User's Manual

RX62Nグループ
Peripheral Driver Generator
リファレンスマニュアル

ルネサス エレクトロニクス
www.renesas.com

Rev.1.04　2014.5

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。

はじめに

本書は Peripheral Driver Generator 用いた RX62N グループの周辺 I/O ドライバ作成の作成方法について説明します。マイクロコントローラ機種に依存しない Peripheral Driver Generator の基本操作方法については、Peripheral Driver Generator ユーザーズマニュアルを参照してください。

目次

1. 概要

1.1 サポート範囲

Peripheral Driver Generator がサポートする RX62N グループの製品型名、周辺機能、エンディアンは以下の通りです。

(1) 製品型名

型名	パッケージ	型名	パッケージ
R5F562N8BDBG	PLBG0176GA-A	R5F562N8ADFB	PLQP0144KA-A
R5F562N8BDLE	PTLG0145JB-A	R5F562N8ADFP	PLQP0100KB-A
R5F562N8BDFB	PLQP0144KA-A	R5F562N7ADBG	PLBG0176GA-A
R5F562N8BDFP	PLQP0100KB-A	R5F562N7ADLE	PTLG0145JB-A
R5F562N7BDBG	PLBG0176GA-A	R5F562N7ADFB	PLQP0144KA-A
R5F562N7BDLE	PTLG0145JB-A	R5F562N7ADFP	PLQP0100KB-A
R5F562N7BDFB	PLQP0144KA-A	R5F56108VNFPP	LQP0144KAA
R5F562N7BDFP	PLQP0100KB-A	R5F56107VNFPP	LQP0144KAA
R5F562N8ADBG	PLBG0176GA-A	R5F56106VNFPP	LQP0144KAA
R5F562N8ADLE	PTLG0145JB-A	R5F56104VNFPP	LQP0144KAA

(2) 周辺機能

電圧検出回路 (LVD)
クロック発生回路
消費電力低減機能
割り込みコントローラ (ICUa), 例外処理
バス
DMAコントローラ (DMACA)
EXDMAコントローラ (EXDMAC)
データトランスファコントローラ (DTCa)
I/O ポート
マルチファンクションタイマパルスユニット2 (MTU2) *
ポートアウトプットイネーブル2 (POE2)
プログラマブルパルスジェネレータ (PPG)
8 ビットタイマ (TMR)
コンペアマッチタイマ (CMT)
リアルタイムクロック (RTC)
ウォッチドッグタイマ (WDT)
独立ウォッチドッグタイマ (IWDT)
シリアルコミュニケーションインタフェース(SCIa)
CRC 演算器 (CRC)
I2C バスインタフェース (RIIC)
シリアルペリフェラルインタフェース (RSPI)
12 ビットA/D コンバータ (S12AD)
10 ビットA/D コンバータ(ADa)
D/A コンバータ

*相補PWMモード、リセット同期モードは除きます。

(3) エンディアン

リトルエンディアン / ビッグエンディアン

1.2 関連ツール

本バージョンの Peripheral Driver Generator で RX210 グループを使用する際に必要な関連ツールは以下の通りです。

・RXファミリ用C/C++コンパイラパッケージ V.1.02 Release 01

・RX62Nグループ Renesas Peripheral Driver Library V.1.10 (Peripheral Driver Generatorに同梱されています。)

2. プロジェクトの作成

プロジェクトを新規に作成するにはメニューから［ファイル］->［プロジェクトの新規作成］を選択してください。[新規作成]ダイアログボックスが開きます。

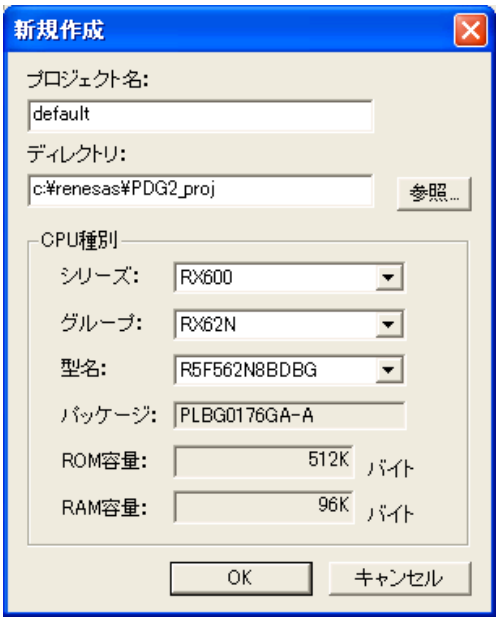


図 2.1 新規作成ダイアログボックス

RX62N グループのプロジェクトを作成するにはシリーズに［RX600］を、グループに［RX62N］を選択してください。使用する製品の型名を選択すると、その製品のパッケージ、ROM 容量、RAM 容量が表示されます。

[OK]をクリックすると新規プロジェクトを作成して開きます。

新規プロジェクトの作成直後は EXTAL 入力周波数が設定されていないためエラーが表示されます。エラーの表示についてはユーザーズマニュアルを参照してください。

図 2.2 新規プロジェクトのエラー表示

ここでは使用するクロック周波数を設定してください。

3. 周辺機能の設定

3.1 設定画面

図 3.1 に周辺モジュール設定ウィンドウの表示例を示します。

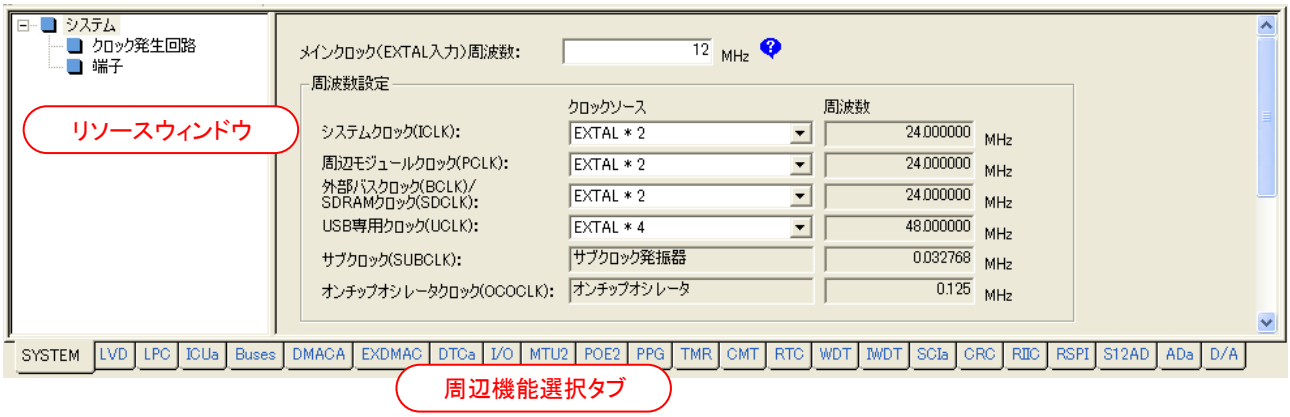


図3.1 周辺機能設定ウィンドウの表示例

周辺機能選択タブおよびリソースウィンドウに表示される項目と、周辺機能の対応を表 3.1 に示します。

表 3.1 周辺機能選択タブおよびリソースウィンドウの項目と周辺機能の対応

タブ	リソースウィンドウ	対応する周辺機能
SYSTEM	クロック発生回路	クロック発生回路
	端子	端子機能
LVD	LVD	電圧検出回路 LVD1および2
LPC	消費電力低減機能	消費電力低減機能
ICUa	割り込み	割り込みコントローラ（ICUa） （高速割り込み，ソフトウェア割り込み、外部割込み(NMI, IRQ0～15)）
	例外	例外処理
Buses	CS0 ～ CS7	CS領域（CS0～CS7）
	SDCS	SDRAM領域
	共通設定	バスエラー監視（不正アドレスアクセス検出、バスタイムアウト検出）
DMACA	DMACA0 ～ DMACA3	DMAコントローラ（DMACA）チャネル0～3
EXDMAC	EXDMAC0、1	EXDMAコントローラ(EXDMAC) チャネル0、1
DTCa	DTCa	データトランスファコントローラ（DTCa）
I/O	ポート0 ～ ポートG	I/Oポート ポート0～G
MTU2 ※	ユニット0（MTU0～MTU5）	マルチファンクションタイマパルスユニット2（MTU2）ユニット0（チャネル0～5）
	ユニット1（MTU6～MTU11）	マルチファンクションタイマパルスユニット2（MTU2）ユニット1（チャネル6～11）
POE2	POE2	ポートアウトプットイネーブル2（POE2）
PPG	Group 0 to Group 7	プログラマブルパルスジェネレータ（PPG）グループ0 ～7
TMR	ユニット0（TMR0、TMR1）	8ビットタイマ（TMR）ユニット0（チャネル0、1）
	ユニット1（TMR2、TMR3）	8ビットタイマ（TMR）ユニット1（チャネル2、3）
CMT	ユニット0（CMT0、CMT1）	コンペアマッチタイマ（CMT）ユニット0（チャネル0、1）
	ユニット1（CMT2、CMT3）	コンペアマッチタイマ（CMT）ユニット1（チャネル2、3）
RTC	RTC	リアルタイムクロック（RTC）
WDT	WDT	ウォッチドッグタイマ（WDT）

IWDT	IWDT	独立ウォッチドッグタイマ（IWDT）
SCIa	SCI0, 1, 2, 3, 5, 6	シリアルコミュニケーションインタフェース（SCIa）チャネル0～3、5～6
CRC	CRC	CRC 演算（CRC）
RIIC	RIIC0、RIIC1	I2Cバスインタフェース（RIIC）チャネル0、1
RSPI	RSPI0、RSPI1	シリアルペリフェラルインタフェース（RSPI）チャネル0、1
S12AD	S12AD0	12 ビットA/D コンバータ（S12AD）ユニット0
ADa	ADa0 ～ ADa3	10ビットA/Dコンバータ ユニット0～3
D/A	DA0、DA1	D/Aコンバータ チャネル0、1

※相補PWMモード、リセット同期モードは除きます。

周辺機能の設定手順については、ユーザーズマニュアルを参照してください。 端子機能の設定については「3.2 端子機能」を参照してください。

3.2 端子機能

周辺機能選択タブから[SYSTEM]を選択し、リソースウィンドウで[端子]を選択すると、端子機能ウィンドウが開きます。

図3.2 端子機能ウィンドウの表示方法

端子機能ウィンドウは[端子機能]シートと、[周辺機能別使用端子]シートで構成されます。

3.2.1 端子機能シート

端子機能シートではマイクロコントローラの全端子を番号順に表示します。

端子番号	端子名	選択機能	入出力	状態
A1	AVSS			
A2	AVCC			
A3	P42/IRQ10/AN2			
A4	P45/IRQ13/AN5			
A5	P46/IRQ14/AN6			
A6	P90/D16/A16			
A7	PD0/D0/POE7#			
A8	PD2/D2/MTIC11W/POE5#			
A9	PD3/D3/MTIC11V/POE4#			
A10	PD4/D4/MTIC11U/POE3#			
A11	PG0/D24			
A12	PD7/D7/MTIC5U/POE0#			
A13	P61/CS1#/SDCS#			
A14	P63/CS3#/CAS#			
A15	PE1/D9/SSLB2			

端子機能 周辺機能別使用端子

図3.3 端子機能ウィンドウ 端子機能シート

各カラムの表示内容を表 3.2 に示します。

表 3.2 端子機能シートの表示内容

カラム	内容
端子番号	端子の番号が表示されます
端子名	端子名（端子に割り当ている全機能）が表示されます
選択機能	周辺機能の設定により選択されている端子機能が表示されます
入出力	周辺機能の設定により選択されている端子の入出力方向が表示されます
状態	設定状態が表示されます

端子の入出力に関連する周辺機能を設定すると、設定の結果がウィンドウに表示されます。例えば 176 ピンのパッケージにおいて、A/D 変換器 AD0 の設定ウィンドウで、アナログ入力端子 AN0 の入力を A/D 変換するよう設定した場合、AN0 が割り当てられている C5 番の端子(P40/IRQ8/AN0)の行は、図 3.4 に示すように表示されます。

端子番号	端子名	選択機能	入出力	状態
C5	P40/IRQ8/AN0	AN0	入力	

図3.4 端子機能の表示例

この状態で I/O ポート P40 を設定すると、図 3.5 に示すように端子機能の競合が警告されます。

端子番号	端子名	選択機能	入出力	状態
C5	P40/IRQ8/AN0	AN0/P40		複数の機能で競合しています

図3.5 端子機能競合時の表示

注意

- RX62N グループでは端子ごとに割り当てる機能を指定することはできません。端子の機能は周辺機能の設定により決まります。本ウィンドウで端子機能を変更することはできません。
- 端子機能によっては割当先の端子を切り替えることができます。端子機能の割当先は周辺機能別使用端子シートで変更することができます。
- 複数の出力機能が 1 つの端子で有効に設定された場合、出力優先度の高い機能の信号が出力されます。詳細についてはハードウェアマニュアルを参照してください。

3.2.2 周辺機能別使用端子シート

周辺機能別使用端子シートでは周辺機能ごとに端子の使用状況が表示されます。左側の周辺機能一覧から選択した周辺機能の端子機能が表示されます。

図3.6 端子機能ウィンドウ 周辺機能別使用端子シート

各カラムの表示内容を表 3.3 に示します。

表 3.3 周辺機能別使用端子シートの表示内容

カラム	内容
端子名	左側の周辺機能一覧で選択した周辺機能の端子機能名が表示されます
選択機能	選択されている端子機能の内容が表示されます
使用端子	割り当て先の端子名（端子に割り当ている全機能）が表示されます
使用端子番号	割り当て先の端子番号が表示されます
入出力	端子の入出力状態が表示されます
状態	設定状態が表示されます

端子の入出力に関連する周辺機能を設定すると、設定の結果がウィンドウに表示されます。例えば周辺機能の設定で外部割込み IRQ9 を設定すると、IRQ9 端子は、図 3.7 に示すように表示されます。

端子名	端子機能	使用端子	使用端子番号	入出力	状態
IRQ9	外部割込み入力	P01/TMCI0/RxD6/IRQ9	D2	入力	

図3.7 使用端子の表示例

この状態で同じ端子を使用する I/O ポート P01 を設定すると、図 3.8 に示すように端子機能の競合が警告されます。

端子名	端子機能	使用端子	使用端子番号	入出力	状態
IRQ9	外部割込み入力	P01/TMCI0/RxD6/IRQ9	D2	入力	他の機能と競合しています

IRQ9 は割当先を変更すことができます。割当先を変更できる端子機能は、使用端子のセルにマウスポインタを置くと、割当先端子の選択肢を開くためのドロップダウンボタンが表示されます。

図3.9 ドロップダウンボタンの表示

端子機能の割当先を変更すには、ドロップダウンボタンをクリックし、表示された選択肢から割当先の端子を指定してください。

図3.10 端子機能の割当先変更

IRQ9 の割当先を P41/IRQ9/AN1 に変更し、変更後の割当先端子が他の機能で使用されていなければ、競合状態を解決することができます。

端子名	端子機能	使用端子	使用端子番号	入出力	状態
IRQ9	外部割込み入力	P41/IRQ9/AN1	D4	入力	

図3.11 端子機能の割当先変更後の表示

割当先を変更できる端子機能を「付録 1 割当先を変更できる端子機能」に示します。

4. 生成関数仕様

RX62N の生成関数を表 4.1 に示します。

表 4.1 RX62N の生成関数

クロック発生回路

生成関数	機能
R_PG_Clock_Set	クロックの設定
R_PG_Clock_Start_SUB	サブクロック発振器開始
R_PG_Clock_Stop_SUB	サブクロック発振器停止
R_PG_Clock_GetMainClockStatus	メインクロックの状態を取得

電圧検出回路(LVD)

生成関数	機能
R_PG_LVD_Set	電圧検出回路の設定
R_PG_LVD_GetLVDDetectionFlag	LVD検知フラグの取得

消費電力低減機能

生成関数	機能
R_PG_LPC_Set	消費電力低減機能の設定
R_PG_LPC_Sleep	スリープモードへ移行
R_PG_LPC_AllModuleClockStop	全モジュールクロックスタンバイモードへ移行
R_PG_LPC_SoftwareStandby	ソフトウェアスタンバイモードへ移行
R_PG_LPC_DeepSoftwareStandby	ディープソフトウェアスタンバイモードへ移行
R_PG_LPC_IOPortRelease	I/Oポート出力保持を解除
R_PG_LPC_GetPowerOnResetFlag	パワーオンリセットフラグの取得
R_PG_LPC_GetLVDDetectionFlag	LVD検知フラグの取得
R_PG_LPC_GetDeepSoftwareStandbyResetFlag	ディープソフトウェアスタンバイリセットフラグの取得
R_PD_LPC_GetDeepSoftwareStandbyCancelFlag	ディープソフトウェアスタンバイ解除要求フラグの取得
R_PG_LPC_GetStatus	消費電力低減機能の状態を取得
R_PG_LPC_WriteBackup	バックアップレジスタへの書き込み
R_PG_LPC_ReadBackup	バックアップレジスタからの読み出し

割り込みコントローラ (ICU)

生成関数	機能
R_PG_ExtInterrupt_Set_*<割り込み種別>*	外部割り込みの設定
R_PG_ExtInterrupt_Disable_*<割り込み種別>*	外部割り込みの設定解除
R_PG_ExtInterrupt_GetRequestFlag_*<割り込み種別>*	外部割り込み要求フラグの取得
R_PG_ExtInterrupt_ClearRequestFlag_*<割り込み種別>*	外部割り込み要求フラグのクリア
R_PG_SoftwareInterrupt_Set	ソフトウェア割り込みの設定
R_PG_SoftwareInterrupt_Generate	ソフトウェア割り込みの生成
R_PG_FastInterrupt_Set	高速割り込みの設定
R_PG_Exception_Set	例外ハンドラの設定

バス

生成関数	機能
R_PG_ExtBus_SetBus	バス端子とバスエラー監視の設定
R_PG_ExtBus_GetErrorStatus	バスエラー検出状態の取得

R_PG_ExtBus_ClearErrorFlags	バスエラーステータスレジスタのクリア
R_PG_ExtBus_SetArea_CS<領域番号>	CS領域の設定
R_PG_ExtBus_DisableArea_CS<領域番号>	CS領域の設定解除
R_PG_ExtBus_SetArea_SDCS	SDRAM領域の設定
R_PG_ExtBus_Initialize_SDCS	SDRAM初期化シーケンスの開始
R_PG_ExtBus_AutoRefreshEnable_SDCS	SDRAMオートリフレッシュの有効化
R_PG_ExtBus_AutoRefreshDisable_SDCS	SDRAMオートリフレッシュの無効化
R_PG_ExtBus_SelfRefreshEnable_SDCS	SDRAMセルフリフレッシュの有効化
R_PG_ExtBus_SelfRefreshDisable_SDCS	SDRAMセルフリフレッシュの無効化
R_PG_ExtBus_AccessEnable_SDCS	SDRAMアクセスの有効化
R_PG_ExtBus_AccessDisable_SDCS	SDRAMアクセスの無効化
R_PG_ExtBus_GetStatus_SDCS	SDRAMステータスの取得

DMAコントローラ（DMAC）

生成関数	機能
R_PG_DMAC_Set_C<*チャネル番号*>	DMACの設定
R_PG_DMAC_Activate_C<*チャネル番号*>	DMACをデータ転送開始トリガの入力待ち状態に設定
R_PG_DMAC_StartTransfer_C<*チャネル番号*>	データ転送の開始(ソフトウェアトリガ)
R_PG_DMAC_Suspend_C<*チャネル番号*>	データ転送の中断
R_PG_DMAC_GetTransferCount_C<*チャネル番号*>	転送カウンタ値の取得
R_PG_DMAC_SetTransferCount_C<*チャネル番号*>	転送カウンタ値の設定
R_PG_DMAC_GetRepeatBlockSizeCount_C<*チャネル番号*>	リピート/ブロックサイズカウンタ値の取得
R_PG_DMAC_SetRepeatBlockSizeCount_C<*チャネル番号*>	リピート/ブロックサイズカウンタの設定
R_PG_DMAC_ClearInterruptFlag_C<*チャネル番号*>	割り込み要求フラグの取得とクリア
R_PG_DMAC_GetTransferEndFlag_C<*チャネル番号*>	転送終了フラグの取得
R_PG_DMAC_ClearTransferEndFlag_C<*チャネル番号*>	転送終了フラグのクリア
R_PG_DMAC_GetTransferEscapeEndFlag_C<*チャネル番号*>	エスケープ転送終了フラグの取得
R_PG_DMAC_ClearTransferEscapeEndFlag_C<*チャネル番号*>	エスケープ転送終了フラグのクリア
R_PG_DMAC_SetSrcAddress_C<*チャネル番号*>	データ転送元アドレスの設定
R_PG_DMAC_SetDestAddress_C<*チャネル番号*>	データ転送先アドレスの設定
R_PG_DMAC_SetAddressOffset_C<*チャネル番号*>	アドレスオフセット値の設定
R_PG_DMAC_SetExtendedRepeatSrc_C<*チャネル番号*>	転送元拡張リピートエリアの設定
R_PG_DMAC_SetExtendedRepeatDest_C<*チャネル番号*>	転送先拡張リピートエリアの設定
R_PG_DMAC_StopModule	DMAC全チャネルの停止

EXDMAコントローラ（EXDMAC）

生成関数	機能
R_PG_EXDMAC_Set_C<*チャネル番号*>	EXDMACの設定
R_PG_EXDMAC_Activate_C<*チャネル番号*>	EXDMACを転送開始トリガの入力待ち状態に設定
R_PG_EXDMAC_StartTransfer_C<*チャネル番号*>	データ転送の開始(ソフトウェアトリガ)
R_PG_EXDMAC_Suspend_C<*チャネル番号*>	データ転送の中断
R_PG_EXDMAC_GetTransferCount_C<*チャネル番号*>	転送カウンタ値の取得
R_PG_EXDMAC_SetTransferCount_C<*チャネル番号*>	転送カウンタ値の設定
R_PG_EXDMAC_GetRepeatBlockSizeCount_C<*チャネル番号*>	リピート/ブロック/クラスタサイズカウンタ値の取得
R_PG_EXDMAC_SetRepeatBlockSizeCount_C<*チャネル番号*>	リピート/ブロック/クラスタサイズカウンタ値の設定
R_PG_EXDMAC_ClearInterruptFlag_C<*チャネル番号*>	割り込み要求フラグの取得とクリア

R_PG_EXDMAC_GetTransferEndFlag_C〈*チャネル番号*〉	転送終了フラグの取得
R_PG_EXDMAC_ClearTransferEndFlag_C〈*チャネル番号*〉	転送終了フラグのクリア
R_PG_EXDMAC_GetTransferEscapeEndFlag_C〈*チャネル番号*〉	転送エスケープ終了フラグの取得
R_PG_EXDMAC_ClearTransferEscapeEndFlag_C〈*チャネル番号*〉	転送エスケープ終了フラグのクリア
R_PG_EXDMAC_SetSrcAddress_C〈*チャネル番号*〉	転送元アドレスの設定
R_PG_EXDMAC_SetDestAddress_C〈*チャネル番号*〉	転送先アドレスの設定
R_PG_EXDMAC_SetAddressOffset_C〈*チャネル番号*〉	アドレスオフセット値の設定
R_PG_EXDMAC_SetExtendedRepeatSrc_C〈*チャネル番号*〉	転送元拡張リピートエリアの設定
R_PG_EXDMAC_SetExtendedRepeatDest_C〈*チャネル番号*〉	転送先拡張リピートエリアの設定
R_PG_EXDMAC_StopModule_C〈*チャネル番号*〉	EXDMACチャネルの停止

データトランスファコントローラ (DTCa)

生成関数	機能
R_PG_DTC_Set	DTCの設定
R_PG_DTC_Set_〈転送開始トリガ〉	DTC転送情報の設定
R_PG_DTC_Activate	DTCをトリガ入力待ち状態にする
R_PG_DTC_SuspendTransfer	DTC転送の停止
R_PG_DTC_GetTransmitStatus	DTC転送状態の取得
R_PG_DTC_StopModule	DTCの停止

I/Oポート

生成関数	機能
R_PG_IO_PORT_Set_P〈*ポート番号*〉	I/Oポートの設定
R_PG_IO_PORT_Set_P〈*ポート番号*〉〈*端子番号*〉	I/Oポート(1端子)の設定
R_PG_IO_PORT_Read_P〈*ポート番号*〉	I/Oポートレジスタからの読み出し
R_PG_IO_PORT_Read_P〈*ポート番号*〉〈*端子番号*〉	I/Oポートレジスタからのビット読み出し
R_PG_IO_PORT_Write_P〈*ポート番号*〉	I/Oポートデータレジスタへの書き込み
R_PG_IO_PORT_Write_P〈*ポート番号*〉〈*端子番号*〉	I/Oポートデータレジスタへのビット書き込み

マルチファンクションタイマパルスユニット2 (MTU2)

生成関数	機能
R_PG_Timer_Set_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	MTUの設定
R_PG_Timer_StartCount_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	カウント動作の開始
R_PG_Timer_StartCount_MTU_U〈*ユニット番号*〉C〈*チャネル番号*〉_〈*相*〉	U、V、W相のカウント動作の開始
R_PG_Timer_HaltCount_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	カウント動作の停止
R_PG_Timer_HaltCount_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉_〈*相*〉	U、V、W相のカウント動作の停止
R_PG_Timer_GetCounterValue_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	カウンタ値の取得
R_PG_Timer_SetCounterValue_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	カウンタ値の設定
R_PG_Timer_GetRequestFlag_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	割り込み要求フラグの取得とクリア
R_PG_Timer_StopModule_MTU_U〈*ユニット番号*〉	MTUのユニットを停止
R_PG_Timer_GetTGR_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	ジェネラルレジスタ値の取得
R_PG_Timer_SetTGR_〈*レジスタ*〉_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	ジェネラルレジスタ値の設定
R_PG_Timer_SetBuffer_AD_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉	A/D変換要求周期設定バッファレジスタの設定

ポートアウトプットイネーブル2 (POE2)

生成関数	機能
R_PG_POE_Set	POEの設定
R_PG_POE_SetHiZ_MTU〈*MTUチャネル番号*〉	MTU端子をハイインピーダンスに設定

R_PG_POE_GetRequestFlagHiZ_MTU*<MTUチャネル番号>*	ハイインピーダンス要求フラグの取得
R_PG_POE_GetShortFlag_MTU*<MTUチャネル番号1>*_*<MTUチャネル番号2>*	MTU端子の出力短絡フラグの取得
R_PG_POE_ClearFlag_MTU*<MTUチャネル番号>*	ハイインピーダンス要求/出力短絡フラグのクリア

プログラマブルパルスジェネレータ（PPG）

生成関数	機能
R_PG_PPG_StartOutput_U*<ユニット番号>*_G*<グループ番号>*	PPGの設定とパルス出力の開始
R_PG_PPG_StopOutput_U*<ユニット番号>*_G*<グループ番号>*	パルス出力の停止
R_PG_PPG_SetOutputValue_U*<ユニット番号>*_G*<グループ番号>*	1グループ(4bit)の出力値の設定
R_PG_PPG_SetOutputValue_U*<ユニット番号>*_G*<グループ番号1>*_G*<グループ番号2>*	2グループ(8bit)の出力値の設定

8ビットタイマ（TMR）

生成関数	機能
R_PG_Timer_Start_TMR_U*<ユニット番号>*(_C*<チャネル番号>*)	TMRを設定しカウント動作を開始
R_PG_Timer_HaltCount_TMR_U*<ユニット番号>*(_C*<チャネル番号>*)	TMRのカウント動作を一時停止
R_PG_Timer_ResumeCount_TMR_U*<ユニット番号>*(_C*<チャネル番号>*)	TMRのカウント動作を再開
R_PG_Timer_GetCounterValue_TMR_U*<ユニット番号>*(_C*<チャネル番号>*)	TMRのカウンタ値を取得
R_PG_Timer_SetCounterValue_TMR_U*<ユニット番号>*(_C*<チャネル番号>*)	TMRのカウンタ値を設定
R_PG_Timer_GetRequestFlag_TMR_U*<ユニット番号>*(_C*<チャネル番号>*)	TMRの割り込み要求フラグの取得とクリア
R_PG_Timer_StopModule _TMR_U*<ユニット番号>*	TMRのユニットを停止

コンペアマッチタイマ（CMT）

生成関数	機能
R_PG_Timer_Start_TMR_U*<ユニット番号>*_C*<チャネル番号>*	CMTを設定しカウント動作を開始
R_PG_Timer_HaltCount_CMT_U*<ユニット番号>*_C*<チャネル番号>*	CMTのカウント動作を一時停止
R_PG_Timer_ResumeCount_CMT_U*<ユニット番号>*(_C*<チャネル番号>*	CMTのカウント動作を再開
R_PG_Timer_GetCounterValue_CMT_U*<ユニット番号>*(_C*<チャネル番号>*	CMTのカウンタ値を取得
R_PG_Timer_SetCounterValue_CMT_U*<ユニット番号>*(_C*<チャネル番号>*	CMTのカウンタ値を設定
R_PG_Timer_StopModule _CMT_U*<ユニット番号>*	CMTのユニットを停止

リアルタイムクロック（RTC）

生成関数	機能
R_PG_RTC_Set	RTCを設定しカウント動作を開始
R_PG_RTC_Stop	RTCのカウント動作を一時停止
R_PG_RTC_Restart	RTCのカウント動作を再開
R_PG_RTC_SetCurrentTime	現在時刻の設定
R_PG_RTC_SetAlarmTime	アラーム時刻の設定
R_PG_RTC_Alarm_Control	アラームの有効化/無効化
R_PG_RTC_Adjust30sec	30秒調整の実行
R_PG_RTC_SetPeriodicInterrupt	周期割り込みの周期設定
R_PG_RTC_GetStatus	RTCの状態を取得
R_PG_RTC_ClockOut_Disable	クロック出力の無効化
R_PG_RTC_ClockOut_Enable	クロック出力の有効化

ウォッチドッグタイマ（WDT）

生成関数	機能
R_PG_Timer_Start_WDT	WDTを設定しカウント動作を開始
R_PG_Timer_HaltCount_WDT	カウント動作の停止

R_PG_Timer_ResetCounter_WDT	カウンタのリセット
R_PG_Timer_ClearOverflowFlag_WDT	オーバフローフラグの取得とクリア

独立ウォッチドッグタイマ（IWDT）

生成関数	機能
R_PG_Timer_Set_IWDT	IWDTの設定
R_PG_Timer_RefreshCounter_IWDT	カウンタのリフレッシュ
R_PG_Timer_GetCounterValue_IWDT	カウンタ値の取得
R_PG_Timer_ClearUnderflowFlag_IWDT	アンダフローフラグの取得とクリア

シリアルコミュニケーションインタフェース（SCIa）

生成関数	機能
R_PG_SCI_Set_C〈*チャネル番号*〉	シリアルI/Oチャネルの設定
R_PG_SCI_StartSending_C〈*チャネル番号*〉	シリアルデータの送信開始
R_PG_SCI_SendAllData_C〈*チャネル番号*〉	シリアルデータを全て送信
R_PG_SCI_GetSentDataCount_C〈*チャネル番号*〉	シリアルデータの送信数取得
R_PG_SCI_StartReceiving_C〈*チャネル番号*〉	シリアルデータの受信開始
R_PG_SCI_ReceiveAllData_C〈*チャネル番号*〉	シリアルデータを全て受信
R_PG_SCI_StopCommunication_C〈*チャネル番号*〉	シリアルデータの送受信停止
R_PG_SCI_GetReceivedDataCount_C〈*チャネル番号*〉	シリアルデータの受信数取得
R_PG_SCI_GetReceptionErrorFlag_C〈*チャネル番号*〉	シリアル受信エラーフラグの取得
R_PG_SCI_GetTransmitStatus_C〈*チャネル番号*〉	シリアルデータ送信状態の取得
R_PG_SCI_SendTargetStationID_C〈*チャネル番号*〉	データ送信先IDの送信
R_PG_SCI_ReceiveStationID_C〈*チャネル番号*〉	自局IDと一致するIDコードの受信
R_PG_SCI_StopModule_C〈*チャネル番号*〉	シリアルI/Oチャネルの停止

CRC演算器（CRC）

生成関数	機能
R_PG_CRC_Set	CRC演算器の設定
R_PG_CRC_InputData	CRC演算器にデータを入力
R_PG_CRC_GetResult	演算結果の取得
R_PG_CRC_StopModule	CRC演算器の停止

I2Cバスインタフェース（RIIC）

生成関数	機能
R_PG_I2C_Set_C〈*チャネル番号*〉	I2Cバスインタフェースチャネルの設定
R_PG_I2C_MasterReceive_C〈*チャネル番号*〉	マスタのデータ受信
R_PG_I2C_MasterReceiveLast_C〈*チャネル番号*〉	マスタのデータ受信終了
R_PG_I2C_MasterSend_C〈*チャネル番号*〉	マスタのデータ送信
R_PG_I2C_MasterSendWithoutStop_C〈*チャネル番号*〉	マスタのデータ送信(STOP条件無し)
R_PG_I2C_GenerateStopCondition_C〈*チャネル番号*〉	マスタのSTOP条件生成
R_PG_I2C_GetBusState_C〈*チャネル番号*〉	バス状態の取得
R_PG_I2C_SlaveMonitor_C〈*チャネル番号*〉	スレーブのバス監視
R_PG_I2C_SlaveSend_C〈*チャネル番号*〉	スレーブのデータ送信
R_PG_I2C_GetDetectedAddress_C〈*チャネル番号*〉	検出したスレーブアドレスの取得
R_PG_I2C_GetTR_C〈*チャネル番号*〉	送信/受信モードの取得
R_PG_I2C_GetEvent_C〈*チャネル番号*〉	検出イベントの取得
R_PG_I2C_GetReceivedDataCount_C〈*チャネル番号*〉	受信済みデータ数の取得

R_PG_I2C_GetSentDataCount_C〈*チャネル番号*〉	送信済みデータ数の取得
R_PG_I2C_Reset_C〈*チャネル番号*〉	バスのリセット
R_PG_I2C_StopModule_C〈*チャネル番号*〉	I2Cバスインタフェースチャネルの停止

シリアルペリフェラルインタフェース（RSPI）

生成関数	機能
R_PG_RSPI_Set_C〈*チャネル番号*〉	RSPIチャネルの設定
R_PG_RSPI_SetCommand_C〈*チャネル番号*〉	コマンドの設定
R_PG_RSPI_StartTransfer_C〈*チャネル番号*〉	データの転送開始
R_PG_RSPI_TransferAllData_C〈*チャネル番号*〉	全データの転送
R_PG_RSPI_GetStatus_C〈*チャネル番号*〉	転送状態の取得
R_PG_RSPI_GetError_C〈*チャネル番号*〉	エラー検出状態の取得
R_PG_RSPI_GetCommandStatus_C〈*チャネル番号*〉	コマンドステータスの取得
R_PG_RSPI_StopModule_C〈*チャネル番号*〉	RSPIチャネルの停止
R_PG_RSPI_LoopBack〈*ループバックモード*〉_C〈*チャネル番号*〉	ループバックモードの設定

12ビットA/Dコンバータ（S12AD）

生成関数	機能
R_PG_ADC_12_Set_S12AD0	12ビットA/Dコンバータの設定
R_PG_ADC_12_StartConversionSW_S12AD0	A/D変換の開始（ソフトウェアトリガ）
R_PG_ADC_12_StopConversion_S12AD0	A/D変換の中断
R_PG_ADC_12_GetResult_S12AD0	A/D変換結果の取得
R_PG_ADC_12_StopModule_S12AD0	12ビットA/Dコンバータの停止

10ビットA/Dコンバータ

生成関数	機能
R_PG_ADC_10_Set_AD〈*ユニット番号*〉	10ビットA/Dコンバータの設定
R_PG_ADC_10_StartConversionSW_AD〈*ユニット番号*〉	A/D変換の開始(ソフトウェアトリガ)
R_PG_ADC_10_StopConversion_AD〈*ユニット番号*〉	A/D変換の中断
R_PG_ADC_10_GetResult_AD〈*ユニット番号*〉	A/D変換結果の取得
R_PG_ADC_10_SetSelfDiag_VREF_〈*電圧値*〉_AD〈*ユニット番号*〉	A/D自己診断機能の設定
R_PG_ADC_10_StartSelfDiag_AD〈*ユニット番号*〉	A/D変換の開始（自己診断機能）
R_PG_ADC_10_StopModule_AD〈*ユニット番号*〉	10ビットA/Dコンバータの停止

D/A コンバータ

生成関数	機能
R_PG_DAC_Set_C〈*チャネル番号*〉	D/Aコンバータチャネルの設定
R_PG_DAC_SetWithInitialValue_C〈*チャネル番号*〉	D/Aコンバータチャネルの設定(初期値指定)
R_PG_DAC_ControlOutput_C〈*チャネル番号*〉	D/A変換値の設定
R_PG_DAC_StopOutput_C〈*チャネル番号*〉	アナログ出力の停止
R_PG_DAC_Set_C0_C1	D/Aコンバータのチャネルを設定(DA0, DA1一括設定)

4.1 クロック発生回路

4.1.1 R_PG_Clock_Set

定義 bool R_PG_Clock_Set(void)

概要 クロックの設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_Clock.c

使用RPDL関数 R_CGC_Set

詳細

- クロック発生回路のレジスタを設定し、EXTALに対するシステムクロック(ICLK)、周辺モジュールクロック(PCLK)、外部バスクロック(BCLK)、SDRAMクロック(SDCLK)の逓倍率を設定します。
- メインクロック発信停止検出機能を設定します。
- 外部バスクロック(BCLK)とSDRAMクロック(SDCLK)端子出力を設定します。
- 本関数を呼び出すとサブクロック発振器は停止します。サブクロックの発振/停止は R_PG_Clock_Start_SUB、R_PG_Clock_Stop_SUB を使用してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //クロック発生回路を設定する
    R_PG_Clock_Set();
}
```

4.1.2 R_PG_Clock_Start_SUB

定義 bool R_PG_Clock_Start_SUB(void)

概要 サブクロックの発振開始

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_Clock.c

使用RPDL関数 R_CGC_Control

詳細

- サブクロックの発振を開始します。
- R_PG_Clock_Setを呼び出すとサブクロック発信器は停止します。R_PG_Clock_Setの呼出し後にサブクロックの発振を開始するには本関数を呼び出してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //クロック発生回路を設定する
    R_PG_Clock_Set();

    //サブクロックの発振開始
    R_PG_Clock_Start_SUB();
}
```

4.1.3 R_PG_Clock_Stop_SUB

定義 bool R_PG_Clock_Stop_SUB(void)

概要 サブクロックの発振停止

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_Clock.c

使用RPDL関数 R_CGC_Control

詳細

- サブクロックの発振を停止します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    //クロック発生回路を設定する
    R_PG_Clock_Set();

    //サブクロックの発振開始
    R_PG_Clock_Start_SUB();
}

void func2(void)
{
    //サブクロックの発振停止
    R_PG_Clock_Stop_SUB();
}
```

4.1.4 R_PG_Clock_GetMainClockStatus

定義 bool R_PG_Clock_GetMainClockStatus (bool * stop)

概要 メインクロック発振停止検出フラグの取得

生成条件 メインクロック発振停止検出機能が有効

引数

bool * stop	メインクロック発振停止検出フラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル R_PG_Clock.c

使用RPDL関数 R_CGC_GetStatus

詳細

- メインクロック発振停止検出フラグを取得します
- メインクロック発振停止検出時にNMIを発生させるには、GUI上のNMI設定で発振停止検出割り込みを有効にしてください。NMIはR_PG_ExtInterrupt_Set_NMIにより設定できます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool main_stop;

void func(void)
{
    // メインクロック発振停止検出フラグの取得
    R_PG_Clock_GetMainClockStatus(&main_stop);
}
```

4.2 電圧検出回路（LVD）

4.2.1 R_PG_LVD_Set

定義　bool R_PG_LVD_Set (void)

概要　電圧検出回路の設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_LVD.c

使用RPDL関数　R_LVD_Control

詳細

- 低電圧検出時の動作(内部リセットまたは割り込み)を設定します。
- 1回の呼び出しでLVD1とLVD2を設定することができます。
- 低電圧検出時の動作に割り込みを選択した場合はNMIを設定する必要があります。低電圧検出時にNMIを発生させるには、GUI上のNMI設定で電源電圧降下検出割り込みを有効にしてください。NMIはR_PG_ExtInterrupt_Set_NMIにより設定できます。
- 低電圧検出フラグ(LVD1およびLVD2)はR_PG_LVD_GetLVDDetectionFlagにより取得できます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //電圧検出回路の設定
    R_PG_LVD_Set (void);
}
```

4.2.2 R_PG_LVD_GetLVDDetectionFlag

定義 bool R_PG_LVD_GetLVDDetectionFlag (bool * lvd1, bool * lvd2)

概要 LVD検知フラグの取得

引数

bool * lvd1	LVD1検知フラグの格納先
bool * lvd2	LVD2検知フラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル R_PG_LVD.c

使用RPDL関数 R_LPC_GetStatus

詳細

- LVD検知フラグを取得します。
- 取得しないフラグには0を指定してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool lvd1;
bool lvd2;

void func(void)
{
    //LVD1、LVD2検知フラグの取得
    R_PG_LVD_GetLVDDetectionFlag ( &lvd1, &lvd2);

    if( lvd1 ){
        //LVD1検出時処理
    }
    if( lvd2 ){
        //LVD2検出時処理
    }
}
```

4.3 消費電力低減機能

4.3.1 R_PG_LPC_Set

定義 bool R_PG_LPC_Set (void)

概要 消費電力低減機能の設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_Create

詳細

- 消費電力低減機能を設定します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //消費電力低減機能の設定
    R_PG_LPC_Set (void);
}
```

4.3.2 R_PG_LPC_Sleep

定義 bool R_PG_LPC_Sleep (void)

概要 スリープモードへの移行

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_Control

詳細

- スリープモードへ移行します

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //スリープモードへの移行
    R_PG_LPC_Sleep();
}
```

4.3.3 R_PG_LPC_AllModuleClockStop

定義 bool R_PG_LPC_AllModuleClockStop (void)

概要 全モジュールクロックスタンバイモードへの移行

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_Control

詳細

- 全モジュールクロックスタンバイモードへ移行します。
- 全モジュールクロックスタンバイモードへ移行する前に、全モジュールクロックスタンバイモード中の動作を許可するTMRユニットを設定します。
- 初期設定では全モジュールクロックスタンバイモード中にTMRは停止します。全モジュールクロックスタンバイモード中にTMRを動作させるには、動作させるTMRのユニットをGUI上で選択してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //全モジュールクロックスタンバイモードへの移行
    R_PG_LPC_AllModuleClockStop();
}
```

4.3.4 R_PG_LPC_SoftwareStandby

定義 bool R_PG_LPC_SoftwareStandby(void)

概要 ソフトウェアスタンバイモードへの移行

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_Control

詳細

- ソフトウェアスタンバイモードへ移行します。
- 本関数を呼ぶ前にR_PG_LPC_Setを呼び出して、ソフトウェアスタンバイモード中の動作を設定してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //消費電力低減機能の設定
    R_PG_LPC_Set();

    //ソフトウェアスタンバイモードへの移行
    R_PG_LPC_SoftwareStandby();
}
```

4.3.5 R_PG_LPC_DeepSoftwareStandby

定義 bool R_PG_LPC_DeepSoftwareStandby(void)

概要 ディープソフトウェアスタンバイモードへの移行

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_Control

詳細

- ディープソフトウェアスタンバイモードへ移行します。
- 本関数を呼ぶ前にR_PG_LPC_Setを呼び出して、ディープソフトウェアスタンバイモード中の動作と解除要因を設定してください。
- ディープソフトウェア解除フラグはディープソフトウェアモード以外の場合も解除要求が発生すると”1”になります。本関数ではディープソフトウェアスタンバイモードに移行する前にディープソフトウェア解除フラグはクリアされません。
  R_PD_LPC_GetDeepSoftwareStandbyCancelFlagによりディープソフトウェア解除フラグをクリアしてからディープソフトウェアスタンバイモードに移行してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //消費電力低減機能の設定
    R_PG_LPC_Set();

    //ディープソフトウェアスタンバイ解除フラグのクリア
    R_PD_LPC_GetDeepSoftwareStandbyCancelFlag(0,0,0,0,0,0,0,0);

    //ディープソフトウェアスタンバイモードへの移行
    R_PG_LPC_DeepSoftwareStandby();
}
```

4.3.6 R_PG_LPC_IOPortRelease

定義 bool R_PG_LPC_IOPortRelease (void)

概要 I/Oポート出力保持を解除

生成条件 GUI上で[I/Oポート状態保持]に［ディープソフトウェアスタンバイ解除後のIOKEEPビットへの"0"書き込みで保持を解除]を選択した場合に出力されます。

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_Control

詳細

- ディープソフトウェアスタンバイ解除後のI/Oポートの出力保持状態を解除します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //I/Oポートの出力保持状態を解除
    R_PG_LPC_IOPortRelease();
}
```

4.3.7 R_PG_LPC_GetPowerOnResetFlag

定義 bool R_PG_LPC_GetPowerOnResetFlag (bool * reset)

概要 パワーオンリセットフラグの取得

引数

bool * reset	パワーオンリセットフラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_GetStatus

詳細

- パワーオンリセットフラグを取得します
- 本関数を呼び出すとRSTSR.LVD1F(LVD1検知フラグ)、RSTSR.LVD2F(LVD2検知フラグ)、RSTSR.DPSRSTF(ディープソフトウェアスタンバイリセットフラグ)およびDPSIFR(ディープソフトウェアスタンバイ解除要求フラグ)がクリアされます。これらのフラグを同時に取得する必要がある場合は本関数の代わりにR_PG_LPC_GetStatusを使用してください。
- RSTSR.PORF(パワーオンリセットフラグ)は端子リセットでのみクリアされます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool reset;

void func(void)
{
	//パワーオンリセットフラグの取得
	R_PG_LPC_GetPowerOnResetFlag( &reset );

	if( reset ){
		//パワーオンリセット検出時処理
	}
}
```

4.3.8 R_PG_LPC_GetLVDDetectionFlag

定義 bool R_PG_LPC_GetLVDDetectionFlag (bool * lvd1, bool * lvd2)

概要 LVD検知フラグの取得

引数

bool * lvd1	LVD1検知フラグの格納先
bool * lvd2	LVD2検知フラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_GetStatus

詳細

- LVD検知フラグを取得します。
- 取得するフラグに対応する引数に、フラグ値の格納先アドレスを指定してください。
- 取得しないフラグには0を指定してください。
- 本関数を呼び出すとRSTSR.LVD1F(LVD1検知フラグ)、RSTSR.LVD2F(LVD2検知フラグ)、RSTSR.DPSRSTF(ディープソフトウェアスタンバイリセットフラグ)およびDPSIFR(ディープソフトウェアスタンバイ解除要求フラグ)がクリアされます。これらのフラグを同時に取得する必要がある場合は本関数の代わりにR_PG_LPC_GetStatusを使用してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool lvd1;
bool lvd2;

void func(void)
{
    //LVD1、LVD2検出フラグの取得
    R_PG_LPC_GetLVDDetectionFlag ( &lvd1, &lvd2);

    if( lvd1 ){
        //LVD1検出時処理
    }
    if( lvd2 ){
        //LVD2検出時処理
    }
}
```

4.3.9 R_PG_LPC_GetDeepSoftwareStandbyResetFlag

定義 bool R_PG_LPC_GetDeepSoftwareStandbyResetFlag(bool *reset)

概要 ディープソフトウェアスタンバイリセットフラグの取得

引数

bool *reset	ディープソフトウェアスタンバイリセットフラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル R_PG_LPC.c

使用RPDL関数 R_LPC_GetStatus

詳細

- ディープソフトウェアスタンバイリセットフラグを取得します
- 本関数を呼び出すとRSTSR.LVD1F(LVD1検知フラグ)、RSTSR.LVD2F(LVD2検知フラグ)、RSTSR.DPSRSTF(ディープソフトウェアスタンバイリセットフラグ)およびDPSIFR(ディープソフトウェアスタンバイ解除要求フラグ)がクリアされます。これらのフラグを同時に取得する必要がある場合は本関数の代わりにR_PG_LPC_GetStatusを使用してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool reset;

void func(void)
{
    //ディープソフトウェアスタンバイリセットフラグの取得
    R_PG_LPC_GetDeepSoftwareStandbyResetFlag ( &reset);

    if( reset ){
        //ディープソフトウェアスタンバイリセット検出時の処理
    }
}
```

4.3.10 R_PD_LPC_GetDeepSoftwareStandbyCancelFlag

定義

bool R_PD_LPC_GetDeepSoftwareStandbyCancelFlag
(bool *irq0, bool *irq1, bool *irq2, bool *irq3, bool *lvd, bool *rtc, bool *usb, bool *nmi)

概要

ディープソフトウェアスタンバイ解除要求フラグの取得

引数

bool *irq0	IRQ0ディープソフトウェアスタンバイ解除要求フラグの格納先
bool *irq1	IRQ1ディープソフトウェアスタンバイ解除要求フラグの格納先
bool *irq2	IRQ2ディープソフトウェアスタンバイ解除要求フラグの格納先
bool *irq3	IRQ3ディープソフトウェアスタンバイ解除要求フラグの格納先
bool *lvd	LVDディープソフトウェアスタンバイ解除要求フラグの格納先
bool *rtc	RTCディープソフトウェアスタンバイ解除要求フラグの格納先
bool *usb	USBディープソフトウェアスタンバイ解除要求フラグの格納先
bool *nmi	NMIディープソフトウェアスタンバイ解除要求フラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル

R_PG_LPC.c

使用RPDL関数

R_LPC_GetStatus

詳細

- ディープソフトウェアスタンバイ解除要求フラグを取得します。
- 取得するフラグに対応する引数に、フラグ値の格納先アドレスを指定してください。
- 取得しないフラグには0を指定してください。
- 本関数を呼び出すとRSTSR.LVD1F(LVD1検知フラグ)、RSTSR.LVD2F(LVD2検知フラグ)、RSTSR.DPSRSTF(ディープソフトウェアスタンバイリセットフラグ)およびDPSIFR(ディープソフトウェアスタンバイ解除要求フラグ)がクリアされます。これらのフラグを同時に取得する必要がある場合は本関数の代わりにR_PG_LPC_GetStatusを使用してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
bool irq0;
bool nmi;
void func(void)
{
    //  ディープソフトウェアスタンバイ解除要求フラグを取得 (IQR0-A および NMI)
    R_PD_LPC_GetDeepSoftwareStandbyCancelFlag ( &irq0, 0, 0, 0, 0, 0, 0, &nmi );
    if( irq0 ){
        // IRQ0-Aからのディープソフトウェアスタンバイ解除要求検出時処理
    }
    if( nmi ){
        //NMIからのディープソフトウェアスタンバイ解除要求検出時処理
    }
}
```

4.3.11 R_PG_LPC_GetStatus

定義 bool R_PG_LPC_GetStatus(uint16_t *data)

概要 消費電力低減機能の状態を取得

引数

uint16_t *data	ステータス情報の格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル R_PG_LPC.h

使用RPDL関数 R_LPC_GetStatus

詳細

- リセットステータスとディープソフトウェアスタンバイ解除要求フラグを取得します。
- 本関数を呼び出すと、RPDLの関数 R_LPC_GetStatus が直接呼び出されます。
- 取得した情報は以下の形式で格納されます。

b15	b14-b11	b10	b9	b8
リセットステータス(RSTSR) (0: 未検出; 1:検出)				
ディープソフトウェアリセット	0	LVD2	LVD1	パワーオンリセット

b7	b6	b5	b4	b3	b2	b1	b0
ディープソフトウェアスタンバイ解除要求検出(DPSIFR) (0: 未検出; 1:検出)							
NMI	USB スタンバイ/レジューム	RTC	LVD	IRQ3-A	IRQ2-A	IRQ1-A	IRQ0-A

- 本関数を呼び出すとRSTSR.LVD1F(LVD1検知フラグ)、RSTSR.LVD2F(LVD2検知フラグ)、RSTSR.DPSRSTF(ディープソフトウェアスタンバイリセットフラグ)およびDPSIFR(ディープソフトウェアスタンバイ解除要求フラグ)がクリアされます。
- RSTSR.PORF(パワーオンリセットフラグ)は端子リセットでのみクリアされます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
uint16_t data;
void func(void)
{
    //消費電力低減機能の状態を取得
    R_PG_LPC_GetStatus( &data );
    //ディープソフトウェアリセットを検出したか
    if( (data >> 15) & 0x1 ){
        if( (data >> 7) &0x1){
            //NMIによるディープソフトウェアスタンバイ解除時処理
        }
        else if( data &0x1){
            //IRQ0-Aによるディープソフトウェアスタンバイ解除時処理
        }
    }
}
```

4.3.12 R_PG_LPC_WriteBackup

定義 bool R_PG_LPC_WriteBackup (uint8_t * data, uint8_t count)

概要 ディープスタンバイバックアップレジスタへの書き込み

引数

uint8_t * data	ディープスタンバイバックアップレジスタに書き込むデータ
uint8_t count	書き込むデータのバイト数 (1～32)

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_LPC.h

使用RPDL関数 R_LPC_WriteBackup

詳細

- ディープスタンバイバックアップレジスタにデータを書き込みます。
- 本関数を呼び出すと、RPDLの関数 R_LPC_WriteBackup が直接呼び出されます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t w_data[]="ABCDEFG";
uint8_t r_data[]="--------";

void func1(void)
{
    //消費電力低減機能の設定
    R_PG_LPC_Set();

    //ディープスタンバイバックアップレジスタへの書き込み
    R_PG_LPC_WriteBackup( w_data, 7 );

    //ディープソフトウェアスタンバイモードへの移行
    R_PG_LPC_DeepSoftwareStandby();
}

void func2(void)
{
    //ディープスタンバイバックアップレジスタからの読み出し
    R_PG_LPC_ReadBackup( r_data, 7 );
}
```

4.3.13 R_PG_LPC_ReadBackup

定義 bool R_PG_LPC_ReadBackup (uint8_t * data, uint8_t count)

概要 ディープスタンバイバックアップレジスタからの読み出し

引数

uint8_t * data	読み出したデータの保存先
uint8_t count	読み出すデータのバイト数 (1～32)

戻り値

true	読み出しに成功した場合
false	読み出しに失敗した場合

出力先ファイル R_PG_LPC.h

使用RPDL関数 R_LPC_ReadBackup

詳細

- ディープスタンバイバックアップレジスタからデータを読み出します。
- 本関数を呼び出すと、RPDLの関数 R_LPC_ReadBackup が直接呼び出されます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t w_data[]="ABCDEFG";
uint8_t r_data[]="--------";

void func1(void)
{
    //消費電力低減機能の設定
    R_PG_LPC_Set();

    //ディープスタンバイバックアップレジスタへの書き込み
    R_PG_LPC_WriteBackup( w_data, 7 );

    //ディープソフトウェアスタンバイモードへの移行
    R_PG_LPC_DeepSoftwareStandby();
}

void func2(void)
{
    //ディープスタンバイバックアップレジスタからの読み出し
    R_PG_LPC_ReadBackup( r_data, 7 );
}
```

4.4 割り込みコントローラ (ICUa)

4.4.1 R_PG_ExtInterrupt_Set_〈割り込み種別〉

定義　bool R_PG_ExtInterrupt_Set_〈割り込み種別〉(void)
〈割り込み種別〉 : IRQ0～IRQ15、またはNMI

概要　外部割り込みの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ExtInterrupt_〈割り込み種別〉.c
〈割り込み種別〉 : IRQ0～IRQ15、またはNMI

使用RPDL関数　R_INTC_CreateExtInterrupt

詳細

- 外部割り込み(IRQ0～IRQ15、またはNMI) を有効にし、使用する外部割り込み端子の入力方向と入力バッファの設定を行います。IRQnは[周辺機能別使用端子]ウィンドウ上の選択に従い、使用端子(IRQn-A/B)の設定を行います。
- GUI上で割り込み通知関数名が指定されている場合、CPUへ割り込みが発生すると指定された名前の関数が呼び出されます。通知関数は次の定義で作成してください。
  void 〈割り込み通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- GUI上で割り込み優先レベルを0に設定した場合、外部割り込み要求信号が入力されてもCPU割り込みは発生しません。割り込み要求フラグは
  R_PG_ExtInterrupt_GetRequestFlag_〈割り込み種別〉 により取得することができ、
  R_PG_ExtInterrupt_ClearRequestFlag_〈割り込み種別〉 によりクリアすることができます。

使用例1　割り込み通知関数名にIrq0IntFuncを指定する場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
}

//IRQ0通知関数
void Irq0IntFunc(void)
{
    func_irq0();    //IRQ0の処理
}
```

使用例2 割り込み優先レベルを0に設定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool flag;

	//IRQ0を設定する
	R_PG_ExtInterrupt_Set_IRQ0();

	do{
		//IRQ0の割り込み要求フラグを取得する
		R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
	}while( ! flag )

	func_irq0();	//IRQ0の処理

	//IRQ0の割り込み要求フラグをクリアする
	R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();
}
```

4.4.2 R_PG_ExtInterrupt_Disable_*〈割り込み種別〉*

定義　bool R_PG_ExtInterrupt_Disable_*〈割り込み種別〉*(void)
〈割り込み種別〉 ： IRQ0～IRQ15

概要　外部割り込みの設定解除

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ExtInterrupt_*〈割り込み種別〉*.c
〈割り込み種別〉 ： IRQ0～IRQ15

使用RPDL関数　R_INTC_ControlExtInterrupt

詳細
- 外部割り込み(IRQ0～IRQ15) を無効にします。
- 外部割込みに使用した端子の設定(入出力方向、入力バッファ設定)は保持されます。

使用例　割り込み通知関数名にIrq0IntFuncを指定する場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
      //IRQ0を設定する
      R_PG_ExtInterrupt_Set_IRQ0();
}

//外部割り込み(IRQ0)通知関数
void Irq0IntFunc (void)
{
      //IRQ0を無効にする
      R_PG_ExtInterrupt_Disable_IRQ0();

      func_irq0();      //IRQ0の処理
}
```

4.4.3 R_PG_ExtInterrupt_GetRequestFlag_*〈割り込み種別〉*

定義　bool R_PG_ExtInterrupt_GetRequestFlag_*〈割り込み種別〉*(bool * flag)
〈割り込み種別〉 ： IRQ0～IRQ15、またはNMI

概要　外部割り込み要求フラグの取得

引数

bool * flag	割り込み要求フラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル　R_PG_ExtInterrupt_*〈割り込み種別〉*.c
〈割り込み種別〉 ： IRQ0～IRQ15、またはNMI

使用RPDL関数　R_INTC_GetExtInterruptStatus

詳細
- 外部割り込み(IRQ0～IRQ15、またはNMI) の割り込み要求フラグを取得します。
  割り込み要求がある場合、flagで指定した格納先にtrueが入ります。

使用例　割り込み優先レベルを0に設定した場合

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    bool flag;

    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();

    do{
        //IRQ0の割り込み要求フラグを取得する
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
    }while( ! flag )

    func_irq0();      //IRQ0の処理

    //IRQ0の割り込み要求フラグをクリアする
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();
}
```

4.4.4 R_PG_ExtInterrupt_ClearRequestFlag_〈*割り込み種別*〉

定義　bool R_PG_ExtInterrupt_ClearRequestFlag_〈*割り込み種別*〉(void)
〈*割り込み種別*〉 ： IRQ0～IRQ15、またはNMI

概要　外部割り込み要求フラグのクリア

引数　なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル　R_PG_ExtInterrupt_〈*割り込み種別*〉.c
〈*割り込み種別*〉 ： IRQ0～IRQ15、またはNMI

使用RPDL関数　R_INTC_ControlExtInterrupt

詳細
- 外部割り込み(IRQ0～IRQ15、またはNMI) の割り込み要求フラグをクリアします。
- 割り込みがLowレベル検出の場合、要求フラグは割り込み入力端子へのHighレベル入力でクリアされます。Lowレベル検出の場合は本関数により外部割込み要求フラグをクリアできません。

使用例　割り込み優先レベルを0に設定した場合

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    bool flag;

    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();

    do{
        //IRQ0の割り込み要求フラグを取得する
        R_PG_ExtInterrupt_GetRequestFlag_IRQ0( &flag );
    }while( ! flag )

    func_irq0();      //IRQ0の処理

    //IRQ0の割り込み要求フラグをクリアする
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();
}
```

4.4.5 R_PG_SoftwareInterrupt_Set

定義 bool R_PG_SoftwareInterrupt_Set(void)

概要 ソフトウェア割り込みの設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_SoftwareInterrupt.c

使用RPDL関数 R_INTC_CreateSoftwareInterrupt

詳細

- ソフトウェア割り込みを設定します。
- 本関数の呼び出しではソフトウェア割り込みは発生しません。ソフトウェア割り込みを発生させるには本関数の呼出し後にR_PG_SoftwareInterrupt_Generateを呼び出してください。

使用例 GUI上でソフトウェア割り込み通知関数名に SwIntFunc を指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void SwIntFunc(void);

void func(void)
{
    //ソフトウェア割り込みを設定する
    R_PG_SoftwareInterrupt_Set();

    //ソフトウェア割り込みを発生させる
    R_PG_SoftwareInterrupt_Generate();
}

void SwIntFunc(void)
{
    //ソフトウェア割り込みの処理
}
```

4.4.6 R_PG_SoftwareInterrupt_Generate

定義 bool R_PG_SoftwareInterrupt_Generate(void)

概要 ソフトウェア割り込みの生成

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_SoftwareInterrupt.c

使用RPDL関数 R_INTC_Write

詳細
- ソフトウェア割り込みを発生させます。
- 本関数の呼び出す前にR_PG_SoftwareInterrupt_Setを呼び出してソフトウェア割り込みを設定してください。

使用例 GUI上でソフトウェア割り込み通知関数名に SwIntFunc を指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void SwIntFunc(void);

void func(void)
{
    //ソフトウェア割り込みを設定する
    R_PG_SoftwareInterrupt_Set();

    //ソフトウェア割り込みを発生させる
    R_PG_SoftwareInterrupt_Generate();
}

void SwIntFunc(void)
{
    //ソフトウェア割り込みの処理
}
```

4.4.7 R_PG_FastInterrupt_Set

定義 bool R_PG_FastInterrupt_Set (void)

概要 高速割り込みの設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_FastInterrupt.c

使用RPDL関数 R_INTC_CreateFastInterrupt

詳細

- GUI上で指定した割り込み要因を高速割り込みに設定します。指定する割り込み要因の設定と有効化は行いません。高速割り込みに設定する割り込み要因の設定と有効化は、周辺機能の関数により行ってください。
- 本関数では高速割り込みベクタレジスタ(FINTV)を設定するために無条件トラップ(BRK命令)を使用しています。割り込みが無効の状態(プロセッサステータスワードの割り込み許可ビット(I)が0の場合)には、本関数はロックします。
- GUI上で高速割込みに指定した割り込みのハンドラは、#pragma interruptでfintを指定してコンパイルすることにより高速割り込みとして処理されます。

使用例 GUI上でIRQ0を高速割り込みに指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //IRQ0を高速割り込みに設定する
    R_PG_FastInterrupt_Set ();

    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
}
```

4.4.8 R_PG_Exception_Set

定義　bool R_PG_Exception_Set (void)

概要　例外ハンドラの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Exception.c

使用RPDL関数　R_INTC_CreateExceptionHandlers

詳細

- 例外通知関数を設定します。GUI上で例外通知関数名が指定されている場合、本関数の呼び出し後に例外が発生すると、指定された名前の関数が呼び出されます。
  例外通知関数は次の定義で作成してください。
  void *〈例外通知関数名〉*(void)
  例外通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上で次の例外通知関数を設定した場合
特権命令例外 ： PrivInstExcFunc
未定義命令例外 ： UndefInstExcFunc
浮動小数点例外 ： FpExcFunc

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //例外ハンドラの設定
    R_PG_Exception_Set();
}

void PrivInstExcFunc(){
    func_pi_excep();    //特権命令例外発生時の処理
}

void UndefInstExcFunc (){
    func_ui_excep();    //未定義命令例外発生時の処理
}

void FpExcFunc (){
    func_fp_excep();    //浮動小数点例外発生時の処理
}
```

4.5 バス

4.5.1 R_PG_ExtBus_SetBus

定義　bool R_PG_ExtBus_SetBus(void)

概要　バス端子とバスエラー監視の設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ExtBus.c

使用RPDL関数　R_BSC_Create

詳細

- バス端子とバスエラー監視を設定します。
- 本関数内でバスエラー割り込みを設定します。GUI上で[バスエラー割り込みを通知関数呼び出しで通知する]を指定した場合、CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。
  void *〈割り込み通知関数名〉*(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- バスエラーの検出状態はR_PG_ExtBus_GetErrorStatusにより取得することができます。
- 使用する端子(-A/-B/-C)は、[周辺機能別使用端子]ウィンドウ上の選択に従い設定されます。
- 外部バスクロック(BCLK)、SDRAMクロック(SDCLK)はR_PG_Clock_Setにより設定してください。

使用例　バスエラー割り込み通知関数名にBusErrFuncを指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_ExtBus_SetBus();   //バス端子とバスエラー監視の設定
}

//バスエラー通知関数
void BusErrFunc(void)
{
    bool addr_err;
    uint8_t master;
    uint16_t err_addr;

    //バスエラー検出状態の取得
    R_PG_ExtBus_GetErrorStatus( &addr_err, 0, &master, &err_addr );
    if( addr_err ){
        //不正アドレスアクセスエラー検出時処理
    }

    //バスエラーステータスレジスタのクリア
    R_PG_ExtBus_ClearErrorFlags();
}
```

4.5.2 R_PG_ExtBus_GetErrorStatus

定義

bool R_PG_ExtBus_GetErrorStatus
(bool * addr_err, bool * time_err, uint8_t * master, uint16_t * err_addr)

概要

バスエラー検出状態の取得

引数

bool * addr_err	不正アドレスアクセスフラグの格納先
bool * time_err	タイムアウトフラグの格納先
uint8_t * master	バスエラーを発生させたバスマスタのIDコードの格納先 バスマスタに対応するIDコード: 0:CPU 3:DMAC/DTC 6:EDMAC 7:EXDMAC
uint16_t * err_addr	バスエラーを起こしたアドレスの上位13ビットの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_ExtBus.c

使用RPDL関数

R_BSC_GetStatus

詳細

- バスエラーステータスレジスタからバスエラー検出状態を取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。

使用例

バスエラー割り込み通知関数名にBusErrFuncを指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_ExtBus_SetBus();   //バス端子とバスエラー監視の設定
}

//バスエラー通知関数
void BusErrFunc(void)
{
    bool addr_err;
    uint8_t master;
    uint16_t err_addr;

    //バスエラー検出状態の取得
    R_PG_ExtBus_GetErrorStatus( &addr_err, 0, &master, &err_addr );
    if( addr_err ){
        //不正アドレスアクセスエラー検出時処理
    }

    //バスエラーステータスレジスタのクリア
    R_PG_ExtBus_ClearErrorFlags();
}
```

4.5.3 R_PG_ExtBus_ClearErrorFlags

定義 bool R_PG_ExtBus_ClearErrorFlags(void)

概要 バスエラーステータスレジスタのクリア

引数 なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル R_PG_ExtBus.c

使用RPDL関数 R_BSC_Control

詳細

- バスエラーステータスレジスタ（不正アドレスアクセスフラグ、タイムアウトフラグ、バスマスタIDコード、アクセス先アドレスの値）をクリアします。
- バスエラー割り込み要求フラグ(IR)は本関数内でクリアされます。

使用例 バスエラー割り込み通知関数名にBusErrFuncを指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_ExtBus_SetBus();   //バス端子とバスエラー監視の設定
}

//バスエラー通知関数
void BusErrFunc(void)
{
    bool addr_err;
    uint8_t master;
    uint16_t err_addr;

    //バスエラー検出状態の取得
    R_PG_ExtBus_GetErrorStatus( &addr_err, 0, &master, &err_addr );
    if( addr_err ){
        //不正アドレスアクセスエラー検出時処理
    }

    //バスエラーステータスレジスタのクリア
    R_PG_ExtBus_ClearErrorFlags();
}
```

4.5.4 R_PG_ExtBus_SetArea_CS〈*CS領域の番号*〉

定義　bool R_PG_ExtBus_SetArea_CS〈*CS領域の番号*〉(void)

〈*CS領域の番号*〉 : 0～7

概要　CS領域の設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ExtBus_CS〈*CS領域の番号*〉.c

〈*CS領域の番号*〉 : 0～7

使用RPDL関数　R_BSC_CreateArea

詳細

- CS領域を設定します。
- 本関数を呼び出す前にR_PG_ExtBus_SetBusを呼び出して、使用する端子とバスエラー監視を設定してください。

使用例　CS0、CS6およびSDRAM領域(SDCS)を設定する場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //バス端子とバスエラー監視の設定
    R_PG_ExtBus_SetBus();

    //CS0の設定
    R_PG_ExtBus_SetArea_CS0();

    //CS6の設定
    R_PG_ExtBus_SetArea_CS6();

    //SDRAM領域(SDCS)の設定
    R_PG_ExtBus_SetArea_SDCS();
}
```

4.5.5 R_PG_ExtBus_DisableArea_CS〈*CS領域の番号*〉

定義 bool R_PG_ExtBus_DisableArea_CS〈*CS領域の番号*〉(void)

〈*CS領域の番号*〉 : 0～7

概要 CS領域の設定解除

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_CS〈*CS領域の番号*〉.c

〈*CS領域の番号*〉 : 0～7

使用RPDL関数 R_BSC_Destroy

詳細

- CS領域の設定を解除します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //バス端子とバスエラー監視の設定
    R_PG_ExtBus_SetBus();

    //CS0の設定
    R_PG_ExtBus_SetArea_CS0();

    //CS6の設定
    R_PG_ExtBus_SetArea_CS6();

    //SDRAM領域(SDCS)の設定
    R_PG_ExtBus_SetArea_SDCS();
}

void func2(void)
{
    //CS0の設定解除
    R_PG_ExtBus_DisableArea_CS0();

    //CS6の設定解除
    R_PG_ExtBus_DisableArea_CS6();

    //SDRAM領域(SDCS)の設定解除
    R_PG_ExtBus_DisableArea_SDCS();
}
```

4.5.6 R_PG_ExtBus_SetArea_SDCS

定義 bool R_PG_ExtBus_SetArea_SDCS(void)

概要 SDRAM領域(SDCS)の設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_SDRAM_CreateArea

詳細

- SDRAM領域(SDCS)を設定します。
- 本関数を呼び出す前にR_PG_ExtBus_SetBusを呼び出して、使用する端子とバスエラー監視を設定してください。

使用例 CS0、CS6およびSDRAM領域(SDCS)を設定する場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //バス端子とバスエラー監視の設定
    R_PG_ExtBus_SetBus();

    //CS0の設定
    R_PG_ExtBus_SetArea_CS0();

    //CS6の設定
    R_PG_ExtBus_SetArea_CS6();

    //SDRAM領域(SDCS)の設定
    R_PG_ExtBus_SetArea_SDCS();
}
```

4.5.7 R_PG_ExtBus_Initialize_SDCS

定義 bool R_PG_ExtBus_Initialize_SDCS(void)

概要 SDRAM初期化シーケンスの開始

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細

- SDRAMの初期化シーケンスを開始します。
- SDRAM初期化シーケンスはSDRAMアクセス、オートリフレッシュ、セルフリフレッシュ無効時に開始してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //バス端子とバスエラー監視の設定
    R_PG_ExtBus_SetBus();

    //SDRAM領域(SDCS)の設定
    R_PG_ExtBus_SetArea_SDCS();

    //SDRAM初期化シーケンスの開始
    R_PG_ExtBus_Initialize_SDCS();

    //オートリフレッシュの開始
    R_PG_ExtBus_AutoRefreshEnable_SDCS();

    //SDRAMアクセスの許可
    R_PG_ExtBus_AccessEnable_SDCS();
}
```

4.5.8 R_PG_ExtBus_AutoRefreshEnable_SDCS

定義 bool R_PG_ExtBus_AutoRefreshEnable_SDCS(void)

概要 SDRAMオートリフレッシュの有効化

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細
- SDRAMオートリフレッシュ開始します。
- SDRAMオートリフレッシュはセルフリフレッシュ無効時に開始してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //バス端子とバスエラー監視の設定
    R_PG_ExtBus_SetBus();

    //SDRAM領域(SDCS)の設定
    R_PG_ExtBus_SetArea_SDCS();

    //SDRAM初期化シーケンスの開始
    R_PG_ExtBus_Initialize_SDCS();

    //SDRAMオートリフレッシュの開始
    R_PG_ExtBus_AutoRefreshEnable_SDCS();

    //SDRAMアクセスの許可
    R_PG_ExtBus_AccessEnable_SDCS();
}

void func2(void)
{
    //SDRAMアクセスの無効化
    R_PG_ExtBus_AccessDisable_SDCS();

    // SDRAMオートリフレッシュの停止
    R_PG_ExtBus_AutoRefreshDisable_SDCS();
}
```

4.5.9 R_PG_ExtBus_AutoRefreshDisable_SDCS

定義 bool R_PG_ExtBus_AutoRefreshDisable_SDCS(void)

概要 SDRAMオートリフレッシュの無効化

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細
- SDRAMオートリフレッシュを停止します。
- SDRAMオートリフレッシュはセルフリフレッシュ無効時に停止してください。

使用例 R_PG_ExtBus_AutoRefreshEnable_SDCSの使用例を参照してください。

4.5.10 R_PG_ExtBus_SelfRefreshEnable_SDCS

定義 bool R_PG_ExtBus_SelfRefreshEnable_SDCS(void)

概要 SDRAMセルフリフレッシュの有効化

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細

- SDRAMセルフリフレッシュモードへ移行します。
- SDRAMセルフリフレッシュモードはSDRAMアクセス無効、オートリフレッシュ有効時に開始してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //バス端子とバスエラー監視の設定
    R_PG_ExtBus_SetBus();

    //SDRAM領域(SDCS)の設定
    R_PG_ExtBus_SetArea_SDCS();

    //SDRAM初期化シーケンスの開始
    R_PG_ExtBus_Initialize_SDCS();

    // SDRAMオートリフレッシュの開始
    R_PG_ExtBus_AutoRefreshEnable_SDCS();

    //SDRAMアクセスの許可
    R_PG_ExtBus_AccessEnable_SDCS();
}

void func2(void)
{
    //SDRAMアクセスの無効化
    R_PG_ExtBus_AccessDisable_SDCS();

    // SDRAMセルフリフレッシュモードへの移行
    R_PG_ExtBus_SelfRefreshEnable_SDCS();
}

void func3(void)
{
    // SDRAMセルフリフレッシュモードの終了
    R_PG_ExtBus_SelfRefreshDisable_SDCS();

    //SDRAMアクセスの有効化
    R_PG_ExtBus_AccessEnable_SDCS();
}
```

4.5.11 R_PG_ExtBus_SelfRefreshDisable_SDCS

定義 bool R_PG_ExtBus_SelfRefreshDisable_SDCS(void)

概要 SDRAMセルフリフレッシュの無効化

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細
- SDRAMセルフリフレッシュモードを終了します。

使用例 R_PG_ExtBus_SelfRefreshEnable_SDCSの使用例を参照してください。

4.5.12 R_PG_ExtBus_AccessEnable_SDCS

定義 bool R_PG_ExtBus_AccessEnable_SDCS(void)

概要 SDRAMアクセスの有効化

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細
- SDRAMアクセスを許可します。

使用例 R_PG_ExtBus_AutoRefreshEnable_SDCS および R_PG_ExtBus_SelfRefreshEnable_SDCSの使用例を参照してください。

4.5.13 R_PG_ExtBus_AccessDisable_SDCS

定義 bool R_PG_ExtBus_AccessDisable_SDCS(void)

概要 SDRAMアクセスの無効化

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_ExtBus_SDCS.c

使用RPDL関数 R_BSC_Control

詳細
- SDRAMアクセスを無効にします。

使用例 R_PG_ExtBus_AutoRefreshEnable_SDCS および R_PG_ExtBus_SelfRefreshEnable_SDCSの使用例を参照してください。

4.5.14 R_PG_ExtBus_GetStatus_SDCS

定義

bool R_PG_ExtBus_GetStatus_SDCS
(bool * mode_setting, bool * initializing, bool * rec_trans)

概要

SDRAMステータスの取得

引数

bool * mode_setting	モードレジスタステータスビットの格納先 (1:モードレジスタセット動作中)
bool * initializing	初期化ステータスビットの格納先 (1:初期化シーケンス中)
bool * rec_trans	セルフリフレッシュ移行/復帰ステータスビットの格納先 (1: セルフリフレッシュ移行/復帰動作中)

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_ExtBus.c

使用RPDL関数

R_BSC_GetStatus

詳細

- SDRAMステータスレジスタからSDRAMのステータスを取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。

使用例

バスエラー割り込み通知関数名にBusErrFuncを指定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool mode_setting, initializing, rec_trans;

//バスエラー通知関数
void BusErrFunc(void)
{
    //SDRAMステータスの取得
    R_PG_ExtBus_GetStatus_SDCS( &mode_setting, &initializing, &rec_trans );
}
```

4.6 DMAコントローラ（DMACA）

4.6.1 R_PG_DMAC_Set_C〈*チャネル番号*〉

定義　bool R_PG_DMAC_Set_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉： 0～3

概要　DMACの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_DMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉： 0～3

使用RPDL関数　R_DMAC_Create

詳細

- DMACのモジュールストップ状態を解除して初期設定します。
- 転送開始要因に割り込みを選択した場合は、本関数を呼び出した後R_PG_DMAC_Activate_C〈*チャネル番号*〉を呼び出すことにより割り込みの入力待ち状態になります。転送開始要因にソフトウェアトリガを選択した場合は、本関数を呼び出した後R_PG_DMAC_StartTransfer_C〈*チャネル番号*〉を呼び出すことにより転送を開始します。
- GUI上で割り込み通知関数名を指定した場合、本関数内でDMA割り込みを設定します。CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。　void 〈*割り込み通知関数名*〉(void)
割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- シリアルの送信データをDMA転送する場合は、GUI上で以下の設定をしてください。

DMAC設定

転送開始要因	： TXI (SCIa送信データエンプティ割り込み)
転送終了時処理	： 起動要因の割り込みフラグをクリアする
転送先開始アドレス	： シリアル送信データレジスタ(TDR)のアドレス *転送先開始アドレスはプログラムからも設定することができます。使用例2,3を参照してください。
転送先アドレス更新モード	： 固定
1データのビット長	： 1バイト

SCIa設定

データ送信方法	： DMAC により送信データを転送する

関数の使用方法については使用例2を参照してください。

- シリアルの受信データをDMA転送する場合は、GUI上で以下の設定をしてください。

DMAC設定

転送開始要因	： RXI (SCIa受信データフル割り込み)
転送終了時処理	： 起動要因の割り込みフラグをクリアする
転送元開始アドレス	： シリアル受信データレジスタ(RDR)のアドレス *転送元開始アドレスはプログラムからも設定することができます。使用例2,3を参照してください。
転送元アドレス更新モード	： 固定
1データのビット長	： 1バイト

SCIa設定

データ受信方法	： DMACにより受信データを転送する

関数の使用方法については使用例3を参照してください。

使用例1 IRQ0割り込みにより転送を開始する場合

- GUI上でDMAC0の転送開始要因をIRQ0割り込みに設定
- GUI上でDMA0割り込み通知関数名に Dmac0IntFunc を指定
- GUI上でIRQ0の割り込み要求先をDMACに設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMACを停止
    R_PG_DMAC_StopModule_C0();
}
```

使用例2 DMA転送によりシリアル送信データを転送する場合

- GUI上でDMA0割り込み通知関数名に Dmac0IntFunc を指定
- SCI0の送信データエンプティ割り込みにより転送開始

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//DMA転送終了フラグ
volatile bool sci_dma_transfer_complete;

//送信データ
uint8_t tr[]="ABCDEFG";

void func(void)
{
    //DMA転送終了フラグの初期化
    sci_dma_transfer_complete = false;

    //SCI0を設定する
    R_PG_SCI_Set_C0();

    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //DMA転送元、転送先アドレス、転送カウンタを設定する
    R_PG_DMAC_SetSrcAddress_C0( tr );
    R_PG_DMAC_SetDestAddress_C0((void*)&(SCI0.TDR));
    R_PG_DMAC_SetTransferCount_C0( 8 );

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();

    //SCI0の送信を有効にする（TXI割り込みが発生し、DMA転送が開始）
    R_PG_SCI_SendAllData_C0(
        PDL_NO_PTR,
        PDL_NO_DATA
    );

    //DMA転送終了を待つ
    while (sci_dma_transfer_complete == false);
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //シリアル送信終了フラグ
    bool sci_transfer_complete;
    sci_transfer_complete = false;

    //シリアル送信の終了を待つ
    do{
        R_PG_SCI_GetTransmitStatus_C0( &sci_transfer_complete );
    } while( ! sci_transfer_complete );

    //シリアル通信を停止
    R_PG_SCI_StopCommunication_C0();

    //DMACを停止
    R_PG_DMAC_StopModule_C0();

    sci_dma_transfer_complete = true;
}
```

使用例3 DMA転送によりシリアル受信データを転送する場合

- GUI上でDMA0割り込み通知関数名に Dmac0IntFunc を指定
- SCI0の受信データフル割り込みにより転送開始

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//受信データ格納先
uint8_t re[]="--------";

void func(void)
{
    //DMA転送終了フラグの初期化
    sci_dma_transfer_complete = false;

    //SCI0を設定する
    R_PG_SCI_Set_C0();

    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //DMA転送元、転送先アドレス、転送カウンタを設定する
    R_PG_DMAC_SetSrcAddress_C0((void*)&(SCI0.RDR) );
    R_PG_DMAC_SetDestAddress_C0( re );
    R_PG_DMAC_SetTransferCount_C0( 8 );

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();

    //SCI0の受信を開始する
    R_PG_SCI_ReceiveAllData_C0(
        PDL_NO_PTR,
        PDL_NO_DATA
    );
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //シリアル通信を停止
    R_PG_SCI_StopCommunication_C0();

    //DMACを停止
    R_PG_DMAC_StopModule_C0();
}
```

4.6.2 R_PG_DMAC_Activate_C〈チャネル番号〉

定義　bool R_PG_DMAC_Activate_C〈チャネル番号〉(void)
〈チャネル番号〉： 0～3

概要　DMACを転送開始トリガの入力待ち状態に設定

生成条件　転送開始要因に割り込みを選択

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_DMAC_C〈チャネル番号〉.c
〈チャネル番号〉： 0～3

使用RPDL関数　R_DMAC_Control

詳細
- 転送開始要因を割り込みに設定したDMACのチャネルをトリガ入力待ち状態に設定します。
- 本関数は転送開始要因に割り込みを指定した場合に生成されます。
- あらかじめR_PG_DMAC_Set_C〈チャネル番号〉によりDMACのチャネルを設定してください。

使用例　GUI上で以下の通り設定した場合
- ノーマル転送モードでDMAC0の転送開始要因をIRQ0割り込みに設定した場合
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMACを停止
    R_PG_DMAC_StopModule_C0();
}
```

4.6.3 R_PG_DMAC_StartTransfer_C〈*チャネル番号*〉

定義　bool R_PG_DMAC_StartTransfer_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉： 0～3

概要　データ転送の開始(ソフトウェアトリガ)

生成条件　転送開始要因にソフトウェアトリガを選択

引数　なし

戻り値

true	転送開始が正しく行われた場合
false	転送開始に失敗した場合

出力先ファイル　R_PG_DMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉： 0～3

使用RPDL関数　R_DMAC_Control

詳細
- 転送開始要因をソフトウェアトリガに設定したDMACチャネルの転送を開始します。
- 本関数は転送開始要因にソフトウェアトリガを指定した場合に生成されます。
- あらかじめR_PG_DMAC_Set_C〈*チャネル番号*〉によりDMACのチャネルを設定してください。

使用例　GUI上で以下の通り設定した場合
- ノーマル転送モードでDMAC0の転送開始要因をソフトウェアトリガに設定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

volatile bool transferred;

void func(void)
{
	transferred = false;

	//DMAC0を設定する
	R_PG_DMAC_Set_C0();

	while( transferred == false ){
		//DMAC0の転送を開始する
		R_PG_DMAC_StartTransfer_C0();
	}
	//DMACを停止
	R_PG_DMAC_StopModule_C0();
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
	transferred = true;
}
```

4.6.4 R_PG_DMAC_Suspend_C〈*チャネル番号*〉

定義　bool R_PG_DMAC_Suspend_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉： 0～3

概要　データ転送の中断

引数　なし

戻り値

true	中断に成功した場合
false	中断に失敗した場合

出力先ファイル　R_PG_DMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉： 0～3

使用RPDL関数　R_DMAC_Control

詳細
- DMA転送を中断します。
- 転送開始要因に割り込みを選択した場合、転送を再開するには転送開始要因の割り込み要求フラグをクリアし、R_PG_DMAC_Activate_C〈*チャネル番号*〉により割り込み入力待ち状態に設定してください。

使用例　GUI上で以下の通り設定した場合
- ノーマル転送モードでDMAC0の転送開始要因をIRQ0割り込みに設定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定
- IRQ1の割り込み通知関数名にIrq1ExtIntFuncを指定
- IRQ2の割り込み通知関数名にIrq2ExtIntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_DMAC_Set_C0();       //DMAC0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();       //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ1();       //IRQ1を設定する
    R_PG_ExtInterrupt_Set_IRQ2();       //IRQ2を設定する
    R_PG_DMAC_Activate_C0();       //DMAC0を転送開始トリガ入力待ち状態にする
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    R_PG_DMAC_StopModule_C0();       //DMAC0を停止
}

//IRQ1割り込みでDMA転送停止
void Irq1ExtIntFunc (void)
{
    R_PG_DMAC_Suspend_C0();       //DMAC0の転送を中断
}

//IRQ2割り込みでDMA転送再開
void Irq2ExtIntFunc (void)
{
    R_PG_ExtInterrupt_ClearRequestFlag_IRQ0();       //トリガの要求フラグクリア
    R_PG_DMAC_Activate_C0();       //DMA転送有効化
}
```

4.6.5 R_PG_DMAC_GetTransferCount_C〈*チャネル番号*〉

定義　bool R_PG_DMAC_GetTransferCount_C〈*チャネル番号*〉(uint16_t * count)

〈*チャネル番号*〉： 0～3

概要　転送カウンタ値の取得

引数

uint16_t * count	転送カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_DMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉： 0～3

使用RPDL関数　R_DMAC_GetStatus

詳細

- 現在の転送カウンタの値を取得します。
- DMA割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。DMA割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前に R_PG_DMAC_ClearInterruptFlag_C〈*チャネル番号*〉を呼び出してください。

使用例　GUI上で以下の通り設定した場合

- DMAC0の転送開始要因に割り込みを選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    uint16_t count;

    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();

    //転送カウンタ値が10未満になるのを待つ
    do{
        R_PG_DMAC_GetTransferCount_C0( & count );
    } while( count >= 10 );

    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
}
```

4.6.6 R_PG_DMAC_SetTransferCount_C〈*チャネル番号*〉

定義

bool R_PG_DMAC_SetTransferCount_C〈*チャネル番号*〉(uint16_t count)

〈 *チャネル番号* 〉: 0～3

概要

転送カウンタ値の設定

引数

uint16_t count	転送カウンタに設定する値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_DMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0～3

使用RPDL関数

R_DMAC_Control

詳細

- 転送カウンタの値を設定します。
- 有効な値はノーマル転送モードでは0～65535 (0:フリーランニングモード) 、リピート転送モードおよびブロック転送モードでは0～1023 (0:1024回)です。

使用例

GUI上で以下の通り設定した場合

- DMAC0の転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();

    // DMAC0の設定を変更
    R_PG_DMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_DMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_DMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.7 R_PG_DMAC_GetRepeatBlockSizeCount_C〈チャネル番号〉

定義

bool R_PG_DMAC_GetRepeatBlockSizeCount_C〈チャネル番号〉(uint16_t * count)

〈 チャネル番号 〉: 0～3

概要

リピート/ブロックサイズカウンタ値の取得

生成条件

転送モードにリピート転送モードまたはブロック転送モードを選択

引数

uint16_t * count	カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_DMAC_C 〈チャネル番号〉.c

〈チャネル番号〉: 0～3

使用RPDL関数

R_DMAC_GetStatus

詳細

- リピート/ブロックサイズカウンタの現在の値を取得します。
- DMA割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。DMA割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前に R_PG_DMAC_ClearInterruptFlag_C〈チャネル番号〉を呼び出してください。

使用例

GUI上で以下の通り設定した場合

- リピート転送モードでDMAC0を設定
- 転送開始要因は割り込み

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    uint16_t count;

    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();

    //リピートサイズカウンタ値が10未満になるのを待つ
    do{
        R_PG_DMAC_GetRepeatBlockSizeCount_C0( & count );
    } while( count >= 10 );

    //転送を中断
    R_PG_DMAC_Suspend_C0();
}
```

4.6.8 R_PG_DMAC_SetRepeatBlockSizeCount_C〈チャネル番号〉

定義 bool R_PG_DMAC_SetRepeatBlockSizeCount_C〈チャネル番号〉(uint16_t count)
〈 チャネル番号 〉: 0～3

概要 リピート/ブロックサイズカウンタ値の設定

生成条件 転送モードにリピート転送モードまたはブロック転送モードを選択

引数

uint16_t count	リピート/ブロックサイズカウンタに設定する値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DMAC_C 〈チャネル番号〉.c
〈チャネル番号〉: 0～3

使用RPDL関数 R_DMAC_Control

詳細
- リピート/ブロックサイズカウンタの現在の値を設定します。
- 有効な値はリピート転送モードでは0～1023 (0:1024回)、ブロック転送モードでは1～1023です。

使用例 GUI上で以下の通り設定した場合
- リピート転送モードでDMAC0を設定
- 転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
    // DMAC0の設定を変更
    R_PG_DMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    R_PG_DMAC_SetRepeatBlockSizeCount_C0( repeat_count );   //リピートサイズカウンタ値
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.9 R_PG_DMAC_ClearInterruptFlag_C〈チャネル番号〉

定義 bool R_PG_DMAC_ClearInterruptFlag_C〈チャネル番号〉(bool* int_request)
〈 チャネル番号 〉: 0～3

概要 割り込み要求フラグの取得とクリア

生成条件 DMA割り込み有効時

引数

bool* int_request	割り込み要求フラグの格納先

戻り値

true	取得とクリアに成功した場合
false	取得とクリアに失敗した場合

出力先ファイル R_PG_DMAC_C 〈チャネル番号〉.c
〈チャネル番号〉: 0～3

使用RPDL関数 R_DMAC_GetStatus

詳細
- DMA割り込み要求フラグ(IRフラグ)を取得し、クリアします。

使用例 GUI上で以下の通り設定した場合
- ノーマル転送モードでDMAC0を設定
- 転送開始要因は割り込み
- DMA割り込み有効
- DMA割り込み優先レベルは0

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool int_request;

	//DMAC0を設定する
	R_PG_DMAC_Set_C0();

	//DMAC0を転送開始トリガ入力待ち状態にする
	R_PG_DMAC_Activate_C0();

	//IRフラグが1になるのを待つ
	do{
		R_PG_DMAC_ClearInterruptFlag_C0( & int_request );
	} while( int_request == false );
}
```

4.6.10 R_PG_DMAC_GetTransferEndFlag_C〈チャネル番号〉

定義

bool R_PG_DMAC_GetTransferEndFlag_C〈チャネル番号〉(bool* end)

〈 チャネル番号 〉: 0～3

概要

転送終了フラグの取得

引数

bool* end	転送終了フラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_DMAC_C 〈チャネル番号〉.c

〈チャネル番号〉: 0～3

使用RPDL関数

R_DMAC_GetStatus

詳細

- 転送終了フラグを取得します。
- DMA割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。DMA割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前にR_PG_DMAC_ClearInterruptFlag_C〈チャネル番号〉を呼び出してください。
- 転送終了フラグは本関数内でクリアされません。転送終了フラグをクリアする必要がある場合はR_PG_DMAC_ClearTransferEndFlag_C〈チャネル番号〉を呼び出してください。

使用例

GUI上で以下の通り設定した場合

- ノーマル転送モードでDMAC0を設定
- 転送開始要因は割り込み
- DMA割り込み無効

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool end;

	//DMAC0を設定する
	R_PG_DMAC_Set_C0();

	//DMAC0を転送開始トリガ入力待ち状態にする
	R_PG_DMAC_Activate_C0();

	//転送終了フラグが1になるのを待つ
	do{
		R_PG_DMAC_GetTransferEndFlag_C0( & end );
	} while( end == false );

	//転送終了フラグをクリアする
	R_PG_DMAC_ClearTransferEndFlag_C0();
}
```

4.6.11 R_PG_DMAC_ClearTransferEndFlag_C〈*チャネル番号*〉

定義 bool R_PG_DMAC_ClearTransferEndFlag_C〈*チャネル番号*〉(void)
〈 *チャネル番号* 〉: 0～3

概要 転送終了フラグのクリア

引数 なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル R_PG_DMAC_C 〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3

使用RPDL関数 R_DMAC_Control

詳細
- 転送終了フラグをクリアします。
- 転送終了フラグを取得するにはR_PG_DMAC_GetTransferEndFlag_C〈*チャネル番号*〉を呼び出してください。

使用例 GUI上で以下の通り設定した場合
- ノーマル転送モードでDMAC0を設定
- 転送開始要因は割り込み
- DMA割り込み無効

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool end;

	//DMAC0を設定する
	R_PG_DMAC_Set_C0();

	//DMAC0を転送開始トリガ入力待ち状態にする
	R_PG_DMAC_Activate_C0();

	//転送終了フラグが1になるのを待つ
	do{
		R_PG_DMAC_GetTransferEndFlag_C0( & end );
	} while( end == false );

	//転送終了フラグをクリアする
	R_PG_DMAC_ClearTransferEndFlag_C0();
}
```

4.6.12 R_PG_DMAC_GetTransferEscapeEndFlag_C〈*チャネル番号*〉

定義

bool R_PG_DMAC_GetTransferEscapeEndFlag_C〈*チャネル番号*〉(bool* end)
〈 *チャネル番号* 〉: 0～3

概要

転送エスケープ終了フラグの取得

生成条件

割り込み出力要因に[リピート/ブロックサイズの転送終了]、[転送元アドレスの拡張リピートエリアオーバフロー]または[転送先アドレスの拡張リピートエリアオーバフロー]が選択された場合

引数

bool* end	転送エスケープ終了フラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_DMAC_C 〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3

使用RPDL関数

R_DMAC_GetStatus

詳細

- 転送エスケープ終了フラグを取得します。
- EXDMA割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。DMA割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前にR_PG_DMAC_ClearInterruptFlag_C〈*チャネル番号*〉を呼び出してください。
- 転送エスケープ終了フラグは本関数内でクリアされません。転送エスケープ終了フラグをクリアする必要がある場合はR_PG_DMAC_ClearTransferEscapeEndFlag_C〈*チャネル番号*〉を呼び出してください。

使用例

GUI上で以下の通り設定した場合

- リピート転送モードでDMAC0を設定
- 転送開始要因は割り込み
- 割り込み出力要因に[リピート/ブロックサイズの転送終了]を指定
- DMA割り込み優先レベルは0

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    bool end;

    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();

    //転送エスケープ終了フラグが1になるのを待つ
    do{
        R_PG_DMAC_GetTransferEscapeEndFlag_C0( & end );
    } while( end == false );

    //転送エスケープ終了フラグをクリアする
    R_PG_DMAC_ClearTransferEscapeEndFlag_C0();
}
```

4.6.13 R_PG_DMAC_ClearTransferEscapeEndFlag_C〈*チャネル番号*〉

定義　bool R_PG_DMAC_ClearTransferEscapeEndFlag_C〈*チャネル番号*〉(void)
〈 *チャネル番号* 〉: 0～3

概要　転送エスケープ終了フラグのクリア

生成条件　割り込み出力要因に[リピート/ブロックサイズの転送終了]、[転送元アドレスの拡張リピートエリアオーバフロー]または[転送先アドレスの拡張リピートエリアオーバフロー]が選択された場合

引数　なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル　R_PG_DMAC_C 〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3

使用RPDL関数　R_DMAC_Control

詳細
- 転送エスケープ終了フラグをクリアします。
- 転送エスケープ終了フラグを取得するにはPG_DMAC_GetTransferEscapeEndFlag_C〈*チャネル番号*〉を呼び出してください。

使用例　GUI上で以下の通り設定した場合
- リピート転送モードでDMAC0を設定
- 転送開始要因は割り込み
- 割り込み出力要因に[リピート/ブロックサイズの転送終了]を指定
- DMA割り込み優先レベルは0

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool end;

	//DMAC0を設定する
	R_PG_DMAC_Set_C0();

	//DMAC0を転送開始トリガ入力待ち状態にする
	R_PG_DMAC_Activate_C0();

	//転送エスケープ終了フラグが1になるのを待つ
	do{
		R_PG_DMAC_GetTransferEscapeEndFlag_C0( & end );
	} while( end == false );

	//転送エスケープ終了フラグをクリアする
	R_PG_DMAC_ClearTransferEscapeEndFlag_C0();
}
```

4.6.14 R_PG_DMAC_SetSrcAddress_C〈*チャネル番号*〉

定義 bool R_PG_DMAC_SetSrcAddress_C〈*チャネル番号*〉(void * src_addr)

〈 *チャネル番号* 〉: 0～3

概要 転送元アドレスの設定

引数

void * src_addr	設定する転送元アドレス

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0～3

使用RPDL関数 R_DMAC_Control

詳細

- 転送元アドレスを設定します

使用例

GUI上で以下の通り設定した場合

- DMAC0の転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
    // DMAC0の設定を変更
    R_PG_DMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_DMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_DMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.15 R_PG_DMAC_SetDestAddress_C〈チャネル番号〉

定義 bool R_PG_DMAC_SetDestAddress_C〈チャネル番号〉(void * dest_addr)

〈 チャネル番号 〉: 0～3

概要 転送先アドレスの設定

引数

void * dest_addr	設定する転送先アドレス

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DMAC_C〈チャネル番号〉.c

〈チャネル番号〉: 0～3

使用RPDL関数 R_DMAC_Control

詳細

- 転送先アドレスを設定します

使用例 GUI上で以下の通り設定した場合

- DMAC0の転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
    // DMAC0の設定を変更
    R_PG_DMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_DMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_DMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.16 R_PG_DMAC_SetAddressOffset_C〈*チャネル番号*〉

定義

bool R_PG_DMAC_SetAddressOffset_C〈*チャネル番号*〉(int32_t offset)

〈 *チャネル番号* 〉: 0～3

概要

アドレスオフセット値の設定

生成条件

転送元アドレス更新モードまたは転送先アドレス更新オードにオフセット加算が選択された場合

引数

int32_t offset	設定するオフセット値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_DMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0～3

使用RPDL関数

R_DMAC_Control

詳細

- アドレスオフセット加算値を設定します
- 有効な値は +FFFFFFh ～ −1000000h です。

使用例

GUI上で以下の通り設定した場合

- DMAC0の転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定
- 転送元アドレス更新モードまたは転送先アドレス更新オードにオフセット加算を選択

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
    // DMAC0の設定を変更
    R_PG_DMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_DMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_DMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    R_PG_DMAC_SetAddressOffset_C0( offset );   //アドレスオフセット
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.17 R_PG_DMAC_SetExtendedRepeatSrc_C〈チャネル番号〉

定義 bool R_PG_DMAC_SetExtendedRepeatSrc_C〈チャネル番号〉(uint32_t area)
〈 チャネル番号 〉: 0～3

概要 転送元拡張リピートエリアの設定

生成条件 転送元が拡張リピートエリアに指定された場合

引数

uint32_t area	設定する転送元拡張リピートエリア値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DMAC_C〈チャネル番号〉.c
〈チャネル番号〉: 0～3

使用RPDL関数 R_DMAC_Control

詳細
- 転送元拡張リピートエリアの範囲を設定します
- 有効な値は2^{1}～2^{27}の2のべき乗です

使用例 GUI上で以下の通り設定した場合
- DMAC0の転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定
- 転送元および転送先を拡張リピートエリアに指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
    // DMAC0の設定を変更
    R_PG_DMAC_SetSrcAddress_C0(src_address);   //転送元アドレス
    R_PG_DMAC_SetDestAddress_C0(dest_address);   //転送先アドレス
    R_PG_DMAC_SetTransferCount_C0(tr_count);   //転送カウンタ値
    R_PG_DMAC_SetExtendedRepeatSrc_C0(src_repeat); //転送元拡張リピートエリアサイズ
    R_PG_DMAC_SetExtendedRepeatDest_C0(dest_repeat); //転送先拡張リピートエリアサイズ
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.18 R_PG_DMAC_SetExtendedRepeatDest_C〈チャネル番号〉

定義 bool R_PG_DMAC_SetExtendedRepeatDest_C〈チャネル番号〉(uint32_t area)
〈 チャネル番号 〉: 0～3

概要 転送先拡張リピートエリアの設定

生成条件 転送先が拡張リピートエリアに指定された場合

引数

uint32_t area	設定する転送先拡張リピートエリア値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DMAC_C〈チャネル番号〉.c
〈チャネル番号〉: 0～3

使用RPDL関数 R_DMAC_Control

詳細
- 転送先拡張リピートエリアの範囲を設定します
- 有効な値は2^{1}～2^{27}の2のべき乗です

使用例 GUI上で以下の通り設定した場合
- DMAC0の転送開始要因にIRQ0を指定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定
- 転送元および転送先を拡張リピートエリアに指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();
    //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0の転送を中断
    R_PG_DMAC_Suspend_C0();
    // DMAC0の設定を変更
    R_PG_DMAC_SetSrcAddress_C0(src_address);   //転送元アドレス
    R_PG_DMAC_SetDestAddress_C0(dest_address);   //転送先アドレス
    R_PG_DMAC_SetTransferCount_C0(tr_count);   //転送カウンタ値
    R_PG_DMAC_SetExtendedRepeatSrc_C0( src_repeat );//転送元拡張リピートエリアサイズ
    R_PG_DMAC_SetExtendedRepeatDest_C0(dest_repeat);//転送先拡張リピートエリアサイズ
    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();
}
```

4.6.19 R_PG_DMAC_StopModule_C〈*チャネル番号*〉

定義　bool R_PG_DMAC_StopModule_C〈*チャネル番号*〉(void)

〈 *チャネル番号* 〉: 0～3

概要　DMACチャネルの停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_DMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0～3

使用RPDL関数　R_DMAC_Destroy

詳細

- DMACのチャネルを停止します。
- DMACの全チャネルとDTCが停止している場合、DMACおよびDTCはモジュールストップ状態に移行します。
- 他の周辺機能が転送開始要因として使用されている場合は、本関数を呼ぶ前に転送開始要因を無効にしてください。

使用例　GUI上で以下の通り設定した場合

- DMAC0の転送開始要因をソフトウェアトリガに設定
- DMA0割り込み通知関数名に Dmac0IntFunc を指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //DMAC0を設定する
    R_PG_DMAC_Set_C0();

    //DMAC0の転送を開始する
    R_PG_DMAC_StartTransfer_C0();
}

//DMA割り込み通知関数
void Dmac0IntFunc (void)
{
    //DMAC0を停止
    R_PG_DMAC_StopModule_C0();
}
```

4.7 EXDMAコントローラ（EXDMAC）

4.7.1 R_PG_EXDMAC_Set_C〈*チャネル番号*〉

定義　bool R_PG_EXDMAC_Set_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉： 0, 1

概要　EXDMACの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_EXDMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉： 0, 1

使用RPDL関数　R_EXDMAC_Create

詳細

- EXDMACのモジュールストップ状態を解除して初期設定します。
- 転送開始要因に割り込みを選択した場合は、本関数を呼び出した後R_PG_EXDMAC_Activate_C〈*チャネル番号*〉を呼び出すことにより割り込みの入力待ち状態になります。転送開始要因にソフトウェアトリガを選択した場合は、本関数を呼び出した後R_PG_EXDMAC_StartTransfer_C〈*チャネル番号*〉を呼び出すことにより転送を開始します。
- 転送開始要因に外部入力信号を指定した場合、およびEDACK出力を使用する場合、本関数で使用する端子を設定します。外部入力およびEDACK出力に使用する端子はGUI上の周辺機能別使用端子ウィンドウで設定してください。
- GUI上で割り込み通知関数名を指定した場合、本関数内でEXDMAC割り込みを設定します。CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。　void 〈*割り込み通知関数名*〉(void)
割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上で以下の通り設定した場合

- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();

    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}

//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMACを停止
    R_PG_EXDMAC_StopModule_C0();
}
```

4.7.2 R_PG_EXDMAC_Activate_C〈*チャネル番号*〉

定義　bool R_PG_EXDMAC_Activate_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉： 0, 1

概要　EXDMACを転送開始トリガの入力待ち状態に設定

生成条件　転送開始要因に外部信号またはMTUを選択

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_EXDMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉： 0, 1

使用RPDL関数　R_EXDMAC_Control

詳細
- 転送開始要因を外部信号またはMTUに設定したEXDMACのチャネルをトリガ入力待ち状態に設定します。
- 本関数は転送開始要因に外部信号またはMTUを指定した場合に生成されます。
- あらかじめR_PG_EXDMAC_Set_C〈*チャネル番号*〉によりEXDMACのチャネルを設定してください。

使用例　GUI上で以下の通り設定した場合
- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();

    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}

//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMACを停止
    R_PG_EXDMAC_StopModule_C0();
}
```

4.7.3 R_PG_EXDMAC_StartTransfer_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_StartTransfer_C〈*チャネル番号*〉(void)

〈*チャネル番号*〉： 0, 1

概要 データ転送の開始(ソフトウェアトリガ)

生成条件 転送開始要因にソフトウェアトリガを選択

引数 なし

戻り値

true	転送開始が正しく行われた場合
false	転送開始に失敗した場合

出力先ファイル R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉： 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細

- 転送開始要因をソフトウェアトリガに設定したEXDMACチャネルの転送を開始します。
- 本関数は転送開始要因にソフトウェアトリガを指定した場合に生成されます。
- あらかじめR_PG_EXDMAC_Set_C〈*チャネル番号*〉によりEXDMACのチャネルを設定してください。

使用例 GUI上で以下の通り設定した場合

- ノーマル転送モードでEXDMAC0の転送開始要因をソフトウェアトリガに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

volatile bool transferred;

void func(void)
{
	transferred = false;

	//EXDMAC0を設定する
	R_PG_EXDMAC_Set_C0();

	while( transferred == false ){
		//EXDMAC0の転送を開始する
		R_PG_EXDMAC_StartTransfer_C0();
	}
	// EXDMACを停止
	R_PG_EXDMAC_StopModule_C0();
}

//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
	transferred = true;
}
```

4.7.4 R_PG_EXDMAC_Suspend_C〈*チャネル番号*〉

定義　bool R_PG_EXDMAC_Suspend_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉： 0, 1

概要　データ転送の中断

引数　なし

戻り値

true	中断が正しく行われた場合
false	中断に失敗した場合

出力先ファイル　R_PG_EXDMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉： 0, 1

使用RPDL関数　R_EXDMAC_Control

詳細
- DMA転送を中断します。
- 転送開始要因に外部信号またはMTUを選択した場合、転送を再開するには R_PG_EXDMAC_Activate_C〈*チャネル番号*〉呼び出してください。

使用例　GUI上で以下の通り設定した場合
- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定
- IRQ1の割り込み通知関数名にIrq1ExtIntFuncを指定
- IRQ2の割り込み通知関数名にIrq2ExtIntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_EXDMAC_Set_C0();       //EXDMAC0を設定する
    R_PG_ExtInterrupt_Set_IRQ0();       //IRQ0を設定する
    R_PG_ExtInterrupt_Set_IRQ1();       //IRQ1を設定する
    R_PG_ExtInterrupt_Set_IRQ2();       //IRQ2を設定する
    R_PG_EXDMAC_Activate_C0();      //EXDMAC0を転送開始トリガ入力待ち状態にする
}

//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    R_PG_EXDMAC_StopModule_C0();       //EXDMAC0を停止
}

//IRQ1割り込みでDMA転送停止
void Irq1ExtIntFunc (void)
{
    R_PG_EXDMAC_Suspend_C0();       //EXDMAC0の転送を中断
}

//IRQ2割り込みでDMA転送再開
void Irq2ExtIntFunc (void)
{
    R_PG_EXDMAC_Activate_C0();      //DMA転送有効化
}
```

4.7.5 R_PG_EXDMAC_GetTransferCount_C〈チャネル番号〉

定義

bool R_PG_EXDMAC_GetTransferCount_C〈チャネル番号〉(uint16_t * count)

〈チャネル番号〉： 0, 1

概要

転送カウンタ値の取得

引数

uint16_t * count	転送カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_EXDMAC_C〈チャネル番号〉.c

〈チャネル番号〉： 0, 1

使用RPDL関数

R_EXDMAC_GetStatus

詳細

- 現在の転送カウンタの値を取得します。
- EXDMAC割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。EXDMAC割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前に R_PG_EXDMAC_ClearInterruptFlag_C〈チャネル番号〉を呼び出してください。

使用例

GUI上で以下の通り設定した場合

- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    uint16_t count;

    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();

    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();

    //転送カウンタ値が10未満になるのを待つ
    do{
        R_PG_EXDMAC_GetTransferCount_C0( & count );
    } while( count >= 10 );

    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
}
```

4.7.6 R_PG_EXDMAC_SetTransferCount_C〈*チャネル番号*〉

定義

bool R_PG_EXDMAC_SetTransferCount_C〈*チャネル番号*〉(uint16_t count)

〈 *チャネル番号* 〉: 0, 1

概要

転送カウンタ値の設定

引数

uint16_t count	転送カウンタに設定する値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_EXDMAC_Control

詳細

- 転送カウンタの値を設定します。
- 有効な値はノーマル転送モードでは0～65535 (0:フリーランニングモード) 、リピート転送モード、ブロック転送モードおよびクラスタ転送モードでは0～1023 (0:1024回)です。

使用例

GUI上で以下の通り設定した場合

- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.7 R_PG_EXDMAC_GetRepeatBlockSizeCount_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_GetRepeatBlockSizeCount_C〈*チャネル番号*〉(uint16_t * count)
〈 *チャネル番号* 〉: 0, 1

概要 リピート/ブロック/クラスタサイズカウンタ値の取得

生成条件 転送モードにリピート転送モード、ブロック転送モードまたはクラスタ転送モードを選択

引数

uint16_t * count	カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_EXDMAC_C 〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_GetStatus

詳細
- リピート/ブロック/クラスタサイズカウンタの現在の値を取得します。
- EXDMAC割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。EXDMAC割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前にR_PG_EXDMAC_ClearInterruptFlag_C〈*チャネル番号*〉を呼び出してください。

使用例 GUI上で以下の通り設定した場合
- リピート転送モードでEXDMAC0を設定
- 転送開始要因はEDREQ0信号またはMTU

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    uint16_t count;

    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();

    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();

    //リピートサイズカウンタ値が10未満になるのを待つ
    do{
        R_PG_EXDMAC_GetRepeatBlockSizeCount_C0( & count );
    } while( count >= 10 );

    //転送を中断
    R_PG_EXDMAC_Suspend_C0();
}
```

4.7.8 R_PG_EXDMAC_SetRepeatBlockSizeCount_C〈*チャネル番号*〉

定義

bool R_PG_EXDMAC_SetRepeatBlockSizeCount_C〈*チャネル番号*〉(uint16_t count)

〈 *チャネル番号* 〉: 0, 1

概要

リピート/ブロック/クラスタサイズカウンタ値の設定

生成条件

転送モードにリピート転送モード、ブロック転送モードまたはクラスタ転送モードを選択

引数

uint16_t count	リピート/ブロック/クラスタサイズカウンタに設定する値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_EXDMAC_C 〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_EXDMAC_Control

詳細

- リピート/ブロック/クラスタサイズカウンタの現在の値を設定します。
- 有効な値はリピート転送モードおよびブロック転送モードでは1～1023、クラスタ転送モードでは1～7です。

使用例

GUI上で以下の通り設定した場合

- リピート転送モードでEXDMAC0を設定
- 転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();

    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    R_PG_EXDMAC_SetRepeatBlockSizeCount_C0( repeat_count );   //リピートサイズカウンタ値
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.9 R_PG_EXDMAC_ClearInterruptFlag_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_ClearInterruptFlag_C〈*チャネル番号*〉(bool* int_request)
〈 *チャネル番号* 〉: 0, 1

概要 割り込み要求フラグの取得とクリア

生成条件 EXDMAC割り込み有効時

引数

bool* int_request	割り込み要求フラグの格納先

戻り値

true	取得とクリアに成功した場合
false	取得とクリアに失敗した場合

出力先ファイル R_PG_EXDMAC_C 〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_GetStatus

詳細
- EXDMAC割り込み要求フラグ(IRフラグ)を取得し、クリアします。

使用例 GUI上で以下の通り設定した場合
- ノーマル転送モードでEXDMAC0を設定
- 転送開始要因はEDREQ0信号またはMTU
- EXDMAC割り込み有効
- EXDMAC割り込み優先レベルは0

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool int_request;

	//EXDMAC0を設定する
	R_PG_EXDMAC_Set_C0();

	//EXDMAC0を転送開始トリガ入力待ち状態にする
	R_PG_EXDMAC_Activate_C0();

	//IRフラグが1になるのを待つ
	do{
		R_PG_EXDMAC_ClearInterruptFlag_C0( & int_request );
	} while( int_request == false );
}
```

4.7.10 R_PG_EXDMAC_GetTransferEndFlag_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_GetTransferEndFlag_C〈*チャネル番号*〉(bool* end)

〈 *チャネル番号* 〉: 0, 1

概要 転送終了フラグの取得

引数

bool* end	転送終了フラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_EXDMAC_C 〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_GetStatus

詳細

- 転送終了フラグを取得します。
- EXDMAC割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。EXDMAC割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前にR_PG_EXDMAC_ClearInterruptFlag_C〈*チャネル番号*〉を呼び出してください。
- 転送終了フラグは本関数内でクリアされません。転送終了フラグをクリアする必要がある場合はR_PG_EXDMAC_ClearTransferEndFlag_C〈*チャネル番号*〉を呼び出してください。

使用例 GUI上で以下の通り設定した場合

- ノーマル転送モードでEXDMAC0を設定
- 転送開始要因はEDREQ0信号またはMTU
- EXDMAC割り込み無効

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool end;

	//EXDMAC0を設定する
	R_PG_EXDMAC_Set_C0();

	//EXDMAC0を転送開始トリガ入力待ち状態にする
	R_PG_EXDMAC_Activate_C0();

	//転送終了フラグが1になるのを待つ
	do{
		R_PG_EXDMAC_GetTransferEndFlag_C0( & end );
	} while( end == false );

	//転送終了フラグをクリアする
	R_PG_EXDMAC_ClearTransferEndFlag_C0();
}
```

4.7.11 R_PG_EXDMAC_ClearTransferEndFlag_C〈チャネル番号〉

定義 bool R_PG_EXDMAC_ClearTransferEndFlag_C〈チャネル番号〉(void)
〈 チャネル番号 〉: 0, 1

概要 転送終了フラグのクリア

引数 なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル R_PG_EXDMAC_C 〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細
- 転送終了フラグをクリアします。
- 転送終了フラグを取得するにはPG_EXDMAC_GetTransferEndFlag_C〈チャネル番号〉を呼び出してください。

使用例 GUI上で以下の通り設定した場合
- ノーマル転送モードでEXDMAC0を設定
- 転送開始要因はEDREQ0信号またはMTU
- EXDMAC割り込み無効

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool end;

	//EXDMAC0を設定する
	R_PG_EXDMAC_Set_C0();

	//EXDMAC0を転送開始トリガ入力待ち状態にする
	R_PG_EXDMAC_Activate_C0();

	//転送終了フラグが1になるのを待つ
	do{
		R_PG_EXDMAC_GetTransferEndFlag_C0( & end );
	} while( end == false );

	//転送終了フラグをクリアする
	R_PG_EXDMAC_ClearTransferEndFlag_C0();
}
```

4.7.12 R_PG_EXDMAC_GetTransferEscapeEndFlag_C〈チャネル番号〉

定義 bool R_PG_EXDMAC_GetTransferEscapeEndFlag_C〈チャネル番号〉(bool* end)
〈 チャネル番号 〉: 0, 1

概要 転送エスケープ終了フラグの取得

生成条件 割り込み出力要因に[リピート/ブロック/クラスタサイズの転送終了]、[転送元アドレスの拡張リピートエリアオーバフロー]または[転送先アドレスの拡張リピートエリアオーバフロー]が選択された場合

引数

bool* end	転送エスケープ終了フラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_EXDMAC_C 〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_EXDMAC_GetStatus

詳細

- 転送エスケープ終了フラグ(EDMSTS.ESIF)を取得します。
- EXDMAC割り込み要求フラグ(IRフラグ)は本関数内でクリアされます。EXDMAC割り込み要求フラグを取得する必要がある場合は、本関数を呼び出す前にR_PG_EXDMAC_ClearInterruptFlag_C〈チャネル番号〉を呼び出してください。
- 転送エスケープ終了フラグは本関数内でクリアされません。転送エスケープ終了フラグをクリアする必要がある場合はR_PG_EXDMAC_ClearTransferEscapeEndFlag_C〈チャネル番号〉を呼び出してください。
- 外部信号またはMTUを転送開始要因に使用している場合、転送エスケープ終了時後に転送を継続するには、R_PG_EXDMAC_Activate_C〈チャネル番号〉を呼び出してください。R_PG_EXDMAC_Activate_C〈チャネル番号〉内で転送エスケープ終了フラグはクリアされ、EXDMACチャネルはトリガ入力待ち状態になります。
- 転送開始要因にソフトウェアトリガを選択した場合、転送エスケープ終了時後に転送を継続するには、R_PG_EXDMAC_StartTransfer_C〈チャネル番号〉を呼び出してください。R_PG_EXDMAC_StartTransfer_C〈チャネル番号〉内で転送エスケープ終了フラグはクリアされ、転送を開始します。

使用例

GUI上で以下の通り設定した場合

- リピート転送モードでEXDMAC0を設定
- 転送開始要因は外部信号またはMTU
- 割り込み出力要因に[リピート/ブロック/クラスタサイズの転送終了]および[転送終了]を指定
- EXDMAC割り込み通知関数名にExdmac0IntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void Exdmac0IntFunc(void)
{
	bool transfer_end;

	R_PG_EXDMAC_GetTransferEndFlag_C0( & transfer_end );
	if( transfer_end ){
		//転送終了
		R_PG_EXDMAC_StopModule_C0();
	}
	else{
		//転送エスケープ終了(リピートサイズの終了)
		//転送エスケープ終了フラグをクリアし転送開始トリガ入力待ちにする
		R_PG_DMAC_Activate_C0();
	}
}

void func(void)
{
	// EXDMAC0を設定する
	R_PG_EXDMAC_Set_C0();

	//EXDMAC0を転送開始トリガ入力待ち状態にする
	R_PG_EXDMAC_Activate_C0();
}
```

4.7.13 R_PG_EXDMAC_ClearTransferEscapeEndFlag_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_ClearTransferEscapeEndFlag_C〈*チャネル番号*〉(void)
〈 *チャネル番号* 〉: 0, 1

概要 転送エスケープ終了フラグのクリア

生成条件 割り込み出力要因に[リピート/ブロック/クラスタサイズの転送終了]、[転送元アドレスの拡張リピートエリアオーバフロー]または[転送先アドレスの拡張リピートエリアオーバフロー]が選択された場合

引数 なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル R_PG_EXDMAC_C 〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細
- 転送エスケープ終了フラグをクリアします。
- 転送エスケープ終了フラグを取得するにはPG_EXDMAC_GetTransferEscapeEndFlag_C〈*チャネル番号*〉を呼び出してください。

使用例 GUI上で以下の通り設定した場合
- リピート転送モードでEXDMAC0を設定
- 転送開始要因はEDREQ0信号またはMTU
- 割り込み出力要因に[リピート/ブロック/クラスタサイズの転送終了]を指定
- EXDMAC割り込み優先レベルは0

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	bool end;

	//EXDMAC0を設定する
	R_PG_EXDMAC_Set_C0();

	//EXDMAC0を転送開始トリガ入力待ち状態にする
	R_PG_EXDMAC_Activate_C0();

	//転送エスケープ終了フラグが1になるのを待つ
	do{
		R_PG_EXDMAC_GetTransferEscapeEndFlag_C0( & end );
	} while( end == false );

	//転送エスケープ終了フラグをクリアする
	R_PG_EXDMAC_ClearTransferEscapeEndFlag_C0();
}
```

4.7.14 R_PG_EXDMAC_SetSrcAddress_C〈*チャネル番号*〉

定義

bool R_PG_EXDMAC_SetSrcAddress_C〈*チャネル番号*〉(void * src_addr)

〈 *チャネル番号* 〉: 0, 1

概要

転送元アドレスの設定

引数

void * src_addr	設定する転送元アドレス

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_EXDMAC_Control

詳細

- 転送元アドレスを設定します

使用例

GUI上で以下の通り設定した場合

- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.15 R_PG_EXDMAC_SetDestAddress_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_SetDestAddress_C〈*チャネル番号*〉(void * dest_addr)

〈 *チャネル番号* 〉: 0, 1

概要 転送先アドレスの設定

引数

void * dest_addr	設定する転送先アドレス

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細

- 転送先アドレスを設定します

使用例

GUI上で以下の通り設定した場合

- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.16 R_PG_EXDMAC_SetAddressOffset_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_SetAddressOffset_C〈*チャネル番号*〉(int32_t offset)

〈 *チャネル番号* 〉: 0, 1

概要 アドレスオフセット値の設定

生成条件 転送元アドレス更新モードまたは転送先アドレス更新オードにオフセット加算が選択された場合

引数

int32_t offset	設定するオフセット値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細

- アドレスオフセット加算値を設定します
- 有効な値は +FFFFFFFh ～ −1000000h です。

使用例 GUI上で以下の通り設定した場合

- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定
- 転送元アドレス更新モードまたは転送先アドレス更新オードにオフセット加算を選択

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetSrcAddress_C0( src_address );   //転送元アドレス
    R_PG_EXDMAC_SetDestAddress_C0( dest_address );   //転送先アドレス
    R_PG_EXDMAC_SetTransferCount_C0( tr_count );   //転送カウンタ値
    R_PG_EXDMAC_SetAddressOffset_C0( offset );   //アドレスオフセット
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.17 R_PG_EXDMAC_SetExtendedRepeatSrc_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_SetExtendedRepeatSrc_C〈*チャネル番号*〉(uint32_t area)

〈 *チャネル番号* 〉: 0, 1

概要 転送元拡張リピートエリアの設定

生成条件 転送元が拡張リピートエリアに指定された場合

引数

uint32_t area	設定する転送元拡張リピートエリア値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細
- 転送元拡張リピートエリアの範囲を設定します
- 有効な値は2^{1}～2^{27}の2のべき乗です

使用例 GUI上で以下の通り設定した場合
- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定
- 転送元および転送先を拡張リピートエリアに指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetSrcAddress_C0(src_address);   //転送元アドレス
    R_PG_EXDMAC_SetDestAddress_C0(dest_address);   //転送先アドレス
    R_PG_EXDMAC_SetTransferCount_C0(tr_count);   //転送カウンタ値
    R_PG_EXDMAC_SetExtendedRepeatSrc_C0(src_repeat); //転送元拡張リピートエリアサイズ
    R_PG_EXDMAC_SetExtendedRepeatDest_C0(dest_repeat); //転送先拡張リピートエリアサイズ
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.18 R_PG_EXDMAC_SetExtendedRepeatDest_C〈*チャネル番号*〉

定義 bool R_PG_EXDMAC_SetExtendedRepeatDest_C〈*チャネル番号*〉(uint32_t area)
〈 *チャネル番号* 〉: 0, 1

概要 転送先拡張リピートエリアの設定

生成条件 転送先が拡張リピートエリアに指定された場合

引数

uint32_t area	設定する転送先拡張リピートエリア値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_EXDMAC_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_EXDMAC_Control

詳細
- 転送先拡張リピートエリアの範囲を設定します
- 有効な値は2^{1}～2^{27}の2のべき乗です

使用例 GUI上で以下の通り設定した場合
- EXDMAC0の転送開始要因をEDREQ0信号またはMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定
- 転送元および転送先を拡張リピートエリアに指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //EXDMAC0の転送を中断
    R_PG_EXDMAC_Suspend_C0();
    // EXDMAC0の設定を変更
    R_PG_EXDMAC_SetSrcAddress_C0(src_address);   //転送元アドレス
    R_PG_EXDMAC_SetDestAddress_C0(dest_address);   //転送先アドレス
    R_PG_EXDMAC_SetTransferCount_C0(tr_count);   //転送カウンタ値
    R_PG_EXDMAC_SetExtendedRepeatSrc_C0( src_repeat );//転送元拡張リピートエリアサイズ
    R_PG_EXDMAC_SetExtendedRepeatDest_C0(dest_repeat);//転送先拡張リピートエリアサイズ
    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}
```

4.7.19 R_PG_EXDMAC_StopModule_C〈*チャネル番号*〉

定義　bool R_PG_EXDMAC_StopModule_C〈*チャネル番号*〉(void)

〈 *チャネル番号* 〉: 0, 1

概要　EXDMACチャネルの停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_EXDMAC_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_EXDMAC_Destroy

詳細
- EXDMACのチャネルを停止します。
- EXDMACの全チャネルが停止している場合、EXDMACはモジュールストップ状態に移行します。
- MTUが転送開始要因として使用されている場合は、本関数を呼ぶ前に転送開始要因を無効にしてください。

使用例　GUI上で以下の通り設定した場合
- EXDMAC0の転送開始要因をMTUに設定
- EXDMAC0割り込み通知関数名に Exdmac0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //EXDMAC0を設定する
    R_PG_EXDMAC_Set_C0();

    //MTUを設定しカウント動作を開始
    R_PG_Timer_Set_MTU_U0_C1();
    R_PG_Timer_StartCount_MTU_U0_C1();

    //EXDMAC0を転送開始トリガ入力待ち状態にする
    R_PG_EXDMAC_Activate_C0();
}

//EXDMAC割り込み通知関数
void Exdmac0IntFunc(void)
{
    //MTUを停止
    R_PG_Timer_StopModule_MTU_U0();

    //EXDMAC0を停止
    R_PG_EXDMAC_StopModule_C0();
}
```

4.8 データトランスファコントローラ（DTCa）

4.8.1 R_PG_DTC_Set

定義　bool R_PG_DTC_Set (void)

概要　DTCの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Dtc.c

使用RPDL関数　R_DTC_Set

詳細

- 転送情報リードスキップ、アドレスモードおよびDTCベクタテーブルのベースアドレスを設定します。

使用例

GUI上で以下の通り設定した場合

- DTCベクタテーブルのアドレスを15000hに設定
- 転送開始要因をIRQ8に指定したDTC転送を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//DTCベクタテーブル
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//DTCの初期設定
void func(void)
{
    //データトランスファコントローラの基本設定
    R_PG_DTC_Set();

    //転送開始要因をIRQ8に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ8();

    //DTCを転送開始トリガ入力待ち状態にする
    R_PG_DTC_Activate_C0();

    //IRQ8の設定
    R_PG_ExtInterrupt_Set_IRQ8();
}
```

4.8.2 R_PG_DTC_Set_〈*転送開始要因*〉

定義

bool R_PG_DTC_Set_〈*転送開始要因*〉(void)

〈*転送開始要因*〉 :
SWINT, CMT0～3, D0FIFO0～1, D1FIFO0～1, SPRI0～1, SPTI0～1, IRQ0～15, ADI0～1, S12ADI0, TGI0A～10D, TGIU5～TGIW11, CMIA0～B3, DMACIA0～3, EXDMACI0～1, RXI0～6, TXI0～6, ICRXI0～1, ICTXI0～1

概要

DTC転送情報の設定

引数

なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_Dtc.c

使用RPDL関数

R_DTC_Create

詳細

- 起動要因によりトリガされる転送情報を指定されたアドレスに保存し、転送情報のアドレスをDTCベクタテーブルに設定します。
- 起動要因によりトリガされるチェイン転送の情報も保存されます。
- 指定されたアドレスに既に転送情報が保存されている場合は上書きされます。
- 本関数では起動要因として使用する割り込みの設定を行いません。起動要因として使用する割り込みは各周辺機能の関数で設定してください。
  起動要因として使用する割り込みは、割り込み要求先をDTCに指定してください。

使用例

GUI上で以下の通り設定した場合

- DTCベクタテーブルのアドレスを15000hに設定
- 転送開始要因をIRQ8に指定したDTC転送を設定
- 転送開始要因をIRQ9に指定したDTC転送を設定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

//DTCベクタテーブル
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//DTCの初期設定
void func(void)
{
    //データトランスファコントローラの基本設定
    R_PG_DTC_Set();

    //転送開始要因をIRQ8に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ8();

    //転送開始要因をIRQ9に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ9();

    //DTCを転送開始トリガ入力待ち状態にする
    R_PG_DTC_Activate_C0();

    //IRQ8,IRQ9の設定
    R_PG_ExtInterrupt_Set_IRQ8();
    R_PG_ExtInterrupt_Set_IRQ9();
}
```

4.8.3 R_PG_DTC_Activate

定義 bool R_PG_DTC_Activate (void)

概要 DTCを転送開始トリガの入力待ち状態に設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_Dtc.c

使用RPDL関数 R_DTC_Control

詳細

- DTCを転送開始トリガの入力待ち状態に設定します。
- あらかじめR_PG_DTC_SetによりDTCを設定し、R_PG_DTC_Set<*転送開始要因*>により転送情報を保存してください。

使用例

GUI上で以下の通り設定した場合

- DTCベクタテーブルのアドレスを15000hに設定
- 転送開始要因をIRQ8に指定したDTC転送を設定
- 割り込みの発生条件に[指定されたデータ転送終了時、CPU割込みが発生]を指定
- チェイン転送無効
- IRQ8の割り込み通知関数名に Irq8IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//DTCベクタテーブル
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//DTCの初期設定
void func(void)
{
    //データトランスファコントローラの基本設定
    R_PG_DTC_Set();

    //転送開始要因をIRQ8に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ8();

    //DTCを転送開始トリガ入力待ち状態にする
    R_PG_DTC_Activate_C0();
}

//IRQ8の割り込み通知関数（指定した回数のDTC転送終了時に割り込み発生）
void Irq8IntFunc(void)
{
    //IRQ8の停止
    //(指定した回数の転送終了後もトリガ入力で転送が継続し、
    //転送カウンタはインクリメントします。転送を終了するには
    //起動要因の割り込みを無効にしてください。)
    R_PG_ExtInterrupt_Disable_IRQ8();
}
```

4.8.4 R_PG_DTC_SuspendTransfer

定義 bool R_PG_DTC_SuspendTransfer (void)

概要 DTC転送の停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_Dtc.c

使用RPDL関数 R_DTC_Control

詳細

- DTC転送を停止します。
- 転送動作中に停止した場合、受付済みの転送要求は処理が終わるまで動作します。
- DTC転送を再開するにはR_DTC_Activateを呼び出してください。

使用例

GUI上で以下の通り設定した場合

- DTCベクタテーブルのアドレスを15000hに設定
- 転送開始要因をIRQ8に指定したDTC転送を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//DTCベクタテーブル
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//DTCの初期設定
void func1(void)
{
    //データトランスファコントローラの基本設定
    R_PG_DTC_Set();

    //転送開始要因をIRQ8に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ8();

    //DTCを転送開始トリガ入力待ち状態にする
    R_PG_DTC_Activate_C0();
}

//DTC転送の中断
void func2(void)
{
    R_PG_DTC_SuspendTransfer();
}

//DTC転送の再開
void func3(void)
{
    R_PG_DTC_Activate_C0();
}
```

4.8.5 R_PG_DTC_GetTransmitStatus

定義 bool R_PG_DTC_GetTransmitStatus (uint8_t * vector, bool * active)

概要 DTC転送状態の取得

引数

uint8_t * vector	転送動作中の場合、現在の転送の起動要因のベクタ番号 (*activeが1の場合に有効化値が格納されます)
bool * active	現在の転送状態 (0:転送動作なし 1:転送動作中)

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_Dtc.c

使用RPDL関数 R_DTC_GetStatus

詳細

- DTCアクティブフラグとDTCアクティブベクタ番号を取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。
  取得しない項目には0を指定してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t vector;
bool active;

void func(void)
{
    //DTC転送状態の取得
    R_PG_DTC_GetTransmitStatus ( &vector, &active);

    if(active){
        switch( vector ){
            case 72:
                //ベクタ番号72の割込みによる転送中の処理
                break;
            case 73:
                //ベクタ番号73の割込みによる転送中の処理
                break;
            default:
        }
    }
}
```

4.8.6 R_PG_DTC_StopModule

定義 bool R_PG_DTC_StopModule (void)

概要 DTCの停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_Dtc.c

使用RPDL関数 R_DTC_Destroy

詳細

- DTCを停止し、モジュールストップ状態に移行します。
- あらかじめ各周辺機能の関数によりDTCのトリガ要因として使用した割り込みを無効にしてください。
- 本関数を呼び出すとDMACもモジュールストップ状態に移行します。DTCのみを停止させる場合はR_PG_DTC_SuspendTransferを使用してください。

使用例 GUI上で以下の通り設定した場合

- DTCベクタテーブルのアドレスを15000hに設定
- 転送開始要因をIRQ8に指定したDTC転送を設定
- 転送開始要因をIRQ9に指定したDTC転送を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//DTCベクタテーブル
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//DTCの初期設定
void func1(void)
{
    //データトランスファコントローラの基本設定
    R_PG_DTC_Set();

    //転送開始要因をIRQ8に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ8();

    //転送開始要因をIRQ9に指定したDTC転送の設定
    R_PG_DTC_Set_IRQ9();

    //DTCを転送開始トリガ入力待ち状態にする
    R_PG_DTC_Activate_C0();

    //IRQ8,IRQ9の設定
    R_PG_ExtInterrupt_Set_IRQ8();
    R_PG_ExtInterrupt_Set_IRQ9();
}

//DTCの停止
void func2(void)
{
    //IRQ8,IRQ9の停止
    R_PG_ExtInterrupt_Disable_IRQ8();
    R_PG_ExtInterrupt_Disable_IRQ9();
    //DTCの停止
    R_PG_DTC_StopModule();
}
```

4.9 I/Oポート

4.9.1 R_PG_IO_PORT_Set_P〈*ポート番号*〉

定義　bool R_PG_IO_PORT_Set_P〈*ポート番号*〉(void)
〈*ポート番号*〉 ： 0～9、A～G

概要　I/Oポートの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_IO_PORT_P〈*ポート番号*〉.c
〈*ポート番号*〉 ： 0～9、A～G

使用RPDL関数　R_IO_PORT_Set

詳細
- GUI上で[I/Oポートとして使用]にチェックされた端子の入出力方向、入力バッファ、プルアップ、オープンドレイン出力の設定を行います。
- [I/Oポートとして使用]がチェックされたポート内の全端子を一括して設定します。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //P0を設定する
    R_PG_IO_PORT_Set_P0();
}
```

4.9.2 R_PG_IO_PORT_Set_P〈*ポート番号*〉〈*端子番号*〉

定義　bool R_PG_IO_PORT_Set_P〈*ポート番号*〉〈*端子番号*〉(void)
〈*ポート番号*〉 ： 0～9、A～G
〈*端子番号*〉 ： 0～7

概要　I/Oポート(1端子)の設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_IO_PORT_P〈*ポート番号*〉.c
〈*ポート番号*〉 ： 0～9、A～G

使用RPDL関数　R_IO_PORT_Set

詳細
- GUI上で[I/Oポートとして使用]にチェックされた端子の入出力方向、入力バッファ、プルアップ、オープンドレイン出力の設定を行います。
- 1端子のみ設定します。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //P00を設定する
    R_PG_IO_PORT_Set_P00();

    //P01を設定する
    R_PG_IO_PORT_Set_P01();

    //P02を設定する
    R_PG_IO_PORT_Set_P02();
}
```

4.9.3 R_PG_IO_PORT_Read_P〈*ポート番号*〉

定義　bool R_PG_IO_PORT_Read_P〈*ポート番号*〉(uint8_t * data)
〈*ポート番号*〉： 0～9、A～G

概要　I/Oポートレジスタからの読み出し

引数

uint8_t * data	読み出した端子状態の格納先

戻り値

true	読み出しに成功した場合
false	読み出しに失敗した場合

出力先ファイル　R_PG_IO_PORT_P〈*ポート番号*〉.c
〈*ポート番号*〉： 0～9、A～G

使用RPDL関数　R_IO_PORT_Read

詳細
- I/Oポートレジスタを読み出し、端子の状態を取得します。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    uint8_t data;

    //P0端子状態を取得する
    R_PG_IO_PORT_Read_P0( &data );
}
```

4.9.4 R_PG_IO_PORT_Read_P〈*ポート番号*〉〈*端子番号*〉

定義

bool R_PG_IO_PORT_Read_P〈*ポート番号*〉〈*端子番号*〉(uint8_t * data)

〈*ポート番号*〉 : 0～9、A～G

〈*端子番号*〉 : 0～7

概要

I/Oポートレジスタからのビット読み出し

引数

uint8_t * data	読み出した端子状態の格納先

戻り値

true	読み出しに成功した場合
false	読み出しに失敗した場合

出力先ファイル

R_PG_IO_PORT_P〈*ポート番号*〉.c

(〈*ポート番号*〉 : 0～9、A～G)

使用RPDL関数

R_IO_PORT_Read

詳細

- I/Oポートレジスタを読み出し、1端子の状態を取得します。
- 値は*dataの下位1ビットに格納されます。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    uint8_t data_p00, data_p01, data_p02;

    //P00端子状態を取得する
    R_PG_IO_PORT_Read_P00( & data_p00);

    //P01端子状態を取得する
    R_PG_IO_PORT_Read_P01( & data_p01);

    //P02端子状態を取得する
    R_PG_IO_PORT_Read_P02( & data_p02);
}
```

4.9.5 R_PG_IO_PORT_Write_P〈*ポート番号*〉

定義

bool R_PG_IO_PORT_Write_P〈*ポート番号*〉(uint8_t data)

〈*ポート番号*〉 : 0～9、A～G

概要

I/Oポートデータレジスタへの書き込み

引数

uint8_t data	書き込む値

戻り値

true	書き込みに成功した場合
false	書き込みに失敗した場合

出力先ファイル

R_PG_IO_PORT_P〈*ポート番号*〉.c

〈*ポート番号*〉 : 0～9、A～G

使用RPDL関数

R_IO_PORT_Write

詳細

- I/Oポートデータレジスタに値を書き込みます。レジスタに書き込んだ値が出力ポートから出力されます。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //P0を設定する
    R_PG_IO_PORT_Set_P0();

    //P0から0x03を出力する
    R_PG_IO_PORT_Write_P0( 0x03 );
}
```

4.9.6 R_PG_IO_PORT_Write_P〈*ポート番号*〉〈*端子番号*〉

定義

bool R_PG_IO_PORT_Write_P〈*ポート番号*〉〈*端子番号*〉(uint8_t data)

〈*ポート番号*〉 : 0～9、A～G

〈*端子番号*〉 : 0～7

概要

I/Oポートデータレジスタへのビット書き込み

引数

uint8_t data	書き込む値

戻り値

true	書き込みに成功した場合
false	書き込みに失敗した場合

出力先ファイル

R_PG_IO_PORT_P〈*ポート番号*〉.c

〈*ポート番号*〉 : 0～9、A～G

使用RPDL関数

R_IO_PORT_Write

詳細

- I/Oポートデータレジスタに値を書き込みます。レジスタに書き込んだ値が出力ポートから出力されます。値はdataの下位1ビットに格納してください。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //P00を設定する
    R_PG_IO_PORT_Set_P00();

    //P01を設定する
    R_PG_IO_PORT_Set_P01();

    //P00からLを出力する
    R_PG_IO_PORT_Write_P00( 0x00 );

    //P01からHを出力する
    R_PG_IO_PORT_Write_P01( 0x01 );
}
```

4.10 マルチファンクションタイマパルスユニット 2 (MTU2)

4.10.1 R_PG_Timer_Set_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉

定義　bool R_PG_Timer_Set_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉(void)
〈*ユニット番号*〉: 0, 1
〈*チャネル番号*〉: 0～11

概要　MTUの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Timer_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉.c
〈*ユニット番号*〉: 0, 1
〈*チャネル番号*〉: 0～11

使用RPDL関数　R_MTU2_Set, R_MTU2_Create

詳細
- MTUのモジュールストップ状態を解除して初期設定します。
- 本関数内でMTUの割り込みを設定します。GUI上で割り込み通知関数名を指定した場合、CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。　void 〈*割り込み通知関数名*〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- GUI上で割り込み優先レベルを0に設定した場合、CPU割り込みは発生しません。割り込み要求フラグは R_PG_Timer_GetRequestFlag_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉により取得することができます。
- 外部入力カウントクロック、外部リセット信号、インプットキャプチャ、パルス出力を使用する場合、本関数内で使用する端子の入出力方向と入力バッファを設定します。
- R_PG_Timer_StartCount_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉(_〈*相*〉)によりカウント動作を開始します。
- 相補PWMモードおよびリセット同期PWMモードは本バージョンではサポートされません。

使用例　GUI上で以下の通り設定した場合
- MTUユニット1チャネル6を設定
- コンペアマッチA割り込み通知関数名にMtu6IcCmAIntFuncを指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Timer_Set_MTU_U1_C6();      // MTU6の設定
    R_PG_Timer_StartCount_MTU_U1_C6();     // カウント動作開始
}

void Mtu6IcCmAIntFunc(void)
{
    //コンペアマッチA割り込み発生時処理
}
```

4.10.2 R_PG_Timer_StartCount_MTU_U*〈ユニット番号〉*_C*〈チャネル番号〉*(_*〈相〉*)

定義

bool R_PG_Timer_StartCount_MTU_U*〈ユニット番号〉*_C*〈チャネル番号〉*(void)

〈ユニット番号〉: 0, 1

〈チャネル番号〉: 0～4, 6～10

bool R_PG_Timer_StartCount_MTU_U*〈ユニット番号〉*_C*〈チャネル番号〉*_*〈相〉*(void)

〈ユニット番号〉: 0, 1

〈チャネル番号〉: 5, 11

〈相〉: U, V, W

概要

MTUのカウント動作開始

引数

なし

戻り値

true	カウント動作の開始が正しく行われた場合
false	カウント動作の開始に失敗した場合

出力先ファイル

R_PG_Timer_MTU_U*〈ユニット番号〉*_C*〈チャネル番号〉*.c

〈ユニット番号〉: 0, 1

〈チャネル番号〉: 0～11

使用RPDL関数

R_MTU2_ControlChannel

詳細

- MTUのカウント動作を開始します。
- あらかじめR_PG_Timer_Set_MTU_U*〈ユニット番号〉*_C*〈チャネル番号〉* によりMTUを初期設定してください。

使用例

GUI上で以下の通り設定した場合

- MTUユニット0チャネル1を設定
- コンペアマッチA割り込み通知関数名にMtu1IcCmAIntFuncを指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();   //MTU1の設定
    R_PG_Timer_StartCount_MTU_U0_C1();   //カウント動作開始
}

void Mtu1IcCmAIntFunc(void)
{
    R_PG_Timer_HaltCount_MTU_U0_C1();   //カウント動作停止

    //コンペアマッチA割り込み発生時処理

    R_PG_Timer_StartCount_MTU_U0_C1();   //カウント動作再開
}
```

4.10.3 R_PG_Timer_HaltCount_MTU_U〈ユニット番号〉_C〈チャネル番号〉_〈相〉

定義 bool R_PG_Timer_HaltCount_MTU_U〈ユニット番号〉_C〈チャネル番号〉(void)

〈ユニット番号〉: 0, 1

〈チャネル番号〉: 0～4, 6～10

bool R_PG_Timer_HaltCount_MTU_U〈ユニット番号〉_C〈チャネル番号〉_〈相〉(void)

〈ユニット番号〉: 0, 1

〈チャネル番号〉: 5, 11

〈相〉: U, V, W

概要 MTUのカウント動作を一時停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_Timer_MTU_U〈ユニット番号〉_C〈チャネル番号〉.c

〈ユニット番号〉: 0, 1

〈チャネル番号〉: 0～11

使用RPDL関数 R_MTU2_ControlChannel

詳細

- MTUのカウント動作を一時停止します。
- カウント動作を再開するには以下の関数を呼び出してください。

  MTU0～MTU4、MTU6～MTU10

  R_PG_Timer_StartCount_MTU_U〈ユニット番号〉_C〈チャネル番号〉

  MTU5、MTU11

  R_PG_Timer_StartCount_MTU_U〈ユニット番号〉_C〈チャネル番号〉_〈相〉

使用例 GUI上で以下の通り設定した場合

- MTUユニット0チャネル1を設定
- コンペアマッチA割り込み通知関数名にMtu1IcCmAIntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
      R_PG_Timer_Set_MTU_U0_C1();       // MTU1の設定
      R_PG_Timer_StartCount_MTU_U0_C1();       //  カウント動作開始
}

void Mtu1IcCmAIntFunc(void)
{
      R_PG_Timer_HaltCount_MTU_U0_C1();       //カウント動作停止

      //コンペアマッチA割り込み発生時処理

      R_PG_Timer_StartCount_MTU_U0_C1();       //カウント動作再開
}
```

4.10.4 R_PG_Timer_GetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉

定義

bool R_PG_Timer_GetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t * counter_val)
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～4, 6～10

bool R_PG_Timer_GetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t * counter_u_val, uint16_t * counter_v_val, uint16_t * counter_w_val)
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 5, 11

概要

MTUのカウンタ値を取得

引数

MTU0～MTU4、MTU6～MTU10

uint16_t * counter_val	カウンタ値の格納先

MTU5、MTU11

uint16_t * counter_u_val	カウンタU値の格納先
uint16_t * counter_v_val	カウンタV値の格納先
uint16_t * counter_w_val	カウンタW値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_Timer_MTU_U〈ユニット番号〉_C〈チャネル番号〉.c
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～11

使用RPDL関数

R_MTU2_ReadChannel

詳細

- MTUのカウンタ値を取得します。

使用例

GUI上で以下の通り設定した場合
- MTUユニット0チャネル0を設定
- TGRAをインプットキャプチャレジスタに設定し、インプットキャプチャA割り込みを有効に設定
- インプットキャプチャA割り込み通知関数名にMtu0IcCmAIntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t counter_val;

void func(void)
{
      R_PG_Timer_Set_MTU_U0_C0();       // MTU0の設定
      R_PG_Timer_StartCount_MTU_U0_C0();       //  カウント動作開始
}

void Mtu0IcCmAIntFunc(void)
{
      //MTUのカウンタ値を取得
      R_PG_Timer_GetCounterValue_MTU_U0_C0( & counter_val );
}
```

4.10.5 R_PG_Timer_SetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉

定義

bool R_PG_Timer_SetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t counter_val)
〈ユニット番号〉: 0, 1 〈チャネル番号〉: 0～4, 6～10

bool R_PG_Timer_SetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉_〈相〉
(uint16_t counter_val)
〈ユニット番号〉: 0, 1 〈チャネル番号〉: 5, 11 〈相〉: U, V, W

bool R_PG_Timer_SetCounterValue_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t counter_u_val, uint16_t counter_v_val, uint16_t counter_w_val)
〈ユニット番号〉: 0, 1 〈チャネル番号〉: 5, 11

概要

MTUのカウンタ値を設定

引数

MTU0～MTU11

uint16_t counter_val	カウンタに設定する値

MTU5、MTU11

uint16_t counter_u_val	カウンタUに設定する値
uint16_t counter_v_val	カウンタVに設定する値
uint16_t counter_w_val	カウンタWに設定する値

戻り値

true	カウンタ値の設定に成功した場合
false	カウンタ値の設定に失敗した場合

出力先ファイル

R_PG_Timer_MTU_U〈ユニット番号〉_C〈チャネル番号〉.c
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～11

使用RPDL関数

R_MTU2_ControlChannel

詳細

- MTUのカウンタ値を設定します。

使用例

GUI上で以下の通り設定した場合
- MTUユニット0チャネル1を設定
- TGRAをアウトプットコンペアレジスタに設定し、コンペアマッチA割り込みを有効に設定
- コンペアマッチA割り込み通知関数名にMtu1IcCmAIntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();   // MTU1の設定
    R_PG_Timer_StartCount_MTU_U0_C1();   // カウント動作開始
}

void Mtu1IcCmAIntFunc(void)
{
    R_PG_Timer_SetCounterValue_MTU_U0_C1( 0 );   //カウンタの0クリア
}
```

4.10.6 R_PG_Timer_GetRequestFlag_MTU_U〈ユニット番号〉_C〈チャネル番号〉

定義

bool R_PG_Timer_GetRequestFlag_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(bool* cm_ic_a, bool* cm_ic_b, bool* cm_ic_c, bool* cm_ic_d,
bool* cm_e, bool* cm_f, bool* ov, bool* un);
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～3, 6～9

bool R_PG_Timer_GetRequestFlag_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(bool* cm_ic_a, bool* cm_ic_b, bool* cm_ic_c, bool* cm_ic_d,
bool* cm_e, bool* cm_f, bool* ov);
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 4, 10

bool R_PG_Timer_GetRequestFlag_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(bool* cm_ic_u, bool* cm_ic_v, bool* cm_ic_w);
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 5, 11

概要

MTUの割り込み要求フラグの取得とクリア

引数

bool* cm_ic_a	コンペアマッチ/インプットキャプチャAフラグの格納先
bool* cm_ic_b	コンペアマッチ/インプットキャプチャBフラグの格納先
bool* cm_ic_c	コンペアマッチ/インプットキャプチャCフラグの格納先
bool* cm_ic_d	コンペアマッチ/インプットキャプチャDフラグの格納先
bool* cm_e	コンペアマッチEフラグの格納先
bool* cm_f	コンペアマッチFフラグの格納先
bool* ov	オーバフローフラグの格納先
bool* un	アンダフローフラグの格納先
bool* cm_ic_u	コンペアマッチ/インプットキャプチャUフラグの格納先
bool* cm_ic_v	コンペアマッチ/インプットキャプチャVフラグの格納先
bool* cm_ic_w	コンペアマッチ/インプットキャプチャWフラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_Timer_MTU_U〈ユニット番号〉_C〈チャネル番号〉.c
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～11

使用RPDL関数

R_MTU2_ReadChannel

詳細

- MTUの割り込み要求フラグを取得します。
- 本関数内で全フラグがクリアされます。
- 取得するフラグに対応する引数に、フラグ値の格納先アドレスを指定してください。
  取得しないフラグには0を指定してください。
- コンペアマッチ/インプットキャプチャCおよびDフラグはチャネル0, 3, 4, 6, 9および10でのみ有効です。他のチャネルでは0を指定してください。
- アンダフローフラグはチャネル1, 2, 7およzび8でのみ有効です。他のチャネルでは0を指定してください。
- コンペアマッチEおよびFフラグはチャネル1および6でのみ有効です。他のチャネルでは0を指定してください。

使用例

GUI上で以下の通り設定した場合

- MTUユニット0チャネル1を設定
- TGRAをアウトプットコンペアレジスタに設定し、コンペアマッチA割り込みを有効に設定
  コンペアマッチA割り込みの優先レベルを0に設定
-

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool cma_flag;

void func(void)
{
	R_PG_Timer_Set_MTU_U0_C1();	// MTU1の設定
	R_PG_Timer_StartCount_MTU_U0_C1();	// カウント動作開始

	//コンペアマッチAの発生を待つ
	do{
		R_PG_Timer_GetRequestFlag_MTU_U0_C1(
			& cma_flag, //a
			0, //b
			0, //c
			0, //d
			0, //e
			0, //f
			0, //e
			0, //ov
			0 //un
		);
	} while( !cma_flag );

	//コンペアマッチA発生時処理

}
```

4.10.7 R_PG_Timer_StopModule_MTU_U〈ユニット番号〉

定義 bool R_PG_Timer_StopModule_MTU_U〈*ユニット番号*〉(void)
〈*ユニット番号*〉: 0, 1

概要 MTUのユニットを停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_Timer_MTU_U〈*ユニット番号*〉.c
〈*ユニット番号*〉: 0, 1

使用RPDL関数 R_MTU2_Destroy

詳細
- MTUのユニットを停止し、モジュールストップ状態に移行します。ユニット単位で停止させます。複数のチャネルが動作している場合、本関数を呼び出すとユニット内の全チャネルが停止します。1チャネルの動作だけを停止させる場合は以下の関数を呼び出してください。

  MTU0～MTU4、MTU6～MTU10
  R_PG_Timer_HaltCount_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉
  MTU5、MTU11
  R_PG_Timer_HaltCount_MTU_U〈*ユニット番号*〉_C〈*チャネル番号*〉_〈*相*〉

使用例 GUI上で以下の通り設定した場合
- MTUユニット0チャネル1を設定
- TGRAをアウトプットコンペアレジスタに設定し、コンペアマッチA割り込みを有効に設定
  コンペアマッチA割り込み通知関数名にMtu1IcCmAIntFuncを指定
-

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();   // MTU1の設定
    R_PG_Timer_StartCount_MTU_U0_C1();   //  カウント動作開始
}

void Mtu1IcCmAIntFunc(void)
{
    // MTUユニット0の停止
    R_PG_Timer_StopModule_MTU_U0();
}
```

4.10.8 R_PG_Timer_GetTGR_MTU_U〈ユニット番号〉_C〈チャネル番号〉

定義

bool R_PG_Timer_GetTGR_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t* tgr_a_val, uint16_t* tgr_b_val, uint16_t* tgr_c_val,
uint16_t* tgr_d_val, uint16_t* tgr_e_val, uint16_t* tgr_f_val);
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～4, 6～10

bool R_PG_Timer_GetTGR_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t * tgr_u_val, uint16_t * tgr_v_val, uint16_t * tgr_w_val);
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 5, 11

概要

ジェネラルレジスタの値の取得

引数

uint16_t* tgr_a_val	ジェネラルレジスタA値の格納先
uint16_t* tgr_b_val	ジェネラルレジスタB値の格納先
uint16_t* tgr_c_val	ジェネラルレジスタC値の格納先
uint16_t* tgr_d_val	ジェネラルレジスタD値の格納先
uint16_t* tgr_e_val	ジェネラルレジスタE値の格納先
uint16_t* tgr_f_val	ジェネラルレジスタF値の格納先
uint16_t* tgr_u_val	ジェネラルレジスタU値の格納先
uint16_t* tgr_v_val	ジェネラルレジスタV値の格納先
uint16_t* tgr_w_val	ジェネラルレジスタW値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_Timer_MTU_U〈ユニット番号〉_C〈チャネル番号〉.c
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 0～11

使用RPDL関数

R_MTU2_ReadChannel

詳細

- ジェネラルレジスタの値を取得します
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。
- ジェネラルレジスタCおよびDフラグはチャネル0, 3, 4, 6, 9および10でのみ有効です。他のチャネルでは0を指定してください。
- ジェネラルレジスタEおよびFフラグはチャネル1および6でのみ有効です。他のチャネルでは0を指定してください。

使用例

GUI上で以下の通り設定した場合

- MTUユニット0チャネル0を設定
- TGRAをインプットキャプチャレジスタに設定し、インプットキャプチャA割り込みを有効に設定
- インプットキャプチャA割り込み通知関数名にMtu0IcCmAIntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t tgr_a_val;

void func(void)
{
	R_PG_Timer_Set_MTU_U0_C0();	// MTU0の設定
	R_PG_Timer_StartCount_MTU_U0_C0();	// カウント動作開始
}

void Mtu0IcCmAIntFunc(void)
{
	//TGRAの値を取得
	R_PG_Timer_GetTGR_MTU_U0_C0(
		& tgr_a_val, //a
		0, //b
		0, //c
		0, //d
		0, //e
		0 //f
	);
}
```

4.10.9 R_PG_Timer_SetTGR_*<ジェネラルレジスタ>*_MTU_U*<ユニット番号>*_C*<チャネル番号>*

定義 bool R_PG_Timer_SetTGR_*<ジェネラルレジスタ>*_MTU_U*<ユニット番号>*_C*<チャネル番号>*
(uint16_t value);
<ジェネラルレジスタ>: A, B, C, D, E, or F (MTU0, MTU6)
A, B, C, or D (MTU3, MTU4, MTU9, MTU10)
A or B (MTU1, MTU2, MTU7, MTU8)
U, V or W (MTU5, MTU11)
<ユニット番号>: 0, 1
<チャネル番号>: 0～11

概要 ジェネラルレジスタの値の設定

引数

uint16_t value	ジェネラルレジスタに設定する値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_Timer_MTU_U*<ユニット番号>*_C*<チャネル番号>*.c
<ユニット番号>: 0, 1
<チャネル番号>: 0～11

使用RPDL関数 R_MTU2_ControlChannel

詳細
- ジェネラルレジスタの値を設定します。

使用例

GUI上で以下の通り設定した場合
- MTUユニット0チャネル1を設定
- TGRAをアウトプットコンペアレジスタに設定し、コンペアマッチA割り込みを有効に設定
  コンペアマッチA割り込み通知関数名にMtu1IcCmAIntFuncを指定
-

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C1();   // MTU1の設定
    R_PG_Timer_StartCount_MTU_U0_C1();   // カウント動作開始
}

void Mtu1IcCmAIntFunc(void)
{
    R_PG_Timer_SetTGR_A_MTU_U0_C1( 1000 );   //TGRAの設定
}
```

4.10.10 R_PG_Timer_SetBuffer_AD_MTU_U〈ユニット番号〉_C〈チャネル番号〉

定義

bool R_PG_Timer_SetBuffer_AD_MTU_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t tadcobr_a_val, uint16_t tadcobr_b_val);
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 4, 10

概要

A/D変換要求周期設定バッファレジスタの設定

生成条件

A/D変換要求周期レジスタ値のバッファ転送が有効

引数

uint16_t tadcobr_a_val	TADCOBRAに設定する値
uint16_t tadcobr_b_val	TADCOBRBに設定する値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_Timer_MTU_U〈ユニット番号〉_C〈チャネル番号〉.c
〈ユニット番号〉: 0, 1
〈チャネル番号〉: 4, 10

使用RPDL関数

R_MTU2_ControlChannel

詳細

- A/D変換要求周期設定バッファレジスタAおよびB(TADCOBRA、TADCOBRB)を設定します。

使用例

GUI上で以下の通り設定した場合

- A/D変換要求周期レジスタ値のバッファ転送を有効に設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
      R_PG_Timer_Set_MTU_U0_C4();   // MTU1の設定
      R_PG_Timer_StartCount_MTU_U0_C4();   // カウント動作開始
}

void func2(void)
{
      // A/D変換要求周期設定バッファレジスタの設定
      R_PG_Timer_SetBuffer_AD_MTU_U0_C4( 0x10, 0x20 );
}
```

4.11 ポートアウトプットイネーブル 2 (POE2)

4.11.1 R_PG_POE_Set

定義　bool R_PG_POE_Set (void)

概要　POEの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_POE.c

使用RPDL関数　R_POE_Set, R_POE_Create

詳細

- GUI上で選択されたMTU0、3、4、6、9、10の出力端子の制御と、ハイインピーダンス要求信号に使用する入力端子、アウトプットイネーブル割り込みを設定します。
- MTUの端子出力は、MTUのGUIおよび関数により設定してください。MTUで出力端子に設定していない端子は、POEで設定しないでください。
- GUI上で割り込み通知関数名を指定した場合、CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。
  void *〈割り込み通知関数名〉*(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上で以下の通り設定した場合

- アウトプットイネーブル割り込み2(OEI2)を有効に設定し、割り込み通知関数名にPoeOei2IntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_POE_Set();     // POEの設定
}

void PoeOei2IntFunc (void)
{
    //アウトプットイネーブル割り込み処理
}
```

4.11.2 R_PG_POE_SetHiZ_MTU<*MTUチャネル番号*>

定義 bool R_PG_POE_SetHiZ_MTU<*MTUチャネル番号*>(void)
<*MTUチャネル番号*>: 0, 3_4, 6, 9_10

概要 MTU端子をハイインピーダンスに設定

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_POE.c

使用RPDL関数 R_POE_Control

詳細
- GUI上でハイインピーダンス制御対象に指定されたMTU0、3、4、6、9、10の出力端子をハイインピーダンス状態にします。

使用例 GUI上で以下の通り設定した場合
- MTU0の端子出力を設定（MTUの設定GUI上）
- MTU0の出力端子をPOEのハイインピーダンス制御対象に指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    R_PG_Timer_Set_MTU_U0_C0();    //MTU0の設定
    R_PG_POE_Set();      // POEの設定
    R_PG_Timer_StartCount_MTU_U0_C0();      //MTU0のカウント動作開始
}

void func2(void)
{
    R_PG_POE_SetHiZ_MTU0();      //MTU0の出力端子をHiZに設定
}
```

4.11.3 R_PG_POE_GetRequestFlagHiZ_MTU<*MTUチャネル番号*>

定義 bool R_PG_POE_GetRequestFlagHiZ_MTU0 (bool* poe8)
bool R_PG_POE_GetRequestFlagHiZ_MTU3_4 (bool* poe0, bool* poe1, bool* poe2, bool* poe3)
bool R_PG_POE_GetRequestFlagHiZ_MTU6 (bool* poe9)
bool R_PG_POE_GetRequestFlagHiZ_MTU9_10 (bool* poe4, bool* poe5, bool* poe6, bool* poe7)

概要 ハイインピーダンス要求フラグの取得

引数

bool* poen (n:0～9)	POEn#(n:0～9)端子のハイインピーダンス要求フラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_POE.c

使用RPDL関数 R_POE_GetStatus

詳細
- POEn#(n:0～9)端子へのハイインピーダンス要求信号入力フラグ(POEnF n:0～9)を取得します。
- 取得するフラグに対応する引数に格納先アドレスを指定してください。取得しないフラグに対応する引数には0を指定してください。
- GUI上でハイインピーダンス要求条件に指定していないPOE端子のフラグには有効な値が格納されません。

使用例 GUI上で以下の通り設定した場合
- MTU3,4の端子出力を設定 (MTUの設定GUI上)
- MTU3,4の出力端子をPOEのハイインピーダンス制御対象に指定
- ハイインピーダンス要求条件にPOE0を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool poe0;

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C3();    //MTUの設定
    R_PG_POE_Set();    // POEの設定
    R_PG_Timer_StartCount_MTU_U0_C3();    //MTUのカウント動作開始

    //ハイインピーダンス要求入力を待つ
    do{
        R_PG_POE_GetRequestFlagHiZ_MTU3_4( &poe0, 0, 0, 0 );
    }while( ! poe0 );

    //ハイインピーダンス要求入力時処理

    R_PG_POE_ClearFlag_MTU3_4();    //ハイインピーダンス要求フラグのクリア
}
```

4.11.4 R_PG_POE_GetShortFlag_MTU<*MTUチャネル番号*>

定義 bool R_PG_POE_GetShortFlag_MTU3_4 (bool * detected)
bool R_PG_POE_GetShortFlag_MTU9_10 (bool * detected)

概要 MTU端子の出力短絡フラグの取得

引数

bool* detected	出力短絡フラグ(MTU3,4:OSF1またはMTU9,10:OSF2)の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_POE.c

使用RPDL関数 R_POE_GetStatus

詳細
- MTU3,4またはMTU9,10の相補PWM出力短絡フラグ(MTU3,4:OSF1またはMTU9,10:OSF2)を取得します。

使用例 GUI上で以下の通り設定した場合
- アウトプットイネーブル割り込み1(OEI1)を有効に設定
- アウトプットイネーブル割り込み1の通知関数名にPoeOei1IntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_POE_Set();    // POEの設定
}

void PoeOei1IntFunc(void)
{
    bool detected;

    //出力短絡フラグの取得
    R_PG_POE_GetShortFlag_MTU3_4 (&detected);

    if( detected ){
        //MTU3,4の出力短絡検出時処理

        R_PG_POE_ClearFlag_MTU3_4();    //出力短絡フラグ(OSF1)のクリア
    }
}
```

4.11.5 R_PG_POE_ClearFlag_MTU<*MTUチャネル番号*>

定義 bool R_PG_POE_ClearFlag_MTU<*MTUチャネル番号*>(void)

<*MTUチャネル番号*>: 0, 3_4, 6, 9_10

概要 ハイインピーダンス要求フラグと出力短絡フラグのクリア

引数 なし

戻り値

true	クリアに成功した場合
false	クリアに失敗した場合

出力先ファイル R_PG_POE.c

使用RPDL関数 R_POE_Control

詳細

- ハイインピーダンス要求フラグと出力短絡フラグをクリアします。
- MTUの各チャネルに対応した関数でクリアされるフラグは次の通りです。

MTU	クリア対象
0	POE8要求フラグ(POE8F)
3, 4	POE0～3要求フラグ(POE0F～POE3F)、MTU3,4出力短絡フラグ(OSF1)
6	POE9要求フラグ(POE9F)
9, 10	POE4～7要求フラグ(POE4F～POE7F)、MTU9,10出力短絡フラグ(OSF2)

使用例 GUI上で以下の通り設定した場合

- アウトプットイネーブル割り込み1(OEI1)を有効に設定
- アウトプットイネーブル割り込み1の通知関数名にPoeOei1IntFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_POE_Set();    // POEの設定
}

void PoeOei1IntFunc(void)
{
    bool detected;

    //出力短絡フラグの取得
    R_PG_POE_GetShortFlag_MTU3_4 (&detected);

    if( detected ){
        //MTU3,4の出力短絡検出時処理

        R_PG_POE_ClearFlag_MTU3_4();    //出力短絡フラグのクリア
    }
}
```

4.12 プログラマブルパルスジェネレータ（PPG）

4.12.1 R_PG_PPG_StartOutput_U〈ユニット番号〉_G〈グループ番号〉

定義　bool R_PG_PPG_StartOutput_U〈ユニット番号〉_G〈グループ番号〉(void)
〈ユニット番号〉: 0, 1
〈グループ番号〉: 0～7

概要　PPGの設定とパルス出力の開始

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_PPG_U〈ユニット番号〉.c
〈ユニット番号〉: 0, 1

使用RPDL関数　R_PPG_Create

詳細
- PPGユニット(0または1)のモジュールストップ状態を解除してパルス出力グループを初期設定し、GUI上で選択した出力端子からの出力を開始します。
- 本関数内で出力端子からの初期出力値と、1回目の更新時の出力値が設定されます。
- MTUは本関数で設定されません。MTUのGUIおよび関数により設定してください。

使用例　GUI上で以下の通り設定した場合
- PPGパルス出力グループ2のPO8～PO11を設定
- MTU0のコンペアマッチA割り込みを有効にし、Mtu0IcCmAIntFuncを割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t output_val;

void func(void)
{
    output_val=1;

    R_PG_Timer_Set_MTU_U0_C0();      // MTU0の設定
    R_PG_PPG_StartOutput_U0_G2();    // PPG出力グループ2の設定と出力の開始
    R_PG_Timer_StartCount_MTU_U0_C0();    // MTU0のカウント動作開始
}

//MTU0コンペアマッチA割り込み通知関数
void Mtu0IcCmAIntFunc (void)
{
    //次の出力値の設定
    R_PG_PPG_SetOutputValue_U0_G2( output_val );
}
```

4.12.2 R_PG_PPG_StopOutput_U〈ユニット番号〉_G〈グループ番号〉

定義 bool R_PG_PPG_StopOutput_U〈ユニット番号〉_G〈グループ番号〉(void)
〈ユニット番号〉: 0, 1
〈グループ番号〉: 0～7

概要 パルス出力の停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_PPG_U〈ユニット番号〉.c
〈ユニット番号〉: 0, 1

使用RPDL関数 R_PPG_Destroy

詳細
- パルス出力グループに含まれる出力端子からのパルス出力を停止します。
- ユニット内の全グループが停止する場合、PPGのユニットはモジュールストップ状態に移行します。

使用例 GUI上で以下の通り設定した場合
- PPGパルス出力グループ2のPO8～PO11を設定
- MTU0のコンペアマッチA割り込みを有効にし、Mtu0IcCmAIntFuncを割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t output_val;

void func(void)
{
    output_val=1;

    R_PG_Timer_Set_MTU_U0_C0();      // MTU0の設定
    R_PG_PPG_StartOutput_U0_G2();    // PPG出力グループ2の設定と出力の開始
    R_PG_Timer_StartCount_MTU_U0_C0();    // MTU0のカウント動作開始
}

//MTU0コンペアマッチA割り込み通知関数
void Mtu0IcCmAIntFunc (void)
{
    output_val++;      //出力値のインクリメント

    R_PG_PPG_SetOutputValue_U0_G2( output_val );      //次の出力値の設定

    if(output_val >= 0x0f){
        R_PG_PPG_StopOutput_U0_G2();      //出力の停止
    }
}
```

4.12.3 R_PG_PPG_SetOutputValue_U〈ユニット番号〉_G〈グループ番号〉

定義 bool R_PG_PPG_SetOutputValue_U〈ユニット番号〉_G〈グループ番号〉(uint8_t output_val)
〈ユニット番号〉: 0, 1
〈グループ番号〉: 0～7

概要 1グループ(4bit)の出力値の設定

引数

uint8_t output_val	次の更新タイミング(MTUのコンペアマッチ)での出力値 グループ0,2,4,6：bit3～bit0が有効 グループ1,3,5,7：bit7～bit4が有効

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_PPG_U〈ユニット番号〉.c 〈ユニット番号〉: 0, 1

使用RPDL関数 R_PPG_Control

詳細
- 1つのパルス出力グループの次の更新タイミング(MTUのコンペアマッチ)での出力値を設定します。
- 引数(output_val)の各ビットと出力端子の関係は次の通りです。
  出力グループ1,3,5,7では上位4ビットに出力値を設定してください。

出力グループのペア	出力グループ 1, 3, 5 or 7				出力グループ 0, 2, 4 or 6			
	b7	b6	b5	b4	b3	b2	b1	b0
1、0	PO7	PO6	PO5	PO4	PO3	PO2	PO1	PO0
3、2	PO15	PO14	PO13	PO12	PO11	PO10	PO9	PO8
5、4	PO23	PO22	PO21	PO20	PO19	PO18	PO17	PO16
7、6	PO31	PO30	PO29	PO28	PO27	PO26	PO25	PO24

使用例 GUI上で以下の通り設定した場合
- PPGパルス出力グループ1のPO4～PO7を設定
- MTU0のコンペアマッチA割り込みを有効にし、Mtu0IcCmAIntFuncを割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t output_val;

void func(void)
{
    output_val=1;

    R_PG_Timer_Set_MTU_U0_C0();      // MTU0の設定
    R_PG_PPG_StartOutput_U0_G1();    // PPG出力グループ1の設定と出力の開始
    R_PG_Timer_StartCount_MTU_U0_C0();    // MTU0のカウント動作開始
}

void Mtu0IcCmAIntFunc (void)
{
    output_val++;      //出力値のインクリメント

    R_PG_PPG_SetOutputValue_U0_G1( output_val << 4 );      //次の出力値の設定

    if(output_val >= 0x0f){
        R_PG_PPG_StopOutput_U0_G1();      //出力の停止
    }
}
```

4.12.4 R_PG_PPG_SetOutputValue_U〈ユニット番号〉_G〈グループ番号 1〉_G〈グループ番号 2〉

定義 bool R_PG_PPG_SetOutputValue_ U〈ユニット番号〉_G〈グループ番号1〉_G〈グループ番号2〉
(uint8_t output_val)
〈ユニット番号〉: 0, 1
〈グループ番号1〉: 1, 3, 5, 7
〈グループ番号2〉: 0, 2, 4, 6

概要 2グループ(8bit)の出力値の設定
生成条件 ペアとなる2つのパルス出力グループが設定された場合

引数

uint8_t output_val	次の更新タイミング(MTUのコンペアマッチ)での出力値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_PPG_U〈ユニット番号〉.c 〈ユニット番号〉: 0, 1

使用RPDL関数 R_PPG_Control

詳細
- ペアとなる2つのパルス出力グループ(0と1、2と3、4と5、または6と7)の次の更新タイミング(MTUのコンペアマッチ)での出力値を設定します。
- 引数(output_val)の各ビットと出力端子の関係は次の通りです。

出力グループのペア	出力グループ 1, 3, 5 or 7				出力グループ 0, 2, 4 or 6			
	b7	b6	b5	b4	b3	b2	b1	b0
1、 0	PO7	PO6	PO5	PO4	PO3	PO2	PO1	PO0
3、 2	PO15	PO14	PO13	PO12	PO11	PO10	PO9	PO8
5、 4	PO23	PO22	PO21	PO20	PO19	PO18	PO17	PO16
7、 6	PO31	PO30	PO29	PO28	PO27	PO26	PO25	PO24

使用例 GUI上で以下の通り設定した場合
- PPGパルス出力グループ0および1を設定
- MTU0のコンペアマッチA割り込みを有効にし、Mtu0IcCmAIntFuncを割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t output_val;

void func(void)
{
    R_PG_Timer_Set_MTU_U0_C0();      // MTU0の設定
    R_PG_PPG_StartOutput_U0_G0();    // PPG出力グループ0の設定と出力の開始
    R_PG_PPG_StartOutput_U0_G1();    // PPG出力グループ1の設定と出力の開始
    R_PG_Timer_StartCount_MTU_U0_C0();    // MTU0のカウント動作開始
}

void Mtu0IcCmAIntFunc (void)
{
    R_PG_PPG_SetOutputValue_U0_G0_G1( output_val  );    //次の出力値の設定
}
```

4.13 8 ビットタイマ（TMR）

4.13.1 R_PG_Timer_Start_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉

定義　bool R_PG_Timer_Start_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉(void)
〈*ユニット番号*〉： 0, 1
〈*チャネル番号*〉： 0～3
（(_C〈*チャネル番号*〉) は8ビットモード時に付加します）

概要　TMRを設定しカウント動作を開始

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Timer_TMR_U〈*ユニット番号*〉.c
〈*ユニット番号*〉： 0, 1

使用RPDL関数　R_TMR_CreateChannel (8ビットモード時)
R_TMR_CreateUnit (16ビットモード時)

詳細
- TMRのモジュールストップ状態を解除して初期設定し、カウント動作を開始します。8ビットモード時はチャネルごとに、16ビットモード（ユニット内の2チャネルをカスケード接続）時はユニットごとに設定します。
- 本関数内でTMRの割り込みを設定します。GUI上で割り込み通知関数名を指定した場合、CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。　void 〈*割り込み通知関数名*〉(void)
割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- GUI上で割り込み優先レベルを0に設定した場合、CPU割り込みは発生しません。割り込み要求フラグは R_PG_Timer_GetRequestFlag_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉により取得することができます。
- 外部入力カウントクロック、外部リセット信号、パルス出力を使用する場合、本関数内で使用する端子の入出力方向と入力バッファを設定します。

使用例

16ビットタイマモードでTMRのユニット1を設定
GUI上で次の割り込み通知関数を設定した場合
オーバフロー割り込み ： TmrOf2IntFunc
コンペアマッチA割り込み ： TmrCma2IntFunc
コンペアマッチB割り込み ： TmrCma2IntFunc

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //TMRユニット1を16ビットモードで設定する
    R_PG_Timer_Start_TMR_U0();
}

void TmrOf2IntFunc(void)
{
    func_of();      //オーバフロー割り込み発生時の処理
}

void TmrCma2IntFunc(void)
{
    func_cmA();     //コンペアマッチA割り込み発生時の処理
}

void TmrCma2IntFunc(void)
{
    func_cmb();     //コンペアマッチB割り込み発生時の処理
}
```

GUI上でTMR0を8ビットタイマモードに設定
GUI上で割り込みフラグのチェックにより割り込み要求の有無を確認

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    bool cma_flag;

    //TMR0を8ビットモードで設定し、カウント動作を開始する
    R_PG_Timer_Start_TMR_U0_C0();

    While(1){
        //コンペアマッチA割り込み要求フラグを取得する
        R_PG_Timer_GetRequestFlag_TMR_U0_C0( &cma_flag, 0, 0 );

        if( cma_flag ){
            func_cmA0();    //コンペアマッチA割り込みの処理
        }
    }
}
```

4.13.2 R_PG_Timer_HaltCount_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉

定義　bool R_PG_Timer_HaltCount_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉(void)
〈*ユニット番号*〉： 0, 1
〈*チャネル番号*〉： 0～3
(_C〈*チャネル番号*〉は8ビットモード時に付加します)

概要　TMRのカウント動作を一時停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_Timer_TMR_U〈*ユニット番号*〉.c
〈*ユニット番号*〉： 0, 1

使用RPDL関数　R_TMR_ControlChannel (8ビットモード時)
R_TMR_ControlUnit (16ビットモード時)

詳細
- TMRのカウント動作を一時停止します。カウント動作を再開するには
  R_PG_Timer_ResumeCount_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉
  を呼び出してください。

使用例　GUI上でTMR0を8ビットタイマモードに設定
GUI上でコンペアマッチA割り込み関数名に TmrCma0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //TMR0を8ビットモードで設定する
    R_PG_Timer_Start_TMR_U0_C0();
}

void TmrCma0IntFunc(void)
{
    //TMR0のカウント動作を一時停止
    R_PG_Timer_HaltCount_TMR_U0_C0();

    func_cmA();     //コンペアマッチA割り込み発生時の処理

    //TMR0のカウント動作を再開
    R_PG_Timer_ResumeCount_TMR_U0_C0();
}
```

4.13.3 R_PG_Timer_ResumeCount_TMR_U〈ユニット番号〉_C〈チャネル番号〉)

定義 bool R_PG_Timer_ResumeCount_TMR_U〈ユニット番号〉_C〈チャネル番号〉) (void)
〈ユニット番号〉 : 0, 1
〈チャネル番号〉 : 0～3
((_C〈チャネル番号〉) は8ビットモード時に付加します)

概要 TMRのカウント動作を再開

引数 なし

戻り値

true	カウント動作の再開が正しく行われた場合
false	カウント動作の再開に失敗した場合

出力先ファイル R_PG_Timer_TMR_U〈ユニット番号〉.c
〈ユニット番号〉 : 0, 1

使用RPDL関数 R_TMR_ControlChannel (8ビットモード時)
R_TMR_ControlUnit (16ビットモード時)

詳細
- R_PG_Timer_HaltCount_TMR_U〈ユニット番号〉_C〈チャネル番号〉)により停止したTMRのカウント動作を再開します。

使用例 GUI上でTMR0を8ビットタイマモードに設定
GUI上でコンペアマッチA割り込み関数名に TmrCma0IntFunc を指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //TMR0を8ビットモードで設定する
    R_PG_Timer_Start_TMR_U0_C0();
}

void TmrCma0IntFunc(void)
{
    //TMR0のカウント動作を一時停止
    R_PG_Timer_HaltCount_TMR_U0_C0();

    func_cmA();     //コンペアマッチA割り込み発生時の処理

    //TMR0のカウント動作を再開
    R_PG_Timer_ResumeCount_TMR_U0_C0();
}
```

4.13.4 R_PG_Timer_GetCounterValue_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉

定義

・8ビットモード時

bool R_PG_Timer_GetCounterValue_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉
(uint8_t * counter_val)

〈*ユニット番号*〉 : 0, 1

〈*チャネル番号*〉 : 0～3

・16ビットモード時

bool R_PG_Timer_GetCounterValue_TMR_U〈*ユニット番号*〉(uint16_t * counter_val)

〈*ユニット番号*〉 : 0, 1

概要

TMRのカウンタ値を取得

引数

uint8_t * counter_val uint16_t * counter_val	カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_Timer_TMR_U〈*ユニット番号*〉.c

〈*ユニット番号*〉 : 0, 1

使用RPDL関数

R_TMR_ReadChannel (8ビットモード時)

R_TMR_ReadUnit (16ビットモード時)

詳細

・ TMRのカウンタ値を取得します。

8ビットタイマモード時は指定したチャネルの8ビットカウンタ値が、16ビットモード時は次のように各チャネルのカウンタ値が格納されます。

ユニット	b15 - b8	b7 - b0
0	TMR0カウンタ	TMR1カウンタ
1	TMR2カウンタ	TMR3カウンタ

※16ビットモード時はTMR0(TMR2)が上位ビットとして動作します。

使用例

GUI上でTMR0を8ビットタイマモードに設定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    //TMR0を8ビットモードで設定する
    R_PG_Timer_Start_TMR_U0_C0();
}

unt8_t func2(void)
{
    uint8_t counter_val;

    //TMR0のカウンタ値を取得
    R_PG_Timer_GetCounterValue_TMR_U0_C0( &counter_val );

    return data;
}
```

4.13.5 R_PG_Timer_SetCounterValue_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉

定義 ・8ビットモード時

bool R_PG_Timer_SetCounterValue_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉(uint8_t counter_val)

〈*ユニット番号*〉 : 0, 1

〈*チャネル番号*〉 : 0～3

・16ビットモード時

bool R_PG_Timer_SetCounterValue_TMR_U〈*ユニット番号*〉(uint16_t counter_val)

〈*ユニット番号*〉 : 0, 1

概要 TMRのカウンタ値を設定

引数

uint8_t counter_val (8ビットモード時) uint16_t counter_val (16ビットモード時)	カウンタに設定する値

戻り値

true	カウンタ値の設定に成功した場合
false	カウンタ値の設定に失敗した場合

出力先ファイル R_PG_Timer_TMR_U〈*ユニット番号*〉.c

〈*ユニット番号*〉 : 0, 1

使用RPDL関数 R_TMR_ControlChannel (8ビットモード時)

R_TMR_ControlUnit (16ビットモード時)

詳細 ・ TMRのカウンタ値を設定します。

8ビットタイマモード時は指定したチャネルの8ビットカウンタ値が、16ビットモード時は次のように各チャネルのカウンタ値が格納されます。

ユニット	b15 - b8	b7 - b0
0	TMR0カウンタ	TMR1カウンタ
1	TMR2カウンタ	TMR3カウンタ

※16ビットモード時はTMR0(TMR2)が上位ビットとして動作します。

使用例 GUI上でTMR0を8ビットタイマモードに設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    //TMR0を8ビットモードで設定する
    R_PG_Timer_Start_TMR_U0_C0();
}

void func2(void)
{
    //TMR0のカウンタ値を設定
    R_PG_Timer_SetCounterValue_TMR_U0_C0( 0 );
}
```

4.13.6 R_PG_Timer_GetRequestFlag_TMR_U〈*ユニット番号*〉_C〈*チャネル番号*〉)

定義　bool R_PG_Timer_GetRequestFlag_TMR_U〈*ユニット番号*〉(_C〈*チャネル番号*〉)
(bool* cma, bool* cmb, bool* ov);
〈*ユニット番号*〉 : 0, 1
〈*チャネル番号*〉 : 0～3
((_C〈*チャネル番号*〉) は8ビットモード時に付加します)

概要　TMRの割り込み要求フラグの取得とクリア

引数　なし

引数

bool* cma	コンペアマッチAフラグの格納先
bool* cmb	コンペアマッチBフラグの格納先
bool* ov	オーバフローフラグの格納先

戻り値

true	フラグの取得に成功した場合
false	フラグの取得に失敗した場合

出力先ファイル　R_PG_Timer_TMR_U〈*ユニット番号*〉.c
〈*ユニット番号*〉 : 0, 1

使用RPDL関数　R_TMR_ReadChannel(8ビットモード時)
R_TMR_ReadUnit(16ビットモード時)

詳細
- TMRの割り込み要求フラグを取得します。
- 本関数内で全フラグがクリアされます。
- 取得するフラグに対応する引数に、フラグ値の格納先アドレスを指定してください。
- 取得しないフラグには0を指定してください。

使用例　GUI上でTMR0を8ビットタイマモードに設定
GUI上でコンペアマッチA割り込み関数名に TmrCma0IntFunc を指定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

bool cma_flag;

void func(void)
{
    //TMR0を8ビットモードで設定する
    R_PG_Timer_Start_TMR_U0_C0();

    //コンペアマッチAの検出を待つ
    do{
        R_PG_Timer_GetRequestFlag_TMR_U0_C0(
            & cma_flag,
            0,
            0
        );
    } while( !cma_flag );

    func_cmA();      //コンペアマッチA割り込み発生時の処理
}
```

4.13.7 R_PG_Timer_StopModule_TMR_U〈ユニット番号〉

定義　bool R_PG_Timer_StopModule_TMR_U〈ユニット番号〉(void)
〈ユニット番号〉： 0, 1

概要　TMRのユニットを停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_Timer_TMR_U〈ユニット番号〉.c
〈ユニット番号〉： 0, 1

使用RPDL関数　R_TMR_Destroy

詳細

- TMRのユニットを停止し、モジュールストップ状態に移行します。ユニット単位で停止させます。ユニット0のTMR0とTMR1（ユニット1はTMR2とTMR3）が両方動作している場合、本関数を呼び出すとユニット内の2チャネルが停止します。片方のチャネルの動作だけを停止させる場合は、
R_PG_Timer_HaltCount_TMR_U〈ユニット番号〉_C〈チャネル番号〉
を使用してください。

使用例　GUI上でTMR0を8ビットタイマモードに設定
GUI上でコンペアマッチA割り込み関数名に TmrCma0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //TMR0を8ビットモードで設定する
    R_PG_Timer_Start_TMR_U0_C0();
}

void TmrCma0IntFunc(void)
{
    func_cmA();     //コンペアマッチA割り込み発生時の処理

    //TMRユニット0を停止
    R_PG_Timer_StopModule_TMR_U0();
}
```

4.14 コンペアマッチタイマ（CMT）

4.14.1 R_PG_Timer_Start_CMT_U〈ユニット番号〉_C〈チャネル番号〉

定義　bool R_PG_Timer_Start_CMT_U〈ユニット番号〉_C〈チャネル番号〉(void)
〈ユニット番号〉： 0, 1
〈チャネル番号〉： 0～3

概要　CMTを設定しカウント動作を開始

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Timer_CMT_U〈ユニット番号〉.c
〈ユニット番号〉： 0, 1

使用RPDL関数　R_CMT_Create

詳細
- CMTのモジュールストップ状態を解除して初期設定し、カウント動作を開始します。
- 本関数内でCMTの割り込みを設定します。GUI上で割り込み通知関数名を指定した場合、CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。　void 〈割り込み通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上でコンペアマッチ割り込み通知関数名に Cmt0IntFunc を設定した場合

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //CMT0を設定する
    R_PG_Timer_Start_CMT_U0_C0();
}

void Cmt0IntFunc(void)
{
    func_cmt0();    //コンペアマッチ割り込み発生時の処理
}
```

4.14.2 R_PG_Timer_HaltCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉

定義　bool R_PG_Timer_HaltCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉(void)
〈ユニット番号〉： 0, 1
〈チャネル番号〉： 0～3

概要　CMTのカウント動作を一時停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_Timer_CMT_U〈ユニット番号〉.c
〈ユニット番号〉： 0, 1

使用RPDL関数　R_CMT_Control

詳細
- CMTのカウント動作を一時停止します。カウント動作を再開するには R_PG_Timer_ResumeCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉 を呼び出してください。

使用例　GUI上でコンペアマッチ割り込み関数名に Cmt0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //CMT0を設定する
    R_PG_Timer_Start_CMT_U0_C0();
}

void Cmt0IntFunc(void)
{
    //CMT0のカウント動作を一時停止
    R_PG_Timer_HaltCount_CMT_U0_C0();

    func_cmt0();    //コンペアマッチ割り込み発生時の処理

    //CMT0のカウント動作を再開
    R_PG_Timer_ResumeCount_CMT_U0_C0();
}
```

4.14.3 R_PG_Timer_ResumeCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉

定義　bool R_PG_Timer_ResumeCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉(void)
〈ユニット番号〉： 0, 1
〈チャネル番号〉： 0～3

概要　CMTのカウント動作を再開

引数　なし

戻り値

true	カウント動作の再開が正しく行われた場合
false	カウント動作の再開に失敗した場合

出力先ファイル　R_PG_Timer_CMT_U〈ユニット番号〉.c
〈ユニット番号〉： 0, 1

使用RPDL関数　R_CMT_Control

詳細
- R_PG_Timer_HaltCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉 により停止したCMTのカウント動作を再開します。

使用例　GUI上でコンペアマッチ割り込み関数名に Cmt0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	//CMT0を設定する
	R_PG_Timer_Start_CMT_U0_C0();
}

void Cmt0IntFunc(void)
{
	//CMT0のカウント動作を一時停止
	R_PG_Timer_HaltCount_CMT_U0_C0();

	func_cmt0();	//コンペアマッチ割り込み発生時の処理

	//CMT0のカウント動作を再開
	R_PG_Timer_ResumeCount_CMT_U0_C0();
}
```

4.14.4 R_PG_Timer_GetCounterValue_CMT_U〈ユニット番号〉_C〈チャネル番号〉

定義 bool R_PG_Timer_GetCounterValue_CMT_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t * counter_val)
〈ユニット番号〉 : 0, 1
〈チャネル番号〉 : 0～3

概要 CMTのカウンタ値を取得

引数

uint16_t * counter_val	カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_Timer_CMT_U〈ユニット番号〉.c
〈ユニット番号〉 : 0, 1

使用RPDL関数 R_CMT_Read

詳細
- CMTのカウンタ値を取得します。

使用例 GUI上でCMT0を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t counter_val;

void func1(void)
{
	//CMT0を設定する
	R_PG_Timer_Start_CMT_U0_C0();
}

void func2(void)
{
	//CMT0のカウンタ値を取得
	R_PG_Timer_GetCounterValue_CMT_U0_C0( &counter_val );
}
```

4.14.5 R_PG_Timer_SetCounterValue_CMT_U〈ユニット番号〉_C〈チャネル番号〉

定義 bool R_PG_Timer_SetCounterValue_CMT_U〈ユニット番号〉_C〈チャネル番号〉
(uint16_t counter_val)
〈ユニット番号〉 : 0, 1
〈チャネル番号〉 : 0～3

概要 CMTのカウンタ値を設定

引数

uint16_t counter_val	カウンタに設定する値

戻り値

true	カウンタ値の設定に成功した場合
false	カウンタ値の設定に失敗した場合

出力先ファイル R_PG_Timer_CMT_U〈ユニット番号〉.c
〈ユニット番号〉 : 0, 1

使用RPDL関数 R_CMT_Control

詳細
- CMTのカウンタ値を設定します。

使用例 GUI上でCMT0を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
	//CMT0を設定する
	R_PG_Timer_Start_CMT_U0_C0();
}

void func2(void)
{
	//CMT0のカウンタ値を設定
	R_PG_Timer_SetCounterValue_CMT_U0_C0( 0 );
}
```

4.14.6 R_PG_Timer_StopModule_CMT_U〈ユニット番号〉

定義　bool R_PG_Timer_StopModule_CMT_U〈ユニット番号〉(void)
〈ユニット番号〉： 0, 1

概要　CMTのユニットを停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_Timer_CMT_U〈ユニット番号〉.c
〈ユニット番号〉： 0, 1

使用RPDL関数　R_CMT_Destroy

詳細

- CMTのユニットを停止し、モジュールストップ状態に移行します。ユニット単位で停止させます。ユニット0のCMT0とCMT1（ユニット1はCMT2とCMT3）が両方動作している場合、本関数を呼び出すとユニット内の2チャネルが停止します。片方のチャネルの動作だけを停止させる場合は、
  R_PG_Timer_HaltCount_CMT_U〈ユニット番号〉_C〈チャネル番号〉
  を使用してください。

使用例　GUI上でコンペアマッチ割り込み関数名に Cmt0IntFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    //CMT0を設定する
    R_PG_Timer_Start_CMT_U0_C0();
}

void Cmyt0IntFunc(void)
{
    func_cmt();     //コンペアマッチ割り込み発生時の処理

    //CMTユニット0を停止
    R_PG_Timer_StopModule_CMT_U0();
}
```

4.15 リアルタイムクロック（RTC）

4.15.1 R_PG_RTC_Start

定義　bool R_PG_RTC_Start(void)

概要　RTCを設定しカウント動作を開始

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Create

詳細

- アラーム割り込みと周期割り込み、RTCOUT端子からの1Hzクロック出力を設定し、カウント動作を開始します。
- 本関数を呼び出す前にR_PG_Clock_Setによりクロックを設定し、R_PG_Clock_Start_SUBによりサブクロックを有効にしてください。
- 現在時刻は設定されません。アラーム割り込みを使用する場合、現在時刻は本関数を呼び出した後にR_PG_RTC_SetCurrentTimeにより設定してください。
- GUI上でアラーム日時を設定した場合はアラームレジスタが設定されます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set(); //クロックの設定
    R_PG_Clock_Start_SUB(); //サブクロックの発振開始
    R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
}
```

4.15.2 R_PG_RTC_Stop

定義　bool R_PG_RTC_Stop(void)

概要　RTCのカウント動作を一時停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細

- RTCのカウント動作を停止します。
- カウント動作を再開するにはR_PG_RTC_Restartを呼び出してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	R_PG_Clock_Set(); //クロックの設定
	R_PG_Clock_Start_SUB(); //サブクロックの発振開始
	R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
}

void func2(void)
{
	R_PG_RTC_Stop ();   //カウント動作の停止
}

void func3(void)
{
	R_PG_RTC_Restart();   //カウント動作の再開
}
```

4.15.3 R_PG_RTC_Restart

定義　bool R_PG_RTC_Restart(void)

概要　RTCのカウント動作を再開

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細

- R_PG_RTC_Stopにより停止したRTCのカウント動作を再開します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	R_PG_Clock_Set(); //クロックの設定
	R_PG_Clock_Start_SUB(); //サブクロックの発振開始
	R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
}

void func2(void)
{
	R_PG_RTC_Stop ();   //カウント動作の停止
}

void func3(void)
{
	R_PG_RTC_Restart();   //カウント動作の再開
}
```

4.15.4 R_PG_RTC_SetCurrentTime

定義

```
bool R_PG_RTC_SetCurrentTime
(uint8_t seconds,  uint8_t minutes,  uint8_t hours,
 uint8_t day,  uint8_t month,  uint16_t year  )
```

概要　現在時刻の設定

生成条件　アラーム割り込みが設定されている場合

引数

uint8_t seconds	秒（有効な値:0x00～0x59 (BCDコード))
uint8_t minutes	分（有効な値:0x00～0x59 (BCDコード))
uint8_t hours	時間（有効な値:0x00～0x23 (BCDコード))
uint8_t day	日（有効な値:0x01～monthで指定される月の日数（BCDコード))
uint8_t month	月（有効な値:0x01～0x12 (BCDコード))
uint16_t year	年（有効な値: 0x0000～0x9999 (BCDコード))

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細

- 現在時刻を設定します。
- 曜日カウンタの値は指定された年月日から算出され、曜日カウンタに設定されます。
- カウント動作中に本関数を呼び出した場合、現在時刻のカウンタ設定中はカウント動作が停止し、設定後にカウントを再開します。
- アラーム割り込みで使用しない項目にも有効な範囲の値を設定してください。

使用例

GUI上で以下の通り設定した場合

- アラーム割り込みを設定
- RtcAlmIntFuncをアラーム割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set(); //クロックの設定
    R_PG_Clock_Start_SUB(); //サブクロック発振開始
    R_PG_RTC_Start(); //RTCの設定とカウント動作の開始

    R_PG_RTC_SetCurrentTime( //現在時刻の設定(2000年11月22日03時44分55秒)
        0x55, //55秒
        0x44, //44分
        0x03, //03時
        0x22, //22日
        0x11, //11月
        0x2000 //2000年
    );

    R_PG_RTC_SetAlarmTime( //アラーム時刻の設定(2000年11月22日03時45分00秒)
        0x00, //00秒
        0x45, //45分
        0x03, //03時
        0xff, //曜日(0xff : 指定した日時から自動計算する)
        0x22, //22日
        0x11, //11月
        0x2000 //2000年
    );

    R_PG_RTC_Alarm_Control( //年、月、日、曜日、時、分、秒アラーム有効化
        1, //秒アラーム有効
        1, //分アラーム有効
        1, //時アラーム有効
        1, //曜日アラーム有効
        1, //日アラーム有効
        1, //月アラーム有効
        1 //年アラーム有効
    );
}

void RtcAlmIntFunc(void)
{
    //アラーム割り込み処理
}
```

4.15.5 R_PG_RTC_SetAlarmTime

定義

```
bool R_PG_RTC_SetAlarmTime
( uint8_t seconds,  uint8_t minutes,  uint8_t hours,  uint8_t day_of_week,
  uint8_t day,      uint8_t month,    uint16_t year );
```

概要　アラーム時刻の設定

生成条件　アラーム割り込みが設定されている場合

引数

uint8_t seconds	秒（有効な値:0x00～0x59 (BCDコード))
uint8_t minutes	分（有効な値:0x00～0x59 (BCDコード))
uint8_t hours	時間（有効な値:0x00～0x23 (BCDコード))
uint8_t day_of_week	曜日（有効な値:0x00(日曜)～0x06(土曜)） 0xffを設定するとday,month,yearで指定した値から算出されます
uint8_t day	日（有効な値:0x01～monthで指定される月の日数（BCDコード))
uint8_t month	月（有効な値:0x01～0x12 (BCDコード))
uint16_t year	年（有効な値: 0x0000～0x9999 (BCDコード))

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細
- アラーム時刻を設定します。
- アラームで使用しない項目にも有効な範囲の値を設定してください。

使用例　R_PG_RTC_SetCurrentTimeの使用例を参照してください。

4.15.6 R_PG_RTC_Alarm_Control

定義

```
bool R_PG_RTC_Alarm_Control
( bool sec_enable,  bool min_enable,  bool hour_enable,  bool day_of_week_enable,
  bool day_enable,  bool month_enable,  bool year_enable  );
```

概要　アラームの有効化/無効化

生成条件　アラーム割り込みが設定されている場合

引数

bool sec_enable	秒アラーム設定（1:有効　0:無効）
bool min_enable	分アラーム設定（1:有効　0:無効）
bool hour_enable	時アラーム設定（1:有効　0:無効）
bool day_of_week_enable	曜日アラーム設定（1:有効　0:無効）
bool day_enable	日アラーム設定（1:有効　0:無効）
bool month_enable	月アラーム設定（1:有効　0:無効）
bool year_enable	年アラーム設定（1:有効　0:無効）

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細

- 秒、分、時、曜日、日、月、年アラームの有効/無効を設定します。

使用例

GUI上で以下の通り設定した場合

- アラーム割り込みを設定
- RtcAlmIntFuncをアラーム割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set(); //クロックの設定
    R_PG_Clock_Start_SUB(); //サブクロック発振開始
    R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
    R_PG_RTC_SetCurrentTime( //現在時刻の設定(0000年01月01日22時33分44秒)
        0x44, //44秒
        0x33, //33分
        0x22, //22時
        0x01, //01日
        0x01, //01月
        0x0000 //0000年
    );

    R_PG_RTC_SetAlarmTime( //アラーム時刻の設定(0000年01月01日22時33分55秒)
        0x55, //55秒
        0x33, //33分
        0x22, //22時
        0xff, //曜日(0xff : 指定した日時から自動計算する)
        0x01, //01日
        0x01, //01月
        0x0000 //0000年
    );

    R_PG_RTC_Alarm_Control( //時、分、秒アラーム有効化
        1, //秒アラーム有効
        1, //分アラーム有効
        1, //時アラーム有効
        0, //曜日アラーム無効
        0, //日アラーム無効
        0, //月アラーム無効
        0 //年アラーム無効
    );
}

void RtcAlmIntFunc(void)
{
    //アラーム割り込み処理
}
```

4.15.7 R_PG_RTC_Adjust30sec

定義　bool R_PG_RTC_Adjust30sec(void)

概要　30秒調整を行う

生成条件　アラーム割り込みが設定されている場合

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細

- 30秒調整（30秒未満は00秒に切り捨て、30秒以降は1分に桁上げ）を実行します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
	R_PG_Clock_Set(); //クロックの設定
	R_PG_Clock_Start_SUB(); //サブクロック発振開始
	R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
}

void func2(void)
{
	R_PG_RTC_Adjust30sec(); //30秒調整の実行
}
```

4.15.8 R_PG_RTC_SetPeriodicInterrupt

定義　bool R_PG_RTC_SetPeriodicInterrupt(float frequency)

概要　周期割り込みの周期設定

生成条件　周期割り込みが設定されている場合

引数

float frequency	割り込みの周波数(Hz)（有効な値: 0.5, 1, 2, 4, 16, 64, 256）

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細
- 周期割り込みの発生周期を変更します。

使用例

GUI上で以下の通り設定した場合
- 周期割り込みを設定
- RtcPrdIntFuncを周期割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
	R_PG_Clock_Set(); //クロックの設定
	R_PG_Clock_Start_SUB(); //サブクロック発振開始
	R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
}

void RtcAlmIntFunc(void)
{
	//周期割り込みの処理

	R_PG_RTC_SetPeriodicInterrupt( 4 );  //周期割り込みの発生周期を1/4秒に設定
}
```

4.15.9 R_PG_RTC_GetStatus

定義

```
bool R_PG_RTC_GetStatus
( uint8_t * seconds,  uint8_t * minutes,  uint8_t * hours,  uint8_t * day_of_week,
  uint8_t * day,      uint8_t * month,    uint16_t * year,  bool * carry,
  bool * alarm,       bool * period,      bool * adjustment,  bool * reset,
  bool * running  )
```

概要

RTCの状態を取得

引数

uint8_t * seconds	現在の秒カウンタ値の格納先
uint8_t * minutes	現在の分カウンタ値の格納先
uint8_t * hours	現在の時カウンタ値の格納先
uint8_t * day_of_week	現在の曜日カウンタ値の格納先
uint8_t * day	現在の日カウンタ値の格納先
uint8_t * month	現在の月カウンタ値の格納先
uint16_t * year	現在の年カウンタ値の格納先
bool * carry	桁上げ割り込みフラグの格納先
bool * alarm	アラーム割り込みフラグの格納先
bool * period	周期割り込みフラグの格納先
bool * adjustment	30秒調整ビットの格納先 (0:通常動作　1:調整中)
bool * reset	リセットビットの格納先 (0:通常動作　1:リセット中)
bool * running	スタートビットの格納先 (0:クロック停止　1:クロック動作中)

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_RTC.c

使用RPDL関数

R_RTC_Read

詳細

- RTCの状態を取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。
- 割り込みフラグは本関数内でクリアされます。
- 桁上げ割り込みフラグが1の場合、状態の取得中に現在時刻が変更されているため、再読み出しが必要です。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
	R_PG_Clock_Set(); //クロックの設定
	R_PG_Clock_Start_SUB(); //サブクロック発振開始
	R_PG_RTC_Start(); //RTCの設定とカウント動作の開始
}

void func2(void)
{
	do{
		//現在時刻と桁上げ割り込みフラグの取得
		bool R_PG_RTC_GetStatus(
			&seconds,	//秒
			&minutes,	//分
			&hours,		//時
			0,		//曜日
			0,		//日
			0,		//月
			0,		//年
			&carry,		//桁上げ割り込みフラグ
			0,		//アラーム割り込みフラグ
			0,		//周期割り込みフラグ
			0,		//30秒調整ビット
			0,		//リセットビット
			0		//スタートビット
		);
	} while( carry );
}
```

4.15.10R_PG_RTC_ClockOut_Disable

定義　bool R_PG_RTC_ClockOut_Disable (void)

概要　クロック出力の無効化

生成条件　RTCOUTからの1Hzクロック出力が有効な場合

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細
- RTCOUTからの1Hzクロック出力を停止します。

使用例

GUI上で以下の通り設定した場合
- RTCOUTからの1Hzクロック出力を有効に設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
	R_PG_Clock_Set(); //クロックの設定
	R_PG_Clock_Start_SUB(); //サブクロック発振開始
	R_PG_RTC_Start(); //RTCの設定、カウント動作とクロック出力の開始
}

void func2(void)
{
	R_PG_RTC_ClockOut_Disable();  //クロック出力の停止
}

void func3(void)
{
	R_PG_RTC_ClockOut_Enable();  //クロック出力の再開
}
```

4.15.11 R_PG_RTC_ClockOut_Enable

定義　bool R_PG_RTC_ClockOut_Enable (void)

概要　クロック出力の有効化

生成条件　RTCOUTからの1Hzクロック出力が有効な場合

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RTC.c

使用RPDL関数　R_RTC_Control

詳細
- RTCOUTからの1Hzクロック出力を開始します。

使用例

GUI上で以下の通り設定した場合
- RTCOUTからの1Hzクロック出力を有効に設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    R_PG_Clock_Set(); //クロックの設定
    R_PG_Clock_Start_SUB(); //サブクロック発振開始
    R_PG_RTC_Start(); //RTCの設定、カウント動作とクロック出力の開始
}

void func2(void)
{
    R_PG_RTC_ClockOut_Disable();  //クロック出力の停止
}

void func3(void)
{
    R_PG_RTC_ClockOut_Enable();  //クロック出力の再開
}
```

4.16 ウォッチドッグタイマ（WDT）

4.16.1 R_PG_Timer_Start_WDT

定義　bool R_PG_Timer_Start_WDT (void)

概要　WDTを設定しカウント動作を開始

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Timer_WDT.c

使用RPDL関数　R_WDT_Create

詳細

- WDTを初期設定し、カウント動作を開始します。
- GUI上で動作モードにインターバルタイマモードを指定した場合、本関数内でインターバルタイマ割り込みを設定します。GUI上で割り込み通知関数名を指定した場合、CPUへの割り込みが発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。
  void *〈割り込み通知関数名〉*(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上で以下の通り設定した場合

- 動作モードをインターバルタイマモードに設定
- WdtIntFuncをインターバルタイマ割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_Timer_Start_WDT();   //WDTの設定とカウント動作の開始
}

void WdtIntFunc(void)
{
    //WDTのカウンタオーバフロー時処理
}
```

4.16.2 R_PG_Timer_HaltCount_WDT

定義 bool R_PG_Timer_HaltCount_WDT (void)

概要 WDTのカウント動作を停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_Timer_WDT.c

使用RPDL関数 R_WDT_Control

詳細
- WDTのカウント動作を停止します。
- カウント動作を再開するにはR_PG_Timer_Start_WDTを呼び出してください。

使用例 GUI上で以下の通り設定した場合
- 動作モードをインターバルタイマモードに設定
- WdtIntFuncをインターバルタイマ割り込み通知関数名に指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_Timer_Start_WDT();   //WDTの設定とカウント動作の開始
}

void WdtIntFunc(void)
{
    R_PG_Timer_HaltCount_WDT();    //WDTのカウント動作停止

    //WDTのカウンタオーバフロー時処理

    R_PG_Timer_Start_WDT();   //WDTの設定とカウント動作の再開
}
```

4.16.3 R_PG_Timer_ResetCounter_WDT

定義 bool R_PG_Timer_ResetCounter_WDT(void)

概要 カウンタのリセット

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_Timer_WDT.c

使用RPDL関数 R_WDT_Control

詳細

- WDTのカウンタをリセットします

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_Timer_Start_WDT();   //WDTの設定とカウント動作の開始
}

void func2(void)
{
    R_PG_Timer_ResetCounter_WDT();    //WDTのカウンタをリセット
}
```

4.16.4 R_PG_Timer_ClearOverflowFlag_WDT

定義 bool R_PG_Timer_ClearOverflowFlag_WDT (bool* ov)

概要 オーバフローフラグの取得とクリア

引数

bool* ov	オーバフローフラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_Timer_WDT.c

使用RPDL関数 R_WDT_Control

詳細
- オーバフローフラグの取得し、クリアします。
- フラグを取得しない場合は引数に0を指定してください。

使用例

GUI上で以下の通り設定した場合
- 動作モードをインターバルタイマモードに設定
- インターバルタイマ割り込み優先レベルを0に設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool ov;

void func(void)
{
    R_PG_Clock_Set();   //クロックの設定
    R_PG_Timer_Start_WDT();   //WDTの設定とカウント動作の開始

    do{
        R_PG_Timer_ClearOverflowFlag_WDT( &ov ); //オーバフローフラグの取得
    }while( !ov );

    //オーバフロー発生時処理
}
```

4.17 独立ウォッチドッグタイマ（IWDT）

4.17.1 R_PG_Timer_Set_IWDT

定義　bool R_PG_Timer_Set_IWDT (void)

概要　IWDTの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_Timer_IWDT.c

使用RPDL関数　R_IWDT_Set

詳細

- IWDTを設定します。
- カウント動作はカウンタのリフレッシュにより開始します。本関数を呼び出した後、R_PG_Timer_RefreshCounter_IWDTを呼び出すことによりカウント動作が開始します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t output_val;

void func1(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //IWDTの設定
    R_PG_Timer_Set_WDT();

    //カウンタのリフレッシュによりカウント動作開始
    R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
    R_PG_Timer_RefreshCounter_IWDT();   //カウンタのリフレッシュ
}
```

4.17.2 R_PG_Timer_RefreshCounter_IWDT

定義 bool R_PG_Timer_RefreshCounter_IWDT (void)

概要 カウンタのリフレッシュ

引数 なし

戻り値

true	リフレッシュに成功した場合
false	リフレッシュに失敗した場合

出力先ファイル R_PG_Timer_IWDT.c

使用RPDL関数 R_IWDT_Control

詳細
- IWDTのカウンタをリフレッシュします。
- カウント動作を開始するにはR_PG_Timer_Set_IWDTによりIWDTを設定した後、本関数を呼び出してください。
- カウント動作開始後、本関数によりアンダフロー発生までにカウンタをリフレッシュしてください。

使用例 R_PG_Timer_Set_IWDTの使用例を参照してください。

4.17.3 R_PG_Timer_GetCounterValue_IWDT

定義 bool R_PG_Timer_GetCounterValue_IWDT(uint16_t * counter_val)

概要 カウンタ値の取得

引数

uint16_t * counter_val	カウンタ値の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_Timer_IWDT.c

使用RPDL関数 R_IWDT_Read

詳細

- IWDTのカウンタ値を取得します。
- 本関数内でアンダフローフラグはクリアされます。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t counter_val;

void func1(void)
{
	//クロックの設定
	R_PG_Clock_Set();

	//IWDTの設定
	R_PG_Timer_Set_WDT();

	//カウンタのリフレッシュによりカウント動作開始
	R_PG_Timer_RefreshCounter_IWDT();
}

void func2(void)
{
	R_PG_Timer_GetCounterValue_IWDT( &counter_val );

	if( counter_val < 0x1000){
		//カウンタのリフレッシュ
		R_PG_Timer_RefreshCounter_IWDT();	//カウンタのリフレッシュ
	}
}
```

4.17.4 R_PG_Timer_ClearUnderflowFlag_IWDT

定義 bool R_PG_Timer_ClearUnderflowFlag_IWDT(bool * un)

概要 アンダフローフラグの取得とクリア

引数

bool * un	アンダフローフラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_Timer_IWDT.c

使用RPDL関数 R_IWDT_Read

詳細

- アンダフローフラグを取得し、クリアします。
- フラグを取得しない場合は引数に0を指定してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool un;

void func(void)
{
    R_PG_Timer_ClearUnderflowFlag_IWDT ( &un );
    if(un){
        //アンダフロー発生によるリセット後処理
    }
}
```

4.18 シリアルコミュニケーションインタフェース（SCIa）

4.18.1 R_PG_SCI_Set_C〈*チャネル番号*〉

定義　bool R_PG_SCI_Set_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルI/Oチャネルの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Create

詳細
- SCIチャネルのモジュールストップ状態を解除して初期設定し、使用する端子の入出力方向、入力バッファを設定します。
  本関数を使用する場合、あらかじめR_PG_Clock_Setによりクロックを設定してください。
- GUI上で通知関数名を指定した場合、対応するイベントが発生すると指定した名前の関
- 数が呼び出されます。通知関数は次の定義で作成してください。
  void 〈割り込み通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- 本関数によりTXD5端子からCS4またはCS7の外部バス出力を停止することはできません。SCI5チャネルを使用する場合、TXD5端子をCS4#_DまたはCS7#_D端子として使用することはできません。

使用例　SCI0をGUI上で設定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();       //クロックの設定
    R_PG_SCI_Set_C0();      //SCI0を設定
}
```

4.18.2 R_PG_SCI_StartSending_C〈*チャネル番号*〉

定義　bool R_PG_SCI_StartSending_C〈*チャネル番号*〉(uint8_t * data, uint16_t count)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルデータの送信開始

生成条件
- GUI上でSCIチャネルの送信機能を設定
- データ送信方法に“全データの送信完了を関数呼び出しで通知する”を選択

引数

uint8_t * data	送信するデータの先頭のアドレス
uint16_t count	送信するデータ数 0を指定した場合はNULLのデータまで送信します。

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Send

詳細
- シリアルデータを送信します。
- 本関数はGUI上でデータ送信方法に“全データの送信完了を関数呼び出しで通知する”が選択されている場合に出力されます。
- 本関数はすぐにリターンし、指定した数のデータ送信完了時に指定した名前の関数が呼ばれます。通知関数は次の定義で作成してください。
  void 〈*通知関数名*〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- R_PG_SCI_GetSentDataCount_C〈*チャネル番号*〉により送信済みデータ数を取得することができます。R_PG_SCI_StopCommunication_C〈*チャネル番号*〉により、最終バイトの送信完了を待たずに送信を中断することができます。
- 65536バイトのデータが送信されると、0番目のデータに戻ります。

使用例　GUI上でSCI0の送信終了通知関数名にSci0TrFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
uint8_t data[255];
void func(void)
{
    R_PG_Clock_Set();           //クロックの設定
    R_PG_SCI_Set_C0();          //SCI0を設定
    R_PG_SCI_Send_C0(data, 255);        //255バイトのデータを送信する
}
//全データが送信されると呼び出される送信終了通知関数
void Sci0TrFunc(void)
{
    //SCI0を停止
    R_PG_SCI_StopModule_C0();
}
```

4.18.3 R_PG_SCI_SendAllData_C〈*チャネル番号*〉

定義

bool R_PG_SCI_SendAllData_C〈*チャネル番号*〉(uint8_t * data, uint16_t count)

〈*チャネル番号*〉: 0～3、5～6

概要

シリアルデータを全て送信

生成条件

- GUI上でSCIチャネルの送信機能を設定
- データ送信方法に“全データの送信完了を関数呼び出しで通知する”以外を選択

引数

uint8_t * data	送信するデータの先頭のアドレス
uint16_t count	送信するデータ数 0を指定した場合はNULLのデータまで送信します。

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_SCI_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数

R_SCI_Send

詳細

- シリアルデータを送信します。
- 本関数はGUI上でデータ送信方法に“全データの送信完了を関数呼び出しで通知する”以外が選択されている場合に出力されます。
- 指定した数のデータ送信完了まで関数内でウェイトします。
- 65536バイトのデータが送信されると、0番目のデータに戻ります。
- 送信データをDMACで転送する場合の使用方法は、R_PG_DMAC_Set_C〈*チャネル番号*〉の使用例2を参照してください。

使用例

GUI上でSCI0のデータ送信方法に“全データの送信完了まで待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_SendAllData_C0(data, 255);          //255バイトのデータを送信する
    R_PG_SCI_StopModule_C0();          //SCI0を停止
}
```

4.18.4 R_PG_SCI_GetSentDataCount_C〈*チャネル番号*〉

定義　bool R_PG_SCI_GetSentDataCount_C〈*チャネル番号*〉(uint16_t * count)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルデータの送信数取得

生成条件　GUI上でSCIチャネルの送信機能を設定し、データ送信方法に“全データの送信完了を関数呼び出しで通知する”を選択

引数

uint16_t * count	現在の送信処理で送信されたデータ数の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_GetStatus

詳細
- GUI上で送信終了通知に“最終バイトの送信終了を関数呼び出しで通知する”が選択されている場合、本関数により送信済みデータ数を取得することができます。

使用例　GUI上でSCI0の送信機能を設定
送信終了通知関数名にSci0TrFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_Send_C0(data, 255);           //255バイトのデータを送信する
}

//全データが送信されると呼び出される送信終了通知関数
void Sci0TrFunc(void)
{
    R_PG_SCI_StopModule_C0();      //SCI0を停止
}

//送信済みデータ数をチェックし、送信を中断する関数
void func_terminate_SCI(void)
{
    uint8_t count;
    R_PG_SCI_GetSentDataCount_C0(&count);   //送信済みデータ数を取得

    if( count > 32 ){
        R_PG_SCI_StopCommunication_C0();      //送信を中断
    }
}
```

4.18.5 R_PG_SCI_StartReceiving_C〈*チャネル番号*〉

定義　bool R_PG_SCI_StartReceiving_C〈*チャネル番号*〉(uint8_t * data, uint16_t count)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルデータの受信開始

生成条件
- GUI上でSCIチャネルの受信機能を設定
- データ受信方法に“全データの受信完了を関数呼び出しで通知する”を選択

引数

uint8_t * data	受信したデータの格納先の先頭のアドレス
uint16_t count	受信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Receive

詳細
- シリアルデータを受信します。
- 本関数はGUI上でデータ受信方法に“全データの受信完了を関数呼び出しで通知する”が選択されている場合に生成されます。
- 本関数はすぐにリターンし、指定した数のデータ受信完了時に指定した名前の関数が呼ばれます。通知関数は次の定義で作成してください。
  void 〈*通知関数名*〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- R_PG_SCI_GetReceivedDataCount_C 〈*チャネル番号*〉により受信済みデータ数を取得することができます。R_PG_SCI_StopCommunication_C〈*チャネル番号*〉 により、最終バイトの受信完了を待たずに受信を中断することができます。
- 最大受信データ数は65535です。

使用例　GUI上でSCI0の受信機能を設定
受信終了通知関数名にSci0ReFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_Receive_C0(data, 255);          //255バイトのデータを受信する
}
//全データを受信すると呼び出される受信終了通知関数
void Sci0ReFunc(void)
{
    //SCI0を停止
    R_PG_SCI_StopModule_C0();
}
```

4.18.6 R_PG_SCI_ReceiveAllData_C〈*チャネル番号*〉

定義

bool R_PG_SCI_ReceiveAllData_C〈*チャネル番号*〉(uint8_t * data, uint16_t count)

〈*チャネル番号*〉: 0～3、5～6

概要

シリアルデータを全て受信

生成条件

- GUI上でSCIチャネルの受信機能を設定
- データ受信方法に“全データの受信完了を関数呼び出しで通知する”以外を選択

引数

uint8_t * data	受信したデータの格納先の先頭のアドレス
uint16_t count	受信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_SCI_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数

R_SCI_Receive

詳細

- シリアルデータを受信します。
- 本関数はGUI上でデータ受信方法に “全データの受信完了を関数呼び出しで通知する” 以外が選択されている場合に出力されます。
- 本関数は指定した数のデータ受信完了までウェイトします。
- 最大受信データ数は65535です。
- 受信データをDMACで転送する場合の使用方法は、R_PG_DMAC_Set_C〈*チャネル番号*〉の使用例3を参照してください。

使用例

GUI上でSCI0の受信機能を設定

データ受信方法に“全データの受信完了まで待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_Receive_C0(data, 255);          //255バイトのデータを受信する
    R_PG_SCI_StopModule_C0();          //SCI0を停止
}
```

4.18.7 R_PG_SCI_StopCommunication_C〈*チャネル番号*〉

定義　R_PG_SCI_StopCommunication_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルデータの送受信停止

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Control

詳細
- シリアルの送受信を停止します。
- GUI上でデータ送信方法に“全データの送信完了を関数呼び出しで通知する”が選択されている場合、本関数によりR_PG_SCI_StartSending_C〈*チャネル番号*〉で指定した全データの送信完了を待たずに送信を中断することができます。
- GUI上でデータ受信方法に“全データの受信完了を関数呼び出しで通知する”が選択されている場合、本関数によりR_PG_SCI_StartReceiving_C〈*チャネル番号*〉で指定した全データの受信完了を待たずに受信を中断することができます。

使用例　GUI上でSCI0の受信機能を設定
受信終了通知関数名にSci0ReFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
uint8_t data[255];
void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_Receive_C0(data, 255);          //255バイトのデータを送信する
}
//全データを受信すると呼び出される受信終了通知関数
void Sci0ReFunc(void)
{
    R_PG_SCI_StopModule_C0();      //SCI0を停止
}
//受信済みデータ数をチェックし、受信を中断する関数
void func_terminate_SCI(void)
{
    uint8_t count;
    R_PG_SCI_GetReceivedDataCount_C0(&count);  //受信済みデータ数を取得
    if( count > 32 ){
        R_PG_SCI_ StopCommunication_C0();      //受信を中断
    }
}
```

4.18.8 R_PG_SCI_GetReceivedDataCount_C〈*チャネル番号*〉

定義　bool R_PG_SCI_GetReceivedDataCount_C〈*チャネル番号*〉(uint16_t * count)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルデータの受信数取得

生成条件　GUI上でSCIチャネルの受信機能が設定され、データ受信方法に“全データの受信完了を関数呼び出しで通知する”

引数

uint16_t * count	現在の受信処理で受信したデータ数の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_GetStatus

詳細
- GUI上でデータ受信方法に“全データの受信完了を関数呼び出しで通知する”が選択されている場合、本関数により受信済みデータ数を取得することができます。

使用例　GUI上でSCI0の受信機能を設定
受信終了通知関数名にSci0ReFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_Receive_C0(data, 255);          //255バイトのデータを送信する
}

//全データを受信すると呼び出される受信終了通知関数
void Sci0ReFunc(void)
{
    R_PG_SCI_StopModule_C0();      //SCI0を停止
}

//受信済みデータ数をチェックし、受信を中断する関数
void func_terminate_SCI(void)
{
    uint16_t count;
    R_PG_SCI_GetReceivedDataCount_C0(&count);  //受信済みデータ数を取得

    if( count > 32 ){
        R_PG_SCI_StopReceiving_C0();      //受信を中断
    }
}
```

4.18.9 R_PG_SCI_GetReceptionErrorFlag_C〈チャネル番号〉

定義

bool R_PG_SCI_GetReceptionErrorFlag_C〈チャネル番号〉
(bool * parity, bool * framing, bool * overrun)
〈チャネル番号〉: 0～3、5～6

概要

シリアル受信エラーフラグの取得

生成条件

GUI上でSCIチャネルの受信機能を設定

引数

bool * parity	パリティエラーフラグ格納先
bool * framing	フレーミングエラーフラグ格納先
bool * overrun	オーバランエラーフラグ格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_SCI_C〈チャネル番号〉.c
〈チャネル番号〉: 0～3、5～6

使用RPDL関数

R_SCI_GetStatus

詳細

- 受信エラーフラグを取得します。
- 取得しないフラグは0を設定してください。
- 検出したエラーのフラグには1が設定されます。

使用例

GUI上でSCI0の受信機能を設定
受信終了通知関数名にSci0ReFuncを指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[255];

//受信エラーフラグ
bool parity;
bool framing;
bool overrun;

void func(void)
{
	R_PG_Clock_Set();		//クロックの設定
	R_PG_SCI_Set_C0();		//SCI0を設定
	R_PG_SCI_Receive_C0(data, 1);		//1バイトのデータを送信する
}

//全データを受信すると呼び出される受信終了通知関数
void Sci0ReFunc(void)
{
	//受信エラーを取得
	R_PG_SCI_GetReceptionErrorFlag_C0( &parity, &framing, & overrun );
}
```

4.18.10R_PG_SCI_GetTransmitStatus_C〈*チャネル番号*〉

定義　bool R_PG_SCI_GetTransmitStatus_C〈*チャネル番号*〉(bool * complete)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルデータ送信状態の取得

生成条件　GUI上でSCIチャネルの送信機能を設定

引数

bool * complete	送信終了フラグ格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_GetStatus

詳細

- シリアルデータの送信状態を取得します。

送信終了フラグ

0	送信中
1	送信終了

使用例　R_PG_DMAC_Set_C〈*チャネル番号*〉 の使用例2を参照してください。

4.18.11 R_PG_SCI_SendTargetStationID_C〈*チャネル番号*〉

定義　bool R_PG_SCI_SendTargetStationID_C〈*チャネル番号*〉(uint8_t id)
〈*チャネル番号*〉: 0～3、5～6

概要　データ送信先IDの送信

生成条件
- GUI上でSCIチャネルの送信機能を設定
- 調歩同期式通信方式でマルチプロセッサ通信機能を有効に設定

引数

uint8_t id	送信するIDコード (0～255)

戻り値

true	送信に成功した場合
false	送信に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Send

詳細
- マルチプロセッサモードのID送信サイクルを生成し、データ送信先の受信局IDコードを出力します。
- 本関数はID送信サイクル終了までウェイトします。

使用例　GUI上で以下の通り設定した場合
- SCI2チャネルの送信機能を設定
- 調歩同期式通信方式でマルチプロセッサ通信機能を有効に設定
- データ送信方法に“全データの送信完了まで待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
uint8_t data[10] = ABCDEFGHIJ;

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C2();             //SCI2を設定
    R_PG_SCI_SendTargetStationID_C2( 5 );      //IDコードの送信 (ID:5)
    R_PG_SCI_SendAllData_C2( data, 10 );       //データの送信
}
```

4.18.12R_PG_SCI_ReceiveStationID_C〈*チャネル番号*〉

定義　bool R_PG_SCI_ReceiveStationID_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0～3、5～6

概要　自局IDと一致するIDコードの受信

生成条件
- GUI上でSCIチャネルの受信機能を設定
- 調歩同期式通信方式でマルチプロセッサ通信機能を有効に設定

引数　なし

戻り値

true	受信に成功した場合
false	受信に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Receive

詳細
- 本関数は自局のIDと一致するIDコードを受信するまでウェイトします。

使用例　GUI上で以下の通り設定した場合
- SCI5チャネルの受信機能を設定
- 調歩同期式通信方式でマルチプロセッサ通信機能を有効に設定
- データ受信方法に“全データの受信完了を関数呼び出しで通知する”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[10];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C5();             //SCI5設定
    R_PG_SCI_ReceiveStationID_C5();    //IDの受信を待つ
    R_PG_SCI_StartReceiving_C5( data, 10 );    //受信開始
}
```

4.18.13R_PG_SCI_StopModule_C〈*チャネル番号*〉

定義　bool R_PG_SCI_StopModule_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0～3、5～6

概要　シリアルI/Oチャネルの停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_SCI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0～3、5～6

使用RPDL関数　R_SCI_Destroy

詳細
- SCIのチャネルを停止し、モジュールストップ状態に移行します。

使用例
GUI上で以下の通り設定した場合
- GUI上でSCI0の受信機能を設定
- データ受信方法に“最終バイトの受信終了まで受信関数内で待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint8_t data[255];

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_SCI_Set_C0();             //SCI0を設定
    R_PG_SCI_Receive_C0(data, 255);          //255バイトのデータを受信する
    R_PG_SCI_StopModule_C0();          //SCI0を停止
}
```

4.19 CRC演算器（CRC）

4.19.1 R_PG_CRC_Set

定義　bool R_PG_CRC_Set(void)

概要　CRC演算器の設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_CRC.c

使用RPDL関数　R_CRC_Create

詳細

- CRC演算器のモジュールストップ状態を解除して初期設定します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t data;

void func(void)
{
    R_PG_CRC_Set();   //CRC演算器の設定
    R_PG_CRC_InputData(0xf0);     //ペイロードデータ入力
    R_PG_CRC_InputData(0x8f);     //前半チェックサム入力
    R_PG_CRC_InputData(0x7f);     //後半チェックサム入力
    R_PG_CRC_GetResult (&data);   //演算結果取得
    R_PG_CRC_StopModule();   //CRC演算器停止
}
```

4.19.2 R_PG_CRC_InputData

定義 bool R_PG_CRC_InputData (uint8_t data)

概要 データの入力

引数

uint8_t data	入力するデータ

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_CRC.c

使用RPDL関数 R_CRC_Write

詳細
- CRCデータ入力レジスタにデータを設定します。

使用例 R_PG_CRC_Setの使用例を参照してください。

4.19.3 R_PG_CRC_GetResult

定義 bool R_PG_CRC_GetResult (uint16_t * result)

概要 演算結果の取得

引数

uint16_t * result	演算結果の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_CRC.c

使用RPDL関数 R_CRC_Read

詳細
- 演算結果を取得します。

使用例 R_PG_CRC_Setの使用例を参照してください。

4.19.4 R_PG_CRC_StopModule

定義 bool R_PG_CRC_StopModule(void)

概要 CRC演算器の停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_CRC.c

使用RPDL関数 R_CRC_Destroy

詳細
- CRC演算器を停止し、モジュールストップ状態に移行します。

使用例 R_PG_CRC_Setの使用例を参照してください。

4.20 I2Cバスインタフェース（RIIC）

4.20.1 R_PG_I2C_Set_C〈*チャネル番号*〉

定義　bool R_PG_I2C_Set_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0, 1

概要　I2Cバスインタフェースチャネルの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_Create

詳細
- I2Cバスインタフェースチャネルのモジュールストップ状態を解除して初期設定し、使用する端子の入出力方向、入力バッファを設定します。
- 本関数を使用する場合、あらかじめR_PG_Clock_Setによりクロックを設定してください。

使用例　GUI上でRIIC0を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();      //クロックの設定
    R_PG_I2C_Set_C0();     //RIIC0を設定
}
```

4.20.2 R_PG_I2C_MasterReceive_C〈チャネル番号〉

定義 bool R_PG_I2C_MasterReceive_C〈チャネル番号〉
(bool addr_10bit, uint16_t slave, uint8_t* data, uint16_t count)
〈チャネル番号〉: 0, 1

概要 マスタのデータ受信

生成条件 マスタ機能を使用

引数

bool addr_10bit	スレーブアドレスフォーマット（1:10ビット　0:7ビット）
uint16_t slave	スレーブアドレス
uint8_t* data	受信したデータの格納先の先頭のアドレス
uint16_t count	受信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_I2C_C〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_IIC_MasterReceive

詳細

- スレーブからデータを読み出します。指定した数のデータを受信するとSTOP条件を生成し転送を終了します。
- GUI上でマスタ受信方法に“全データの受信完了まで待つ”が選択されている場合、本関数は転送終了までウェイトします。GUI上でマスタ受信方法に“全データの受信完了を関数呼び出しで通知する”が選択されている場合、本関数はすぐにリターンし、転送終了時に指定した名前の関数が呼ばれます。通知関数は次の定義で作成してください。
  void 〈通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- 通信の最初にSTART条件が生成されます。前回の転送でSTOP条件が生成されていない場合は反復START条件が生成されます。
- スレーブアドレスは、7ビットアドレスの場合は指定した値の7～1ビットが出力されます。10ビットアドレスの場合は10～1ビットが出力されます。
- R_PG_I2C_GetReceivedDataCount_C〈チャネル番号〉により受信済みデータ数を取得することができます。
- 10ビットアドレスを使用する場合、GUI上のマスタ受信方法は“全データの受信完了を関数で通知する”以外を選択してください。

使用例 GUI上で以下の通り設定した場合

- RIIC0 をマスタとして使用
- マスタ受信方法に“全データの受信完了まで待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//受信データの格納先
uint8_t iic_data[10];

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ受信
    R_PG_I2C_MasterReceive_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //受信データの格納先アドレス
        10          //受信データ数
    );

    //RIIC0を停止
    R_PG_I2C_StopModule_C0();
}
```

4.20.3 R_PG_I2C_MasterReceiveLast_C〈*チャネル番号*〉

定義

bool R_PG_I2C_MasterReceiveLast_C〈*チャネル番号*〉(uint8_t* data)

〈*チャネル番号*〉: 0, 1

概要

マスタのデータ受信終了

生成条件

- マスタ機能を使用
- GUI上でマスタ受信方法にDMACまたはDTCによる転送を選択

引数

uint8_t* data	受信したデータの格納先のアドレス

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル

R_PG_I2C_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_IIC_MasterReceiveLast

詳細

- 本関数はGUI上で[マスタ受信方法]に[受信データをDMACで転送する]または[受信データをDTCで転送する] が選択されている場合に出力されます。
- マスタのデータ受信において、受信したデータをDMACまたはDTCで転送する場合、本関数を呼び出すことにより、NACKとストップ条件を発行して受信を終了します。
- DMACまたはDTCの転送終了時に受信を終了する場合は、DMACまたはDTCの転送終了割り込み通知関数から本関数を呼び出してください。
- 本関数内で、受信データレジスタから受信データを1バイト追加取得します。
- 受信中に検出したイベントや受信データ数は、R_PG_I2C_GetEvent_CnおよびR_PG_I2C_GetReceivedDataCount_Cnで取得することができます。

使用例 GUI上で以下の通り設定し、マスタが受信したデータをDMACで転送する場合

- RIIC0の設定でマスタ受信方法に[受信データをDMACで転送する]を指定。
- DMAC0の設定で以下通り設定。

転送開始要因 ： ICRXI0(RIIC0受信データフル割り込み)
転送方式 ： 単一オペランド転送
単位データサイズ ： 1byte
1オペランドのデータ数 ：1
転送データサイズ ： RIIC0が受信するデータ数
転送元スタートアドレス ： RIIC0受信データレジスタのアドレス
転送先スタートアドレス ： RIIC0受信データの転送先開始アドレス
DMAC0転送終了割り込み通知関数名 ： Dmac0IntFunc

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void Dmac0IntFunc(){
    uint8_t data; //追加データの格納先

    //NACK, STOP条件を発行し転送終了
    R_PG_I2C_MasterReceiveLast( &data );
}

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //DMAC0を設定
    R_PG_DMAC_Set_C0();

    //DMAC0を転送開始トリガ入力待ち状態にする
    R_PG_DMAC_Activate_C0();

    //マスタ受信
    R_PG_I2C_MasterReceive_C0(
        0,      //スレーブアドレスフォーマット
        6,      //スレーブアドレス
        PDL_NO_PTR, //受信データ格納先 (DMAC転送の場合はPDL_NO_PTR)
        0       //受信データ数 (DMAC転送の場合は0)
    );
}
```

4.20.4 R_PG_I2C_MasterSend_C〈チャネル番号〉

定義 bool R_PG_I2C_MasterSend_C〈チャネル番号〉
(bool addr_10bit, uint16_t slave, uint8_t* data, uint16_t count)
〈チャネル番号〉: 0, 1

概要 マスタのデータ送信

生成条件 マスタ機能を使用

引数

bool addr_10bit	スレーブアドレスフォーマット (1:10ビット 0:7ビット)
uint16_t slave	スレーブアドレス
uint8_t* data	送信するデータの格納先の先頭のアドレス
uint16_t count	送信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_I2C_C〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_IIC_MasterSend

詳細

- スレーブにデータを送信します。指定した数のデータを送信するとSTOP条件を生成し転送を終了します。
- GUI上でマスタ送信方法に“全データの送信完了まで待つ”が選択されている場合、本関数は転送終了または他のイベント検出までウェイトします。検出したイベントはR_PG_I2C_GetEvent_C〈チャネル番号〉により取得できます。GUI上でマスタ送信方法に“全データの送信完了を関数呼び出しで通知する”が選択されている場合、本関数はすぐにリターンし、転送終了時に指定した名前の関数が呼ばれます。通知関数は次の定義で作成してください。
  void 〈通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- 通信の最初にSTART条件が生成されます。前回の転送でSTOP条件が生成されていない場合は反復START条件が生成されます。
- スレーブアドレスは、7ビットアドレスの場合は指定した値の7～1ビットが出力されます。10ビットアドレスの場合は10～1ビットが出力されます。
- R_PG_I2C_GetSentDataCount_C〈チャネル番号〉により送信済みデータ数を取得することができます。
- 10ビットアドレスを使用する場合、GUI上のマスタ送信方法は“全データの送信完了を関数で通知する”以外に設定してください。

使用例 GUI上で以下の通り設定した場合

- RIIC0 をマスタとして使用
- マスタ送信方法に“全データの送信完了まで待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//送信データの格納先
uint8_t iic_data[10];

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ送信
    R_PG_I2C_MasterSend_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //送信データの格納先アドレス
        10          //送信データ数
    );

    //RIIC0を停止
    R_PG_I2C_StopModule_C0();
}
```

4.20.5 R_PG_I2C_MasterSendWithoutStop_C〈チャネル番号〉

定義 R_PG_I2C_MasterSendWithoutStop_C〈チャネル番号〉
(bool addr_10bit, uint16_t slave, uint8_t* data, uint16_t count)
〈チャネル番号〉: 0, 1

概要 マスタのデータ送信（STOP条件無し）

生成条件 マスタ機能を使用

引数

bool addr_10bit	スレーブアドレスフォーマット（1:10ビット　0:7ビット）
uint16_t slave	スレーブアドレス
uint8_t* data	送信するデータの格納先の先頭のアドレス
uint16_t count	送信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_I2C_C〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_IIC_MasterSend

詳細

- スレーブにデータを送信します。転送が終了してもSTOP条件を生成しません。本関数によるデータの送信後再び転送を開始した場合は反復START条件が生成されます。STOP条件を生成するにはR_PG_I2C_GenerateStopCondition_C〈チャネル番号〉を呼び出してください。
- GUI上でマスタ送信方法に“全データの送信完了まで待つ”が選択されている場合、本関数は転送終了または他のイベント検出までウェイトします。検出したイベントはR_PG_I2C_GetEvent_C〈チャネル番号〉により取得できます。GUI上でマスタ送信方法に“全データの送信完了を関数呼び出しで通知する”が選択されている場合、本関数はすぐにリターンし、転送終了時に指定した名前の関数が呼ばれます。通知関数は次の定義で作成してください。
  void 〈通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
- 通信の最初にSTART条件が生成されます。前回の転送でSTOP条件が生成されていない場合は反復START条件が生成されます。
- スレーブアドレスは、7ビットアドレスの場合は指定した値の7～1ビットが出力されます。10ビットアドレスの場合は10～1ビットが出力されます。
- R_PG_I2C_GetSentDataCount_C〈チャネル番号〉により送信済みデータ数を取得することができます。
- 10ビットアドレスを使用する場合、GUI上のマスタ送信方法は“全データの送信完了を関数で通知する”以外に設定してください。

使用例 GUI上で以下の通り設定した場合

- RIIC0 をマスタとして使用
- マスタ送信方法に“全データの送信完了を関数呼び出しで通知する”を選択
- マスタ送信の通知関数名に IIC0MasterTrFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//送信データの格納先
uint8_t iic_data[10];

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ送信
    R_PG_I2C_MasterSendWithoutStop_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //送信データの格納先アドレス
        10          //送信データ数
    );
}

void IIC0MasterTrFunc(void)
{
    //STOP条件を生成
    R_PG_I2C_GenerateStopCondition_C0();

    //RIIC0を停止
    R_PG_I2C_StopModule_C0();
}
```

4.20.6 R_PG_I2C_GenerateStopCondition_C〈*チャネル番号*〉

定義 R_PG_I2C_GenerateStopCondition_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0, 1

概要 マスタのSTOP条件生成

生成条件 マスタ機能を使用

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_I2C_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_IIC_Control

詳細
- R_PG_I2C_MasterSendWithoutStop_C〈*チャネル番号*〉 により転送を開始した場合、STOP条件を生成することができます。

使用例 GUI上で以下の通り設定した場合
- RIIC0 をマスタとして使用
- マスタ送信方法に “全データの送信完了を関数呼び出しで通知する” を選択
- マスタ送信の通知関数名に IIC0MasterTrFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//送信データの格納先
uint8_t iic_data[10];

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ送信
    R_PG_I2C_MasterSendWithoutStop_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //送信データの格納先アドレス
        10          //送信データ数
    );
}

void IIC0MasterTrFunc(void)
{
    //STOP条件を生成
    R_PG_I2C_GenerateStopCondition_C0();

    //RIIC0を停止
    R_PG_I2C_StopModule_C0();
}
```

4.20.7 R_PG_I2C_GetBusState_C〈*チャネル番号*〉

定義 R_PG_I2C_GetBusState_C〈*チャネル番号*〉(bool *busy)

〈*チャネル番号*〉: 0, 1

概要 バス状態の取得

生成条件 マスタ機能を使用

引数

bool *busy	バスビジー検出フラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル R_PG_I2C_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_IIC_GetStatus

詳細

- バスビジー検出フラグを取得します。

バスビジー検出フラグ

0	バスが開放状態（バスフリー状態）
1	バスが占有状態（バスビジー状態またはバスフリーの期間中）

使用例 GUI上で以下の通り設定した場合

- RIIC0 をマスタとして使用

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
//送信データの格納先
uint8_t iic_data[10];
//バスビジー検出フラグの格納先
bool busy;
void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();
    //RIIC0を設定
    R_PG_I2C_Set_C0();
    //バスフリー状態を待つ
    do{
        R_PG_I2C_GetBusState_C0( & busy );
    } while( busy );
    //マスタ送信
    R_PG_I2C_MasterSend_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //送信データの格納先アドレス
        10          //送信データ数
    );
}
```

4.20.8 R_PG_I2C_SlaveMonitor_C〈チャネル番号〉

定義 R_PG_I2C_SlaveMonitor_C〈チャネル番号〉(uint8_t *data, uint16_t count)
〈チャネル番号〉: 0, 1

概要 スレーブのバス監視

生成条件 スレーブ機能を使用

引数

uint8_t *data	受信したデータの格納先の先頭のアドレス
uint16_t count	受信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_I2C_C〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_IIC_SlaveMonitor

詳細

- マスタからのアクセスを監視します。
- GUI上でスレーブモニタ方法に“全データの受信完了、スレーブリード要求、ストップ条件検出を関数呼び出しで通知する”が選択されている場合、マスタからの読み出し要求またはマスタからの受信後にSTOP条件を検出すると、指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。
  void 〈通知関数名〉(void)
  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。
  GUI上でスレーブモニタ方法に“全データの受信完了、スレーブリード要求、ストップ条件検出まで待つ”が選択されている場合、本関数はマスタからの読み出し要求またはマスタからの受信後にSTOP条件を検出するまでウェイトします。
- マスタからデータが送信された場合は指定した領域に受信データが格納されます。受信データ量が格納領域を上回らないよう、受信データ数設定してください。
  指定したデータ数を上回るデータがマスタから送信された場合はNACKを生成します。
- R_PG_I2C_GetTR_C〈チャネル番号〉 により送信/受信モードを取得することができます。マスタから送信(読み出し)が要求された場合、R_PG_I2C_SlaveSend_C〈チャネル番号〉 によりデータを送信できます。
- 検出したスレーブアドレスを取得するには R_PG_I2C_GetDetectedAddress_C〈チャネル番号〉 を使用してください。START条件、STOP条件等の検出イベントを取得するには R_PG_I2C_GetEvent_C〈チャネル番号〉 を使用してください。
- 10ビットアドレスを使用する場合、GUI上のスレーブモニタ方法は “全データの受信完了、スレーブリード要求、ストップ条件検出を関数呼び出しで通知する” 以外に設定してください。

使用例 GUI上で以下の通り設定した場合

・ RIIC0 をスレーブとして使用

スレーブモニタの通知関数名に IIC0SlaveFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"
//受信データの格納先
uint8_t iic_data_re[10];
//送信データの格納先(スレーブアドレス0)
uint8_t iic_data_tr_0[10];
//送信データの格納先(スレーブアドレス1)
uint8_t iic_data_tr_1[10];
void func(void)
{
	//クロックの設定
	R_PG_Clock_Set();
	//RIIC0を設定
	R_PG_I2C_Set_C0();
	//スレーブモニタ
	R_PG_I2C_SlaveMonitor_C0(
		iic_data_re,	//受信データの格納先アドレス
		10		//受信データ数
	);
}
void IIC0SlaveFunc(void)
{
	bool transmit, start, stop;
	bool addr0, addr1;
	//イベントを取得する
	R_PG_I2C_GetEvent_C0(0, &stop, &start, 0, 0);
	//送受信モードを取得する
	R_PG_I2C_GetTR_C0(&transmit);
	//検出アドレスを取得する
	R_PG_I2C_GetDetectedAddress_C0(&addr0, &addr1, 0, 0, 0, 0);
	if(start && transmit && addr0){
		R_PG_I2C_SlaveSend_C(
			iic_data_tr_0,
			10
		);
	}
	else if(start && transmit && addr1){
		R_PG_I2C_SlaveSend_C(
			iic_data_tr_1,
			10
```

```
            );
        }
}
```

4.20.9 R_PG_I2C_SlaveSend_C〈*チャネル番号*〉

定義 R_PG_I2C_SlaveSend_C〈*チャネル番号*〉(uint8_t *data, uint16_t count)
〈*チャネル番号*〉: 0, 1

概要 スレーブのデータ送信

生成条件 スレーブ機能を使用

引数

uint8_t *data	送信するデータの格納先の先頭のアドレス
uint16_t count	送信するデータ数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_I2C_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_IIC_SlaveSend

詳細
- マスタにデータを送信します。
- マスタが送信データ数を上回るデータを要求する場合、先頭のアドレスに戻って送信します。

使用例 R_PG_I2C_SlaveMonitor_C〈*チャネル番号*〉 の使用例を参照してください。

4.20.10 R_PG_I2C_GetDetectedAddress_C〈*チャネル番号*〉

定義　R_PG_I2C_GetDetectedAddress_C〈*チャネル番号*〉
(bool *addr0, bool *addr1, bool *addr2, bool *general, bool *device, bool *host)
〈*チャネル番号*〉: 0, 1

概要　検出したスレーブアドレスの取得

生成条件　スレーブ機能を使用

引数

bool *addr0	スレーブアドレス0検出フラグ格納先
bool *addr1	スレーブアドレス1検出フラグ格納先
bool *addr2	スレーブアドレス2検出フラグ格納先
bool *general	ジェネラルコールアドレス検出フラグ格納先
bool *device	デバイスID検出フラグ格納先
bool *host	ホストアドレス検出フラグ格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_GetStatus

詳細
- 検出したアドレスを取得します。
- 取得しないフラグは0を設定してください。
- 検出したアドレスのフラグには1が設定されます。

使用例　R_PG_I2C_SlaveMonitor_C〈*チャネル番号*〉 の使用例を参照してください。

4.20.11 R_PG_I2C_GetTR_C〈*チャネル番号*〉

定義　R_PG_I2C_GetTR_C〈*チャネル番号*〉(bool * transmit)
〈*チャネル番号*〉: 0, 1

概要　送信/受信モードの取得

生成条件　スレーブ機能を使用

引数

bool * transmit	送信/受信モードフラグの格納先 送信/受信モードフラグ 0:受信モード 1:送信モード

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_GetStatus

詳細
- 送信/受信モードを取得します。

使用例　R_PG_I2C_SlaveMonitor_C〈*チャネル番号*〉 の使用例を参照してください。

4.20.12 R_PG_I2C_GetEvent_C〈チャネル番号〉

定義

R_PG_I2C_GetEvent_C〈チャネル番号〉
(bool *nack, bool *stop, bool *start, bool *lost, bool *timeout)
〈チャネル番号〉: 0, 1

概要

検出イベントの取得

引数

bool *nack	NACK検出フラグ格納先
bool *stop	STOP条件検出フラグ格納先
bool *start	START条件検出フラグ格納先
bool *lost	アービトレーションロスト検出フラグ格納先
bool *timeout	タイムアウト検出フラグ格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_I2C_C〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数

R_IIC_GetStatus

詳細

- 検出したイベントを取得します。
- 取得しないフラグは0を設定してください。
- 検出したイベントのフラグには1が設定されます。

使用例

R_PG_I2C_SlaveMonitor_C〈チャネル番号〉 の使用例を参照してください。

4.20.13R_PG_I2C_GetReceivedDataCount_C〈*チャネル番号*〉

定義　bool R_PG_I2C_GetReceivedDataCount_C〈*チャネル番号*〉(uint16_t *count)

〈*チャネル番号*〉: 0, 1

概要　受信済みデータ数の取得

引数

uint16_t *count	受信データ数の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_GetStatus

詳細

- 現在の転送で受信したデータ数を取得します。

使用例　GUI上で以下の通り設定した場合

- RIIC0 をマスタとして使用
- マスタ受信方法に“全データの受信完了を関数呼び出しで通知する”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//受信データの格納先
uint8_t iic_data[256];

//受信データ数の格納先
uint16_t count;

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ受信
    R_PG_I2C_MasterReceive_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //受信データの格納先アドレス
        256         //受信データ数
    );

    //64バイト受信するまで待つ
    do{
        R_PG_I2C_GetReceivedDataCount_C0( &count );
    } while( count < 64 );
}
```

4.20.14 R_PG_I2C_GetSentDataCount_C〈*チャネル番号*〉

定義　bool R_PG_I2C_GetSentDataCount_C〈*チャネル番号*〉(uint16_t *count)

〈*チャネル番号*〉: 0, 1

概要　送信済みデータ数の取得

引数

uint16_t *count	送信データ数の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_GetStatus

詳細

- 現在の転送で送信したデータ数を取得します。

使用例　GUI上で以下の通り設定した場合

- RIIC0 をマスタとして使用
- マスタ送信方法に“全データの送信完了を関数呼び出しで通知する”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//送信データの格納先
uint8_t iic_data[256];

//送信データ数の格納先
uint16_t count;

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ送信
    R_PG_I2C_MasterSend_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //受信データの格納先アドレス
        256         //受信データ数
    );

    //64バイト送信するまで待つ
    do{
        R_PG_I2C_GetSentDataCount_C0( &count );
    } while( count < 64 );
}
```

4.20.15 R_PG_I2C_Reset_C〈*チャネル番号*〉

定義　R_PG_I2C_Reset_C〈*チャネル番号*〉(void)

〈*チャネル番号*〉: 0, 1

概要　バスのリセット

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_Control

詳細
- モジュールをリセットします。
- 設定は維持されます。

使用例　GUI上で以下の通り設定した場合
- RIIC0 をマスタとして使用
- マスタ送信方法に“全データの送信完了を関数呼び出しで通知する”を選択
- マスタ送信の通知関数名に IIC0MasterTrFunc を指定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//送信データの格納先
uint8_t iic_data[10];

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ送信
    R_PG_I2C_MasterSend_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //送信データの格納先アドレス
        10          //送信データ数
    );
}

void IIC0MasterTrFunc(void)
{
    if( error ){
        R_PG_I2C_Reset_C0();
    }
}
```

4.20.16 R_PG_I2C_StopModule_C〈*チャネル番号*〉

定義　bool R_PG_I2C_StopModule_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0, 1

概要　I2Cバスインタフェースチャネルの停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_I2C_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_IIC_Destroy

詳細
- I2Cバスインタフェースチャネルを停止し、モジュールストップ状態に移行します。

使用例　GUI上で以下の通り設定した場合
- RIIC0 をマスタとして使用
- マスタ受信方法に“全データの受信完了まで待つ”を選択

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

//受信データの格納先
uint8_t iic_data[10];

void func(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //RIIC0を設定
    R_PG_I2C_Set_C0();

    //マスタ受信
    R_PG_I2C_MasterReceive_C0(
        0,          //スレーブアドレスフォーマット
        6,          //スレーブアドレス
        iic_data,   //受信データの格納先アドレス
        10          //受信データ数
    );

    //RIIC0を停止
    R_PG_I2C_StopModule_C0();
}
```

4.21 シリアルペリフェラルインタフェース（RSPI）

4.21.1 R_PG_RSPI_Set_C〈*チャネル番号*〉

定義　bool R_PG_RSPI_Set_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0, 1

概要　RSPIチャネルの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_SPI_Create

詳細
- シリアルペリフェラルインタフェースチャネルのモジュールストップ状態を解除して初期設定し、使用する端子の入出力方向、入力バッファを設定します。
- 本関数を使用する場合、あらかじめR_PG_Clock_Setによりクロックを設定してください。
- 本関数でコマンドは設定されません。コマンドを設定するには R_PG_RSPI_SetCommand_C〈*チャネル番号*〉を呼び出してください。

使用例　GUI上でRSPI0を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();       //クロック発生回路の設定
    R_PG_RSPI_Set_C0();     //RSPI0の設定
    R_PG_RSPI_SetCommand_C0();      //コマンドの設定
}
```

4.21.2 R_PG_RSPI_SetCommand_C〈*チャネル番号*〉

定義　bool R_PG_RSPI_SetCommand_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0, 1

概要　コマンドの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_SPI_Command

詳細
- RPSIコマンドレジスタを設定します。
- GUI上で設定した最大8コマンドを全て設定します。

使用例　GUI上でRSPI0を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();       //クロック発生回路の設定
    R_PG_RSPI_Set_C0();     //RSPI0の設定
    R_PG_RSPI_SetCommand_C0();      //コマンドの設定
}
```

4.21.3 R_PG_RSPI_StartTransfer_C〈*チャネル番号*〉

定義

・ 送信および受信(全二重同期式シリアル通信)

bool R_PG_RSPI_StartTransfer_C〈*チャネル番号*〉

(uint32_t * tx_start, uint32_t * rx_start, uint16_t sequence_loop_count)

〈*チャネル番号*〉: 0, 1

・ 送信動作のみのシリアル通信

bool R_PG_RSPI_StartTransfer_C〈*チャネル番号*〉

(uint32_t * tx_start, uint16_t sequence_loop_count)

〈*チャネル番号*〉: 0, 1

概要 データの転送開始

生成条件 転送方法に“転送完了、エラー検出を関数呼び出しで通知する”を選択

引数

uint32_t * tx_start	送信するデータの先頭のアドレス
uint32_t * rx_start	受信したデータの格納先の先頭のアドレス
uint16_t sequence_loop_count	コマンドシーケンスの繰り返し回数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_RSPI_C〈*チャネル番号*〉.c

〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_SPI_Transfer

詳細

- データの転送を開始します。
- 本関数はGUI上で転送方法に“転送完了、エラー検出を関数呼び出しで通知する”が選択されている場合に出力されます。
- 本関数はすぐにリターンし、エラー検出時または指定した回数のコマンドシーケンス完了時に、指定した名前の通知関数が呼ばれます。通知関数は次の定義で作成してください。

  void 〈*通知関数名*〉(void)

  通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例 GUI上で以下の通り設定した場合

- RSPI0をSPI動作マスタモードで設定
- 転送方法に“転送完了、エラー検出を関数呼び出しで通知する”を指定
- 通知関数名にrsi0_int_funcを指定
- コマンド数:1 フレーム数:4
- コマンド0のビット長:8

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint32_t  tx_data[4] = { 0x11, 0x22, 0x33, 0x44 };
uint32_t  rx_data[4] = { 0x00, 0x00, 0x00, 0x00 };
bool   over_run,   mode_fault,   parity_error;

void func(void)
{
      R_PG_Clock_Set();       //クロック発生回路の設定
      R_PG_RSPI_Set_C0();       //RSPI0の設定
      R_PG_RSPI_SetCommand_C0();       //コマンドの設定
      R_PG_RSPI_StartTransfer_C0( tx_data, rx_data, 1 ); //8bit*4フレーム転送
}

void rsi0_int_func (void)
{
      R_PG_RSPI_GetError_C0 ( &over_run, &mode_fault, &parity_error );   //エラー取得
      if( over_run || mode_fault || parity_error ){
            //エラー検出時処理
      }
      R_PG_RSPI_StopModule_C0();
}
```

4.21.4 R_PG_RSPI_TransferAllData_C〈*チャネル番号*〉

定義

・ 送信および受信(全二重同期式シリアル通信)
bool R_PG_RSPI_TransferAllData_C〈*チャネル番号*〉
(uint32_t * tx_start, uint32_t * rx_start, uint16_t sequence_loop_count)
〈*チャネル番号*〉: 0, 1

・ 送信動作のみのシリアル通信
bool R_PG_RSPI_TransferAllData_C〈*チャネル番号*〉
(uint32_t * tx_start, uint16_t sequence_loop_count)
〈*チャネル番号*〉: 0, 1

・ DTC/DMAC転送
bool R_PG_RSPI_TransferAllData_C〈*チャネル番号*〉
(uint16_t sequence_loop_count)
〈*チャネル番号*〉: 0, 1

概要 全データの転送

生成条件 転送方法に“転送完了、エラー検出を関数呼び出しで通知する”以外を選択

引数

uint32_t * tx_start	送信するデータの先頭のアドレス
uint32_t * rx_start	受信したデータの格納先の先頭のアドレス
uint16_t sequence_loop_count	コマンドシーケンスの繰り返し回数

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_SPI_Transfer

詳細

- 全データを転送します。
- 本関数はGUI上で転送方法に“転送完了、エラー検出を関数呼び出しで通知する”以外が選択されている場合に出力されます。
- 本関数はエラー検出または指定した回数のコマンドシーケンス完了までウェイトします。

使用例

GUI上で以下の通り設定した場合

- RSPI0をSPI動作マスタモードで設定
- 転送方法に“転送完了まで待つ”を指定
- 通知関数名にrsi0_int_funcを指定
- コマンド数:1 フレーム数:4
- コマンド0のビット長:8

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint32_t  tx_data[4] = { 0x11, 0x22, 0x33, 0x44 };
uint32_t  rx_data[4] = { 0x00, 0x00, 0x00, 0x00 };
bool   over_run,   mode_fault,   parity_error;

void func(void)
{
      R_PG_Clock_Set();       //クロック発生回路の設定
      R_PG_RSPI_Set_C0();       //RSPI0の設定
      R_PG_RSPI_SetCommand_C0();       //コマンドの設定
      R_PG_RSPI_StartTransfer_C0( tx_data, rx_data, 1 ); //8bit*4フレーム転送

      R_PG_RSPI_GetError_C0 ( &over_run, &mode_fault, &parity_error );    //エラー取得
      if( over_run || mode_fault || parity_error ){
            //エラー検出時処理
      }
      R_PG_RSPI_StopModule_C0();
}
```

4.21.5 R_PG_RSPI_GetStatus_C〈*チャネル番号*〉

定義

bool R_PG_RSPI_GetStatus_C〈*チャネル番号*〉
(bool * idle, bool * receive_full, bool * transmit_empty)
〈*チャネル番号*〉: 0, 1

概要

転送状態の取得

引数

bool * idle	アイドルフラグの格納先 (0:アイドル状態 1:転送状態)
bool * receive_full	受信バッファフルフラグの格納先 (0:受信バッファにデータなし 1:受信バッファにデータあり)
bool * transmit_empty	送信バッファエンプティフラグの格納先 (0:送信バッファにデータあり 1:送信バッファにデータなし)

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_SPI_GetStatus

詳細

- データの転送状態を取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。
- 本関数内でエラーフラグ(オーバランエラーフラグ、モードフォルトエラーフラグ、パリティエラーフラグ)はクリアされます。エラーフラグを取得する場合は本関数を呼び出す前にR_PG_RSPI_GetError_C〈*チャネル番号*〉を呼び出してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool idle;

void func(void)
{
    do{
        R_PG_RSPI_GetStatus_C0( &idle, 0, 0 );
    }while( idle );
}
```

4.21.6 R_PG_RSPI_GetError_C〈*チャネル番号*〉

定義

bool R_PG_RSPI_GetError_C〈*チャネル番号*〉
（bool * over_run, bool * mode_fault, bool * parity_error）
〈*チャネル番号*〉: 0, 1

概要

エラー検出状態の取得

引数

bool * over_run	オーバランエラーフラグの格納先
bool * mode_fault	モードフォルトエラーフラグの格納先
bool * parity_error	パリティエラーフラグの格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_SPI_GetStatus

詳細

- エラーフラグを取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。
- 本関数内でエラーフラグはクリアされます。

使用例

R_PG_RSPI_StartTransfer_C〈*チャネル番号*〉、R_PG_RSPI_TransferAllData_C〈*チャネル番号*〉、およびR_PG_RSPI_GetCommandStatus_C〈*チャネル番号*〉の使用例を参照してください。

4.21.7 R_PG_RSPI_GetCommandStatus_C〈*チャネル番号*〉

定義

bool R_PG_RSPI_GetCommandStatus_C〈*チャネル番号*〉
（ uint8_t * current_command, uint8_t * error_command ）
〈*チャネル番号*〉: 0, 1

概要

コマンドステータスの取得

生成条件

RSPIチャネルをマスタモードに設定した場合

引数

uint8_t * current_command	現在のコマンドポインタ(0～7)の格納先
uint8_t * error_command	エラー検出時のコマンドポインタ(0～7)の格納先

戻り値

true	取得に成功した場合
false	取得に失敗した場合

出力先ファイル

R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数

R_SPI_GetStatus

詳細

- 現在のコマンドポインタ(0～7)と、エラー検出時のコマンドポインタ(0～7)を取得します。
- 取得する項目に対応する引数に、値の格納先アドレスを指定してください。取得しない項目には0を指定してください。
- 本関数内でエラーフラグ(オーバランエラーフラグ、モードフォルトエラーフラグ、パリティエラーフラグ)はクリアされます。エラーフラグを取得する場合は本関数を呼び出す前にR_PG_RSPI_GetError_C〈*チャネル番号*〉を呼び出してください。

使用例

GUI上で以下の通り設定した場合

- GUI上でRSPI0をSPI動作マスタモードで設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

bool   over_run,   mode_fault,   parity_error;
uint8_t error_command;

void func(void)
{
      R_PG_RSPI_GetError_C0 ( &over_run, &mode_fault, &parity_error );   //エラー取得
      if( over_run || mode_fault || parity_error ){
            R_PG_RSPI_GetCommandStatus_C0( &error_command );
            //エラー検出時処理
      }
}
```

4.21.8 R_PG_RSPI_StopModule_C〈チャネル番号〉

定義 bool R_PG_RSPI_StopModule_C〈チャネル番号〉(void)
〈チャネル番号〉: 0, 1

概要 RSPIチャネルの停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_RSPI_C〈チャネル番号〉.c
〈チャネル番号〉: 0, 1

使用RPDL関数 R_SPI_Destroy

詳細
- RSPIチャネルを停止し、モジュールストップ状態に移行します。

使用例 R_PG_RSPI_StartTransfer_C〈チャネル番号〉およびR_PG_RSPI_TransferAllData_C〈チャネル番号〉の使用例を参照してください。

4.21.9 R_PG_RSPI_LoopBack〈*ループバックモード*〉_C〈*チャネル番号*〉

定義　bool R_PG_RSPI_LoopBack〈*ループバックモード*〉_C〈*チャネル番号*〉(void)
〈*ループバックモード*〉: Direct, Reversed, Disable
〈*チャネル番号*〉: 0, 1

概要　ループバックモードの設定

生成条件　ループバックモードが設定されている場合

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_RSPI_C〈*チャネル番号*〉.c
〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_SPI_Control

詳細

- 端子をループバックモードに設定または無効化します。
- R_PG_RSPI_LoopBackDirect_C〈*チャネル番号*〉を呼び出すとシフトレジスタの入力経路と出力経路を接続します。(送信データ＝受信データ)
- R_PG_RSPI_LoopBackReversed_C〈*チャネル番号*〉を呼び出すとシフトレジスタの入力経路と出力経路の反転を接続します。(送信データの反転＝受信データ)
- R_PG_RSPI_LoopBackDisable_C〈*チャネル番号*〉を呼び出すとループバックモードを無効にします。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
      R_PG_RSPI_LoopBackDirect_C0();   //ループバックモードの設定
}
```

4.22 12 ビットA/Dコンバータ（S12AD）

4.22.1 R_PG_ADC_12_Set_S12AD0

定義　bool R_PG_ADC_12_Set_S12AD0 (void)

概要　12ビットA/Dコンバータの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ADC_12_S12AD0.c

使用RPDL関数　R_ADC_12_Create

詳細

- 12ビットA/Dコンバータのモジュールストップ状態を解除して初期設定し、変換開始トリガ入力待ち状態にします。変換開始トリガにソフトウェアを選択した場合は、R_PG_ADC_12_StartConversionSW_S12AD0により変換を開始します。
- 本関数を呼び出す前にR_PG_Clock_Setによりクロックを設定してください。
- 本関数内でアナログ入力端子として使用する端子の入出力方向を入力に設定し、入力バッファを無効にします。
- 本関数内でA/D変換終了割り込みを設定します。GUI上で割り込み通知関数名を指定した場合、CPUへの割り込み要求が発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。

  void <割り込み通知関数名> (void)

  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上で以下の通り設定した場合

- アナログ入力端子にAN1を指定
- A/D変換終了割り込み通知関数名に S12ad0IntFunc を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t result[2];   //A/D変換結果の格納先(AN0～AN1 (AN0は無効) )

void func(void)
{
      R_PG_Clock_Set();                 //クロックの設定
      R_PG_ADC_12_Set_S12AD0();       //12ビットA/Dコンバータ(S12AD0)を設定
}

//AD変換終了割り込み通知関数
void S12ad0IntFunc(void)
{
      R_PG_ADC_12_GetResult_S12AD0(result);     //A/D変換結果の取得
}
```

4.22.2 R_PG_ADC_12_StartConversionSW_S12AD0

定義　bool R_PG_ADC_12_StartConversionSW_S12AD0(void)

概要　A/D変換の開始（ソフトウェアトリガ）

生成条件　変換開始要因にソフトウェアトリガが指定された場合

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ADC_12_S12AD0.c

使用RPDL関数　R_ADC_12_Control

詳細
- 起動要因にソフトウェアトリガを選択したA/D変換器のA/D変換を開始します。

使用例

GUI上で以下の通り設定した場合
- 起動要因にソフトウェアトリガを選択
- A/D変換終了割り込み通知関数名に S12ad0IntFunc を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();                   //クロックの設定
    R_PG_ADC_12_Set_S12AD0();       //12ビットA/Dコンバータ(S12AD0)を設定

    //ソフトウェアトリガによりAD変換開始
    R_PG_ADC_12_StartConversionSW_S12AD0();
}
```

4.22.3 R_PG_ADC_12_StopConversion_S12AD0

定義　bool R_PG_ADC_12_StopConversion_S12AD0(void)

概要　A/D変換の中断

引数　なし

戻り値

true	変換停止に成功した場合
false	変換停止に失敗した場合

出力先ファイル　R_PG_ADC_12_S12AD0.c

使用RPDL関数　R_ADC_12_Control

詳細

- 連続スキャンモードのA/D変換を停止します。1サイクルスキャンモードではA/D変換完了後に本関数を呼び出す必要はありません。
  本関数でA/D変換を停止した後、A/D変換開始トリガを入力すると連続スキャンを再開します。連続スキャンを終了するにはR_PG_ADC_12_StopModule_S12AD0を呼び出し、A/D変換ユニットを停止状態にしてください。

使用例　GUI上で以下の通り設定した場合

- 動作モードを連続スキャンモードに設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
    R_PG_Clock_Set();                   //クロックの設定
    R_PG_ADC_12_Set_S12AD0();       //12ビットA/Dコンバータ(S12AD0)を設定
}

void func2(void)
{
    //連続スキャンを停止
    R_PG_ADC_12_StopConversion_S12AD0();
}
```

4.22.4 R_PG_ADC_12_GetResult_S12AD0

定義 bool R_PG_ADC_12_GetResult_S12AD0(uint16_t * result)

概要 A/D変換結果の取得

引数

uint16_t * result	A/D変換結果の格納先

戻り値

true	結果の取得に成功した場合
false	結果の取得に失敗した場合

出力先ファイル R_PG_ADC_12_S12AD0.c

使用RPDL関数 R_ADC_12_Read

詳細

- 取得するデータの数は、使用するA/D変換チャネルの数に依ります。使用するチャネルのA/D変換結果を格納するのに必要な領域を結果の格納先に確保してください。
- GUI上で割り込み通知関数名を指定していない場合、本関数を呼び出した時点でA/D変換中であったときは、結果を読み出す前に変換が終了するまで本関数内で待ちます。

使用例 GUI上で以下の通り設定した場合

- アナログ入力端子にAN0～AN3の4チャネルを指定
- A/D変換終了割り込み通知関数名に S12ad0IntFunc を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();                   //クロックの設定
    R_PG_ADC_12_Set_S12AD0();       //12ビットA/Dコンバータ(S12AD0)を設定
}

//AD変換終了割り込み通知関数
void S12ad0IntFunc(void)
{
    uint16_t result[4];   //AN0～AN3のA/D変換結果の格納先
    uint16_t result_an0;   //AN0のA/D変換結果の格納先
    uint16_t result_an1;   //AN1のA/D変換結果の格納先
    uint16_t result_an2;   //AN2のA/D変換結果の格納先
    uint16_t result_an3;   //AN3のA/D変換結果の格納先

    //A/D変換結果の取得
    R_PG_ADC_12_GetResult_S12AD0( result );

    result_an0 = result[0];
    result_an1 = result[1];
    result_an2 = result[2];
    result_an3 = result[3];
}
```

4.22.5 R_PG_ADC_12_StopModule_S12AD0

定義　bool R_PG_ADC_12_StopModule_S12AD0(void)

概要　12ビットA/Dコンバータの停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_ADC_12_S12AD0.c

使用RPDL関数　R_ADC_12_Destroy

詳細

- 12ビットA/Dコンバータを停止し、モジュールストップ状態に移行します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t result[8];   //A/D変換結果の格納先(AN0～AN7)

void func1(void)
{
      R_PG_Clock_Set();                 //クロックの設定
      R_PG_ADC_12_Set_S12AD0();       //12ビットA/Dコンバータ(S12AD0)を設定
}

void func2(void)
{
      //連続スキャンを停止
      R_PG_ADC_12_StopConversion_S12AD0();

      //A/D変換結果の取得
      R_PG_ADC_12_GetResult_S12AD0( result );

      //12ビットA/Dコンバータ(S12AD0)を停止
      R_PG_ADC_12_StopModule_S12AD0();
}
```

4.23 10 ビットA/Dコンバータ（ADa）

4.23.1 R_PG_ADC_10_Set_AD〈*ユニット番号*〉

定義　bool R_PG_ADC_10_Set_AD〈*ユニット番号*〉(void)　〈*ユニット番号*〉： 0～1

概要　10ビットA/Dコンバータの設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈*ユニット番号*〉.c　〈*ユニット番号*〉： 0～1

使用RPDL関数　R_ADC_10_Create

詳細

- A/D変換器のモジュールストップ状態を解除して初期設定し、変換開始トリガ入力待ち状態にします。変換開始トリガにソフトウェアを選択した場合は、R_PG_ADC_10_StartConversionSW_AD〈*チャネル番号*〉により変換を開始します。
- 本関数を呼び出す前にR_PG_Clock_Setによりクロックを設定してください。
- アナログ入力端子として使用する端子の入出力方向を入力に設定し、入力バッファを無効にします。
- 本関数内でA/D変換終了割り込みを設定します。GUI上で割り込み通知関数名を指定した場合、CPUへの割り込み要求が発生すると指定した名前の関数が呼び出されます。通知関数は次の定義で作成してください。

  void 〈割り込み通知関数名〉(void)

  割り込み通知関数については本章末尾の「通知関数に関する注意事項」の内容に注意してください。

使用例　GUI上で以下の通り設定した場合

- AD0を設定
- A/D変換終了割り込み通知関数名に Ad0IntFunc を設定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t data;   //A/D変換結果の格納先

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_ADC_10_Set_AD0();         //AD0を設定
}

//AD変換終了割り込み通知関数
void Ad0IntFunc(void)
{
    R_PG_ADC_10_GetResult_AD0(&data);      //A/D変換結果の取得
}
```

4.23.2 R_PG_ADC_10_StartConversionSW_AD〈ユニット番号〉

定義　bool R_PG_ADC_10_StartConversionSW_AD〈ユニット番号〉(void)
〈ユニット番号〉： 0～1

概要　A/D変換の開始（ソフトウェアトリガ）

生成条件　変換開始トリガにソフトウェアトリガが指定された場合

引数　なし

戻り値

true	変換開始に成功した場合
false	変換開始に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈ユニット番号〉.c
〈ユニット番号〉： 0～1

使用RPDL関数　R_ADC_10_Control

詳細
- 起動要因にソフトウェアトリガを選択したA/D変換器のA/D変換を開始します。

使用例　GUI上で以下の通り設定した場合
- AD0のモードを連続スキャンモードで設定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_ADC_10_Set_AD0();         //AD0を設定

    //ソフトウェアトリガによりAD変換開始
    R_PG_ADC_10_StartConversionSW_AD0();
}
```

4.23.3 R_PG_ADC_10_StopConversion_AD〈ユニット番号〉

定義　bool R_PG_ADC_10_StopConversion_AD〈ユニット番号〉(void)
〈ユニット番号〉： 0～1

概要　A/D変換の中断

引数　なし

戻り値

true	変換停止に成功した場合
false	変換停止に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈ユニット番号〉.c
〈ユニット番号〉： 0～1

使用RPDL関数　R_ADC_10_Control

詳細
- 連続スキャンモードのA/D変換を停止します。シングルモードおよび1サイクルスキャンモードではA/D変換完了後に本関数を呼び出す必要はありません。
- 本関数でA/D変換を停止した後、A/D変換開始トリガを入力すると連続スキャンを再開します。連続スキャンを終了するにはR_PG_ADC_10_StopModule_AD〈ユニット番号〉を呼び出し、A/D変換ユニットを停止状態にしてください。

使用例　GUI上で以下の通り設定した場合
- AD0の起動要因をソフトウェアトリガに指定して設定

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t data;  //A/D変換結果の格納先

void func1(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_ADC_10_Set_AD0();         //AD0を設定
}

void func2(void)
{
    //連続スキャンを停止
    R_PG_ADC_10_StopConversion_AD0();

    //A/D変換結果の取得
    R_PG_ADC_10_GetResult_AD0(&data);
}
```

4.23.4 R_PG_ADC_10_GetResult_AD〈*ユニット番号*〉

定義　bool R_PG_ADC_10_GetResult_AD〈*ユニット番号*〉(uint16_t * result)

〈*ユニット番号*〉： 0～1

概要　A/D変換結果の取得

引数

uint16_t * result	A/D変換結果の格納先

戻り値

true	結果の取得に成功した場合
false	結果の取得に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈*ユニット番号*〉.c　〈*ユニット番号*〉： 0～1

使用RPDL関数　R_ADC_10_Read

詳細

- 取得するデータの数は、使用するA/D変換チャネルの数に依ります。使用するチャネルのA/D変換結果を格納するのに必要な領域を確保してください。
- GUI上で割り込み通知関数名を指定していない場合、本関数を呼び出した時点でA/D変換中であったときは、結果を読み出す前に変換が終了するまで本関数内で待ちます。

使用例　GUI上で以下の通り設定した場合

- AD0を1サイクルスキャンモードで設定
- アナログ入力端子にAN0～AN3の4チャネルを指定
- A/D変換終了割り込み通知関数名に Ad0IntFunc を設定

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_ADC_10_Set_AD0();         //AD0を設定
}

//AD変換終了割り込み通知関数
void Ad0IntFunc(void)
{
    uint16_t result[4];   //全チャネルのA/D変換結果の格納先
    uint16_t result_an0;  //AN0のA/D変換結果の格納先
    uint16_t result_an1;  //AN1のA/D変換結果の格納先
    uint16_t result_an2;  //AN2のA/D変換結果の格納先
    uint16_t result_an3;  //AN3のA/D変換結果の格納先

    //A/D変換結果の取得
    R_PG_ADC_10_GetResult_AD0( result );

    result_an0 = result[0];
    result_an1 = result[1];
    result_an2 = result[2];
    result_an3 = result[3];
}
```

4.23.5 R_PG_ADC_10_SetSelfDiag_VREF_〈電圧値〉_AD〈ユニット番号〉

定義　bool R_PG_ADC_10_SetSelfDiag_VREF_〈電圧値〉_AD〈ユニット番号〉(void)
〈電圧値〉： 0, 0_5, 1 (0:Vref*0, 0_5:Vref/2, 1:Vref)　〈ユニット番号〉： 0～1

概要　A/D自己診断機能の設定

生成条件　自己診断機能を有効に設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈ユニット番号〉.c　〈ユニット番号〉： 0～1

使用RPDL関数　R_ADC_10_Create

詳細

- 自己診断機能を設定します。
- 本関数内でA/D変換モードはシングルモードに、変換開始トリガはソフトウェアトリガに設定されます。A/Dコンバータを再設定するにはR_PG_ADC_10_Set_AD〈ユニット番号〉を呼び出してください。
- 自己診断を開始するには R_PG_ADC_10_StartSelfDiag_AD〈ユニット番号〉 を、自己診断結果を取得するには R_PG_ADC_10_GetResult_AD〈ユニット番号〉 を呼び出してください。

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"
uint16_t SelfDiagnostic_0(void)
{
	uint16_t result;
	R_PG_ADC_10_SetSelfDiag_VREF_0_AD0();
	R_PG_ADC_10_StartSelfDiag_AD0();
	R_PG_ADC_10_GetResult_AD0 (&result);
	return result;
}
uint16_t SelfDiagnostic_0_5(void)
{
	uint16_t result;
	R_PG_ADC_10_SetSelfDiag_VREF_0_5_AD0();
	R_PG_ADC_10_StartSelfDiag_AD0();
	R_PG_ADC_10_GetResult_AD0 (&result);
	return result;
}
uint16_t SelfDiagnostic_1(void)
{
	uint16_t result;
	R_PG_ADC_10_SetSelfDiag_VREF_1_AD0();
	R_PG_ADC_10_StartSelfDiag_AD0();
	R_PG_ADC_10_GetResult_AD0 (&result);
	return result;
}
```

4.23.6 R_PG_ADC_10_StartSelfDiag_AD〈ユニット番号〉

定義　bool R_PG_ADC_10_StartSelfDiag_AD〈ユニット番号〉(void)
〈ユニット番号〉： 0～1

概要　A/D変換の開始（自己診断機能）

生成条件　自己診断機能を有効に設定

引数　なし

戻り値

true	変換開始に成功した場合
false	変換開始に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈ユニット番号〉.c
〈ユニット番号〉： 0～1

使用RPDL関数　R_ADC_10_Control

詳細
- 自己診断のA/D変換を開始します。
- 本関数を呼び出す前に R_PG_ADC_10_SetSelfDiag_VREF_〈電圧値〉_AD〈ユニット番号〉を呼び出し、自己診断機能を設定してください。
- 自己診断結果を取得するには R_PG_ADC_10_GetResult_AD〈ユニット番号〉を呼び出してください。

使用例　R_PG_ADC_10_SetSelfDiag_VREF_〈電圧値〉_AD〈ユニット番号〉の使用例を参照してください。

4.23.7 R_PG_ADC_10_StopModule_AD〈ユニット番号〉

定義　bool R_PG_ADC_10_StopModule_AD〈ユニット番号〉(void)
〈ユニット番号〉： 0～1

概要　10ビットA/Dコンバータの停止

引数　なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル　R_PG_ADC_10_AD〈ユニット番号〉.c
〈ユニット番号〉： 0～1

使用RPDL関数　R_ADC_10_Destroy

詳細
- 10ビットA/Dコンバータのユニットを停止し、モジュールストップ状態に移行します。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

uint16_t data;  //A/D変換結果の格納先

void func1(void)
{
    R_PG_Clock_Set();              //クロックの設定
    R_PG_ADC_10_Set_AD0();         //AD0を設定
}

void func2(void)
{
    //連続スキャンを停止
    R_PG_ADC_10_StopConversion_AD0();

    //A/D変換結果の取得
    R_PG_ADC_10_GetResult_AD0(&data);

    //AD0を停止
    R_PG_ADC_10_StopModule_AD0();
}
```

4.24 D/A コンバータ

4.24.1 R_PG_DAC_Set_C〈*チャネル番号*〉

定義　bool R_PG_DAC_Set_C〈*チャネル番号*〉(void)　〈*チャネル番号*〉: 0, 1

概要　D/Aコンバータのチャネルを設定

引数　なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_DAC_C〈*チャネル番号*〉.c　〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_DAC_10_Create

詳細

- D/Aコンバータのチャネルを設定します。
- D/Aコンバータのモジュールストップ状態が解除されます。
- アナログ出力端子からは、モジュールストップ状態解除後のD/Aデータレジスタの初期値(0)の変換結果が出力されます。初期値を指定して出力を開始する場合はR_DAC_SetWithInitialValue_C〈*チャネル番号*〉を使用してください。

評価版での制限事項

- DA0とDA1を同時に使用できません
- 出力関数仕様、GUI使用は製品版までに変更される可能性があります

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
      //DA0端子を設定
      R_PG_DAC_Set_C0();
}

void func2( uint16_t   output_val )
{
      //D/A変換値の変更
      R_PG_DAC_ControlOutput_C0( output_val );
}
```

4.24.2 R_PG_DAC_SetWithInitialValue_C〈*チャネル番号*〉

定義　bool R_PG_DAC_SetWithInitialValue_C〈*チャネル番号*〉(uint16_t initial_val)
〈*チャネル番号*〉: 0, 1

概要　初期値を指定してD/Aコンバータのチャネルを設定

引数

uint16_t initial_val	D/A変換値の初期値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル　R_PG_DAC_C〈*チャネル番号*〉.c　〈*チャネル番号*〉: 0, 1

使用RPDL関数　R_DAC_10_Create

詳細
- D/A変換値の初期値を指定してD/Aコンバータのチャネルを設定し、出力を開始します。
- 他のチャネルが設定されていない場合、D/Aコンバータのモジュールストップ状態が解除されます。

評価版での制限事項
- DA0とDA1を同時に使用できません
- 出力関数仕様、GUI使用は製品版までに変更される可能性があります

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func1(uint16_t   initial_val)
{
      //DA0端子を設定し出力を開始
      R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2( uint16_t   output_val )
{
      //D/A変換値の変更
      R_PG_DAC_ControlOutput_C0( output_val );
}
```

4.24.3 R_PG_DAC_ControlOutput_C〈*チャネル番号*〉

定義 bool R_PG_DAC_ControlOutput_C〈*チャネル番号*〉(uint16_t output_val)

〈*チャネル番号*〉: 0, 1

概要 D/A変換値の設定

引数

uint16_t output_val	D/Aデータレジスタに設定するD/A変換値

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DAC_C〈*チャネル番号*〉.c 〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_DAC_10_Write

詳細
- D/Aデータレジスタに D/A変換値を設定します。

評価版での制限事項
- 出力関数仕様、GUI使用は製品版までに変更される可能性があります

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func1(uint16_t  initial_val)
{
    //DA0端子を設定し出力を開始
    R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2( uint16_t  output_val )
{
    //D/A変換値の変更
    R_PG_DAC_ControlOutput_C0( output_val );
}
```

4.24.4 R_PG_DAC_StopOutput_C〈*チャネル番号*〉

定義 bool R_PG_DAC_StopOutput_C〈*チャネル番号*〉(void)
〈*チャネル番号*〉: 0, 1

概要 アナログ出力の停止

引数 なし

戻り値

true	停止に成功した場合
false	停止に失敗した場合

出力先ファイル R_PG_DAC_C〈*チャネル番号*〉.c 〈*チャネル番号*〉: 0, 1

使用RPDL関数 R_DAC_10_Destroy

詳細
- アナログ出力を停止します。
- 全チャネルの出力が停止する場合、D/Aコンバータはモジュールストップ状態に移行します。

評価版での制限事項
- 出力関数仕様、GUI使用は製品版までに変更される可能性があります

使用例

```
//この関数を使用するには"R_PG_〈PDGプロジェクト名〉.h"をインクルードしてください
#include "R_PG_default.h"

void func1(uint16_t  initial_val)
{
      //DA0端子を設定し出力を開始
      R_PG_DAC_SetWithInitialValue_C0( initial_val );
}

void func2(void)
{
      //アナログ出力の停止
      R_PG_DAC_StopOutput_C0();
}
```

4.24.5 R_PG_DAC_Set_C0_C1

定義 bool R_PG_DAC_Set_C0_C1 (void)

概要 D/Aコンバータのチャネルを設定 (DA0, DA1一括設定)

生成条件 DA0, DA1を両方使用する場合

引数 なし

戻り値

true	設定が正しく行われた場合
false	設定に失敗した場合

出力先ファイル R_PG_DAC.c

使用RPDL関数 R_DAC_10_Create

詳細

- D/Aコンバータのチャネルを一括設定します。
- D/Aコンバータのモジュールストップ状態が解除されます。
- アナログ出力端子からは、モジュールストップ状態解除後のD/Aデータレジスタの初期値(0)の変換結果が出力されます。初期値を指定して出力を開始する場合は R_DAC_SetWithInitialValue_C*<チャネル番号>*を使用してください。

使用例

```
//この関数を使用するには"R_PG_<PDGプロジェクト名>.h"をインクルードしてください
#include "R_PG_default.h"

void func1(void)
{
	//DA0, DA1端子を一括設定
	R_PG_DAC_Set_C0_C1();
}

void func2( uint16_t output_val_c0, uint16_t output_val_c1 )
{
	//D/A変換値の変更
	R_PG_DAC_ControlOutput_C0( output_val_c0 );
	R_PG_DAC_ControlOutput_C1( output_val_c1 );
}
```

4.25 通知関数に関する注意事項

4.25.1 割り込みとプロセッサモード

RX CPU は、スーパバイザモード、およびユーザモードの 2 つのプロセッサモードをサポートします。PDG の出力関数および RPDL の関数はユーザモードで実行されますが、各通知関数は RPDL の割り込みハンドラから呼び出されるため、スーパバイザモードで動作します。スーパバイザモードでは特権命令(RTFI、RTE、WAIT)を使用できますが、通知関数と通知関数から呼び出される他の関数では以下の点に注意してください。

- RTFI および RTE 命令は RPDL の割り込みハンドラで実行するため、ユーザプログラムでこれらを実行する必要はありません。
- PDG の出力関数および RPDL の関数では消費電力低減のために wait()命令を呼び出しています。ユーザプログラムから wait()を呼び出さないでください。

プロセッサモードについての詳細は RX ファミリ ソフトウェアマニュアルを参照してください。

4.25.2 割り込みとDSP命令

アキュムレータ(ACC)は以下の命令で変更されます。

- DSP 機能命令(MACHI、MACLO、MULHI、MULLO、MVTACHI、MVTACLO、および RACW)
- 乗算命令、積和演算命令(EMUL、EMULU、FMUL、MUL、および RMPA)

RPDL の割り込みハンドラでは ACC の値をスタックに退避しません。各通知関数は RPDL の割り込みハンドラから呼び出されるため、通知関数内でこれらの命令を使用する場合は ACC の値を退避し、通知関数が終了する前に再設定してください。

5. 生成ファイルのIDEへの登録とビルド

Peripheral Driver Generator で生成したファイルの IDE(High-performance Embedded Workshop／CubeSuite+／e2studio)への登録とビルドについては以下の点に注意してください。

(1) Peripheral Driver Generator が生成するソースファイルにはスタートアッププログラムは含まれません。IDE のプロジェクト作成時にプロジェクトタイプとして Application を指定してスタートアッププログラムを作成してください。

(2) Peripheral Driver Generator が IDE に登録するソースファイルには割り込みハンドラとベクタテーブルが含まれます。IDE で生成したスタートアッププログラムに含まれる割り込みハンドラ、ベクタテーブルとの重複を避けるため、Peripheral Driver Generator から IDE にソースファイルを登録する際、intprg.c と vecttbl.c（e2 studio の場合は interrupt_handlers.c と vector_table.c）はビルドの対象から除外されます。

(3) Peripheral Driver Generator が IDE に登録する割り込みハンドラを含むソースファイル Interrupt_<周辺機能名>.c は、Peripheral Driver Generator のソース生成時に上書きされます。

(4) Renesas Peripheral Driver Library ライブラリファイルは、デフォルトのオプションで作成しています。(ただし、double 型の精度は倍精度に設定して作成しています) お客様のプロジェクトでデフォルト以外のオプションを指定する場合は、お客様の責任で Renesas Peripheral Driver Library ライブラリのソースファイルを利用してください。

(5) Renesas Peripheral Driver Library は double 型の精度を倍精度に設定して作成されています。そのため、Peripheral Driver Generator が生成したソースを含むプログラムをビルドするには、以下のように IDE のビルダ設定で double 型の精度を指定してください。(e2 studio ではソース登録と同時に自動で変更します)

CubeSuite+

1. プロジェクトツリーの［CC-RX(ビルド・ツール)］をダブルクリックし、[CC-RXのプロパティ］を表示してください。
2. [CPU]カテゴリ内の［double型、およびlong double型の精度］に［倍精度として扱う］を設定してください。

High-performance Embedded Workshop

1. メインメニューから［ビルド］->［RX Standard Toolchain］を選択し、[RX Standard Toolchain］ダイアログボックスを開いてください。
2. ［CPU］タブを選択してください。
3. ［詳細］ボタンをクリックし、[CPU詳細］ダイアログボックスを開いてください。
4. ［double型の精度］に［倍精度］を設定してください。

(6) Renesas Peripheral Driver Library は FIXEDVECT セクションの開始アドレスを、0xFFFFFFD0 にして作成しています。そのため PDG2 が生成したソースを含むプログラムをビルドするには、以下のようにビルダの設定で FIXEDVECT セクションのアドレスを変更してください。(CubeSuite+ および High-performance Embedded Workshop では変更不要)

e2 studio

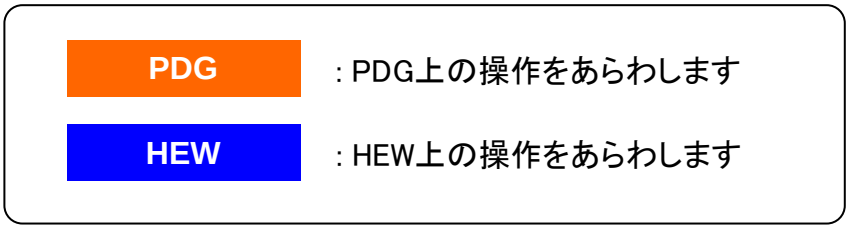
1. プロジェクトエクスプローラでプロジェクトを選んでください。
2. メニューから[ファイル]->[プロパティ]を選択し[プロパティ]を表示してください。
3. プロパティの[C/C++ビルド]の[設定]を選んでください。
4. 構成:で[全ての構成]を選んでください。
5. [Linker]の[セクション]を選択し、「セクション・ビューアー」を表示してください。
5. [セクション・ビューアー]で、FIXEDVECTセクションのアドレスを0xFFFFFFD0に設定してください。

6. チュートリアル

本章では、PDG と HEW を使用して RX62N 用 Renesas Starter Kit+のボードを動作させる以下のチュートリアルプログラムの作成方法を示しながら、PDG の使用手順を紹介します。

- 8 ビットタイマ(TMR)の割り込みで LED を点滅
- マルチファンクションタイマパルスユニット 2(MTU2)の PWM 波で LED を点滅
- 10 ビット A/D コンバータ（ADa）の連続スキャン
- IRQ による DTC 転送のトリガ
- SCIa チャネル 2 とチャネル 5 で調歩同期通信

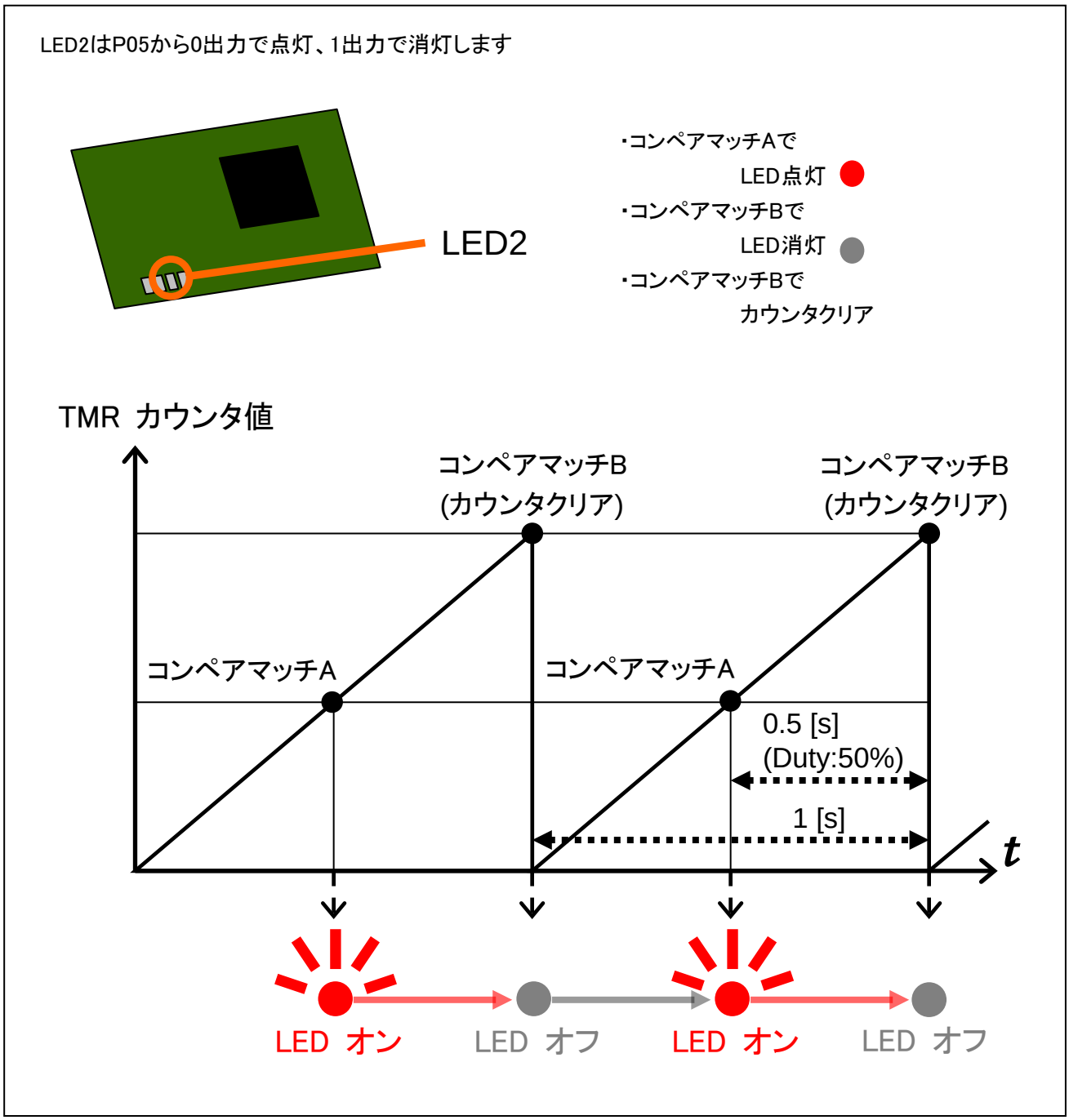
説明の中にある以下の表示はそれぞれ PDG、HEW 上での操作をあらわします。



6.1 8 ビットタイマ(TMR)の割り込みでLEDを点滅

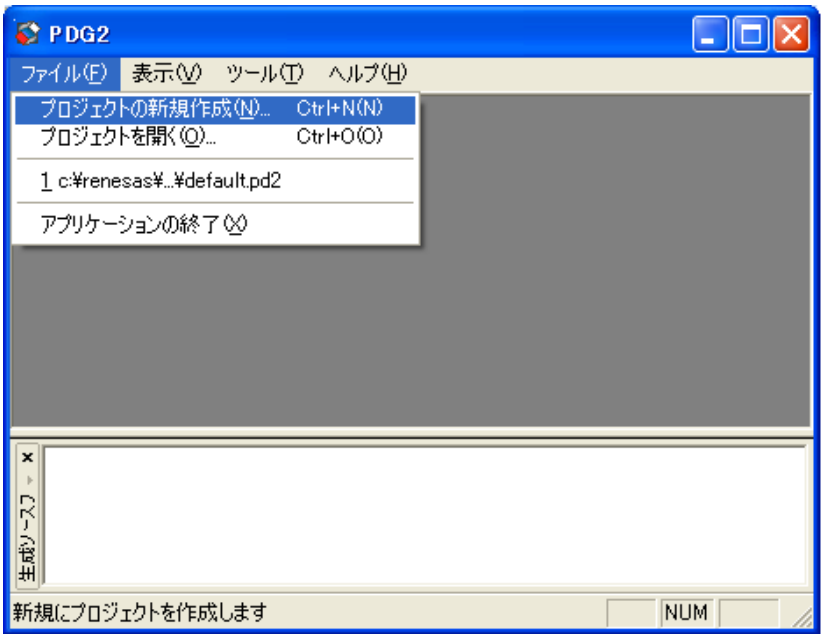
RSK+ボード上の LED2 は P05 に接続されています。 このチュートリアルでは 8 ビットタイマ(TMR)と I/O ポートを設定し、LED を次のように点滅させます。

使用する RSK+ボード上に P05 の有効/無効を切り替えるスイッチがある場合は有効にしてください。

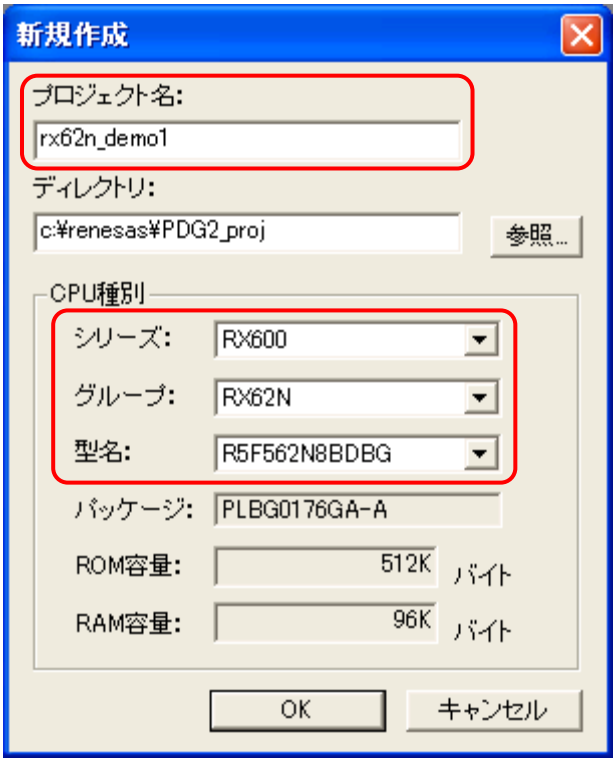


(1) PDG プロジェクトの作成 **PDG**

1. PDG を起動してください。
2. メニューから [ファイル]->[プロジェクトの新規作成] を選択してください。

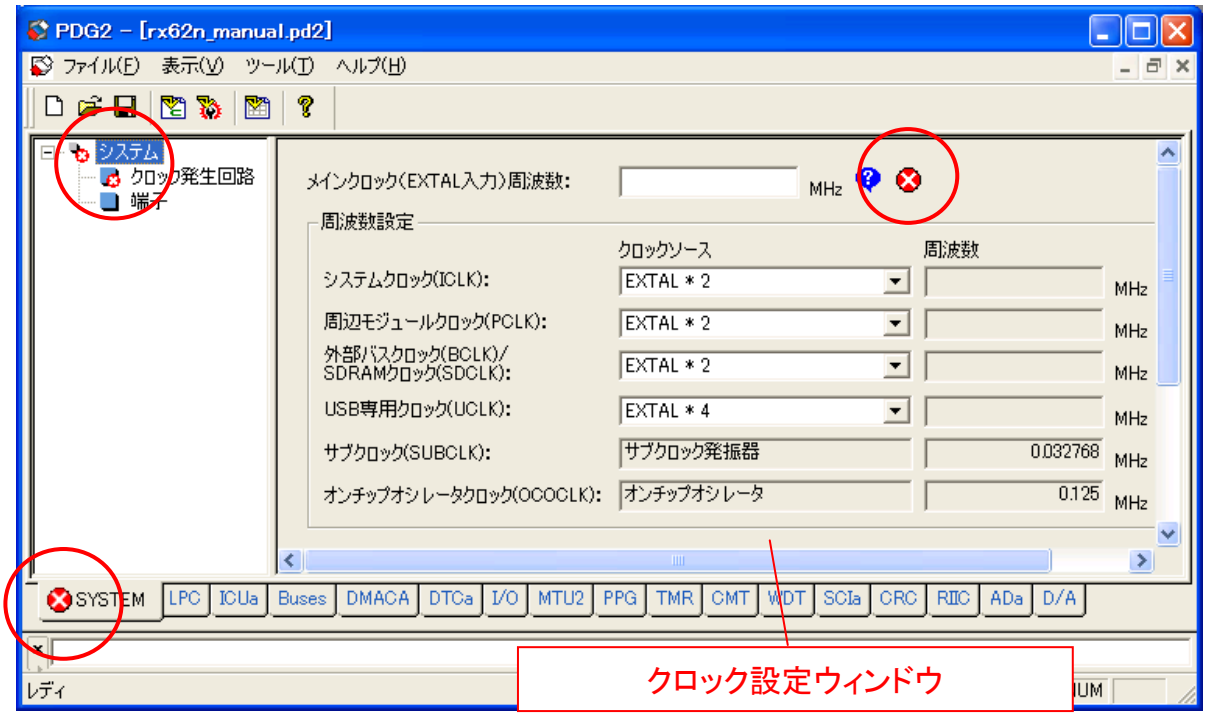


3. プロジェクト名に“rx62n_demo1”を指定してください。
   CPU 種別は以下の通り設定してください。但し使用する RSK+ボードに他の型名のチップが搭載されている場合は、ボードに合わせて設定してください。
   シリーズ : RX600
   グループ : RX62N
   型名 : R5F562N8BDBG

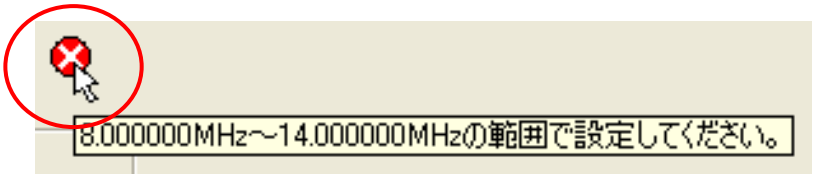


(2) 初期状態 PDG

・プロジェクトの作成直後はクロック設定ウィンドウが開き、エラーアイコンが表示されます。



・エラーアイコンの上にマウスポインタを置くと、エラーの内容が表示されます。



PDG には3 種類のアイコンがあります。

エラー
設定は許可されません。
設定にエラーがある場合、ソースファイルの生成はできません。

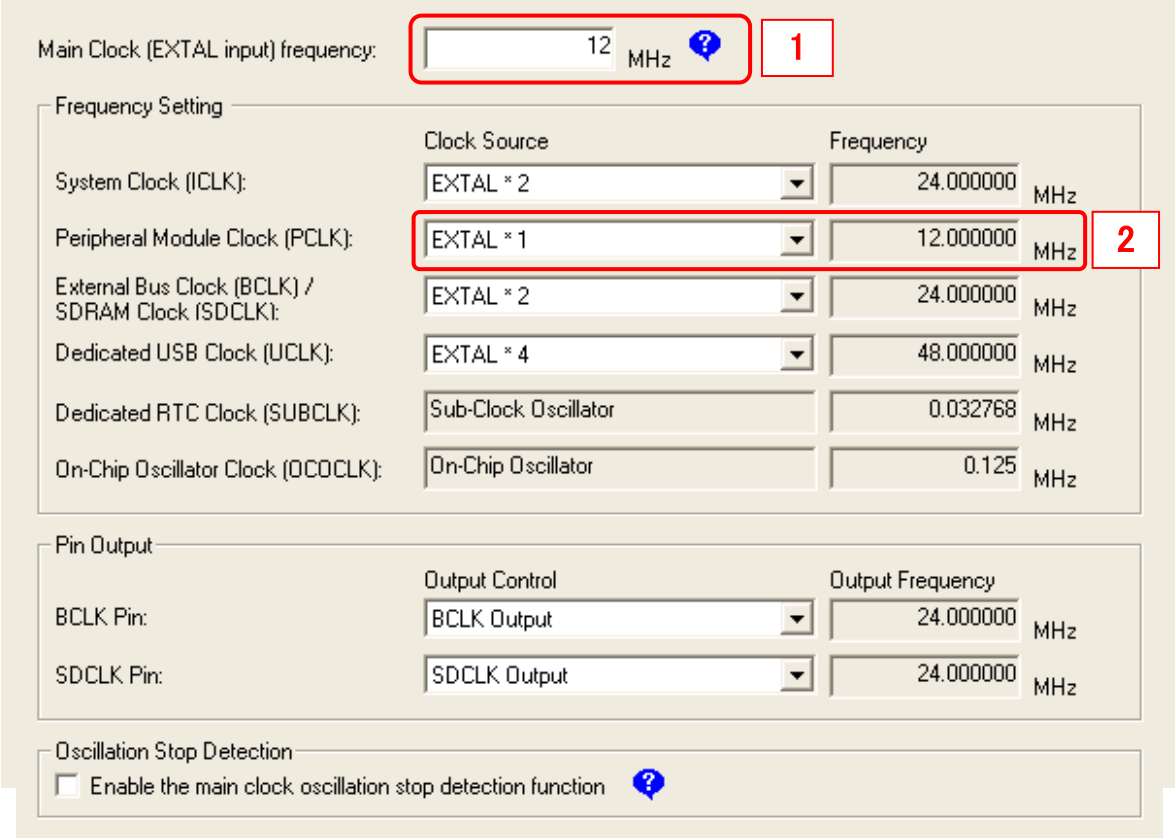
警告
設定は可能ですが、誤っている可能性があります。
ソースファイルの生成は可能です。

インフォメーション
複雑な設定箇所の付加情報です。

設定ウィンドウ上のアイコンのみツールチップを表示できます。

(3) クロックの設定 **PDG**

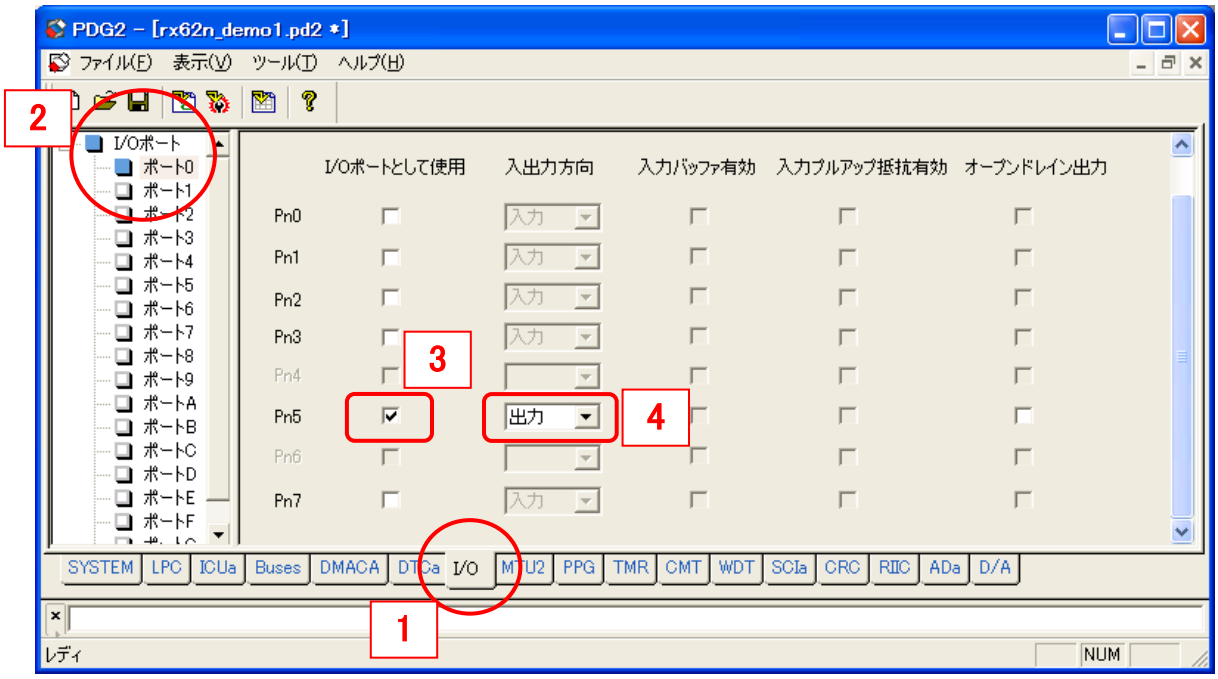
1. 最初にメインクロック(EXTAL 入力)周波数を設定してください。
   RSK+ボードの外部入力周波数は 12MHz です。“12”と入力してください。
2. 周辺モジュールクロック(PCLK)は 12MHzで使用します。
   PCLK の倍率に“EXTAL *1”を選択し、PCLK 周波数を 12MHzに設定してください。



(4) I/O ポートの設定 PDG

LED2 が接続されている P05 を出力ポートに設定します。

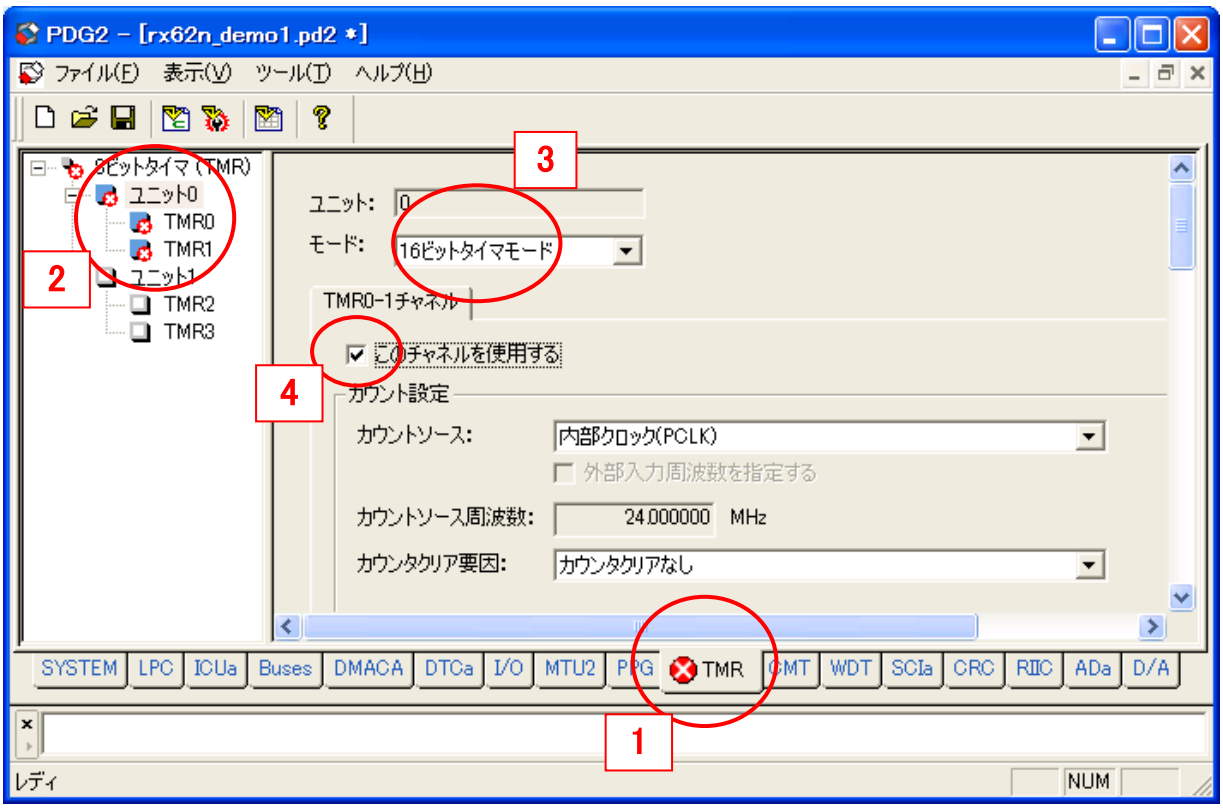
1. [I/O] タブを選択してください
2. [ポート0] を選択してください
3. [Pn5] をチェックしてください
4. [出力] を選択してください



(5) TMR の設定-1 PDG

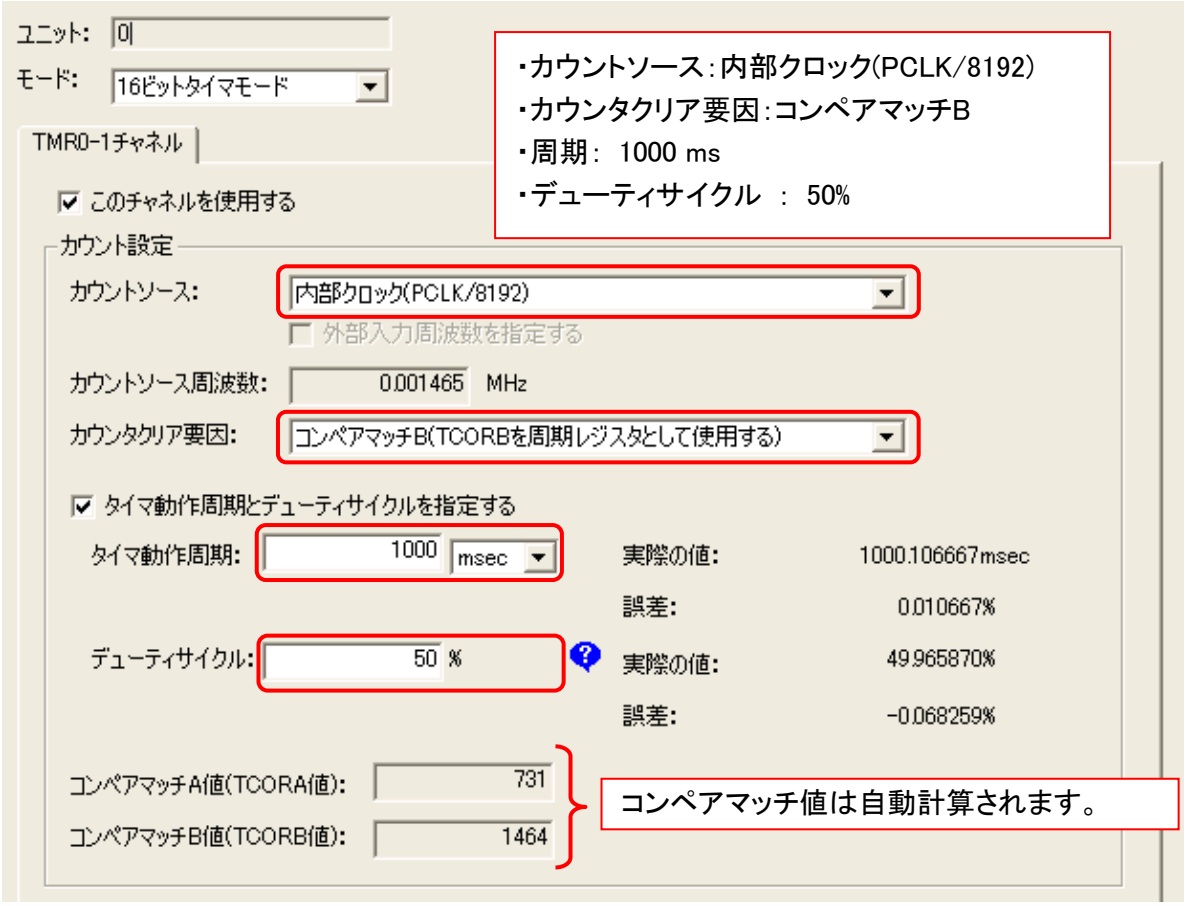
このチュートリアルでは TMR (8 ビットタイマ) のユニット 0 を 16 ビットモード (2 つの 8 ビットタイマをカスケード接続するモード)で使用します。

1. [TMR] タブを選択してください。
2. [ユニット0] を選択してください。
3. [16ビットタイマモード] を選択してください。
4. [このチャネルを使用する] を選択してください。



(6) TMR の設定-2 **PDG**

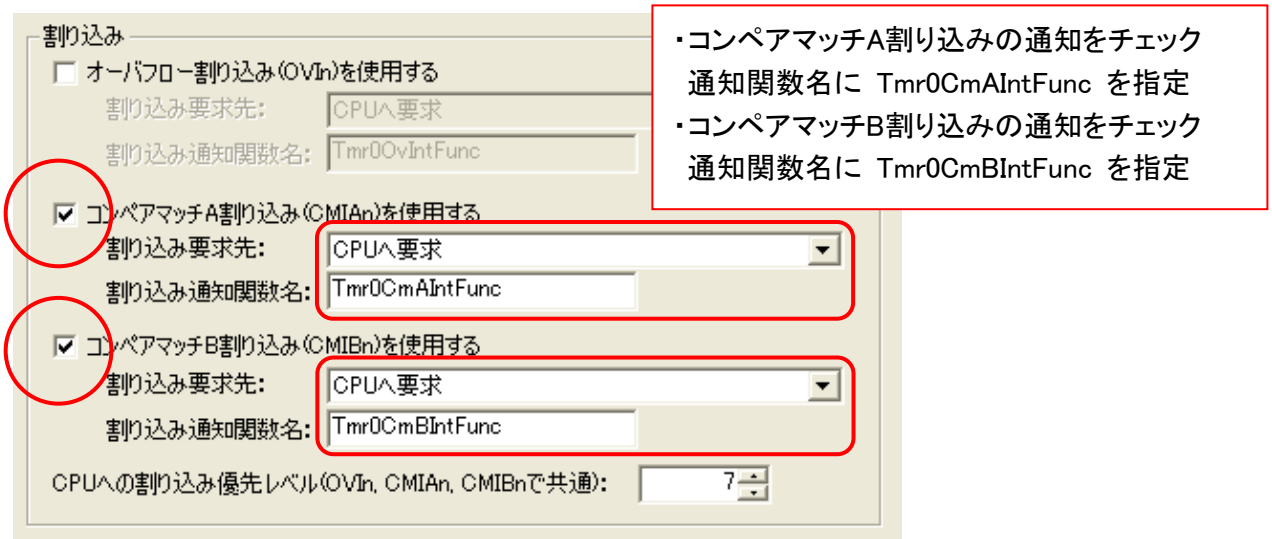
TMR の他の項目を以下の通り設定してください。



(7) TMR の設定-3 **PDG**

割り込み通知関数を設定します。

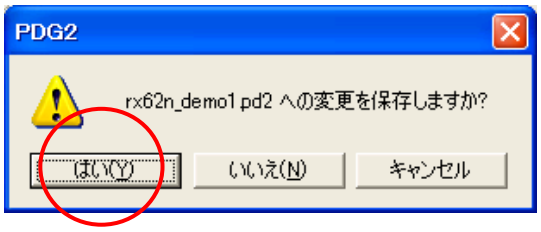
これらの関数は割り込みが発生すると呼ばれます。



(8) ソースファイルの生成 **PDG**

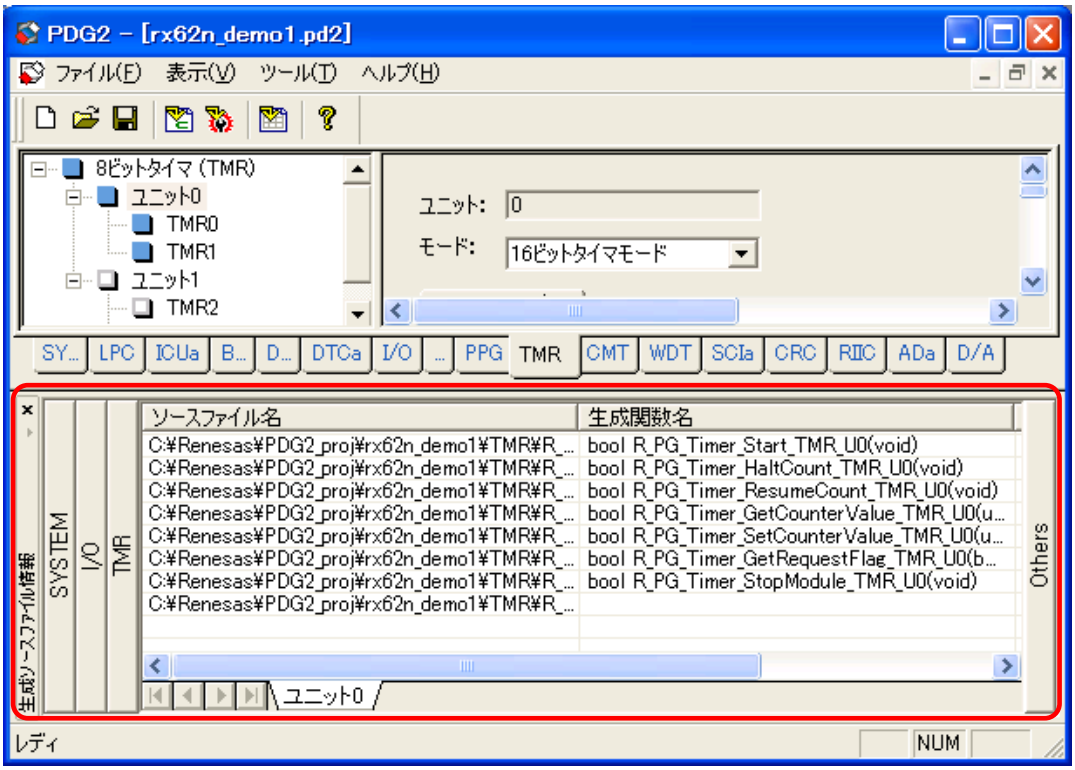
1. ツールバー上の  をクリックするとソースファイルが生成されます。

2. プロジェクトの保存を確認するダイアログボックスが表示されます。[はい]をクリックしてください。



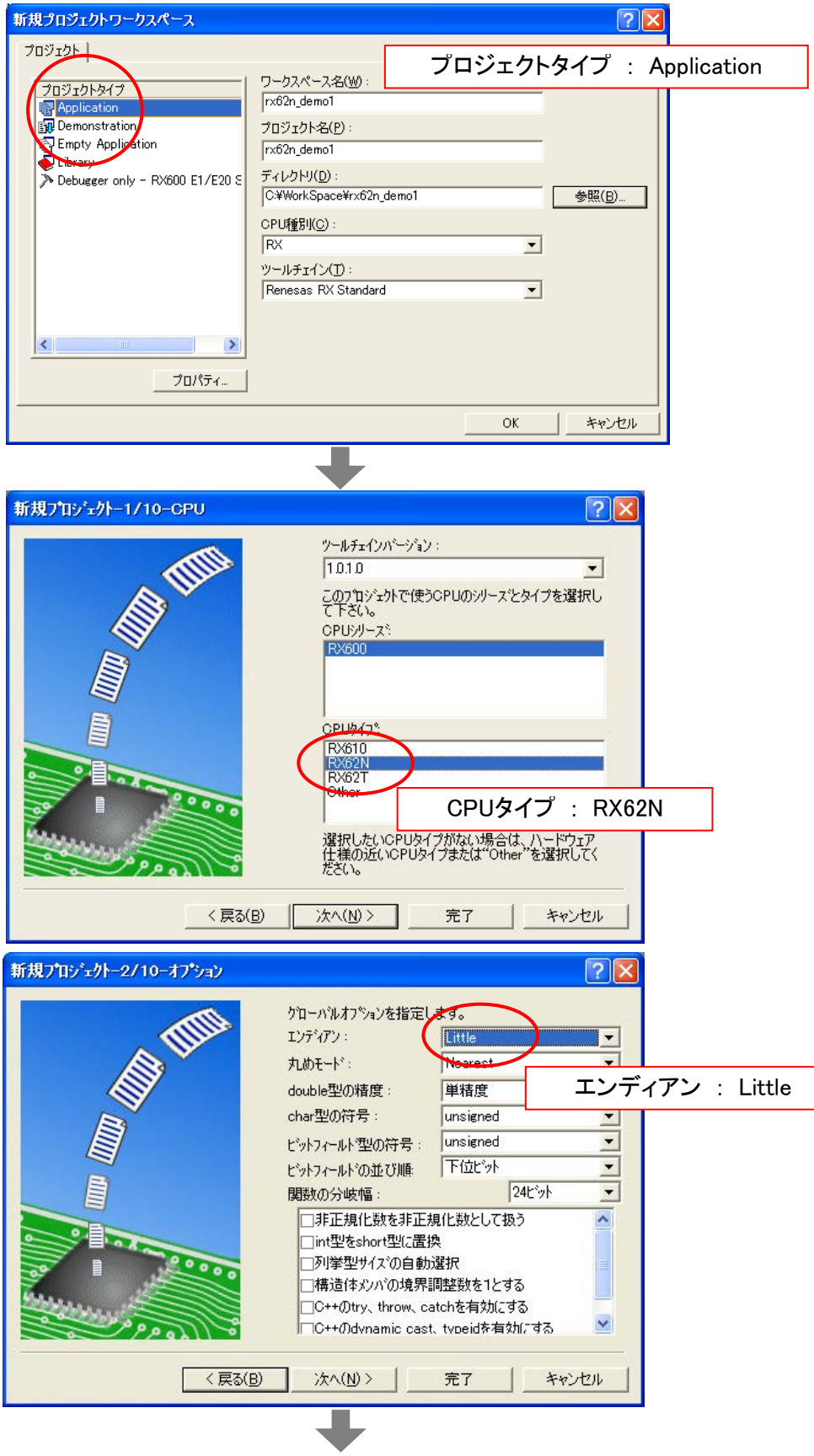
3. 登録の完了を示すダイアログボックスが表示されます。 [OK]をクリックしてください。

4. 生成された関数が下部のウィンドウに表示されます。
関数をダブルクリックするとソースファイルが開きます。

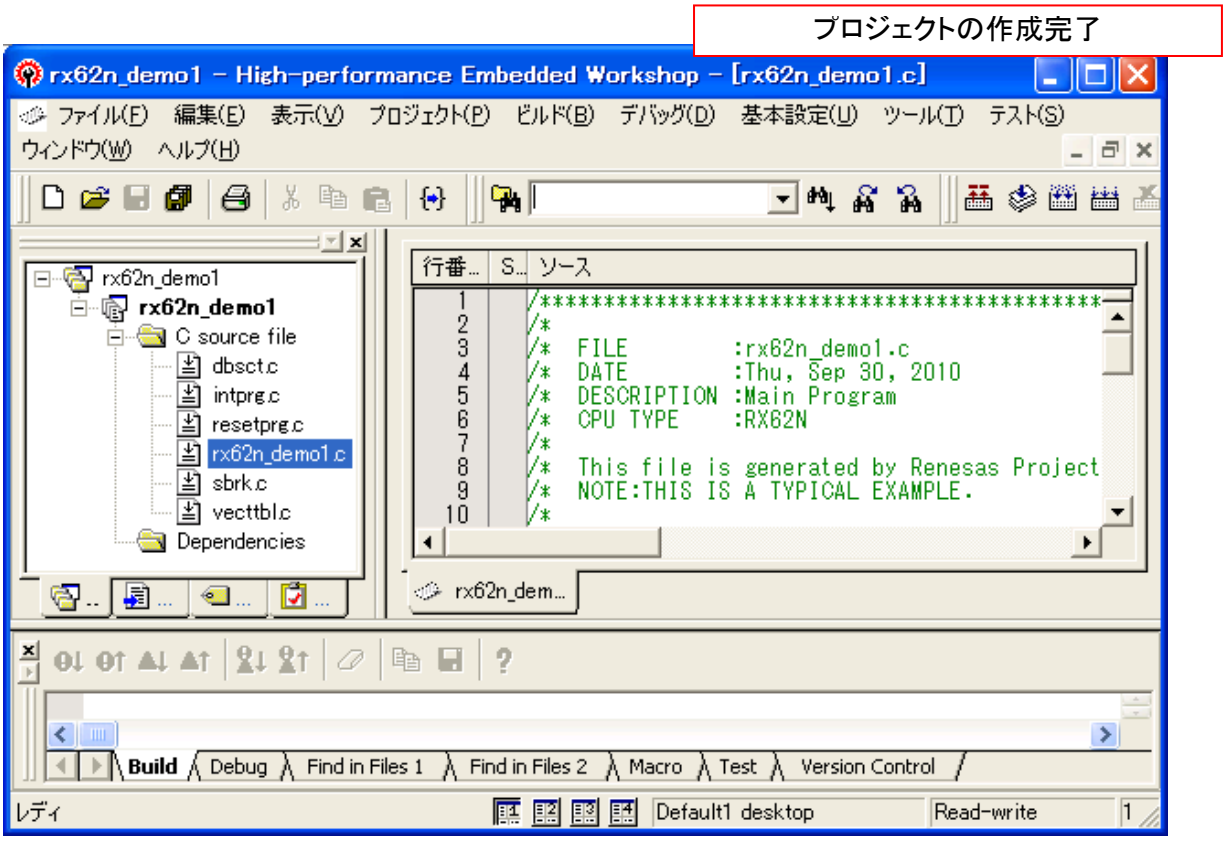


(9) HEW プロジェクトの準備 HEW

HEW を起動し、RX62N 用の新規ワークスペースを作成します。



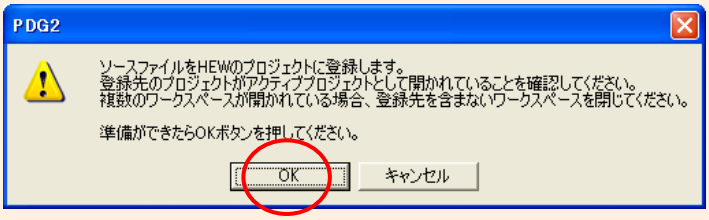

新規ﾌﾟﾛｼﾞｪｸﾄ-8/10-ﾃﾞﾊﾞｯｶﾞ
ターゲットエミュレータを指定
ﾀｰｹﾞｯﾄ :
RX600 E1/E20 SYSTEM
RX600 Simulator
ﾀｰｹﾞｯﾄﾀｲﾌﾟ :
RX600
ﾀｰｹﾞｯﾄCPU :
RX62N
< 戻る(B)
次へ(N) >
完了
ｷｬﾝｾﾙ


プロジェクトの作成完了
rx62n_demo1 - High-performance Embedded Workshop - [rx62n_demo1.c]
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) 基本設定(U) ツール(T) テスト(S)
ウィンドウ(W) ヘルプ(H)
rx62n_demo1
rx62n_demo1
C source file
dbsct.c
intprg.c
resetprg.c
rx62n_demo1.c
sbrk.c
vecttbl.c
Dependencies
行番... S... ソース
/*
/* FILE :rx62n_demo1.c
/* DATE :Thu, Sep 30, 2010
/* DESCRIPTION :Main Program
/* CPU TYPE :RX62N
/*
/* This file is generated by Renesas Project
/* NOTE:THIS IS A TYPICAL EXAMPLE.
/*
rx62n_dem...
Build
Debug
Find in Files 1
Find in Files 2
Macro
Test
Version Control
レディ
Default1 desktop
Read-write

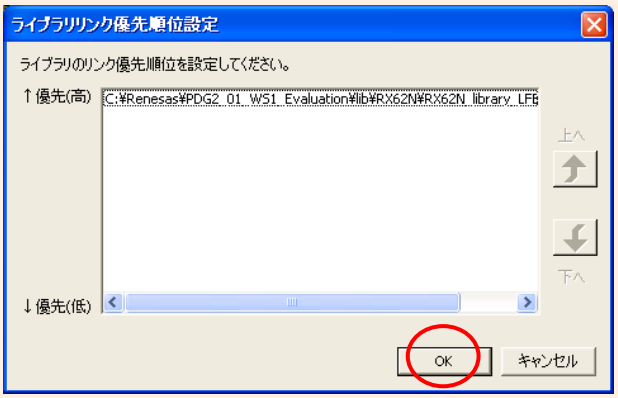
(10) PDG 生成ファイルの HEW への登録

PDG

1. ファイルを HEW に追加するには
   PDG のツールバー上の  をクリックします。
2. 確認のダイアログボックスで[OK]をクリックしてください。

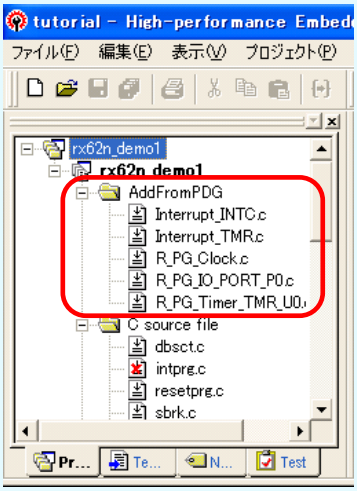


3. RPDL とのリンク設定のためのダイアログが開きます。
   複数の lib ファイルとリンクする場合、このダイアログ上でリンク順を設定できます。



HEW

4. HEW のプロジェクトにファイルが追加されます。
   追加されたファイルは
   AddFromPDG
   フォルダに格納されます。



ソースファイルはHEW Target Server経由で追加されます。追加を実行する前にHEW Target Serverが設定されていることを確認してください。詳細についてはPDGのユーザーズマニュアルを参照してください。

(11) プログラムの作成 **HEW**

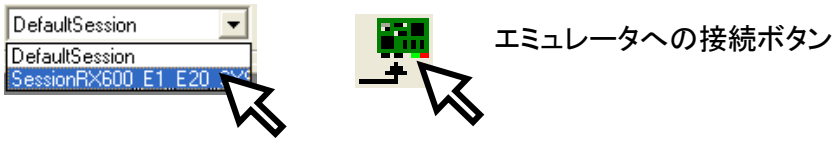
HEW 上で main 関数の部分を変更し、以下のプログラムを作成してください。

```
//Include "R_PG_<PDGプロジェクト名>.h"
#include "R_PG_rx62n_demo1.h"

void main(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //ポートP05の設定
    R_PG_IO_PORT_Set_P0();

    //TMRユニット0を設定しカウントを開始
    R_PG_Timer_Start_TMR_U0();

    while(1);
}


//コンペアマッチA割り込みの通知関数
void TmrOCmAIntFunc(void)
{
    //LED点灯
    R_PG_IO_PORT_Write_P05(0);
}

//コンペアマッチB割り込みの通知関数
void TmrOCmBIntFunc(void)
{
    //LED消灯
    R_PG_IO_PORT_Write_P05(1);
}
```

(12) エミュレータの接続、プログラムのビルド、実行 **HEW**

1. エミュレータを接続する前に、RSK+ボード上のSW4/Pin3がON(CPUはリトルエンディアン)にセットされていることを確認してください。

2. エミュレータに接続してください。

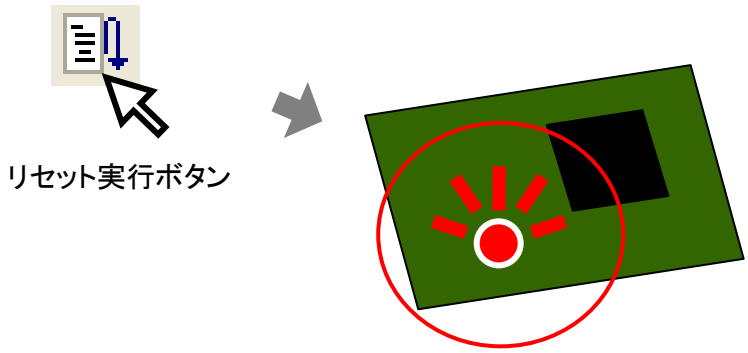


3. RPDLのライブラリとインクルードディレクトリはソースの登録時に設定されているため、[ビルド]ボタンをクリックするだけでビルドすることができます。

注意: RXファミリC/C++コンパイラパッケージ V.1.01以上を使用している場合、ビルド時にエラーメッセージが出力される場合があります。詳細については 5.(5) を参照してください。

4. プログラムをダウンロードしてください。

5. プログラムを実行し、RSK+ボード上のLEDを確認してください。

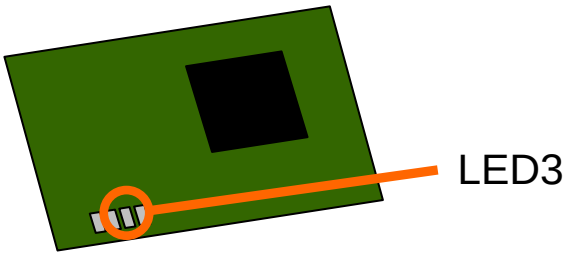


6.2 マルチファンクションタイマパルスユニット 2(MTU2)のPWM波でLEDを点滅

RX62N RSK+ボードでは P34 端子に LED3 が接続されています。P34 はマルチファンクションタイマパルスユニット 2(MTU2)の PWM 波形出力端子(MTIOC0A) としても使用することができます。このチュートリアルではマルチファンクションタイマパルスユニット 2(MTU2)を PWM モード 1 で動作させ、その出力パルスで LED を点滅させます。

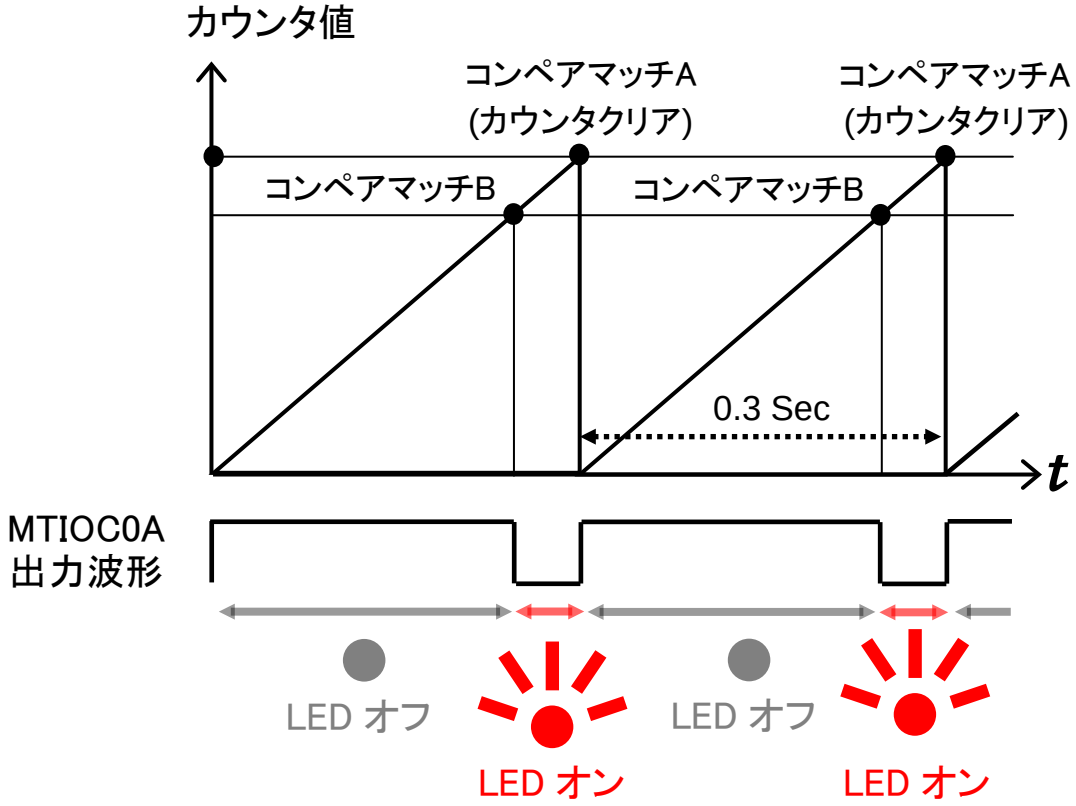
使用する RSK+ボード上に P34(MTIOC0A)の有効/無効を切り替えるスイッチがある場合は有効にしてください。

LED3はP34から0出力で点灯、1出力で消灯します



MTU2のチャネル0(MTU0)をPWMモード1で動作させます。PWMモード1は、コンペアマッチAおよびBでMTIOC0Aの出力レベルを制御するモードです。

設定するタイマの動作
- ・コンペアマッチBで0出力 → LED点灯
- ・コンペアマッチAで1出力 → LED消灯
- ・コンペアマッチAでカウンタクリア（カウンタクリア周期は0.3msec）

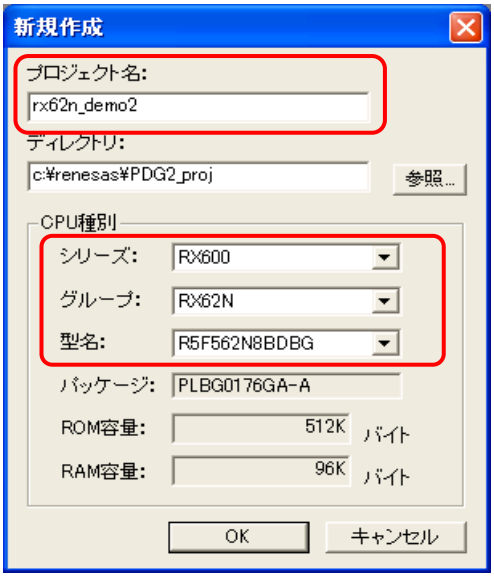


(1) PDG プロジェクトの作成 **PDG**

プロジェクト名に“rx62n_demo2”を指定し、PDG の新規プロジェクトを作成してください。(プロジェクト作成方法の詳細については「6.1 (1)PDG プロジェクトの作成」を参照してください。)

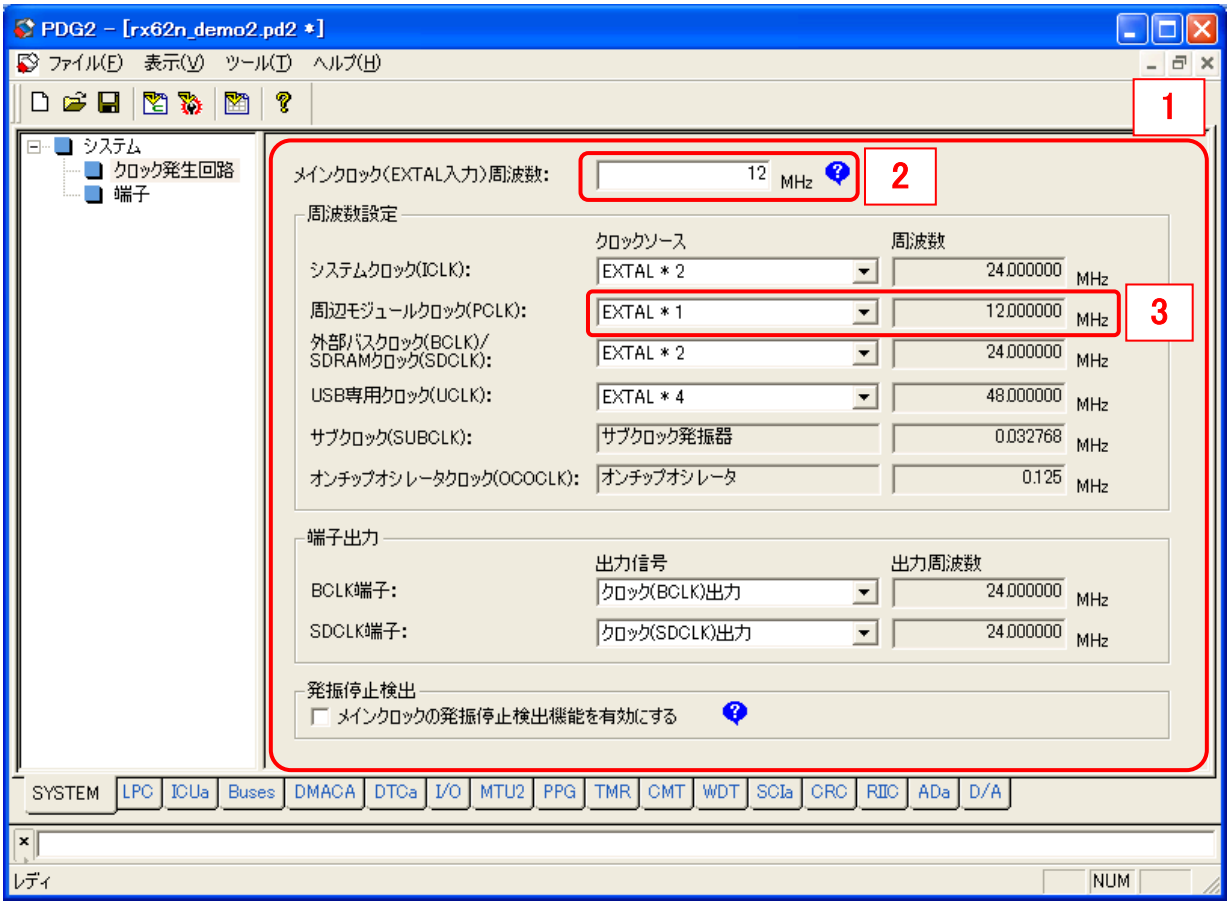
CPU 種別は以下の通り設定してください。但し使用する RSK+ボードに他の型名のチップが搭載されている場合は、ボードに合わせて設定してください。

シリーズ : RX600
グループ : RX62N
型名 : R5F562N8BDBG



(2) クロックの設定 **PDG**

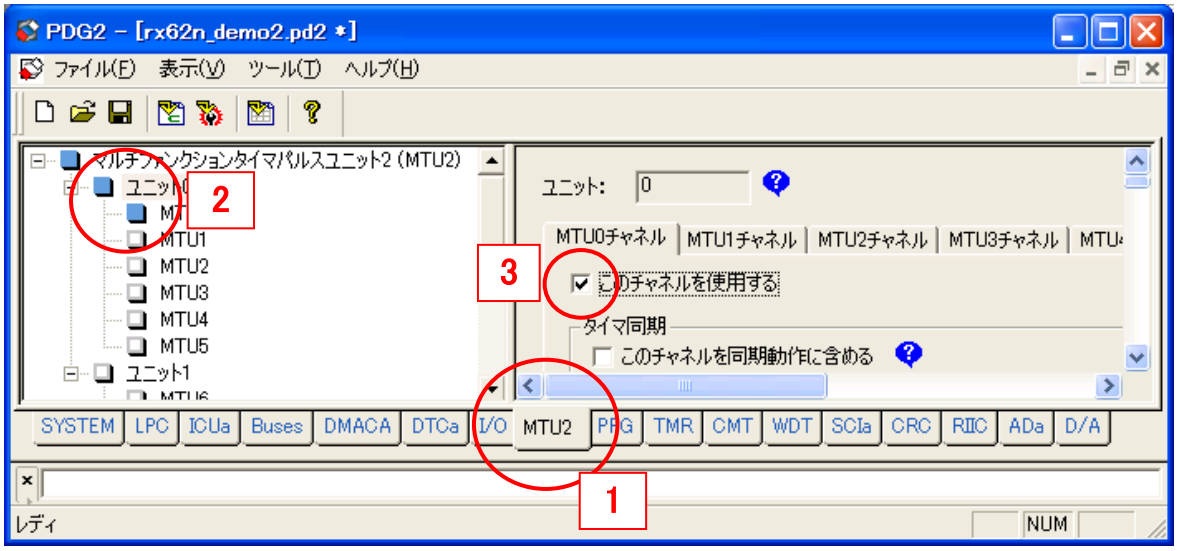
1. プロジェクトを作成するとクロック設定ウィンドウが開きます。設定画面上の や などのアイコンについては、「6.1 (1)初期状態」を参照してください。
2. RSK+ボードの外部入力周波数は 12MHz です。“12”と入力してください。
3. 周辺モジュールクロック(PCLK)は 12MHzで使用します。
   PCLK の倍率に“EXTAL *1”を選択し、PCLK 周波数を 12MHzに設定してください。



(3) MTU2 の設定-1 PDG

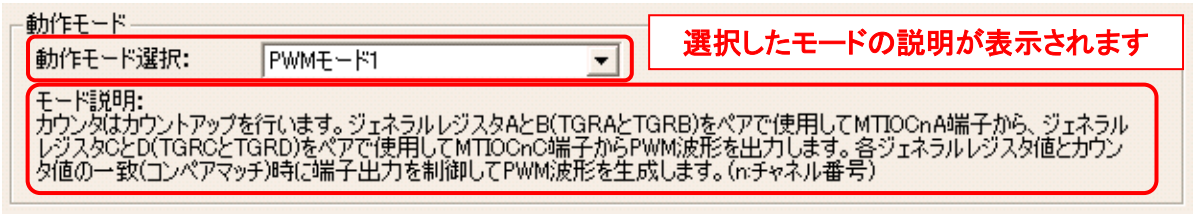
MTU2 チャネル 0(MTU0)を設定します。

1. [MTU2] タブを選択してください。
2. [MTU0] を選択してください。
3. [このチャネルを使用する] をチェックしてください。



(4) MTU2 の設定-2 PDG

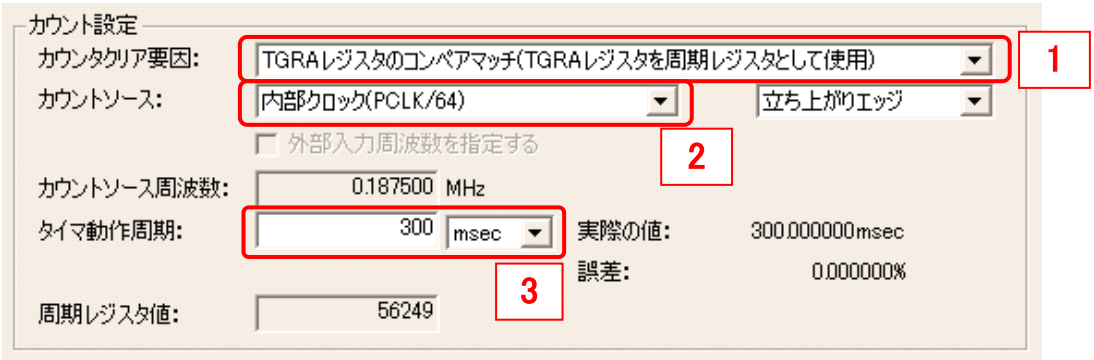
動作モードに[PWM モード 1]を指定してください。



(5) MTU2 の設定-3 PDG

以下の通りカウンタの動作を設定してください。

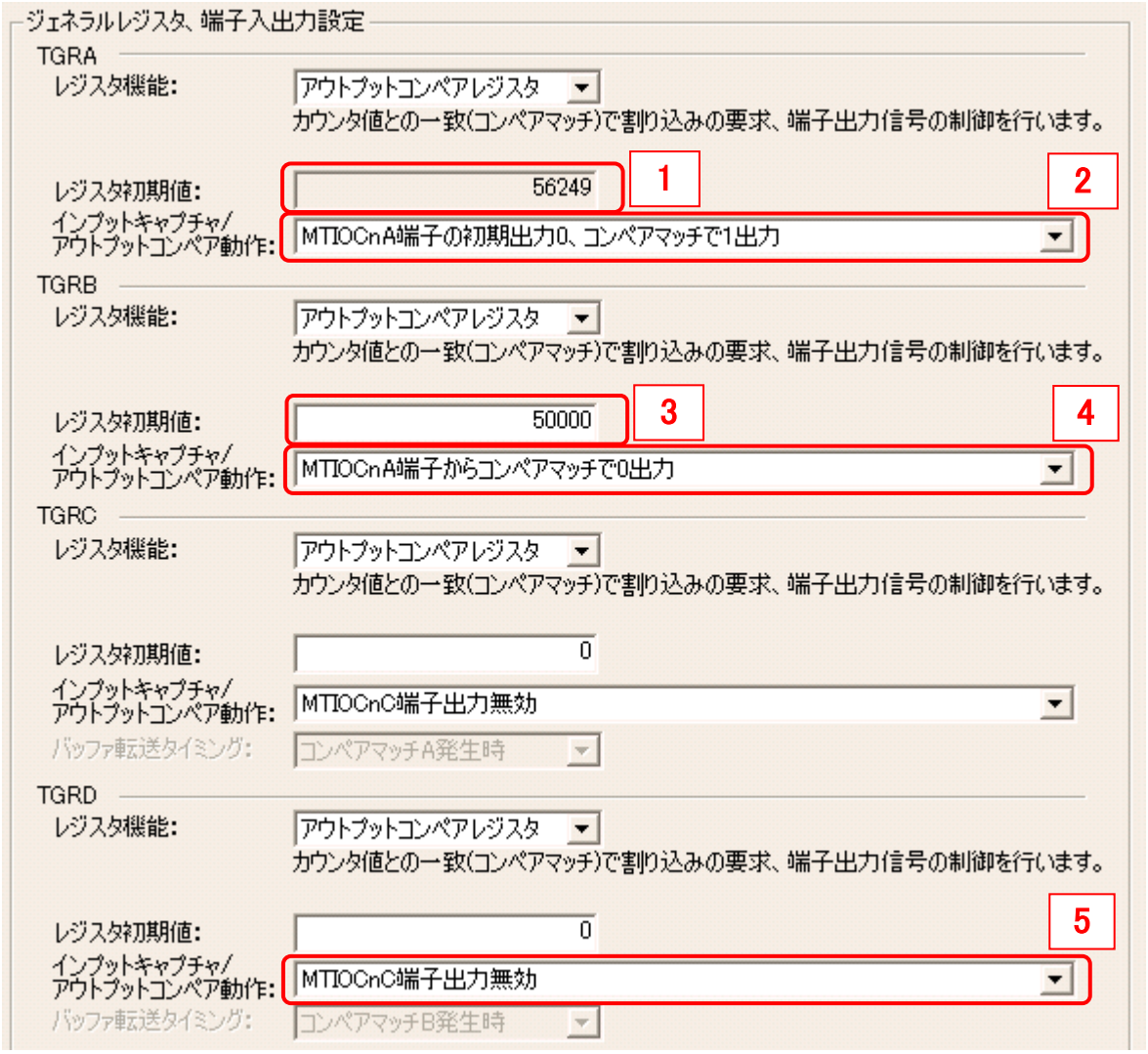
1. カウンタクリア要因に[TGRAレジスタのコンペアマッチ] を選択してください。
2. カウントソースに[内部クロック(PCLK/64)] を選択してください。
3. タイマ動作周期に300msecを指定してください。



(6) MTU2 の設定-4 PDG

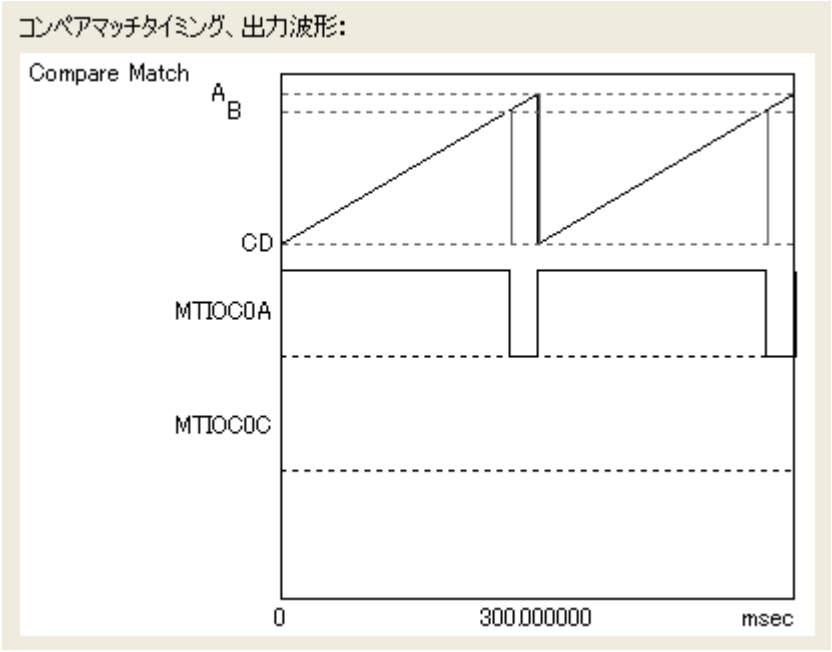
以下の通りジェネラルレジスタを設定してください。

1. カウント設定においてカウンタクリア要因にコンペアマッチAを指定したので、TGRAの値はカウントソース周波数と入力したタイマ動作周期を元に算出されます。
2. TGRAのアウトプットコンペア動作に[MTIOCnA端子の初期出力0、コンペアマッチで1出力]を選択してください。
3. TGRBのレジスタ初期値に50000を設定してください。
4. TGRAのアウトプットコンペア動作に[MTIOCnA端子からコンペアマッチで0出力]を選択してください。
5. TGRCとTGRDをペアで使用すると、MTIOCnC端子からPWM出力することが可能です。ここでは使用しませんので、TGRDのアウトプットコンペア動作には[MTIOCnC端子出力無効]を選択してください。



(7) MTU2 の設定-5 **PDG**

設定内容に応じて、コンペアマッチのタイミングと出力波形が図示されます。



(8) ソースファイルの生成 **PDG**

ツールバー上の  をクリックしてソースファイルを生成してください。 ソースファイル生成の詳細については「6.1 (8)ソースファイルの生成」を参照してください。

(9) HEW プロジェクトの準備 **HEW**

HEW を起動し、RX62N 用のワークスペースを作成してください。作成方法については「6.1 (9)HEW プロジェクトの準備」を参照してください。

(10) PDG 生成ファイルの HEW への登録 **PDG**

ツールバー上の  をクリックして PDG が生成したソースファイルを HEW のプロジェクトに登録してください。ソースファイル生成の詳細については「6.1 (10)PDG 生成ファイルの HEW への登録」を参照してください。

(11) プログラムの作成 HEW

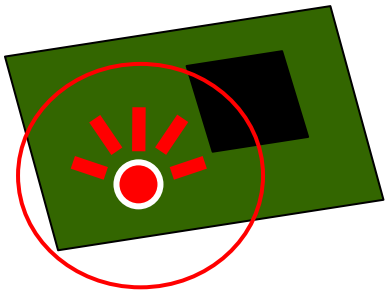
HEW 上で main 関数の部分を変更し、以下のプログラムを作成してください。

```
//Include "R_PG_<PDGプロジェクト名>.h"
#include "R_PG_rx62n_demo2.h"

void main(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //MTU2チャネル0の設定
    R_PG_Timer_Set_MTU_U0_C0();

    // MTU2チャネル0のカウント開始
    R_PG_Timer_StartCount_MTU_U0_C0();

    while(1);
}
```

(12) エミュレータの接続、プログラムのビルド、実行 HEW

作成したプログラムをビルドし、実行してください。 LED が点滅します。

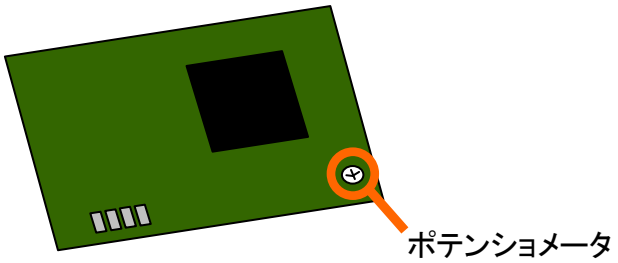
エミュレータの接続、プログラムのビルド、実行の方法については「6.1 (12) エミュレータの接続、プログラムのビルド、実行」 を参照してください。



注意: RXファミリC/C++コンパイラパッケージ V.1.01以上を使用している場合、ビルド時にエラーメッセージが出力される場合があります。詳細については 5.(5) を参照してください。

6.3 10 ビットA/Dコンバータ（ADa）の連続スキャン

RX62N RSK+ボードではポテンショメータが AN0 アナログ入力端子に接続されています。
このチュートリアルでは AD0 の A/D 変換を連続スキャンし、A/D 変換結果を HEW 上でリアルタイムに確認します。



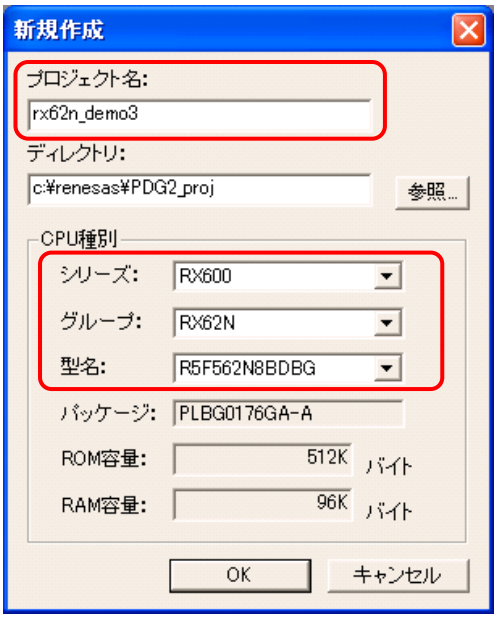
使用する RSK+ボード上に AN0 の有効/無効を切り替えるスイッチがある場合は有効にしてください。

(1) PDG プロジェクトの作成 PDG

プロジェクト名に“rx62n_demo3”を指定し、PDG の新規プロジェクトを作成してください。(プロジェクト作成方法の詳細については「6.1 (1)PDG プロジェクトの作成」を参照してください。)

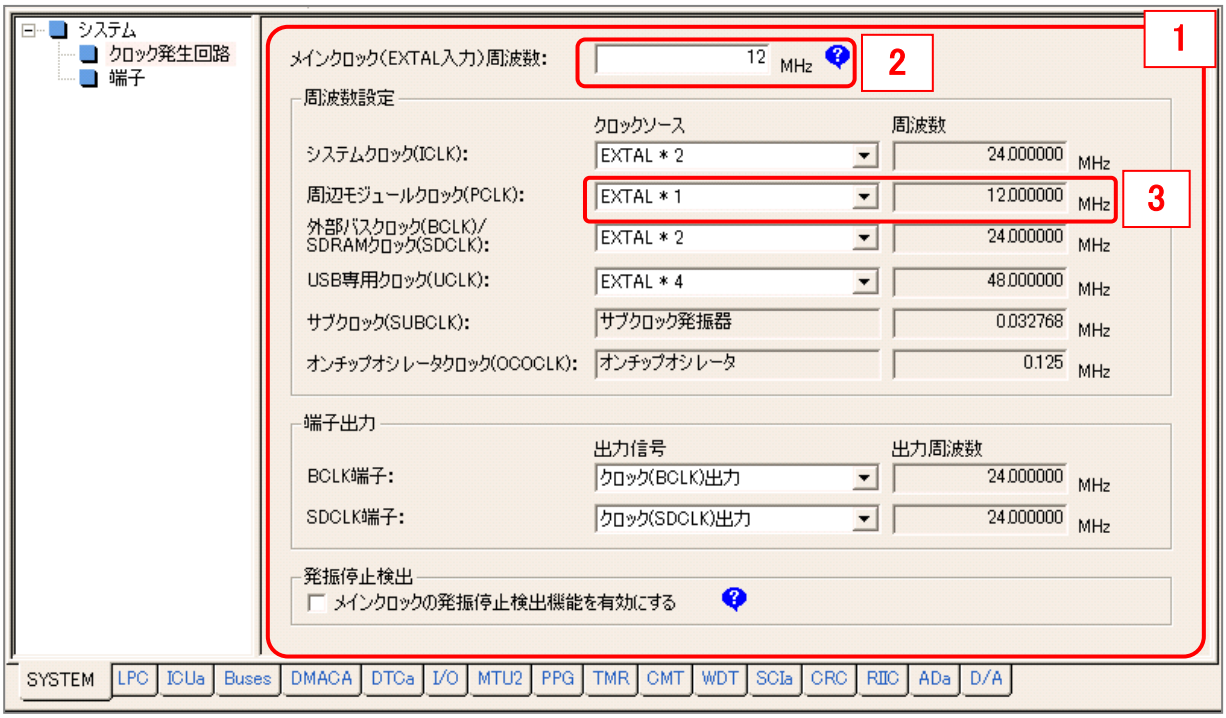
CPU 種別は以下の通り設定してください。但し使用する RSK+ボードに他の型名のチップが搭載されている場合は、ボードに合わせて設定してください。

シリーズ : RX600
グループ : RX62N
型名 : R5F562N8BDBG



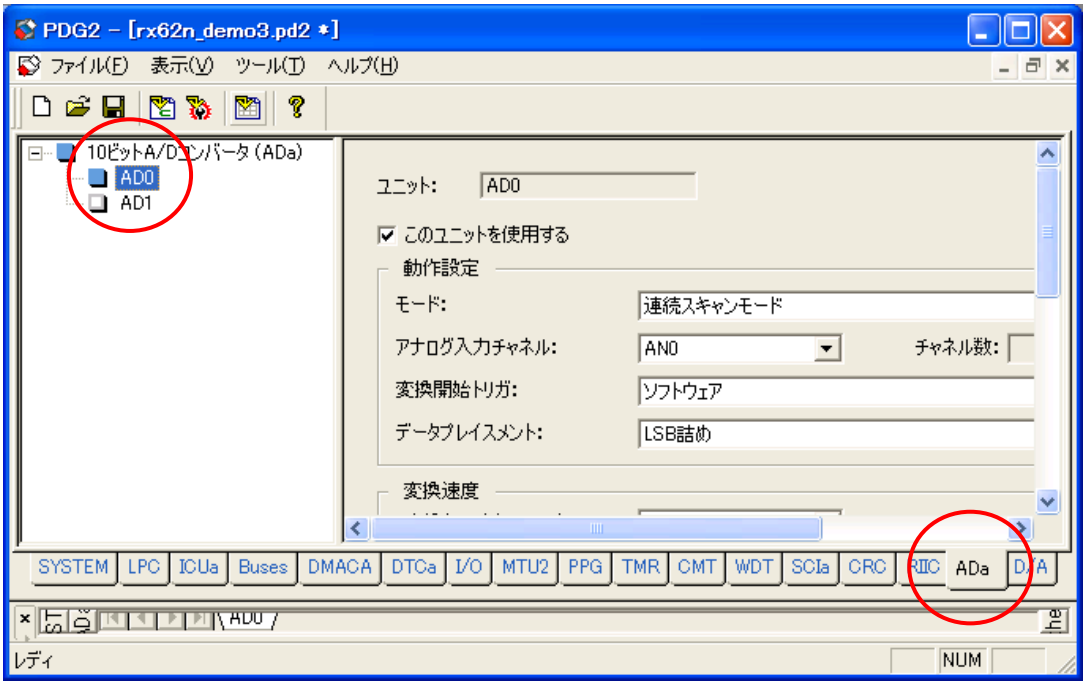
(2) クロックの設定 **PDG**

1. プロジェクトを作成するとクロック設定ウィンドウが開きます。設定画面上の や などのアイコンについては、「6.1 (1)初期状態」を参照してください。
2. RSK+ボードの外部入力周波数は 12MHz です。“12”と入力してください。
3. 周辺モジュールクロック(PCLK)は 12MHzで使用します。
   PCLK の倍率に“EXTAL *1”を選択し、PCLK 周波数を 12MHzに設定してください。



(3) A/D 変換器の設定-1 **PDG**

ADa タブを選択し、ツリー表示上で AD0 を選択してください。



(4) A/D 変換器の設定-2 PDG

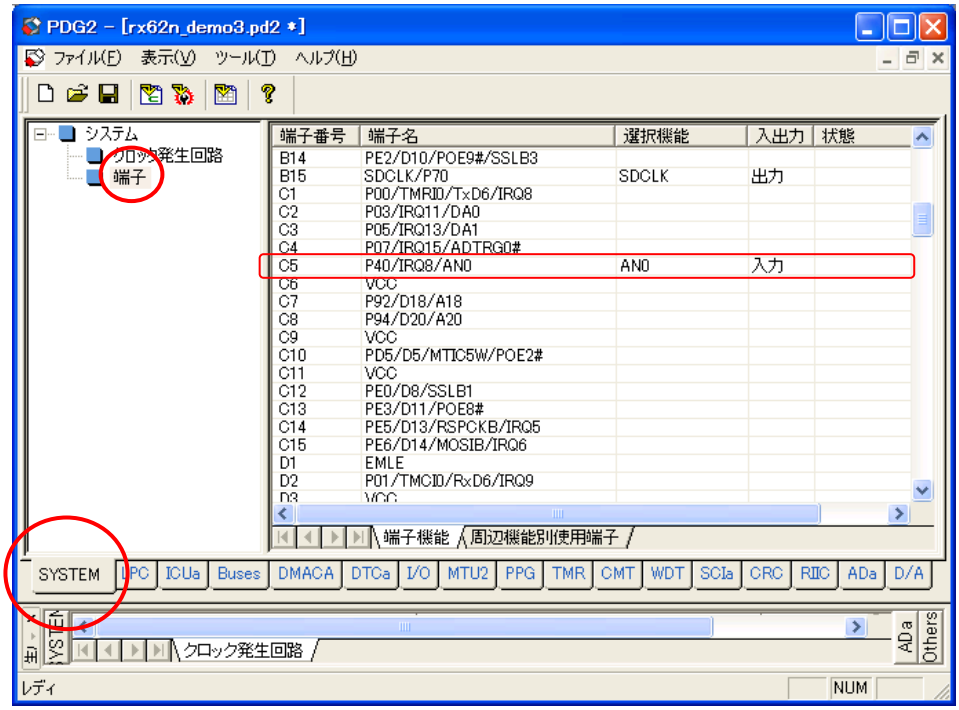
AD0 を以下の通り設定してください。

1. [このユニットを使用する]をチェック
2. モード ： [連続スキャンモード]
3. 変換開始トリガ ： [ソフトウェア]
4. アナログ入力チャネル ： [AN0]
5. 変換クロック ： [内部クロック(PCLK/2)]
6. サンプリングステートレジスタ値 ： 25 (初期値)
7. [A/D変換終了割り込み(ADIn)を使用する]をチェック
8. A/D変換終了割り込み通知関数名 ： Ad0IntFunc

(5) 端子使用状況の確認 PDG

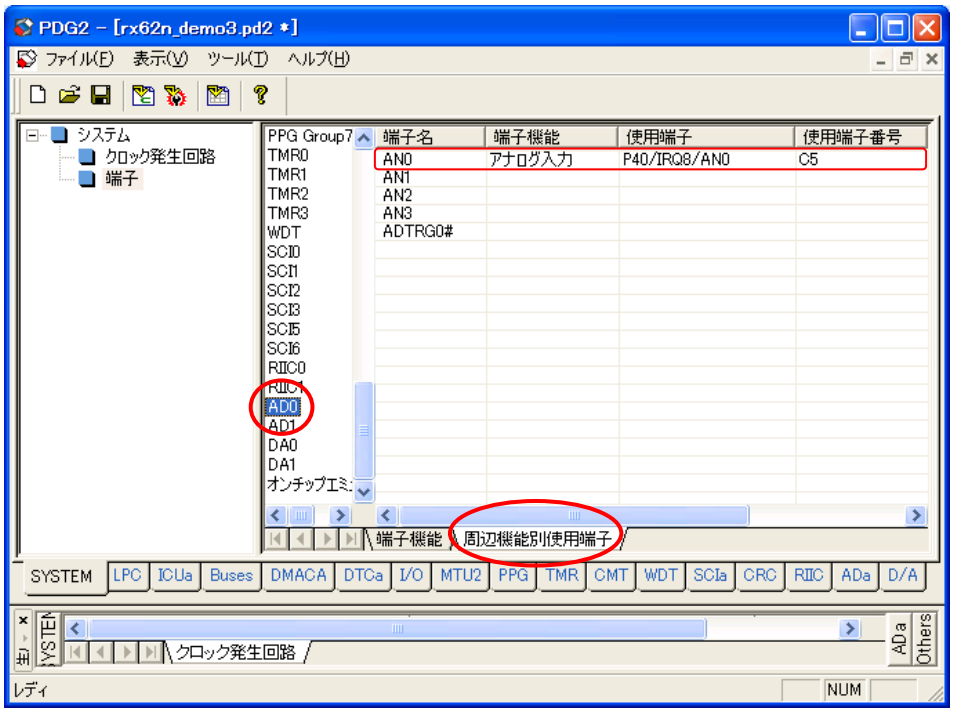
・端子機能ウィンドウで端子の使用状況を確認することができます。

1. AD0を設定後、[SYSTEM]タブを選択し、ツリー表示上で[端子]を選択してください。
2. [端子機能]ウィンドウ上で C5 ピンが AN0 として使用されていることを確認してください。



・周辺機能ごとの端子の使用状況は周辺機能別使用端子ウィンドウで確認することができます。

[周辺機能別使用端子]タブをクリックし、周辺機能の一覧からAD0を選択してAN0端子の使用状況を確認してください。



(6) ソースファイルの生成 PDG

ツールバー上の  をクリックしてソースファイルを生成してください。 ソースファイル生成の詳細については「6.1 (8)ソースファイルの生成」を参照してください。

(7) HEW プロジェクトの準備 HEW

HEW を起動し、RX62N 用のワークスペースを作成してください。作成方法については「6.1 (9)HEW プロジェクトの準備」を参照してください。

(8) PDG 生成ファイルの HEW への登録 PDG

ツールバー上の  をクリックして PDG が生成したソースファイルを HEW のプロジェクトに登録してください。ソースファイル生成の詳細については「6.1 (10)PDG 生成ファイルの HEW への登録」を参照してください。

(9) プログラムの作成 HEW

HEW 上で main 関数の部分を変更し、以下のプログラムを作成してください。

```
//Include "R_PG_<PDGプロジェクト名>.h"
#include "R_PG_rx62n_demo3.h"

void main(void)
{
    //クロックの設定
    R_PG_Clock_Set();

    //A/Dコンバータ AD0の設定
    R_PG_ADC_10_Set_AD0();

    //AD0のA/D変換開始
    R_PG_ADC_10_StartConversionSW_AD0();

    while(1);
}

//変換結果格納先変数
uint16_t result;

//AD0変換終了割り込み通知関数
void Ad0IntFunc(void)
{
    //変換結果の取得
    R_PG_ADC_10_GetResult_AD0(&result);
}
```

(10) エミュレータの接続、プログラムのビルド、ダウンロード **HEW**

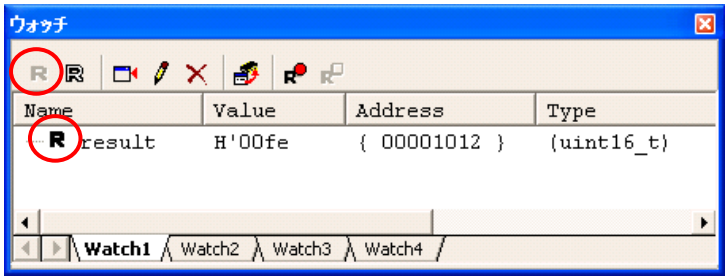
作成したプログラムをビルドし、ダウンロードしてください。

エミュレータの接続、プログラムのビルド方法については「6.1 (12) エミュレータの接続、プログラムのビルド、実行」 を参照してください。

注意: RXファミリC/C++コンパイラパッケージ V.1.01以上を使用している場合、ビルド時にエラーメッセージが出力される場合があります。詳細については 5.(5) を参照してください。

(11) A/D 変換結果格納変数のウォッチウィンドウ登録 **HEW**

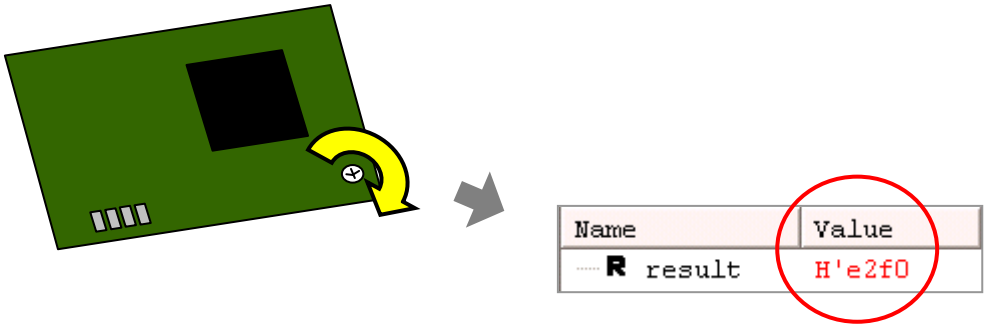
HEW のウォッチウィンドウを開き、変数 ″result″ を登録してください。″result″をリアルタイム更新に設定すると、実行中に値の変化を確認することができます。



(12) プログラムの実行と A/D 変換結果の確認 **HEW**

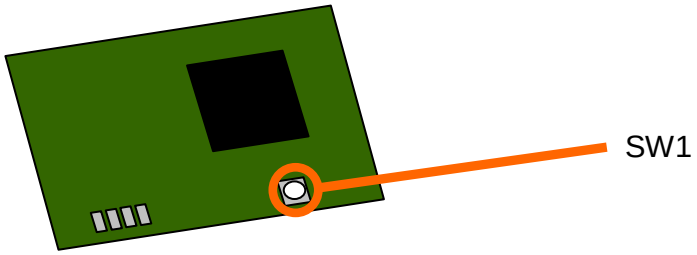
プログラを実行し、実行中ポテンショメータを回してアナログ入力電圧を変動させてください。

ウォッチウィンドウ上の ″result″ の値が変化します。



6.4 IRQによるDTC転送のトリガ

RX62N RSK+ボードではスイッチ 1 (SW1) が IRQ8 外部割込み入力端子に接続されています。
このチュートリアルでは IRQ8 をトリガとした DTC 転送を行います。



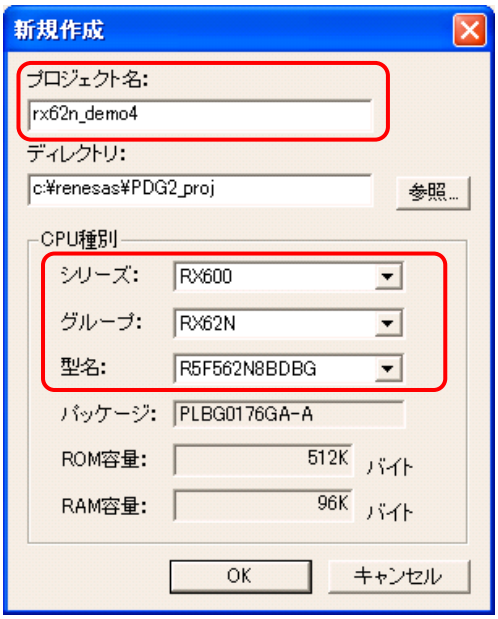
使用する RSK+ボード上に IRQ8 の有効/無効を切り替えるスイッチがある場合は有効にしてください。

(1) PDG プロジェクトの作成 PDG

プロジェクト名に“rx62n_demo4”を指定し、PDG の新規プロジェクトを作成してください。(プロジェクト作成方法の詳細については「6.1 (1)PDG プロジェクトの作成」を参照してください。)

CPU 種別は以下の通り設定してください。但し使用する RSK+ボードに他の型名のチップが搭載されている場合は、ボードに合わせて設定してください。

シリーズ : RX600
グループ : RX62N
型名 : R5F562N8BDBG

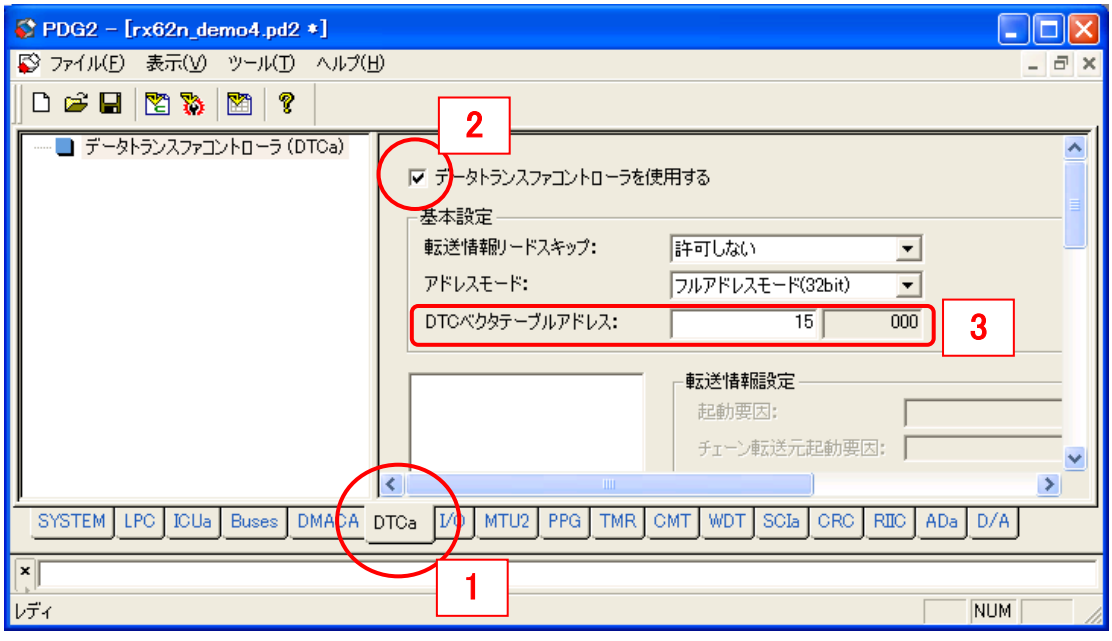


(2) クロックの設定 **PDG**

1. プロジェクトを作成するとクロック設定ウィンドウが開きます。設定画面上の や などのアイコンについては、「6.1 (1)初期状態」を参照してください。
2. RSK+ボードの外部入力周波数は 12MHz です。“12”と入力してください。

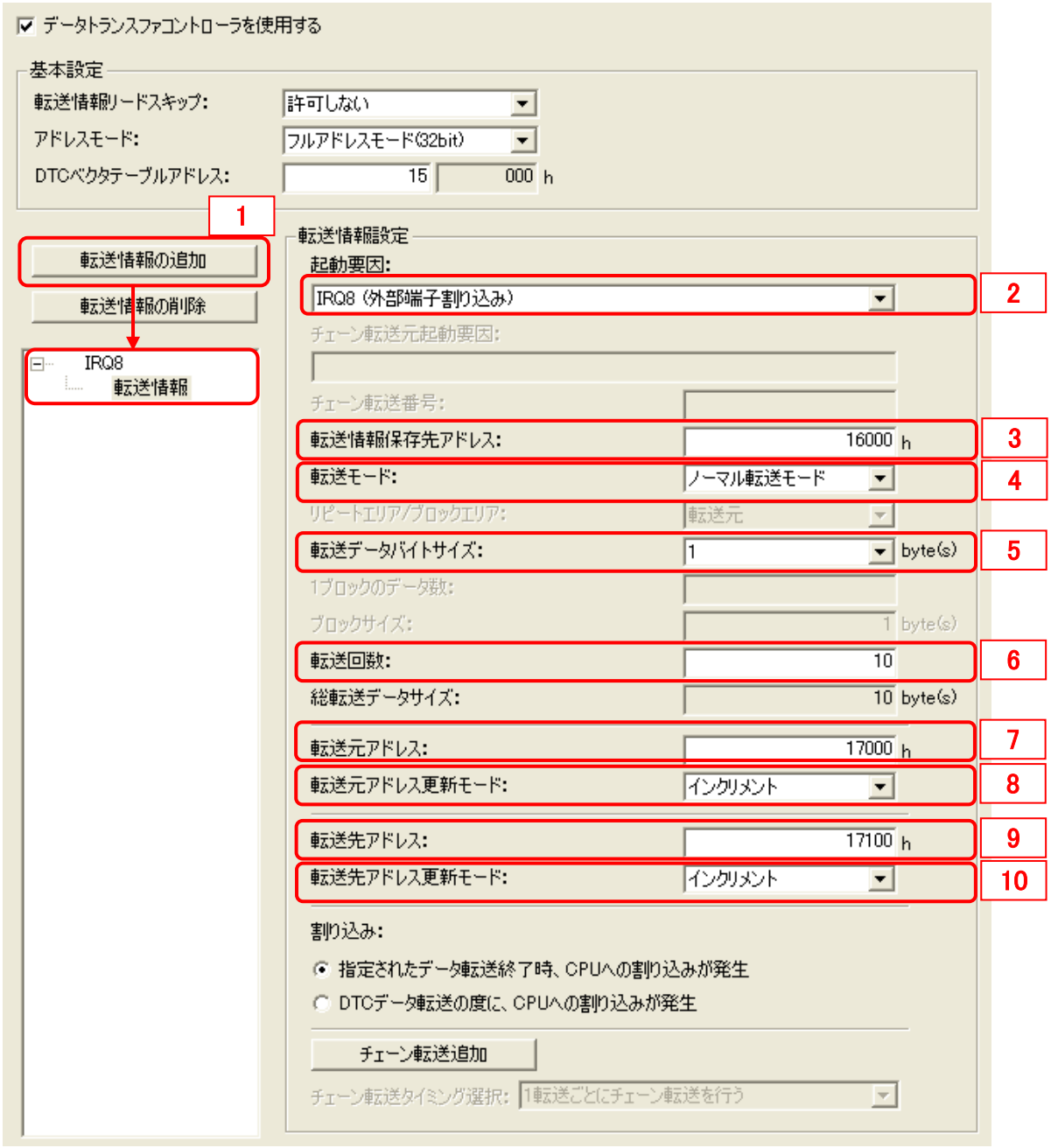
(3) DTC の設定-1 **PDG**

1. DTCa タブを選択し、DTC の設定ウィンドウを開いてください。
2. [データトランスファコントローラを使用する]をチェックしてください。
3. DTC ベクタテーブルアドレスは 15000 に配置します。15 と入力してください。



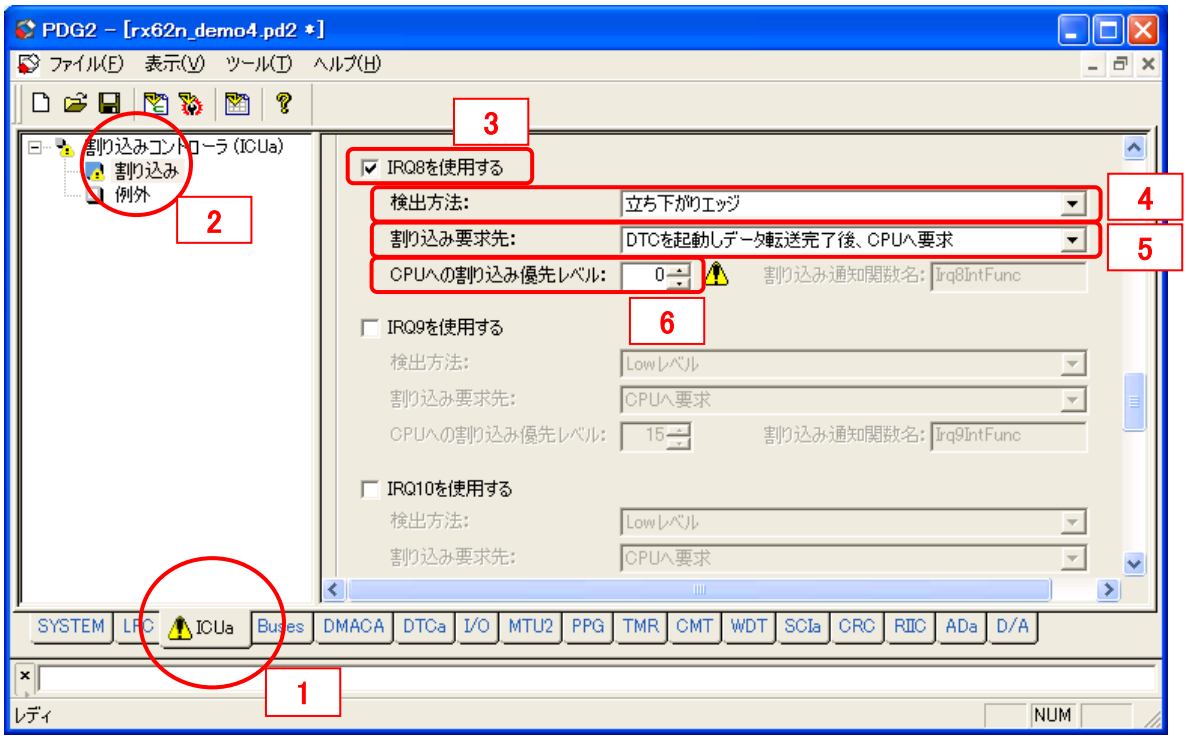
(4) DTC の設定-2 **PDG**

1. [転送情報の追加]ボタンをクリックすると、転送情報が追加されます。
2. 起動要因に[IRQ8 (外部端子割り込み)]を指定してください。
3. 転送情報保存先アドレスに 16000 を指定してください。
4. 転送モードに[ノーマル転送モード]を指定してください。
5. 転送データバイトサイズに 1 を指定してください。
6. 転送回数に 10 を指定してください。
7. 転送元アドレスに 17000 を指定してください。
8. 転送元アドレス更新モードに[インクリメント]を指定してください。
9. 転送先アドレスに 17100 を指定してください。
10. 転送先アドレス更新モードに[インクリメント]を指定してください。



(5) IRQ の設定 PDG

1. ICUa タブを選択してください。
2. ツリー表示上で[割り込み]を選択してください。
3. [IRQ8 を使用する]をチェックしてください。
4. 検出方法に[立ち下がりエッジ]を指定してください。
5. 割り込み要求先に[DTC を起動しデータ転送完了後、CPU へ要求]を指定してください。
6. IRQ8 の CPU 割り込みは使用しません。割り込み優先レベルに 0 を指定してください。



(6) ソースファイルの生成 PDG

ツールバー上の  をクリックしてソースファイルを生成してください。 ソースファイル生成の詳細については「6.1 (8)ソースファイルの生成」を参照してください。

(7) HEW プロジェクトの準備 HEW

HEW を起動し、RX62N 用のワークスペースを作成してください。作成方法については「6.1 (9)HEW プロジェクトの準備」を参照してください。

(8) PDG 生成ファイルの HEW への登録 PDG

ツールバー上の  をクリックして PDG が生成したソースファイルを HEW のプロジェクトに登録してください。ソースファイル生成の詳細については「6.1 (10)PDG 生成ファイルの HEW への登録」を参照してください。

(9) プログラムの作成 **HEW**

HEW 上で main 関数の部分を変更し、以下のプログラムを作成してください。

```
//Include "R_PG_<PDGプロジェクト名>.h"
#include "R_PG_rx62n_demo4.h"

//DTCベクタテーブル
#pragma address dtc_vector_table = 0x00015000
uint32_t dtc_vector_table [256];

//DTC転送情報保存先(IRQ8)
#pragma address dtc_transfer_data_IRQ0 = 0x00016000
uint32_t dtc_transfer_data_IRQ0 [2];

//転送元
#pragma address dtc_src_data = 0x00017000
uint8_t dtc_src_data [10] = "ABCDEFGHIJ";

//転送先
#pragma address dtc_dest_data = 0x00017100
uint8_t dtc_dest_data [10];

void main(void)
{
    //転送先初期化
    int i;
    for(i=0; i<10; i++ ){
        dtc_dest_data[i] = 0;
    }

    //クロックの設定
    R_PG_Clock_Set();

    //DTCの設定(ベクタテーブルアドレスなど)
    R_PG_DTC_Set();

    //DTCの設定(IRQ8をトリガとする転送の設定)
    R_PG_DTC_Set_IRQ8();

    //IRQ8の設定
    R_PG_ExtInterrupt_Set_IRQ8();

    //DTC転送開始
    R_PG_DTC_Activate();
    while(1);
}
```

(10) エミュレータの接続、プログラムのビルド、ダウンロード **HEW**

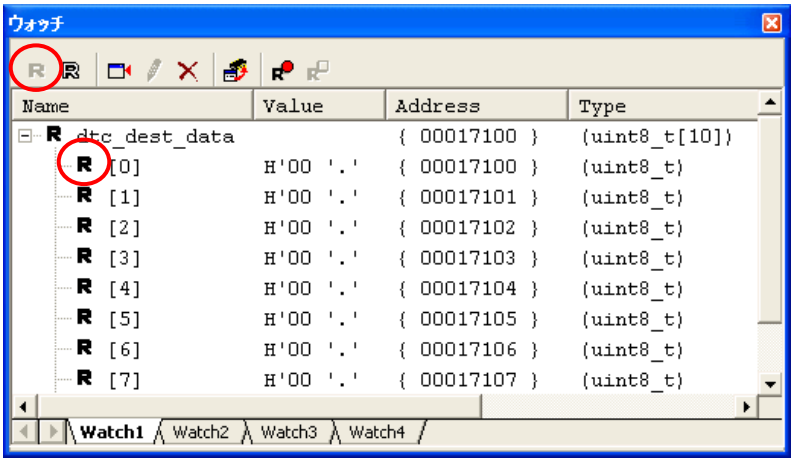
作成したプログラムをビルドし、ダウンロードしてください。

エミュレータの接続、プログラムのビルド方法については「6.1 (12) エミュレータの接続、プログラムのビルド、実行」 を参照してください。

注意: RXファミリC/C++コンパイラパッケージ V.1.01以上を使用している場合、ビルド時にエラーメッセージが出力される場合があります。詳細については 5.(5) を参照してください。

(11) 転送先変数のウォッチウィンドウ登録 **HEW**

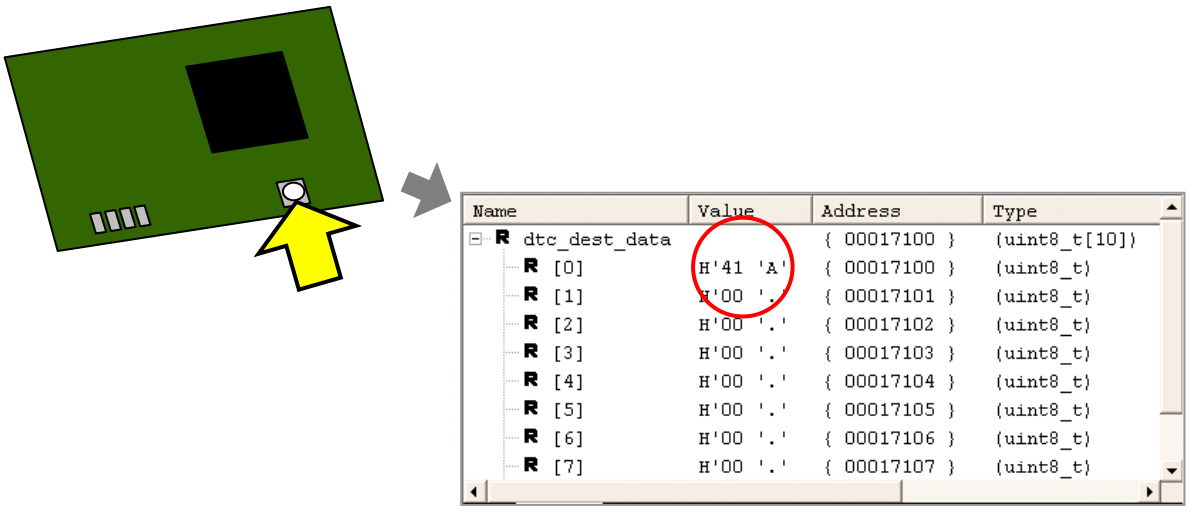
HEW のウォッチウィンドウを開き、転送先変数 "dtc_dest_data" を登録してください。"dtc_dest_data"を展開しリアルタイム更新に設定すると、実行中に値の変化を確認することができます。



(12) プログラムの実行と転送結果の確認 **HEW**

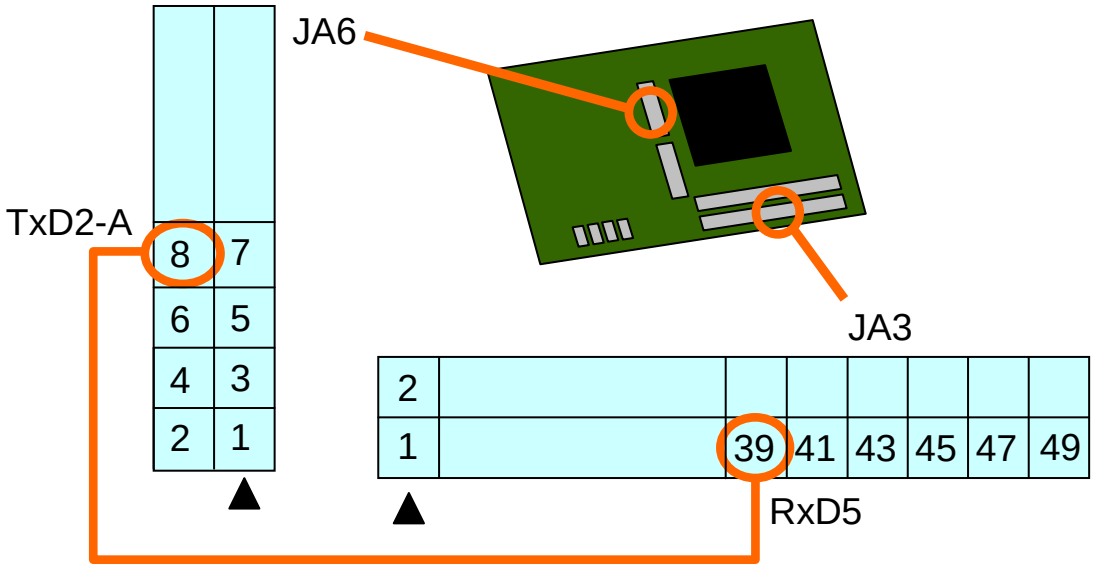
プログラを実行し、実行中 SW1 を押して IRQ8 割り込みを発生させてください。

ボタンを押すたびにデータが転送されます。



6.5 SCIa チャネル 2 とチャネル 5 で調歩同期通信

このチュートリアルでは、シリアルのチャネル 2 からチャネル 5 に調歩同期モードでデータを送信します。RSK+ボード上でチャネル 2 の送信端子(TxD2-A)とチャネル 5 の受信端子(RXD5)を図の様に接続してください。TxD2-A は RSK+ボードの JA6/No.8、RXD5 は JA3/No.39 です。



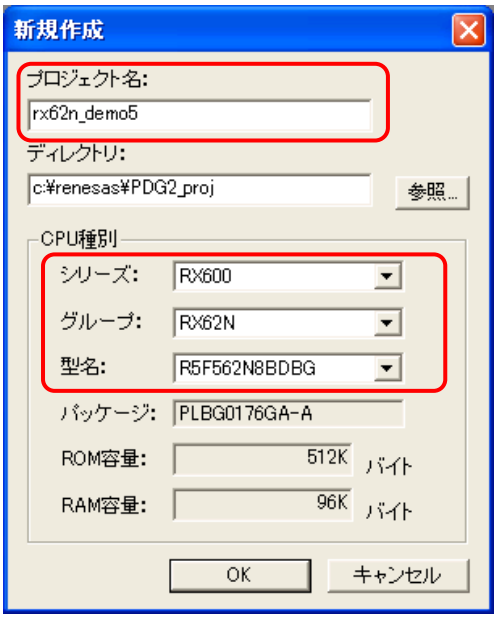
使用する RSK+ボード上に TxD2-A、RxD5 の有効/無効を切り替えるスイッチがある場合は有効にしてください。

(1) PDG プロジェクトの作成 PDG

プロジェクト名に“rx62n_demo5”を指定し、PDG の新規プロジェクトを作成してください。(プロジェクト作成方法の詳細については「6.1 (1)PDG プロジェクトの作成」を参照してください。)

CPU 種別は以下の通り設定してください。但し使用する RSK+ボードに他の型名のチップが搭載されている場合は、ボードに合わせて設定してください。

シリーズ : RX600
グループ : RX62N
型名 : R5F562N8BDBG

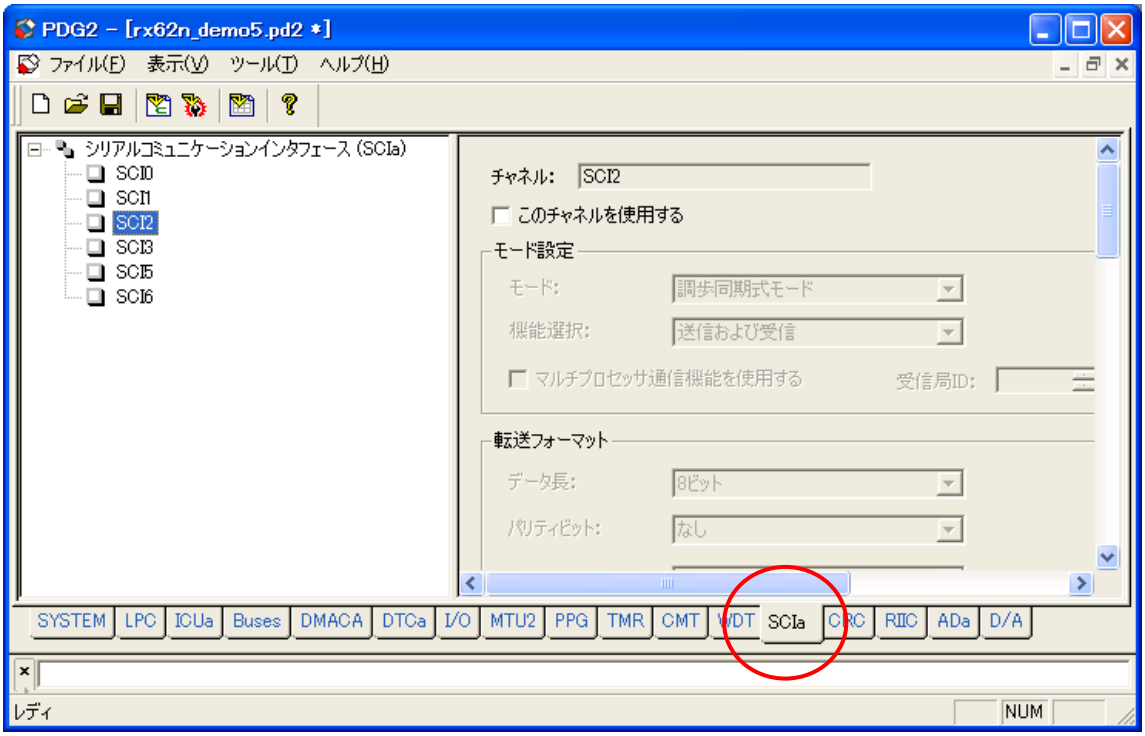


(2) クロックの設定 PDG

1. プロジェクトを作成するとクロック設定ウィンドウが開きます。設定画面上の や などのアイコンについては、「6.1 (1)初期状態」を参照してください。
2. RSK+ボードの外部入力周波数は 12MHz です。“12”と入力してください。

(3) SCIa の設定 PDG

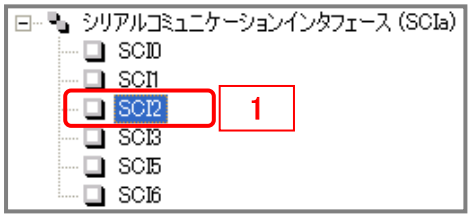
SCIa タブを選択し、SCIa の設定ウィンドウを開いてください。



(4) SCI2(送信側)の設定 PDG

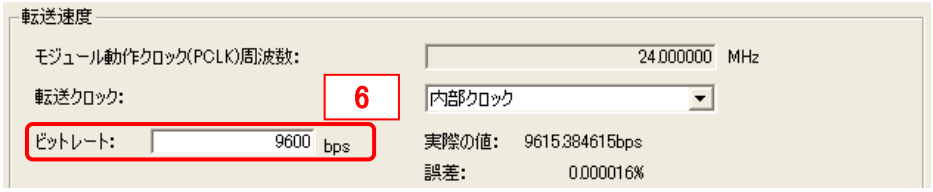
SCI2 を以下の通り設定してください。

1. ツリー表示上で SCI2 を選択してください。

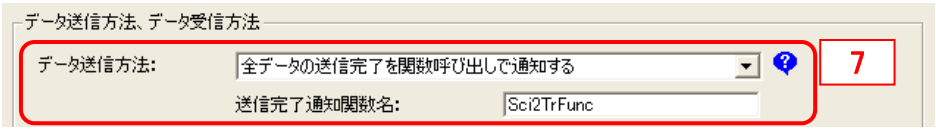


2. [このチャネルを使用する]をチェックしてください。
3. モードに[調歩同期式モード]を選択してください。
4. 機能選択に[送信]を指定してください。
5. 転送フォーマットは初期設定のままとしてください。

6. 転送速度設定のビットレートに 9600bps を設定してください。



7. データ送信方法に[全データの送信完了を関数呼び出しで通知する]を指定し、送信完了通知関数名を初期設定の”Sci2TrFunc”としてください。



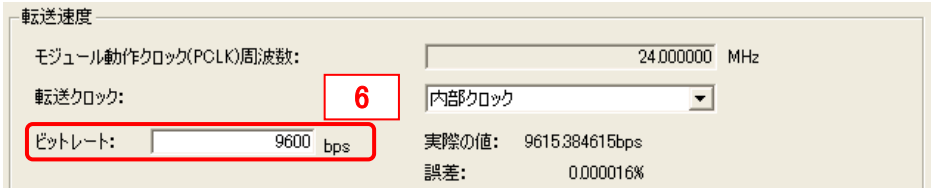
(5) SCI5(受信側)の設定 PDG

SCI5 を以下の通り設定してください。

1. ツリー表示上で SCI5 を選択してください。

2. [このチャネルを使用する]をチェックしてください。
3. モードに[調歩同期式モード]を選択してください。
4. 機能選択に[受信]を指定してください。
5. 転送フォーマットは初期設定のままとしてください。

6. 転送速度設定のビットレートに 9600bps を設定してください。

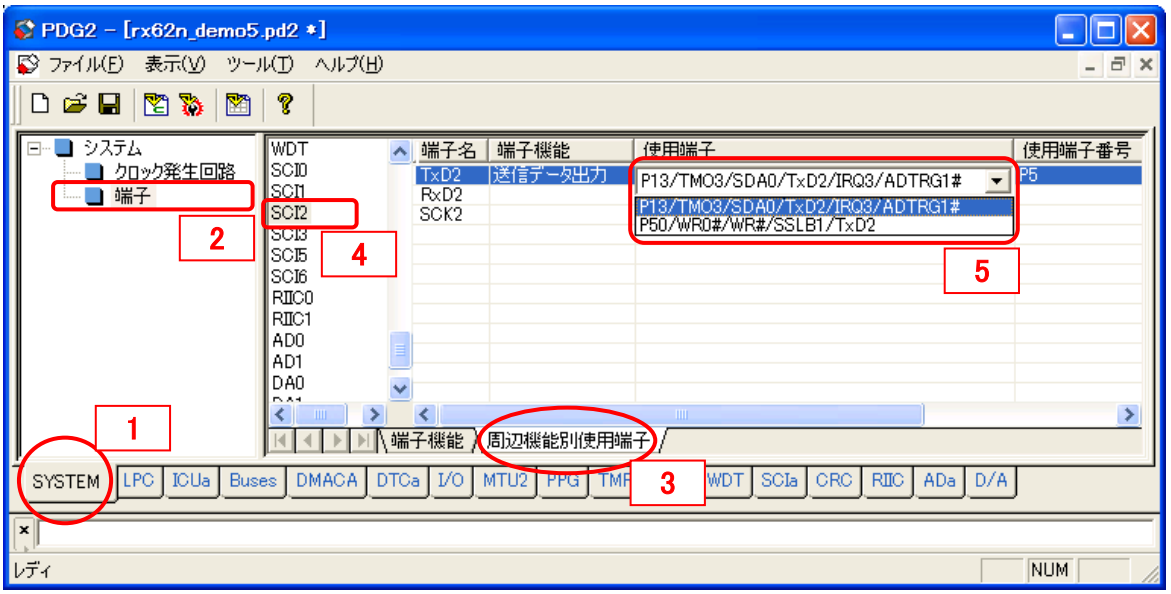


7. データ受信方法に[全データの受信完了を関数呼び出しで通知する]を指定し、受信完了通知関数名を初期設定の”Sci5ReFunc”としてください。

(6) 使用する端子の設定 **PDG**

TxD2 は TxD2-A (P13) と TxD2-B (P50) から選択することができます。以下の方法で使用する端子を選択してください。

1. SYSTEM タブを選択してください。
2. ツリー表示上で[端子]を選択してください。
3. [周辺機能別使用端子]タブを選択してください。
4. 周辺機能の一覧から SCI2 を選択してください。
5. TxD2 の行で[使用端子カラム]カラムにマウスポインタを置くと、端子選択のドロップダウンボタンが表示されます。ドロップダウンリストから[P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#]を選択してください。



(7) ソースファイルの生成 **PDG**

ツールバー上の  をクリックしてソースファイルを生成してください。 ソースファイル生成の詳細については「6.1 (8)ソースファイルの生成」を参照してください。

(8) HEW プロジェクトの準備 **HEW**

HEW を起動し、RX62N 用のワークスペースを作成してください。作成方法については「6.1 (9)HEW プロジェクトの準備」を参照してください。

(9) PDG 生成ファイルの HEW への登録 **PDG**

ツールバー上の  をクリックして PDG が生成したソースファイルを HEW のプロジェクトに登録してください。ソースファイル生成の詳細については「6.1 (10)PDG 生成ファイルの HEW への登録」を参照してください。

(10) プログラムの作成 **HEW**

HEW 上で main 関数の部分を変更し、以下のプログラムを作成してください。

```
//Include "R_PG_<PDGプロジェクト名>.h"
#include "R_PG_rx62n_demo5.h"
//SCI2送信データ
uint8_t tr_data[10] = "ABCDEFGHIJ";
//SCI5受信データ
uint8_t re_data[10] = "----------";
void main(void)
{
    //クロックの設定
    R_PG_Clock_Set();
    //SCI2の設定
    R_PG_SCI_Set_C2();
    //SCI5の設定
    R_PG_SCI_Set_C5();
    //SCI5受信開始 (受信データ数:10)
    R_PG_SCI_StartReceiving_C5( re_data, 10 );
    //SCI2送信開始 (送信データ数:10)
    R_PG_SCI_StartSending_C2( tr_data, 10 );
    while(1);
}
//SCI2送信完了通知関数
void Sci2TrFunc(void)
{
    //SCI2通信終了
    R_PG_SCI_StopCommunication_C2();
}
//SCI5受信完了通知関数
void Sci5ReFunc(void)
{
    //SCI5通信終了
    R_PG_SCI_StopCommunication_C5();
}
```

(11) エミュレータの接続、プログラムのビルド、ダウンロード **HEW**

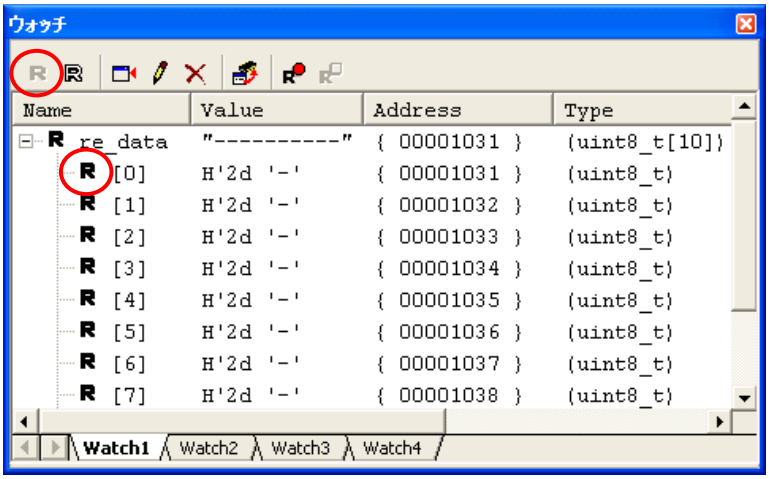
作成したプログラムをビルドし、ダウンロードしてください。

エミュレータの接続、プログラムのビルド方法については「6.1 (12) エミュレータの接続、プログラムのビルド、実行」 を参照してください。

注意: RXファミリC/C++コンパイラパッケージ V.1.01以上を使用している場合、ビルド時にエラーメッセージが出力される場合があります。詳細については 5.(5) を参照してください。

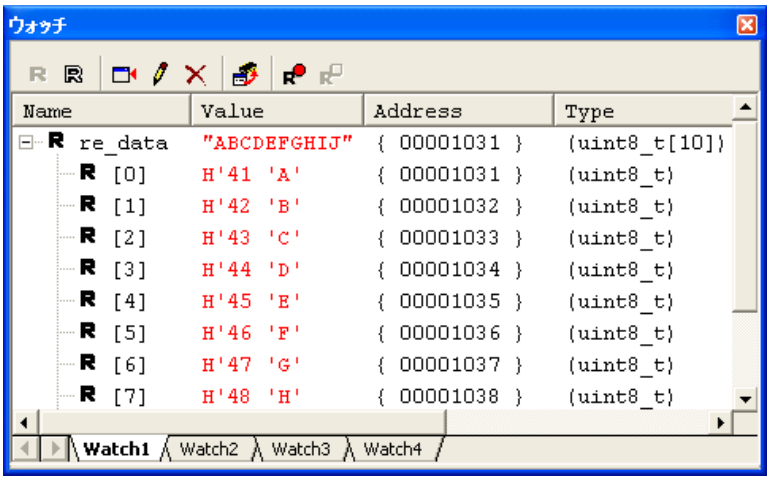
(12) 受信データ格納変数のウォッチウィンドウ登録 **HEW**

HEW のウォッチウィンドウを開き、転送先変数 ″re_data″ を登録してください。″re_data″を展開しリアルタイム更新に設定すると、実行中に値の変化を確認することができます。



(13) プログラムの実行と転送結果の確認 **HEW**

プログラを実行し、変数の値を確認してください。



付録 1. 割当先を変更できる端子機能

表 a-1.1 176-pin LFBGA （上段が初期設定です）

周辺機能	端子機能	割り当て先	Pin No.
ICU（外部割込み）	IRQ0	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
		P10/USB1_DPUPE/MTIC5W/TMRI3/IRQ0	N7
	IRQ1	P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K2
		P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	IRQ2	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J2
		P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
	IRQ3	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	K1
		P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	P5
	IRQ4	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4	J4
		P14/USB0_OVRCURA/USB0_DPUPE/TMRI2/IRQ4	P4
	IRQ5	PE5/D13/RSPCKB/IRQ5	C14
		P15/USB1_OVRCURA/USB1_DPUPE/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	N5
	IRQ6	PE6/D14/MOSIB/IRQ6	C15
		P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/TMO2/PO14/RxD3/IRQ6	P3
	IRQ7	PE7/D15/MISOB/IRQ7	D14
		P17/USB1_VBUS/USB1_OVRCURB/USB1_VBUSEN/MTIOC3A/PO15/TxD3/IRQ7	N4
	IRQ8	P00/TMRI0/TxD6/IRQ8	C1
		P40/IRQ8/AN0	C5
	IRQ9	P01/TMCI0/RxD6/IRQ9	D2
		P41/IRQ9/AN1	D4
	IRQ10	P02/TMCI1/SCK6/IRQ10	B1
		P42/IRQ10/AN2	A3
	IRQ11	P03/IRQ11/DA0	C2
		P43/IRQ11/AN3	D5
	IRQ13	P05/IRQ13/DA1	C3
		P45/IRQ13/AN5	A4
	IRQ15	P07/IRQ15/ADTRG0#	C4
		P47/IRQ15/AN7	B5
アドレスバス	A16	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	M12
	*1	P90/D16/A16	A6
	A17	PC1/A17/ET_RXD2/MTCLKH/SSLA2/SCK5	P14
	*1	P91/D17/A17	B6
	A18	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	N12
	*1	P92/D18/A18	C7
	A19	PC3/A19/ET_TX_ER/MTCLKF/TxD5	N11
	*1	P93/D19/A19	D7
	A20	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	*1	P94/D20/A20	C8
	A21	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*1	P95/D21/A21	D8
	A22	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	*1	P96/D22/A22	B8
	A23	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	*1	P97/D23/A23	B9

バス制御	CS0#	P60/CS0#	B11
		PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	CS1#	P61/CS1#/SDCS#	A13
		P71/CS1#/ET_MDIO	K13
		PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	CS2#	P62/CS2#/RAS#	B12
		P72/CS2#/ET_MDC	K14
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	CS3#	P63/CS3#/CAS#	A14
		P73/CS3#/ET_WOL	N14
		PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	CS4#	P64/CS4#/WE#	B13
		P74/CS4#/ET_ERXD1/RMII_RXD1	N13
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	P1
	CS5#	P65/CS5#/CKE	D15
		P75/CS5#/ET_ERXD0/RMII_RXD0	R15
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	M2
	CS6#	P66/CS6#/DQM0	E14
		P76/CS6#/ET_RX_CLK/REF50CK	P13
		P26/CS6#/USB1_ID/MTIOC2A/TMO1/PO6/MOSIB/TxD1	N1
	CS7#	P67/CS7#/DQM1	E15
		P77/CS7#/ET_RX_ER/RMII_RX_ER	R14
		P27/CS7#/USB1_EXICEN/MTIOC2B/PO7/RSPCKB/SCK1	L2
	WAIT#	P57/WAIT#/WR3#/BC3#/EDREQ1	N6
		P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D	M6
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
		P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	M8
MTU0-5	MTCLKA	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	P1
	*2	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	MTCLKB	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	M2
	*2	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	MTCLKC	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	M3
	*2	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	MTCLKD	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2
	*2	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
MTU6-11	MTCLKE	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	N12
	*3	PB4/A12/MTIOC10A/MTCLKE/PO28	L13
	MTCLKF	PC3/A19/ET_TX_ER/MTCLKF/TxD5	N11
	*3	PB5/A13/MTIOC10C/MTCLKF/PO29	N15
	MTCLKG	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	M12
	*3	PB2/A10/MTIOC9B/MTCLKG/PO26	M15
	MTCLKH	PC1/A17/ET_RXD2/MTCLKH/SSLA2/SCK5	P14
	*3	PB3/A11/MTIOC9D/MTCLKH/PO27	L14
MTU3	MTIOC3B	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	M3
	*4	P80/EDREQ0/ET_TX_EN/RMII_TXDN/MTIOC3B	R13
	MTIOC3C	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	P3
		P56/WR2#/BC2#/EDACK1/MTIOC3C	P7

	MTIOC3D	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2
	*4	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D	M11
MTU4	MTIOC4A	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/PO4/SCK3	P1
	*5	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A	P11
	MTIOC4B	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
	*6	P54/EDACK0/ET_LINKSTA/MTIOC4B	M7
	MTIOC4C	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/RxD3/ADTRG0#	M2
	*5	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C	R11
	MTIOC4D	P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K2
	*6	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D	M6
MTU5	MTIC5U	P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
	*7	PD7/D7/MTIC5U/POE0#	A12
	MTIC5V	P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	*7	PD6/D6/MTIC5V/POE1#	B10
	MTIC5W	P10/USB1_DPUPE/MTIC5W/TMRI3/IRQ0	N7
	*7	PD5/D5/MTIC5W/POE2#	C10
MTU11	MTIC11U	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	*8	PD4/D4/MTIC11U/POE3#	A10
	MTIC11V	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	*8	PD3/D3/MTIC11V/POE4#	A9
	MTIC11W	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*8	PD2/D2/MTIC11W/POE5#	A8
TMR0	TMCI0	P01/TMCI0/RxD6/IRQ9	D2
	*9	P21/USB0_EXICEN/MTIOC1B/TMCI0/PO1/SCL1/RxD0	R1
	TMRI0	P00/TMRI0/TxD6/IRQ8	C1
	*9	P20/USB0_ID/MTIOC1A/TMRI0/PO0/SDA1/TxD0	N3
TMR1	TMCI1	P02/TMCI1/SCK6/IRQ10	B1
		P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
TMR2	TMCI2	P15/USB1_OVRCURA/USB1_DPUPE/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	N5
		P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K3
TMR3	TMCI3	P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	*10	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4	J4
	TMRI3	P10/USB1_DPUPE/MTIC5W/TMRI3/IRQ0	N7
	*10	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
SCI1	TxD1	P26/CS6#/USB1_ID/MTIOC2A/TMO1/PO6/MOSIB/TxD1	N1
	*11	PF0/TxD1/TDO	K3
	RxD1	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
	*11	PF2/RxD1/TDI	L1
	SCK1	P27/CS7#/USB1_EXICEN/MTIOC2B/PO7/RSPCKB/SCK1	L2
	*11	PF1/SCK1/TCK	M1
SCI2	TxD2	P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	P5
	*12	P50/WR0#/WR#/SSLB1/TxD2	P10
	RxD2	P12/MTIC5U/TMCI1/SCL0/RxD2/IRQ2	R3
	*12	P52/RD#/SSLB3/RxD2	N7
	SCK2	P11/USB1_VBUSEN/MTIC5V/TMCI3/SCK2/IRQ1	M5
	*12	P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	M8
SCI3	TxD3	P17/USB1_VBUS/USB1_OVRCURB/USB1_VBUSEN/MTIOC3A/PO15/TxD3/IRQ7	N4
	*13	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2

	RxD3	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/TMO2/PO14/RxD3/IRQ6	P3
	*13	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/RxD3/ADTRG0#	M2
	SCK3	P15/USB1_OVRCURA/USB1_DPUPE/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	N5
	*13	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/PO4/SCK3	P1
SCI6	TxD6	P00/TMRI0/TxD6/IRQ8	C1
	*14	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J2
	RxD6	P01/TMCI0/RxD6/IRQ9	D2
	*14	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	K1
	SCK6	P02/TMCI1/SCK6/IRQ10	B1
	*14	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4	J4
AD0, S12AD0	ADTRG0#	P07/IRQ15/ADTRG0#	C4
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/RxD3/ADTRG0#	M2
EXDMAC0	EDREQ0	P80/EDREQ0/ET_TX_EN/RMII_TXDN/MTIOC3B	R13
		P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	M3
	*15	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D	M6
	EDACK0	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D	M11
		P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	N2
	*15	P54/EDACK0/ET_LINKSTA/MTIOC4B	M7
EXDMAC1	EDREQ1	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A	P11
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/PO4/SCK3	P1
	*16	P57/WAIT#/WR3#/BC3#/EDREQ1	N6
	EDACK1	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C	R11
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/RxD3/ADTRG0#	M2
	*16	P56/WR2#/BC2#/EDACK1/MTIOC3C	P7
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*17	PA5/A5/MTIOC7B/PO21/RSPCKA	J13
	MOSIA	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M10
	*17	PA6/A6/MTIOC8A/PO22/MOSIA	J15
	MISOA	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	R12
	*17	PA7/A7/MTIOC8B/PO23/MISOA	J14
	SSLA0	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	P12
	*17	PA4/A4/MTIOC7A/PO20/SSLA0	H14
	SSLA1	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	M12
	*17	PA0/A0/BC0#/DQM2/MTIOC6A/PO16/SSLA1	F14
	SSLA2	PC1/A17/ET_RXD2/MTCLKH/SSLA2/SCK5	P14
	*17	PA1/A1/DQM3/MTIOC6B/PO17/SSLA2	G15
	SSLA3	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	N12
	*17	PA2/A2/MTIOC6C/PO18/SSLA3	H13
RSPI1	RSPCKB	P27/CS7#/USB1_EXICEN/MTIOC2B/PO7/RSPCKB/SCK1	L2
	*18	PE5/D13/RSPCKB/IRQ5	C14
	MOSIB	P26/CS6#/USB1_ID/MTIOC2A/TMO1/PO6/MOSIB/TxD1	N1
	*18	PE6/D14/MOSIB/IRQ6	C15
	MISOB	P30/USB1_DRPD/MTIOC4B/TMRI3/PO8/MISOB/RxD1/IRQ0	L4
	*18	PE7/D15/MISOB/IRQ7	D14
	SSLB0	P31/USB1_DPRPD/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1	K2

	*18	PE4/D12/SSLB0	D13
	SSLB1	P50/WR0#/WR#/SSLB1/TxD2	P10
	*18	PE0/D8/SSLB1	C12
	SSLB2	P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	M8
	*18	PE1/D9/SSLB2	A15
	SSLB3	P52/RD#/SSLB3/RxD2	N8
	*18	PE2/D10/POE9#/SSLB3	B14

*1 ～ 18 設定は連動して変更されます。

表 a-1.2 145-pin TFLGA （上段が初期設定です）

周辺機能	端子機能	割り当て先	Pin No.
ICU	IRQ2	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J4
(外部		P12/TMCI1/SCL0/RxD2/IRQ2	L4
割込み)	IRQ3	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	J1
		P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	N4
	IRQ4	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4/TRST#	H2
		P14/USB0_OVRCURA/USB0_DPUPE/TMRI2/IRQ4	M5
	IRQ5	PE5/D13/RSPCKB/IRQ5	C11
		P15/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	M4
	IRQ6	PE6/D14/MOSIB/IRQ6	D13
		P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	N2
	IRQ7	PE7/D15/MISOB/IRQ7	D12
		P17/MTIOC3A/PO15/TxD3/IRQ7	L3
	IRQ8	P00/TMRI0/TxD6/IRQ8	E3
		P40/IRQ8/AN0	B4
	IRQ9	P01/TMCI0/RxD6/IRQ9	C1
		P41/IRQ9/AN1	C4
	IRQ10	P02/TMCI1/SCK6/IRQ10	D3
		P42/IRQ10/AN2	A4
	IRQ11	P03/IRQ11/DA0	B1
		P43/IRQ11/AN3	D4
	IRQ13	P05/IRQ13/DA1	C2
		P45/IRQ13/AN5	B5
	IRQ15	P07/IRQ15/ADTRG0#	B2
		P47/IRQ15/AN7	A6
アドレス	A16	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	K12
バス	*1	P90/A16	B6
	A17	PC1/A17/ET_ERXD2/MTCLKH/SSLA2/SCK5	M12
	*1	P91/A17	A7
	A18	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	M11
	*1	P92/A18	C6
	A19	PC3/A19/ET_TX_ER/MTCLKF/TxD5	K9
	*1	P93/A19	D7
バス制御	CS0#	P60/CS0#	A11
		PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	CS1#	P61/CS1#/SDCS#	C9
		P71/CS1#/ET_MDIO	H11
		PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	CS2#	P62/CS2#/RAS#	A12

		P72/CS2#/ET_MDC	J13
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	CS3#	P63/CS3#/CAS#	C10
		P73/CS3#/ET_WOL	K11
		PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	N11
	CS4#	P64/CS4#/WE#	A13
		P74/CS4#/ET_ERXD1/RMII_RXD1	N13
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	CS5#	P65/CS5#/CKE	E10
		P75/CS5#/ET_ERXD0/RMII_RXD0	L11
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	CS6#	P66/CS6#/DQM0	E13
		P76/CS6#/ET_RX_CLK/REF50CK	N12
		P26/CS6#/MTIOC2A/TMO1/PO6/MOSIB/TxD1/TDO	K4
	CS7#	P67/CS7#/DQM1	E11
		P77/CS7#/ET_RX_ER/RMII_RX_ER	L10
		P27/CS7#/MTIOC2B/PO7/RSPCKB/SCK1/TCK	J2
	WAIT#	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D/TRDATA3	M7
		PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
		P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	N8
MTU0-5	MTCLKA	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	*2	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	MTCLKB	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	*2	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	MTCLKC	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	L2
	*2	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	N11
	MTCLKD	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	*2	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
MTU6-11	MTCLKE	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	M11
	*3	PB4/A12/MTIOC10A/MTCLKE/PO28	J11
	MTCLKF	PC3/A19/ET_TX_ER/MTCLKF/TxD5	K9
	*3	PB5/A13/MTIOC10C/MTCLKF/PO29	J12
	MTCLKG	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	K12
	*3	PB2/A10/MTIOC9B/MTCLKG/PO26	J10
	MTCLKH	PC1/A17/ET_ERXD2/MTCLKH/SSLA2/SCK5	M12
	*3	PB3/A11/MTIOC9D/MTCLKH/PO27	K13
MTU3	MTIOC3B	P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	L2
	*4	P80/EDREQ0/ET_TX_EN/RMII_TXD_EN/MTIOC3B/TRDATA0	M10
	MTIOC3C	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	N2
		P56/EDACK1/MTIOC3C	L6
	MTIOC3D	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	*4	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D/TRDATA1	L9
MTU4	MTIOC4A	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	*5	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A/TRSYNC	K8
	MTIOC4B	P30/MTIOC4B/TMRI3/PO8/RxD1/MISOB/IRQ0/TDI	K1
	*6	P54/EDACK0/ET_LINKSTA/MTIOC4B/TRDATA2	N7

	MTIOC4C	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	*5	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C/TRCLK	L8
	MTIOC4D	P31/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1/TMS	K3
	*6	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D/TRDATA3	M7
MTU11	MTIC11U	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	*7	PD4/D4/MTIC11U/POE3#	D8
	MTIC11V	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	*7	PD3/D3/MTIC11V/POE4#	A9
	MTIC11W	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*7	PD2/D2/MTIC11W/POE5#	C7
TMR0	TMCI0	P01/TMCI0/RxD6/IRQ9	C1
	*8	P21/USB0_EXICEN/MTIOC1B/TMCI0/PO1/SCL1/RxD0	N1
	TMRI0	P00/TMRI0/TxD6/IRQ8	E3
	*8	P20/USB0_ID/MTIOC1A/TMRI0/PO0/SDA1/TxD0	M2
TMR1	TMCI1	P02/TMCI1/SCK6/IRQ10	D3
		P12/TMCI1/SCL0/RxD2/IRQ2	L4
TMR2	TMCI2	P15/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	M4
		P31/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1/TMS	K3
SCI2	TxD2	P13/TMO3/SDA0/TxD2/IRQ3/ADTRG1#	N4
	*9	P50/WR0#/WR#/SSLB1/TxD2	M8
	RxD2	P12/TMCI1/SCL0/RxD2/IRQ2	L4
	*9	P52/RD#/SSLB3/RxD2	L7
SCI3	TxD3	P17/MTIOC3A/PO15/TxD3/IRQ7	L3
	*10	P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	RxD3	P16/USB0_VBUS/USB0_OVRCURB/USB0_VBUSEN/MTIOC3C/ TMO2/PO14/RxD3/IRQ6	N2
	*10	P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	SCK3	P15/MTIOC0B/TMCI2/PO13/SCK3/IRQ5	M4
	*10	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
SCI6	TxD6	P00/TMRI0/TxD6/IRQ8	E3
	*11	P32/MTIOC0C/PO10/RTCOUT/CTX0/TxD6/IRQ2	J4
	RxD6	P01/TMCI0/RxD6/IRQ9	C1
	*11	P33/MTIOC0D/PO11/CRX0/RxD6/IRQ3	J1
	SCK6	P02/TMCI1/SCK6/IRQ10	D3
	*11	P34/MTIOC0A/TMCI3/PO12/SCK6/IRQ4/TRST#	H2
AD0, S12AD0	ADTRG0#	P07/IRQ15/ADTRG0#	B2
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
EXDMAC0	EDREQ0	P80/EDREQ0/ET_TX_EN/RMII_TXD_EN/MTIOC3B/TRDATA0	M10
		P22/EDREQ0/USB0_DRPD/MTIOC3B/MTCLKC/TMO0/PO2/SCK0	L2
	*12	P55/WAIT#/EDREQ0/ET_EXOUT/MTIOC4D/TRDATA3	M7
	EDACK0	P81/EDACK0/ET_ETXD0/RMII_TXD0/MTIOC3D/TRDATA1	L9
		P23/EDACK0/USB0_DPUPE/MTIOC3D/MTCLKD/PO3/TxD3	M1
	*12	P54/EDACK0/ET_LINKSTA/MTIOC4B/TRDATA2	N7
EXDMAC1	EDREQ1	P82/EDREQ1/ET_ETXD1/RMII_TXD1/MTIOC4A/TRSYNC	K8
		P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2
	*13	P24/CS4#/EDREQ1/USB0_VBUSEN/MTIOC4A/MTCLKA/TMRI1/ PO4/SCK3	K2

	EDACK1	P83/EDACK1/ET_CRS/RMII_CRS_DV/MTIOC4C/TRCLK	L8
		P25/CS5#/EDACK1/USB0_DPRPD/MTIOC4C/MTCLKB/PO5/ RxD3/ADTRG0#	L1
	*13	P56/EDACK1/MTIOC3C	L6
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/ET_ETXD2/MTIC11W/MTCLKD/RSPCKA	N10
	*14	PA5/A5/MTIOC7B/PO21/RSPCKA	G11
	MOSIA	PC6/A22/CS1#/ET_ETXD3/MTIC11V/MTCLKA/MOSIA	M9
	*14	PA6/A6/MTIOC8A/PO22/MOSIA	G12
	MISOA	PC7/A23/CS0#/ET_COL/MTIC11U/MTCLKB/MISOA	K7
	*14	PA7/A7/MTIOC8B/PO23/MISOA	H13
	SSLA0	PC4/A20/CS3#/ET_TX_CLK/MTCLKC/SSLA0	N11
	*14	PA4/A4/MTIOC7A/PO20/SSLA0	G13
	SSLA1	PC0/A16/ET_ERXD3/MTCLKG/SSLA1	K12
	*14	PA0/A0/BC0#/MTIOC6A/PO16/SSLA1	E12
	SSLA2	PC1/A17/ET_ERXD2/MTCLKH/SSLA2/SCK5	M12
	*14	PA1/A1/MTIOC6B/PO17/SSLA2	F10
	SSLA3	PC2/A18/ET_RX_DV/MTCLKE/SSLA3/RxD5	M11
	*14	PA2/A2/MTIOC6C/PO18/SSLA3	F13
RSPI1	RSPCKB	P27/CS7#/MTIOC2B/PO7/RSPCKB/SCK1/TCK	J2
	*15	PE5/D13/RSPCKB/IRQ5	C11
	MOSIB	P26/CS6#/MTIOC2A/TMO1/PO6/MOSIB/TxD1/TDO	K4
	*15	PE6/D14/MOSIB/IRQ6	D13
	MISOB	P30/MTIOC4B/TMRI3/PO8/RxD1/MISOB/IRQ0/TDI	K1
	*15	PE7/D15/MISOB/IRQ7	D12
	SSLB0	P31/MTIOC4D/TMCI2/PO9/SSLB0/IRQ1/TMS	K3
	*15	PE4/D12/SSLB0	B13
	SSLB1	P50/WR0#/WR#/SSLB1/TxD2	M8
	*15	PE0/D8/SSLB1	B10
	SSLB2	P51/WR1#/BC1#/WAIT#/SSLB2/SCK2	N8
	*15	PE1/D9/SSLB2	B12
	SSLB3	P52/RD#/SSLB3/RxD2	L7
	*15	PE2/D10/POE9#/SSLB3	B11

*1 ～ 15 設定は連動して変更されます。

表 a-1.3 144-pin LQFP （上段が初期設定です）

周辺機能	端子機能	割り当て先	Pin No.
ICU (外部 割込み)	IRQ2	P32/PO10/MTIOC0C/TxD6/CTX0/IRQ2/RTCOUT	27
		P12/TMCI1/RxD2/SCL0/IRQ2	45
	IRQ3	P33/PO11/MTIOC0D/RxD6/CRX0/IRQ3	26
		P13/ADTRG1#/TMO3/TxD2/SDA0/IRQ3	44
	IRQ4	P34/PO12/MTIOC0A/TMCI3/SCK6/IRQ4/TRST#	25
		P14/TMRI2/IRQ4/USB0_OVRCURA/USB0_DPUPE	43
	IRQ5	PE5/D13/RSPCKB/IRQ5	106
		P15/PO13/MTIOC0B/TMCI2/SCK3/IRQ5	42
	IRQ6	PE6/D14/MOSIB/IRQ6	102
		P16/PO14/MTIOC3C/TMO2/RxD3/IRQ6/USB0_VBUS/ USB0_OVRCURB/USB0_VBUSEN	40
	IRQ7	PE7/D15/MISOB/IRQ7	101
		P17/PO15/MTIOC3A/TxD3/IRQ7	38
	IRQ8	P00/TMRI0/TxD6/IRQ8	8

		P40/AN0/IRQ8	141
	IRQ9	P01/TMCI0/RxD6/IRQ9	7
		P41/AN1/IRQ9	139
	IRQ10	P02/TMCI1/SCK6/IRQ10	6
		P42/AN2/IRQ10	138
	IRQ11	P03/DA0/IRQ11	4
		P43/AN3/IRQ11	137
	IRQ13	P05/DA1/IRQ13	2
		P45/AN5/IRQ13	135
	IRQ15	P07/ADTRG0#/IRQ15	144
		P47/AN7/IRQ15	133
アドレスバス	A16	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	75
	*1	P90/A16	131
	A17	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	73
	*1	P91/A17	129
	A18	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	70
	*1	P92/A18	128
	A19	PC3/A19/MTCLKF/TxD5/ET_TX_ER	67
	*1	P92/A19	127
バス制御	CS0#	P60/CS0#	117
		PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	CS1#	P61/CS1#/SDCS#	115
		P71/CS1#/ET_MDIO	86
		PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	CS2#	P62/CS2#/RAS#	114
		P72/CS2#/ET_MDC	85
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	62
	CS3#	P63/CS3#/CAS#	113
		P73/CS3#/ET_WOL	77
		PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	66
	CS4#	P64/CS4#/WE#	112
		P74/CS4#/ET_ERXD1/RMII_RXD1	72
		P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	CS5#	P65/CS5#/CKE	100
		P75/CS5#/ET_ERXD0/RMII_RXD0	71
		P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	CS6#	P66/CS6#/DQM0	99
		P76/CS6#/ET_RX_CLK/REF50CK	69
		P26/CS6#/PO6/MTIOC2A/TMO1/TxD1/MOSIB/TDO	31
	CS7#	P67/CS7#/DQM1	98
		P77/CS7#/ET_RX_ER/RMII_RX_ER	68
		P27/CS7#/PO7/MTIOC2B/SCK1/RSPCKB/TCK	30
	WAIT#	P55/WAIT#/EDREQ0/MTIOC4D/ET_EXOUT/TRDATA3	51
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	62
		P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	55
MTU0-5	MTCLKA	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	*2	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	MTCLKB	P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32

	*2	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	MTCLKC	P22/EDREQ0/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/ USB0_DRPD	35
	*2	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	66
	MTCLKD	P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	34
	*2	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ ET_ETXD2	62
MTU6-11	MTCLKE	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	70
	*3	PB4/A12/PO28/MTIOC10A/MTCLKE	81
	MTCLKF	PC3/A19/MTCLKF/TxD5/ET_TX_ER	67
	*3	PB5/A13/PO29/MTIOC10C/MTCLKF	80
	MTCLKG	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	75
	*3	PB2/A10/PO26/MTIOC9B/MTCLKG	83
	MTCLKH	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	73
	*3	PB3/A11/PO27/MTIOC9D/MTCLKH	82
MTU3	MTIOC3B	P22/EDREQ0/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/ USB0_DRPD	35
	*4	P80/EDREQ0/MTIOC3B/ET_TX_EN/RMII_TXD_EN/ TRDATA0	65
	MTIOC3C	P16/PO14/MTIOC3C/TMO2/RxD3/IRQ6/USB0_VBUS/ USB0_OVRCURB/USB0_VBUSEN	40
		P56/EDACK1/MTIOC3C	50
	MTIOC3D	P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	34
	*4	P81/EDACK0/MTIOC3D/ET_ETXD0/RMII_TXD0/TRDATA1	64
MTU4	MTIOC4A	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	*5	P82/EDREQ1/MTIOC4A/ET_ETXD1/RMII_TXD1/TRSYNC	63
	MTIOC4B	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	29
	*6	P54/EDACK0/MTIOC4B/ET_LINKSTA/TRDATA2	52
	MTIOC4C	P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	*5	P83/EDACK1/MTIOC4C/ET_CRS/RMII_CRS_DV/TRCLK	58
	MTIOC4D	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	28
	*6	P55/WAIT#/EDREQ0/MTIOC4D/ET_EXOUT/TRDATA3	51
MTU11	MTIC11U	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	*7	PD4/D4/MTIC11U/POE3#	122
	MTIC11V	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	*7	PD3/D3/MTIC11V/POE4#	123
	MTIC11W	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ ET_ETXD2	62
	*7	PD2/D2/MTIC11W/POE5#	124
TMR0	TMCI0	P01/TMCI0/RxD6/IRQ9	7
	*8	P21/PO1/MTIOC1B/TMCI0/RxD0/SCL1/USB0_EXICEN	36
	TMRI0	P00/TMRI0/TxD6/IRQ8	8
	*8	P20/PO0/MTIOC1A/TMRI0/TxD0/SDA1/USB0_ID	37
TMR1	TMCI1	P02/TMCI1/SCK6/IRQ10	6
		P12/TMCI1/RxD2/SCL0/IRQ2	45
TMR2	TMCI2	P15/PO13/MTIOC0B/TMCI2/SCK3/IRQ5	42
		P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	28
SCI2	TxD2	P13/ADTRG1#/TMO3/TxD2/SDA0/IRQ3	44
	*9	P50/WR0#/WR#/TxD2/SSLB1	56
	RxD2	P12/TMCI1/RxD2/SCL0/IRQ2	45

	*9	P52/RD#/RxD2/SSLB3	54
SCI3	TxD3	P17/PO15/MTIOC3A/TxD3/IRQ7	38
	*10	P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	34
	RxD3	P16/PO14/MTIOC3C/TMO2/RxD3/IRQ6/USB0_VBUS /USB0_OVRCURB/USB0_VBUSEN	40
	*10	P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	SCK3	P15/PO13/MTIOC0B/TMCI2/SCK3/IRQ5	42
	*10	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
SCI6	TxD6	P00/TMRI0/TxD6/IRQ8	8
	*11	P32/PO10/MTIOC0C/TxD6/CTX0/IRQ2/RTCOUT	27
	RxD6	P01/TMCI0/RxD6/IRQ9	7
	*11	P33/PO11/MTIOC0D/RxD6/CRX0/IRQ3	26
	SCK6	P02/TMCI1/SCK6/IRQ10	6
	*11	P34/PO12/MTIOC0A/TMCI3/SCK6/IRQ4/TRST#	25
AD0, S12AD0	ADTRG0#	P07/ADTRG0#/IRQ15	144
		P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
EXDMAC0	EDREQ0	P80/EDREQ0/MTIOC3B/ET_TX_EN/RMII_TXD_EN/TRDATA0	65
		P22/EDREQ0/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/USB0_DRPD	35
	*12	P55/WAIT#/EDREQ0/MTIOC4D/ET_EXOUT/TRDATA3	51
	EDACK0	P81/EDACK0/MTIOC3D/ET_ETXD0/RMII_TXD0/TRDATA1	64
		P23/EDACK0/PO3/MTIOC3D/MTCLKD/TxD3/ USB0_DPUPE	34
	*12	P54/EDACK0/MTIOC4B/ET_LINKSTA/TRDATA2	52
EXDMAC1	EDREQ1	P82/EDREQ1/MTIOC4A/ET_ETXD1/RMII_TXD1/TRSYNC	63
		P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	*13	P24/CS4#/EDREQ1/PO4/MTIOC4A/MTCLKA/TMRI1/ SCK3/USB0_VBUSEN	33
	EDACK1	P83/EDACK1/MTIOC4C/ET_CRS/RMII_CRS_DV/TRCLK	58
		P25/CS5#/EDACK1/ADTRG0#/PO5/MTIOC4C/MTCLKB/ RxD3/USB0_DPRPD	32
	*13	P56/EDACK1/MTIOC3C	50
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	62
	*14	PA5/A5/PO21/MTIOC7B/RSPCKA	90
	MOSIA	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	61
	*14	PA6/A6/PO22/MTIOC8A/MOSIA	89
	MISOA	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	60
	*14	PA7/A7/PO23/MTIOC8B/MISOA	88
	SSLA0	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	66
	*14	PA4/A4/PO20/MTIOC7A/SSLA0	92
	SSLA1	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	75
	*14	PA0/A0/BC0#/PO16/MTIOC6A/SSLA1	97
	SSLA2	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	73
	*14	PA1/A1/PO17/MTIOC6B/SSLA2	96
	SSLA3	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	70
	*14	PA2/A2/PO18/MTIOC6C/SSLA3	95
RSPI1	RSPCKB	P27/CS7#/PO7/MTIOC2B/SCK1/RSPCKB/TCK	30
	*15	PE5/D13/RSPCKB/IRQ5	106
	MOSIB	P26/CS6#/PO6/MTIOC2A/TMO1/TxD1/MOSIB/TDO	31

	*15	PE6/D14/MOSIB/IRQ6	102
	MISOB	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	29
	*15	PE7/D15/MISOB/IRQ7	101
	SSLB0	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	28
	*15	PE4/D12/SSLB0	107
	SSLB1	P50/WR0#/WR#/TxD2/SSLB1	56
	*15	PE0/D8/SSLB1	111
	SSLB2	P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	55
	*15	PE1/D9/SSLB2	110
	SSLB3	P52/RD#/RxD2/SSLB3	54
	*15	PE2/D10/SSLB3/POE9#	109

*1 ～ 15 設定は連動して変更されます。

表 a-1.4 100-pin LQFP （上段が初期設定です）

周辺機能	端子機能	割り当て先	Pin No.
ICU	IRQ2	P32/PO10/MTIOC0C/TxD6/CTX0/IRQ2/RTCOUT	18
(外部		P12/TMCI1/RxD2/SCL0/IRQ2	34
割込み)	IRQ3	P33/PO11/MTIOC0D/RxD6/CRX0/IRQ3	17
		P13/ADTRG1#/PO13/MTIOC0B/TMO3/TxD2/SDA0/IRQ3	33
	IRQ4	P34/PO12/MTIOC0A/TMCI3/SCK6/IRQ4/TRST#	16
		P14/PO15/MTIOC3A/TMRI2/IRQ4/USB0_OVRCURA/ USB0_DPUPE	32
	IRQ6	PE6/D14/MOSIB/IRQ6	72
		P16/PO14/MTIOC3C/TMO2/IRQ6/USB0_VBUS/ USB0_OVRCURB/USB0_VBUSEN	30
	IRQ13	P05/DA1/IRQ13	100
		P45/AN5/IRQ13	89
	IRQ15	P07/ADTRG0#/IRQ15	98
		P47/AN7/IRQ15	87
バス制御	WAIT#	P55/WAIT#/MTIOC4D	39
		PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	47
		P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	43
MTU0-5	MTCLKA	P24/CS4#/PO4/MTIOC4A/MTCLKA/TMRI1/SCK3/ USB0_VBUSEN	24
	*1	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	46
	MTCLKB	P25/CS5#/ADTRG0#/PO5/MTIOC4C/MTCLKB/RxD3/ USB0_DPRPD	23
	*1	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	45
	MTCLKC	P22/PO2/MTIOC3B/MTCLKC/TMO0/SCK0/USB0_DRPD	26
	*1	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	48
	MTCLKD	P23/PO3/MTIOC3D/MTCLKD/TxD3/USB0_DPUPE	25
	*1	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	47
MTU6-11	MTCLKE	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	50
	*2	PB4/A12/PO28/MTIOC10A/MTCLKE/ET_TX_EN/RMII_TXD_EN	56
	MTCLKF	PC3/A19/MTCLKF/TxD5/ET_TX_ER	49
	*2	PB5/A13/PO29/MTIOC10C/MTCLKF/ET_ETXD0/RMII_TXD0	55
	MTCLKG	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	52
	*2	PB2/A10/PO26/MTIOC9B/MTCLKG/ET_RX_CLK/REF50CK	58
	MTCLKH	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	51
	*2	PB3/A11/PO27/MTIOC9D/MTCLKH/ET_RX_ER/RMII_RX_ER	57

MTU4	MTIOC4B	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	20
	*3	P54/MTIOC4B	40
	MTIOC4D	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	19
	*3	P55/WAIT#/MTIOC4D	39
MTU11	MTIC11U	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	45
	*4	PD4/D4/MTIC11U/POE3#	82
	MTIC11V	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	46
	*4	PD3/D3/MTIC11V/POE4#	83
	MTIC11W	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ ET_ETXD2	47
	*4	PD2/D2/MTIC11W/POE5#	84
SCI2	TxD2	P13/ADTRG1#/PO13/MTIOC0B/TMO3/TxD2/SDA0/IRQ3	33
	*5	P50/WR0#/WR#/TxD2/SSLB1	44
	RxD2	P12/TMCI1/RxD2/SCL0/IRQ2	34
	*5	P52/RD#/RxD2/SSLB3	42
AD0, S12AD0	ADTRG0#	P07/ADTRG0#/IRQ15	98
		P25/CS5#/ADTRG0#/PO5/MTIOC4C/MTCLKB/RxD3/ USB0_DPRPD	23
RSPI0	RSPCKA	PC5/A21/CS2#/WAIT#/MTIC11W/MTCLKD/RSPCKA/ET_ETXD2	47
	*6	PA5/A5/PO21/MTIOC7B/RSPCKA/ET_LINKSTA	65
	MOSIA	PC6/A22/CS1#/MTIC11V/MTCLKA/MOSIA/ET_ETXD3	46
	*6	PA6/A6/PO22/MTIOC8A/MOSIA/ET_EXOUT	64
	MISOA	PC7/A23/CS0#/MTIC11U/MTCLKB/MISOA/ET_COL	45
	*6	PA7/A7/PO23/MTIOC8B/MISOA/ET_WOL	63
	SSLA0	PC4/A20/CS3#/MTCLKC/SSLA0/ET_TX_CLK	48
	*6	PA4/A4/PO20/MTIOC7A/SSLA0/ET_MDC	66
	SSLA1	PC0/A16/MTCLKG/SSLA1/ET_ERXD3	52
	*6	PA0/A0/BC0#/PO16/MTIOC6A/SSLA1	70
	SSLA2	PC1/A17/MTCLKH/SCK5/SSLA2/ET_ERXD2	51
	*6	PA1/A1/PO17/MTIOC6B/SSLA2	69
	SSLA3	PC2/A18/MTCLKE/RxD5/SSLA3/ET_RX_DV	50
	*6	PA2/A2/PO18/MTIOC6C/SSLA3	68
RSPI1	RSPCKB	P27/CS7#/PO7/MTIOC2B/SCK1/RSPCKB/TCK	21
	*7	PE5/D13/RSPCKB/IRQ5	73
	MOSIB	P26/CS6#/PO6/MTIOC2A/TMO1/TxD1/MOSIB/TDO	22
	*7	PE6/D14/MOSIB/IRQ6	72
	MISOB	P30/PO8/MTIOC4B/TMRI3/RxD1/MISOB/IRQ0/TDI	20
	*7	PE7/D15/MISOB/IRQ7	71
	SSLB0	P31/PO9/MTIOC4D/TMCI2/SSLB0/IRQ1/TMS	19
	*7	PE4/D12/SSLB0	74
	SSLB1	P50/WR0#/WR#/TxD2/SSLB1	44
	*7	PE0/D8/SSLB1	78
	SSLB2	P51/WR1#/BC1#/WAIT#/SCK2/SSLB2	43
	*7	PE1/D9/SSLB2	77
	SSLB3	P52/RD#/RxD2/SSLB3	42
	*7	PE2/D10/SSLB3/POE9#	76

*1 ～ 7 設定は連動して変更されます。

RX62Nグループ
Peripheral Driver Generator
リファレンスマニュアル

発行年月日　2014年5月16日　Rev.1.04

発行　ルネサス エレクトロニクス株式会社
〒211-8668 神奈川県川崎市中原区下沼部1753

編集　株式会社ルネサス ソリューションズ
ツールビジネス本部 ツール開発第四部

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 〒100-0004 千代田区大手町2-6-2（日本ビル）

■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：http://japan.renesas.com/contact/

RX62Nグループ

Peripheral Driver Generator

リファレンスマニュアル

ルネサス エレクトロニクス株式会社

R20UT0565JJ0104