RX ファミリ リアルタイム OS RI600PX V1.02.00 リリースノート

R20UT3090JJ0102
Rev.1.02
2017.06.09

この度は、RX リアルタイム OS RI600PX をご使用いただきまして、誠にありがとうございます。
本資料では、本製品をお使いいただく上での制限事項および注意事項を記載しております。ご使用の前に、必ずお読みくださいますようお願い申し上げます。

1. 製品構成

RI600PX は型名により、以下のように契約形態と提供物が異なります。

型 名	契約形態	提供物
R0R5RX00PCW011	評価契約、インストール可能な PC は 1 台	A
R0R5RX00PCW01A	評価契約、インストール可能な PC は無制限	A
R0R5RX00PCW01K	量産契約、量産数は 3000 台まで	A
R0R5RX00PCW01U	量産契約、量産数は無制限	A
R0R5RX00PCW01Z	量産契約、量産数は無制限、ソース・コード付き	B

提供物は以下となります。

提供物		ツール名	バージョン
B	A	リアルタイム OS RI600PX カーネル オブジェクト	V1.02.00
		コマンドライン・コンフィギュレータ CFG600PX	V 1.01.01.001
		CS+ for CC プラグイン	
		リアルタイム OS ビルド設定プラグイン（共通部）	V3.02.01.01
		リアルタイム OS ビルド設定プラグイン（RI600PX 依存部）	V3.00.00.06
		リアルタイム OS 解析制御プラグイン（共通部）	V3.00.00.03
		リアルタイム OS 解析制御プラグイン（uITRON4 依存部）	V3.00.00.02
		リアルタイム OS 解析制御プラグイン（RI600PX 依存部）	V3.00.00.02
		リアルタイム OS リソース情報表示プラグイン（共通部）	V3.01.00.01
		リアルタイム OS リソース情報表示プラグイン（uITRON4 依存部）	V3.00.00.06
		リアルタイム OS RI600PX カーネル ソース・コード	V1.02.00

2. ユーザーズマニュアルについて

本製品に対応したユーザーズマニュアルを以下に示します。本文書と合わせてお読みください。

マニュアル名	資料番号
RI シリーズ リアルタイム・オペレーティング・システム ユーザーズマニュアル 起動編	R20UT0751JJ0106
RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編	R20UT0964JJ0101
RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル デバッグ編	R20UT0950JJ0100
RI シリーズ リアルタイム・オペレーティング・システム ユーザーズマニュアル メッセージ編	R20UT0756JJ0105

なお、ユーザーズマニュアルは PDF ファイルで提供媒体にパッケージされています。またルネサスエレクトロニクスのホームページから入手することができます。

3. 対象デバイスについて

本製品は、以下のデバイスに対応しています。

- MPU（Memory Protection Unit）を搭載した RX700 シリーズ MCU
- MPU（Memory Protection Unit）を搭載した RX600 シリーズ MCU
- MPU（Memory Protection Unit）を搭載した RX200 シリーズ MCU

4. 動作環境

本製品を使用するには、次の環境が必要になります。

4.1. ハードウェア環境

以下のハードウェア環境に対応しています。

- プロセッサ：1GHz 以上（ハイパー・スレッディング、マルチ・コア CPU に対応）
- メモリ容量：推奨 2GB 以上。最低 1GB 以上（64 ビット版 Windows®では最低 2GB）
- ディスプレイ：1024×768 以上の解像度、65536 色以上

4.2. ソフトウェア環境

以下の OS に対応しています。

- Windows 7（32 ビット版、64 ビット版）
- Windows 8.1（32 ビット版、64 ビット版）
- Windows Vista（32 ビット版、64 ビット版）
- Windows 10（32 ビット版、64 ビット版）

※何れの OS も、最新の Service Pack がインストールされていることを推奨します。

以下のランタイム・ライブラリが必要です。

- .NET Framework 4.5.2
- Microsoft Visual C++ 2010 SP1 ランタイム・ライブラリ

4.3. 対応ツール

本製品は次の開発ツールに対応しています。

ツール名	提供元	バージョン
統合開発環境 CS+ for CC	ルネサス エレクトロニクス	V3.02.00 以降
C/C++コンパイラ CC-RX	ルネサス エレクトロニクス	V2.04.01 以降

5. インストールに関する注意事項

本章では、インストール、アンインストール時の注意事項について説明します。

5.1. インストール時の注意事項

5.1.1. 管理者権限に関する注意事項

インストールするには、Windows®の管理者権限が必要です。

5.1.2. 実行環境に関する注意事項

Windows®には、.NET Framework と Visual C++ のランタイム・ライブラリがインストールされている必要があります（CS+ for CC を実行するために必要です）。

5.1.3. ネットワーク・ドライブに関する注意事項

ネットワーク・ドライブからのインストールはできません。また、ネットワーク・ドライブへのインストールもできません。

5.1.4. インストール先フォルダ名に関する注意事項

インストール先フォルダ名に指定可能な文字は、Windows®に準じます。 / * : < > ? | " ¥ ; 、 の 11 文字は使用できません。また、空白文字ではじまるものと空白文字で終わるものは指定できません。

指定する際に、絶対パスで指定し、相対パスでは指定しないでください。

また、インストール先フォルダの区切り子には ¥ を使用してください。/ は使用しないでください。

5.1.5. 機能の変更や修復に関する注意事項

インストール済みのツールに対して、機能の変更や修復を行う場合は、そのツールのインストール・パッケージを用意し、インストール用プログラムを実行すると起動するプログラムの保守画面で「変更」または「修復」を実行してください。

コントロールパネルの「プログラムと機能」の［変更］ボタンから行うとエラーになります。

5.1.6. インストール・フォルダの変更に関する注意事項

インストール後にできる次のフォルダ（含むフォルダ以下のファイル）には、ツールが動作するために必要なファイル類がありますので削除しないでください。

- Windows®が 32 ビット版で、システムドライブが C:の場合
  C:¥Program Files¥Common Files¥Renesas Electronics CubeSuite+¥
- Windows®が 64 ビット版で、システムドライブが C:の場合
  C:¥Program Files (x86)¥Common Files¥Renesas Electronics CubeSuite+¥

5.1.7. インストールするバージョンに関する注意事項

新しいバージョンがインストールされている場合には､古いバージョンがインストールされない可能性があります。

5.1.8. インストーラの起動に関する注意事項

日本語版以外の Windows®で、インストーラを起動するパスに多バイト文字が含まれているとエラーとなりインストールを実行することができません。

5.1.9. プラグインの有効化

本製品のインストール直後など、本製品のプラグインが無効になっている場合があります。「7.7 プラグインの有効化」にしたがって本製品のプラグインを有効にしてください。

5.2. アンインストール時の注意事項

5.2.1. 管理者権限に関する注意事項

アンインストール（フォルダ／ファイル削除）するには、Windows®の管理者権限が必要です。

5.2.2. アンインストールのフォルダに関する注意事項

ツールのアンインストールの実行順序によっては、フォルダが完全に削除されない場合があります。この場合、アンインストールした後に残ったフォルダは、エクスプローラ等で削除してください。

5.2.3. インストーラ以外での追加／修正に関する注意事項

ツール、および、マニュアル類をインストールしたフォルダに、本製品のインストーラ以外の手段によって、追加または修正されたファイルは、アンインストール時に削除できません。

5.2.4. アンインストール時の選択キーワード

本製品をアンインストールする場合は、2 つの方法があります。

- 統合アンインストーラを使用する（CS+ for CC 自体をアンインストールする）
- 個別にアンインストールする（本製品のみをアンインストールする）

個別にアンインストールを行なう場合、コントロールパネルの

- 「プログラムと機能」

から、以下を削除してください。

- CS+ Realtime OS Common Plugins
- CS+ Realtime OS RI600PX Plugins
- CS+ Realtime OS RI600PX Object Release（量産契約、ソース・コード付き「以外」の場合）
- CS+ Realtime OS RI600PX Source Release（量産契約、ソース・コード付きの場合）

6. 前リリース版との相違点

6.1. カーネルの相違点

カーネルに相違はありません。

6.2. コンフィギュレータの相違点

コンフィギュレータに相違はありません。

6.1 および 6.2 より、カーネル、および、コンフィギュレータに相違がないため、パッケージ・バージョンは前リリース版と同じになっています。

6.3. リアルタイム OS ビルド設定プラグインの相違点

（1） CS+ for CC に対応
CS+ for CC に対応しました。なお、本プラグインは CubeSuite+上では動作しません。

（2） ［リアルタイム OS］タブ、および［システムコンフィギュレーションファイル関連情報］タブからのヘルプジャンプするように変更しました。

6.4. リアルタイム OS リソース情報表示プラグインの相違点

（1） CS+ for CC に対応
CS+ for CC に対応しました。なお、本プラグインは CubeSuite+上では動作しません。

（2） 待ち要因で表示される資源を、ID 番号から名称に変更
待ち要因で表示される資源を、今までは ID 番号で表示していましたが、今版では名称に変更して判別しやすくしました。

（3） リソース選択タブの視認性を向上
リソースを選択するタブを二段にし、さらにリソース名の前にアイコンを付加することで、視認性を向上しました。

（4） メッセージを一部改善
エラー時などに表示されるメッセージを一部改善しました。

（5） 表示メニュー、または、ツールバーのボタンを選択してリアルタイム OS リソース情報パネルを開いても、パネルがアクティブにならない制限を解除しました。

6.5. CS+用サンプル・プログラムの相違点

（1） Firmware Integration Technology モジュールを組み込んだサンプル・プログラムを新規に追加
FIT モジュールを組み込んだ CS+用のサンプル・プログラムを新規に追加しました。詳細は、「9.2 Firmware Integration Technology モジュールを組み込んだサンプル・プログラム」を参照してください。

7. 注意事項

7.1. カーネル・バージョンの区別について

以下の変数を参照することで、カーネル・バージョンを区別することができます。

```
const UW _RI600PX_VERSION = <設定値>;
```

カーネルのバージョンは、X.YY.ZZ.aa の形式で表されます。設定値のビット 31～24 が X、ビット 23～16 が YY、ビット 15～8 が ZZ、ビット 7～0 が aa を表します。

実際のバージョンは以下になります。

カーネル・バージョン （製品バージョン）	_RI600PX_VERSION 値	備考
V1.01.00（V1.01.00, V1.01.01）	(変数の定義なし)	過去のバージョン
V1.02.00.03（V1.02.00）	0x01020003	本バージョン

7.2. 以前のバージョンからの移行

RI600PX の以前のバージョンから移行した場合は、必ずリビルドを行ってください。

7.3. タイマ・テンプレート・ファイル

以下に、RI600PX が提供するタイマ・テンプレート・ファイルと、対応している MCU を示します。

なお、タイマ・テンプレート・ファイルは、システム・コンフィギュレーション・ファイルの"clock.template"に指定するファイルです。

表 7-1 タイマ・テンプレート・ファイル

テンプレート・ファイル	対応 MCU
rx62t.tpl *1	RX600 シリーズ RX62T グループ
rx62n.tpl	RX600 シリーズ RX62G グループ RX600 シリーズ RX62N グループ RX600 シリーズ RX621 グループ
rx630.tpl	RX700 シリーズ RX71M グループ *2 RX600 シリーズ RX65N グループ *2 RX600 シリーズ RX651 グループ *2 RX600 シリーズ RX64M グループ *2 RX600 シリーズ RX630 グループ RX600 シリーズ RX63N グループ RX600 シリーズ RX631 グループ RX600 シリーズ RX634 グループ RX600 シリーズ RX63T グループ RX200 シリーズ RX21A グループ RX200 シリーズ RX230 グループ RX200 シリーズ RX231 グループ RX200 シリーズ RX23T グループ RX200 シリーズ RX24T グループ RX200 シリーズ RX24U グループ

*1 本ファイルは RI600PX V1.02.00 には含まれていないため、別途入手する必要があります。
弊社営業または特約店までお問い合わせください。

*2 システム・コンフィギュレーション・ファイルで、clock.timer に”CMT2”および”CMT3”を指定してはなりません。

7.4. カーネル・ソース・コードのビルド方法

RI600PX カーネルはライブラリで提供されているため、通常はカーネル・ソース・コードをビルドしてカーネル・ライブラリを再生成する必要はありません。ソース・コードが付属するのは、ソース付き量産契約版(R0R5RX00PCW01Z)のみです。

カーネルのソース・コードは、”<インストール・フォルダ>¥src600 に格納されます。カーネルをビルドするためには、カレント・フォルダをこのフォルダとし、以下のように”nmake.exe”[1]を実行してください。これにより、<インストール・フォルダ>¥library”下にライブラリが生成されます。

- ”<インストール・フォルダ>¥library¥rxv1”フォルダのライブラリ生成コマンド
  `nmake release_install(RET)`
  備考：製品添付のライブラリは、CC-RX V1.02.01 でビルドされています。
- ”<インストール・フォルダ>¥library¥rxv2”フォルダのライブラリ生成コマンド
  `nmake –f make_rxv2.mak release_install(RET)`
  備考：製品添付のライブラリは、CC-RX V2.01.00 でビルドされています。

インストール・フォルダに対する書き込み権限がない場合、インストール・フォルダを書き込み可能なフォルダにコピーしてビルドしてください。ビルド後、インストール・フォルダに対する書き込み権限のあるユーザにて、生成されたライブラリをインストール・フォルダの”library¥rxv1”または”library¥rxv2” フォルダにコピーしてください。

1 “nmake.exe”は、米国 Microsoft Corporation により提供されるプロジェクトをビルドするためのツールです。”nmake.exe”は、Microsoft Visual Studio 2008 等に含まれています。

7.5. スタック使用量について

7.5.1. 基本クロック割り込みハンドラのスタック使用量(*clocksz1*、*clocksz2*、*clocksz3*)

「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の付録 D.4 節に記載の *clocksz1*、*clocksz2* および *clocksz3* の値は、以下の通りです。

- *clocksz1*=120
- *clocksz2*=120
- *clocksz3*=200

7.5.2. サービス・コールのスタック使用量(*svcsz*)

カーネルは、システム・スタックを使用します。

「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の付録 D.4 節に記載の *svcsz* は、システムで使用しているサービス・コールの使用サイズ（下記表を参照）と以下の算出式の最大値としてください。

- アクセス例外ハンドラ（_RI_sys_access_exception()）を基点とする関数ツリーでの使用量＋16
- タイマ初期化コールバック関数（_RI_init_cmt_knl()）を基点とする関数ツリーでの使用量＋8

表 7-2 サービス・コールのスタック使用料

	サービス・コール	使用量	備 考
タスク管理機能			
1	cre_tsk	28	
2	acre_tsk	28	
3	del_tsk	28	
4	act_tsk	28	
5	iact_tsk	24	
6	can_act	24	
7	ican_act	24	
8	sta_tsk	28	
9	ista_tsk	24	
10	ext_tsk	60	タスク開始関数からのリターン時にも ext_tsk が呼び出されます。
11	exd_tsk	56	
12	ter_tsk	108	
13	chg_pri	36	
14	ichg_pri	52	
15	get_pri	28	
16	iget_pri	28	
17	ref_tsk	36	
18	iref_tsk	36	
19	ref_tst	28	
20	iref_tst	28	
タスク付属同期機能			
21	slp_tsk	28	
22	tslp_tsk	28	
23	wup_tsk	32	
24	iwup_tsk	48	
25	can_wup	24	
26	ican_wup	24	
27	rel_wai	104	
28	irel_wai	120	
29	sus_tsk	28	
30	isus_tsk	24	
31	rsm_tsk	28	
32	irsm_tsk	24	
33	frsm_tsk	28	
34	ifrsm_tsk	24	
35	dly_tsk	28	
タスク例外処理機能			
36	def_tex	28	
37	ras_tex	28	
38	iras_tex	24	

	サービス・コール	使用量	備 考
39	dis_tex	24	
40	ena_tex	28	
41	sns_tex	24	
42	ref_tex	24	
43	iref_tex	24	
セマフォ			
44	cre_sem	28	
45	acre_sem	28	
46	del_sem	48	
47	sig_sem	32	
48	isig_sem	48	
49	wai_sem	28	
50	pol_sem	24	
51	ipol_sem	24	
52	twai_sem	32	
53	ref_sem	28	
54	iref_sem	28	
イベントフラグ			
55	cre_flg	28	
56	acre_flg	28	
57	del_flg	48	
58	set_flg	48	
59	iset_flg	64	
60	clr_flg	24	
61	iclr_flg	24	
62	wai_flg	32	
63	pol_flg	28	
64	ipol_flg	28	
65	twai_flg	36	
66	ref_flg	28	
67	iref_flg	28	
データ・キュー			
68	cre_dtq	28	
69	acre_dtq	28	
70	del_dtq	48	
71	snd_dtq	32	
72	psnd_dtq	32	
73	ipsnd_dtq	48	
74	tsnd_dtq	36	
75	fsnd_dtq	32	
76	ifsnd_dtq	52	
77	rcv_dtq	32	
78	prcv_dtq	32	
79	iprcv_dtq	48	
80	trcv_dtq	32	
81	ref_dtq	32	
82	iref_dtq	32	

	サービス・コール	使用量	備 考
メールボックス			
83	cre_mbx	28	
84	acre_mbx	28	
85	del_mbx	48	
86	snd_mbx	32	
87	isnd_mbx	52	
88	rcv_mbx	28	
89	prcv_mbx	28	
90	iprcv_mbx	28	
91	trcv_mbx	32	
92	ref_mbx	28	
93	iref_mbx	28	
ミューテックス			
94	cre_mtx	28	
95	acre_mtx	28	
96	del_mtx	52	
97	loc_mtx	28	
98	ploc_mtx	28	
99	tloc_mtx	32	
100	unl_mtx	44	
101	ref_mtx	28	
メッセージ・バッファ			
102	cre_mbf	28	
103	acre_mbf	28	
104	del_mbf	48	
105	snd_mbf	36	
106	psnd_mbf	36	
107	ipsnd_mbf	56	
108	tsnd_mbf	36	
109	rcv_mbf	56	
110	prcv_mbf	56	
111	trcv_mbf	56	
112	ref_mbf	28	
113	iref_mbf	28	
固定長メモリ・プール			
114	cre_mpf	28	
115	acre_mpf	28	
116	del_mpf	48	
117	get_mpf	28	
118	pget_mpf	28	
119	ipget_mpf	28	
120	tget_mpf	32	
121	rel_mpf	32	
122	irel_mpf	48	
123	ref_mpf	28	
124	iref_mpf	28	
可変長メモリ・プール			

	サービス・コール	使用量	備 考
125	cre_mpl	80	
126	acre_mpl	80	
127	del_mpl	48	
128	get_mpl	88	
129	pget_mpl	104	
130	ipget_mpl	104	
131	tget_mpl	88	
132	rel_mpl	100	
133	ref_mpl	28	
134	iref_mpl	28	
時間管理機能			
135	set_tim	28	
136	iset_tim	28	
137	get_tim	28	
138	iget_tim	28	
周期ハンドラ			
139	cre_cyc	28	
140	acre_cyc	28	
141	del_cyc	28	
142	sta_cyc	24	
143	ista_cyc	24	
144	stp_cyc	24	
145	istp_cyc	24	
146	ref_cyc	28	
147	iref_cyc	28	
アラームハンドラ			
148	cre_alm	28	
149	acre_alm	28	
150	del_alm	28	
151	sta_alm	24	
152	ista_alm	24	
153	stp_alm	24	
154	istp_alm	24	
155	ref_alm	28	
156	iref_alm	28	
システム状態管理機能			
157	rot_rdq	28	
158	irot_rdq	24	
159	get_tid	28	
160	iget_tid	28	
161	loc_cpu	24	
162	iloc_cpu	16	
163	unl_cpu	28	

	サービス・コール	使用量	備 考
164	iunl_cpu	24	
165	dis_dsp	16	
166	ena_dsp	28	
167	sns_ctx	24	
168	sns_loc	24	
169	sns_dsp	24	
170	sns_dpn	24	
171	vsta_knl	88	ISP を初期化後に使用します。
172	ivsta_knl	88	
173	vsys_dwn	24	
174	ivsys_dwn	24	
割り込み管理機能			
175	chg_ims	28	
176	ichg_ims	16	
177	get_ims	28	
178	iget_ims	28	
179	カーネル管理割り込みハンドラ	36	カーネル管理割り込みハンドラ終了時に、割り込み発生前のシステム・スタック・ポインタから 36 バイトを使用します。
システム構成管理機能			
180	ref_ver	28	
181	iref_ver	28	
オブジェクト・リセット機能			
182	vrst_dtq	40	
183	vrst_mbx	28	
184	vrst_mbf	40	
185	vrst_mpf	40	
186	vrst_mpl	76	
メモリブジェクト管理機能			
187	ata_mem	48	
188	det_mem	44	
189	sac_mem	60	
190	vprb_mem	28	
191	ref_mem	52	

7.5.3. カーネル・ライブラリをビルドした場合

コンパイラのバージョンやオプション設定を変更してカーネル・ライブラリをビルドした場合、カーネルのスタック使用量が変わる場合があるので、注意してください。

7.6. 大域最適化コンパイル・オプションの注意事項

RI600PX を組み込んだプログラムでは、大域最適化オプション（-ip_optimize、-merge_files、-whole_program）は利用できません。

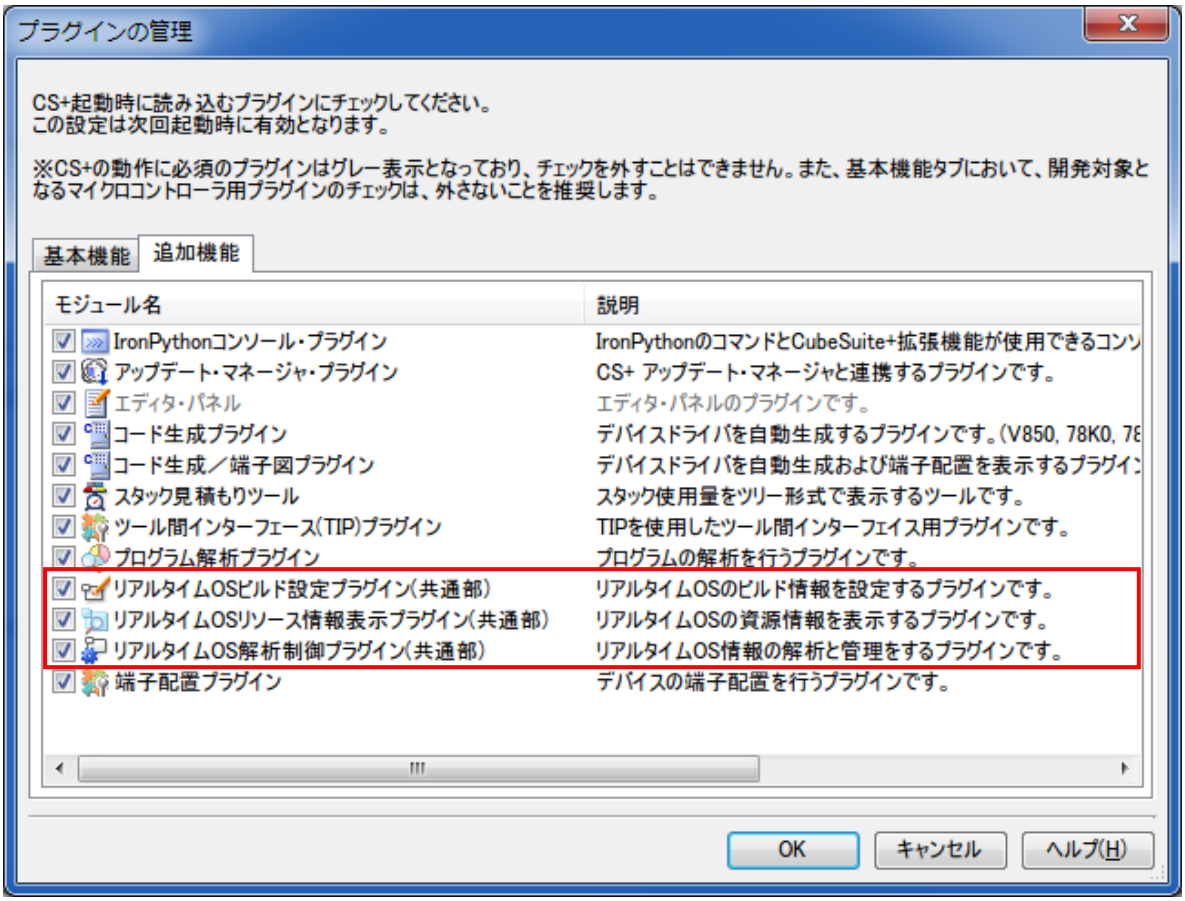
7.7. プラグインの有効化

本製品のインストール直後は、本製品のプラグインが CS+ for CC に読み込まれず、無効になっている場合があります。本製品のプラグインが無効になっていると、ビルドできないなどの問題が生じます。

CS+ for CC の［プラグインの管理］ダイアログの［追加機能］タブで、以下のプラグインを有効にしてください。

- リアルタイム OS ビルド設定プラグイン（共通部）
- リアルタイム OS 解析制御プラグイン（共通部）
- リアルタイム OS リソース情報表示プラグイン（共通部）

図 7-1 プラグインの管理



7.8. CS+のプロジェクト作成

本製品を使用したプロジェクトを作成するには、以下の 2 つの方法があります。

- 本製品添付のサンプル・プロジェクトを流用する
- 新しいプロジェクトを作成する

7.8.1. 本製品添付のサンプル・プロジェクトを流用する

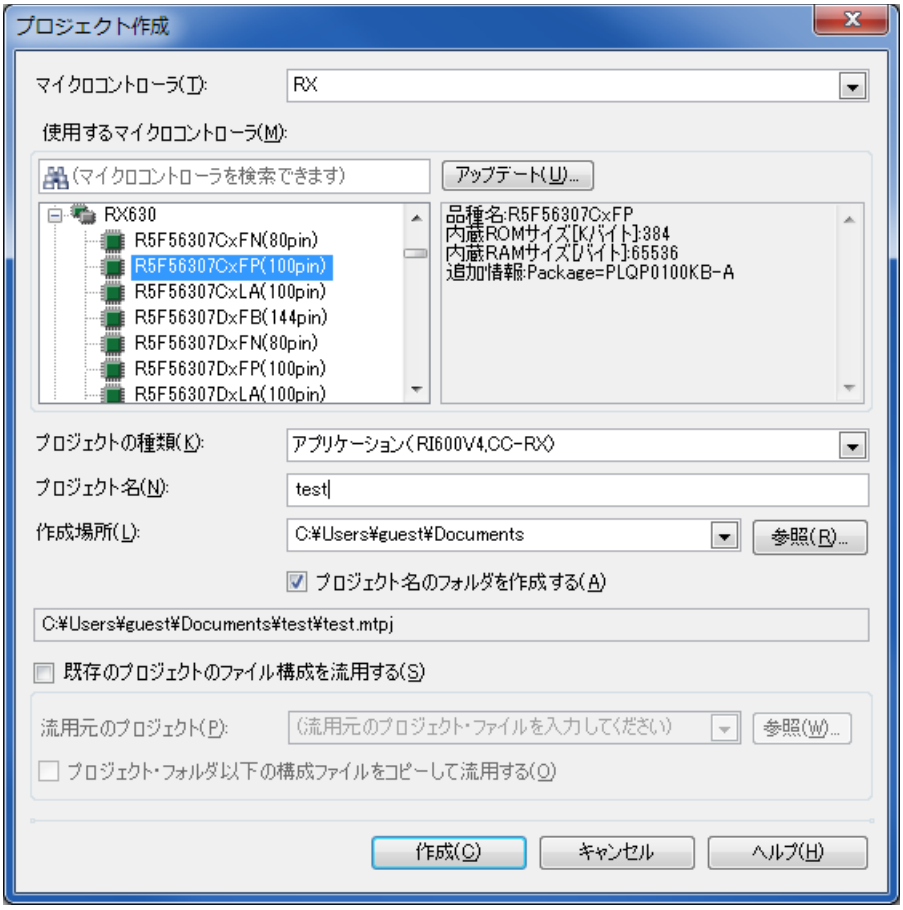
CS+のスタートパネルの［サンプル・プロジェクトを読み込む］エリアで［RX］タブを選択し、”RX???_RI600PX”という名称のプロジェクトを選択してください。

7.8.2. 新しいプロジェクトを作成する

（1） プロジェクトの作成

CS+のスタートパネルの[新しいプロジェクトを作成する]エリアの[GO]ボタンを押し、[プロジェクト作成]ダイアログをオープンします。

図 7-2 プロジェクト作成ダイアログ（プロジェクト新規作成）



- ［マイクロコントローラ］：”RX”を選択してください。
- ［プロジェクトの種類］：”アプリケーション（RI600PX, CC-RX）”を選択してください。

［作成］ボタンを押すと、プロジェクトが作成されます。

（2） ファイルの登録

プロジェクト作成直後は、何もファイルが登録されていません。「RI600PX コーディング編」の「第 2 章 システム構築」を参考に、以下のようなファイルを登録してください。

- タスクやハンドラなどの処理プログラム・ファイル（「RI600PX コーディング編」の 2.2 節を参照）
- システム・コンフィギュレーション・ファイル（「RI600PX コーディング編」の 2.3 節を参照）
- ユーザ・オウン・コーディング部（「RI600PX コーディング編」の 2.4 節を参照）

（3） ビルド・オプションの設定

「RI600PX コーディング編」の「2.5 ロード・モジュールの生成」および「2.6 ビルド・オプション」を参考に、適切なビルド・オプションを設定してください。

7.9. リアルタイム OS リソース情報パネルに関する注意事項

7.9.1. 参照はリアルタイム OS 初期化後に行う

リアルタイム OS リソース情報パネルを参照する場合は、リアルタイム OS 初期化後に参照してください。リアルタイム OS の初期化完了前は、リアルタイム OS リソース情報パネルの表示が不定となります。

7.9.2. デバッグ情報を生成したプログラムを使用する

リアルタイム OS リソース情報パネルを使用する際は、デバッグ情報を生成したプログラムをダウンロードしてください。デバッグ情報がないプログラムをダウンロードして、リアルタイム OS リソース情報パネルを表示しようとした場合、エラーが発生します。

デバッグ情報を生成するには「ビルド・ツール」の「リンク・オプション」のプロパティで「デバッグ情報を出力する」を「はい」に設定してください。

8. 制限事項

8.1. CS+ for CC 使用時の制限事項

8.1.1. リアルタイム OS ビルド設定プラグイン

下記に現状の制限事項を記載します。

（1） **ビルド・モード未対応の制限事項**

下記の制限により、複数のビルド・モードを使用しないでください。

- ビルド・モードごとにコンフィギュレータのオプションを保存しません。そのため、複数のビルド・モードを作成しても、すべてのビルド・モードで同じコンフィギュレータ・オプションで起動します。
- ビルド・モードを切り替えるたびに、ビルド・ツールの「追加のインクルード・パス」に kernel_id.h へのパスが追加されてしまいます。正しいパスはリアルタイム OS ビルド設定プラグインが「システム・インクルード・パス」に設定していますが、IDE が「追加のインクルード・パス」に、ビルド・モードを切り替える前のパスを設定してしまい、ビルド時に IDE が設定したパスを先行して参照します。ビルド・モードを切り替えた後に kernel_id.h が変更されるようなコンフィギュレーション・ファイル編集を行った場合、その変更がビルドに反映されないことになります。

（2） **流用プロジェクト機能に関する制限**

流用元のプロジェクトに sit.s などのコンフィギュレータが生成するファイルが存在しない（クリーンされている状況）かつ、流用元のファイルを「コピーして流用プロジェクトを作成する」という操作が行われた場合、本来グレー表示でプロジェクト・ツリーに登録されている sit.s ファイルなどがプロジェクト・ツリーから削除されてしまいます。

8.1.2. リアルタイム OS リソース情報表示プラグイン

（1） **待ちタスク表示（子ノード表示）で表示リセットを選択すると、タスク・タブの表示がリセットされる制限**

待ちタスクのカラム情報をリセットすると、タスクのカラム情報もリセットします。ただし、表示情報の内容としては問題ありません。

（2） **タスク、周期ハンドラ、アラームハンドラにおける「残り時間」表示で、実際の表示値よりも 1 多い値が表示されることがある**

以下の項目に表示される値が、本来の値より最大で TIC_NUME だけ大きくなる場合があります。

- ［タスク］タブの［残り時間］
- ［周期ハンドラ］タブの［残り時間］
- ［アラームハンドラ］タブの［残り時間］

本来の値は以下のように算出してください。

- 表示された値＞TIC_NUME の場合

  本来の値＝([残り時間]に表示された値) – TIC_NUME
- 表示された値≦TIC_NUME の場合

  本来の値＝0

9. サンプル・プログラム

9.1. CS+用サンプル・プログラム

本章では、RI600PX V1.02.00 が提供するサンプル・プログラム「RX630_RI600PX」について説明します。

9.1.1. 概 要

ドメインは、「マスタ・ドメイン」、「ドメイン A」、「ドメイン B」の 3 つです。

マスタ・ドメイン（domid #1）は、「信頼されたドメイン」です。マスタ・ドメインは、ドメイン A・B の実行に必要な各種オブジェクトを動的に生成します。マスタ・ドメインに属するタスク（MasterDom_Task）は、システム・コンフィギュレーション・ファイルによって静的に生成・起動されます。

ドメイン A（domid #2）とドメイン B（domid #3）は、「信頼されたドメイン」ではありません。

ドメイン A に属するタスクは AppDomA_Task、ドメイン B に属するタスクは AppDomB_Task です。

AppDomA_Task と AppDomB_Task は、セマフォ（ID_SEM1）を使って排他制御しながら共有変数 g_ulSharedData をアクセスします。

また、AppDomA_Task は、データ・キュー（ID_DTQ1）にデータを送信し、AppDomB_Task はそれを受信します。

表 9-1 オブジェクト一覧

種 別	IDなど	解 説
ドメイン	1	マスタ・ドメイン 信頼されたドメイン 所属タスク：MasterDom_Task
	2	ドメイン A 信頼されていないドメイン 所属タスク：AppDomA_Task
	3	ドメイン B 信頼されていないドメイン 所属タスク：AppDomB_Task
タスク	ID_MASTERDOMTASK	システム・コンフィギュレーション・ファイルで生成・起動される。
	ID_DOM_A_TASK	MasterDom_Taskによって生成・起動される。
	ID_DOM_B_TASK	MasterDom_Taskによって生成・起動される。
セマフォ	ID_SEM1	MasterDom_Task によって、生成される。 AppDomA_Task と AppDomB_Task からの変数"g_ulSharedData"へのアクセスの排他制御に使用。
データ・キュー	ID_DTQ1	MasterDom_Task によって、生成される。 AppDomA_Task と AppDomB_Task の間での通信に使用。 データ・キュー領域は BS セクションとして生成される。BS セクションは非メモリ・オブジェクト。
可変長メモリ・プール	ID_MPL1	MasterDom_Task によって、生成される。 単なるダミー プール領域は BU_SH セクションであり、memory_object[4]内。
周期ハンドラ	ID_CYC1	システム・コンフィギュレーション・ファイルで生成・開始される。 AppDomA_Task と AppDomB_Task のレディキューを回転する。
アラームハンドラ	ID_ALM1	システム・コンフィギュレーション・ファイルで生成される。 単なるダミー

割り込みハンドラ	可変ベクタ#64	システム・コンフィギュレーション・ファイルで定義される。 単なるダミー

9.1.2. ファイル構成

「RX630_RI600PX」サンプル・プログラムは、以下のフォルダに格納されています。

<CS+_root>¥SampleProjects¥RX¥RX630_RI600PX

- <CS+_root>
  CS+ のインストール・フォルダを表しています。
  デフォルトでは、” C:¥Program Files¥Renesas Electronics¥CS+¥CC”となります。

（1） appli¥source¥reset フォルダ

- resetprg.c
  ブート処理ファイルです。ブート処理ファイルについては、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の 17.2 節を参照してください。
- dbsct.c
  セクション情報ファイルです。セクション情報ファイルについては、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の 17.4 節を参照してください。

（2） appli¥source¥kernel フォルダ

- sample.cfg
  システム・コンフィギュレーション・ファイルです。システム・コンフィギュレーション・ファイルについては、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の第 20 章を参照してください。
- access_exc.c
  アクセス例外ハンドラです。アクセス例外ハンドラについては、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の 3.10 節を参照してください。
- init_cmt.c
  基本クロック用タイマ初期化ルーチンです。基本クロック用タイマ初期化ルーチンについては、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の 10.9 節を参照してください。
- sysdwn.c
  システム・ダウン・ルーチンです。システム・ダウン・ルーチンについては、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」の 15.2 節を参照してください。
- handler.c
  各種ハンドラです。

（3） appli¥source¥master_dom フォルダ

- master_dom.c
  マスタ・ドメインに所属するタスクです。

（4） appli¥source¥dom_A フォルダ

- dom_A.c
  ドメイン A に所属するタスクです。

（5） appli¥source¥dom_B フォルダ

- dom_B.c
  ドメイン B に所属するタスクです。

（6） appli¥source¥common フォルダ

- common.c
  ドメイン間で共有する関数とデータです。

9.1.3. メモリ・マップ

[ ]で示したセクションは、16 バイト境界に配置するためにリンカの aligned_section オプションを指定しています。詳細は、「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」マニュアルの 2.6.4 節を参照してください。

9.1.3.1. RAM 領域

表 9-2 RAM 領域

アドレス	セクション並び（リンカ設定）	解説	メモリ・オブジェクト
0～0x0001FFFF	SI	システム・スタック	非メモリ・オブジェクト
	BRI_RAM, RRI_RAM	RI600PXデータ	
	BS, BS_1, BS_2, RS, RS_1, RS_2	ハンドラ専用データ	
	[SURI_STACK]	ユーザ・スタック	
	[BU_MASTERDOM],BU_MASTERDOM_1, BU_MASTERDOM_2, RU_MASTERDOM,RU_MASTERDOM_1, RU_MASTERDOM_2	マスタ・ドメイン専用データ	memory_object[1]
	[BU_DOM_A], BU_DOM_A_1, BU_DOM_A_2, RU_DOM_A, RU_DOM_A_1, RU_DOM_A_2	ドメインA専用データ	memory_object[2]
	[BU_DOM_B], BU_DOM_B_1, BU_DOM_B_2, RU_DOM_B,RU_DOM_B_1,RU_DOM_B_2	ドメインB専用データ	memory_object[3]
	[BURI_HEAP], BU_SH, BU_SH_1, BU_SH_2, RU_SH,RU_SH_1,RU_SH_2	ドメイン間共有データ	memory_object[4]

9.1.3.2. ROM 領域

表 9-3 ROM 領域

アドレス	セクション並び（リンカ設定）	解説	メモリ・オブジェクト
0xFFFF0000～ 0xFFFFFF7F	[PU_MASTERDOM], CU_MASTERDOM, CU_MASTERDOM_1, CU_MASTERDOM_2, DU_MASTERDOM, DU_MASTERDOM_1, DU_MASTERDOM_2	マスタ・ドメイン専用コード・定数	memory_object[5]
	[PU_DOM_A],CU_DOM_A,CU_DOM_A_1, CU_DOM_A_2,DU_DOM_A,DU_DOM_A_1, DU_DOM_A_2	ドメインA専用コード・定数	memory_object[6]
	[PU_DOM_B],CU_DOM_B,CU_DOM_B_1, CU_DOM_B_2,DU_DOM_B,DU_DOM_B_1, DU_DOM_B_2	ドメインB専用コード・定数	memory_object[7]
	[PU_SH], WU_SH, WU_SH_1, WU_SH_2, LU_SH, CU_SH, CU_SH_1, CU_SH_2, DU_SH,DU_SH_1,DU_SH_2	ドメイン間共有コード・定数	memory_object[8]
	INTERRUPT_VECTOR	可変ベクタ・テーブル	非メモリ・オブジェクト
	PRI_KERNEL	RI600PXコード	
	CRI_ROM, DRI_ROM	RI600PX定数	
	C$*,PS,CS,CS_1,CS_2,DS,DS_1,DS_2	ハンドラ専用コード・定数	
0xFFFFFF80～ 0xFFFFFFFF	FIX_INTERRUPT_VECTOR	固定ベクタ・テーブル	

9.1.3.3. メモリ・オブジェクト

メモリ・オブジェクトは全部で 8 個あり、システム・コンフィギュレーション・ファイルで登録します。以下に、システム・コンフィギュレーション・ファイルでのメモリ・オブジェクト登録内容を示します。

（1） memory_object[1]：マスタ・ドメイン専用データ

```
memory_object[1]{
    start_address = BU_MASTERDOM;
    end_address   = RU_MASTERDOM_2;
    acptn1        = 0x0001;          マスタ・ドメインのみオペランド・リード・アクセス許可
    acptn2        = 0x0001;          マスタ・ドメインのみオペランド・ライト・アクセス許可
    acptn3        = 0;               全ドメインの実行アクセス禁止
};
```

（2） memory_object[2]：ドメイン A 専用データ

```
memory_object[2]{
    start_address = BU_DOM_A;
    end_address   = RU_DOM_A_2;
    acptn1        = 0x0002;          ドメイン A のみオペランド・リード・アクセス許可
    acptn2        = 0x0002;          ドメイン A のみオペランド・ライト・アクセス許可
    acptn3        = 0;               全ドメインの実行アクセス禁止
};
```

(3) memory_object[3] : ドメイン B 専用データ

```
memory_object[3]{
    start_address = BU_DOM_B;
    end_address   = RU_DOM_B_2;
    acptn1        = 0x0004;              ドメイン B のみオペランド・リード・アクセス許可
    acptn2        = 0x0004;              ドメイン B のみオペランド・ライト・アクセス許可
    acptn3        = 0;                   全ドメインの実行アクセス禁止
};
```

(4) memory_object[4] : ドメイン間共有データ

```
memory_object[4]{
    start_address = BURI_HEAP;
    end_address   = RU_SH_2;
    acptn1        = TACP_SHARED;         全ドメインのオペランド・リード・アクセス許可
    acptn2        = TACP_SHARED;         全ドメインのオペランド・ライト・アクセス許可
    acptn3        = 0;                   全ドメインの実行アクセス禁止
};
```

(5) memory_object[5] : マスタ・ドメイン専用コード・定数

```
memory_object[5]{
    start_address = PU_MASTERDOM;
    end_address   = DU_MASTERDOM_2;
    acptn1        = 0x0001;              マスタ・ドメインのみオペランド・リード・アクセス許可
    acptn2        = 0;                   全ドメインのオペランド・ライト・アクセス禁止
    acptn3        = 0x0001;              マスタ・ドメインのみ実行アクセス許可
};
```

(6) memory_object[6] : ドメイン A 専用コード・定数

```
memory_object[6]{
    start_address = PU_DOM_A;
    end_address   = DU_DOM_A_2;
    acptn1        = 0x0002;              ドメイン A のみオペランド・リード・アクセス許可
    acptn2        = 0;                   全ドメインのオペランド・ライト・アクセス禁止
    acptn3        = 0x0002;              ドメイン A のみ実行アクセス許可
};
```

（7） memory_object[7]：ドメイン B 専用コード・定数

```
memory_object[7]{
    start_address = PU_DOM_B;
    end_address   = DU_DOM_B_2;
    acptn1       = 0x0004;          ドメイン B のみオペランド・リード・アクセス許可
    acptn2       = 0;               全ドメインのオペランド・ライト・アクセス禁止
    acptn3       = 0x0004;          ドメイン B のみ実行アクセス許可
};
```

（8） memory_object[8]：ドメイン間共有コード・定数

```
memory_object[8]{
    start_address = PU_SH;
    end_address   = DU_SH_2;
    acptn1       = TACP_SHARED;     全ドメインのオペランド・リード・アクセス許可
    acptn2       = 0;               全ドメインのオペランド・ライト・アクセス禁止
    acptn3       = TACP_SHARED;     全ドメインの実行アクセス許可
};
```

9.1.3.4. ユーザ・スタック

ユーザ・スタックはメモリ・オブジェクト外とする必要があります。本サンプルでは、全タスクのユーザ・スタックをデフォルトと同じ SURI_STACK セクションとしています。

（1） MasterDom_Task のユーザ・スタック

MasterDom_Task は、システム・コンフィギュレーション・ファイルで静的に生成されます。

```
task[]{
    name          = ID_MASTERDOMTASK;
    entry_address   = MasterDom_Task();
    initial_start   = ON;
    stack_size     = 256;
    priority       = 1;
//  stack_section   = SURI_STACK;   "stack_section" を省略すると、ユーザ・スタックは
    exinf         = 1;              SURI_STACK セクションに生成されます。
    domain_num     = 1;
};
```

（2） AppDomA_Task と AppDomB_Task のユーザ・スタック

AppDomA_Task と AppDomB_Task は、MasterDom_Task が呼び出す acre_tsk サービス・コールによって動的に生成されます。acre_tsk には、それぞれのタスクのユーザ・スタックの先頭アドレスとサイズを渡します。

AppDomA_Task と AppDomB_Task のスタック領域は、共に master_dom.c にて#pragma section ディレクティブを使って SURI_STACK セクションに生成しています。

```
////////////////////////////////////////////////////////////////////////
// Stack for AppDomA_Task and AppDomB_Task
////////////////////////////////////////////////////////////////////////
#pragma section B SURI_STACK
static UW   s_ulDomA_Stk[DOM_A_STKSZ/sizeof(UW)];   // Stack area for AppDomA_Task
static UW   s_ulDomB_Stk[DOM_B_STKSZ/sizeof(UW)];   // Stack area for AppDomB_Task
```

9.1.4. セクションに関するビルド・ツールの設定

9.1.4.1. 標準ライブラリ構築ツール

標準ライブラリのセクションは全ドメインからアクセス可能なメモリ・オブジェクトとしています。

表 9-4 標準ライブラリのセクション

領 域	セクション	メモリ・オブジェクト
コード	PU_SH	memory_object[8]
定数	CU_SH	
リテラル	LU_SH	
switch文分岐テーブル	WU_SH, WU_SH_1, WU_SH_2	
初期化データ	DU_SH,DU_SH_1,DU_SH_2	
未初期化データ	BU, BU_SH_1, BU_SH_2	memory_object[4]
初期化データ（RAM）（リンカで指定）	RU_SH,RU_SH_1,RU_SH_2	

9.1.4.2. C/C++コンパイラ

デフォルトのセクションは表 9-4 と同じとし、これ以外のセクションとする場合には、個別にソース・コードに #pragma section ディレクティブを記述することでセクションを切り替えることとしています。

9.1.4.3. リンカ

「RI600PX リアルタイム・オペレーティング・システム ユーザーズマニュアル コーディング編」マニュアルの 2.6.4 節に記載のように、以下のセクションについて、aligned_section オプションを指定する必要があります。

- システム・コンフィギュレーション・ファイルで、memory_object[].start_address に指定したセクション
- システム・コンフィギュレーション・ファイルで、task[].stack_section に指定したセクション（本サンプルでは指定していません）
- SURI_STACK セクション

このため、メモリ・オブジェクトの先頭セクションと SURI_STACK に aligned_section を指定しています。

9.1.5. アクセス違反の対処例

本サンプルでは、以下の例を実装しています。詳細はサンプル・コードを参照してください。

- AppDomA_Task：タスク例外を発生させ、タスク例外処理ルーチンでタスクを再起動する。
- AppDomB_Task：longjmp( )を使って、正常な時点からやり直す。

9.2. Firmware Integration Technology モジュールを組み込んだサンプル・プログラム

本章では、Firmware Integration Technology（以下 FIT と称す）モジュールを組み込んだサンプル・プログラムについて説明します。

9.2.1. 概 要

本サンプル・プログラムは、Renesas Starter Kits ボードで動作するように構成されています。 本サンプル・プログラムをご利用頂くことにより、アプリケーションをより迅速かつ容易に開発することが可能となります。 本サンプル・プログラムは、既存の「RX630_RI600PX」などに必要最小限の FIT モジュールを追加したものです。 ハードウェアの初期化に FIT モジュールを使用していることを除いて、基本的な動作は同じです。

9.2.2. FIT 対応サンプル・プログラムの構成

FIT に対応したサンプル・プログラムは、CS+のサンプル・プロジェクトとして提供します。

今回提供する FIT 対応 RI600PX サンプル・プロジェクト、および使用した FIT モジュールを以下に示します。

- FIT 対応 RI600PX サンプル・プロジェクト

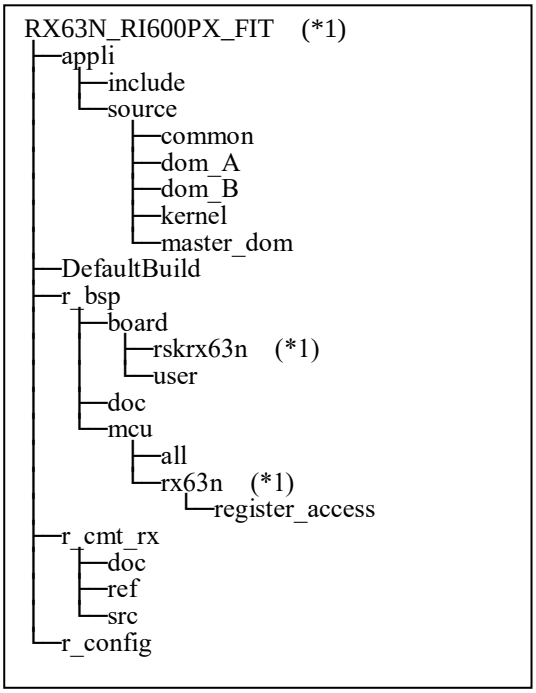
サンプル・プロジェクト名	対象 Renesas Starter Kits ボード名
RX63N_RI600PX_FIT	Renesas Starter Kit+ for RX63N
RX64M_RI600PX_FIT	Renesas Starter Kit+ for RX64M
RX65N_RI600PX_FIT	Renesas Starter Kit+ for RX65N
RX71M_RI600PX_FIT	Renesas Starter Kit+ for RX71M

- 使用した FIT モジュール

FIT モジュール	FIT モジュール名	リビジョン
ボードサポートパッケージ（BSP）	r_bsp	Dec.01.15 Rev.3.10 Oct.01.16 Rev.3.40（RX65N 用）
コンペアマッチタイマ（CMT）	r_cmt_rx	Jun.30.15 Rev.2.60 Oct.01.16 Rev.3.00（RX65N 用）

9.2.3. FIT 対応 RI600PX サンプル・プロジェクトのディレクトリ構成

RX63N 用 FIT 対応 RI600PX サンプル・プロジェクトのディレクトリ構成を以下に示します。



*1 その他のサンプル・プロジェクトでは、各々対応した MCU 名と RSK ボード名に置き換わります。

FIT モジュール（r_bsp と r_cmt_rx）はプロジェクトのルートディレクトリに配置しています。
FIT モジュールの設定変更用ヘッダファイルは r_config フォルダにまとめて格納します。

9.2.4. FIT 対応 RI600PX サンプル・プロジェクトの変更点

FIT 対応 RI600PX サンプル・プロジェクトでは、FIT モジュール及び基にしたサンプル・プログラムを一部変更しています。 FIT モジュールは基本的に RTOS あり／なしのプロジェクトで使えるように変更しています。

本章では主な変更点について説明します。

（1） FIT モジュールによる RSK ボードの初期化

【対象ファイル】

r_bsp¥board¥<RSK ボード名>¥resetprg.c

r_bsp¥board¥<RSK ボード名>¥dbsct.c

※RSK ボード名：rskrx63n, rskrx64m, rskrx65n, rskrx71m

【変更内容】

FIT モジュールのボードサポートパッケージ（r_bsp）を使って RSK ボードを初期化します。
そのため元のサンプル・プログラムから重複する以下のファイルを削除しました。

appli¥source¥reset¥resetprg.c

appli¥source¥reset¥dbsct.c

スタートアップ・ルーチン（resetprg.c 内の PowerON_Reset_PC 関数）では BSP_CFG_RTOS_USED マクロを使って、OS 無し又は RI600PX の場合で処理を切り替えるように

しています。

割り込みベクタ及び固定／例外ベクタの先頭アドレスは OS 無し又は RI600PX の場合で異なります。

OS 無しの場合にはユーザーモードで main 関数を呼び出し、RI600PX の場合にはスーパーバイザ・モードで vsta_knl を呼び出します。

```
void PowerON_Reset_PC(void)
{
#if BSP_CFG_RTOS_USED == 0       /* Non-OS */
      set_intb((void *)__sectop("C$VECT"));
#ifdef __RXV2
      set_extb((void *)__sectop("EXCEPTVECT"));/* RXv2 command */
#endif/* __RXV2 */
#elif BSP_CFG_RTOS_USED == 1    /* FreeRTOS */
#elif BSP_CFG_RTOS_USED == 2    /* SEGGER embOS */
#elif BSP_CFG_RTOS_USED == 3    /* Micrium MicroC/OS */
#elif BSP_CFG_RTOS_USED == 4    /* Renesas RI600V4 & RI600PX */
      set_intb((void *)__sectop("INTERRUPT_VECTOR"));
#ifdef __RXV2
      set_extb((void *)__sectop("FIX_INTERRUPT_VECTOR"));/* RXv2 command */
#endif/* __RXV2 */
#endif/* BSP_CFG_RTOS_USED */

      （省略）

#if BSP_CFG_RTOS_USED == 0       /* Non-OS */
      nop();
      set_psw(PSW_init);
#if BSP_CFG_RUN_IN_USER_MODE==1
      chg_pmusr() ;
#endif
      main();
#if BSP_CFG_IO_LIB_ENABLE == 1
      _CLOSEALL();
#endif
      while(1)
      {
            /* Infinite loop. Put a breakpoint here if you want to catch an exit of main(). */
      }
#elif BSP_CFG_RTOS_USED == 1    /* FreeRTOS */
#elif BSP_CFG_RTOS_USED == 2    /* SEGGER embOS */
#elif BSP_CFG_RTOS_USED == 3    /* Micrium MicroC/OS */
#elif BSP_CFG_RTOS_USED == 4    /* Renesas RI600PX */
      /* Make sure to disable interrupt. */
      clrpsw_i();
      vsta_knl();          /* Start RI600PX and never return */
      brk();
#endif/* BSP_CFG_RTOS_USED */
}
```

RX シミュレータによるデバッグの際、無限待ちを防止するため、USE_SIM_DEBUG マクロでクロック初期化ルーチンをスキップできるようにしています。

詳細は「9.2.5 FIT 対応 RI600PX サンプル・プロジェクトの注意点」の「（6）RX シミュレータによるデバッグ」を参照してください。

dbsct.c では BSP_CFG_RTOS_USED マクロを使うことにより、初期化セクション設定を OS 無し又は RI600PX の場合で切り替えています。

（2） 割り込みベクタの集約

【対象ファイル】

r_bsp¥board¥<RSK ボード名>¥vecttbl.c

r_bsp¥mcu¥<MCU 名>¥mcu_interrupts.c

r_cmt_rx¥src¥r_cmt_rx.c

appli¥source¥kernel¥sample.cfg

※RSK ボード名：rskrx63n, rskrx64m, rskrx65n, rskrx71m

※MCU 名：rx64m, rx65n, rx71m

【変更内容】

FIT モジュールで定義されている割り込みベクタの登録を RTOS 側のシステム・コンフィギュレーション・ファイル（sample.cfg）へ集約します。

vecttbl.c では BSP_CFG_RTOS_USED マクロを使って、以下の記述を除外しています。

・割り込みハンドラ関数の#pragma interrupt

・割り込みベクタ・テーブルの定義（#pragma section C FIXEDVECT 以降）

```
#if BSP_CFG_RTOS_USED == 0       /* Non-OS */
#pragma interrupt (non_maskable_isr)
#elif BSP_CFG_RTOS_USED == 1    /* FreeRTOS */
#elif BSP_CFG_RTOS_USED == 2    /* SEGGER embOS */
#elif BSP_CFG_RTOS_USED == 3    /* Micrium MicroC/OS */
#elif BSP_CFG_RTOS_USED == 4    /* Renesas RI600V4 & RI600PX */
#endif
void non_maskable_isr(void)
{
        （省略）
}

#if BSP_CFG_RTOS_USED == 0       /* Non-OS */
#pragma section C FIXEDVECT

void * const Fixed_Vectors[] =
{
      （省略）
      (void *) non_maskable_isr,         /* 0xffffffff8    NMI */
      (void *) PowerON_Reset_PC      /* 0xfffffffc    RESET */
};
#elif BSP_CFG_RTOS_USED == 1    /* FreeRTOS */
#elif BSP_CFG_RTOS_USED == 2    /* SEGGER embOS */
#elif BSP_CFG_RTOS_USED == 3    /* Micrium MicroC/OS */
#elif BSP_CFG_RTOS_USED == 4    /* Renesas RI600V4 & RI600PX */
#endif
```

RX64M, RX65N, RX71M では mcu_interrupts.c 内の以下のグループ割り込みハンドラから#pragma interrupt を除外しています。

group_al0_handler_isr

group_al1_handler_isr

group_bl0_handler_isr

group_bl1_handler_isr

group_bl2_handler_isr（RX65N 用のみ）

上記の割り込みハンドラ関数はすべて sample.cfg ファイルに登録します。

```
// BSP Interrupt Handler Definition (VECT_ICU_GROUPBL0)
interrupt_vector[110]{
        os_int                  = YES;
        entry_address       = group_bl0_handler_isr();
        pragma_switch       = E,ACC;
};
```

r_cmt_rx.c では、以下の FIT タイマ API 用の割り込みハンドラから#pragma interrupt 行と static 宣言を除外しています。

cmt0_isr

cmt1_isr

cmt2_isr

cmt3_isr

上記の割り込みハンドラ関数は sample.cfg ファイルで以下のように登録しています。

【RX71M, RX65N, RX64M の場合】

割り込みベクタ 29 ：cmt1_isr

割り込みベクタ 128：cmt2_isr

割り込みベクタ 129：cmt3_isr

【RX63N の場合】

割り込みベクタ 29：cmt1_isr

割り込みベクタ 30：cmt2_isr

割り込みベクタ 31：cmt3_isr

（3） RTOS 用ヘッダファイルを FIT 側でインクルード

【対象ファイル】

r_bsp¥board¥<RSK ボード名>¥r_bsp.h

※RSK ボード名：rskrx63n, rskrx64m, rskrx65n, rskrx71m

【変更内容】

以下のヘッダファイルを r_bsp.h でインクルードしています。

kernel.h

kernel_id.h

```
#if BSP_CFG_RTOS_USED == 0       /* Non-OS */
#elif BSP_CFG_RTOS_USED == 1    /* FreeRTOS */
#elif BSP_CFG_RTOS_USED == 2    /* SEGGER embOS */
#elif BSP_CFG_RTOS_USED == 3    /* Micrium MicroC/OS */
#elif BSP_CFG_RTOS_USED == 4    /* Renesas RI600V4 & RI600PX */
#include       "kernel.h"
#include       "kernel_id.h"
#endif/* BSP_CFG_RTOS_USED */
```

r_bsp.h は platform.h 内でインクルードされているので、RTOS のソースでは platform.h のみをインクルードします。

（4） r_cmt_rx モジュールで RI600PX が使用するタイマーリソースを除外

【対象ファイル】

r_cmt_rx¥src¥r_cmt_rx.c

r_config¥r_cmt_rx_config.h

【変更内容】

本モジュールのタイマ API では、RI600PX のシステム時刻更新用（_RI_CLOCK_TIMER マクロ）の CMT チャンネルを除外しています。 また FIT モジュールを RI600V4 と共用するため、トレース用の CMT チャンネルを示す_RI_TRACE_TIMER マクロ（ダミー）を r_cmt_rx_config.h で定義しています。_RI_CLOCK_TIMER と_RI_TRACE_TIMER には同じ値を設定しています。

```
bool R_CMT_Stop (uint32_t channel)
{
    /* Make sure valid channel number was input. */
    if (channel >= CMT_RX_NUM_CHANNELS)
    {
        /* Invalid channel number was used. */
        return false;
    }

#if BSP_CFG_RTOS_USED == 0      /* Non-OS */
#elif BSP_CFG_RTOS_USED == 1    /* FreeRTOS */
#elif BSP_CFG_RTOS_USED == 2    /* SEGGER embOS */
#elif BSP_CFG_RTOS_USED == 3    /* Micrium MicroC/OS */
#elif BSP_CFG_RTOS_USED == 4    /* Renesas RI600V4 & RI600PX */
    /* Exclude RTOS timers */
    if (channel == _RI_CLOCK_TIMER || channel == _RI_TRACE_TIMER)
    {
        return false;
    }
#endif/* BSP_CFG_RTOS_USED */

    /* Stop counter. */
    cmt_counter_stop(channel);
```

（5） r_cmt_rx モジュールで RTOS 予約チャンネルの初期状態を設定

【対象ファイル】

r_cmt_rx¥src¥r_cmt_rx.c

【変更内容】

CMT チャンネルの利用状況を記録する g_cmt_modes 配列に RTOS 側で使用する CMT チャンネルの初期値（CMT_RX_MODE_PERIODIC）を設定しています。

※ペアとなる CMT チャンネルのパワーダウン防止のため。

（6） RSK ボード上の LED 制御

【対象ファイル】

appli¥include¥hw_control.h

appli¥source¥common¥hw_control.c

appli¥source¥kernel¥sample.cfg

【変更内容】

RSK ボード上の LED の点灯／消灯を制御する set_LED 関数を作成しました。

（7） デバッグ用メッセージ出力

【対象ファイル】

appli¥include¥rtos_sample_config.h

【変更内容】

本サンプルでは RX シミュレータ／E1 エミュレータでデバッグ中に、printf 関数を使ってデバッグ・コンソールへ任意のメッセージを出力できます。 ただし、サンプル・プログラムでは printf 関数は直接呼び出さずに、rtos_sample_config.h で定義した DEBUG_print マクロを使用します。
DEBUG_print マクロは同じヘッダファイルで定義した以下のマクロにより有効／無効を制御できます。

USE_DEBUG_MESSAGE （定義ありの場合、デバッグ・コンソールにメッセージを出力）

（8） メモリアクセス例外発生時のハンドラの呼出し

【対象ファイル】

r_bsp¥board¥<RSK ボード名>¥vecttbl.c

appli¥source¥kernel¥access_exc.c

※RSK ボード名：rskrx64m, rskrx65n, rskrx71m

【変更内容】

FIT では例外ベクタ 21 番にアクセス例外ハンドラ excep_access_isr を登録しますが、RI600PX ではタスク例外で同じ例外ベクタ 21 番を使用しています。

そこで RI600PX のアクセス例外ハンドラ_RI_sys_access_exception（access_exc.c）から excep_access_isr 関数を呼び出すようにしました。

（9） サンプル・プログラムの変更

【対象ファイル】

appli¥source¥common¥common.c

appli¥source¥dom_A¥dom_A.c

appli¥source¥dom_B¥dom_B.c

appli¥source¥kernel¥access_exc.c

appli¥source¥kernel¥handler.c

appli¥source¥kernel¥init_cmt.c

appli¥source¥kernel¥sysdwn.c

appli¥source¥master_dom¥master_dom.c

【変更内容】

・すべての対象 C ソースで kernel.h と kernel_id.h の代わりに platform.h をインクルードします。
・タスクおよびタスク例外ハンドラ内に DEBUG_print マクロを使ったメッセージ出力を追加。
・タスク MasterDom_Task（master_dom.c）の最後で LED1 を点灯。
・タスク AppDomA_Task（dom_A.c）先頭で LED2 を消灯。

・タスク例外ハンドラ AppDomA_TaskTex（dom_A.c）の実行中に LED2 を点灯。
・タスク AppDomB_Task（dom_B.c）先頭で LED3 を消灯。
・タスク例外ハンドラ AppDomB_TaskTex（dom_B.c）の実行中に LED3 を点灯。
・周期ハンドラ cyh1（handler.c）の呼び出し回数を分周して LED0 の点灯／消灯を切り替えます。
分周には rtos_sample_config.h で定義した LED_BLINK_DIV_RATIO マクロを使用。
・システム・ダウン・ルーチン_RI_sys_dwn__（sysdwn.c）でエラーメッセージを
デバッグ・コンソールに出力します。
・_RI_init_cmt_knl 関数（init_cmt.c 内）で r_bsp の API を使って CMT チャンネルをロックします。

（10） 個別コンパイル・オプションの設定

【対象ファイル】

r_bsp¥board¥<RSK ボード名> ¥resetprg.c

※RSK ボード名： rskrx63n, rskrx64m, rskrx65n, rskrx71m

【変更内容】

スタック領域を 4 バイト境界に配置するため、個別コンパイル・オプションを設定しています。
設定内容は、[その他]-[その他の追加オプション] に”-nostuff”を追加しています。

（11） CC-RX（ビルド・ツール）のオプション変更

ビルド関連で以下のオプションを変更しています。

【コンパイル・オプション】

・[ソース]-[C ソース・ファイルの言語]を C89 から C99 に変更しています。
・[最適化]-[最適化レベル]を 2 から 0 に変更しています。

【リンク・オプション】

・[リスト]-[シンボル情報を出力する] を「はい」に変更しています。

【ライブラリ・ジェネレート・オプション】

・[標準ライブラリ]-[ライブラリ構成]を C89 から C99 に変更しています。

9.2.5. FIT 対応 RI600PX サンプル・プロジェクトの注意点

本章では FIT 対応 RI600PX サンプル・プロジェクトご使用時の主な注意点について説明します。

（1） RI600PX サンプルで使用するセクション名について

RI600PX サンプルで使用しているセクション名は殆どがアプリケーション依存になります。
セクション名を追加／変更する時は、以下の変更を忘れないようにしてください。

- ソース内でのセクション配置先の指定（#pragma section）
- リンク・オプションのセクション設定
  1．セクションの開始アドレス（順番）
  2．セクション・アライメント
  3．ROM から RAM へマップするセクション
- dbsct.c の初期化セクション設定
- システム・コンフィギュレーション・ファイル（sample.cfg）のメモリ・オブジェクト設定

（2） セクション配置について

セクション配置は sample.cfg に記述されたメモリ・オブジェクトと深く関係しています。
メモリ・オブジェクトでは２つのセクション名を使って範囲を指定します。
範囲指定は複数のセクションを挟んでいます。
セクションの順番を変更する場合は充分に注意してください。

（3） CMT チャンネルの制約

RI600PX ではデフォルトで CMT0 をシステム時刻の更新に使用しています。
CMT1 以降のチャンネルは FIT のタイマ API で動的に利用されます。

（4） エミュレータからの電源供給

本サンプルではデフォルトでエミュレータ（USB）から電源供給する設定にしていますが、
開発の段階で消費電流が増加して USB 電源では供給能力が不足する可能性があるため、RX64M 以降の RSK ボードでは外部電源を使用して開発を行ってください。
「RX E1(JTAG)（デバッグ・ツール）」のプロパティの「接続用設定」タブで以下を変更してください。
[ターゲット・ボードとの接続]-[エミュレータから電源供給をする（最大 200mA）]

（5） FIT モジュールの更新

本サンプルに付属する FIT モジュール（r_bsp と r_cmt_rx）は、RTOS 用にカスタマイズされています。そのため、それらを該当 FIT モジュールの最新バージョンで上書きしないでください。

（6） RX シミュレータによるデバッグ

RX71M, RX65N, RX64M のクロック初期化ルーチンはエミュレータを前提としているため、
RX シミュレータでデバッグするとレジスタの読み出しで無限待ちループに入ります。
この問題を回避するため、コンパイル・オプションの[ソース]-[マクロ定義]で USE_SIM_DEBUG マクロを定義するか、resetprg.c の中の以下のコメントを外してビルドしてください。

```
//#define USE_SIM_DEBUG
```

マクロの定義によりクロック初期化ルーチンがスキップされます。

（7） デバッグ時のメッセージ出力機能

DEBUG_print マクロ関数によりデバッグ実行中のログやエラーメッセージの出力が可能になります。
ただし、本サンプルでは DEBUG_print はデフォルトで無効にしています。
CS+のデバッグ・コンソール・プラグインがデフォルトで無効になっているためです。
以下の手順で DEBUG_print を有効に出来ます。

１．デバッグ・コンソール・プラグインの有効化
CS+のメニューから[ツール]-[プラグインの管理]で「プラグインの管理」ダイアログを開き、「追加機能」タブの中の「デバッグ・コンソール・プラグイン」にチェックを入れます。

２．USE_DEBUG_MESSAGE マクロを定義してビルド
コンパイル時に USE_DEBUG_MESSAGE マクロを定義するか、rtos_sample_config.h の中の以下のコメントを外してビルドしてください。

```
//#define USE_DEBUG_MESSAGE
```

（8） タスク・スタック・サイズ

本サンプルは printf 標準関数の利用を前提としています。
printf 標準関数はスタックを多く消費します。（400 バイト以上）
そのため、タスク・スタック・サイズは想定よりも多く確保してください。

（9） R_CMT_Control の制限

FIT タイマ API の R_CMT_Control 関数で、CMT チャンネルに_RI_CLOCK_TIMER マクロ（0）と同じ値を指定した場合は、エラーを返します。
RTOS 予約チャンネルを内部処理で除外しているためです。

（10） FIT API の制約

FIT では様々な API を提供していますが、ご使用に際して以下の制約があります。

表 9-5 RI600PX での r_bsp API の使用可否

r_bsp API 名	カーネル開始前	カーネル開始後	
	スタートアップ・ルーチン（PowerON_Reset_PC）	タスク（ユーザ・モード）	非タスク（スーパーバイザ・モード）
R_BSP_GetVersion	✓	✓	✓
R_BSP_InterruptsDisable	✓	(効果なし)	✓
R_BSP_InterruptsEnable	✓	(効果なし)	✓
R_BSP_CpuInterruptLevelRead	✓	✓	✓
R_BSP_CpuInterruptLevelWrite	✓	(効果なし) ***1**	(非推奨) ***1**
R_BSP_RegisterProtectEnable	✓	✓	✓
R_BSP_RegisterProtectDisable	✓	✓	✓
R_BSP_SoftwareLock	✓	✓	✓
R_BSP_SoftwareUnlock	✓	✓	✓
R_BSP_HardwareLock	✓	✓	✓
R_BSP_HardwareUnlock	✓	✓	✓
R_BSP_InterruptWrite	✓	✓	✓
R_BSP_InterruptRead	✓	✓	✓
R_BSP_InterruptControl	✓	✓	✓
R_BSP_SoftwareDelay	✓	✓	✔

*1 代わりに RI600PX のサービス・コール chg_ims 又は ichg_ims をご使用ください。

表 9-6 RI600PX での r_cmt_rx API の使用可否

r_cmt_rx API 名	カーネル開始前	カーネル開始後	
	スタートアップ・ルーチン（PowerON_Reset_PC）	タスク（ユーザ・モード）	非タスク（スーパーバイザ・モード）
R_CMT_CreatePeriodic	✓	✓ *1	✓ *1
R_CMT_CreateOneShot	✓	✓ *1	✓ *1
R_CMT_Control	✓	✓	✓
R_CMT_Stop	✓	✓	✓
R_CMT_GetVersion	✓	✔	✔

*1 RI600PX の周期ハンドラ／アラームハンドラをご使用ください。

（11） 未定義割り込みについて

本サンプルで未定義割り込みが発生した場合、カーネル内部のハンドラからシステム・ダウン・ルーチン（_RI_sys_dwn__）が呼ばれます。

r_bsp モジュールの未定義割り込みハンドラ（undefined_interrupt_source_isr）は呼び出されません。そのため、未定義割り込みのコールバック関数を登録しても、そのコールバック関数が呼び出されることはありません。

未定義割り込み発生時の処理はシステム・ダウン・ルーチンに記述してください。

（12） 基本クロック周期の設定について

RI600PX では基本クロック用タイマ割り込みの周期を 1 ミリ秒に設定することを推奨しています。

FIT 対応 RI600PX サンプル・プロジェクトでは、以下のファイルで設定した PCLKB のクロック周波数を、sample.cfg の clock.timer_clock に設定してください。

r_config¥r_bsp_config.h

これによりシステム時刻の単位が 1 ミリ秒になります。

```
// RX64M の場合の sample.cfg 設定例
clock{
    template        = rx630.tpl;
    timer           = CMT0;
    timer_clock     = 60MHz; // PCLKB の周波数
    IPL             = 13;
};
```

確認の方法は、周期ハンドラの起動周期を 125 に設定して、以下のファイルで LED_BLINK_DIV_RATIO マクロに 2 を設定します。

appli¥include¥rtos_sample_config.h

これにより、約 1 秒間隔で LED0 が点滅します。

9.2.6. FIT モジュールの追加方法

FIT 対応 RI600PX サンプルに新たな FIT モジュールを追加する手順について説明します。

（1） FIT モジュールのソースを格納した ZIP ファイルを入手します。

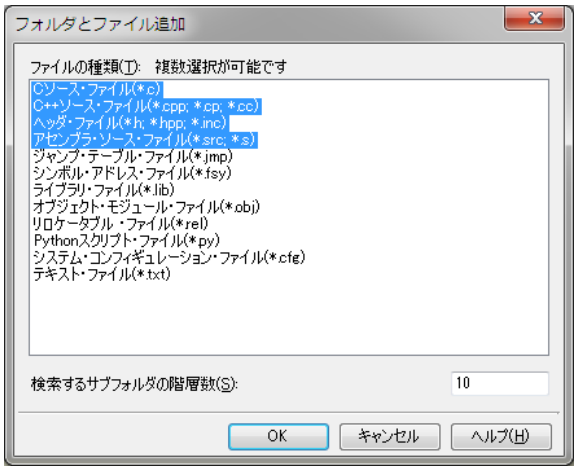
（2） CS+プロジェクトのルートディレクトリに FIT モジュールのソースを解凍します。

（3） r_config フォルダに FIT モジュールのコンフィギュレーション・ファイルを作成します。
通常リファレンス・ファイルがあるので、それをコピーしてファイル名を変更します。
コピー元： ref¥<FIT モジュール名>_config_reference.h
コピー先： r_config¥< FIT モジュール名>_config.h

（4） Windows エクスプローラで解凍した FIT モジュールのトップディレクトリを
CS+のプロジェクト・ツリーの「ファイル」へドラッグ＆ドロップします。

（5） 「フォルダとファイル追加」ダイアログで以下を設定して「OK」ボタンを押します。
・プロジェクトへ追加するファイルの種類を選択します。
・「検索するサブフォルダの階層数」を最大階層数以上（例：10）に変更します。

図 9-1 「フォルダとファイル追加」ダイアログ



（6） 「CC-RX（ビルド・ツール）」の「コンパイル・オプション」タブで、
[ソース]-[追加のインクルード・パス]を確認します。
CS+では上記の方法により、追加したディレクトリの全相対パスが「追加のインクルード・パス」へ登録されます。

（7） FIT モジュール内の割り込みハンドラをシステム・コンフィギュレーション・ファイルで登録します。
以下の手順でソースを変更してください。
1．FIT モジュール側のソースで”#pragma interrupt” の行をコメントアウトします。
※該当ソースでは platform.h または kernel_id.h のインクルードが必要です。確認してください。
2．割り込みハンドラ関数の static 宣言を削除します。
3．sample.cfg で割り込みハンドラを登録します。

以 上

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.7.31	—	新規発行
1.01	2016.5.31	34	FIT 対応サンプル・プロジェクトの説明を追加
1.02	2017.6.9	5	2.「ユーザーズマニュアルについて」で、ドキュメント番号を更新。
		13	7.3 タイマ・テンプレート・ファイルで以下の変更。 ・RX65N,RX651,RX24T,RX24U,RX230,RX62T のデバイス・グループを追加。
		34	9.2.2「FIT 対応サンプル・プログラムの構成」で以下の変更。 ・FIT 対応 RI600PX サンプル・プロジェクトに RX65N_RI600PX_FIT を追加。 ・RX65N,RX651 用 FIT モジュールのリビジョンを追加。
		35	9.2.4「FIT 対応 RI600PX サンプル・プロジェクトの変更点」で以下の変更。 ・rskrx65n,rx65n,RX65N を追加。 ・RX65N 用グループ割り込みハンドラとして group_bl2_handler_isr を追加。 ・resetprg.c に個別コンパイル・オプションを設定している理由を追加。 ・USE_SIM_DEBUG マクロの説明に次章への参照を追加。
		42	9.2.5 「FIT 対応 RI600PX サンプル・プロジェクトの注意点」で以下の変更。 ・(6)のマクロ定義先を変更。 ・(12)に基本クロック周期の設定方法を追加。
		46	9.2.6 「FIT モジュールの追加方法」で以下の変更。 ・(3)と(5)に説明を追加。 ・(7)にインクルードの注意を追加。 ・旧アプリケーションノートの参照を削除。

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/

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生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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、全部または一部を問わず、改造、改変、複製、その他の不適切に使用しないでください。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）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   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（宇宙、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に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損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。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. 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、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）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、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
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。
12. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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。

(Rev.3.0-1　2016.11)

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/