RX開発環境移行ガイド

V850からRXへの移行

(コンパイラ編)

(CA850/CX→CC-RX)

2017/04/20
R20UT2608JJ0101

ソフトウェア事業部ソフトウエア技術部
ルネサス システムデザイン株式会社

© 2017 Renesas System Design Co., Ltd. All rights reserved.

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、コンパイラに関する注意事項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

© 2017 Renesas System Design Co., Ltd. All rights reserved.

アジェンダ

© 2017 Renesas System Design Co., Ltd. All rights reserved.

言語仕様

- V850とRXの言語仕様の違い

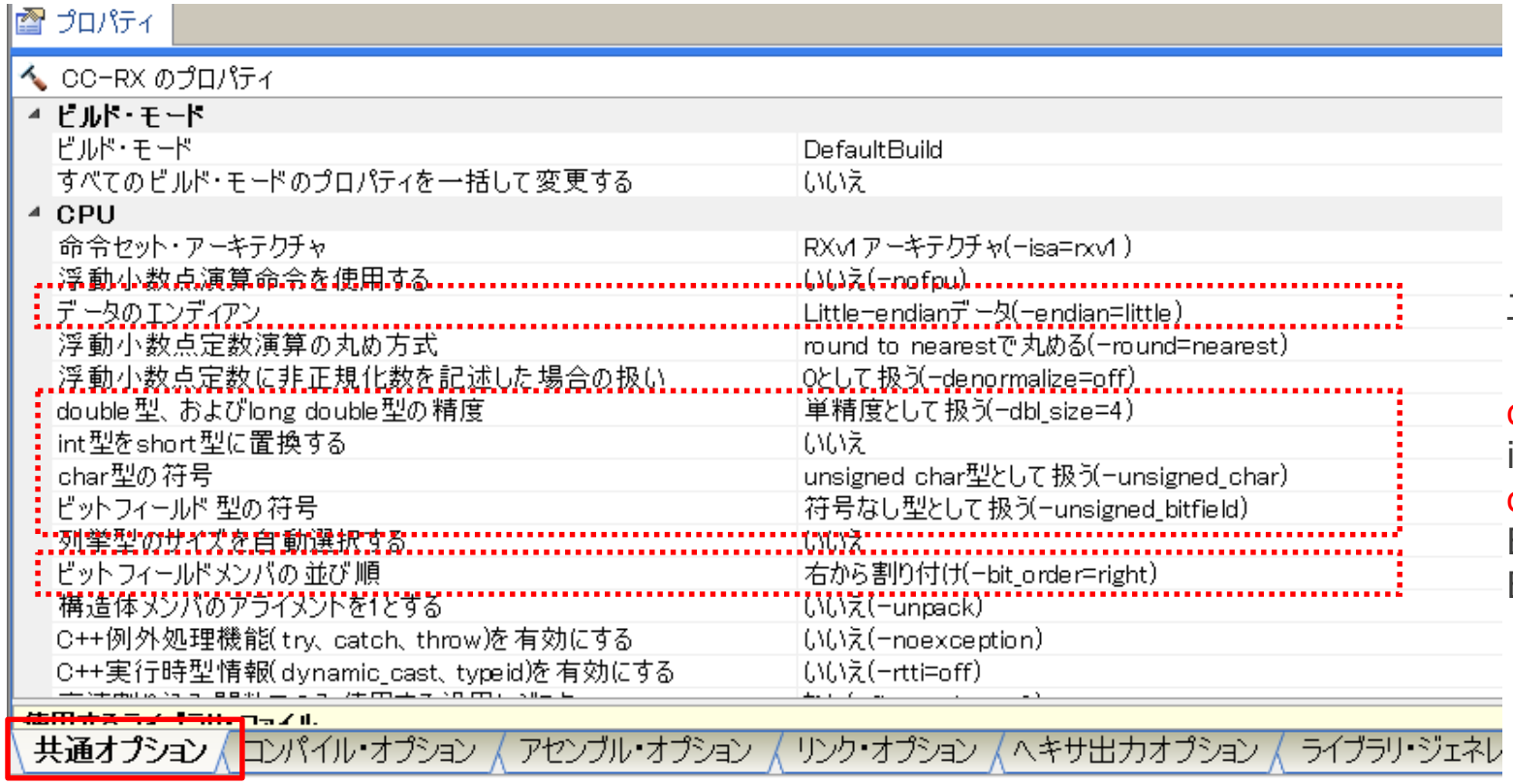
	CA	CX	SH(参考)	RX	
言語	C	C	C/C++	C/C++	
言語規格	C89	C90/C99(一部)	C89	C89/C99	
エンディアン	little	little	big	little	オプションでBigに対応
ビットフィールド	LSB	LSB	MSB	LSB	オプションでMSBに対応
char	符号あり	符号あり	符号あり	符号なし	オプションで符号ありに対応
long long	4byte	8byte	8byte	8byte	
double	4byte	8byte	8byte	4byte	オプションで倍精度に対応
bool	-	サポート	サポート	サポート	
enum	符号付4バイト整数	符号付バイト整数	符号付4バイト整数	符号付4バイト整数	
volatile	アクセス、サイズを保証	アクセス、サイズを保証	アクセス、サイズを保証	アクセスを保証	__evenaccessの宣言を追加

基本的にRXはCA、CXの言語仕様を包含した仕様になっています。
CA、CX環境下で作成されたCソースは、RXで流用が可能*1です。
*1　但し、拡張機能、組み込み関数等はサポート内容が異なりますので修正が必要です。
　　コンパイラユーザーズマニュアルをご参照の上、ご対応ください。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

言語仕様
RXコンパイラでのオプション対応

- V850とRXの言語仕様の違いに対するコンパイルオプション
  CS+のCC-RX（ビルド・ツール）プロパティの共通オプションタブ内で設定



CXと異なるdouble型のサイズCA、CXと異なるchar型の符号有無は、オプション設定にて対応可能

エンディアン

double型のサイズ
int型変数の2byte化
char型の符号指定
Bit field メンバの符号指定
Bit field メンバの割り付け

© 2017 Renesas System Design Co., Ltd. All rights reserved.

言語仕様

enum

- CA、CXとRXはともに符号付き4バイト整数で扱います。
  但し、オプション設定により扱われ方が異なります。
  - CA、CX
    -Xenum_type=*string*を指定した場合、指定した型で処理します。
  - RX
    -auto_enumを指定した場合、列挙値が収まる最小型として処理します。

表 2.10　列挙型のとりうる値と型の関係

列挙子		選択される型
最小値	最大値	
-128	127	signed char
0	255	unsigned char
-32768	32767	signed short
0	65535	unsigned short
上記以外		signed long

© 2017 Renesas System Design Co., Ltd. All rights reserved.

言語仕様

volatile

- CA、CXとRXではvolatile宣言の扱われ方が異なります。

  - CA、CX
    - volatile宣言した変数は、その変数へのアクセスとその変数のサイズでアクセスすることを保証します。

  - RX
    - volatile宣言した変数は、その変数へのアクセスを保証します。
      その変数のサイズでアクセスすることは保証しません。
      宣言した変数のサイズでアクセスするには、
      __evenaccess
      の記述を付加してください。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能

V850とRXの拡張機能(#pragma)、キーワード

	CA	CX	SH(参考)	RX	
セクションの切り替え指定	section text	section text	section	section	
割り込み関数の作成	interrupt	interrupt	interrupt	interrupt	
関数のインライン展開を指定	inline	inline	inline	inline	
アセンブリ記述関数のインライン展開	-	-	inline_asm	inline_asm	
アセンブラ埋め込み機能	asm endasm	asm endasm	-	-	
構造体・共用体/クラスのアライメント数を指定	pack([1|2|4|8])	pack([1|2|4|8])	pack unpack	pack unpack	
割り込み関数の作成	__interrupt	-	-	-	
埋め込みアセンブル機能	__asm	__asm	-	-	
セクションアドレス演算子	__s __e	__s __e	__sectop __secend	__sectop __secend	CA,CXの_S,_eはアセンブラソースのみで使用可能

RXでも基本機能はサポートしていますが、オプションの設定方法は異なります。
詳細は、コンパイラユーザーズマニュアルを参照し変更してください。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能

V850とRXの拡張機能(#pragma)、キーワード

	CA	CX	SH(参考)	RX	
周辺I/Oレジスタ名有効化指定	ioreg	ioreg	-	-	RXは、iodefine.hファイルを使用
割り込み禁止関数指定	block_interrupt	block_interrupt	-	-	RXは、未サポート 割り込み禁止/許可の設定でご対応ください。
スマート・コレクション指定	-	smart_correct	-	-	RXは、未サポート
ポジション・インディペンデント・アクセス 固定アドレス・アクセス	-	pic nopic	-	-	RXは、pic/pidはコンパイラオプションでサポート

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能
セクション指定

- #pragma sectionの使用は、CA、CXとRXでは異なります。
  - CA

    変数のセクション指定：
    #pragma section セクション種別 “作成するセクション名” begin
    ・・・
    #pragma section セクション種別 “作成するセクション名” end
    関数のセクション指定：
    #pragma text “作成するセクション名” 関数名
  - CX

    #pragma sectionセクション種別 “作成するセクション名”
    ・・・
    #pragma section default
  - RX

    #pragma section [<セクション種別>] [△<変更セクション名>]
    ・・・
    #pragma section
    <セクション種別>: { P | C | D | B }

RX用のフォーマットに変更してください。
また、セクション名は、CA、CXとRXでは異なります。詳細は、コンパイラユーザーズマニュアルをご参照ください。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能
RXコンパイラのセクション

- セクションは、CA、CXとRXでは異なります。

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 名称	セクション 属性	形式種別	初期値 書き込み操作	アライメント数	内容
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	C$INIT	romdata	相対	有 不可	4byte	グローバルクラスオブジェクトに対して呼び出されるコンストラクタおよびデストラクタのアドレスを格納
7	C++ 仮想関数表領域	C$VTBL	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	$ADDR_ <section>_ <address> *3	data	絶対	有 / 無 可 / 不可 *4	—	#pragma address 指定した変数を格納
12	可変ベクタ領域	C$VECT	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

RXのセクション一覧
コンパイラユーザーズマニュアルにも記載していますのでご参照ください。

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。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能
割り込み関数

- #pragma interruptの記述方法は、CA、CXとRXでは異なります。

CA

割り込み／例外ハンドラの指定：
#pragma interrupt 割り込み要求名 関数名 配置方法
__interrupt 関数定義，または関数宣言
多重割り込み／例外ハンドラを指定：
#pragma interrupt 割り込み要求名 関数名 配置方法
__multi_interrupt 関数定義，または関数宣言

CX

#pragma interrupt 割り込み要求名 関数名 配置方法 オプション
__interruptは不要。多重割り込みは[オプション]で指定。

RX

#pragma interrupt [(]<関数名>[(<割り込み仕様> [,…])][,…][)]
割り込み仕様：
ベクタテーブル指定　vect=ベクタ番号、高速割り込み指定　fint
割り込み関数レジスタ制限指定　save、多重割り込み許可指定 enable
ACC 保存指定　acc、 ACC非保存指定　no_acc

RXのスタートアップファイル

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

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

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

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

尚、プロジェクトを作成した際に
生成するintprg.c、vect.hファイルを
使用すればCPUの割り込み関数はすべて
宣言してあります。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能
アセンブリ埋め込み機能

- RXでは、C/C++ソースの関数内に直接アセンブリソースを記述する
  拡張機能はサポートしていません。アセンブリソース部分を関数化しご対応ください。

  なお、CA、CXとRXは命令セットは異なりますのでアセンブリ記述自体を
  変更して頂く必要があります。

CA、CX

```
・・・
__asm("nop");
・・・
```

RX

```
・・・
func();
・・・

#pragma inline_asm func
void func(void)
{
        nop
}
```

© 2017 Renesas System Design Co., Ltd. All rights reserved.

拡張機能
周辺I/Oレジスタ名有効化指定

- RXでは、#pragma ioregはサポートしていません。
  周辺レジスタへアクセスするには、提供しているiodefine.hファイルをご使用下さい。

```
<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

© 2017 Renesas System Design Co., Ltd. All rights reserved.

組み込み関数

- V850とRXの組み込み関数の違いは以下となっています。

	CA	CX	SH(参考)	RX	
割り込み制御	__DI() __EI()	__DI() __EI()	-	setpsw_i() clrpsw_i()	
ノー・オペレーション	__nop()	__nop()	nop()	nop()	
プロセッサの停止	__halt()	__halt()	-	-	
飽和加算	__satadd()	__satadd()	addv() ovf_addv()	-	
飽和減算	__satsub()	__satsub()	subv() unf_subv()	-	
ハーフワード・データのバイト・スワップ	__bsh()	__bsh()	swapb()	-	
ワード・データのバイト・スワップ	__bsw()	__bsw()	end_cnvl()	-	
ワード・データのハーフワード・スワップ	__hsw()	__hsw()	swapw()	-	
バイト・データの符号拡張	__sxb()	__sxb()	-	-	
ハーフワード・データの符号拡張	__sxh()	__sxh()	-	-	
mul命令を用いて32ビット×32ビットの符号つき乗算結果の64ビットを変数に代入する命令	-	__mul()	dmuls_l()	-	
mulu命令を用いて32ビット×32ビットの符号なし乗算結果の64ビットを変数に代入する命令	-	__mulu()	dmuls_h()	-	

© 2017 Renesas System Design Co., Ltd. All rights reserved.

組み込み関数

	CA	CX	SH(参考)	RX	
mul32命令を用いて乗算結果の上位32ビットを変数に代入する命令	__mul32()	__mul32()	dmulu_l()	-	
mulu32命令を用いて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	__mul32u()	__mul32u()	dmulu_h()	-	
論理左シフト付きフラグ条件の設定	__sasf()	__sasf()	-	-	
MSB側からのビット（0）検索	-	__sch0l()	-	-	
LSB側からのビット（0）検索	-	__sch0r()	-	-	
MSB側からのビット（1）検索	-	__sch1l()	-	-	
LSB側からのビット（1）検索	-	__sch1r()	-	-	
システム・レジスタへのロード	-	__ldsr()	set_cr()	set_psw()	
システム・レジスタの内容のストア	-	__stsr()	get_cr()	get_psw()	
汎用レジスタへのロード	-	__ldgr()	-	-	
汎用レジスタの内容のストア	-	__stgr()	-	-	
比較と交換	-	__caxi()	-	-	

RXは、CA、CXとサポートしている組み込み関数が異なります。
ソースファイル内で使用している組み込み関数は、RX用に変更、またはアセンブラ記述での対応をしてください。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

マクロ

- V850とRXのマクロの違いは以下となっています。

	CA	CX	SH(参考)	RX	
	__CA850　__CA850__	__CX　__CX__	_SH	__RX	
	CPUマクロ　*1	CPUマクロ　*1	-	-	
	__DATE__　__TIME__	__DATE__　__TIME__	__DATE__　__TIME__	__DATE__　__TIME__	
	__FILE__　__LINE__	__FILE__　__LINE__	-	__FILE__　__LINE__	
	__DOUBLE_IS_32BITS__ __DOUBLE_IS_32BITS	-	-	__DBL4	
	-	__DOUBLE_IS_64BITS__	-	__DBL8	
	__CHAR_SIGNED__	__CHAR_SIGNED__	-	__SCHAR	
	__CHAR_UNSIGNED__	__CHAR_UNSIGNED__	-	__UCHAR	
	レジスタ・モード・マクロ*2	-	-		

*1 CPUマクロは、ターゲットCPUを示すマクロで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“__”を付けたものが定義されます。
*2 レジスタ・モード・マクロは、ターゲットCPUを示すマクロで10進定数1。レジスタ・モードと定義されるマクロは次のとおりです。
32レジスタ・モード ：__reg32__、 26レジスタ・モード ：__reg26__、 22レジスタ・モード ：__reg22__

RXは、CA、CXとサポートしているマクロ名が異なります。
ソースファイル内でマクロを使用している場合は、RX用に変更してください。

© 2017 Renesas System Design Co., Ltd. All rights reserved.

ルネサス システムデザイン株式会社

© 2017 Renesas System Design Co., Ltd. All rights reserved.