ユーザーズ・マニュアル

Applilet®3

デバイス・ドライバ・コンフィギュレータ

API 編

V850 マイクロコントローラ

資料番号　ZUD-CD-10-0165（第 1 版）
発行年月　July 2010

［メ　　モ］

- 本資料に記載されている内容は 2010 年 6 月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品の使用に関連し発生した第三者の特許権，著作権その他の知的財産権の侵害等に関し，一切その責を負いません。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。
- 本資料に記載された回路，ソフトウエ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。お客様の機器の設計において，回路，ソフトウエ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，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。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。当社製品をお客様の機器にご使用の際には，当社製品の不具合の結果として，生命，身体および財産に対する損害や社会的損害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を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「特定水準」に分類しております。また，各品質水準は，以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認ください。

  「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

  「特別水準」：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，生命維持を目的として設計されていない医療機器

  「特定水準」：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

  当社製品のデータ・シート，データ・ブック等の資料で特に品質水準の表示がない場合は，標準水準製品であることを表します。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

注 1. 本事項において使用されている「当社」とは，NEC エレクトロニクス株式会社および NEC 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う。

注 2. 本事項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，製造製品をいう。

（M8E0909J）

はじめに

対 象 者　このマニュアルは，78K0 マイクロコントローラ，78K0R マイクロコントローラ，および V850 マイクロコントローラ用デバイス・ドライバ・コンフィギュレータ Applilet3 の機能を理解し，それを用いたアプリケーション・システムを設計するユーザを対象とします。

目　　的　このマニュアルは，Appliet3 の持つソフトウエア機能をユーザに理解していただき，これを使用するシステムのハードウエア，ソフトウエア開発の参照用資料として役立つことを目的としています。

構　　成　このマニュアルは，大きく分けて次の内容で構成しています。

- 第１章　概説
- 第２章　出力ファイル
- 第３章　API 関数

読 み 方　このマニュアルを読むにあたっては，電気，論理回路，マイクロコンピュータに関する一般的知識が必要となります。

・Applilet3 の機能の詳細を理解しようとするとき
　→ 目次に従ってお読みください。

凡　　例

[　　]	：メニュー名，ボタン名，タブ名
「　　」	：ダイアログ名，ウインドウ名
注	：本文中につけた注の説明
注意	：気をつけて読んでいただきたい内容
備考	：本文中の補足説明
数の表記	：2 進数　…　××××または××××b
	10 進数　…　××××
	16 進数　…　××××H または 0x××××

用語説明　このマニュアルで使用する用語を次の表に示します。

用　語	意　味
NEC 環境	ルネサス エレクトロニクス社製の言語ツールおよび統合開発環境プラットフォームを使用してプログラム開発を行う環境
IAR 環境	IAR システムズ社製の言語ツールおよび統合開発環境プラットフォームを使用してプログラム開発を行う環境
GHS 環境	Green Hills Software 社製の言語ツールおよび統合開発環境プラットフォームを使用してプログラム開発を行う環境
78K0 系 Applilet3	78K0S および 78K0 マイクロコントローラ用の Applilet3
78K0R 系 Applilet3	78K0R マイクロコントローラ用の Applilet3
V850 系 Applilet3	V850 マイクロコントローラ用の Applilet3

備考　Applilet3 は，製品ごとに GUI デザインが異なります。

関連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

資 料 名		資料番号
CC78K0 Ver.3.70 ユーザーズ・マニュアル	操作編	U17201J
	言語編	U17200J
CC78K0R Ver.2.00 ユーザーズ・マニュアル	操作編	U18548J
	言語編	U18549J
CA850 Ver.3.20 ユーザーズ・マニュアル	操作編	U18512J
	言語編	U18513J
RA78K0 Ver.3.70 ユーザーズ・マニュアル	操作編	U17199J
	言語編	U17198J
RA78K0R Ver.1.20 ユーザーズ・マニュアル	操作編	U18546J
	言語編	U18547J
PM plus Ver.5.20 ユーザーズ・マニュアル		U16934J
PM+ Ver.6.30 ユーザーズ・マニュアル		U18416J
QB-MINI2 注 ユーザーズ・マニュアル		U18371J
QB-MINI2 注 セットアップ・マニュアル	パートナー・ツール編	U19158J

注　QB-MINI2：プログラミング機能付きオンチップ・デバッグ・エミュレータ MINICUBE®2

注意　上記関連資料は，予告なしに内容を変更することがあります。設計などには，必ず最新の資料を使用してください。

目　　次

第 1 章 概 説

本章では，Applilet3 の概要について説明します。

1.1 概 要

Applilet3 は，GUI ベースで各種情報を設定することにより，マイクロコントローラが提供している周辺機能（クロック発生回路の機能，ポートの機能など）を制御するうえで必要なソース・コード（デバイス・ドライバ・プログラム：C ソース・ファイル，ヘッダ・ファイル）を出力することができます。

1.2 特 長

以下に，Applilet3 の特長を示します。

- コード生成機能

  Applilet3 では，GUI ベースで設定した情報に応じたデバイス・ドライバ・プログラムを出力するだけでなく，main 関数を含んだサンプル・プログラム，リンク・ディレクティブ・ファイルなどといったビルド環境一式を出力することもできます。

- プロジェクト／ワークスペース・ファイル生成機能

  Applilet3 では，アプリケーション・システムの統合開発環境（PM+，MULTI，または IAR Embedded Workbench）で利用可能なプロジェクト／ワークスペース・ファイルを出力することができます。

- レポート機能

  Applilet3 を用いて設定した情報を各種形式のファイルで出力し，設計資料として利用することができます。

- リネーム機能

  Applilet3 が出力するファイル名，およびソース・コードに含まれている API 関数の関数名については，デフォルトの名前が付与されますが，ユーザ独自の名前に変更することができます。

第 2 章　出力ファイル

本章では，Applilet3 が出力するファイルについて説明します。

2.1　概　　要

以下に，Applilet3 が出力するファイルの一覧を示します。

表 2—1　出力ファイル

出力単位	ファイル名	出力内容
各周辺機能	*周辺機能名* .c	初期化関数，API 関数
	周辺機能名 _user.c	割り込み関数，コールバック関数
	周辺機能名 .h	レジスタへの代入値マクロを定義
プロジェクト	option.asm	オプション・バイト，MINICUBE2 用 ROM 確保
	option.inc	オプション・バイトへの設定値マクロを定義
	systeminit.c	各周辺機能の初期化関数コール CG_ReadResetSource のコール
	main.c	main 関数
	macrodriver.h	全ソース・ファイルで共通使用するマクロを定義
	user_define.h	空ファイル（ユーザ定義用）
	lk.dir	リンク・ディレクティブ
	プロジェクト名 .prw	PM+ 用ワーク・スペース
	プロジェクト名 .prj	PM+ 用プロジェクト
	プロジェクト名 .gpj	MULTI 用プロジェクト
	プロジェクト名 .eww	IAR Embedded Workbench 用ワーク・スペース
	プロジェクト名 .ewp	IAR Embedded Workbench 用プロジェクト

2.2　出力ファイル

以下に，Applilet3 が出力するファイル（各周辺機能）を示します。

表 2—2　出力ファイル（各周辺機能）

周辺機能	ファイル名	含まれる API 関数名
システム	CG_system.c	CLOCK_Init CG_ChangeClockMode CG_ChangeFrequency CG_SelectPowerSaveMode

周辺機能	ファイル名	含まれる API 関数名
システム	CG_system.c	CG_SelectStabTime CG_SelectPllMode CG_SelectSSCGMode WDT2_Restart CRC_Start CRC_SetData CRC_GetResult
	CG_system_user.c	MD_INTWDT2 CLOCK_UserInit CG_ReadResetSource
	CG_system.h	—
外部バス	CG_bus.c	BUS_Init
	CG_bus_user.c	BUS_UserInit
	CG_bus.h	—
ポート	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput
	CG_port_user.c	PORT_UserInit
	CG_port.h	—
割り込み	CG_int.c	INTP_Init KEY_Init INT_MaskableInterruptEnable INTPn_Disable INTPn_Enable KEY_Disable KEY_Enable
	CG_int_user.c	MD_INTNMI MD_INTP*n* MD_INTKR INTP_UserInit KEY_UserInit
	CG_int.h	—
シリアル	CG_serial.c	UARTAn_Init UARTAn_Start UARTAn_Stop UARTAn_SendData UARTAn_ReceiveData UARTBn_Init UARTBn_Start UARTBn_Stop UARTBn_SendData UARTBn_ReceiveData

周辺機能	ファイル名	含まれる API 関数名
シリアル	CG_serial.c	UARTCn_Init UARTCn_Start UARTCn_Stop UARTCn_SendData UARTCn_ReceiveData CSIBn_Init CSIBn_Start CSIBn_Stop CSIBn_SendData CSIBn_ReceiveData CSIBn_SendReceiveData CSIEn_Init CSIEn_Start CSIEn_Stop CSIEn_SendData CSIEn_ReceiveData CSIEn_SendReceiveData CSIFn_Init CSIFn_Start CSIFn_Stop CSIFn_SendData CSIFn_ReceiveData CSIFn_SendReceiveData IIC0n_Init IIC0n_Stop IIC0n_StopCondition IIC0n_MasterSendStart IIC0n_MasterReceiveStart IIC0n_SlaveSendStart IIC0n_SlaveReceiveStart
	CG_serial_user.c	MD_INTUA*n*T MD_INTUA*n*R MD_INTUB*n*TIT MD_INTUB*n*TIF MD_INTUB*n*TIR MD_INTUB*n*TIRE MD_INTUB*n*TITO MD_INTCB*n*T MD_INTCB*n*R MD_INTUC*n*T MD_INTUC*n*R MD_INTCE*n*T MD_INTCE*n*TIOF MD_INTCF*n*T

周辺機能	ファイル名	含まれる API 関数名
シリアル	CG_serial_user.c	MD_INTCF*n*R MD_INTIIC*n* UARTAn_UserInit UARTAn_SendEndCallback UARTAn_ReceiveEndCallback UARTAn_ErrorCallback UARTAn_SoftOverRunCallback UARTBn_UserInit UARTBn_SendEndCallback UARTBn_ReceiveEndCallback UARTBn_SingleErrorCallback UARTBn_FIFOErrorCallback UARTBn_TimeoutErrorCallback UARTBn_SoftOverRunCallback UARTCn_UserInit UARTCn_SendEndCallback UARTCn_ReceiveEndCallback UARTCn_ErrorCallback UARTCn_SoftOverRunCallback CSIBn_UserInit CSIBn_SendEndCallback CSIBn_ReceiveEndCallback CSIBn_ErrorCallback CSIEn_UserInit CSIEn_SendEndCallback CSIEn_ReceiveEndCallback CSIEn_ErrorCallback CSIFn_UserInit CSIFn_SendEndCallback CSIFn_ReceiveEndCallback CSIFn_ErrorCallback IIC0n_UserInit IIC0n_MasterSendEndCallback IIC0n_MasterReceiveEndCallback IIC0n_MasterErrorCallback IIC0n_SlaveSendEndCallback IIC0n_SlaveReceiveEndCallback IIC0n_SlaveErrorCallback IIC0n_GetStopConditionCallback
	CG_serial.h	–
A/D コンバータ	CG_ad.c	AD_Init AD_Start AD_Stop AD_SelectADChannel

周辺機能	ファイル名	含まれる API 関数名
A/D コンバータ	CG_ad.c	AD_SetPFTCondition AD_Read AD_ReadByte
	CG_ad_user.c	MD_INTAD AD_UserInit
	CG_ad.h	—
D/A コンバータ	CG_da.c	DAn_Init DAn_Start DAn_Stop DAn_SetValue
	CG_da_user.c	DAn_UserInit
	CG_da.h	—
タイマ	CG_timer.c	TMPn_Init TMPn_Start TMPn_Stop TMPn_ChangeTimerCondition TMPn_GetPulseWidth TMPn_GetFreeRunningValue TMPn_ChangeDuty TMPn_SoftwareTriggerOn TMQ0_Init TMQ0_Start TMQ0_Stop TMQ0_ChangeTimerCondition TMQ0_GetPulseWidth TMQ0_GetFreeRunningValue TMQ0_ChangeDuty TMQ0_SoftwareTriggerOn TAAn_Init TAAn_Start TAAn_Stop TAAn_ChangeTimerCondition TAAn_ControlOutputToggle TAAn_GetPulseWidth TAAn_GetFreeRunningValue TAAn_ChangeDuty TAAn_SoftwareTriggerOn TABn_Init TABn_Start TABn_Stop TABn_ChangeTimerCondition TABn_ControlOutputToggle TABn_GetPulseWidth

周辺機能	ファイル名	含まれる API 関数名
タイマ	CG_timer.c	TABn_GetFreeRunningValue TABn_ChangeDuty TABn_SoftwareTriggerOn TMT0_Init TMT0_Start TMT0_Stop TMT0_ChangeTimerCondition TMT0_GetPulseWidth TMT0_GetFreeRunningValue TMT0_ChangeDuty TMT0_SoftwareTriggerOn TMT0_EnableHold TMT0_DisableHold TMT0_ChangeCountValue TMMn_Init TMMn_Start TMMn_Stop TMMn_ChangeTimerCondition
	CG_timer_user.c	MD_INTTP*n*OV MD_INTTP*n*CC*m* MD_INTTQ0OV MD_INTTQ0CC*m* MD_INTTAA*n*OV MD_INTTAA*n*CC*m* MD_INTTAB*n*OV MD_INTTAB*n*CC*m* MD_INTTT0EC MD_INTTT0OV MD_INTTT0CC*m* MD_INTTM*n*EQ0 TMPn_UserInit TMQ0_UserInit TAAn_UserInit TABn_UserInit TMT0_UserInit TMMn_UserInit
	CG_timer.h	–
時計タイマ	CG_wt.c	WT_Init WT_Start WT_Stop
	CG_wt_user.c	MD_INTWT MD_INTWTI WT_UserInit
	CG_wt.h	–

周辺機能	ファイル名	含まれる API 関数名
リアルタイム・カウンタ	CG_rtc.c	RTC_Init RTC_CounterEnable RTC_CounterDisable RTC_SetHourSystem RTC_CounterSet RTC_CounterGet RTC_ConstPeriodInterruptEnable RTC_ConstPeriodInterruptDisable RTC_AlarmEnable RTC_AlarmDisable RTC_AlarmSet RTC_AlarmGet RTC_IntervalStart RTC_IntervalStop RTC_IntervalInterruptEnable RTC_IntervalInterruptDisable RTC_RC1CK1HZ_OutputEnable RTC_RC1CK1HZ_OutputDisable RTC_RC1CKO_OutputEnable RTC_RC1CKO_OutputDisable RTC_RC1CKDIV_OutputEnable RTC_RC1CKDIV_OutputDisable RTC_RTC1HZ_OutputEnable RTC_RTC1HZ_OutputDisable RTC_RTCCL_OutputEnable RTC_RTCCL_OutputDisable RTC_RTCDIV_OutputEnable RTC_RTCDIV_OutputDisable RTC_ChangeCorrectionValue
	CG_rtc_user.c	MD_INTRTC*n* RTC_UserInit
	CG_rtc.h	—
リアルタイム出力機能	CG_rto.c	RTOn_Init RTOn_Enable RTOn_Disable RTOn_Set2BitsData RTOn_Set4BitsData RTOn_Set6BitsData RTOn_Set8BitsData RTOn_SetHigh2BitsData RTOn_SetLow2BitsData RTOn_SetHigh4BitsData RTOn_SetLow4BitsData RTOn_GetValue

周辺機能	ファイル名	含まれる API 関数名
リアルタイム出力機能	CG_rto_user.c	RTOn_UserInit
	CG_rto.h	—
DMA コントローラ	CG_dma.c	DMAn_Init DMAn_Enable DMAn_Disable DMAn_CheckStatus DMAn_SetData DMAn_SoftwareTriggerOn
	CG_dma_user.c	MD_INTDMA*n* DMAn_UserInit
	CG_dma.h	—
低電圧検出回路	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Start LVI_Stop
	CG_lvi_user.c	MD_INTLVI LVI_UserInit
	CG_lvi.h	—

第 3 章　API 関数

本付録では，Applilet3 が出力する API 関数について説明します。

3. 1　概　　要

以下に，Applilet3 が API 関数を出力する際の命名規則を示します。

- マクロ名

  すべて大文字。

  なお，先頭に“数字”が付与されている場合，該当数字（16 進数値）とマクロ値は同値。

- ローカル変数名

  すべて小文字。

- グローバル変数名

  先頭に“g”を付与し，構成単語の先頭のみ大文字。

- ローカル変数へのポインタ名

  先頭に“p”を付与し，すべて小文字。

- グローバル変数へのポインタ名

  先頭に“gp”を付与し，構成単語の先頭のみ大文字。

- 列挙指定子 enum の要素名

  すべて大文字。

3. 2　出力関数

以下に，Applilet3 が出力する API 関数の一覧を示します。

表 3—1　API 関数一覧

周辺機能	API 関数名	機能概要
システム	CLOCK_Init	クロックの機能を制御するうえで必要となる初期化処理を行います。
	CLOCK_UserInit	クロックに関するユーザ独自の初期化処理を行います。
	CG_ReadResetSource	リセットの発生に伴う処理を行います。
	CG_ChangeClockMode	CPU クロックを変更します。

周辺機能	API 関数名	機能概要
システム	CG_ChangeFrequency	CPU クロックの分周比を変更します。
	CG_SelectPowerSaveMode	CPU のスタンバイ・モードを設定します。
	CG_SelectStabTime	STOP モードが解除された際に必要となる X1 発振回路の発振安定時間を選択します。
	CG_SelectPllMode	PLL 機能の動作モードを選択します。
	CG_SelectSSCGMode	SSCG（Spread Spectrum Clock Generator）の動作状態を選択します。
	WDT2_Restart	ウォッチドッグ・タイマのカウンタをクリアしたのち，カウント処理を再開します。
	CRC_Start	データ・ブロックの誤り検出動作を開始します。
	CRC_SetData	CRC インプット・レジスタ（CRCIN）にデータを設定します。
	CRC_GetResult	CRC データ・レジスタ（CRCD）に格納されている演算結果を読み出します。
外部バス	BUS_Init	外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要となる初期化処理を行います。
	BUS_UserInit	外部バス・インタフェースに関するユーザ独自の初期化処理を行います。
ポート	PORT_Init	ポートの機能を制御するうえで必要となる初期化処理を行います。
	PORT_UserInit	ポートに関するユーザ独自の初期化処理を行います。
	PORT_ChangePmnInput	端子の入出力モードを出力モードから入力モードへと切り替えます。
	PORT_ChangePmnOutput	端子の入出力モードを入力モードから出力モードへと切り替えます。
割り込み	INTP_Init	外部割り込み INTP*n* の機能を制御するうえで必要となる初期化処理を行います。
	INTP_UserInit	外部割り込み INTP*n* に関するユーザ独自の初期化処理を行います。
	KEY_Init	キー割り込み INTKR の機能を制御するうえで必要となる初期化処理を行います。
	KEY_UserInit	キー割り込み INTKR に関するユーザ独自の初期化処理を行います。
	INT_MaskableInterruptEnable	マスカブル割り込みの受け付けを禁止／許可します。
	INTPn_Disable	マスカブル割り込み（外部割込み要求）INTP*n* の受け付けを禁止します。

周辺機能	API 関数名	機能概要
割り込み	INTPn_Enable	マスカブル割り込み（外部割込み要求）INTP*n* の受け付けを許可します。
	KEY_Disable	キー割り込み INTKR の受け付けを禁止します。
	KEY_Enable	キー割り込み INTKR の受け付けを許可します。
シリアル	UARTAn_Init	アシンクロナス・シリアル・インタフェース A（UARTA）の機能を制御するうえで必要となる初期化処理を行います。
	UARTAn_UserInit	アシンクロナス・シリアル・インタフェース A（UARTA）に関するユーザ独自の初期化処理を行います。
	UARTAn_Start	アシンクロナス・シリアル・インタフェース A（UARTA）を動作許可状態へと移行します。
	UARTAn_Stop	アシンクロナス・シリアル・インタフェース A（UARTA）を動作禁止状態へと移行します。
	UARTAn_SendData	データの UARTA*n* 送信を開始します。
	UARTAn_ReceiveData	データの UARTA*n* 受信を開始します。
	UARTAn_SendEndCallback	UARTA*n* の連続送信許可割り込み INTUA*n*T の発生に伴う処理を行います。
	UARTAn_ReceiveEndCallback	UARTA*n* の受信終了割り込み INTUA*n*R の発生に伴う処理を行います。
	UARTAn_ErrorCallback	UARTA*n* 受信エラー割り込み INTUA*n*R の発生に伴う処理を行います。
	UARTAn_SoftOverRunCallback	オーバラン・エラーの発生に伴う処理を行います。
	UARTBn_Init	アシンクロナス・シリアル・インタフェース B（UARTB）の機能を制御するうえで必要となる初期化処理を行います。
	UARTBn_UserInit	アシンクロナス・シリアル・インタフェース B（UARTB）に関するユーザ独自の初期化処理を行います。
	UARTBn_Start	アシンクロナス・シリアル・インタフェース B（UARTB）を動作許可状態へと移行します。
	UARTBn_Stop	アシンクロナス・シリアル・インタフェース B（UARTB）を動作禁止状態へと移行します。
	UARTBn_SendData	データの UARTB*n* 送信を開始します。
	UARTBn_ReceiveData	データの UARTB*n* 受信を開始します。
	UARTBn_SendEndCallback	UARTB*n* の送信許可割り込み INTUB*n*TIT，および FIFO 送信完了割り込み INTUB*n*TIF の発生に伴う処理を行います。
	UARTBn_ReceiveEndCallback	UARTB*n* の受信完了割り込み INTUB*n*TIR の発生に伴う処理を行います。

周辺機能	API 関数名	機能概要
シリアル	UARTBn_SingleErrorCallback	受信エラー割り込み INTUB*n*TIRE（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。
	UARTBn_FIFOErrorCallback	受信エラー割り込み INTUB*n*TIRE（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。
	UARTBn_TimeoutErrorCallback	受信タイムアウト割り込み INTUB*n*TITO の発生に伴う処理を行います。
	UARTBn_SoftOverRunCallback	オーバラン・エラーの発生に伴う処理を行います。
	UARTCn_Init	アシンクロナス・シリアル・インタフェース C（UARTC）の機能を制御するうえで必要となる初期化処理を行います。
	UARTCn_UserInit	アシンクロナス・シリアル・インタフェース C（UARTC）に関するユーザ独自の初期化処理を行います。
	UARTCn_Start	アシンクロナス・シリアル・インタフェース C（UARTC）を動作許可状態へと移行します。
	UARTCn_Stop	アシンクロナス・シリアル・インタフェース C（UARTC）を動作禁止状態へと移行します。
	UARTCn_SendData	データの UARTC*n* 送信を開始します。
	UARTCn_ReceiveData	データの UARTC*n* 受信を開始します。
	UARTCn_SendEndCallback	UARTC*n* の連続送信許可割り込み INTUC*n*T の発生に伴う処理を行います。
	UARTCn_ReceiveEndCallback	UARTC*n* の受信終了割り込み INTUC*n*R の発生に伴う処理を行います。
	UARTCn_ErrorCallback	UARTC*n* 受信エラー割り込み INTUC*n*R の発生に伴う処理を行います。
	UARTCn_SoftOverRunCallback	オーバラン・エラーの発生に伴う処理を行います。
	CSIBn_Init	3 線式可変長シリアル I/O B（CSIB）の機能を制御するうえで必要となる初期化処理を行います。
	CSIBn_UserInit	3 線式可変長シリアル I/O B（CSIB）に関するユーザ独自の初期化処理を行います。
	CSIBn_Start	3 線式可変長シリアル I/O B（CSIB）を動作許可状態へと移行します。
	CSIBn_Stop	3 線式可変長シリアル I/O B（CSIB）を動作禁止状態へと移行します。
	CSIBn_SendData	データの CSIB 送信を開始します。
	CSIBn_ReceiveData	データの CSIB 受信を開始します。
	CSIBn_SendReceiveData	データの CSIB 送受信を開始します。

周辺機能	API 関数名	機能概要
シリアル	CSIBn_SendEndCallback	CSIB*n* の受信終了割り込み INTCB*n*R，および CSIB*n* の連続送信書き込み許可割り込み INTCB*n*T の発生に伴う処理を行います。
	CSIBn_ReceiveEndCallback	CSIB*n* の受信終了割り込み INTCB*n*R の発生に伴う処理を行います。
	CSIBn_ErrorCallback	CSIB*n* の受信エラー割り込み INTCB*n*R（オーバラン・エラー）の発生に伴う処理を行います。
	CSIEn_Init	3 線式可変長シリアル I/O E（CSIE）の機能を制御するうえで必要となる初期化処理を行います。
	CSIEn_UserInit	3 線式可変長シリアル I/O E（CSIE）に関するユーザ独自の初期化処理を行います。
	CSIEn_Start	3 線式可変長シリアル I/O E（CSIE）を動作許可状態へと移行します。
	CSIEn_Stop	3 線式可変長シリアル I/O E（CSIE）を動作禁止状態へと移行します。
	CSIEn_SendData	データの CSIE 送信を開始します。
	CSIEn_ReceiveData	データの CSIE 受信を開始します。
	CSIEn_SendReceiveData	データの CSIE 送受信を開始します。
	CSIEn_SendEndCallback	CSIE*n* の送受信完了割り込み INTCE*n*T の発生に伴う処理を行います。
	CSIEn_ReceiveEndCallback	CSIE*n* の送受信完了割り込み INTCE*n*T の発生に伴う処理を行います。
	CSIEn_ErrorCallback	CSIE*n* の CSIE*n*BUF オーバフロー割り込み INTCE*n*TIOF の発生に伴う処理を行います。
	CSIFn_Init	3 線式可変長シリアル I/O F（CSIF）の機能を制御するうえで必要となる初期化処理を行います。
	CSIFn_UserInit	3 線式可変長シリアル I/O F（CSIF）に関するユーザ独自の初期化処理を行います。
	CSIFn_Start	3 線式可変長シリアル I/O F（CSIF）を動作許可状態へと移行します。
	CSIFn_Stop	3 線式可変長シリアル I/O F（CSIF）を動作禁止状態へと移行します。
	CSIFn_SendData	データの CSIF 送信を開始します。
	CSIFn_ReceiveData	データの CSIF 受信を開始します。
	CSIFn_SendReceiveData	データの CSIF 送受信を開始します。
	CSIFn_SendEndCallback	CSIF*n* の送受信完了割り込み INTCF*n*T の発生に伴う処理を行います。
	CSIFn_ReceiveEndCallback	CSIF*n* の送受信完了割り込み INTCF*n*T の発生に伴う処理を行います。

周辺機能	API 関数名	機能概要
シリアル	CSIFn_ErrorCallback	CSIF*n* の受信エラー割り込み INTCF*n*R（オーバラン・エラー）の発生に伴う処理を行います。
	IIC0n_Init	IIC バスの機能を制御するうえで必要となる初期化処理を行います。
	IIC0n_UserInit	IIC バスに関するユーザ独自の初期化処理を行います。
	IIC0n_Stop	IIC0*n* 通信を終了します。
	IIC0n_StopCondition	ストップ・コンディションを生成します。
	IIC0n_MasterSendStart	データの IIC0*n* マスタ送信を開始します。
	IIC0n_MasterReceiveStart	データの IIC0*n* マスタ受信を開始します。
	IIC0n_MasterSendEndCallback	IIC0*n* マスタ送信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
	IIC0n_MasterReceiveEndCallback	IIC0*n* マスタ受信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
	IIC0n_MasterErrorCallback	IIC0*n* マスタ通信エラーの検出に伴う処理を行います。
	IIC0n_SlaveSendStart	データの IIC0*n* スレーブ送信を開始します。
	IIC0n_SlaveReceiveStart	データの IIC0*n* スレーブ受信を開始します。
	IIC0n_SlaveSendEndCallback	IIC0*n* スレーブ送信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
	IIC0n_SlaveReceiveEndCallback	IIC0*n* スレーブ受信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
	IIC0n_SlaveErrorCallback	IIC0*n* スレーブ通信エラーの検出に伴う処理を行います。
	IIC0n_GetStopConditionCallback	ストップ・コンディションの検出に伴う処理を行います。
A/D コンバータ	AD_Init	A/D コンバータの機能を制御するうえで必要となる初期化処理を行います。
	AD_UserInit	A/D コンバータに関するユーザ独自の初期化処理を行います。
	AD_Start	A/D 変換を開始します。
	AD_Stop	A/D 変換を終了します。
	AD_SelectADChannel	A/D 変換するアナログ電圧の入力端子を設定します。
	AD_SetPFTCondition	パワー・フェイル比較モードで動作する際の情報（比較値，A/D 変換終了割り込み INTAD の発生要因）を設定します。
	AD_Read	A/D 変換結果（10 ビット）を読み出します。
	AD_ReadByte	A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を読み出します。

周辺機能	API 関数名	機能概要
D/A コンバータ	DAn_Init	D/A コンバータの機能を制御するうえで必要となる初期化処理を行います。
	DAn_UserInit	D/A コンバータに関するユーザ独自の初期化処理を行います。
	DAn_Start	D/A 変換を開始します。
	DAn_Stop	D/A 変換を終了します。
	DAn_SetValue	ANOn 端子に出力するアナログ電圧値を設定します。
タイマ	TMPn_Init	16 ビット・タイマ／イベント・カウンタ P（TMP）の機能を制御するうえで必要となる初期化処理を行います。
	TMPn_UserInit	16 ビット・タイマ／イベント・カウンタ P（TMP）に関するユーザ独自の初期化処理を行います。
	TMPn_Start	16 ビット・タイマ／イベント・カウンタ P（TMP）のカウントを開始します。
	TMPn_Stop	16 ビット・タイマ／イベント・カウンタ P（TMP）のカウントを終了します。
	TMPn_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ P（TMP）のカウント値を変更します。
	TMPn_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ P（TMP）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
	TMPn_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ P（TMP）がキャプチャした値を読み出します。
	TMPn_ChangeDuty	PWM 信号のデューティ比を変更します。
	TMPn_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
	TMQ0_Init	16 ビット・タイマ／イベント・カウンタ Q（TMQ）の機能を制御するうえで必要となる初期化処理を行います。
	TMQ0_UserInit	16 ビット・タイマ／イベント・カウンタ Q（TMQ）に関するユーザ独自の初期化処理を行います。
	TMQ0_Start	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のカウントを開始します。
	TMQ0_Stop	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のカウントを終了します。
	TMQ0_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のカウント値を変更します。

周辺機能	API 関数名	機能概要
タイマ	TMQ0_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
	TMQ0_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ Q（TMQ）がキャプチャした値を読み出します。
	TMQ0_ChangeDuty	PWM 信号のデューティ比を変更します。
	TMQ0_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
	TAAn_Init	16 ビット・タイマ／イベント・カウンタ AA（TAA）の機能を制御するうえで必要となる初期化処理を行います。
	TAAn_UserInit	16 ビット・タイマ／イベント・カウンタ AA（TAA）に関するユーザ独自の初期化処理を行います。
	TAAn_Start	16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウントを開始します。
	TAAn_Stop	16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウントを終了します。
	TAAn_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウント値を変更します。
	TAAn_ControlOutputToggle	16 ビット・タイマ／イベント・カウンタ AA（TAA）のトグル制御を変更します。
	TAAn_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ AA（TAA）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
	TAAn_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ AA（TAA）がキャプチャした値を読み出します。
	TAAn_ChangeDuty	PWM 信号のデューティ比を変更します。
	TAAn_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
	TABn_Init	16 ビット・タイマ／イベント・カウンタ AB（TAB）の機能を制御するうえで必要となる初期化処理を行います。
	TABn_UserInit	16 ビット・タイマ／イベント・カウンタ AB（TAB）に関するユーザ独自の初期化処理を行います。
	TABn_Start	16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウントを開始します。
	TABn_Stop	16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウントを終了します。

周辺機能	API 関数名	機能概要
タイマ	TABn_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウント値を変更します。
	TABn_ControlOutputToggle	16 ビット・タイマ／イベント・カウンタ AB（TAB）のトグル制御を変更します。
	TABn_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ AB（TAB）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
	TABn_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ AB（TAB）がキャプチャした値を読み出します。
	TABn_ChangeDuty	PWM 信号のデューティ比を変更します。
	TABn_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
	TMT0_Init	16 ビット・タイマ／イベント・カウンタ T（TMT）の機能を制御するうえで必要となる初期化処理を行います。
	TMT0_UserInit	16 ビット・タイマ／イベント・カウンタ T（TMT）に関するユーザ独自の初期化処理を行います。
	TMT0_Start	16 ビット・タイマ／イベント・カウンタ T（TMT）のカウントを開始します。
	TMT0_Stop	16 ビット・タイマ／イベント・カウンタ T（TMT）のカウントを終了します。
	TMT0_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ T（TMT）のカウント値を変更します。
	TMT0_GetPulseWidth	16 ビット・タイマ・イベント・カウンタ T（TMT）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
	TMT0_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ T（TMT）がキャプチャした値を読み出します。
	TMT0_ChangeDuty	PWM 信号のデューティ比を変更します。
	TMT0_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
	TMT0_EnableHold	16 ビット・タイマ／イベント・カウンタ T（TMT）のエンコーダ・カウンタ制御を保持動作へと変更します。
	TMT0_DisableHold	16 ビット・タイマ／イベント・カウンタ T（TMT）のエンコーダ・カウンタ制御を通常動作へと変更します。
	TMT0_ChangeCountValue	16 ビット・タイマ／イベント・カウンタ T（TMT）の初期カウント値を変更します。

周辺機能	API 関数名	機能概要
タイマ	TMMn_Init	16 ビット・インターバル・タイマ M（TMM）の機能を制御するうえで必要となる初期化処理を行います。
	TMMn_UserInit	16 ビット・インターバル・タイマ M（TMM）に関するユーザ独自の初期化処理を行います。
	TMMn_Start	16 ビット・インターバル・タイマ M（TMM）のカウントを開始します。
	TMMn_Stop	16 ビット・インターバル・タイマ M（TMM）のカウントを終了します。
	TMMn_ChangeTimerCondition	16 ビット・インターバル・タイマ M（TMM）のカウント値を変更します。
時計タイマ	WT_Init	時計タイマの機能を制御するうえで必要となる初期化処理を行います。
	WT_UserInit	時計タイマに関するユーザ独自の初期化処理を行います。
	WT_Start	時計タイマのカウンタをクリアしたのち，カウント処理を再開します。
	WT_Stop	時計タイマのカウント処理を中断します。
リアルタイム・カウンタ	RTC_Init	リアルタイム・カウンタの機能を制御するうえで必要となる初期化処理を行います。
	RTC_UserInit	リアルタイム・カウンタに関するユーザ独自の初期化処理を行います。
	RTC_CounterEnable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを開始します。
	RTC_CounterDisable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを終了します。
	RTC_SetHourSystem	リアルタイム・カウンタの時間制（12 時間制，24 時間制）を設定します。
	RTC_CounterSet	リアルタイム・カウンタにカウント値（年，月，曜日，日，時，分，秒）を設定します。
	RTC_CounterGet	リアルタイム・カウンタのカウント値（年，月，曜日，日，時，分，秒）を読み出します。
	RTC_ConstPeriodInterruptEnable	割り込み INTRTC0 の発生周期を設定したのち，定周期割り込み機能を開始します。
	RTC_ConstPeriodInterruptDisable	定周期割り込み機能を終了します。
	RTC_AlarmEnable	アラーム割り込み機能を開始します。
	RTC_AlarmDisable	アラーム割り込み機能を終了します。
	RTC_AlarmSet	アラームの発生条件（曜日，時，分）を設定します。

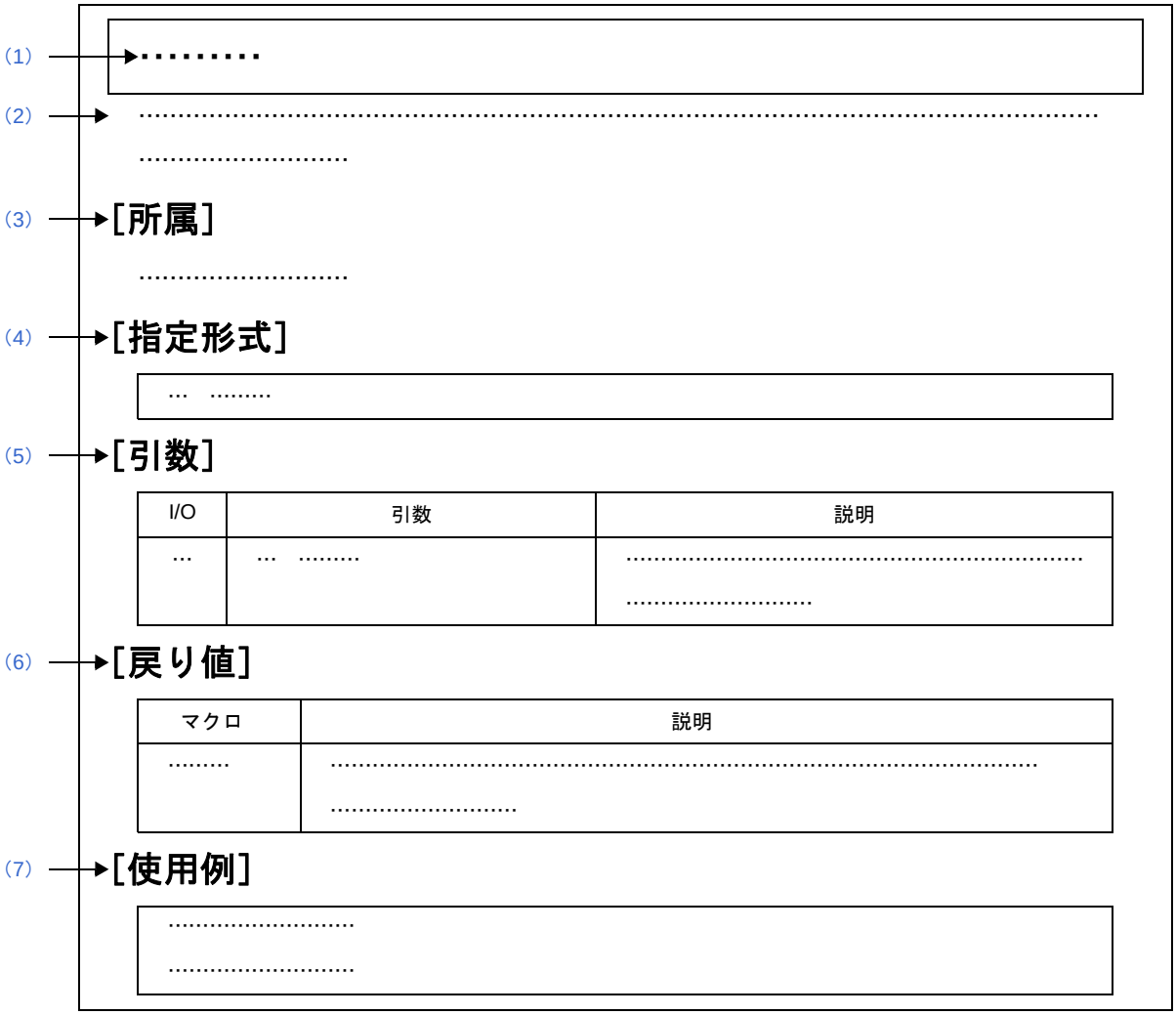
周辺機能	API 関数名	機能概要
リアルタイム・カウンタ	RTC_AlarmGet	アラームの発生条件（曜日，時，分）を読み出します。
	RTC_IntervalStart	インターバル割り込み機能を開始します。
	RTC_IntervalStop	インターバル割り込み機能を終了します。
	RTC_IntervalInterruptEnable	割り込み INTRTC2 の発生周期を設定したのち，インターバル割り込み機能を開始します。
	RTC_IntervalInterruptDisable	インターバル割り込み機能を終了します。
	RTC_RC1CK1HZ_OutputEnable	RC1CK1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。
	RTC_RC1CK1HZ_OutputDisable	RC1CK1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。
	RTC_RC1CKO_OutputEnable	RC1CKO 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。
	RTC_RC1CKO_OutputDisable	RC1CKO 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。
	RTC_RC1CKDIV_OutputEnable	RC1CKDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。
	RTC_RC1CKDIV_OutputDisable	RC1CKDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
	RTC_RTC1HZ_OutputEnable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。
	RTC_RTC1HZ_OutputDisable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。
	RTC_RTCCL_OutputEnable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。
	RTC_RTCCL_OutputDisable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。
	RTC_RTCDIV_OutputEnable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。
	RTC_RTCDIV_OutputDisable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
	RTC_ChangeCorrectionValue	時計誤差を補正するタイミング，および補正値を変更します。
リアルタイム出力機能	RTOn_Init	リアルタイム出力機能を制御するうえで必要となる初期化処理を行います。
	RTOn_UserInit	リアルタイム出力に関するユーザ独自の初期化処理を行います。
	RTOn_Enable	リアルタイム出力を許可します。
	RTOn_Disable	リアルタイム出力を禁止します。

周辺機能	API 関数名	機能概要
リアルタイム出力機能	RTOn_Set2BitsData	リアルタイム出力する 2 ビット・データを設定します。
	RTOn_Set4BitsData	リアルタイム出力する 4 ビット・データを設定します。
	RTOn_Set6BitsData	リアルタイム出力する 6 ビット・データを設定します。
	RTOn_Set8BitsData	リアルタイム出力する 8 ビット・データを設定します。
	RTOn_SetHigh2BitsData	リアルタイム出力する上位 2 ビット・データを設定します。
	RTOn_SetLow2BitsData	リアルタイム出力する下位 2 ビット・データを設定します。
	RTOn_SetHigh4BitsData	リアルタイム出力する上位 4 ビット・データを設定します。
	RTOn_SetLow4BitsData	リアルタイム出力する下位 4 ビット・データを設定します。
	RTOn_GetValue	リアルタイム出力しているデータを読み出します。
DMA コントローラ	DMAn_Init	DMA コントローラの機能を制御するうえで必要となる初期化処理を行います。
	DMAn_UserInit	DMA コントローラに関するユーザ独自の初期化処理を行います。
	DMAn_Enable	チャネル *n* を動作許可状態に設定します。
	DMAn_Disable	チャネル *n* を動作停止状態に設定します。
	DMAn_CheckStatus	転送状態（転送終了，転送中）を読み出します。
	DMAn_SetData	転送先／転送元の RAM アドレス，およびデータの転送回数を設定します。
	DMAn_SoftwareTriggerOn	DMA 転送の起動要因として，ソフトウエア・トリガを使用します。
低電圧検出回路	LVI_Init	低電圧検出回路の機能を制御するうえで必要となる初期化処理を行います。
	LVI_UserInit	低電圧検出回路に関するユーザ独自の初期化処理を行います。
	LVI_InterruptModeStart	低電圧検出動作を開始します（割り込み発生モード時）。
	LVI_ResetModeStart	低電圧検出動作を開始します（内部リセット・モード時）。
	LVI_Start	低電圧検出動作を開始します。
	LVI_Stop	低電圧検出動作を停止します。

3.3 関数リファレンス

本節では，Applilet3 が出力する API 関数について，次の記述フォーマットに従って説明します。

図 3—1 API 関数の記述フォーマット



(1) 名称

API 関数の名称を示しています。

(2) 機能

API 関数の機能概要を示しています。

(3) [所属]

API 関数が出力される C ソース・ファイル名を示しています。

(4) [指定形式]

API 関数を C 言語で呼び出す際の記述形式を示しています。

(5) [引数]

API 関数の引数を次の形式で示しています。

I/O	引数	説明
(a)	(b)	(c)

(a) I/O

引数の種類

I … 入力引数

O … 出力引数

(b) 引数

引数のデータ・タイプ

(c) 説明

引数の説明

(6) [戻り値]

API 関数からの戻り値を次の形式で示しています。

マクロ	説明
(a)	(b)

(a) マクロ

戻り値のマクロ

(b) 説明

戻り値の説明

(7) [使用例]

API 関数の使用例を示しています。

3. 3. 1 システム

以下に，Applilet3 がシステム用として出力する API 関数の一覧を示します。

表 3—2 システム用 API 関数

API 関数名	機能概要
CLOCK_Init	クロックの機能を制御するうえで必要となる初期化処理を行います。
CLOCK_UserInit	クロックに関するユーザ独自の初期化処理を行います。
CG_ReadResetSource	リセットの発生に伴う処理を行います。
CG_ChangeClockMode	CPU クロックを変更します。
CG_ChangeFrequency	CPU クロックの分周比を変更します。
CG_SelectPowerSaveMode	CPU のスタンバイ・モードを設定します。
CG_SelectStabTime	STOP モードが解除された際に必要となる X1 発振回路の発振安定時間を選択します。
CG_SelectPllMode	PLL 機能の動作モードを選択します。
CG_SelectSSCGMode	SSCG（Spread Spectrum Clock Generator）の動作状態を選択します。
WDT2_Restart	ウォッチドッグ・タイマのカウンタをクリアしたのち，カウント処理を再開します。
CRC_Start	データ・ブロックの誤り検出動作を開始します。
CRC_SetData	CRC インプット・レジスタ（CRCIN）にデータを設定します。
CRC_GetResult	CRC データ・レジスタ（CRCD）に格納されている演算結果を読み出します。

CLOCK_Init

クロックの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_system.c

[指定形式]

```
void    CLOCK_Init ( void );
```

[引数]

なし

[戻り値]

なし

CLOCK_UserInit

クロックに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，CLOCK_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_system_user.c

［指定形式］

```
void    CLOCK_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

CG_ReadResetSource

リセットの発生に伴う処理を行います。

[所属]

CG_system_user.c

[指定形式]

```
void    CG_ReadResetSource ( void );
```

[引数]

なし

[戻り値]

なし

[使用例]

以下に，リセットの発生要因別に異なる処理を実行する際の例を示します。

【CG_systeminit.c】

```
void systeminit ( void ) {
    CG_ReadResetSource ();      /* リセットの発生要因別に処理を実行 */
    ……
}
```

【CG_system_user.c】

```
#include    "CG_macrodriver.h"
void CG_ReadResetSource ( void ) {
    UCHAR   resetflag = RESF;   /* リセット・コントロール・フラグ・レジスタ：RESF の内容確保 */
    if ( resetflag & 0x1 ) {            /* 発生要因の判別：LVIRF フラグのチェック */
        ……    /* 低電圧検出回路が低電圧を検出した際の処理 */
    } else if ( resetflag & 0x2 ) {     /* 発生要因の判別：CLMRF フラグのチェック */
        ……    /* クロック・モニタの発振停止を検出した際の処理 */
    } else if ( resetflag & 0x10 ) {    /* 発生要因の判別：WDT2RF フラグのチェック */
        ……    /* ウォッチドッグ・タイマ 2 のオーバフローを検出した際の処理 */
    }
    ……
```

```
}
```

CG_ChangeClockMode

CPU クロックを変更します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangeClockMode ( enum ClockMode mode );
```

[引数]

I/O	引数	説明
I	enum ClockMode *mode*;	CPU クロックの種類 MAINOSCCLK： メイン・クロック発振回路（fxx） SUBCLK： サブクロック発振回路（fXT）

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了 - サブクロック動作からメイン・クロック動作への変更ができませんでした
MD_ERROR2	異常終了 - メイン・クロック動作からサブクロック動作への変更ができませんでした
MD_ARGERROR	引数の指定が不正

CG_ChangeFrequency

CPU クロックの分周比を変更します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_ChangeFrequency ( enum CPUClock clock );
```

[引数]

I/O	引数	説明
I	enum CPUClock *clock*;	分周比の種類 SYSTEMCLOCK : fxx SYSONEHALF : fxx/2 SYSONEFOURTH : fxx/4 SYSONEEIGHTH : fxx/8 SYSONESIXTEENTH : fxx/16 SYSONETHIRTYSECOND : fxx/32

備考 fxx は，メイン・クロック発振回路の周波数を意味します。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了
MD_ARGERROR	引数の指定が不正

CG_SelectPowerSaveMode

CPU のスタンバイ・モードを選択します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectPowerSaveMode ( enum PSLevel level );
```

[引数]

I/O	引数	説明
I	enum PSLevel *level*;	スタンバイ・モードの種類 【E/Sx3-H】【ES/Jx3-E】【ES/Jx3-H】 PSSTOP : STOP モード PSHALT : HALT モード PSIDLE1 : IDLE1 モード PSIDLE2 : IDLE2 モード 【ES/Jx3】【ES/Jx3-L】 PSSTOP : STOP モード PSHALT : HALT モード

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CG_SelectStabTime

STOP モードが解除された際に必要となる X1 発振回路の発振安定時間を選択します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectStabTime ( enum StabTime waittime );
```

[引数]

I/O	引数	説明
I	enum StabTime *waittime*;	発振安定時間の種類 STLEVEL0： 2^10/fx STLEVEL1： 2^11/fx STLEVEL2： 2^12/fx STLEVEL3： 2^13/fx STLEVEL4： 2^14/fx STLEVEL5： 2^15/fx STLEVEL6： 2^16/fx

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CG_SelectPllMode

PLL 機能の動作モードを選択します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectPllMode ( enum PllMode pllmode );
```

[引数]

I/O	引数	説明
I	enum PllMode *pllmode*;	動作モードの種類 【E\Sx3-H】【ES/Jx3】【ES/Jx3-L】 SYSPLLOFF： クロック・スルー・モード SYS4PLL： 4 逓倍（PLL 機能使用時） SYS8PLL： 8 逓倍（PLL 機能使用時） 【ES/Jx3-E】【ES/Jx3-H】 SYSPLLOFF： PLL 停止 SYSPLLON： PLL 動作

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了【ES/Jx3】【ES/Jx3-E】【ES/Jx3-H】【ES/Jx3-L】
MD_ERROR1	異常終了【E/Sx3-H】 - 動作モードの変更はできません。
MD_ERROR2	異常終了【E/Sx3-H】 - 4 逓倍への変更はできません。
MD_ERROR3	異常終了【E/Sx3-H】 - 8 逓倍への変更はできません。
MD_ARGERROR	引数の指定が不正

CG_SelectSSCGMode

SSCG（Spread Spectrum Clock Generator）の動作状態を選択します。

［所属］

CG_system.c

［指定形式］

```
#include    "CG_macrodriver.h"
#include    "CG_system.h"
MD_STATUS   CG_SelectSSCGMode ( enum SSCGMode sscgmode );
```

［引数］

I/O	引数	説明
I	enum SSCGMode *sscgmode*;	動作状態の種類 SYSSSCGON： 動作許可状態 SYSSSCGOFF： 動作禁止状態

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了
MD_ARGERROR	引数の指定が不正

WDT2_Restart

ウォッチドッグ・タイマのカウンタをクリアしたのち，カウント処理を再開します。

［所属］

CG_system.c

［指定形式］

```
void    WDT2_Restart ( void );
```

［引数］

なし

［戻り値］

なし

CRC_Start

データ・ブロックの誤り検出動作を開始します。

［所属］

CG_system.c

［指定形式］

```
void    CRC_Start ( void );
```

［引数］

なし

［戻り値］

なし

CRC_SetData

CRC インプット・レジスタ（CRCIN）にデータを設定します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    CRC_SetData ( UCHAR data );
```

[引数]

I/O	引数	説明
I	UCHAR *data*;	設定するデータ

[戻り値]

なし

CRC_GetResult

CRC データ・レジスタ（CRCD）に格納されている演算結果を読み出します。

[所属]

CG_system.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    CRC_GetResult ( USHORT *result );
```

[引数]

I/O	引数	説明
O	USHORT *result;	読み出した演算結果を格納する領域へのポインタ

[戻り値]

なし

3. 3. 2　外部バス

以下に，Applilet3 が外部バス用として出力する API 関数の一覧を示します。

表 3—3　外部バス用 API 関数

API 関数名	機能概要
BUS_Init	外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要となる初期化処理を行います。
BUS_UserInit	外部バス・インタフェースに関するユーザ独自の初期化処理を行います。

BUS_Init

外部バス・インタフェースの機能（外部バスを内蔵 ROM，RAM，SFR 以外の領域に接続する機能）を制御するうえで必要となる初期化処理を行います。

［所属］

CG_bus.c

［指定形式］

```
void    BUS_Init ( void );
```

［引数］

なし

［戻り値］

なし

BUS_UserInit

外部バス・インタフェースに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，BUS_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_bus_user.c

［指定形式］

```
void    BUS_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

3. 3. 3 ポート

以下に，Applilet3 がポート用として出力する API 関数の一覧を示します。

表 3—4 ポート用 API 関数

API 関数名	機能概要
PORT_Init	ポートの機能を制御するうえで必要となる初期化処理を行います。
PORT_UserInit	ポートに関するユーザ独自の初期化処理を行います。
PORT_ChangePmnInput	端子の入出力モードを出力モードから入力モードへと切り替えます。
PORT_ChangePmnOutput	端子の入出力モードを入力モードから出力モードへと切り替えます。

PORT_Init

ポートの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_port.c

[指定形式]

```
void    PORT_Init ( void );
```

[引数]

なし

[戻り値]

なし

PORT_UserInit

ポートに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，PORT_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_port_user.c

［指定形式］

```
void    PORT_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

PORT_ChangeP*mn*Input

端子の入出力モードを出力モードから入力モードへと切り替えます。

[所属]

CG_port.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP*mn*Input ( void );
```

備考 *mn* は，ポート番号を意味します。

[引数]

なし

[戻り値]

なし

[使用例]

以下に，P00 端子の入出力モードを出力モードから入力モードへと切り替える際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP00Input ( );        /* 入出力モードの切り替え */
    .........
}
```

PORT_ChangeP*mn*Output

端子の入出力モードを入力モードから出力モードへと切り替えます。

[所属]

CG_port.c

[指定形式]

本 API 関数の指定形式は，対象端子で N-ch オープン・ドレーン出力が行われるか否かにより異なります。

- 【N-ch オープン・ドレーン出力：なし】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP*mn*Output ( BOOL initialvalue );
```

- 【N-ch オープン・ドレーン出力：あり】

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP*mn*Output ( BOOL enablench, BOOL initialvalue );
```

備考 *nm* は，ポート番号を意味します。

[引数]

I/O	引数	説明
I	BOOL *enablench*;	出力モードの種類 MD_TRUE： N-ch オープン・ドレーン出力（V_{DD} 耐圧）モード MD_FALSE： 通常出力モード
I	BOOL *initialvalue*;	初期出力値 MD_SET： High レベル“1”を出力 MD_CLEAR： Low レベル“0”を出力

[戻り値]

なし

[使用例 1]

以下に，P00 端子（N-ch オープン・ドレーン出力：なし）を

入出力モードの種類： 出力モード

初期出力値： High レベル“1”を出力

に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP00Output ( MD_SET );    /* 入出力モードの切り替え */
    .........
}
```

［使用例 2］

以下に，P04 端子（N-ch オープン・ドレーン出力：あり）を

入出力モードの種類： 出力モード

出力モードの種類： N-ch オープン・ドレーン出力（VDD 耐圧）モード

初期出力値： Low レベル“0”を出力

に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    PORT_ChangeP04Output ( MD_TRUE, MD_CLEAR ); /* 入出力モードの切り替え */
    .........
}
```

3. 3. 4　割り込み

以下に，Applilet3 が割り込み用として出力する API 関数の一覧を示します。

表 3—5　割り込み用 API 関数

API 関数名	機能概要
INTP_Init	外部割り込み INTP*n* の機能を制御するうえで必要となる初期化処理を行います。
INTP_UserInit	外部割り込み INTP*n* に関するユーザ独自の初期化処理を行います。
KEY_Init	キー割り込み INTKR の機能を制御するうえで必要となる初期化処理を行います。
KEY_UserInit	キー割り込み INTKR に関するユーザ独自の初期化処理を行います。
INT_MaskableInterruptEnable	マスカブル割り込みの受け付けを禁止／許可します。
INTPn_Disable	マスカブル割り込み（外部割込み要求）INTP*n* の受け付けを禁止します。
INTPn_Enable	マスカブル割り込み（外部割込み要求）INTP*n* の受け付けを許可します。
KEY_Disable	キー割り込み INTKR の受け付けを禁止します。
KEY_Enable	キー割り込み INTKR の受け付けを許可します。

INTP_Init

外部割り込み INTP*n* の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_int.c

［指定形式］

```
void    INTP_Init ( void );
```

［引数］

なし

［戻り値］

なし

INTP_UserInit

外部割り込み INTP*n* に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，INTP_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_int_user.c

［指定形式］

```
void    INTP_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

KEY_Init

キー割り込み INTKR の機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_int.c

[指定形式]

```
void    KEY_Init ( void );
```

[引数]

なし

[戻り値]

なし

KEY_UserInit

キー割り込み INTKR に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，KEY_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_int_user.c

［指定形式］

```
void    KEY_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

INT_MaskableInterruptEnable

マスカブル割り込みの受け付けを禁止／許可します。

[所属]

CG_int.c

[指定形式]

- 【E/Sx3-H】【ES/Jx3-E】【ES/Jx3-H】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

- 【ES/Jx3】【ES/Jx3-L】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void    INT_MaskableInterruptEnable ( enum MaskableSource name, BOOL enableflag );
```

[引数]

I/O	引数	説明
I	enum MaskableSource *name*;	マスカブル割り込みの種類 INT_*xxx*： マスカブル割り込み
I	BOOL *enableflag*;	受け付けの禁止／許可 MD_TRUE： 受け付けを許可 MD_FALSE： 受け付けを禁止

備考 マスカブル割り込みの種類 INT_*xxx* についての詳細は，ヘッダ・ファイル CG_int.h を参照してください。

[戻り値]

- 【E/Sx3-H】【ES/Jx3-E】【ES/Jx3-H】

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

- 【ES/Jx3】【ES/Jx3-L】

なし

［使用例 1］

以下に，マスカブル割り込み INTP0 の受け付けを“禁止”に設定する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ……
    INT_MaskableInterruptEnable ( INT_INTP0, MD_FALSE );  /* マスカブル割り込みINTP0の受け付け禁止
*/
    ……
}
```

［使用例 2］

以下に，マスカブル割り込み INTP0 の受け付けを“許可”に設定する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
void main ( void ) {
    ……
    INT_MaskableInterruptEnable ( INT_INTP0, MD_TRUE );   /* マスカブル割り込みINTP0の受け付け許可
*/
    ……
}
```

INTP*n*_Disable

マスカブル割り込み（外部割り込み要求）INTP*n* の受け付けを禁止します。

［所属］

CG_int.c

［指定形式］

```
void    INTPn_Disable ( void );
```

備考　*n* は，割り込み要因番号を意味します。

［引数］

なし

［戻り値］

なし

INTP*n*_Enable

マスカブル割り込み（外部割り込み要求）INTP*n* の受け付けを許可します。

［所属］

CG_int.c

［指定形式］

```
void    INTPn_Enable ( void );
```

備考　*n* は，割り込み要因番号を意味します。

［引数］

なし

［戻り値］

なし

KEY_Disable

キー割り込み INTKR の受け付けを禁止します。

[所属]

CG_int.c

[指定形式]

```
void    KEY_Disable ( void );
```

[引数]

なし

[戻り値]

なし

KEY_Enable

キー割り込み INTKR の受け付けを許可します。

[所属]

CG_int.c

[指定形式]

```
void    KEY_Enable ( void );
```

[引数]

なし

[戻り値]

なし

3. 3. 5 シリアル

以下に，Applilet3 がシリアル用として出力する API 関数の一覧を示します。

表 3—6 シリアル用 API 関数

API 関数名	機能概要
UARTAn_Init	アシンクロナス・シリアル・インタフェース A（UARTA）の機能を制御するうえで必要となる初期化処理を行います。
UARTAn_UserInit	アシンクロナス・シリアル・インタフェース A（UARTA）に関するユーザ独自の初期化処理を行います。
UARTAn_Start	アシンクロナス・シリアル・インタフェース A（UARTA）を動作許可状態へと移行します。
UARTAn_Stop	アシンクロナス・シリアル・インタフェース A（UARTA）を動作禁止状態へと移行します。
UARTAn_SendData	データの UARTA*n* 送信を開始します。
UARTAn_ReceiveData	データの UARTA*n* 受信を開始します。
UARTAn_SendEndCallback	UARTA*n* の連続送信許可割り込み INTUA*n*T の発生に伴う処理を行います。
UARTAn_ReceiveEndCallback	UARTA*n* の受信終了割り込み INTUA*n*R の発生に伴う処理を行います。
UARTAn_ErrorCallback	UARTA*n* 受信エラー割り込み INTUA*n*R（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。
UARTAn_SoftOverRunCallback	オーバラン・エラーの発生に伴う処理を行います。
UARTBn_Init	アシンクロナス・シリアル・インタフェース B（UARTB）の機能を制御するうえで必要となる初期化処理を行います。
UARTBn_UserInit	アシンクロナス・シリアル・インタフェース B（UARTB）に関するユーザ独自の初期化処理を行います。
UARTBn_Start	アシンクロナス・シリアル・インタフェース B（UARTB）を動作許可状態へと移行します。
UARTBn_Stop	アシンクロナス・シリアル・インタフェース B（UARTB）を動作禁止状態へと移行します。
UARTBn_SendData	データの UARTB*n* 送信を開始します。
UARTBn_ReceiveData	データの UARTB*n* 受信を開始します。
UARTBn_SendEndCallback	UARTB*n* の送信許可割り込み INTUB*n*TIT，および FIFO 送信完了割り込み INTUB*n*TIF の発生に伴う処理を行います。
UARTBn_ReceiveEndCallback	UARTB*n* の受信完了割り込み INTUB*n*TIR の発生に伴う処理を行います。
UARTBn_SingleErrorCallback	受信エラー割り込み INTUB*n*TIRE（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。
UARTBn_FIFOErrorCallback	受信エラー割り込み INTUB*n*TIRE（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。
UARTBn_TimeoutErrorCallback	受信タイムアウト割り込み INTUB*n*TITO の発生に伴う処理を行います。
UARTBn_SoftOverRunCallback	オーバラン・エラーの発生に伴う処理を行います。

API 関数名	機能概要
UARTCn_Init	アシンクロナス・シリアル・インタフェース C（UARTC）の機能を制御するうえで必要となる初期化処理を行います。
UARTCn_UserInit	アシンクロナス・シリアル・インタフェース C（UARTC）に関するユーザ独自の初期化処理を行います。
UARTCn_Start	アシンクロナス・シリアル・インタフェース C（UARTC）を動作許可状態へと移行します。
UARTCn_Stop	アシンクロナス・シリアル・インタフェース C（UARTC）を動作禁止状態へと移行します。
UARTCn_SendData	データの UARTC*n* 送信を開始します。
UARTCn_ReceiveData	データの UARTC*n* 受信を開始します。
UARTCn_SendEndCallback	UARTC*n* の連続送信許可割り込み INTUC*n*T の発生に伴う処理を行います。
UARTCn_ReceiveEndCallback	UARTC*n* の受信終了割り込み INTUC*n*R の発生に伴う処理を行います。
UARTCn_ErrorCallback	UARTC*n* 受信エラー割り込み INTUC*n*R（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。
UARTCn_SoftOverRunCallback	オーバラン・エラーの発生に伴う処理を行います。
CSIBn_Init	3 線式可変長シリアル I/O B（CSIB）の機能を制御するうえで必要となる初期化処理を行います。
CSIBn_UserInit	3 線式可変長シリアル I/O B（CSIB）に関するユーザ独自の初期化処理を行います。
CSIBn_Start	3 線式可変長シリアル I/O B（CSIB）を動作許可状態へと移行します。
CSIBn_Stop	3 線式可変長シリアル I/O B（CSIB）を動作禁止状態へと移行します。
CSIBn_SendData	データの CSIB 送信を開始します。
CSIBn_ReceiveData	データの CSIB 受信を開始します。
CSIBn_SendReceiveData	データの CSIB 送受信を開始します。
CSIBn_SendEndCallback	CSIB*n* の受信終了割り込み INTCB*n*R，および CSIB*n* の連続送信書き込み許可割り込み INTCB*n*T の発生に伴う処理を行います。
CSIBn_ReceiveEndCallback	CSIB*n* の受信終了割り込み INTCB*n*R の発生に伴う処理を行います。
CSIBn_ErrorCallback	CSIB*n* の受信エラー割り込み INTCB*n*R（オーバラン・エラー）の発生に伴う処理を行います。
CSIEn_Init	3 線式可変長シリアル I/O E（CSIE）の機能を制御するうえで必要となる初期化処理を行います。
CSIEn_UserInit	3 線式可変長シリアル I/O E（CSIE）に関するユーザ独自の初期化処理を行います。
CSIEn_Start	3 線式可変長シリアル I/O E（CSIE）を動作許可状態へと移行します。
CSIEn_Stop	3 線式可変長シリアル I/O E（CSIE）を動作禁止状態へと移行します。
CSIEn_SendData	データの CSIE 送信を開始します。
CSIEn_ReceiveData	データの CSIE 受信を開始します。
CSIEn_SendReceiveData	データの CSIE 送受信を開始します。
CSIEn_SendEndCallback	CSIE*n* の送受信完了割り込み INTCE*n*T の発生に伴う処理を行います。

API 関数名	機能概要
CSIEn_ReceiveEndCallback	CSIE*n* の送受信完了割り込み INTCE*n*T の発生に伴う処理を行います。
CSIEn_ErrorCallback	CSIE*n* の CSIE*n*BUF オーバフロー割り込み INTCE*n*TIOF の発生に伴う処理を行います。
CSIFn_Init	3 線式可変長シリアル I/O F（CSIF）の機能を制御するうえで必要となる初期化処理を行います。
CSIFn_UserInit	3 線式可変長シリアル I/O F（CSIF）に関するユーザ独自の初期化処理を行います。
CSIFn_Start	3 線式可変長シリアル I/O F（CSIF）を動作許可状態へと移行します。
CSIFn_Stop	3 線式可変長シリアル I/O F（CSIF）を動作禁止状態へと移行します。
CSIFn_SendData	データの CSIF 送信を開始します。
CSIFn_ReceiveData	データの CSIF 受信を開始します。
CSIFn_SendReceiveData	データの CSIF 送受信を開始します。
CSIFn_SendEndCallback	CSIF*n* の送受信完了割り込み INTCF*n*T の発生に伴う処理を行います。
CSIFn_ReceiveEndCallback	CSIF*n* の送受信完了割り込み INTCF*n*T の発生に伴う処理を行います。
CSIFn_ErrorCallback	CSIF*n* の受信エラー割り込み INTCF*n*R（オーバラン・エラー）の発生に伴う処理を行います。
IIC0n_Init	IIC バスの機能を制御するうえで必要となる初期化処理を行います。
IIC0n_UserInit	IIC バスに関するユーザ独自の初期化処理を行います。
IIC0n_Stop	IIC0*n* 通信を終了します。
IIC0n_StopCondition	ストップ・コンディションを生成します。
IIC0n_MasterSendStart	データの IIC0*n* マスタ送信を開始します。
IIC0n_MasterReceiveStart	データの IIC0*n* マスタ受信を開始します。
IIC0n_MasterSendEndCallback	IIC0*n* マスタ送信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
IIC0n_MasterReceiveEndCallback	IIC0*n* マスタ受信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
IIC0n_MasterErrorCallback	IIC0*n* マスタ通信エラーの検出に伴う処理を行います。
IIC0n_SlaveSendStart	データの IIC0*n* スレーブ送信を開始します。
IIC0n_SlaveReceiveStart	データの IIC0*n* スレーブ受信を開始します。
IIC0n_SlaveSendEndCallback	IIC0*n* スレーブ送信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
IIC0n_SlaveReceiveEndCallback	IIC0*n* スレーブ受信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。
IIC0n_SlaveErrorCallback	IIC0*n* スレーブ通信エラーの検出に伴う処理を行います。
IIC0n_GetStopConditionCallback	ストップ・コンディションの検出に伴う処理を行います。

UARTA*n*_Init

アシンクロナス・シリアル・インタフェース A（UARTA）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_serial.c

［指定形式］

```
void    UARTAn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTA*n*_UserInit

アシンクロナス・シリアル・インタフェース A（UARTA）に関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，UARTAn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_serial_user.c

［指定形式］

```
void    UARTAn_UserInit ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTA*n*_Start

アシンクロナス・シリアル・インタフェース A（UARTA）を動作許可状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    UARTAn_Start ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTA*n*_Stop

アシンクロナス・シリアル・インタフェース A（UARTA）を動作禁止状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    UARTAn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTA*n*_SendData

データの UARTA*n* 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の UARTA*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. UARTA*n* 送信を行う際には，本 API 関数の呼び出し以前に UARTAn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTAn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正
MD_DATAEXISTS	異常終了 - UA*n*TX レジスタに“次に転送すべきデータ”が存在する（データの送信中）

UARTA*n*_ReceiveData

データの UARTA*n* 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の UARTA*n* 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

2. 実際の UARTA*n* 受信は，本 API 関数の呼び出し後，UARTAn_Start を呼び出すことにより開始されます。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTAn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

UARTA*n*_SendEndCallback

UARTA*n* の連続送信許可割り込み INTUA*n*T の発生に伴う処理を行います。

備考 本 API 関数は，UARTA*n* の連続送信許可割り込み INTUA*n*T に対応した割り込み処理 MD_INTUA*n*T のコールバック・ルーチン（UARTAn_SendData の引数 *txnum* で指定された総数の UARTA*n* 送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTAn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTA*n*_ReceiveEndCallback

UARTA*n* の受信終了割り込み INTUA*n*R の発生に伴う処理を行います。

備考 本 API 関数は，UARTA*n* の受信終了割り込み INTUA*n*R に対応した割り込み処理 MD_INTUA*n*R のコールバック・ルーチン（UARTAn_ReceiveData の引数 *rxnum* で指定された総数の UARTA*n* 受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTAn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTA*n*_ErrorCallback

UARTA*n* 受信エラー割り込み INTUA*n*R（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。

備考 本 API 関数は，UARTA*n* 受信エラー割り込み INTUA*n*R に対応した割り込み処理 MD_INTUA*n*R のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTAn_ErrorCallback ( UCHAR err_type );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *err_type*;	エラー割り込みの発生要因 00000*xx*1B：オーバラン・エラー 00000*x*1*x*B：フレーミング・エラー 000001*xx*B：パリティ・エラー

[戻り値]

なし

UARTA*n*_SoftOverRunCallback

オーバラン・エラーの発生に伴う処理を行います。

備考 本 API 関数は，UARTA*n* の受信エラー割り込み INTUA*n*R に対応した割り込み処理 MD_INTUA*n*R のコールバック・ルーチン（UARTAn_ReceiveData の引数 *rxnum* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTAn_SoftOverRunCallback ( UCHAR rx_data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *rx_data*;	受信したデータ

[戻り値]

なし

UARTB*n*_Init

アシンクロナス・シリアル・インタフェース A（UARTB）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_serial.c

［指定形式］

```
void    UARTBn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTB*n*_UserInit

アシンクロナス・シリアル・インタフェースB（UARTB）に関するユーザ独自の初期化処理を行います。

備考　本API関数は，UARTBn_Initのコールバック・ルーチンとして呼び出されます。

［所属］

CG_serial_user.c

［指定形式］

```
void    UARTBn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTB*n*_Start

アシンクロナス・シリアル・インタフェースB（UARTB）を動作許可状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    UARTBn_Start ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTB*n*_Stop

アシンクロナス・シリアル・インタフェースB（UARTB）を動作禁止状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    UARTBn_Stop ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTB*n*_SendData

データの UARTB*n* 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の UARTB*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. UARTB*n* 送信（シングル・モード）を行う際には，本 API 関数の呼び出し以前に UARTBn_Start を呼び出す必要があります。

3. UARTB*n* 送信（FIFO モード）を行う際には，本 API 関数の呼び出し後に UARTBn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTBn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正 - *txnum* が送信 FIFO トリガ数の倍数でない
MD_DATAEXISTS	異常終了 - UB*n*TX レジスタに“次に転送すべきデータ”が存在する（データの送信中）

UARTB*n*_ReceiveData

データの UARTB*n* 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の UARTB*n* 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

2. 実際の UARTB*n* 受信は，本 API 関数の呼び出し後，UARTBn_Start を呼び出すことにより開始されます。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTBn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正 - *rxnum* が受信 FIFO トリガ数の倍数でない

UARTB*n*_SendEndCallback

UARTB*n* の送信許可割り込み INTUB*n*TIT，および FIFO 送信完了割り込み INTUB*n*TIF の発生に伴う処理を行います。

備考 本 API 関数は，UARTB*n*（シングル・モード）の送信許可割り込み INTUB*n*TIT に対応した割り込み処理 MD_INTUB*n*TIT，および UARTB*n*（FIFO モード）の FIFO 送信完了割り込み INTUB*n*TIF に対応した割り込み処理 MD_INTUB*n*TIF のコールバック・ルーチン（UARTBn_SendData の引数 *txnum* で指定された総数の UARTB*n* 送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTBn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTB*n*_ReceiveEndCallback

UARTB*n* の受信完了割り込み INTUB*n*TIR の発生に伴う処理を行います。

備考 本 API 関数は，UARTB*n* の受信完了割り込み INTUB*n*TIR に対応した割り込み処理 MD_INTUB*n*TIR のコールバック・ルーチン（UARTBn_ReceiveData の引数 *rxnum* で指定された総数の UARTB*n* 受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTBn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTB*n*_SingleErrorCallback

受信エラー割り込み INTUB*n*TIRE（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。

備考 本 API 関数は，UARTB*n*（シングル・モード）の受信エラー割り込み INTUBA*n*TIRE に対応した割り込み処理 MD_INTUB*n*TIRE のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTBn_SingleErrorCallback ( UCHAR err_type );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *err_type*;	受信エラー割り込みの発生要因 00000*xx*1B：オーバラン・エラー 00000*x*1*x*B：フレーミング・エラー 000001*xx*B：パリティ・エラー

[戻り値]

なし

UARTB*n*_FIFOErrorCallback

受信エラー割り込み INTUB*n*TIRE（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。

備考 本 API 関数は，UARTB*n*（FIFO モード）の受信エラー割り込み INTUBA*n*TIRE に対応した割り込み処理 MD_INTUB*n*TIRE のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTBn_FIFOErrorCallback ( UCHAR err_type1, UCHAR err_type2 );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *err_type1*;	受信エラー割り込みの発生要因 00001000B：オーバラン・エラー
O	UCHAR *err_type2*;	受信エラー割り込みの発生要因 000000*x*1B：フレーミング・エラー 0000001*x*B：パリティ・エラー

[戻り値]

なし

UARTB*n*_TimeoutErrorCallback

受信タイムアウト割り込み INTUB*n*TITO の発生に伴う処理を行います。

備考 本 API 関数は，UARTB*n*（FIFO モード）の受信タイムアウト割り込み INTUBA*n*TITO に対応した割り込み処理 MD_INTUB*n*TITO のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTBn_TimeoutErrorCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTB*n*_SoftOverRunCallback

オーバラン・エラーの発生に伴う処理を行います。

備考 本 API 関数は，UARTB*n* の受信エラー割り込み INTUB*n*TIRE に対応した割り込み処理 MD_INTUB*n*TIRE のコールバック・ルーチン（UARTBn_ReceiveData の引数 *rxnum* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTBn_SoftOverRunCallback ( UCHAR rx_data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *rx_data*;	受信したデータ

[戻り値]

なし

UARTC*n*_Init

アシンクロナス・シリアル・インタフェースC（UARTC）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_serial.c

［指定形式］

```
void    UARTCn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTC*n*_UserInit

アシンクロナス・シリアル・インタフェース（UARTC）に関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，UARTCn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_serial_user.c

［指定形式］

```
void    UARTCn_UserInit ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTC*n*_Start

アシンクロナス・シリアル・インタフェース C（UARTC）を動作許可状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    UARTCn_Start ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTC*n*_Stop

アシンクロナス・シリアル・インタフェース C（UARTC）を動作禁止状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    UARTCn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

UARTC*n*_SendData

データの UARTC*n* 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の UARTC*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. UARTC*n* 送信を行う際には，本 API 関数の呼び出し以前に UARTCn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTCn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正
MD_DATAEXISTS	異常終了 - UC*n*TX レジスタに“次に転送すべきデータ”が存在する（データの送信中）

UARTC*n*_ReceiveData

データの UARTC*n* 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の UARTC*n* 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

2. 実際の UARTC*n* 受信は，本 API 関数の呼び出し後，UARTCn_Start を呼び出すことにより開始されます。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTCn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

UARTC*n*_SendEndCallback

UARTC*n* の連続送信許可割り込み INTUC*n*T の発生に伴う処理を行います。

備考 本 API 関数は，UARTC*n* の連続送信許可割り込み INTUC*n*T に対応した割り込み処理 MD_INTUC*n*T のコールバック・ルーチン（UARTCn_SendData の引数 *txnum* で指定された総数の UARTC*n* 送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTCn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTC*n*_ReceiveEndCallback

UARTC*n* の受信終了割り込み INTUC*n*R の発生に伴う処理を行います。

備考 本 API 関数は，UARTC*n* の受信終了割り込み INTUC*n*R に対応した割り込み処理 MD_INTUC*n*R のコールバック・ルーチン（UARTCn_ReceiveData の引数 *rxnum* で指定された総数の UARTC*n* 受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTCn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

UARTC*n*_ErrorCallback

UARTC*n* 受信エラー割り込み INTUC*n*R（オーバラン・エラー，フレーミング・エラー，パリティ・エラー）の発生に伴う処理を行います。

備考 本 API 関数は，UARTC*n* 受信エラー割り込み INTUC*n*R に対応した割り込み処理 MD_INTUC*n*R のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTCn_ErrorCallback ( UCHAR err_type );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *err_type*;	エラー割り込みの発生要因 00000*xx*1B：オーバラン・エラー 00000*x*1*x*B：フレーミング・エラー 000001*xx*B：パリティ・エラー

[戻り値]

なし

UARTC*n*_SoftOverRunCallback

オーバラン・エラーの発生に伴う処理を行います。

備考 本 API 関数は，UARTC*n* の受信エラー割り込み INTUC*n*R に対応した割り込み処理 MD_INTUC*n*R のコールバック・ルーチン（UARTCn_ReceiveData の引数 *rxnum* で指定された数以上のデータを受信した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTCn_SoftOverRunCallback ( UCHAR rx_data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *rx_data*;	受信したデータ

[戻り値]

なし

CSIB*n*_Init

3 線式可変長シリアル I/O B（CSIB）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_serial.c

［指定形式］

```
void    CSIBn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIB*n*_UserInit

3 線式可変長シリアル I/O B（CSIB）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，CSIBn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_serial_user.c

［指定形式］

```
void    CSIBn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIB*n*_Start

3 線式可変長シリアル I/O B（CSIB）を動作許可状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    CSIBn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIB*n*_Stop

3 線式可変長シリアル I/O B（CSIB）を動作禁止状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    CSIBn_Stop ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIB*n*_SendData

データの CSIB*n* 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の CSIB*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. CSIB*n* 送信を行う際には，本 API 関数の呼び出し以前に CSIBn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   CSIBn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIB*n*_ReceiveData

データの CSIB*n* 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の CSIB*n* 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

2. CSIB*n* 受信を行う際には，本 API 関数の呼び出し以前に CSIBn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIBn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIB*n*_SendReceiveData

データの CSIB*n* 送受信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の CSIB*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. 本 API 関数では，1 バイト単位の CSIB*n* 受信を引数 *txnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

3. CSIB*n* 送受信を行う際には，本 API 関数の呼び出し以前に CSIBn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIBn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *txbuf;	受信したデータを格納するバッファへのポインタ
I	USHORT txnum;	送受信するデータの総数
I	UCHAR *rxbuf;	送信するデータを格納したバッファへのポインタ

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIB*n*_SendEndCallback

CSIB*n* の受信終了割り込み INTCB*n*R，および CSIB*n* の連続書き込み許可割り込み INTCB*n*T の発生に伴う処理を行います。

備考 本 API 関数は，CSIB*n* の受信終了割り込み INTCB*n*R に対応した割り込み処理 MD_INTCB*n*R，および CSIB*n* の連続書き込み許可割り込み INTCB*n*T に対応した割り込み処理 MD_INTCB*n*T のコールバック・ルーチン（CSIBn_SendData，または CSIBn_SendReceiveData の引数 *txnum* で指定された総数の CSIB*n* 送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIBn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIB*n*_ReceiveEndCallback

CSIB*n* の受信終了割り込み INTCB*n*R の発生に伴う処理を行います。

備考 本 API 関数は，CSIB*n* の受信終了割り込み INTCB*n*R に対応した割り込み処理 MD_INTCB*n*R のコールバック・ルーチン（CSIBn_ReceiveData，または CSIBn_SendReceiveData の引数 *rxnum* で指定された総数の CSIB*n* 受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIBn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIB*n*_ErrorCallback

CSIB*n* の受信エラー割り込み INTCB*n*R（オーバラン・エラー）の発生に伴う処理を行います。

備考 本 API 関数は，CSIB*n* の受信エラー割り込み INTCB*n*R に対応した割り込み処理 MD_INTCB*n*R のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    CSIBn_ErrorCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIE*n*_Init

3 線式可変長シリアル I/O E（CSIE）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_serial.c

［指定形式］

```
void    CSIEn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIE*n*_UserInit

3 線式可変長シリアル I/O E（CSIE）に関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，CSIEn_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIEn_UserInit ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIE*n*_Start

3 線式可変長シリアル I/O E（CSIE）を動作許可状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    CSIEn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIE*n*_Stop

3 線式可変長シリアル I/O E（CSIE）を動作禁止状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    CSIEn_Stop ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIE*n*_SendData

データの CSIE*n* 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の CSIE*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. CSIE*n* 送信を行う際には，本 API 関数の呼び出し以前に CSIEn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   CSIEn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIE*n*_ReceiveData

データの CSIE*n* 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の CSIE*n* 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

2. CSIE*n* 受信を行う際には，本 API 関数の呼び出し以前に CSIEn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIEn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIE*n*_SendReceiveData

データの CSIE*n* 送受信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の CSIE*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. 本 API 関数では，1 バイト単位の CSIE*n* 受信を引数 *txnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

3. CSIE*n* 送受信を行う際には，本 API 関数の呼び出し以前に CSIEn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIEn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *txbuf;	受信したデータを格納するバッファへのポインタ
I	USHORT txnum;	送受信するデータの総数
I	UCHAR *rxbuf;	送信するデータを格納したバッファへのポインタ

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIE*n*_SendEndCallback

CSIE*n* の送受信完了割り込み INTCE*n*T の発生に伴う処理を行います。

備考 本 API 関数は，CSIE*n* の送受信完了割り込み INTCE*n*T に対応した割り込み処理 MD_INTCE*n*T のコールバック・ルーチン（CSIEn_SendData，または CSIEn_SendReceiveData の引数 *txnum* で指定された総数の CSIE*n* 送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIEn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIE*n*_ReceiveEndCallback

CSIE*n* の送受信完了割り込み INTCE*n*T の発生に伴う処理を行います。

備考 本 API 関数は，CSIE*n* の送受信完了割り込み INTCE*n*T に対応した割り込み処理 MD_INTCE*n*T のコールバック・ルーチン（CSIEn_ReceiveData，または CSIEn_SendReceiveData の引数 *rxnum* で指定された総数の CSIE*n* 受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIEn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIE*n*_ErrorCallback

CSIE*n* の CSIE*n*BUF オーバフロー割り込み INTCE*n*TIOF の発生に伴う処理を行います。

備考 本 API 関数は，CSIE*n* の CSIE*n*BUF オーバフロー割り込み INTCE*n*TIOF に対応した割り込み処理 MD_INTCE*n*TIOF のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    CSIEn_ErrorCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIF*n*_Init

3 線式可変長シリアル I/O F（CSIF）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_serial.c

［指定形式］

```
void    CSIFn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIF*n*_UserInit

3 線式可変長シリアル I/O F（CSIF）に関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，CSIFn_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIFn_UserInit ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIF*n*_Start

3 線式可変長シリアル I/O F（CSIF）を動作許可状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    CSIFn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIF*n*_Stop

3 線式可変長シリアル I/O F（CSIF）を動作禁止状態へと移行します。

［所属］

CG_serial.c

［指定形式］

```
void    CSIFn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

CSIF*n*_SendData

データの CSIF*n* 送信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の CSIF*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. CSIF*n* 送信を行う際には，本 API 関数の呼び出し以前に CSIFn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   CSIFn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIF*n*_ReceiveData

データの CSIF*n* 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の CSIF*n* 受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

2. CSIF*n* 受信を行う際には，本 API 関数の呼び出し以前に CSIFn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIFn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIF*n*_SendReceiveData

データの CSIF*n* 送受信を開始します。

備考 1. 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の CSIF*n* 送信を引数 *txnum* で指定された回数だけ繰り返し行います。

2. 本 API 関数では，1 バイト単位の CSIF*n* 受信を引数 *txnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

3. CSIF*n* 送受信を行う際には，本 API 関数の呼び出し以前に CSIFn_Start を呼び出す必要があります。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   CSIFn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *txbuf;	受信したデータを格納するバッファへのポインタ
I	USHORT txnum;	送受信するデータの総数
I	UCHAR *rxbuf;	送信するデータを格納したバッファへのポインタ

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

CSIF*n*_SendEndCallback

CSIF*n* の送受信完了割り込み INTCF*n*T の発生に伴う処理を行います。

備考 本 API 関数は，CSIF*n* の送受信完了割り込み INTCF*n*T に対応した割り込み処理 MD_INTCF*n*T のコールバック・ルーチン（CSIFn_SendData，または CSIFn_SendReceiveData の引数 *txnum* で指定された総数の CSIF*n* 送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIFn_SendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIF*n*_ReceiveEndCallback

CSIF*n* の送受信完了割り込み INTCF*n*T の発生に伴う処理を行います。

備考 本 API 関数は，CSIF*n* の送受信完了割り込み INTCF*n*T に対応した割り込み処理 MD_INTCF*n*T のコールバック・ルーチン（CSIFn_ReceiveData，または CSIFn_SendReceiveData の引数 *rxnum* で指定された総数の CSIF*n* 受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    CSIFn_ReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSIF*n*_ErrorCallback

CSIF*n* の受信エラー割り込み INTCF*n*R（オーバラン・エラー）の発生に伴う処理を行います。

備考 本 API 関数は，CSIF*n* の受信エラー割り込み INTCF*n*R に対応した割り込み処理 MD_INTCF*n*R のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    CSIFn_ErrorCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_Init

IIC バスの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    IIC0n_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_UserInit

IIC バスに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，IIC0n_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC0n_UserInit ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_Stop

IIC0*n* 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    IIC0n_Stop ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_StopCondition

ストップ・コンディションを生成します。

［所属］

CG_serial.c

［指定形式］

```
void    IIC0n_StopCondition ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

IIC0*n*_MasterSendStart

データの IIC0*n* マスタ送信を開始します。

備考 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の IIC0*n* マスタ送信を引数 *txnum* で指定された回数だけ繰り返し行います。

[所属]

CG_serial.c

[指定形式]

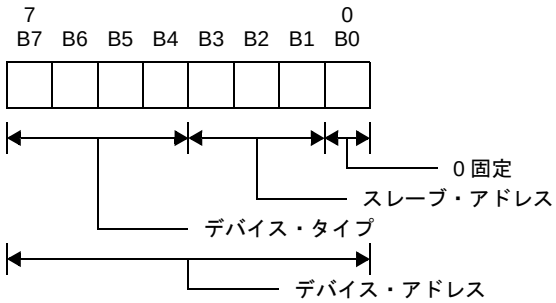
```
#include    “CG_macrodriver.h“
MD_STATUS   IIC0n_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *adr*;	デバイス・アドレス
I	UCHAR **txbuf*;	送信するデータを格納したバッファへのポインタ
I	USHORT *txnum*;	送信するデータの総数
I	UCHAR *wait*;	スタート・コンディションのセットアップ時間

備考 デバイス・アドレス *adr* は，デバイス・タイプとスレーブ・アドレスから構成されます。



[戻り値]

マクロ	説明
MD_OK	正常終了

マクロ	説明
MD_ERROR	異常終了

IIC0*n*_MasterReceiveStart

データの IIC0*n* マスタ受信を開始します。

備考 本 API 関数では，1 バイト単位の IIC0*n* マスタ受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

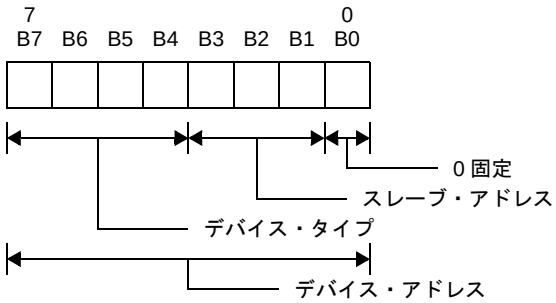
```
#include    "CG_macrodriver.h"
MD_STATUS   IIC0n_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *adr*;	デバイス・アドレス
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数
I	UCHAR *wait*;	スタート・コンディションのセットアップ時間

備考 デバイス・アドレス *adr* は，デバイス・タイプとスレーブ・アドレスから構成されます。



[戻り値]

マクロ	説明
MD_OK	正常終了

マクロ	説明
MD_ERROR	異常終了

IIC0*n*_MasterSendEndCallback

IIC0*n* マスタ送信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC0*n* マスタ送信の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（IIC0n_MasterSendStart の引数 *txnum* で指定された総数の IIC0*n* マスタ送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC0n_MasterSendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_MasterReceiveEndCallback

IIC0*n* マスタ受信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC0*n* マスタ受信の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（IIC0n_MasterReceiveStart の引数 *rxnum* で指定された総数の IIC0*n* マスタ受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC0n_MasterReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_MasterErrorCallback

IIC0*n* マスタ通信エラーの検出に伴う処理を行います。

備考 本 API 関数は，IIC0*n* の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（IIC0*n* マスタ通信エラーを検出した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IIC0n_MasterErrorCallback ( MD_STATUS flag );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	MD_STATUS *flag*;	IIC0*n* マスタ通信エラーの発生要因 MD_SPT： IIC0*n* マスタ通信中にストップ・コンディションを検出 MD_NACK：アクノリッジを未検出

[戻り値]

なし

IIC0*n*_SlaveSendStart

データの IIC0*n* スレーブ送信を開始します。

備考 本 API 関数では，引数 *txbuf* で指定されたバッファから 1 バイト単位の IIC0*n* スレーブ送信を引数 *txnum* で指定された回数だけ繰り返し行います。

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
void    IIC0n_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *txbuf;	送信するデータを格納したバッファへのポインタ
I	USHORT txnum;	送信するデータの総数

[戻り値]

なし

IIC0*n*_SlaveReceiveStart

データの IIC0*n* スレーブ受信を開始します。

備考 本 API 関数では，1 バイト単位の IIC0*n* スレーブ受信を引数 *rxnum* で指定された回数だけ繰り返し行い，引数 *rxbuf* で指定されたバッファに格納します。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IIC0n_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR **rxbuf*;	受信したデータを格納するバッファへのポインタ
I	USHORT *rxnum*;	受信するデータの総数

[戻り値]

なし

IIC0*n*_SlaveSendEndCallback

IIC0*n* スレーブ送信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC0*n* スレーブ送信の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（IIC0n_SlaveSendStart の引数 *txnum* で指定された総数の IIC0*n* スレーブ送信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC0n_SlaveSendEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_SlaveReceiveEndCallback

IIC0*n* スレーブ受信の転送終了割り込み INTIIC*n* の発生に伴う処理を行います。

備考 本 API 関数は，IIC0*n* スレーブ受信の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（IIC0n_SlaveReceiveStart の引数 *rxnum* で指定された総数の IIC0*n* スレーブ受信が完了した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC0n_SlaveReceiveEndCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC0*n*_SlaveErrorCallback

IIC0*n* スレーブ通信エラーの検出に伴う処理を行います。

備考 本 API 関数は，IIC0*n* の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（IIC0*n* スレーブ通信エラーを検出した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    IIC0n_SlaveErrorCallback ( MD_STATUS flag );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	MD_STATUS *flag*;	IIC0*n* マスタ通信エラーの発生要因 MD_ERROR： アドレスの不一致を検出 MD_NACK： アクノリッジを未検出

[戻り値]

なし

IIC0*n*_GetStopConditionCallback

ストップ・コンディションの検出に伴う処理を行います。

備考 本 API 関数は，IIC0*n* の転送終了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチン（ストップ・コンディションを検出した際の処理）として呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IIC0n_GetStopConditionCallback ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

3. 3. 6 A/D コンバータ

以下に，Applilet3 が A/D コンバータ用として出力する API 関数の一覧を示します。

表 3—7 A/D コンバータ用 API 関数

API 関数名	機能概要
AD_Init	A/D コンバータの機能を制御するうえで必要となる初期化処理を行います。
AD_UserInit	A/D コンバータに関するユーザ独自の初期化処理を行います。
AD_Start	A/D 変換を開始します。
AD_Stop	A/D 変換を終了します。
AD_SelectADChannel	A/D 変換するアナログ電圧の入力端子を設定します。
AD_SetPFTCondition	パワー・フェイル比較モードで動作する際の情報（比較値，A/D 変換終了割り込み INTAD の発生要因）を設定します。
AD_Read	A/D 変換結果（10 ビット）を読み出します。
AD_ReadByte	A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を読み出します。

AD_Init

A/D コンバータの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_ad.c

[指定形式]

```
void    AD_Init ( void );
```

[引数]

なし

[戻り値]

なし

AD_UserInit

A/D コンバータに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，AD_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_ad_user.c

［指定形式］

```
void    AD_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

AD_Start

A/D 変換を開始します。

備考 本 API 関数の呼び出しから AD_Stop が呼び出されるまでの間，A/D 変換処理が繰り返し実行されます。

[所属]

CG_ad.c

[指定形式]

```
void    AD_Start ( void );
```

[引数]

なし

[戻り値]

なし

[使用例]

以下に，アナログ電圧を A/D 変換する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_ad.h"
BOOL    gFlag;                          /* A/D 変換完了フラグ */
void main ( void ) {
    USHORT  buffer = 0;
    int     wait = 100;
    gFlag = 1;                          /* A/D 変換完了フラグの初期化 */
    ……
    AD_Start ();                        /* A/D 変換の開始 */
    while ( gFlag );                    /* 割り込み INTAD の発生待ち */
    AD_Read ( &buffer );                /* A/D 変換結果の読み出し */
    AD_Stop ();                         /* A/D 変換の終了 */
    ……
}
```

【CG_ad_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* A/D 変換完了フラグ */
__interrupt void MD_INTAD ( void ) {    /* 割り込み INTAD 発生時の割り込み処理 */
    gFlag = 0;                          /* A/D 変換完了フラグの設定 */
}
```

AD_Stop

A/D 変換を終了します。

［所属］

CG_ad.c

［指定形式］

```
void    AD_Stop ( void );
```

［引数］

なし

［戻り値］

なし

AD_SelectADChannel

A/D 変換するアナログ電圧の入力端子を設定します。

[所属]

CG_ad.c

[指定形式]

```
#include    "CG_ad.h"
MD_STATUS   AD_SelectADChannel ( enum ADChannel channel );
```

[引数]

I/O	引数	説明
I	enum ADChannel *channel*;	アナログ電圧の入力端子 ADCHANNEL*n*： 入力端子

備考 アナログ電圧の入力端子 ADCHANNEL*n* についての詳細は，ヘッダ・ファイル CG_ad.h を参照してください。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

AD_SetPFTCondition

パワー・フェイル比較モードで動作する際の情報（比較値，A/D 変換終了割り込み INTAD の発生要因）を設定します。

［所属］

CG_ad.c

［指定形式］

```
#include    "CG_ad.h"
MD_STATUS   AD_SetPFTCondition ( UCHAR pftvalue, enum ADPFTMode mode );
```

［引数］

I/O	引数	説明
I	UCHAR *pftvalue*;	比較値
I	enum ADPFTMode *mode*;	A/D 変換終了割り込み INTAD の発生要因 EACHEND： A/D 変換の終了時に INTAD を発生 PFTHIGHER： ADA0CR*n*H ≧ ADA0PFT の際に INTAD を発生 PFTLOWER： ADA0CR*n*H ＜ ADA0PFT の際に INTAD を発生

備考 引数 *pftvalue* に指定された値は，引数 *mode* に PFTHIGHER，または PFTLOWER が指定された際，パワー・フェイル比較しきい値レジスタ（ADA0PFT）に設定され，A/D 変換結果レジスタ *n*H（ADA0CR*n*H）との比較に使用されます。

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

AD_Read

A/D 変換結果（10 ビット）を読み出します。

［所属］

CG_ad.c

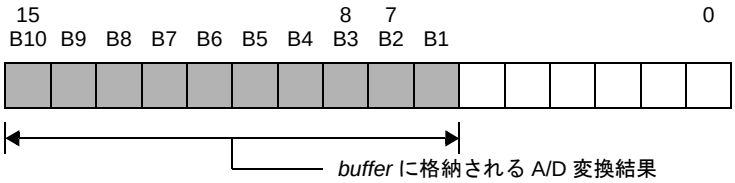
［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   AD_Read ( USHORT *buffer );
```

［引数］

I/O	引数	説明
O	USHORT *buffer;	読み出した A/D 変換結果（10 ビット）を格納する領域へのポインタ

備考 以下に，*buffer* に格納される A/D 変換結果を示します。



［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了

AD_ReadByte

A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を読み出します。

［所属］

CG_ad.c

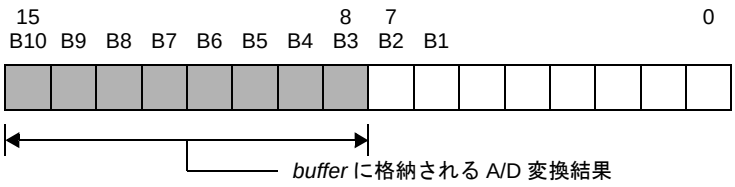
［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   AD_ReadByte ( UCHAR *buffer );
```

［引数］

I/O	引数	説明
O	UCHAR *buffer;	読み出した A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を格納する領域へのポインタ

備考 以下に，*buffer* に格納される A/D 変換結果を示します。



［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了

3. 3. 7 D/A コンバータ

以下に，Applilet3 が D/A コンバータ用として出力する API 関数の一覧を示します。

表 3—8 D/A コンバータ用 API 関数

API 関数名	機能概要
DAn_Init	D/A コンバータの機能を制御するうえで必要となる初期化処理を行います。
DAn_UserInit	D/A コンバータに関するユーザ独自の初期化処理を行います。
DAn_Start	D/A 変換を開始します。
DAn_Stop	D/A 変換を終了します。
DAn_SetValue	ANO*n* 端子に出力するアナログ電圧値を設定します。

DA*n*_Init

D/A コンバータの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_da.c

[指定形式]

```
void    DAn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

DA*n*_UserInit

D/A コンバータに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，DAn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_da_user.c

［指定形式］

```
void    DAn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

DA*n*_Start

D/A 変換を開始します。

［所属］

CG_da.c

［指定形式］

```
void    DAn_Start ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

DA*n*_Stop

D/A 変換を終了します。

［所属］

CG_da.c

［指定形式］

```
void    DAn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

DA*n*_SetValue

ANO*n* 端子に出力するアナログ電圧値を設定します。

[所属]

CG_da.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    DAn_SetValue ( UCHAR value );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *value*;	アナログ電圧値（0x0 ～ 0xff）

[戻り値]

なし

[使用例]

以下に，チャネル 0，およびチャネル 1 に“アナログ電圧値”を設定する際の例を示します。

【CG_main.c】

```
void main ( void ) {
    ……
    DA0_Start ();                       /* D/A 変換の開始 */
    DA1_Start ();                       /* D/A 変換の開始 */
    ……
    DA0_SetValue ( 0x7f );              /* アナログ電圧値の設定 */
    ……
}
```

【CG_tau_user.c】

```
#include    "CG_macrodriver.h"
```

```
UCHAR   gValue = 0;
__interrupt void MD_INTTP2CC0 ( void ) {    /* 割り込み INTTP2CC0 発生時の割り込み処理 */
    DA1_SetValue ( gValue++ );              /* アナログ電圧値の設定 */
}
```

3. 3. 8 タ イ マ

以下に，Applilet3 がタイマ用として出力する API 関数の一覧を示します。

表 3—9 タイマ用 API 関数

API 関数名	機能概要
TMPn_Init	16 ビット・タイマ／イベント・カウンタ P（TMP）の機能を制御するうえで必要となる初期化処理を行います。
TMPn_UserInit	16 ビット・タイマ／イベント・カウンタ P（TMP）に関するユーザ独自の初期化処理を行います。
TMPn_Start	16 ビット・タイマ／イベント・カウンタ P（TMP）のカウントを開始します。
TMPn_Stop	16 ビット・タイマ／イベント・カウンタ P（TMP）のカウントを終了します。
TMPn_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ P（TMP）のカウント値を変更します。
TMPn_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ P（TMP）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
TMPn_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ P（TMP）がキャプチャした値を読み出します。
TMPn_ChangeDuty	PWM 信号のデューティ比を変更します。
TMPn_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
TMQ0_Init	16 ビット・タイマ／イベント・カウンタ Q（TMQ）の機能を制御するうえで必要となる初期化処理を行います。
TMQ0_UserInit	16 ビット・タイマ／イベント・カウンタ Q（TMQ）に関するユーザ独自の初期化処理を行います。
TMQ0_Start	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のカウントを開始します。
TMQ0_Stop	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のカウントを終了します。
TMQ0_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のカウント値を変更します。
TMQ0_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ Q（TMQ）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
TMQ0_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ Q（TMQ）がキャプチャした値を読み出します。
TMQ0_ChangeDuty	PWM 信号のデューティ比を変更します。
TMQ0_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
TAAn_Init	16 ビット・タイマ／イベント・カウンタ AA（TAA）の機能を制御するうえで必要となる初期化処理を行います。
TAAn_UserInit	16 ビット・タイマ／イベント・カウンタ AA（TAA）に関するユーザ独自の初期化処理を行います。

API 関数名	機能概要
TAAn_Start	16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウントを開始します。
TAAn_Stop	16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウントを終了します。
TAAn_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウント値を変更します。
TAAn_ControlOutputToggle	16 ビット・タイマ／イベント・カウンタ AA（TAA）のトグル制御を変更します。
TAAn_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ AA（TAA）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
TAAn_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ AA（TAA）がキャプチャした値を読み出します。
TAAn_ChangeDuty	PWM 信号のデューティ比を変更します。
TAAn_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
TABn_Init	16 ビット・タイマ／イベント・カウンタ AB（TAB）の機能を制御するうえで必要となる初期化処理を行います。
TABn_UserInit	16 ビット・タイマ／イベント・カウンタ AB（TAB）に関するユーザ独自の初期化処理を行います。
TABn_Start	16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウントを開始します。
TABn_Stop	16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウントを終了します。
TABn_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウント値を変更します。
TABn_ControlOutputToggle	16 ビット・タイマ／イベント・カウンタ AB（TAB）のトグル制御を変更します。
TABn_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ AB（TAB）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
TABn_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ AB（TAB）がキャプチャした値を読み出します。
TABn_ChangeDuty	PWM 信号のデューティ比を変更します。
TABn_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
TMT0_Init	16 ビット・タイマ／イベント・カウンタ T（TMT）の機能を制御するうえで必要となる初期化処理を行います。
TMT0_UserInit	16 ビット・タイマ／イベント・カウンタ T（TMT）に関するユーザ独自の初期化処理を行います。
TMT0_Start	16 ビット・タイマ／イベント・カウンタ T（TMT）のカウントを開始します。
TMT0_Stop	16 ビット・タイマ／イベント・カウンタ T（TMT）のカウントを終了します。

API 関数名	機能概要
TMT0_ChangeTimerCondition	16 ビット・タイマ／イベント・カウンタ T（TMT）のカウント値を変更します。
TMT0_GetPulseWidth	16 ビット・タイマ／イベント・カウンタ T（TMT）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。
TMT0_GetFreeRunningValue	16 ビット・タイマ／イベント・カウンタ T（TMT）がキャプチャした値を読み出します。
TMT0_ChangeDuty	PWM 信号のデューティ比を変更します。
TMT0_SoftwareTriggerOn	タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。
TMT0_EnableHold	16 ビット・タイマ／イベント・カウンタ T（TMT）のエンコーダ・カウンタ制御を保持動作へと変更します。
TMT0_DisableHold	16 ビット・タイマ／イベント・カウンタ T（TMT）のエンコーダ・カウンタ制御を通常動作へと変更します。
TMT0_ChangeCountValue	16 ビット・タイマ／イベント・カウンタ T（TMT）の初期カウント値を変更します。
TMMn_Init	16 ビット・インターバル・タイマ M（TMM）の機能を制御するうえで必要となる初期化処理を行います。
TMMn_UserInit	16 ビット・インターバル・タイマ M（TMM）に関するユーザ独自の初期化処理を行います。
TMMn_Start	16 ビット・インターバル・タイマ M（TMM）のカウントを開始します。
TMMn_Stop	16 ビット・インターバル・タイマ M（TMM）のカウントを終了します。
TMMn_ChangeTimerCondition	16 ビット・インターバル・タイマ M（TMM）のカウント値を変更します。

TMP*n*_Init

16 ビット・タイマ／イベント・カウンタ P（TMP）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_timer.c

［指定形式］

```
void    TMPn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMP*n*_UserInit

16 ビット・タイマ／イベント・カウンタ P（TMP）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，TMPn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_timer_user.c

［指定形式］

```
void    TMPn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMP*n*_Start

16 ビット・タイマ／イベント・カウンタ P（TMP）のカウントを開始します。

備考 本 API 関数を呼び出してからカウント処理を開始するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TMPn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMP*n*_Stop

16 ビット・タイマ／イベント・カウンタ P（TMP）のカウントを終了します。

備考　本 API 関数を呼び出してからカウント処理を終了するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TMPn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMP*n*_ChangeTimerCondition

16 ビット・タイマ／イベント・カウンタ P（TMP）のカウント値を変更します。

備考 引数 *arrar_reg* に指定された値は，TMP*n* キャプチャ／コンペア・レジスタ *m*（TP*n*CCR*m*）に設定されます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TMPn_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	USHORT **array_reg*;	カウント値（0x0 ～ 0xffff）を格納した領域へのポインタ
I	UCHAR *array_num*;	変更対象レジスタ 1： TP*n*CCR0 2： TP*n*CCR0，TP*n*CCR1

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

［使用例］

以下に，インターバル時間を半分に変更する際の例を示します。

なお，下記は，チャネル 0 がインターバル・タイマ用として選択された場合の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flag_finish = 1;
```

```
    USHORT  array_reg = TMP_TP0CCR0_VALUE >> 1; /* TMP_TP0CCR0_VALUE：現在のインターバル時間 */
    UCHAR   array_num = 1;
    .........
    TMP0_Start ();                                          /* カウントの開始 */
    while ( flag_finish );                                  /* タイムアップの確認 */
    .........
    TMP0_ChanneTimerCondition ( &array_reg, array_num );    /* カウント値の変更 */
   .........
}
```

TMP*n*_GetPulseWidth

16 ビット・タイマ／イベント・カウンタ P（TMP）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。

備考 1. 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ P（TMP）をパルス幅測定用として使用している場合に限られます。

2. パルス幅の計測中にオーバフロー（2 回以上）が発生した場合，正常なパルス幅を読み出すことはできません。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TMPn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	ULONG *activewith;	読み出したハイ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ
O	ULONG *inactivewidth;	読み出したロウ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

[戻り値]

なし

TMP*n*_GetFreeRunningValue

16 ビット・タイマ／イベント・カウンタ P（TMP）がキャプチャした値を読み出します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ P（TMP）をフリー・ランニング・タイマ用として使用し，TMP*n* キャプチャ／コンペア・レジスタ *m*（TP*n*CCR*m*）がキャプチャ・レジスタとして選択されている場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMPn_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	`ULONG *count;`	読み出した幅を格納する領域へのポインタ
I	`enum TMChannel channel;`	読み出し対象チャネル TMCHANNEL0： チャネル 0（TP*n*CCR0） TMCHANNEL1： チャネル 1（TP*n*CCR1）

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TMP*n*_ChangeDuty

PWM 信号のデューティ比を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ P（TMP）を外部トリガ・パルス出力／PWM 出力用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
void    TMPn_ChangeDuty ( UCHAR array_duty );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	UCHAR *array_duty*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *array_duty* に設定する値は，10 進数に限られます。

［戻り値］

なし

［使用例］

以下に，デューティ比を 25% に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flagStatus = 1;
    UCHAR   array_duty = 25;
    ……
    TMP0_Start ();                      /* カウントの開始 */
    while ( flagStatus );
```

```
    TMP0_ChangeDuty ( array_duty );     /* デューティ比の変更 */
    .........
}
```

TMP*n*_SoftwareTriggerOn

タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ P（TMP）を外部トリガ・パルス出力／ワンショット・パルス出力用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
void    TMPn_SoftwareTriggerOn ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMQ0_Init

16 ビット・タイマ／イベント・カウンタ（TMQ）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_timer.c

［指定形式］

```
void    TMQ0_Init ( void );
```

［引数］

なし

［戻り値］

なし

TMQ0_UserInit

16 ビット・タイマ／イベント・カウンタ（TMQ）に関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，TMQ0_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_timer_user.c

［指定形式］

```
void    TMQ0_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

TMQ0_Start

16 ビット・タイマ／イベント・カウンタ（TMQ）のカウントを開始します。

備考　本 API 関数を呼び出してからカウント処理を開始するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TMQ0_Start ( void );
```

［引数］

なし

［戻り値］

なし

TMQ0_Stop

16 ビット・タイマ／イベント・カウンタ（TMQ）のカウントを終了します。

備考　本 API 関数を呼び出してからカウント処理を開始するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TMQ0_Stop ( void );
```

［引数］

なし

［戻り値］

なし

TMQ0_ChangeTimerCondition

16 ビット・タイマ／イベント・カウンタ（TMQ）のカウント値を変更します。

備考 引数 *arrar_reg* に指定された値は，TMQ0 キャプチャ／コンペア・レジスタ *m*（TQ0CCR*m*）に設定されます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TMQ0_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

［引数］

I/O	引数	説明
I	USHORT *array_reg;	カウント値（0x0 ～ 0xffff）を格納した領域へのポインタ
I	UCHAR array_num;	変更対象レジスタ 1： TQ0CCR0 2： TQ0CCR0，TQ0CCR1 3： TQ0CCR0，TQ0CCR1，TQ0CCR2 4： TQ0CCR0，TQ0CCR1，TQ0CCR2，TQ0CCR3

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

［使用例］

以下に，インターバル時間を半分に変更する際の例を示します。

なお，下記は，チャネル 0 がインターバル・タイマ用として選択された場合の例となっています。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    int     flag_finish = 1;
```

```
    USHORT  array_reg = TMQ_TQ0CCR0_VALUE >> 1; /* TMQ_TP0CCR0_VALUE：現在のインターバル時間 */
    UCHAR   array_num = 1;
    .........
    TMQ0_Start ();                                          /* カウントの開始 */
    while ( flag_finish );                                  /* タイムアップの確認 */
    .........
    TMQ0_ChanneTimerCondition ( &array_reg, array_num );    /* カウント値の変更 */
   .........
}
```

TMQ0_GetPulseWidth

16 ビット・タイマ／イベント・カウンタ（TMQ）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。

備考 1. 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ（TMQ）をパルス幅測定用として使用している場合に限られます。

2. パルス幅の計測中にオーバフロー（2 回以上）が発生した場合，正常なパルス幅を読み出すことはできません。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
void    TMQ0_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

［引数］

I/O	引数	説明
O	ULONG *activewith;	読み出したハイ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ
O	ULONG *inactivewidth;	読み出したロウ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

［戻り値］

なし

TMQ0_GetFreeRunningValue

16 ビット・タイマ／イベント・カウンタ（TMQ）がキャプチャした値を読み出します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ（TMQ）をフリー・ランニング・タイマ用として使用し，TMQ0 キャプチャ／コンペア・レジスタ *m*（TQ0CCR*m*）がキャプチャ・レジスタとして選択されている場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMQ0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

［引数］

I/O	引数	説明
O	`ULONG *count;`	読み出した幅を格納する領域へのポインタ
I	`enum TMChannel channel;`	読み出し対象チャネル TMCHANNEL0： チャネル 0（TQ0CCR0） TMCHANNEL1： チャネル 1（TQ0CCR1） TMCHANNEL2： チャネル 2（TQ0CCR2） TMCHANNEL3： チャネル 3（TQ0CCR3）

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TMQ0_ChangeDuty

PWM 信号のデューティ比を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ（TMQ）を外部トリガ・パルス出力／PWM 出力用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TMQ0_ChangeDuty ( UCHAR *array_duty, UCHAR array_num );
```

［引数］

I/O	引数	説明
I	UCHAR *array_duty;	デューティ比（0 ～ 100，単位：%）を格納した領域へのポインタ
I	UCHAR array_num;	変更対象レジスタ 1： TQ0CCR1 2： TQ0CCR1，TQ0CCR2 3： TQ0CCR1，TQ0CCR2，TQ0CCR3

備考 デューティ比 *array_duty* に設定する値は，10 進数に限られます。

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

［使用例］

以下に，デューティ比を 25% に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
```

```
    int     flagStatus = 1;
    UCHAR   array_duty = 25;
    UCHAR   array_num = 1;
    ……
    TMQ0_Start ();                                  /* カウントの開始 */
    while ( flagStatus );
    TMQ0_ChangeDuty ( &array_duty, array_num );     /* デューティ比の変更 */
    ……
}
```

TMQ0_SoftwareTriggerOn

タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ（TMQ）を外部トリガ・パルス出力／ワンショット・パルス出力用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
void    TMQ0_SoftwareTriggerOn ( void );
```

[引数]

なし

[戻り値]

なし

TAA*n*_Init

16 ビット・タイマ／イベント・カウンタ AA（TAA）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_timer.c

［指定形式］

```
void    TAAn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAA*n*_UserInit

16 ビット・タイマ／イベント・カウンタ AA（TAA）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，TAAn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_timer_user.c

［指定形式］

```
void    TAAn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAA*n*_Start

16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウントを開始します。

備考 本 API 関数を呼び出してからカウント処理を開始するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TAAn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAA*n*_Stop

16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウントを終了します。

備考 本 API 関数を呼び出してからカウント処理を終了するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TAAn_Stop ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAA*n*_ChangeTimerCondition

16 ビット・タイマ／イベント・カウンタ AA（TAA）のカウント値を変更します。

備考 引数 *arrar_reg* に指定された値は，TAA*n* キャプチャ／コンペア・レジスタ *m*（TAA*n*CCR*m*）に設定されます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TAAn_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	USHORT **array_reg*;	カウント値（0x0 ～ 0xffff）を格納した領域へのポインタ
I	UCHAR *array_num*;	変更対象レジスタ 1： TAA*n*CCR0 2： TAA*n*CCR0，TAA*n*CCR1

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAA*n*_ControlOutputToggle

16 ビット・タイマ／イベント・カウンタ AA（TAA）のトグル制御を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AA（TAA）をインターバル・タイマ／フリー・ランニング・タイマ用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TAAn_ControlOutputToggle ( enum TMOutput toggle, enum TMChannel channel );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	`enum TMOutput toggle;`	トグル制御 STANDARD： 通常のトグル動作 INVACTIVE： リセット要求 ACTIVE： セット要求 FREEZE： キープ要求
I	`enum TMChannel channel;`	対象端子 TMCHANNEL0： TOAA*n*0 端子 TMCHANNEL1： TOAA*n*1 端子

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAA*n*_GetPulseWidth

16 ビット・タイマ／イベント・カウンタ AA（TAA）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。

備考 1. 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AA（TAA）をパルス幅測定用として使用している場合に限られます。

2. パルス幅の計測中にオーバフロー（2 回以上）が発生した場合，正常なパルス幅を読み出すことはできません。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAAn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	ULONG *activewith;	読み出したハイ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ
O	ULONG *inactivewidth;	読み出したロウ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

[戻り値]

なし

TAA*n*_GetFreeRunningValue

16 ビット・タイマ／イベント・カウンタ AA（TAA）がキャプチャした値を読み出します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AA（TAA）をフリー・ランニング・タイマ用として使用し，TAA*n* キャプチャ／コンペア・レジスタ *m*（TAA*n*CCR*m*）がキャプチャ・レジスタとして選択されている場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TAAn_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	`ULONG *count;`	読み出した幅を格納する領域へのポインタ
I	`enum TMChannel channel;`	読み出し対象チャネル TMCHANNEL0： チャネル 0（TAA*n*CCR0） TMCHANNEL1： チャネル 1（TAA*n*CCR1）

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAA*n*_ChangeDuty

PWM 信号のデューティ比を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AA（TAA）を外部トリガ・パルス出力／PWM 出力用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAAn_ChangeDuty ( UCHAR array_duty );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *array_duty*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *array_duty* に設定する値は，10 進数に限られます。

[戻り値]

なし

TAA*n*_SoftwareTriggerOn

タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AA（TAA）を外部トリガ・パルス出力／ワンショット・パルス出力用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
void    TAAn_SoftwareTriggerOn ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAB*n*_Init

16 ビット・タイマ／イベント・カウンタ AB（TAB）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_timer.c

［指定形式］

```
void    TABn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAB*n*_UserInit

16 ビット・タイマ／イベント・カウンタ AB（TAB）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，TABn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_timer_user.c

［指定形式］

```
void    TABn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAB*n*_Start

16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウントを開始します。

備考 本 API 関数を呼び出してからカウント処理を開始するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TABn_Start ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAB*n*_Stop

16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウントを終了します。

備考　本 API 関数を呼び出してからカウント処理を終了するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TABn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TAB*n*_ChangeTimerCondition

16 ビット・タイマ／イベント・カウンタ AB（TAB）のカウント値を変更します。

備考 引数 *arrar_reg* に指定された値は，TAB*n* キャプチャ／コンペア・レジスタ *m*（TAB*n*CCR*m*）に設定されます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TABn_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	USHORT **array_reg*;	カウント値（0x0 ～ 0xffff）を格納した領域へのポインタ
I	UCHAR *array_num*;	変更対象レジスタ 1： TAB*n*CCR0 2： TAB*n*CCR0，TAB*n*CCR1 3： TAB*n*CCR0，TAB*n*CCR1，TAB*n*CCR2 4： TAB*n*CCR0，TAB*n*CCR1，TAB*n*CCR2，TAB*n*CCR3

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAB*n*_ControlOutputToggle

16 ビット・タイマ／イベント・カウンタ AB（TAB）のトグル制御を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AB（TAB）をインターバル・タイマ／フリー・ランニング・タイマ用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TABn_ControlOutputToggle ( enum TMOutput toggle, enum TMChannel channel );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	`enum TMOutput` *`toggle`*`;`	トグル制御 STANDARD： 通常のトグル動作 INVACTIVE： リセット要求 ACTIVE： セット要求 FREEZE： キープ要求
I	`enum TMChannel` *`channel`*`;`	対象端子 TMCHANNEL0： TOAB*n*0 端子 TMCHANNEL1： TOAB*n*1 端子

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAB*n*_GetPulseWidth

16 ビット・タイマ／イベント・カウンタ AB（TAB）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。

備考 1. 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AB（TAB）をパルス幅測定用として使用している場合に限られます。

2. パルス幅の計測中にオーバフロー（2 回以上）が発生した場合，正常なパルス幅を読み出すことはできません。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TABn_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	ULONG *activewith;	読み出したハイ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ
O	ULONG *inactivewidth;	読み出したロウ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

[戻り値]

なし

TAB*n*_GetFreeRunningValue

16 ビット・タイマ／イベント・カウンタ AB（TAB）がキャプチャした値を読み出します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AB（TAB）をフリー・ランニング・タイマ用として使用し，TAB*n* キャプチャ／コンペア・レジスタ *m*（TAB*n*CCR*m*）がキャプチャ・レジスタとして選択されている場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TABn_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	`ULONG *count;`	読み出した幅を格納する領域へのポインタ
I	`enum TMChannel channel;`	読み出し対象チャネル TMCHANNEL0： チャネル 0（TAB*n*CCR0） TMCHANNEL1： チャネル 1（TAB*n*CCR1） TMCHANNEL2： チャネル 2（TAB*n*CCR2) TMCHANNEL3： チャネル 3（TAB*n*CCR3）

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAB*n*_ChangeDuty

PWM 信号のデューティ比を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AB（TAB）を外部トリガ・パルス出力／PWM 出力用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   TABn_ChangeDuty ( UCHAR array_duty, UCHAR array_num );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *array_duty*;	デューティ比（0 ～ 100，単位：%）
I	UCHAR *array_num*;	変更対象レジスタ 1： TAB*n*CCR1 2： TAB*n*CCR1，TAB*n*CCR2 3： TAB*n*CCR1，TAB*n*CCR2，TAB*n*CCR3

備考 デューティ比 *array_duty* に設定する値は，10 進数に限られます。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TAB*n*_SoftwareTriggerOn

タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ AB（TAB）を外部トリガ・パルス出力／ワンショット・パルス出力用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
void    TABn_SoftwareTriggerOn ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

TMT0_Init

16 ビット・タイマ／イベント・カウンタ T（TMT）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_timer.c

［指定形式］

```
void    TMT0_Init ( void );
```

［引数］

なし

［戻り値］

なし

TMT0_UserInit

16 ビット・タイマ／イベント・カウンタ T（TMT）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，TMT0_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_timer_user.c

［指定形式］

```
void    TMT0_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

TMT0_Start

16 ビット・タイマ／イベント・カウンタ T（TMT）のカウントを開始します。

備考 本 API 関数を呼び出してからカウント処理を開始するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

[所属]

CG_timer.c

[指定形式]

```
void    TMT0_Start ( void );
```

[引数]

なし

[戻り値]

なし

TMT0_Stop

16 ビット・タイマ／イベント・カウンタ T（TMT）のカウントを終了します。

備考 本 API 関数を呼び出してからカウント処理を終了するまでの時間は，該当機能の種類（インターバル・タイマ，外部イベント・カウンタ，外部トリガ・パルス出力など）により異なります。

［所属］

CG_timer.c

［指定形式］

```
void    TMT0_Stop ( void );
```

［引数］

なし

［戻り値］

なし

TMT0_ChangeTimerCondition

16 ビット・タイマ／イベント・カウンタ T（TMT）のカウント値を変更します。

備考 引数 *arrar_reg* に指定された値は，TMT0 キャプチャ／コンペア・レジスタ *m*（TT0CCR*m*）に設定されます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TMT0_ChangeTimerCondition ( USHORT *array_reg, UCHAR array_num );
```

［引数］

I/O	引数	説明
I	USHORT *array_reg;	カウント値（0x0 ～ 0xffff）を格納した領域へのポインタ
I	UCHAR array_num;	変更対象レジスタ 1： TT0CCR0 2： TT0CCR0，TT0CCR1

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TMT0_GetPulseWidth

16 ビット・タイマ／イベント・カウンタ T（TMT）のパルス幅（ハイ・レベル幅，ロウ・レベル幅）を読み出します。

備考 1. 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）をパルス幅測定用として使用している場合に限られます。

2. パルス幅の計測中にオーバフロー（2 回以上）が発生した場合，正常なパルス幅を読み出すことはできません。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
void    TMT0_GetPulseWidth ( ULONG *activewith, ULONG *inactivewidth );
```

［引数］

I/O	引数	説明
O	ULONG *activewith;	読み出したハイ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ
O	ULONG *inactivewidth;	読み出したロウ・レベル幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

［戻り値］

なし

TMT0_GetFreeRunningValue

16 ビット・タイマ／イベント・カウンタ T（TMT）がキャプチャした値を読み出します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）をフリー・ランニング・タイマ用として使用し，TMT0 キャプチャ／コンペア・レジスタ *m*（TT0CCR*m*）がキャプチャ・レジスタとして選択されている場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMT0_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[引数]

I/O	引数	説明
O	`ULONG *count;`	読み出した幅を格納する領域へのポインタ
I	`enum TMChannel channel;`	読み出し対象チャネル TMCHANNEL0： チャネル 0（TT0CCR0） TMCHANNEL1： チャネル 1（TT0CCR1）

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TMT0_ChangeDuty

PWM 信号のデューティ比を変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）を外部トリガ・パルス出力／PWM 出力用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   TMT0_ChangeDuty ( UCHAR array_duty );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	UCHAR *array_duty*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *array_duty* に設定する値は，10 進数に限られます。

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

TMT0_SoftwareTriggerOn

タイマ出力のためのトリガ（ソフトウエア・トリガ）を発生させます。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）を外部トリガ・パルス出力／ワンショット・パルス出力用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
void    TMT0_SoftwareTriggerOn ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

TMT0_EnableHold

16 ビット・タイマ／イベント・カウンタ T（TMT）のエンコーダ・カウンタ制御を保持動作へと変更します。

備考 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）をエンコーダ・カウント用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
void    TMT0_EnableHold ( void );
```

[引数]

なし

[戻り値]

なし

TMT0_DisableHold

16 ビット・タイマ／イベント・カウンタ T（TMT）のエンコーダ・カウンタ制御を通常動作へと変更します。

備考　本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）をエンコーダ・カウント用として使用している場合に限られます。

［所属］

CG_timer.c

［指定形式］

```
void    TMT0_DisableHold ( void );
```

［引数］

なし

［戻り値］

なし

TMT0_ChangeCountValue

16 ビット・タイマ／イベント・カウンタ T（TMT）の初期カウント値を変更します。

備考 1. 引数 *regvalue* に指定された値は，TMT0 カウンタ・ライト・レジスタ（TT0TCW）に設定されます。

2. 本 API 関数の呼び出しは，16 ビット・タイマ／イベント・カウンタ T（TMT）をエンコーダ・カウント用として使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TMT0_ChangeCountValue ( USHORT regvalue );
```

[引数]

I/O	引数	説明
I	USHORT *regvalue*;	カウント値（0x0 ～ 0xffff）

[戻り値]

なし

TMM*n*_Init

16 ビット・インターバル・タイマ M（TMM）の機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_timer.c

［指定形式］

```
void    TMMn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMM*n*_UserInit

16 ビット・インターバル・タイマ M（TMM）に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，TMMn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_timer_user.c

［指定形式］

```
void    TMMn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMM*n*_Start

16 ビット・インターバル・タイマ M（TMM）のカウントを開始します。

［所属］

CG_timer.c

［指定形式］

```
void    TMMn_Start ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMM*n*_Stop

16 ビット・インターバル・タイマ M（TMM）のカウントを終了します。

［所属］

CG_timer.c

［指定形式］

```
void    TMMn_Stop ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

TMM*n*_ChangeTimerCondition

16 ビット・インターバル・タイマ M（TMM）のカウント値を変更します。

備考 引数 *regvalue* に指定された値は，TMM*n* コンペア・レジスタ 0（TM*n*CMP0）に設定されます。

［所属］

CG_timer.c

［指定形式］

```
#include    "CG_macrodriver.h"
void    TMMn_ChangeTimerCondition ( USHORT regvalue );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	USHORT *regvalue*;	カウント値（0x0 ～ 0xffff）

［戻り値］

なし

3. 3. 9 時計タイマ

以下に，Applilet3 が時計タイマ用として出力する API 関数の一覧を示します。

表 3—10 時計タイマ用 API 関数

API 関数名	機能概要
WT_Init	時計タイマの機能を制御するうえで必要となる初期化処理を行います。
WT_UserInit	時計タイマに関するユーザ独自の初期化処理を行います。
WT_Start	時計タイマのカウンタをクリアしたのち，カウント処理を再開します。
WT_Stop	時計タイマのカウント処理を終了します。

WT_Init

時計タイマの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_wt.c

[指定形式]

```
void    WT_Init ( void );
```

[引数]

なし

[戻り値]

なし

WT_UserInit

時計タイマに関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，WT_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_wt_user.c

[指定形式]

```
void    WT_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

WT_Start

時計タイマのカウンタをクリアしたのち，カウント処理を再開します。

［所属］

CG_wt.c

［指定形式］

```
void    WT_Start ( void );
```

［引数］

なし

［戻り値］

なし

WT_Stop

時計タイマのカウント処理を終了します。

[所属]

CG_wt.c

[指定形式]

```
void    WT_Stop ( void );
```

[引数]

なし

[戻り値]

なし

[使用例]

以下に，時計タイマの機能を利用する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
ULONG   INT_flg = 0;
void main ( void ) {
    WT_Start ();            /* カウントの再開 */
    while ( !INT_flg );
    WT_Stop ();             /* カウントの終了 */
    ……
}
```

【CG_wt_user.c】

```
#include    "CG_macrodriver.h"
extern  ULONG   INT_flg;
__interrupt void MD_INTWT ( void ) {    /* 割り込み INTWT 発生時の割り込み処理 */
    INT_flg = 1;
}
```

3. 3. 10 リアルタイム・カウンタ

以下に，Applilet3 がリアルタイム・カウンタ用として出力する API 関数の一覧を示します。

表 3—11 リアルタイム・カウンタ用 API 関数

API 関数名	機能概要
RTC_Init	リアルタイム・カウンタの機能を制御するうえで必要となる初期化処理を行います。
RTC_UserInit	リアルタイム・カウンタに関するユーザ独自の初期化処理を行います。
RTC_CounterEnable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを開始します。
RTC_CounterDisable	リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを終了します。
RTC_SetHourSystem	リアルタイム・カウンタの時間制（12 時間制，24 時間制）を設定します。
RTC_CounterSet	リアルタイム・カウンタにカウント値（年，月，曜日，日，時，分，秒）を設定します。
RTC_CounterGet	リアルタイム・カウンタのカウント値（年，月，曜日，日，時，分，秒）を読み出します。
RTC_ConstPeriodInterruptEnable	割り込み INTRTC0 の発生周期を設定したのち，定周期割り込み機能を開始します。
RTC_ConstPeriodInterruptDisable	定周期割り込み機能を終了します。
RTC_AlarmEnable	アラーム割り込み機能を開始します。
RTC_AlarmDisable	アラーム割り込み機能を終了します。
RTC_AlarmSet	アラームの発生条件（曜日，時，分）を設定します。
RTC_AlarmGet	アラームの発生条件（曜日，時，分）を読み出します。
RTC_IntervalStart	インターバル割り込み機能を開始します。
RTC_IntervalStop	インターバル割り込み機能を終了します。
RTC_IntervalInterruptEnable	割り込み INTRTC2 の発生周期を設定したのち，インターバル割り込み機能を開始します。
RTC_IntervalInterruptDisable	インターバル割り込み機能を終了します。
RTC_RC1CK1HZ_OutputEnable	RC1CK1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。
RTC_RC1CK1HZ_OutputDisable	RC1CK1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。
RTC_RC1CKO_OutputEnable	RC1CKO 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。
RTC_RC1CKO_OutputDisable	RC1CKO 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。
RTC_RC1CKDIV_OutputEnable	RC1CKDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。

API 関数名	機能概要
RTC_RC1CKDIV_OutputDisable	RC1CKDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
RTC_RTC1HZ_OutputEnable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。
RTC_RTC1HZ_OutputDisable	RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。
RTC_RTCCL_OutputEnable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。
RTC_RTCCL_OutputDisable	RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。
RTC_RTCDIV_OutputEnable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。
RTC_RTCDIV_OutputDisable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
RTC_ChangeCorrectionValue	時計誤差を補正するタイミング，および補正値を変更します。

RTC_Init

リアルタイム・カウンタの機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_Init ( void );
```

[引数]

なし

[戻り値]

なし

RTC_UserInit

リアルタイム・カウンタに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，RTC_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_rtc_user.c

［指定形式］

```
void    RTC_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

RTC_CounterEnable

リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを開始します。

［所属］

CG_rtc.c

［指定形式］

```
void    RTC_CounterEnable ( void );
```

［引数］

なし

［戻り値］

なし

RTC_CounterDisable

リアルタイム・カウンタ（年，月，曜日，日，時，分，秒）のカウントを終了します。

［所属］

CG_rtc.c

［指定形式］

```
void    RTC_CounterDisable ( void );
```

［引数］

なし

［戻り値］

なし

RTC_SetHourSystem

リアルタイム・カウンタの時間制（12 時間制，24 時間制）を設定します。

［所属］

CG_rtc.c

［指定形式］

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_SetHourSystem ( enum RTCHourSystem hoursystem );
```

［引数］

I/O	引数	説明
I	enum RTCHourSystem *hoursystem*;	時間制の種類 HOUR12：12 時間制 HOUR24：24 時間制

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）
MD_BUSY2	カウント処理を停止中（設定変更後）
MD_ARGERROR	引数の指定が不正

備考 MD_BUSY1，または MD_BUSY2 が返却される場合は，カウンタの動作が停止している，またはカウンタの動作開始待ち時間が短いことに起因している可能性があるため，ヘッダ・ファイル CG_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

［使用例］

以下に，リアルタイム・カウンタの時間制を“24 時間制”に設定する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
```

```
    .........
    RTC_CounterEnable ();           /* カウントの開始 */
    .........
    RTC_SetHourSystem ( HOUR24 );   /* 時間制の設定 */
    .........
}
```

RTC_CounterSet

リアルタイム・カウンタにカウント値を設定します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterSet ( struct RTCCounterValue counterwriteval );
```

[引数]

I/O	引数	説明
I	struct RTCCounterValue *counterwriteval*;	カウント値

備考 以下に，リアルタイム・カウンタのカウント値 RTCCounterValue の構成を示します。

```
struct  RTCCounterValue {
    UCHAR   Sec;    /* 秒 */
    UCHAR   Min;    /* 分 */
    UCHAR   Hour;   /* 時 */
    UCHAR   Day;    /* 日 */
    UCHAR   Week;   /* 曜日（0：日曜日，6：土曜日）*/
    UCHAR   Month;  /* 月 */
    UCHAR   Year;   /* 年 */
};
```

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（設定変更前）
MD_BUSY2	カウント処理を停止中（設定変更後）

備考 MD_BUSY1，または MD_BUSY2 が返却される場合は，カウンタの動作が停止している，またはカウンタの動作開始待ち時間が短いことに起因している可能性があるため，ヘッダ・ファイル CG_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

［使用例］

以下に，リアルタイム・カウンタのカウント値として，“2008 年 12 月 25 日（木）17 時 30 分 00 秒”を設定する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( main ) {
    struct  RTCCounterValue counterwriteval;
    ……
    RTC_CounterEnable ();               /* カウントの開始 */
    ……
    counterwriteval.Year = 0x08;
    counterwriteval.Month = 0x12;
    counterwriteval.Day = 0x25;
    counterwriteval.Week = 0x05;
    counterwriteval.Hour = 0x17;
    counterwriteval.Min = 0x30;
    counterwriteval.Sec = 0;
    RTC_SetHourSystem ( HOUR24 );       /* 時間制の設定 */
    RTC_CounterSet ( counterwriteval ); /* カウント値の設定 */
    ……
}
```

RTC_CounterGet

リアルタイム・カウンタのカウント値を読み出します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_CounterGet ( struct RTCCounterValue *counterreadval );
```

[引数]

I/O	引数	説明
O	struct RTCCounterValue *counterreadval;	読み出したカウント値を格納する構造体へのポインタ

備考 カウント値 RTCCounterValue についての詳細は，RTC_CounterSet を参照してください。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_BUSY1	カウント処理を実行中（読み出し前）
MD_BUSY2	カウント処理を停止中（読み出し後）

備考 MD_BUSY1，または MD_BUSY2 が返却される場合は，カウンタの動作が停止している，またはカウンタの動作開始待ち時間が短いことに起因している可能性があるため，ヘッダ・ファイル CG_rtc.h で定義されているマクロ RTC_WAITTIME の値を大きくしてください。

[使用例]

以下に，リアルタイム・カウンタのカウント値を読み出す際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
```

```
    struct  RTCCounterValue counterreadval;
    .........
    RTC_CounterEnable ();               /* カウントの開始 */
    .........
    RTC_CounterGet ( &counterreadval ); /* カウント値の読み出し */
    .........
}
```

RTC_ConstPeriodInterruptEnable

割り込み INTRTC0 の発生周期を設定したのち，定周期割り込み機能を開始します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_rtc.h"
MD_STATUS   RTC_ConstPeriodInterruptEnable ( enum RTCINTPeriod period );
```

[引数]

I/O	引数	説明
I	enum RTCINTPeriod *period*;	割り込み INTRTC0 の発生周期 HALFSEC： 0.5 秒 ONESEC： 1 秒 ONEMIN： 1 分 ONEHOUR： 1 時間 ONEDAY： 1 日 ONEMONTH： 1ヵ月

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

[使用例]

以下に，割り込み INTRTC0 の発生周期を設定したのち，定周期割り込み機能を開始する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    .........
    RTC_ConstPeriodInterruptDisable ();          /* 定周期割り込み機能の終了 */
    .........
    RTC_ConstPeriodInterruptEnable ( HALFSEC ); /* 定周期割り込み機能の開始 */
```

```
    .........
}
```

RTC_ConstPeriodInterruptDisable

定周期割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_ConstPeriodInterruptDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmEnable

アラーム割り込み機能を開始します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_AlarmEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmDisable

アラーム割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_AlarmDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmSet

アラームの発生条件（曜日，時，分）を設定します。

［所属］

CG_rtc.c

［指定形式］

```
#include    "CG_rtc.h"
void    RTC_AlarmSet ( struct RTCAlarmValue alarmval );
```

［引数］

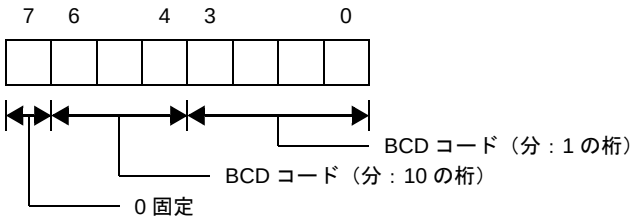
I/O	引数	説明
I	struct RTCAlarmValue *alarmval*;	アラームの発生条件（曜日，時，分）

備考 以下に，アラームの発生条件 RTCAlarmValue の構成を示します。

```
struct  RTCAlarmValue {
    UCHAR   Alarmwm;    /* 分 */
    UCHAR   Alarmwh;    /* 時 */
    UCHAR   Alarmww;    /* 曜日 */
};
```

- Alarmwm（分）

以下に，構成メンバ Alarmwm の各ビットに対する意味を示します。



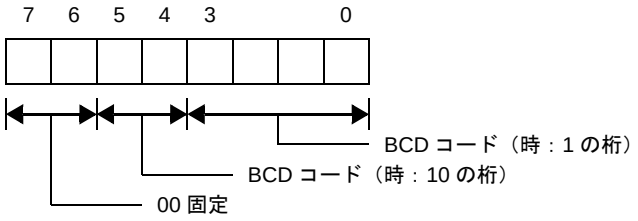
- Alarmwh（時）

以下に，構成メンバ Alarmwh の各ビットに対する意味を示します。

なお，ビット 5 は，リアルタイム・カウンタが 12 時間制の場合，以下の意味となります。

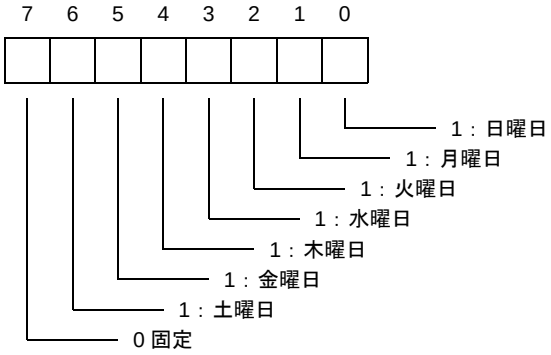
0： 午前

1： 午後



- Alarmww（曜日）

以下に，構成メンバ Alarmww の各ビットに対する意味を示します。



［戻り値］

なし

［使用例 1］

以下に，アラームの発生条件として，“月曜日／火曜日／水曜日の 17 時 30 分”を設定する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ……
    RTC_AlarmEnable ();             /* アラーム割り込み機能の開始 */
    RTC_CounterEnable ();           /* カウントの開始 */
    ……
    RTC_SetHourSystem ( HOUR24 );   /* 時間制の設定 */
    alarmval.Alarmww = 0xe;
    alarmval.Alarmwh = 0x17;
    alarmval.Alarmwm = 0x30;
    RTC_AlarmSet ( alarmval );      /* 発生条件の設定 */
    ……
```

```
}
```

［使用例 2］

以下に，アラームの発生条件を“土曜日／日曜日（時分はそのまま）”に変更する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ……
    RTC_AlarmEnable ();             /* アラーム割り込み機能の開始 */
    ……
    alarmval.Alarmww = 0x41;
    RTC_AlarmSet ( alarmval );      /* 発生条件の変更 */
    ……
}
```

RTC_AlarmGet

アラームの発生条件（曜日，時，分）を読み出します。

［所属］

CG_rtc.c

［指定形式］

```
#include    "CG_rtc.h"
void    RTC_AlarmGet ( struct RTCAlarmValue *alarmval );
```

備考 アラームの発生条件 RTCAlarmValue についての詳細は，RTC_AlarmSet を参照してください。

［引数］

I/O	引数	説明
O	struct RTCAlarmValue *alarmval;	読み出した発生条件を格納する構造体へのポインタ

［戻り値］

なし

［使用例］

以下に，アラームの発生条件を読み出す際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    struct  RTCAlarmValue   alarmval;
    ……
    RTC_AlarmEnable ();         /* アラーム割り込み機能の開始 */
    ……
    RTC_AlarmGet ( &alarmval ); /* 発生条件の読み出し */
    ……
}
```

RTC_IntervalStart

インターバル割り込み機能を開始します。

備考 割り込み INTRTC2 の発生周期を設定したのち，インターバル割り込み機能を開始する場合は，RTC_IntervalInterruptEnable を呼び出します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_IntervalStart ( void );
```

[引数]

なし

[戻り値]

なし

RTC_IntervalStop

インターバル割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_IntervalStop ( void );
```

[引数]

なし

[戻り値]

なし

RTC_IntervalInterruptEnable

割り込み INTRTC2 の発生周期を設定したのち，インターバル割り込み機能を開始します。

備考 割り込み INTRTC2 の発生周期を設定することなく，インターバル割り込み機能を開始する場合は，RTC_IntervalStart を呼び出します。

[所属]

CG_rtc.c

[指定形式]

```
#include    "CG_rtc.h"
MD_STATUS   RTC_IntervalInterruptEnable ( enum RTCINTInterval interval );
```

[引数]

I/O	引数	説明
I	enum RTCINTInterval *interval*;	割り込み INTRTC2 の発生周期 INTERVAL0： 2^6/fRTC INTERVAL1： 2^7/fRTC INTERVAL2： 2^8/fRTC INTERVAL3： 2^9/fRTC INTERVAL4： 2^10/fRTC INTERVAL5： 2^11/fRTC INTERVAL6： 2^12/fRTC

備考 fRTC は，サブシステム・クロックの周波数を意味します。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

[使用例]

以下に，インターバル間隔を変更したのち，インターバル割り込み機能を再開始する際の例を示します。

【CG_main.c】

```
#include    "CG_rtc.h"
void main ( void ) {
    .........
    RTC_IntervalStart ();                       /* インターバル割り込み機能の開始 */
    .........
    RTC_IntervalStop ();                        /* インターバル割り込み機能の終了 */
    .........
    RTC_IntervalInterruptEnable ( INTERVAL6 );  /* インターバル割り込み機能の開始 */
    .........
}
```

RTC_IntervalInterruptDisable

インターバル割り込み機能を終了します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_IntervalInterruptDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RC1CK1HZ_OutputEnable

RC1CK1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RCT_RC1CK1HZ_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RC1CK1HZ_OutputDisable

RC1CK1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。

［所属］

CG_rtc.c

［指定形式］

```
void    RTC_RC1CK1HZ_OutputDisable ( void );
```

［引数］

なし

［戻り値］

なし

RTC_RC1CKO_OutputEnable

RC1CKO 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RC1CKO_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RC1CKO_OutputDisable

RC1CKO 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RC1CKO_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RC1CKDIV_OutputEnable

RC1CKDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RC1CKDIV_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RC1CKDIV_OutputDisable

RC1CKDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RC1CKDIV_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTC1HZ_OutputEnable

RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RCT_RTC1HZ_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTC1HZ_OutputDisable

RTC1HZ 端子に対するリアルタイム・カウンタ補正クロック（1 Hz）の出力を禁止します。

［所属］

CG_rtc.c

［指定形式］

```
void    RTC_RTC1HZ_OutputDisable ( void );
```

［引数］

なし

［戻り値］

なし

RTC_RTCCL_OutputEnable

RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCCL_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCCL_OutputDisable

RTCCL 端子に対するリアルタイム・カウンタ・クロック（32 kHz 原発）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCCL_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCDIV_OutputEnable

RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を許可します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCDIV_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTCDIV_OutputDisable

RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_RTCDIV_OutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_ChangeCorrectionValue

時計誤差を補正するタイミング，および補正値を変更します。

［所属］

CG_rtc.c

［指定形式］

```
#include    "CG_macrodriver.h"
#include    "CG_rtc.h"
MD_STATUS   RTC_ChangeCorrectionValue ( enum RTCCorectionTiming timing, UCHAR corectval );
```

［引数］

I/O	引数	説明
I	enum RTCCorectionTiming *timing*;	時計誤差の補正タイミング EVERY20S： 秒桁が 00，20，40 の時 EVERY60S： 秒桁が 00 の時
I	UCHAR *corectval*;	時計誤差の補正値

備考 本 API 関数では，補正値 *corectVal* に 0x0，0x1，0x40，または 0x41 が指定された際，時計誤差の補正処理を行いません。

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

3. 3. 11 リアルタイム出力機能

以下に，Applilet3 がリアルタイム出力機能用として出力する API 関数の一覧を示します。

表 3—12 リアルタイム出力機能用 API 関数

API 関数名	機能概要
RTOn_Init	リアルタイム出力機能を制御するうえで必要となる初期化処理を行います。
RTOn_UserInit	リアルタイム出力に関するユーザ独自の初期化処理を行います。
RTOn_Enable	リアルタイム出力を許可します。
RTOn_Disable	リアルタイム出力を禁止します。
RTOn_Set2BitsData	リアルタイム出力する 2 ビット・データを設定します。
RTOn_Set4BitsData	リアルタイム出力する 4 ビット・データを設定します。
RTOn_Set6BitsData	リアルタイム出力する 6 ビット・データを設定します。
RTOn_Set8BitsData	リアルタイム出力する 8 ビット・データを設定します。
RTOn_SetHigh2BitsData	リアルタイム出力する上位 2 ビット・データを設定します。
RTOn_SetLow2BitsData	リアルタイム出力する下位 2 ビット・データを設定します。
RTOn_SetHigh4BitsData	リアルタイム出力する上位 4 ビット・データを設定します。
RTOn_SetLow4BitsData	リアルタイム出力する下位 4 ビット・データを設定します。
RTOn_GetValue	リアルタイム出力しているデータを読み出します。

RTO*n*_Init

リアルタイム出力機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_rto.c

[指定形式]

```
void    RTOn_Init ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

RTO*n*_UserInit

リアルタイム出力に関するユーザ独自の初期化処理を行います。

備考 本 API 関数は，RTOn_Init のコールバック・ルーチンとして呼び出されます。

[所属]

CG_rto_user.c

[指定形式]

```
void    RTOn_UserInit ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

RTO*n*_Enable

リアルタイム出力を許可します。

［所属］

CG_rto.c

［指定形式］

```
void    RTOn_Enable ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

RTO*n*_Disable

リアルタイム出力を禁止します。

［所属］

CG_rto.c

［指定形式］

```
void    RTOn_Disable ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

RTO*n*_Set2BitsData

リアルタイム出力する 2 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

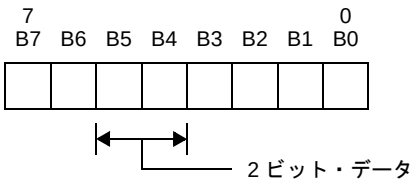
```
#include    "CG_macrodriver.h"
void    RTOn_Set2BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	2 ビット・データ

備考 本 API 関数では，4 ～ 5 ビットに設定された値を 2 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_Set4BitsData

リアルタイム出力する 4 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

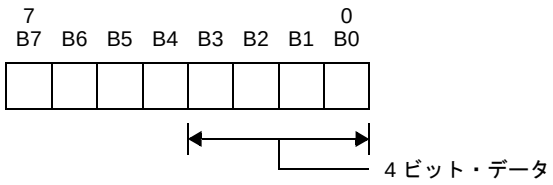
```
#include    "CG_macrodriver.h"
void    RTOn_Set4BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	4 ビット・データ

備考 本 API 関数では，0 ～ 3 ビットに設定された値を 4 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_Set6BitsData

リアルタイム出力する 6 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

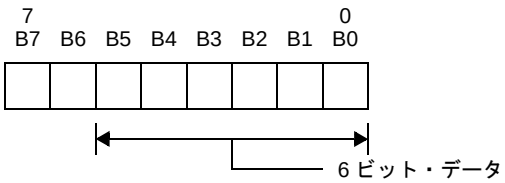
```
#include    "CG_macrodriver.h"
void    RTOn_Set6BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	6 ビット・データ

備考 本 API 関数では，0 ～ 5 ビットに設定された値を 6 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_Set8BitsData

リアルタイム出力する 8 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

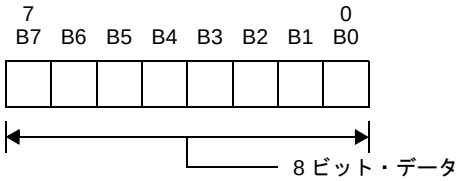
```
#include    "CG_macrodriver.h"
void    RTOn_Set8BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	8 ビット・データ

備考 本 API 関数では，0 ～ 7 ビットに設定された値を 8 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_SetHigh2BitsData

リアルタイム出力する上位 2 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

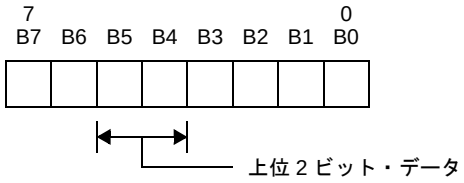
```
#include    "CG_macrodriver.h"
void    RTOn_SetHigh2BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	上位 2 ビット・データ

備考 本 API 関数では，4 ～ 5 ビットに設定された値を上位 2 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_SetLow2BitsData

リアルタイム出力する下位 2 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

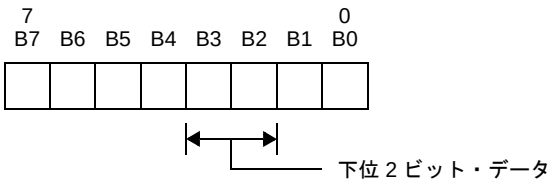
```
#include    "CG_macrodriver.h"
void    RTOn_SetLow2BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	下位 2 ビット・データ

備考 本 API 関数では，2 ～ 3 ビットに設定された値を下位 2 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_SetHigh4BitsData

リアルタイム出力する上位 4 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

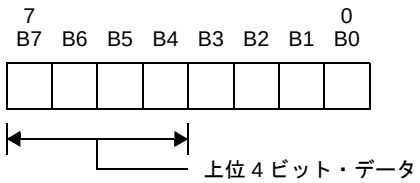
```
#include    "CG_macrodriver.h"
void    RTOn_SetHigh4BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	上位 4 ビット・データ

備考 本 API 関数では，4 ～ 7 ビットに設定された値を上位 4 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_SetLow4BitsData

リアルタイム出力する下位 4 ビット・データを設定します。

[所属]

CG_rto.c

[指定形式]

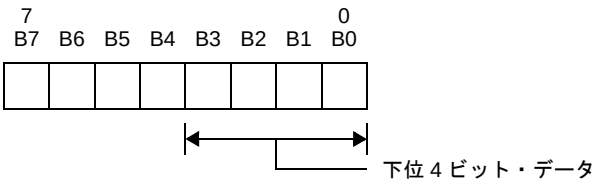
```
#include    "CG_macrodriver.h"
void    RTOn_SetLow4BitsData ( UCHAR data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *data*;	下位 4 ビット・データ

備考 本 API 関数では，0 ～ 3 ビットに設定された値を下位 4 ビット・データとして扱います。



[戻り値]

なし

RTO*n*_GetValue

リアルタイム出力している値を読み出します。

[所属]

CG_rto.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    RTOn_GetValue ( UCHAR *value );
```

備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
O	UCHAR *value;	読み出した値を格納する領域へのポインタ

[戻り値]

なし

[使用例]

以下に，リアルタイム・カウンタのカウント値を読み出す際の例を示します。

【CG_main.c】

```
#include    "CG_rto.h"
void main ( void ) {
    RTO0_Set2BitData ( 0x30 );          /* 出力データの設定 */
    RTO0_Enable ();                     /* リアルタイム出力の許可 */
    ……
    RTO0_Disable ();                    /* リアルタイム出力の禁止 */
    ……
}
```

【CG_timer_user.c】

```
#include    "CG_macrodriver.h"
```

```
__interrupt void MD_INTTP4CC0 ( void ) {  /* 割り込み INTTP4CC0 発生時の割り込み処理 */
    UCHAR   value = 0;
    RTO0_GetValue ( &value );             /* 出力データの読み出し */
    value = ~value;
    RTO0_Set2BitData ( value );           /* 出力データの設定 */
}
```

3. 3. 12 DMA コントローラ

以下に，Applilet3 が DMA コントローラ用として出力する API 関数の一覧を示します。

表 3—13 DMA コントローラ用 API 関数

API 関数名	機能概要
DMAn_Init	DMA コントローラの機能を制御するうえで必要となる初期化処理を行います。
DMAn_UserInit	DMA コントローラに関するユーザ独自の初期化処理を行います。
DMAn_Enable	チャネル *n* を動作許可状態に設定します。
DMAn_Disable	チャネル *n* を動作停止状態に設定します。
DMAn_CheckStatus	転送状態（転送終了，転送中）を読み出します。
DMAn_SetData	転送先／転送元の RAM アドレス，およびデータの転送回数を設定します。
DMAn_SoftwareTriggerOn	DMA 転送の起動要因として，ソフトウエア・トリガを使用します。

DMA*n*_Init

DMA コントローラの機能を制御するうえで必要となる初期化処理を行います。

［所属］

CG_dma.c

［指定形式］

```
void    DMAn_Init ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

DMA*n*_UserInit

DMA コントローラに関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，DMAn_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_dma_user.c

［指定形式］

```
void    DMAn_UserInit ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

DMA*n*_Enable

チャネル *n* を動作許可状態に設定します。

［所属］

CG_dma.c

［指定形式］

```
void    DMAn_Enable ( void );
```

備考　*n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

DMA*n*_Disable

チャネル *n* を動作停止状態に設定します。

備考 1. 本 API 関数は，DMA 転送を強制終了させるものではありません。

2. 本 API 関数は，DMAn_CheckStatus による“転送終了”の確認後に呼び出す必要があります。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Disable ( void );
```

備考 *n* は，チャネル番号を意味します。

[引数]

なし

[戻り値]

なし

[使用例]

以下に，チャネル 0 の動作モードを“動作停止状態”へと移行させる際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
void main ( void ) {
    .........
    while ( MD_COMPLETED == DMA0_CheckStatus () );  /* 転送状態の確認 */
    DMA0_Disable ();                                /* 動作停止状態への移行 */
    .........
}
```

DMA*n*_CheckStatus

転送状態（転送終了，転送中）を読み出します。

［所属］

CG_dma.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_CheckStatus ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

マクロ	説明
MD_UNDEREXEC	転送中
MD_COMPLETED	転送終了

DMA*n*_SetData

転送先／転送元の RAM アドレス，およびデータの転送回数を設定します。

備考 本 API 関数を転送中に呼び出した場合，転送は強制終了します。

［所属］

CG_dma.c

［指定形式］

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( UINT srcaddr, UINT dstaddr, UINT count );
```

備考 *n* は，チャネル番号を意味します。

［引数］

I/O	引数	説明
I	UINT *srcaddr*;	転送元の RAM アドレス
I	UINT *dstaddr*;	転送先の RAM アドレス
I	UINT *count*;	データの転送回数（1 ～ 1024）

［戻り値］

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	引数の指定が不正

DMA*n*_SoftwareTriggerOn

DMA 転送の起動要因として，ソフトウエア・トリガ（本 API 関数の呼び出し）を使用します。

［所属］

CG_dma.c

［指定形式］

```
void    DMAn_SoftwareTriggerOn ( void );
```

備考 *n* は，チャネル番号を意味します。

［引数］

なし

［戻り値］

なし

［使用例］

以下に，DMA 転送の起動要因をソフトウエア・トリガとした場合の例を示します。

【CG_main.c】

```
void main ( void ) {
    ……
    DMA0_Enable ();              /* 動作許可状態への移行 */
    DMA0_SoftwareTriggerOn ();   /* DMA 転送の開始 */
    ……
}
```

3. 3. 13 低電圧検出回路

以下に，Applilet3 が低電圧検出回路用として出力する API 関数の一覧を示します。

表 3—14 低電圧検出回路用 API 関数

API 関数名	機能概要
LVI_Init	低電圧検出回路の機能を制御するうえで必要となる初期化処理を行います。
LVI_UserInit	低電圧検出回路に関するユーザ独自の初期化処理を行います。
LVI_InterruptModeStart	低電圧検出動作を開始します（割り込み発生モード時）。
LVI_ResetModeStart	低電圧検出動作を開始します（内部リセット・モード時）。
LVI_Start	低電圧検出動作を開始します。
LVI_Stop	低電圧検出動作を停止します。

LVI_Init

低電圧検出回路の機能を制御するうえで必要となる初期化処理を行います。

[所属]

CG_lvi.c

[指定形式]

```
void    LVI_Init ( void );
```

[引数]

なし

[戻り値]

なし

LVI_UserInit

低電圧検出回路に関するユーザ独自の初期化処理を行います。

備考　本 API 関数は，LVI_Init のコールバック・ルーチンとして呼び出されます。

［所属］

CG_lvi_user.c

［指定形式］

```
void    LVI_UserInit ( void );
```

［引数］

なし

［戻り値］

なし

LVI_InterruptModeStart

低電圧検出動作を開始します（割り込み発生モード時）。

[所属]

CG_lvi.c

[指定形式]

```
void    LVI_InterruptModeStart ( void );
```

[引数]

なし

[戻り値]

なし

[使用例]

以下に，低電圧を検出した際の動作モードが割り込み発生モード（割り込み INTLVI を発生させる）における例を示します。

【CG_main.c】

```
void main ( void ) {
    ……
    LVI_InterruptModeStart ( );          /* 低電圧検出動作の開始 */
    ……
}
```

【CG_lvi_user.c】

```
__interrupt void MD_INTLVI ( void ) {    /* 割り込み INTLVI 発生時の割り込み処理 */
    if ( LVIF == 1 ) {                   /* 発生要因の判別：LVIF フラグのチェック */
        ……    /* “電源電圧（VDD）< 検出電圧（VLVI）”を検出した際の処理 */
    } else {
        ……    /* “電源電圧（VDD）≧ 検出電圧（VLVI）”を検出した際の処理 */
    }
}
```

LVI_ResetModeStart

低電圧検出動作を開始します（内部リセット・モード時）。

[所属]

CG_lvi.c

[指定形式]

```
MD_STATUS   LVI_ResetModeStart ( void );
```

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了 - 低電圧検出回路の機能を使用しない設定が行われている - 低電圧検出対象が外部電圧（V_{DD}）の際，電源電圧（V_{DD}）≦ 検出電圧（V_{LVI}） - 低電圧検出対象が外部入力電圧（EXLVI）の際，外部入力電圧（EXLVI）≦ 検出電圧（V_{EXLVI}）

LVI_Start

低電圧検出動作を開始します。

[所属]

CG_lvi.c

[指定形式]

```
MD_STATUS   LVI_Start ( void );
```

[引数]

なし

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ARGERROR	異常終了 - 低電圧検出回路の機能を使用しない設定が行われている - 低電圧検出対象が外部電圧（V_{DD}）の際，電源電圧（V_{DD}）≦ 検出電圧（V_{LVI}） - 低電圧検出対象が外部入力電圧（EXLVI）の際，外部入力電圧（EXLVI）≦ 検出電圧（V_{EXLVI}）

LVI_Stop

低電圧検出動作を停止します。

[所属]

CG_lvi.c

[指定形式]

```
void    LVI_Stop ( void );
```

[引数]

なし

[戻り値]

なし

付録 A 索 引

【T】

【た行】

【は行】

【ら行】

【わ行】

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2010.09.01	―	初版発行

［メモ］

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。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。
3. 当社製品を改造、改変、複製等しないでください。
4. 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。
5. 輸出に際しては、「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。本資料に記載されている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の目的で使用しないでください。また、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に使用することができません。
6. 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。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されてい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生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   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

   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、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社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。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、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、社会的損害など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、機器またはシステムとしての出荷保証をお願いいたします。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、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、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。
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固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

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

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

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・電話番号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 エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル）　(03)5201-5307

■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/inquiry