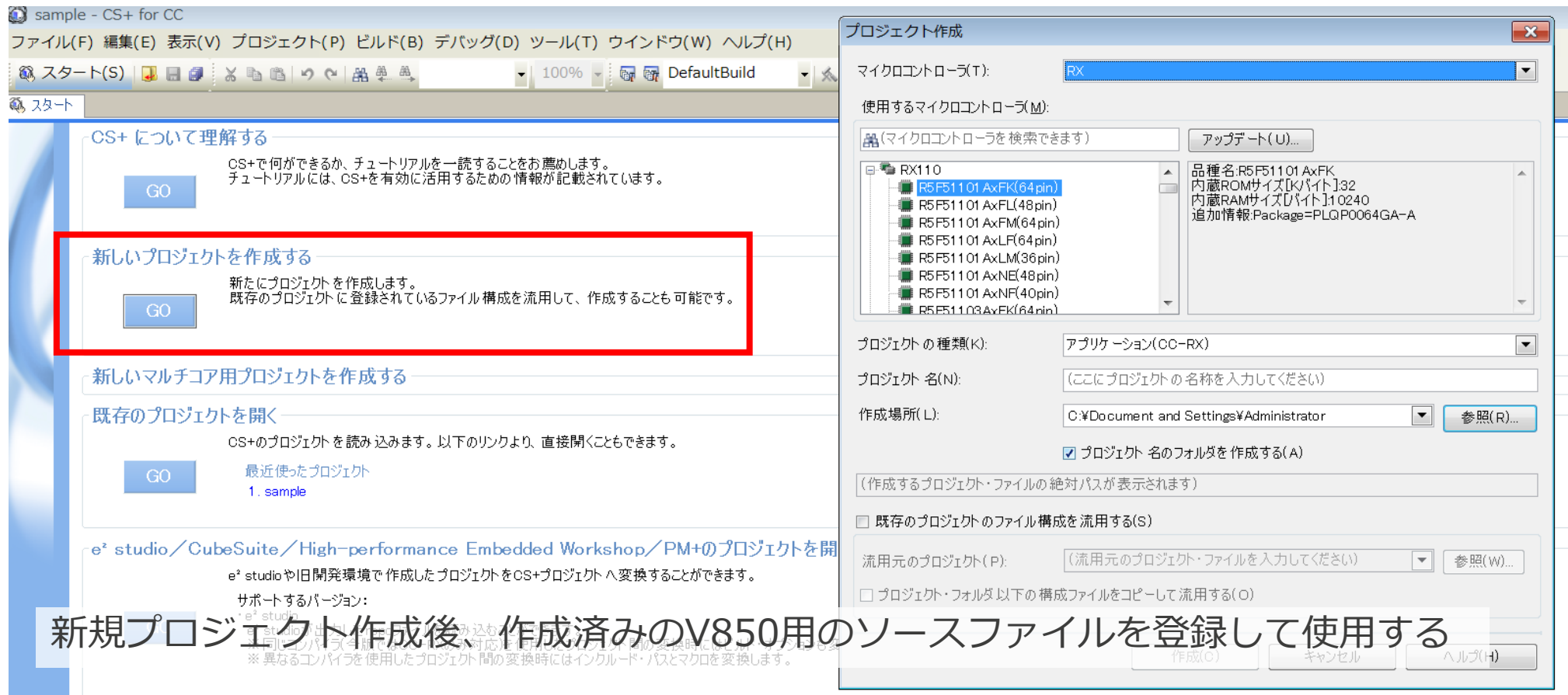
V850用PM+、CubeSuiteのプロジェクトをCS+で直接読み込む

但し、流用する場合はファイル構成を継承しますが、RX用のスタートアップファイルの生成やオプション設定は行いません。

プロジェクト作成後はファイル差し替えやオプション設定を行ってください。

# プロジェクトの新規作成によるプロジェクト移行

- CS+で新規にRX用のプロジェクトを作成



sample - CS+ for CC  
ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) ツール(T) ウィンドウ(W) ヘルプ(H)  
スタート(S) 100% DefaultBuild

CS+ について理解する  
GO  
CS+で何が出来るか、チュートリアルを一読することをお勧めします。  
チュートリアルには、CS+を有効に活用するための情報が記載されています。

**新しいプロジェクトを作成する**  
GO  
新たにプロジェクトを作成します。  
既存のプロジェクトに登録されているファイル構成を流用して、作成することも可能です。

新しいマルチコア用プロジェクトを作成する

既存のプロジェクトを開く  
GO  
CS+のプロジェクトを読み込みます。以下のリンクより、直接開くこともできます。  
最近使ったプロジェクト  
1. sample

e<sup>2</sup> studio / CubeSuite / High-performance Embedded Workshop / PM+のプロジェクトを開く  
GO  
e<sup>2</sup> studioや旧開発環境で作成したプロジェクトをCS+プロジェクトへ変換することができます。  
サポートするバージョン:  
※異なるコンパイラを使用したプロジェクト間の変換時にはインクルードパスとマクロを変換します。

プロジェクト作成

マイクロコントローラ(T): RX

使用するマイクロコントローラ(M):  
(マイクロコントローラを検索できます) アップデート(U)...

RX110  
R5F51101 AxFK(64 pin)  
R5F51101 AxFL(48 pin)  
R5F51101 AxFM(64 pin)  
R5F51101 AxLF(64 pin)  
R5F51101 AxLM(36 pin)  
R5F51101 AxNE(48 pin)  
R5F51101 AxNF(40 pin)  
R5F51103 AxFK(64 pin)

品種名:R5F51101 AxFK  
内蔵ROMサイズ[kバイト]:32  
内蔵RAMサイズ[バイト]:10240  
追加情報:Package=PLQP0064GA-A

プロジェクトの種類(K): アプリケーション(CO-RX)

プロジェクト名(N): (ここにプロジェクトの名称を入力してください)

作成場所(L): C:\Document and Settings\Administrator 参照(R)...

プロジェクト名のフォルダを作成する(A)

(作成するプロジェクト・ファイルの絶対パスが表示されます)

既存のプロジェクトのファイル構成を流用する(S)

流用元のプロジェクト(P): (流用元のプロジェクト・ファイルを入力してください) 参照(W)...

プロジェクト・フォルダ以下の構成ファイルをコピーして流用する(O)

作成(C) キャンセル ヘルプ(H)

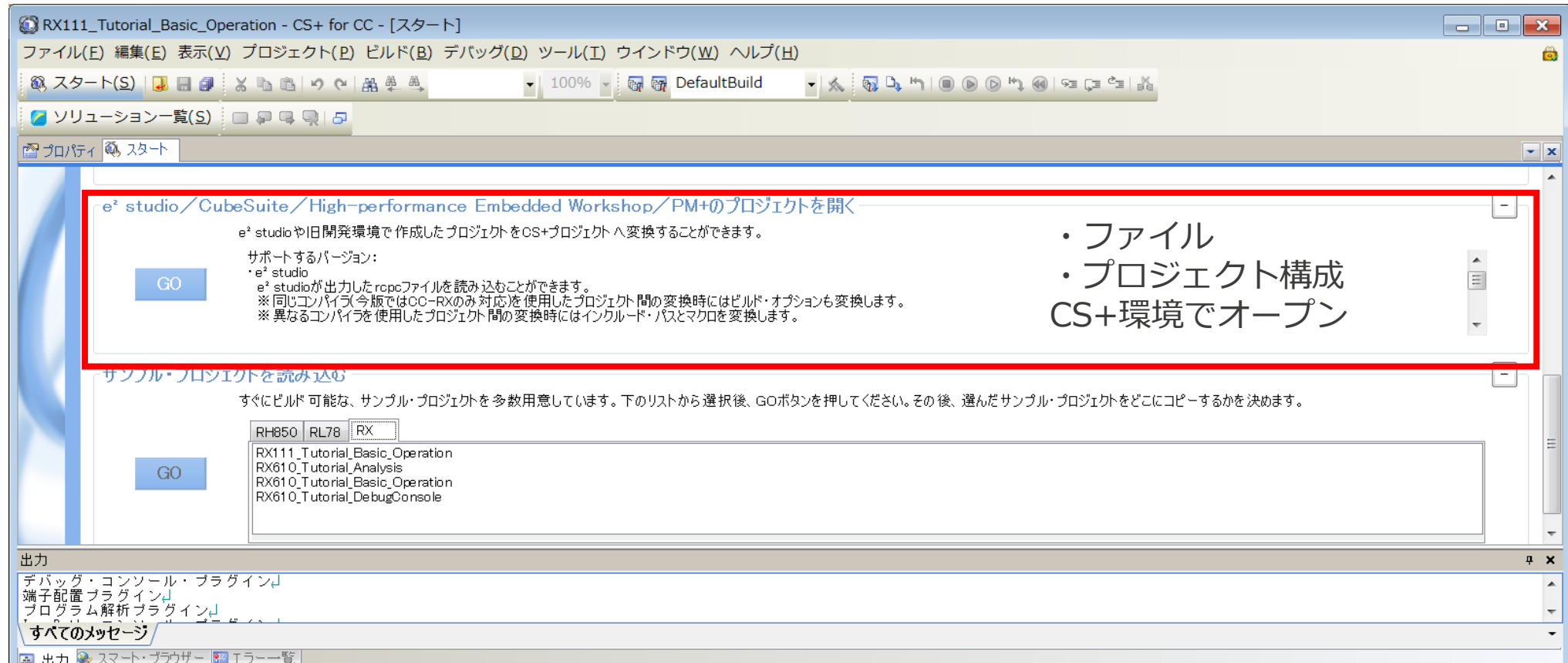
新規プロジェクト作成後、作成済みのV850用のソースファイルを登録して使用する



# 既存プロジェクトの流用2



- PM+, CubeSuiteで作成したワークスペースをCS+環境で読み込み、使用することが可能





# 既存プロジェクトの流用2



- 既存プロジェクトを流用した場合、CPUを選択してプロジェクトを作成

プロジェクト変換設定

プロジェクト (P): sample

CS+のプロジェクトへ変換します

左のプロジェクト一覧よりプロジェクトを選択し、必要であれば変換に関する設定変更を行ってください。[Ctrl]を押しながら選択することで、複数プロジェクトの設定を一度に変えることができます。

変更を行わずに[OK]ボタンを押した場合、読み込んだ旧フォーマットのプロジェクトと同じ場所に新しいプロジェクトを作成し、旧プロジェクトは\*.org付きフォルダにバックアップします。

[注意]  
High-performance Embedded Workshopのプロジェクトへ行ってください。

プロジェクト設定

変換先マイクロコントローラ

マイクロコントローラ(I): RX

使用するマイクロコントローラ(M): RX

品名 R5F51101 AxFK (64pin)  
内蔵ROMサイズ[バイト]32  
内蔵RAMサイズ[バイト]10240  
追加情報 Package=PLQP0064GA-A

変換先プロジェクト

プロジェクトの種類(K): 空のアプリケーション(CO-RX)

プロジェクト名(N): sample

C:\Document and Settings\Administrator

プロジェクト変換情報

プロジェクト変換情報

変換対象デバイスを選択

CS+のプロジェクト

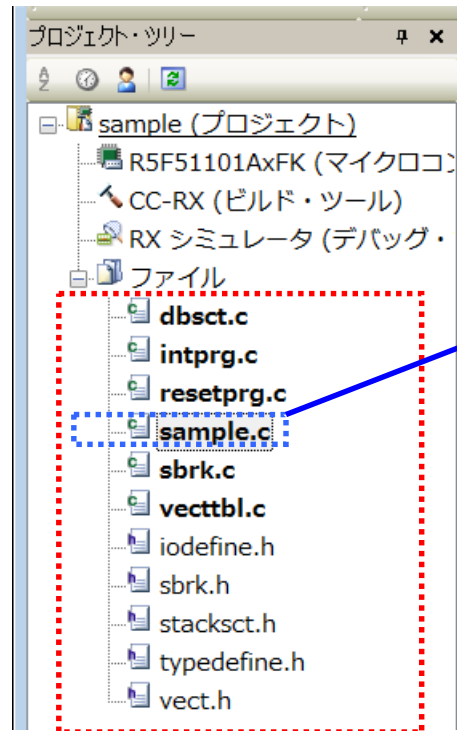
```
1 // .....  
2 // FILE: intprg.c ✓  
3 // DATE: 1983, Jan 28, 2017 ✓  
4 // DESCRIPTION: Intersil Program ✓  
5 // CPU TYPE: 1883A/1527 ✓  
6 // This file is generated by Renesas Project Generator (Ver.4.05). ✓  
7 // .....  
8  
9  
10  
11  
12  
13 #include "teaching.h"  
14 Storage section IntPRG  
15 // vector 0 Reserved  
16  
17 // vector 2 Reserved  
18  
19 // vector 3 Reserved  
20  
21 // vector 4 Illegal code  
22 @__intprgsect__ void __INT__Illegal_code(rol@19 sleep): {}  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100
```

CS+環境で直接対応  
デバイスを選択して変換が可能

# V850用プロジェクトとの違い

# スタートアップファイル

- RX用のプロジェクトを新規に作成された場合、スタートアップファイルを生成します。



プロジェクト名.c内  
お客様に作成して頂く  
main()関数があります

```
void main(void)
{
}

```

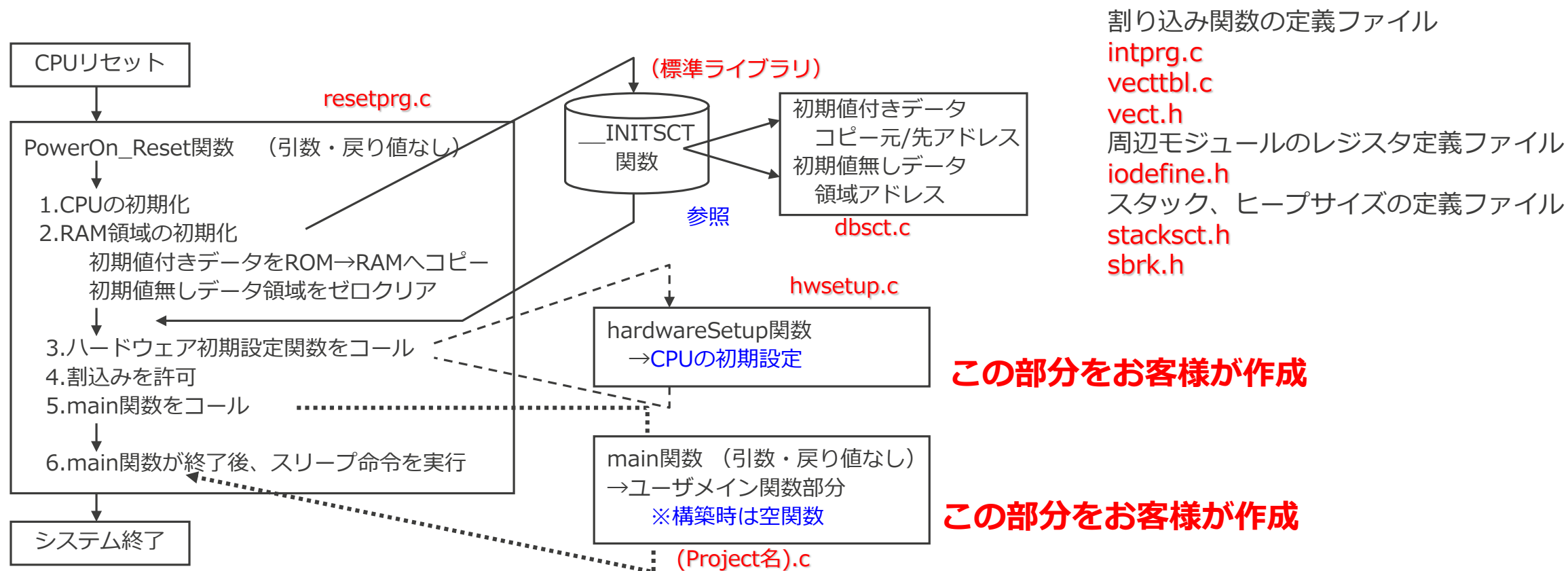
注

V850、RL78、78K0、78K0R用プロジェクトを作成した場合はRXで生成しているスタートアップファイルは生成しません。

既存プロジェクトを流用された場合は、これらのスタートアップファイルを別プロジェクトで生成して頂き、プロジェクトへの登録をしてご使用下さい。

# スタートアップファイル

- プロジェクトツリーに登録されるスタートアップファイルの構成は、以下になります。



# スタートアップファイル

## iodefine.hファイル

- RXの周辺モジュールのレジスタ（SFR）へアクセスするCソースの記述はこのファイル内の宣言を使用することで可能です。

```
<iodefine.h>
...
struct st_tmr0 {
    union {
        unsigned char BYTE;
        struct {
            unsigned char CMIEB:1;
            unsigned char CMIEA:1;
            unsigned char OVIE:1;
            unsigned char CCLR:2;
        } BIT;
    } TCR;
    char wk0[1];
    union {
        unsigned char BYTE;
        struct {
            unsigned char :4;
            unsigned char OSB:2;
            unsigned char OSA:2;
        } BIT;
    } TCSR;
    char wk1[1];
    unsigned char TCORA;
    ...
#define TMR0 (*(volatile struct st_tmr0 __evenaccess *)0x88200)
...

```

```
<レジスタへアクセスするファイル>
#include "iodefine.h"
...
void main(void)
{
    ...
    TMR0.TCR.BYTE = 0x12;
    TMR0.TCSR.BIT.OSB = 0x01;
    TMR0.TCR.BIT.OVIE = 1;
    TMR0.TCORA = 0x12;
    ...
}
...
記述例
```

### <記述方法>

iodefine.hファイル内の記述を使用して

<モジュール名>.<レジスタ名>.<アクセスサイズ>

<モジュール名>.<レジスタ名>.BIT.<ビット名>

<モジュール名>.<レジスタ名>

で周辺モジュールのレジスタへアクセス可能

# スタートアップファイル

## 割込み関数 intprg.c、vect.hファイル

- RXの割込み関数は全て定義済み、生成されたスタートアップに割込み処理を記述するだけで割込み処理の実装が可能（ベクタテーブルは自動生成）

```
<intprg.c>
. . .
// IRQ0
void Excep_IRQ0(void){ }

// IRQ1
void Excep_IRQ1(void){ }
. . .
```

IRQ1の割込みが発生した際に  
実行される関数の実体  
この部分に処理を追記

```
<vect.h>
. . .
// IRQ0
#pragma interrupt (Excep_IRQ0(vect=64))
void Excep_IRQ0(void);

// IRQ1
#pragma interrupt (Excep_IRQ1(vect=65))
void Excep_IRQ1(void);
. . .
```

Excep\_IRQ1()が割込み関数で  
あることを宣言

IRQ1の割込みが発生した際に  
ベクタテーブルの65に格納されている  
Excep\_IRQ1()のアドレス値を参照して  
本関数へ分岐。分岐する際に必要な  
レジスタ退避・回復を実行。

### <割込み関数>

#pragma interruptを使用すると

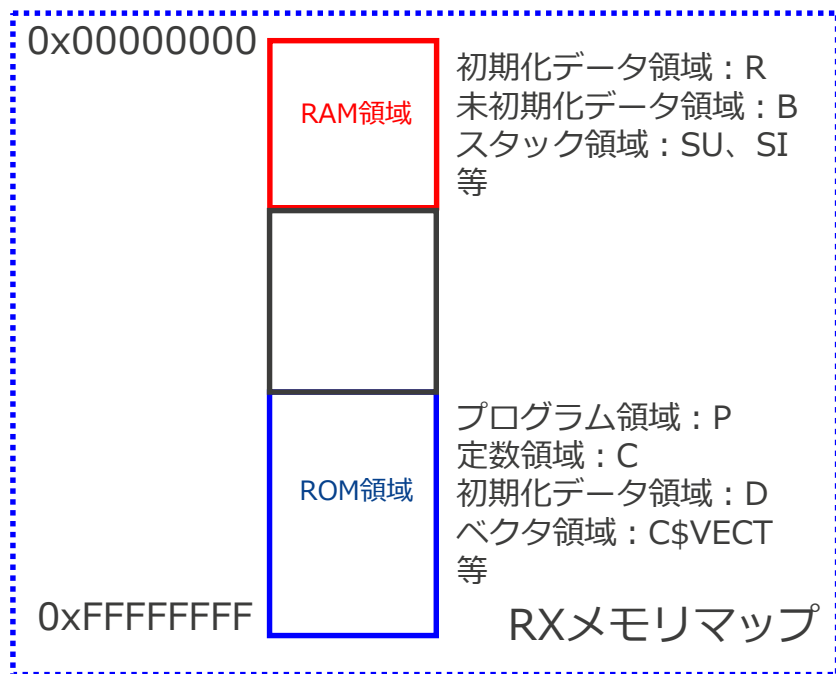
- 割込み関数として宣言
- 必要なレジスタ退避・回復命令を生成
- vect=XXを記述することで割込み関数に必要なベクタテーブルを自動生成

# オプション設定

## ROM、RAM配置

プログラムやデータは、ROMやRAM領域へ配置指定。

CC-RX (ビルド・ツール) プロパティのリンク・オプションタブで設定

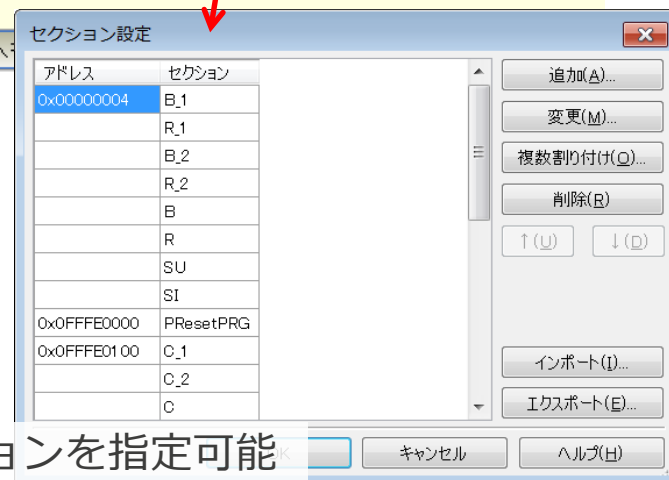


セクション	セクションの開始アドレス	B,1,R,1,B,2,R,2,B,R,SU,SI/04,PRResetPRG/0FFFE0000,C_1,C_2,C,C\$DSEC,C\$BSEC,C\$INIT
外部定義シンボルをファイル出力するセクション		外部定義シンボルをファイル出力するセクション[0]
セクション・アライメント		セクション・アライメント[0]
ROMからRAMへマップするセクション		ROMからRAMへマップするセクション[3]
バリエーション		
その他		

セクション  
共通オプション / コンパイル・オプション / アセンブル・オプション / **リンク・オプション** / ...

配置は、コンパイラが生成するセクション名を指定

使用するCPUのROM/RAM空間に合わせて変更して下さい。



注 RH850のメモリマップ表記は、RXと異なり  
0x00000000と0xFFFFFFFFが逆になっています。

任意のアドレスに任意のセクションを指定可能

# オプション設定 セクション

- RXコンパイラでは、デフォルト名でセクションを生成

表 6.1 メモリ領域の種類とその性質の概要

No.	名称	セクション		形式 種別	初期値 書き込み 操作	アライ メント 数	内容
		名称	属性				
1	プログラム領域	P *1 *6	code	相対	有 不可	1byte *7	機械語を格納
2	定数領域	C *1 *2 *6 *8	romdata	相対	有 不可	4byte	const 型のデータを格納
		C_2 *1 *2 *6 *8	romdata	相対	有 不可	2byte	
		C_1 *1 *2 *6 *8	romdata	相対	有 不可	1byte	
3	初期化データ領域	D *1 *2 *6 *8	romdata	相対	有 可	4byte	初期値のあるデータを格納
		D_2 *1 *2 *6 *8	romdata	相対	有 可	2byte	
		D_1 *1 *2 *6 *8	romdata	相対	有 可	1byte	
4	未初期化データ領域	B *1 *2 *6 *8	data	相対	無 可	4byte	初期値のないデータを格納
		B_2 *1 *2 *6 *8	data	相対	無 可	2byte	
		B_1 *1 *2 *6 *8	data	相対	無 可	1byte	
5	switch 文分岐 テーブル領域	W *1 *2	romdata	相対	有 不可	4byte	switch 文の分岐テーブルを格納
		W_2 *1 *2	romdata	相対	有 不可	2byte	
		W_1 *1 *2	romdata	相対	有 不可	1byte	
6	C++ 初期処理 後処理データ領域	CSINIT	romdata	相対	有 不可	4byte	グローバルクラスオブジェクトに対して呼び出されるコンストラクタおよびデストラクタのアドレスを格納

7	C++ 仮想関数 表領域	CSVTBL	romdata	相対	有 不可	4byte	クラス宣言中に仮想関数があるときに仮想関数をコールするためのデータを格納
8	ユーザスタック領域	SU	data	相対	無 可	4byte	プログラム実行に必要な領域
9	割り込みスタック領域	SI	data	相対	無 可	4byte	プログラム実行に必要な領域
10	ヒープ領域	—	—	相対	無 可	—	ライブラリ関数 malloc、realloc、calloc、new で使用する領域 *9
11	絶対アドレス 変数領域	\$ADDR_ <section>_ <address> *3	data	絶対	有 / 無 可 / 不可 *4	—	#pragma address 指定した変数を格納
12	可変ベクタ領域	CSVECT	romdata	相対	無 可	4byte	可変ベクタテーブル
13	リテラル領域	L *5	romdata	相対	有 可 / 不可	4byte	文字列リテラルおよび集成体の動的初期化で用いる初期化子を格納

本表は、  
コンパイラにより生成される  
デフォルトのセクション名です。

**注**  
RXC V1.01～は、  
文字列リテラルを、  
Lセクションに出力

注 1. section オプションでセクション名を切り替えることができます。

注 2. セクション名切り替えの際に、アライメント数が4のセクションを指定することで、アライメントが1または2のセクション名も変更されます。

注 3. <section> はC,D,Bのセクション名称、<address> は絶対アドレス値(16進数)になります。

注 4. 初期値、書き込み操作は<section>の属性に従います。

注 5. section オプションでセクション名を変更することができます。このとき、変更後の名前にCセクションを選択することも可能です。

注 6. #pragma section でセクション名を変更することができます。

注 7. instalign4 オプション、instalign8 オプション、#pragma instalign4 または #pragma instalign8 のいずれかを使用すると、アライメント数は4または8になります。

注 8. #pragma endian でendian オプションと異なる指定のエンディアンを指定した場合、#pragma endian big であれば\_Bを、#pragma endian little であれば\_Lを、セクション名の後ろに付加した専用のセクションを生成し、該当データを格納します。

注 9. これらの関数を使用するためには、最小で16バイトのヒープ領域が必要です。



# オプション設定

## セクション名の変更 - ソースファイルの全体 -

- RXコンパイラでは、デフォルト名でセクション名を変更可能  
C/C++ソースコード内に#pragmaを使用して、任意のセクション名の付加も可能

CC-RX のプロパティ	
入力プログラムの文字コード	SJIS コード(-sjis)
<b>オブジェクト</b>	
出力ファイル形式	オブジェクト・モジュール・ファイル(-output=obj)
デバッグ情報を出力する	はい(-debug)
プログラム領域のセクション名	P
定数領域のセクション名	C
初期化データ領域のセクション名	D
未初期化データ領域のセクション名	B
リテラル領域のセクション名	L
switch文分岐テーブル領域のセクション名	W
初期値なし変数をアライメント4のセクションに配置する	いいえ
初期値あり変数をアライメント4のセクションに配置する	いいえ
const修飾変数をアライメント4のセクションに配置する	いいえ
switch文分岐テーブルをアライメント4のセクションに配置する	いいえ
分岐先の命令実行向け整合	なし(-noinstalign)
除算、剰余算をDIV, DIVU, FDIV命令で生成する	はい
出力アセンブリ言語ファイルの文字コード	SJISコード(-outcode=sjis)
<b>品質向上関連</b>	
スタック破壊検出を行う	いいえ(なし)

共通オプション / **コンパイル・オプション** / アセンブル・オプション / リンク・オプション / ヘキサ出力オプション / ラ

デフォルトで設定している  
セクション名は任意の名前  
に変更可能

# オプション設定

## セクション名の変更 – ソースファイルの一部 –

- RXコンパイラでは、デフォルト名でセクション名を変更可能  
#pragmaを使用した任意のセクション名の付加も可能

<任意のセクション名>

コンパイラが生成するセクション名以外に任意のセクション名をつけることも可能。

- (1) コンパイラのオプション設定によるデフォルトセクション名の設定
- (2) 標準ライブラリのオプション設定によるデフォルトセクション名の設定
- (3) Cソース内へのセクション設定

例)

```
#pragma section AAA
int data1;           // BAAAセクションを生成
int data2=2;        // DAAAセクションを生成

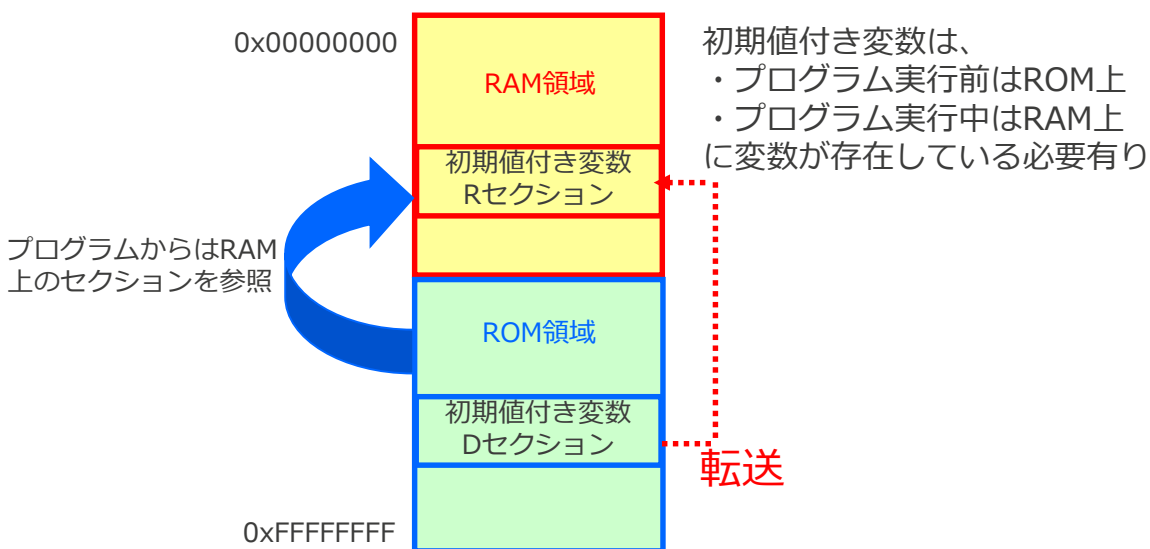
void func(void)     // PAAAセクションを生成
{
    data1++;
    data2++;
}
#pragma section    // 以降は、デフォルト設定名
```

#pragmaで指定した任意のセクション名 (AAA) の前にデフォルトセクション名 (B,D,P等) を付加したセクションを生成

このBAAA、DAAA、PAAAセクションを最適化リンカで配置指定

# オプション設定

## ROMからRAMへの転送



例)  
int data1 = 1; // 初期値1を持つ変数data1は、ROM上のDセクション  
void func()  
{  
  data1++; // プログラム実行中のdata1は、RAM上のRセクション  
}

最適化リンカの“ROMからRAMへマップするセクション”のオプションでは、プログラム実行中はRAM上に配置されたセクションの変数を参照するようにアドレス解決。

転送は、プロジェクトを作成し生成されるスタートアップコード内で自動的に実施。

```
<resetprg.c>
...
void PowerON_Reset_PC(void)
{
    set_intb((unsigned long)__sectop("C$VECT"));
    set_fpsw(FPSW_init);

    _INITSCT(); // 本関数内で転送を実施
    ...
}

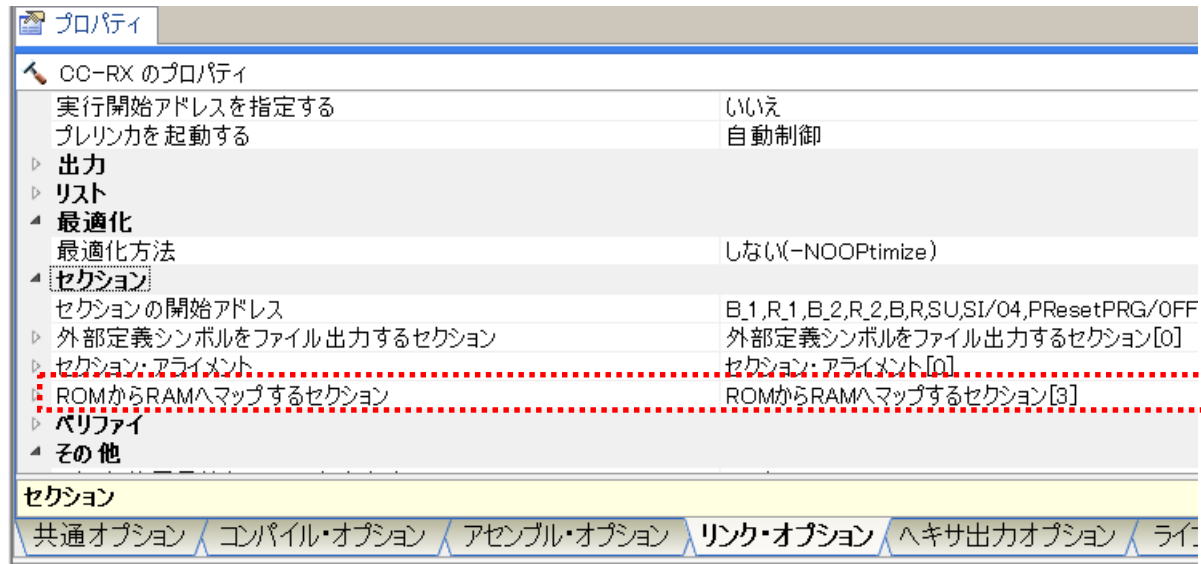
<dbsect.c>
...
#pragma section C C$DSEC
extern const struct {
    _UBYTE *rom_s; /* 初期化データセクションのROM 上の先頭アドレス */
    _UBYTE *rom_e; /* 初期化データセクションのROM 上の最終アドレス */
    _UBYTE *ram_s; /* 初期化データセクションのRAM 上の先頭アドレス */
} _DTBL[] = {
    { __sectop("D"), __sectend("D"), __sectop("R") },
    { __sectop("D_2"), __sectend("D_2"), __sectop("R_2") }, 転送アドレスを_INITSCT()へ渡す
    { __sectop("D_1"), __sectend("D_1"), __sectop("R_1") }
};
...
```

ROMからRAMへマップするセクションを追加した場合、本ファイルのこの部分にも追記が必要

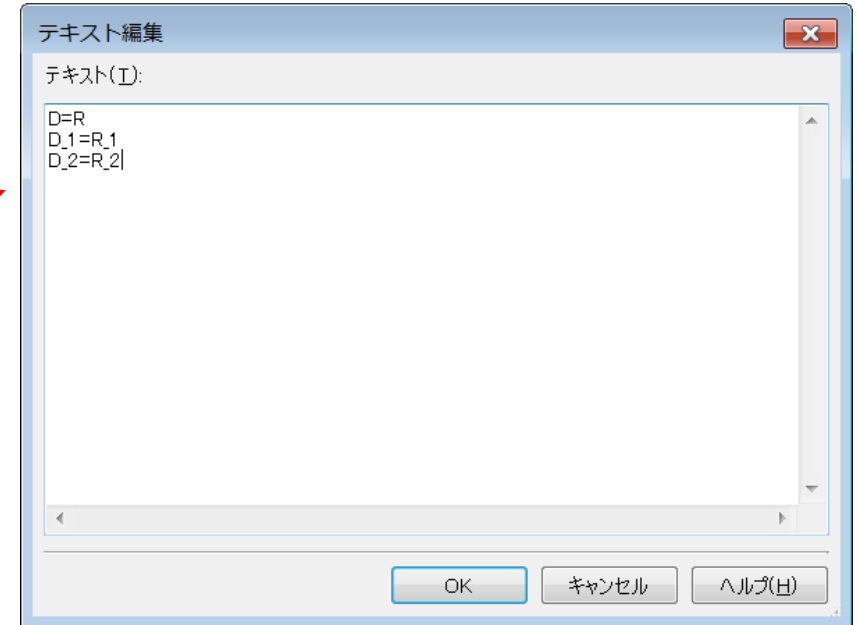
# オプション設定

## ROM化支援オプション

- 最適化リンカのromオプションを使用することでアドレス解決



この設定は転送をするわけではありません。  
アドレス解決(RAM側のアドレスを参照)  
するためのオプション設定です。



新規に初期値付き変数のセクション (D\*) を作成した場合、  
このROMからRAMへマップするセクション (romオプション)  
の設定とdbsct.hファイルに追記することでROM化をサポート  
(初期値付き変数のROMからRAMへの転送を実行)

# オプション設定

## 未初期化領域の初期化

- 未初期化領域Bセクションは、最適化リンカの-startオプションで配置指定をしています。  
#pragma sectionで任意の名前にしたセクションに対して実際に初期化を行うには、プロジェクト作成し生成されるスタートアップコードに追記が必要です。

初期化は、プロジェクトを作成し生成されるスタートアップコード内で自動的に実施。

```

<resetprg.c>                                <dbsect.c>
. . .                                         . . .
void PowerON_Reset_PC(void)                  };
{                                             #pragma section C C$BSEC
    set_intb((unsigned long)__sectop("C$VECT"))extern const struct {
    set_fpsw(FPSW_init);                      _UBYTE *b_s; /* 未初期化データセクションの先頭アドレス */
                                             _UBYTE *b_e; /* 未初期化データセクションの最終アドレス */
    _INITSCT(); // 本関数内で初期化を実施    } _BTBL[] = {
    . . .                                     { __sectop("B"), __sectend("B") }, 初期化アドレスを_INITSCT()へ渡す
                                             { __sectop("B_2"), __sectend("B_2") } 未初期化領域のセクションを追加した場合、
                                             { __sectop("B_1"), __sectend("B_1") } 本ファイルのこの部分にも追記が必要
    };
    . . .

```

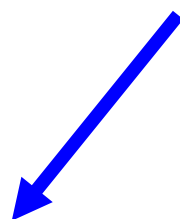
# オプション設定

## スタック領域

- RXのスタック領域は、stacksct.hファイルとCC-RX（ビルド・ツール）プロパティのリンク・オプションタブの設定で実現

```
<stacksct.h>  
#pragma stacksize su=0x100  
#pragma stacksize si=0x300
```

スタックサイズの指定



```
MVTC #00001504,USP  
MVTC #00001804,ISP
```

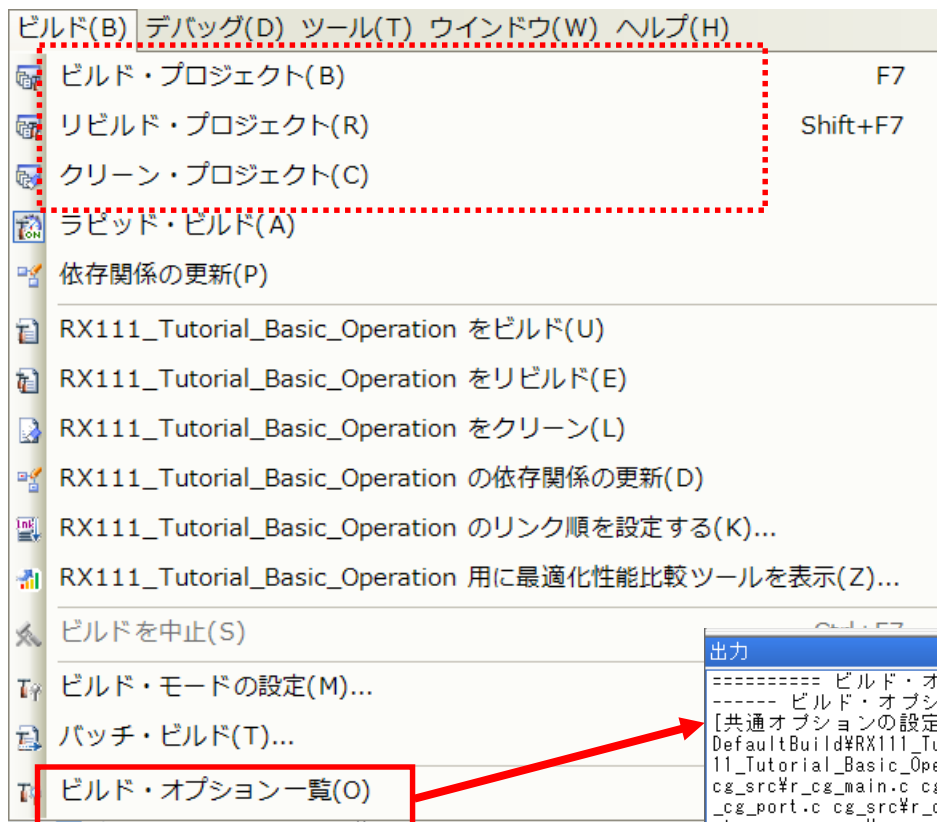
ビルド実行により自動的に  
コードを生成

最適化	
最適化方法	しない(-NOOptimize)
セクション	
セクションの開始アドレス	B_1,R_1,B_2,R_2,B,R,SU,SI/04,PRResetPRG/0FFFE0000,C_1,C_2,C,C\$DSEC,C\$BSEC
外部定義シンボルをファイル出力するセクション	外部定義シンボルをファイル出力するセクション[0]
セクション・アライメント	セクション・アライメント[0]
ROMからRAMへマップするセクション	ROMからRAMへマップするセクション[3]
パリアファイ	
その他	

SU、SIのセクションが配置されたアドレス  
+  
stacksct.hファイルで指定したサイズ  
が指すアドレスがスタックポインタの  
初期値として設定されるコードを生成

アドレス	セクション
0x00000004	B_1
	R_1
	B_2
	R_2
	B
	R
	SU
	SI
0x0FFFE000	PRResetPRG
0x0FFFE0100	C_1
	C_2
	C

# オプション設定 ビルド実行

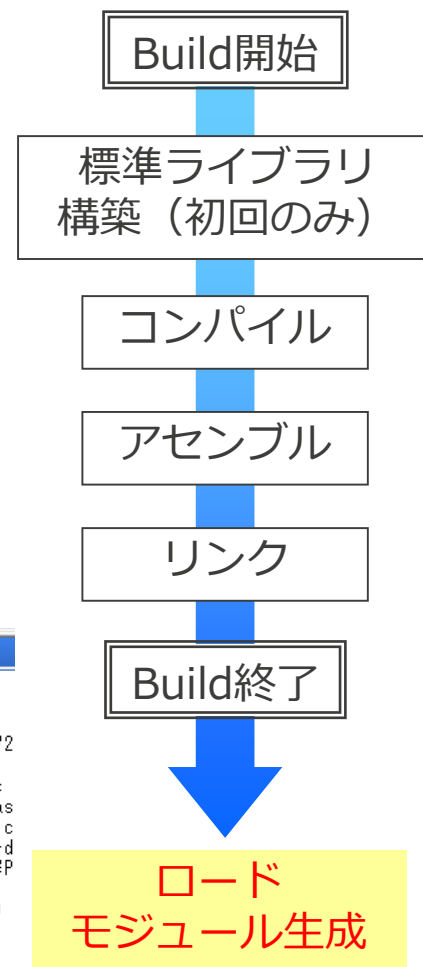


ビルドを選択

- ・ビルド・プロジェクト  
： 変更のあったファイルがビルド対象
- ・リビルド・プロジェクト  
： 全てビルド対象
- ・クリーン・プロジェクト  
： 削除した上で全てビルド対象

設定オプションの一覧の表示も可能

RXコンパイラのビルドフェーズ



# 最適化オプションの設定方法

---

Renesas製 コンパイラ、リンカの最適化技術を適用し、RXマイコンに適した更なる最適化へ強化（最適なレジスタ割り付け、最適命令選択、命令スケジューリング等）、コンパクトなコードを生成

## 1、コンパイラの最適化

### (1)最適化の簡単選択

- Size or Speed優先の選択
- 最適化レベル（-optimize=0 or 1 or 2 or max）の選択

### (2)コンパイル時の広範囲最適化

- ファイル間にまたがる関数インライン展開
- 最適化リンカの外部シンボル割り付け情報を用いたシンボルアクセス最適化

### (3)最適化の詳細設定

- ループ展開、命令並び替え、最適化範囲の選択、レジスタ割り付け etc

## 2、最適化リンカの最適化

### (1)モジュール間の最適化

- 未参照シンボルの削除、共通コードの統合、分岐命令の最適化、アドレッシングモードの短縮
- 最適化抑止の詳細設定

## 3、標準ライブラリの最適化

### (1)コンパイラ最適化の適用

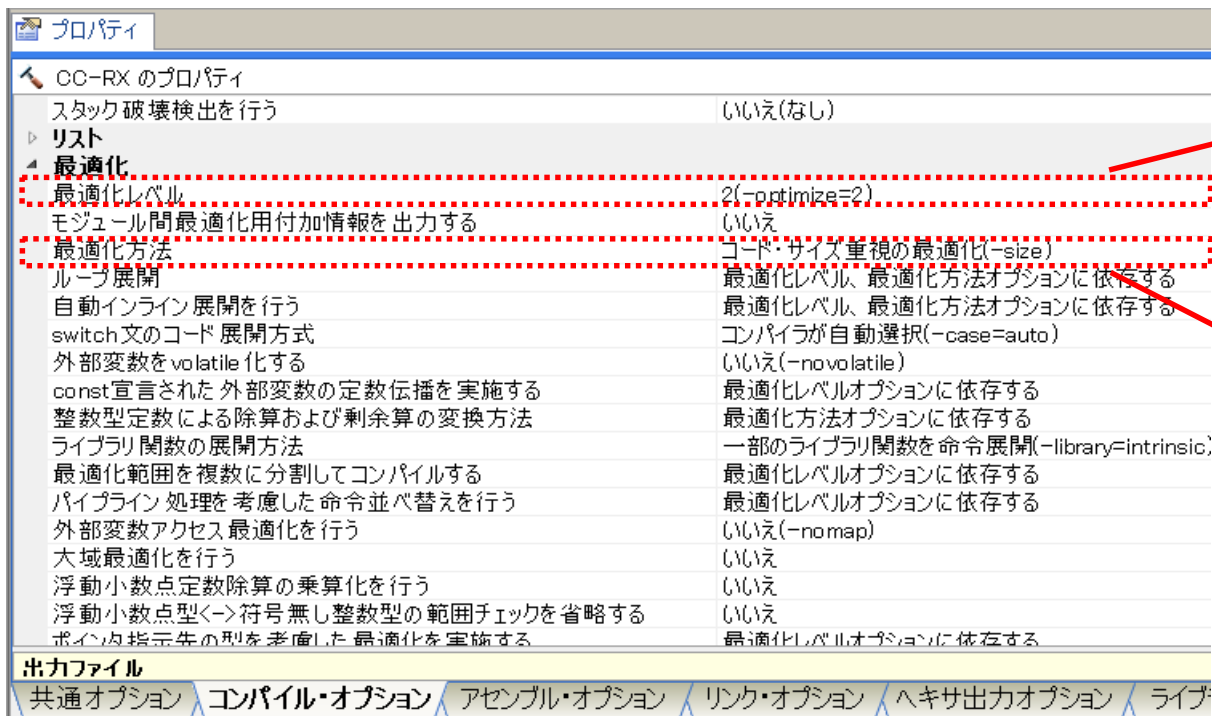
- 標準ライブラリも最適化の対象、プログラム全体でコンパクトなコードを生成



# 最適化オプションの設定方法

## オプション設定

- CC-RX (ビルド・ツール) プロパティのコンパイル・オプションタブで設定



0(-optimize=0)  
1(-optimize=1)  
2(-optimize=2)  
Max(-optimize=max)

実行性能重視の最適化(-speed)  
コード・サイズ重視の最適化(-size)

実行性能重視の最適化、コード・サイズ重視の最適化と  
最適化レベル 0 or 1 or 2 or max  
の選択により最適化の実施が可能

更に詳細のチューニングも可能

# 最適化オプションの設定方法

## 最適化の詳細

- ROMサイズ優先か実行スピード優先、レベル (0,1,2,max) の最適化を実施するかの選択で設定  
 <optimize=max 指定時> さらに詳細設定でチューニングも可能

	ループ展開	インライン展開	定数除算の乗除化	命令並び替え	Const修飾変数	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=8	Inline=250	const_div	schedule	const_copy	noscope	map,nomap	alias=ansi
size	loop=1	Inline=0	nocost_div	schedule	const_copy	noscope	map,nomap	alias=ansi

<optimize=2 指定時>

	ループ展開	インライン展開	定数除算の乗除化	命令並び替え	Const修飾変数	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=2	Inline=100	const_div	schedule	const_copy	scope	nomap	alias=noansi
size	loop=1	noInline	nocost_div	schedule	const_copy	scope	nomap	alias=noansi

<optimize=0 または optimize=1 指定時>

	ループ展開	インライン展開	定数除算の乗除化	命令並び替え	Const修飾変数	最適化範囲分割	外部変数アクセス最適化	ポインタ指示先の型を考慮した最適化
speed	loop=1	noInline	const_div	noschedule	noconst_copy	scope	nomap	alias=noansi
size	loop=1	noInline	nocost_div	noschedule	noconst_copy	scope	nomap	alias=noansi

オプション設定内容の詳細は、コンパイラユーザーズマニュアルをご参照下さい。

# 最適化オプションの設定方法

## 外部変数最適化

大域変数は、言語仕様上、必ずメモリに割り付けられ、そのアクセスには #IMM:32 のアドレスロード命令が必要となる。外部変数最適化では

- 最適化リンケージエディタが生成する外部シンボル割り付け情報を元にベースアドレスを設定し、外部変数もしくは静的変数のアクセスをベースアドレス相対で行うコードを生成します。
- コンパイル対象ファイル内で定義された外部変数もしくは静的変数についてベースアドレスを設定し、アクセスをベースアドレス相対で行うコードを生成します。

最適化未実施コード

```

short a, b;
int c;
extern short x, y;
extern int z;

void main(void)
{
    c = a + b;
    z = x + y;
}
CODE LABEL INSTRUCTION OPERAND
FB42rrrrrrr _main: MOV.L #_a,R4
FB32rrrrrrr MOV.L #_b,R3
DC45 MOV.W [R4],R5
DC34 MOV.W [R3],R4
FB32rrrrrrr MOV.L #_y,R3
4B54 ADD R5,R4
FB52rrrrrrr MOV.L #_c,R5
E354 MOV.L R4,[R5]
FB42rrrrrrr MOV.L #_x,R4
DC45 MOV.W [R4],R5
DC34 MOV.W [R3],R4
4B54 ADD R5,R4
FB52rrrrrrr MOV.L #_z,R5
E354 MOV.L R4,[R5]
02 RTS
    
```

最適化 (モジュール間) 実施コード

```

CODE LABEL INSTRUCTION OPERAND
FB22rrrrrrr _main: MOV.L #_a,R2
DC25 MOV.W [R2],R5
FB12rrrrrrr MOV.L #_c,R1
06492501 ADD 02H[R2].W,R5
E315 MOV.L R5,[R1]
98A5 MOV.W 04H[R2],R5
06492503 ADD 06H[R2].W,R5
A01D MOV.L R5,04H[F
02 RTS
    
```

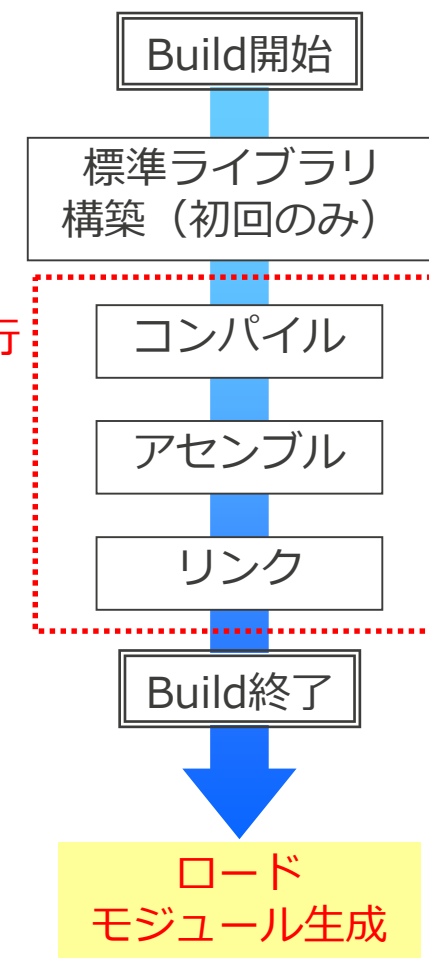
モジュール内の効果

モジュール間の効果

	スピード	サイズ
最適化未実施	16	53
最適化実施 モジュール間	13	29

a,b : モジュール内の最適化対象(-smap、-map指定時)  
x,y : モジュール間の最適化対象(-map指定時)

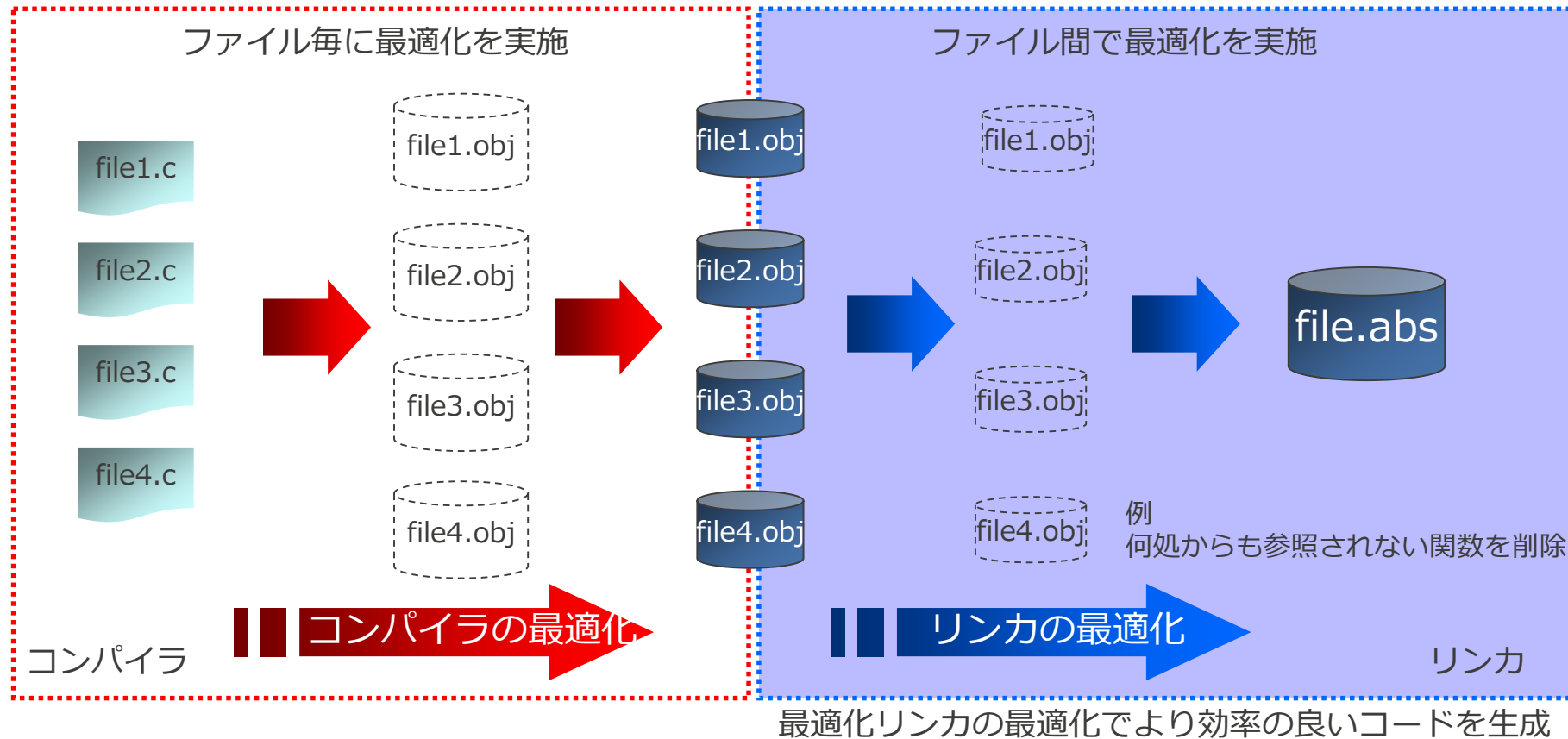
RXコンパイラのビルドフェーズ



# 最適化オプションの設定方法

## RXコンパイラの最適化（最適化リンカ）

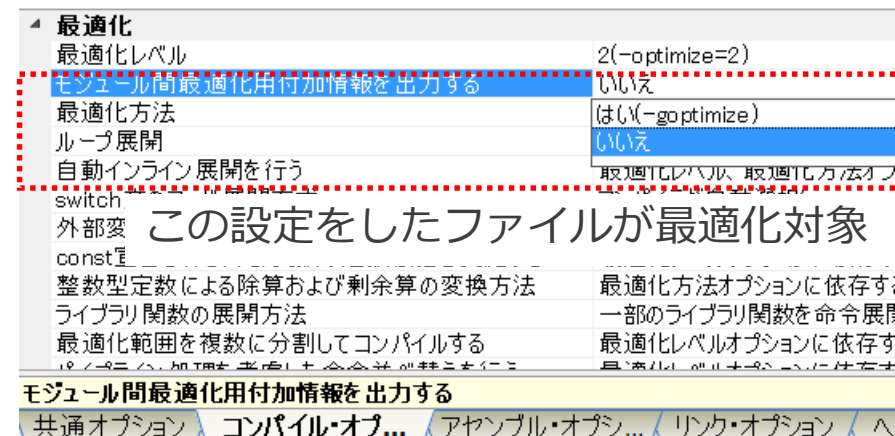
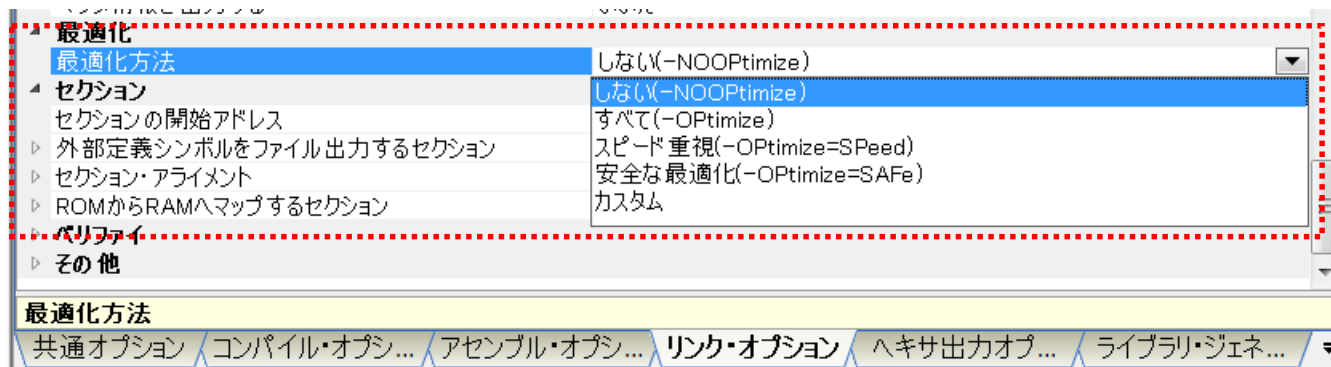
- RXのビルド環境では、コンパイラの最適化のほかにリンカにも最適化有りモジュール間（\*.cファイル間）の最適化を実施しより効率の良いコードを生成



# 最適化オプションの設定方法

## オプション設定

- CC-RX（ビルド・ツール）プロパティのリンク・オプションタブで設定



注 リンカの設定のみでは本最適化は実施しない  
コンパイラの“モジュール間最適化用付加情報を出力”を  
“はい”にする必要有り

### リンク・オプション設定 最適化方法

	全て	スピード重視	安全な最適化	カスタム
未参照シンボルを削除する	○	○	○	任意
複数の同一命令列をサブルーチン化する	○	-	-	任意
コードサイズがより小さくなる命令に置き換える	○	○	-	任意
分岐命令サイズを最適化する	○	○	○	任意

---

ルネサス システムデザイン株式会社