RH850 開発環境移行ガイド

V850E2 から RH850 への移行(コンパイラ編)(CX→CC-RH)

R20UT3233JJ0101
Rev.1.01
2017.04.21

要旨

本書では、V850E2 用コンパイラ（以降、CX）から、RH850 ファミリ用コンパイラ（以降、CC-RH）へ移行する際に主に注意すべき点について説明します。

目次

1. オプション

本章では、CX のオプションに相当する CC-RH のオプションを比較表にて示します。なお、CC-RH では、コンパイル・オプション、アセンブル・オプションの大文字／小文字は区別しますが、リンク・オプションの大文字と小文字は区別しません。

1.1 コンパイル･オプション

分類	説明	CX（cx.exe）	CC-RH（ccrh.exe）
バージョン／ヘルプ表示指定	バージョン情報を表示	-V	-V
	オプション情報を表示	-h	-h
出力ファイル指定	出力ファイル名を指定	-o	-o
	オブジェクト・ファイルの保存先を指定	-Xobj_path	-Xobj_path
	アセンブリ・ファイルの保存先を指定	-Xasm_path	-Xasm_path
	アセンブル・リスト・ファイルの保存先を指定	-Xprn_path	-Xasm_option=-Xprn_path
	作業用フォルダを指定	-Xtemp_path	なし
	ROM 化処理前のロード・モジュール・ファイルを保存	-Xlink_output	なし（注1）
ソース・デバッグ制御	デバッグ情報を出力	-g	-g
	メモリアクセスサイズの変更を禁止	-Xkeep_access_size	なし
デバイス指定	ターゲット・デバイスを指定	-C	なし（注2）
	デバイス共通のオブジェクト・ファイルを生成	-Xcommon	-Xcommon（注3）
	デバイス・ファイルを検索するフォルダを指定	-Xdev_path	なし（注2）
	プログラマブル周辺I/Oレジスタの開始アドレスを指定	-Xprogrammable_io	なし
処理中断処理	プリプロセス処理まで実行	-P	-P
	コンパイル処理まで実行	-S	-S
	アセンブル処理まで実行	-c	-c
プリプロセッサ制御	プリプロセッサ・マクロを定義	-D	-D
	プリプロセッサ・マクロの定義を解除	-U	-U
	インクルード・ファイルを検索するフォルダを指定	-I	-I
	プリプロセス結果の出力を制御	-Xpreprocess	-Xprerocess
C言語制御	ANSI 規格に厳密にあわせて処理	-Xansi	-Xansi
	char型に対して符号を指定	-Xchar	なし（注4）
	列挙型に対してどの整数型として扱うかを指定	-Xenum_type	-Xenum_type
	変数の仮定義を定義	-Xdef_var	なし（注5）
変数／関数情報指定	シンボル情報ファイルを利用	-Xsymbol_file	なし
日本語／中国語文字列制御	日本語／中国語の文字コードを指定	-Xcharacter_set	-Xcharacter_set

注 1：CC-RH はデフォルトで ROM 化しません。リンク時に-ROm オプションの指定が必要です。
注 2：CC-RH はデバイス・ファイルをサポートしていません。
注 3：指定可能な引数は異なります。
注 4：CC-RH は符号が付かない単なる char 型に対して常に符号付きとして扱います。
注 5：CC-RH は変数の仮定義を常に定義として扱います。

分類	説明	CX（cx.exe）	CC-RH（ccrh.exe）
最適化指定	最適化のレベルまたは各最適化項目の詳細を指定	-O	-O（注6）
	未使用関数を削除	-Xdelete_func	なし
	外部変数をソート	-Xsort_var	なし
	標準ライブラリ関数strcpy，strcmp，memcpy，memset の呼び出しをインライン展開	-Xinline_strcpy	-Xinline_strcpy
	プロローグ／エピローグ処理の指定	-Xpro_epi_runtime	なし
	ライブラリ関数のインライン展開を抑止	-Xcall_lib	なし
	文字列定数をマージ	-Xmerge_string	-Xmerge_string
生成コード制御	構造体パッキング	-Xpack	-Xpack（注7）
	アセンブラ・ソース・ファイル中にCソース・プログラムをコメントとして出力	-Xpass_source	-Xpass_source
	switch 文のコード出力方式を指定	-Xswitch	-Xswitch
	4 バイトの分岐テーブルを生成	-Xword_case	なし
	外部変数の割り付け先のレジスタを指定	-Xr	なし
	レジスタ・モードを指定	-Xreg_mode	-Xreg_mode（注8）
	.sdata/.sbssセクションに配置するデータの最大サイズを指定	-Xsdata	-Xsection（注9）
	.sconstセクションへの定数データの配置を指定	-Xsconst	
	浮動小数点演算命令の生成を制御	-Xfloat	-Xfloat
	far jumpの出力を制御	-Xfar_jump	-Xfar_jump
	除算に対してdivおよびdivu命令を生成	-Xdiv	-Xdiv
アセンブラ制御指定	アセンブラ・ソース・ファイルに対してfar jumpの出力を制御	-Xasm_far_jump	-Xasm_option=-Xasm_far_jump
ライブラリ・リンク制御	リンクするライブラリ・ファイルを指定	-l	-Xlk_option=-LIBrary
	ライブラリ・ファイルを検索するフォルダを指定	-L	なし（注11）
	標準ライブラリのリンクを抑止	-Xno_stdlib	なし（注12）
	標準スタートアップ・ルーチンのリンクを抑止	-Xno_startup	なし（注13）
	スタートアップ・ルーチンを指定	-Xstarup	なし（注13）

注 6 ：指定可能な引数は異なります。
注 7 ：CC-RH は引数に 8 を指定出来ません。
注 8 ：CC-RH には 26 レジスタ・モードはありません。
注 9 ：CC-RH V1.02.00 で追加したオプションです。変数のデフォルト配置セクションを一括変更します。
注 10：CX/CC-RH ともデフォルトで定数データを.const セクションに配置します。
注 11：-Xlk_option=-LIBrary の引数としてライブラリ・ファイルを検索するフォルダを指定します。
注 12：CC-RH はデフォルト標準ライブラリをリンクしません。
注 13：CC-RH はスタートアップ・ルーチンを通常のソース・ファイルとして扱います。

分類	説明	CX（cx.exe）	CC-RH（ccrh.exe）
ROM化制御	ROM 化処理を抑止	-Xno_romize	なし（注14）
	ROM 化用領域確保コード・ファイルを指定	-Xrompcrt	なし
	rompsec セクションの先頭アドレスを指定	-Xrompsec_start	なし
	rompsec セクションに含めるデータ・セクションを指定	-Xrompsec_data	なし
	rompsec セクションに含めるテキスト・セクションを指定	-Xrompsec_text	なし
	rompsec セクションのみを持つロード・モジュール・ファイルを生成	-Xrompsec_only	なし
	ROM 化時のエラー・チェックを省略	-Xromize_check_off	なし
リンク制御	リンク・ディレクティブ・ファイルを指定	-Xlink_directive	なし（注15）
	リンク・マップ・ファイルを出力	-Xmap	-Xlk_option=-LISt
	リンク・マップ・ファイルにシンボル情報を出力	-Xsymbol_dump	-Xlk_option=-SHow
	セキュリティID を設定	-Xsecurity_id	なし（注16）
	ユーザ・オプション・バイトを設定	-Xoprion_byte	なし（注16）
	エントリ・ポイント・アドレスを指定	-Xentry_address	-Xlk_option=-ENTry
	再配置可能なオブジェクト・モジュール・ファイルを生成	-Xrelinkable_object	-Xlk_option= -FOrm=Relocate
	異なるレジスタ・モードが混在している場合に詳細情報を出力	-Xregmode_info	なし
	内部ROM/RAM のオーバフロー時にリンク処理を続行	-Xforce_link	なし（注17）
	-Xsdataオプションのパラメータに対して目安として用いることのできる情報を標準出力に出力	-Xsdata_info	なし
	2 パス・モードでリンク	-Xtwo_pass_link	なし
	リンク時のリロケーション処理において不正箇所があった場合、リンク処理を続行	-Xignore_address_error	なし（注17）
	多重定義されたすべての外部シンボルに対してエラー・メッセージを出力	-Xmultiple_symbol	なし（注17）
	リンク時のチェックを抑止	-Xlink_check_off	なし（注17）
	アライン・ホールの充てん値を指定	-Xalign_fill	なし（注18）
	-l オプションで指定したライブラリ・ファイルの再スキャン	-Xrescan	なし
	デバッグ情報等を削除したロード・モジュール・ファイルを生成	-Xstrip	-Xlk_option=-NODEBug

注 14：CC-RH のリンカはデフォルトで ROM 化しません。リンク時に-ROm オプションの指定が必要です。
注 15：CC-RH はリンク・ディレクティブ・ファイルをサポートしていません。
　　セクションの配置アドレスはリンカの-STARt オプションで指定します。
注 16：RH850 のセキュリティ ID・オプションバイトはフラッシュライタ等で指定します。
注 17：CC-RH のリンカの-CHange_message オプションを指定して、
　　エラーを警告に変更して処理を継続することは可能です。
注 18：CC-RH のリンカの-PADDING、-Space 等により一部代用可能です。

分類	説明	CX（cx.exe）	CC-RH（ccrh.exe）
ヘキサ出力制御	ヘキサ・ファイル名を指定	-Xhex	-Xlk_option=-OUtput
	ヘキサ出力のみ実行	-Xhex_only	-Xlk_option=-FOrm
	ヘキサ・ファイルのフォーマットを指定	-Xhex_format	-Xlk_option=-FOrm
	ヘキサ・ファイルの充てん処理を指定	-Xhex_fill	-Xlk_option=-SPace
	指定セクションをヘキサ変換して出力	-Xhex_section	-Xlk_option=-OUtput
	ブロック長の最大値を指定	-Xhex_block_size	-Xlk_option=-BYte_count
	出力するアドレスのオフセットを指定	-Xhex_offset	なし
	初期値なしデータのセクションに対してnull文字を生成	-Xhex_null	なし
	シンボル・テーブルを変換して出力	-Xhex_symtab	なし
	ヘキサ・ファイルの充てん時に内蔵ROM領域情報を未使用	-Xhex_rom_less	なし
	CRC 演算結果を出力	-Xcrc	なし
	CRC 演算方法を指定	-Xcrc_method	なし
情報ファイル出力制御	静的解析情報ファイルを出力	-Xcref	-Xcref
	静的解析情報ファイルの出力を抑止	-Xno_cref	なし
	シンボル情報ファイルを生成	-Xsfg	なし
	最適な配置情報を出力	-Xsfg_opt	なし
	.tidataセクションのサイズを指定	-Xsfg_size_tidata	なし
	.tidata.byte セクションのサイズを指定	-Xsfg_size_tidata_byte	なし
	.sidataセクションのサイズを指定	-Xsfg_size_sidata	なし
	.sedataセクションのサイズを指定	-Xsfg_size_sedata	なし
	.sdataセクションのサイズを指定	-Xsfg_size_sdata	なし
エラー出力制御	エラー・メッセージをファイルに出力	-Xerror_file	-Xerror_file
警告メッセージ出力制御	指定した警告メッセージを出力	-Xwarning	なし
	指定した警告メッセージの出力を抑止	-Xno_warning	-Xno_warning
フェーズ個別オプション指定	アセンブル対象ファイルを指定	-Xasm_option	-Xasm_option
	リンク対象ファイルを指定	-Xlk_option	-Xlk_option
	共通最適化部オプションを指定	-Xopt_option	なし
コマンド・ファイル指定	コマンド・ファイルを指定	@	@

1.2　アセンブル･オプション

分類	説明	CX（cx.exe）	CC-RH（asrh.exe）
バージョン／ヘルプ表示指定	バージョン情報を表示	-V	-V
	オプション情報を表示	-h	-h
出力ファイル指定	出力ファイル名を指定	-o	-o
	オブジェクト・ファイルの保存先を指定	-Xobj_path	-Xobj_path
	アセンブル・リスト・ファイルの保存先を指定	-Xprn_path	-Xprn_path
ソース・デバッグ制御	デバッグ情報を出力	-g	-g
デバイス指定	ターゲット・デバイスを指定	-C	なし（注1）
	デバイス共通のオブジェクト・ファイルを生成	-Xcommon	-Xcommon（注2）
	デバイス・ファイルを検索するフォルダを指定	-Xdev_path	なし（注1）
	プログラマブル周辺I/Oレジスタの開始アドレスを指定	-Xprogrammable_io	なし
プリプロセッサ制御	アセンブラ・シンボルを定義	-D	-D
	アセンブラ・シンボルの定義を解除	-U	-U
	インクルード・ファイルを検索するフォルダを指定	-I	-I
日本語／中国語文字列制御	日本語／中国語の文字コードを指定	-Xcharacter_set	-Xcharacter_set
生成コード制御	レジスタ・モードを指定	-Xreg_mode	-Xreg_mode（注3）
	.sdata/.sbss セクションに配置するデータの最大サイズを指定	-Xsdata	なし（注4）
アセンブラ制御指定	アセンブラ・ソース・ファイルに対してfar jumpの出力を制御	-Xasm_far_jump	-Xasm_far_jump
エラー出力制御	エラー・メッセージをファイルに出力	-Xerror_file	-Xerror_file
警告メッセージ出力制御	指定した警告メッセージを出力	-Xwarning	なし
	指定した警告メッセージの出力を抑止	-Xno_warning	-Xno_warning
サブコマンド・ファイル指定	コマンド・ファイルを指定	@	@

注 1 ：CC-RH はデバイス・ファイルをサポートしていません。
注 2 ：指定可能な引数は異なります。
注 3 ：CC-RH には 26 レジスタ・モードはありません。
注 4 ：CX はデフォルトで変数を.sdata/.sbss セクションに、CC-RH は.data/.bss セクションに配置します。

1.3　リンク･オプション

分類	説明	CX（cx.exe）	CC-RH（rlink.exe）
バージョン／ヘルプ表示指定	バージョン情報を表示	-V	なし (注 1)
	オプション情報を表示	-h	なし (注 1)
出力ファイル指定	出力ファイル名を指定	-o	-output
	作業用フォルダを指定	-Xtemp_path	なし
デバッグ制御	デバッグ情報を出力	-g	-debug
デバイス指定	ターゲット・デバイスを指定	-C	なし (注2)
	デバイス・ファイルを検索するフォルダを指定	-Xdev_path	なし (注2)
ライブラリ・リンク制御	リンクするライブラリ・ファイルを指定	-l	-library
	ライブラリ・ファイルを検索するフォルダを指定	-L	なし
	標準ライブラリのリンクを抑止	-Xno_stdlib	なし (注3)
リンク・ディレクティブ・ファイル	リンク・ディレクティブ・ファイルを指定	-Xlink_directive	なし (注4)
セキュリティID	セキュリティID を設定	-Xsecurity_id	なし (注5)
オプションバイト制御	ユーザ・オプション・バイトを設定	-Xoprion_byte	なし (注5)
リンク処理続行	内部ROM/RAM のオーバフロー時にリンク処理を続行	-Xforce_link	なし (注6)
エントリ・ポイント・アドレス	エントリ・ポイント・アドレスを指定	-Xentry_address	-entry
リンク・マップ・ファイル	リンク・マップ・ファイルを出力	-Xmap	-list
シンボル情報	リンク・マップ・ファイルにシンボル情報を出力	-Xsymbol_dump	-show
オブジェクト・モジュール・ファイル制御	再配置可能なオブジェクト・モジュール・ファイルを生成	-Xrelinkable_object	-form=relocate

注 1 ：CC-RH はコマンドライン上で rlink[ENTER]の実行で表示されます。
注 2 ：CC-RH はデバイス・ファイルをサポートしていません。
注 3 ：CC-RH はデフォルト標準ライブラリをリンクしません。
注 4 ：CC-RH はリンク・ディレクティブ・ファイルをサポートしていません。
　　セクションの配置アドレスはリンカの-start オプションで指定します。
注 5 ：RH850 のセキュリティ ID・オプションバイトはフラッシュライタ等で指定します。
注 6 ：リンカの-change_message オプションを指定して、エラーを警告に変更して処理を継続可能です。

分類	説明	CX（cx.exe）	CC-RH（rlink.exe）
混在チェック	指定したレジスタ･モードに対して混在チェック	-Xreg_mode	なし
	異なるレジスタ･モードが混在している場合、詳細情報を出力	-Xregmode_info	なし
	生成するデバイス共通オブジェクト・モジュール・ファイルと-C オプションで指定したデバイスの混在チェック	-Xcommon	なし
sdata/sbss 情報	-Xsdata オプションのパラメータに対して目安として用いることのできる情報を標準出力に出力	-Xsdata_info	なし
2 パス・モード	2 パス・モードでリンク	-Xtwo_pass_link	なし
リロケーション解決エラー	リンク時のリロケーション処理において不正箇所があった場合､リンク処理を続行	-Xignore_address_error	なし（注 6）
シンボル多重定義エラー	多重定義されたすべての外部シンボルに対してエラー･メッセージを出力	-Xmultiple_symbol	なし（注 6）
リンク時チェック	リンク時のチェックを抑止	-Xlink_check_off	なし（注 6）
充てん値	アライン・ホールの充てん値を指定	-Xalign_fill	なし（注7）
再スキャン	-l オプションで指定したライブラリ・ファイルの再スキャン	-Xrescan	なし
デバッグ情報セクション	デバッグ情報等を削除したロード･モジュール・ファイルを生成	-Xstrip	-nodebug
ROM 化処理前ロード・モジュール・ファイル保存	ROM 化処理前のロード・モジュール・ファイルを保存	-Xlink_output	なし（注8）
エラー出力制御	エラー･メッセージをファイルに出力	-Xerror_file	なし
警告メッセージ出力制御	指定した警告メッセージを出力	-Xwarning	なし
	指定した警告メッセージの出力を抑止	-Xno_warinig	なし
コマンド･ファイル指定	コマンド・ファイルを指定します。	@	-subcommand

注 6 ：リンカの-change_message オプションを指定して、エラーを警告に変更して処理を継続可能です。
注 7 ：CC-RH のリンカの-PADDING、-Space 等により一部代用可能です。
注 8 ：CC-RH はデフォルトで ROM 化しません。リンク時に-ROm オプションの指定が必要です。

2. 組み込み関数

本章では、CXの組み込み関数に相当するCC-RHの組み込み関数を比較表にて示します。
なお、CXにあってCC-RHにない組み込み関数を呼び出した場合、CC-RHでは通常関数としてコンパイルします。定義が無い場合はリンク時にエラーとなります。

命令	説明	CX	CC-RH
di	割り込み禁止	void __DI(void);	void __DI(void);
ei	割り込み許可	void __EI(void);	void __EI(void);
nop	ノー・オペレーション	void __nop(void);	void __nop(void);
halt	プロセッサの停止	void __halt(void);	void __halt(void);
satadd	飽和加算	long a, b; long __satadd(a, b);	long a, b; long __satadd(a, b);
satsub	飽和減算	long a, b; long __satsub(a, b);	long a, b; long __satsub(a, b);
bsh	ハーフワード・データのバイト・スワップ	long a; long __bsh(a);	long a; long __bsh(a);
bsw	ワード・データのバイト・スワップ	long a; long __bsw(a);	long a; long __bsw(a);
hsw	ワード・データのハーフワード・スワップ	long a; long __hsw(a);	long a; long __hsw(a);
sxb	バイト・データの符号拡張	char a; long __sxb(a);	なし
sxh	ハーフワード・データの符号拡張	short a; long __sxh(a);	なし
mul	符号つき乗算結果の64 ビットを変数に代入する命令	long a, b; long long __mul(a, b);	なし
mulu	符号なし乗算結果の64 ビットを変数に代入する命令	unsigned long a, b; unsigned long long __mulu(a,b);	なし
mul	符号つき乗算結果の上位32ビットを変数に代入する命令	long a, b; long __mul32(a, b);	long a, b; long __mul32(a, b);
mulu	符号なし乗算結果の上位32ビットを変数に代入する命令	unsigned long a, b; unsigned long __mul32u(a, b);	unsigned long a, b; unsigned long __mul32u(a, b);
sasf	論理左シフト付きフラグ条件の設定	long a; unsigned int b; long __sasf(a, b);	なし
sch0l	MSB 側からのビット（0）検索	long a; long __sch0l(a);	long a; long __sch0l(a);
sch0r	LSB 側からのビット（0）検索	long a; long __sch0r(a);	long a; long __sch0r(a);
sch1l	MSB 側からのビット（1）検索	long a; long __sch1l(a);	long a; long __sch1l(a);
sch1r	LSB 側からのビット（1）検索	long a; long __sch1r(a);	long a; long __sch1r(a);
ldsr	システム・レジスタへのロード	long regID; long a; void __ldsr(regID, a);	long regID; unsigned long a; void __ldsr(regID, a);
stsr	システム・レジスタの内容のストア	long regID; unsigned long __stsr(regID);	long regID; unsigned long __stsr(regID);
ldgr	汎用レジスタへのロード	long a; void __ldgr(regID, a);	なし
stgr	汎用レジスタの内容のストア	unsigned long __stgr(regID);	なし
caxi	比較と交換	long *a; long b, c; void __caxi(a, b, c);	long *a; long b, c; long __caxi(a, b, c);
-	割り込みレベルの制御（注1）	int NUM; void __set_il(NUM, “割り込み要求名”);	int NUM; void* ADDR; void __set_il_rh(NUM, ADDR);

注 1：CX はデバイス・ファイルで定義された文字列を“割り込み要求名”に指定します。CC-RH は ADDR に割り込み制御レジスタのアドレスを指定してください。

3. 既定義マクロ

本章では、CXの既定義マクロに相当するCC-RHの既定義マクロを比較表にて示します。

CX のマクロ名	CX の定義	CC-RH のマクロ名
__LINE__	その時点でのソース行の行番号（10 進数）	__LINE__
__FILE__	仮定されたソース・ファイルの名前（文字列定数）	__FILE__
__DATE__	ソース・ファイルの翻訳日付（“Mmm dd yyyy”の形式をもつ文字列定数）	__DATE__
__TIME__	ソース・ファイルの翻訳時間	__TIME__
__STDC__	10 進定数1（-Xansi オプション指定時に定義）	__STDC__
__v850 __v850__ __v850e2 __v850e2__ __v850e2v3 __v850e2v3__	10 進定数1	__RH850 __RH850__ __v850e3v5 __v850e3v5__
__CX __CX__	10 進定数1	__CCRH __CCRH__
__CHAR_SIGNED__	10 進定数1（-Xchar オプションで符号つきを指定した場合、および-Xchar オプションを指定しない場合に定義）	値は設定しない
__CHAR_UNSIGNED__	10 進定数1（-Xchar オプションで符号なしを指定した場合に定義）	なし
__DOUBLE_IS_64BITS__	10 進定数1	値は設定しない
__品種指定名__	10 進定数1 デバイス・ファイル中の「品種指定名」で示される文字列の先頭と末尾に“__”を付けたもの	なし
__reg32__	10 進定数1（-Xreg_mode=32 オプションを指定した場合、および-Xreg_modeオプションを指定しない場合に定義）	値は設定しない （-Xreg_mode=32オプションを指定した場合に定義）
__reg26__	10 進定数1（-Xreg_mode=26 オプションを指定した場合に定義）	なし
__reg22__	10 進定数1（-Xreg_mode=22 オプションを指定した場合に定義）	値は設定しない （-Xreg_mode=22オプションを指定した場合に定義）
__reg_common__	10 進定数1 （ -Xreg_mode=common オプションを指定した場合に定義）	値は設定しない （-Xreg_mode=commonオプションを指定した場合に定義）
_MULTI_CORE__	10 進定数1（-Xmulti オプションを指定した場合に定義）	なし
__MULTI_CMN__	10 進定数1（-Xmulti=cmn オプションを指定した場合に定義）	なし
__MULTI_PEn__	10 進定数1（-Xmulti=pen オプションを指定した場合に定義）	なし

4. 拡張言語仕様

本章では、CX の拡張言語に相当する CC-RH の拡張言語を比較表にて示します。

説明	CX	CC-RH
アセンブラ命令の記述（注1）	#pragma asm アセンブラ命令 #pragma endasm	#pragma inline_asm 関数名[, 関数名]
	__asm(“アセンブラ命令”);	なし
インライン展開指定	#pragma inline 関数名[, 関数名]	#pragma inline 関数名[, 関数名]
データのセクション割り当て（注2）	#pragma section セクション種別 ["セクション名"] 変数宣言／定義 #pragma section default	#pragma section 属性指定文字 ["セクション名"] 変数宣言／定義 #pragma section default
関数のセクション割り当て（注3）	#pragma text “セクション名” 関数名 [,関数名]…	#pragma section text ["セクション名"] 変数定義 #pragma section default
周辺I/O レジスタ名有効化指定（注4）	#pragma ioreg	なし
割り込み／例外ハンドラ指定（注5）	#pragma interrupt 割り込み要求名 関数名 [配置方法] [オプション]	#pragma interrupt 関数名 [割り込み仕様]
割り込み禁止関数指定	#pragma block_interrupt 関数名	#pragma block_interrupt 関数名
タスク指定	#pragma rtos_task [関数名]	なし
構造体パッキング指定	#pragma pack ([1|2|4|8])	#pragma pack ([1|2|4])

注 1：CX は C 言語で記述された関数内にアセンブラ命令を埋め込む際に使用する拡張記述ですが、CC-RH は関数そのものをアセンブラ命令のみとみなして、#pragma inline_asm で宣言したアセンブリ記述関数を呼び出し箇所にインライン展開します。

注 2：CX と CC-RH のセクション名は異なります。そのため、それぞれセクション種別／属性指定文字で指定可能な文字列が異なります。詳細につきましてはコーディング編マニュアルをご参照ください。

注 3：CC-RH で数字から始まるセクション名を指定した場合、数字の前に”_”を自動的に付加します。

注 4：CC-RH は周辺 I/O レジスタ用のヘッダ・ファイルをインクルードして下さい。

注 5：CX は記述した割り込み関数への分岐命令を割り込みハンドラアドレスに自動で配置しますが、CC-RH は割り込み／例外ベクタをお客様自身で定義・配置させる必要があります。またオプション／割り込み仕様に指定可能な文字列もそれぞれ異なります。

5. アセンブラ疑似命令

本章では、CXのアセンブラ疑似命令に相当するCC-RHのアセンブラ疑似命令を比較表にて示します。疑似命令とは、アセンブラが一連の処理を行う際に必要な各種の指示を行うものです。

説明	CX	CC-RH
セクション定義疑似命令	.cseg	.cseg (注1)
	.dseg	.dseg (注1)
	.org	.offset (注2)
	.vseg	なし
シンボル定義疑似命令	.set	.set
データ定義､領域確保疑似命令	.db	.db
	.db2/.dhw	.db2/.dhw
	.dshw	.dshw
	.db4/.dw	.db4/.dw
	.db8/.ddw	.db8/.ddw
	.float	.float
	.double	.double
	.ds	.ds
	.align	.align
外部定義､外部参照疑似命令	.public	.public
	.extern	.extern
	.comm	なし
マクロ疑似命令	.macro	.macro
	.local	.local
	.rept	.rept
	.irp	.irp
	.exitm	.exitm
	.exitma	.exitma
	.endm	.endm

注 1：オペランドで指定する再配置属性は CX とは異なります。
注 2：CC-RH の.org は絶対アドレス形式セクションの開始を指示する疑似命令です。

6. 周辺 I/O レジスタ

本章では、CXとCC-RHの周辺I/Oレジスタの扱いについて説明します。

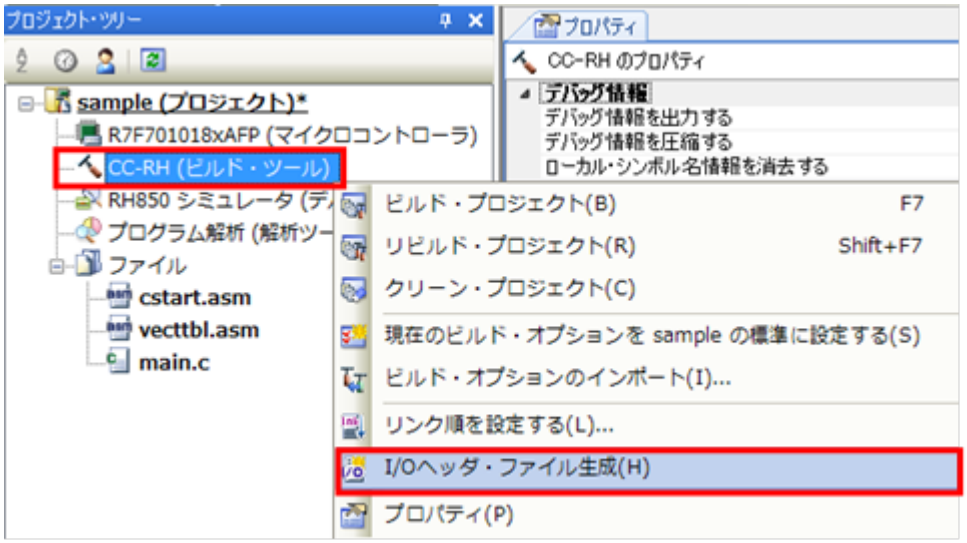
6.1 CX の周辺 I/O レジスタ

CXでは、#pragma 指令を追加することによってC言語で周辺I/Oレジスタ名を用いてアクセスすることが可能です。レジスタ名と対応アドレス一覧はデバイス・ファイルに登録されており、レジスタ名はアセンブル時にアドレスに変換されます。デバイス・ファイルに登録されているレジスタ名は各マイコンのユーザーズマニュアルに記載されています。

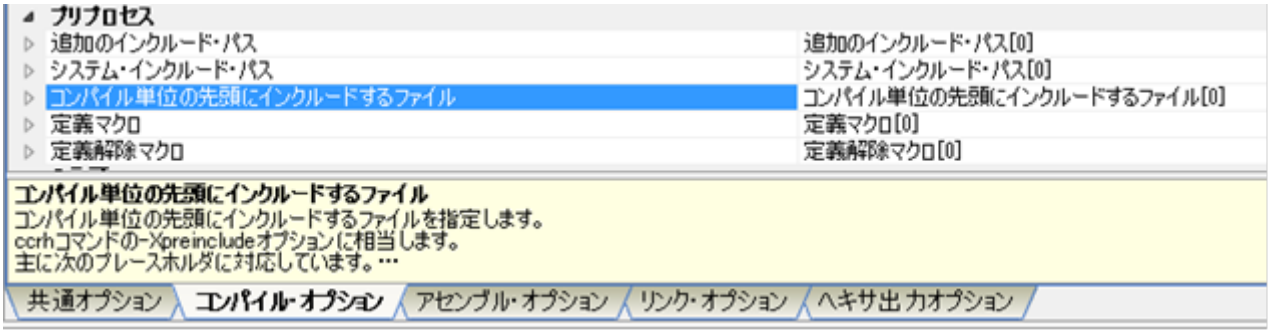
6.2 CC-RH の周辺 I/O レジスタ

CC-RHはデバイス・ファイルをサポートしていないため、お客様自身で周辺I/Oレジスタ名と対応アドレス一覧を記載したファイルを準備して頂く必要があります。

CS+では新規プロジェクト作成時に、プロジェクトで指定している該当マイコン用の I/O ヘッダ・ファイル”iodefine.h”を生成し、プロジェクトにソース登録します。I/O ヘッダ・ファイルでは、マイコンが持つレジスタ名と、そのアドレスが定義されています。また、CS+プロジェクト・ツリーの[CC-RH(ビルド・ツール)]ノードを右クリック => [I/O ヘッダ・ファイル生成]を押下して生成させることも可能です。



C 言語プログラム中でレジスタをアクセスする場合、I/O ヘッダ・ファイルをインクルードしてください。なお、-Xpreinclude オプションの引数に本ファイルを指定することにより、ソース・ファイル中に#include 指定する必要がなくなります。-Xpreinclude オプションは、[コンパイル・オプション]タブ => [プリプロセス]カテゴリ => [コンパイル単位の先頭にインクルードするファイル]にて指定可能です。本プロパティで該当マイコン用の I/O ヘッダ・ファイルを指定してください。



7.　割り込み／例外

本章では、CXとCC-RHの割り込み／例外ハンドラについて説明します。

7.1　CX の周辺 I/O レジスタ

CXは#pragma interrupt 指令により、指定した「割り込み要求名」に対応するハンドラアドレスに「関数名」で指定した関数への分岐命令を埋め込みます。また「関数名」で指定した関数を割り込み関数としてコンパイルします。

```
#pragma interrupt 割り込み要求名　関数名［配置方法］［オプション］
```

「割り込み要求名」にはデバイス・ファイルに登録されている割り込み要求名を指定します。デバイス・ファイルに登録されている割り込み要求名は、各マイコンのユーザーズマニュアルに記載されています。

例えばV850E2/FJ4の割り込み要求名”INTP0” のハンドラアドレスは0x110番地です。この場合、以下の#pragma interrupt 指令により、0x110番地に”jr _func”命令を埋め込みます。またfunc関数を割り込み関数としてコンパイルし、割り込み／例外ハンドラとしてのレジスタの退避／復帰処理を出力します。

```
#pragma interrupt INTP0 func
void func(void) {
  …;
}
```

7.2　CC-RH の割り込み／例外

CC-RHは#pragma interrupt 指令により、「関数名」で指定した関数を「割り込み仕様」に指定した仕様に従ってコンパイルします。

```
#pragma interrupt 関数名［割り込み仕様］
```

例えば以下の#pragma interrupt 指令により、func関数を割り込み関数としてコンパイルします。また、指定した割り込み仕様に従って、ctpc／ctpsw／fpepc／fpsr の退避／復帰処理とei／di 命令を出力します。

```
#pragma interrupt func (enable=true, callt=true, fpu=true)
void func (unsigned long eiic)
{
  …;
}
```

なお、CC-RH は割り込み／例外ベクタをお客様自身で定義・配置させる必要があります。CS+で新規プロジェクトを作成時にソース登録される”boot.asm”では、割り込み／例外ベクタのフォーマットを定義しています。必要に応じてカスタマイズしてください。また、対象マイコンに応じて適切なアドレスに配置させてください。

以下、”boot.asm”の割り込み／例外ベクタについて説明します。

a. RESET

下記の定義により、RESET セクションの先頭に「jr32 __start」命令が埋め込まれます。

```
        .section    "RESET", text
        .align      512
        jr32        __start ; RESET
```

例えば CS+で RH850/F1L 用プロジェクトを新規作成した場合、リンカオプション”-start”により RESET セクションは%ResetVectorPE1%番地に配置指定されています。%ResetVectorPE1%は、プロジェクト・ツリーの[マイクロコントローラ]ノード=>[マイクロコントローラ情報タブ] =>[マイクロコントローラ情報]カテゴリ=>[リセット・ベクタ・アドレス]で指定した値です。デフォルトでは 0x00 番地に「jr32 __start」命令が埋め込まれます。

b. 直接ベクタ方式の例外／割り込み

ハンドラアドレスの基準位置は、RBASE レジスタ、または EBASE レジスタで示されるベース・アドレスに例外要因のオフセットを加算した値を使用します。いずれをベース・アドレスとして利用するかは、PSW.EBV ビットによって選択します。下記の定義では、RBASE をベース・アドレスであるものとして、RESET の直後から割り込み／例外ハンドラを配置しています。

```
        .section    "RESET", text
        .align      512
        jr32        __start ; RESET

        .align      16
        jr32        _Dummy ; SYSERR

        .align      16
        jr32        _Dummy ;

        .align      16
        jr32        _Dummy ; FETRAP

        ...
```

”boot.asm”では、SYSERR／FETRAP 等の対応するオフセット位置に、ダミー関数_Dummy に分岐する命令を配置しています。_Dummy は自分自身への分岐を繰り返すルーチンです。必要に応じてカスタマイズしてください。

カスタマイズ対象とする例外／割り込みに対応するオフセット位置の「_Dummy」を「_割り込み関数名」に変更してください。また、#pragma interrupt 指令によって割り込み関数を定義してください。以下は例外”SYSERR”発生時に割り込み関数”func”を実行する場合の記述例です。

```
        .section    "RESET", text
        .align      512
        jr32        __start ; RESET

                                  「__Dummy」を「_割り込み関数名」に変更
        .align16
        jr32        _func ; SYSERR
        ...
```

```
#pragma interrupt func (priority=SYSERR, callt=true, fpu=true)
void func(unsigned long feic)
{
  …;
}
```

c. テーブル参照方式の例外／割り込み

割り込みの拡張仕様として、テーブル参照方式による割り込みを指定できます。直接ベクタ方式では、EI レベル割り込みのハンドラアドレスはそれぞれの割り込み優先度ごとに1つであり、複数の同一優先度を示す割り込みチャネルは同じ割り込みハンドラアドレスへ分岐します。しかし、アプリケーション上、割り込みハンドラごとに異なるコード領域を利用したい場合などを想定し、テーブル参照方式を定義しています。

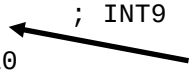
```
      .section    " EIINTTBL ", const
      .align      512
      .dw         #_Dummy_EI ; INT0
      .dw         #_Dummy_EI ; INT1
      .dw         #_Dummy_EI ; INT2
      .rept       512 - 3
      .dw         #_Dummy_EI ; INTn
      .endm
```

”boot.asm”では、EIINTTBL セクションにテーブル参照方式の例外／割り込みテーブルを定義しています。CS+で RH850/F1L 用プロジェクトを新規作成した場合、リンカオプション”-start”により RESET セクションの直後に配置指定されています。

EIINTTBL セクションの先頭から 4 の倍数の領域にダミー関数_Dummy_EI の配置アドレスが埋め込まれています。これにより、割り込み優先度 n(n は 0 から 512)のテーブル参照方式の例外／割り込みが発生した場合、_Dummy_EI に分岐します。_Dummy_EI は自分自身への分岐を繰り返すルーチンです。必要に応じてカスタマイズしてください。

カスタマイズ対象とするチャネルに対応するオフセット位置の「#_Dummy_EI」を「#_割り込み関数名」に変更してください。また、C ソースファイル上で割り込み関数を定義する場合は、#pragma interrupt 指令によって割り込み関数を定義してください。以下はチャネル 9”EIINT9”発生時に割り込み関数 func を実行する場合の記述例です。

```
        .section "EIINTTBL", const
        .align   512
        .dw      #_Dummy_EI ; INT0
        .dw      #_Dummy_EI ; INT1
        .dw      #_Dummy_EI ; INT2
        .dw      #_Dummy_EI ; INT3
        .dw      #_Dummy_EI ; INT4
        .dw      #_Dummy_EI ; INT5
        .dw      #_Dummy_EI ; INT6
        .dw      #_Dummy_EI ; INT7
        .dw      #_Dummy_EI ; INT8
        .dw      #_func     ; INT9
        .rept    512 - 10
        .dw      #_Dummy_EI ; INTn
```



「#_Dummy_EI」を
「#_割り込み関数名」に変更

```
#pragma interrupt func (channel=9 enable=true, callt=true, fpu=true)

void func (unsigned long eiic)
{
  …;
}
```

なお、直接ベクタ方式の例外／割り込みが RH850 のデフォルトであるため、テーブル参照方式に変更する場合は割り込み制御レジスタの値を変更する必要があります。

8. ROM 化

初期値付き変数のデータはROMに配置しておき、リセット後の実行時にRAMにコピーする必要があります。この一連の処理をROM化といいます。ROM化の方法はCXとCC-RHで異なります。本章では、CXとCC-RHのROM化方法について説明します。

8.1 CX の ROM 化方法

CXの場合、初期値あり変数が配置されるセクション（.sdata／.data セクション）はデフォルトでROM化対象としています。初期値データはROMに格納されていますので、_rcopy関数を使用してROMからRAMにコピーします。コピー先のアドレス（sdata／.data セクションのアドレス）はリンク・ディレクティブ・ファイル（*.dir）で指定します。以下はデフォルトのスタートアップ・ルーチン” cstart.asm”の_rcopy関数の呼び出し例です。

```
mov32   #__S_romp, r6          ;  copy romized data
mov      -1, r7
jarl    __rcopy, lp
```

「__S_romp」はROM化用領域確保コード・ファイル”rompcrt.obj”で定義しているシンボルであり、「#__S_romp」はROMに格納されている初期値データの先頭アドレスです。リンカが自動的に決定します。

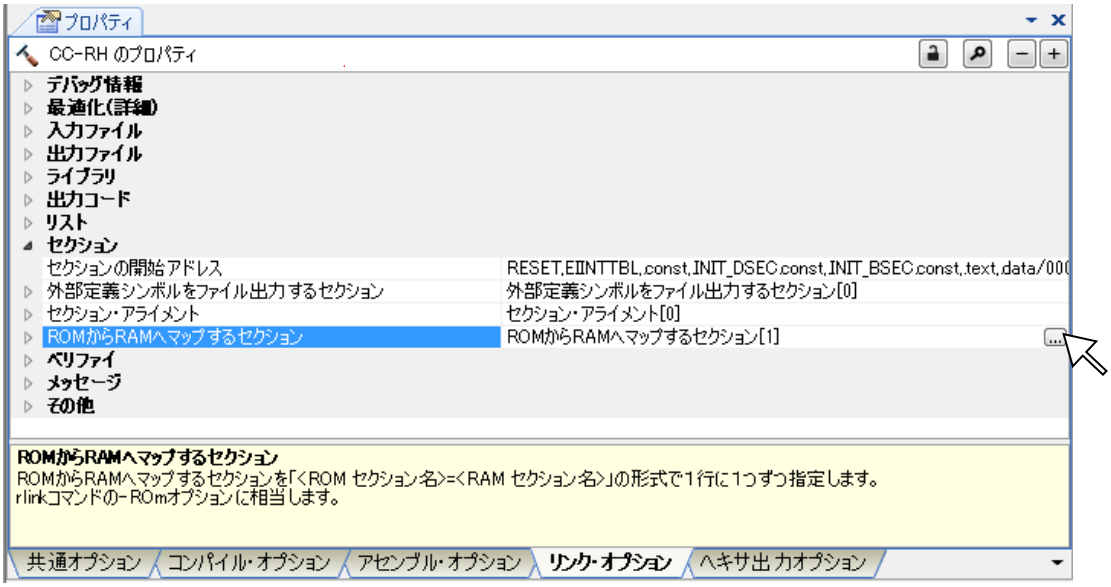
8.2 CC-RH の ROM 化方法

a. ROM化指定

CC-RHの場合、ROM化対象とするセクションをリンカの-romオプションにより指定します。<*ROMセクション名*>がROM化対象のセクションとなります。リンカの-startオプションによって、<*ROMセクション名*>で指定したセクションはROMに配置指定し、<*RAMセクション名*>で指定したセクションはRAMに配置指定してください。

```
-rom=<ROM セクション名>=<RAM セクション名>
```

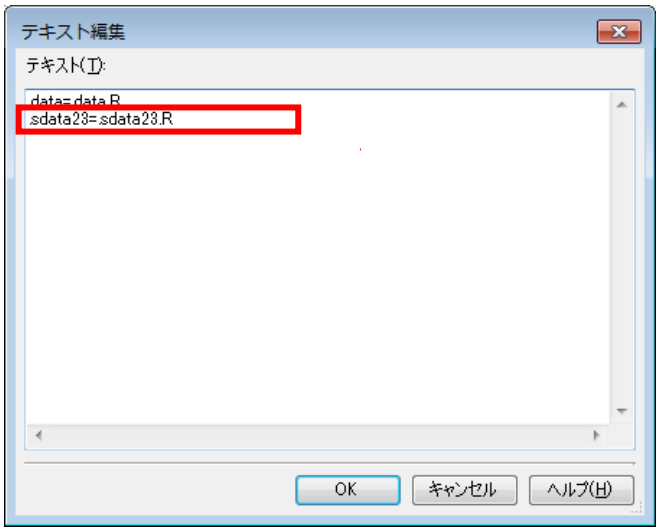
CS+では[リンク・オプション]タブ => [セクション]カテゴリ => [ROMからRAMへマップするセクション] にて右端の[...]ボタンを押下し、ROMからRAMへコピーするセクションを「<*ROMセクション名*>=<*RAMセクション名*>」の形式で、1行に1つずつ指定してください。



CS+で新規プロジェクトを作成した場合、以下のオプションがデフォルトで指定されています。

```
-rom=.data=.data.R
```

.dataセクション以外に初期値あり変数が配置されるセクションを追加した場合、-romオプションによりROM化対象として追加指定してください。以下は.sdata23セクションを追加でROM化対象とする場合（RAMセクション名を.sdata.Rとする場合）の指定例です。



b. 初期化テーブルの定義

CC-RHでは_INITSCT_RH関数を使用してROM化したデータをROMからRAMにコピーします。CS+で新規プロジェクトを作成時にソース登録されるスタートアップ・ルーチン”cstart.asm”では、初期値あり変数のデータをコピーする際に使用する初期化テーブルを以下のように定義しています。

```
;-----------------------------------------------------------------------
;    section initialize table
;-----------------------------------------------------------------------
     .section    ".INIT_DSEC.const", const
     .align      4
     .dw         #__s.data,  #__e.data,  #__s.data.R
```

初期化テーブルは.INIT_DSEC.constセクションに配置指定されており、.dataセクションの先頭アドレス->.dataセクションの終端アドレス->.data.Rセクションの先頭アドレスの順番でそれぞれ4バイトずつ領域が確保されています。

なお、セクション名の頭に”__s”を付けることで、そのセクションの先頭アドレスを値として持つ予約シンボルとなります。同様にセクション名の頭に”__e”を付けることで、そのセクションの終端アドレスを値として持つ予約シンボルとなります。初期化テーブルへの追加はこの予約シンボルを使用頂くことを推奨します。

初期値あり変数が配置されるセクションを追加した場合は、この初期化テーブルにもそれぞれセクションの先頭アドレス・終端アドレスを追加してください。

```
;-------------------------------------------------------------------------
;    section initialize table
;-------------------------------------------------------------------------
     .section    ".INIT_DSEC.const", const
     .align      4
     .dw         #__s.data,    #__e.data,   #__s.data.R
     .dw         #__s.sdata23,       #__e.sdata23,      #__s.sdata23.R
```

CC-RHでは、_INITSCT_RH関数を使用して、初期値なし変数が配置されるセクションをゼロ初期化することも可能です。スタートアップ・ルーチン”cstart.asm”でゼロ初期化テーブルを定義しています。

```
     .section    ".INIT_BSEC.const", const
     .align      4
     .dw         #__s.bss,    #__e.bss
```

ゼロ初期化テーブルは.INIT_BSEC.const セクションに配置指定されており、.bssセクションの先頭アドレス->.bssセクションの終端アドレスの順番でそれぞれ4バイトずつ領域が確保されています。.bssセクション以外に初期値なし変数が配置されるセクションを追加した場合は、同様の手順でそれぞれのアドレスを追加してください。

c.　コピー関数の呼び出し

スタートアップ・ルーチン”cstart.asm”で_INITSCT_RH 関数を呼び出しています。この処理により、各テーブルで定義したセクションを初期化します。

```
     mov     #__s.INIT_DSEC.const, r6
     mov     #__e.INIT_DSEC.const, r7
     mov     #__s.INIT_BSEC.const, r8
     mov     #__e.INIT_BSEC.const, r9
     jarl32  __INITSCT_RH, lp       ;  initialize RAM area
```

9. セクション配置

本章では、CXとCC-RHのセクション配置方法を説明します。

9.1 CX のセクション配置

CXはリンク・ディレクティブ・ファイル（*.dir）というファイルにセクションの配置アドレスを記述し、-Xlink_directiveオプションによりリンク・ディレクティブ・ファイルをリンカに入力することでセクションの配置アドレスを指定します。以下はCXのリンク・ディレクティブ・ファイルのフォーマットです。

```
セグメント名:!セグメントタイプ　?セグメント属性　V アドレス {

        出力セクション名=$セクションタイプ　?セクション属性　入力セクション名;
        出力セクション名=$セクションタイプ　?セクション属性　入力セクション名;
        …
};
```

下記の記述により、.const セクションを0x1000番地から配置し、.const セクションの終端から上位方向に.pro_epi_runtime =>.text セクションを配置します。また、0xfedf6000番地から上位方向に.data => .sdata => .sbss => .bss セクションを配置します。

```
CONST:!LOAD ?R V0x1000 {
        .const = $PROGBITS ?A .const ;
};

TEXT:!LOAD ?RX {
        .pro_epi_runtime = $PROGBITS ?AX .pro_epi_runtime ;
        .text = $PROGBITS ?AX .text ;
};

DATA:!LOAD ?RW V0xfedf6000 {
        .data = $PROGBITS ?AW .data ;
        .sdata = $PROGBITS ?AWG .sdata ;
        .sbss = $NOBITS ?AWG .sbss ;
        .bss = $NOBITS ?AW .bss ;
};
```

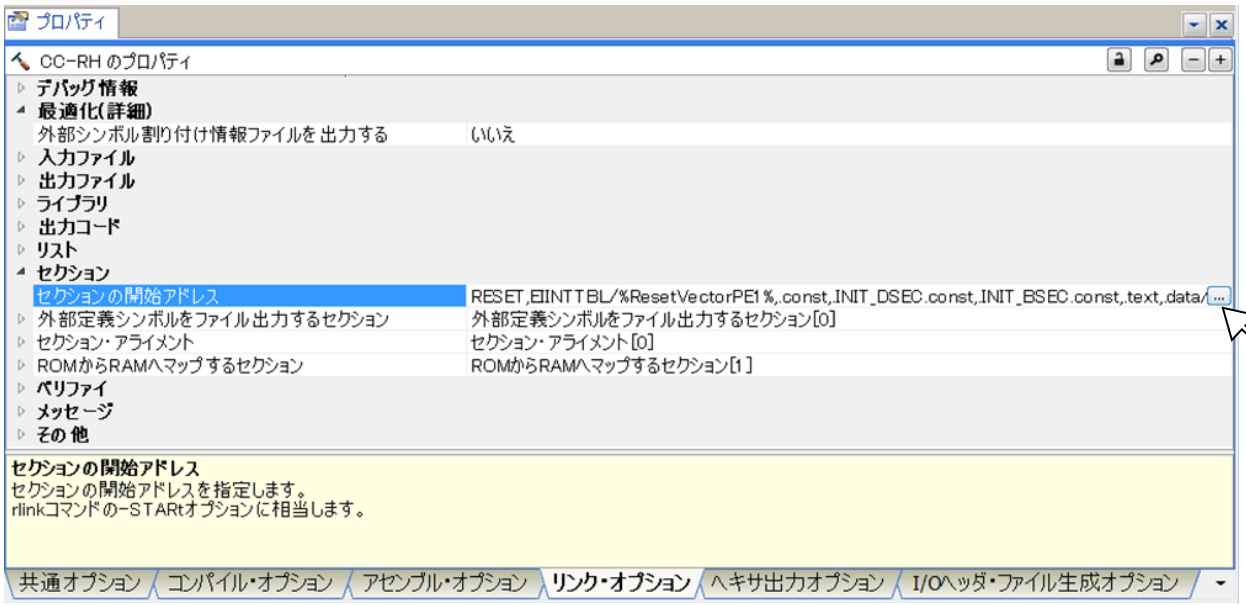
9.2　CC-RH のセクション配置

CC-RHはリンカオプション”-start”によりセクションの配置アドレスを指定します。CXのリンク・ディレクティブ・ファイルのようなファイルで指定する形式ではありません。以下はCS+でRH850/F1L用プロジェクトを新規作成した場合の-startオプションの指定例です。詳細につきましてはビルド編マニュアルをご参照ください。

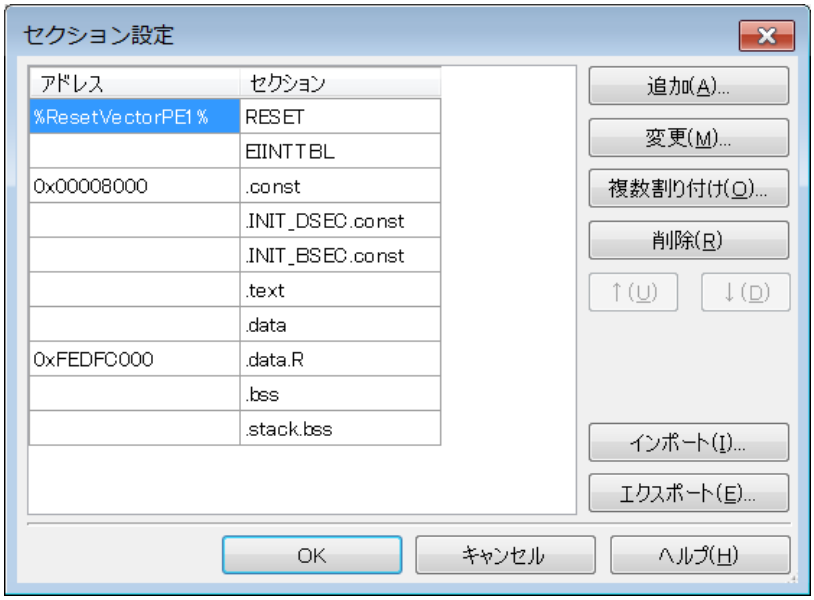
```
-start=RESET,EIINTTBL/%ResetVectorPE1%,.const,.INIT_DSEC.const,.INIT_BSEC.const,.text,
.data/00008000,.data.R,.bss,.stack.bss/FEDEFC00
```

上記オプションにより、RESETセクションを%ResetVectorPE1%番地から配置し、RESETセクションの終端から上位方向にEIINTTBL => .const => .INIT_DSEC.const => .INIT_BSEC.const => .text => .data セクションを配置します。また、0xFEDE0000番地から上位方向にdata.R => .bss => .stack.bss セクションを配置します。

CS+ではGUI上から指定することが可能です。 [リンク・オプション]タブ => [セクション]カテゴリ => [セクションの開始アドレス] にて右端の[...]ボタンを押下してください。



セクション設定ダイアログが起ち上がりますので、このダイアログ上からの操作により配置アドレスやセクションを追加・変更することが可能です。



10. プログラム互換性

本章では、CXでコンパイル可能なプログラム記述がCC-RHではエラーとなる例と、その回避策を説明します。

1. 定数の2進表記

【記述例】

```
int a = 0b00000001;
```

CXでサポートしていた2進数の拡張表記をCC-RHではサポートしていません。16進表記等に変更してください。

【回避策】

```
int a = 0x01;
```

ホームページとサポート窓口<website and support,ws>

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.10.17	－	初版発行
1.01	2017.04.21	－	移行先コンパイラ CC-RH のバージョンを V1.05.00 に変更

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため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、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害に関し、当社は、何らの責任を負うものではありません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。
注2. 本資料において使用されている「当社製品」とは、注１において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/