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

78K0 マイクロコントローラ

資料番号　ZUD-CD-10-0163（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。

- 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。

- 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+，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。

第 2 章　出力ファイル

本付録では，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を示します。

出力単位	ファイル名	出力内容
各周辺機能	CG_ *周辺機能名* .c	初期化関数，API 関数
	CG_ *周辺機能名* _user.c	割り込み関数，コールバック関数
	CG_ *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	CG_option.asm	オプション・バイト，MINICUBE2 用 ROM 確保
	CG_systeminit.c	各周辺機能の初期化関数コール CG_ReadResetSource のコール
	CG_main.c	main 関数
	CG_macrodriver.h	全ソース・ファイルで共通使用するマクロを定義
	CG_user_define.h	空ファイル（ユーザ定義用）
	CG_lk.dir	リンク・ディレクティブ

2.2　出力ファイル

以下に，Applilet3 が出力するファイルの一覧を示します。

表 2―1　ファイル一覧

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 CG_SelectStabTime CG_ChagngePllMode
	CG_system_user.c	CLOCK_UserInit CG_ReadResetSource
	CG_system.h	―
ポート	CG_port.c	PORT_Init PORT_ChangePmnInput PORT_ChangePmnOutput

周辺機能	ファイル名	含まれる API 関数名
ポート	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_INTP*n* MD_INTKR INTP_UserInit KEY_UserInit
	CG_int.h	–
シリアル	CG_serial.c	UART6_Init UART6_Start UART6_Stop UART6_SendData UART6_ReceiveData CSI1n_Init CSI1n_Start CSI1n_Stop CSI1n_ReceiveData CSI1n_SendReceiveData IICA_Init IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart
	CG_serial_user.c	MD_INTSR6 MD_INTSRE6 MD_INTST6 MD_INTCSI1*n* MD_INTIICA0 UART6_UserInit UART6_SendEndCallback UART6_ReceiveEndCallback UART6_SoftOverRunCallback UART6_ErrorCallback CSI1n_UserInit CSI1n_SendEndCallback

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	CSI1n_ReceiveEndCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback
	CG_serial.h	—
オペアンプ	CG_opamp.c	OPAMP_Init PGA_Start PGA_Stop PGA_ChangePGAFactor AMP_Start AMP_Stop AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	—
コンパレータ	CG_comparator.c	Comparator_Init Comparatorn_Start Comparatorn_Stop
	CG_comparator_user.c	MD_INTCMP*n* Comparator_UserInit
	CG_comparator.h	—
A/D コンバータ	CG_ad.c	AD_Init AD_ComparatorOn AD_ComparatorOff AD_Start AD_Stop AD_SelectADChannel AD_Read AD_ReadByte
	CG_ad_user.c	MD_INTAD AD_UserInit
	CG_ad.h	—
タイマ	CG_timer.c	TMX_Init TMXn_Start TMXn_Stop TMXn_ChangeDuty TMXn_ChangeDualDuty

周辺機能	ファイル名	含まれる API 関数名
タイマ	CG_timer.c	TMX_EnableHighImpedanceState TMX_DisableHighImpedanceState TM00_Init TM00_Start TM00_Stop TM00_ChangeTimerCondition TM00_GetFreeRunningValue TM00_SoftwareTriggerOn TM00_ChangeDuty TM00_GetPulseWidth TM5n_Init TM5n_Start TM5n_Stop TM5n_ChangeTimerCondition TM5n_ChangeDuty TMHn_Init TMHn_Start TMHn_Stop TMHn_ChangeTimerCondition TMHn_ChangeDuty TMH1_CarrierOutputEnable TMH1_CarrierOutputDisable
	CG_timer_user.c	MD_INTTMX*n* MD_INTTM0*n*0 MD_INTTM5*n* MD_INTMH*n* TM00_UserInit TM5n_UserInit TMHn_UserInit
	CG_timer.h	–
ウォッチドッグ・タイマ	CG_wdt.c	WDT_Restart
	CG_wdt.h	–
リアルタイム・カウンタ	CG_rtc.c	RTC_Init RTC_PowerOff RTC_CounterEnable RTC_CounterDisable RTC_SetHourSystem RTC_CounterSet RTC_CounterGet RTC_ConstPeriodInterruptEnable RTC_ConstPeriodInterruptDisable RTC_AlarmEnable RTC_AlarmDisable RTC_AlarmSet

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_AlarmGet RTC_IntervalStart RTC_IntervalStop RTC_IntervalInterruptEnable RTC_IntervalInterruptDisable 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 MD_INTRTCI RTC_UserInit RTC_ConstPeriodInterruptCallback RTC_AlarmInterruptCallback
	CG_rtc.h	–
クロック出力	CG_pcl.c	PCL_Init PCL_Start PCL_Stop PCL_ChangeFreq
	CG_pcl_user.c	PCL_UserInit
	CG_pcl.h	–
低電圧検出回路	CG_lvi.c	LVI_Init LVI_InterruptModeStart LVI_ResetModeStart LVI_Stop LVI_SetLVILevel
	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	クロック発生回路の機能，オンチップ・デバッグ機能など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	クロック発生回路，オンチップ・デバッグなどに関するユーザ独自の初期化処理を行います。
	CG_ReadResetSource	内部リセット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
システム	CG_ChangeClockMode	CPU クロック／周辺ハードウエア・クロックを変更します。
	CG_ChangeFrequency	CPU クロック／周辺ハードウエア・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	X1 クロックの発振安定時間を設定します。
	CG_ChagngePllMode	PLL 機能の動作を制御します。
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。
	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	UART6_Init	シリアル・インタフェース（UART6）の初期化処理を行います。
	UART6_UserInit	シリアル・インタフェース（UART6）に関するユーザ独自の初期化処理を行います。
	UART6_Start	UART 通信を待機状態にします。
	UART6_Stop	UART 通信を終了します。
	UART6_SendData	データの UART 送信を開始します。
	UART6_ReceiveData	データの UART 受信を開始します。

周辺機能	API 関数名	機能概要
シリアル	UART6_SendEndCallback	UART 送信完了割り込み INTST6 の発生に伴う処理を行います。
	UART6_ReceiveEndCallback	UART 受信完了割り込み INTSR6 の発生に伴う処理を行います。
	UART6_SoftOverRunCallback	UART 受信完了割り込み INTSR6 の発生に伴う処理を行います。
	UART6_ErrorCallback	UART 通信におけるエラー割り込み INTSRE6 の発生に伴う処理を行います。
	CSI1n_Init	シリアル・インタフェース（CSI1*n*）の初期化処理を行います。
	CSI1n_UserInit	シリアル・インタフェース（CSI1*n*）に関するユーザ独自の初期化処理を行います。
	CSI1n_Start	CSI1*n* 通信を待機状態にします。
	CSI1n_Stop	CSI1*n* 通信を終了します。
	CSI1n_ReceiveData	データの CSI1*n* 受信を開始します。
	CSI1n_SendReceiveData	データの CSI1*n* 送受信を開始します。
	CSI1n_SendEndCallback	CSI1*n* 通信完了割り込み INTCSI1*n* の発生に伴う処理を行います。
	CSI1n_ReceiveEndCallback	CSI1*n* 通信完了割り込み INTCSI1*n* の発生に伴う処理を行います。
	IICA_Init	シリアル・インタフェース（IICA）の初期化処理を行います。
	IICA_UserInit	シリアル・インタフェース（IICA）に関するユーザ独自の初期化処理を行います。
	IICA_Stop	IICA 通信を終了します。
	IICA_MasterSendStart	IICA マスタ送信を開始します。
	IICA_MasterReceiveStart	IICA マスタ受信を開始します。
	IICA_StopCondition	ストップ・コンディションを発生します。
	IICA_MasterSendEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。
	IICA_MasterReceiveEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。
	IICA_MasterErrorCallback	IICA マスタ通信におけるエラーの検出に伴う処理を行います。
	IICA_SlaveSendStart	IICA スレーブ送信を開始します。
	IICA_SlaveReceiveStart	IICA スレーブ受信を開始します。
	IICA_SlaveSendEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。
	IICA_SlaveReceiveEndCallback	IICA 通信完了割り込み INTIICA0 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	IICA_SlaveErrorCallback	IICA スレーブ通信におけるエラーの検出に伴う処理を行います。
	IICA_GetStopConditionCallback	IICA スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。
オペアンプ	OPAMP_Init	オペアンプ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。
	OPAMP_UserInit	オペアンプに関するユーザ独自の初期化処理を行います。
	PGA_Start	オペアンプ（PGA モード）の動作を開始します。
	PGA_Stop	オペアンプ（PGA モード）の動作を停止します。
	PGA_ChangePGAFactor	オペアンプ（PGA モード）における入力電圧の増幅率を設定します。
	AMP_Start	オペアンプ（シングル・アンプ・モード）の動作を開始します。
	AMP_Stop	オペアンプ（シングル・アンプ・モード）の動作を停止します。
	AMPn_Start	オペアンプ *n*（シングル・アンプ・モード）の動作を開始します。
	AMPn_Stop	オペアンプ *n*（シングル・アンプ・モード）の動作を停止します。
コンパレータ	Comparator_Init	コンパレ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。
	Comparator_UserInit	コンパレータに関するユーザ独自の初期化処理を行います。
	Comparatorn_Start	コンパレータ *n* の動作を開始します。
	Comparatorn_Stop	コンパレータ *n* の動作を停止します。
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_ComparatorOn	電圧コンパレータを動作許可状態に設定します。
	AD_ComparatorOff	電圧コンパレータを動作停止状態に設定します。
	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_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
タイマ	TMX_Init	16 ビット・タイマ X*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。
	TMXn_Start	16 ビット・タイマ X*n* のカウントを開始します。
	TMXn_Stop	16 ビット・タイマ X*n* のカウントを終了します。
	TMXn_ChangeDuty	TOX0*n* 端子にシングル出力する PWM 信号のデューティ比を変更します。
	TMXn_ChangeDualDuty	TOX0*n* 端子にデュアル出力する PWM 信号のデューティ比を変更します。
	TMX_EnableHighImpedanceState	16 ビット・タイマ X*n* のハイ・インピーダンス出力を開始します。
	TMX_DisableHighImpedanceState	16 ビット・タイマ X*n* のハイ・インピーダンス出力を終了します。
	TM00_Init	16 ビット・タイマ／イベント・カウンタ 00 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。
	TM00_UserInit	16 ビット・タイマ／イベント・カウンタ 00 に関するユーザ独自の初期化処理を行います。
	TM00_Start	16 ビット・タイマ／イベント・カウンタ 00 のカウントを開始します。
	TM00_Stop	16 ビット・タイマ／イベント・カウンタ 00 のカウントを終了します。
	TM00_ChangeTimerCondition	キャプチャ／コンペア・コントロール・レジスタ（CRC00）の内容を変更します。
	TM00_GetFreeRunningValue	キャプチャ・レジスタ（CR0*n*0）の内容を獲得します。
	TM00_SoftwareTriggerOn	ワンショット・パルス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。
	TM00_ChangeDuty	TO00 端子に出力する信号のデューティ比を変更します。
	TM00_GetPulseWidth	TI0*n*0 端子に対する入力信号（入力パルス）のハイ／ロウ・レベルの測定幅を獲得します。
	TM5n_Init	8 ビット・タイマ／イベント・カウンタ 5*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。
	TM5n_UserInit	8 ビット・タイマ／イベント・カウンタ 5*n* に関するユーザ独自の初期化処理を行います。
	TM5n_Start	8 ビット・タイマ／イベント・カウンタ 5*n* のカウントを開始します。
	TM5n_Stop	8 ビット・タイマ／イベント・カウンタ 5*n* 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	TM5n_ChangeTimerCondition	8 ビット・タイマ・コンペア・レジスタ 5*n*（CR5*n*）の内容を変更します。
	TM5n_ChangeDuty	TO5*n* 端子に出力する PWM 信号のデューティ比を変更します。
	TMHn_Init	8 ビット・タイマ H*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。
	TMHn_UserInit	8 ビット・タイマ H*n* に関するユーザ独自の初期化処理を行います。
	TMHn_Start	8 ビット・タイマ H*n* のカウントを開始します。
	TMHn_Stop	8 ビット・タイマ H*n* のカウントを終了します。
	TMHn_ChangeTimerCondition	8 ビット・タイマ H コンペア・レジスタ 0*n* ／ 1*n*（CMP0*n* ／ CMP1*n*）の内容を変更します。
	TMHn_ChangeDuty	TOHn 端子に出力する PWM 信号のデューティ比を変更します。
	TMH1_CarrierOutputEnable	8 ビット・タイマ H1（キャリア・ジェネレータ・モード）のキャリア・パルス出力を開始します。
	TMH1_CarrierOutputDisable	8 ビット・タイマ H1（キャリア・ジェネレータ・モード）のキャリア・パルス出力を終了します。
ウォッチドッグ・タイマ	WDT_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。
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_PowerOff	リアルタイム・カウンタに対するクロック供給を停止します。
	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 INTRTC 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。

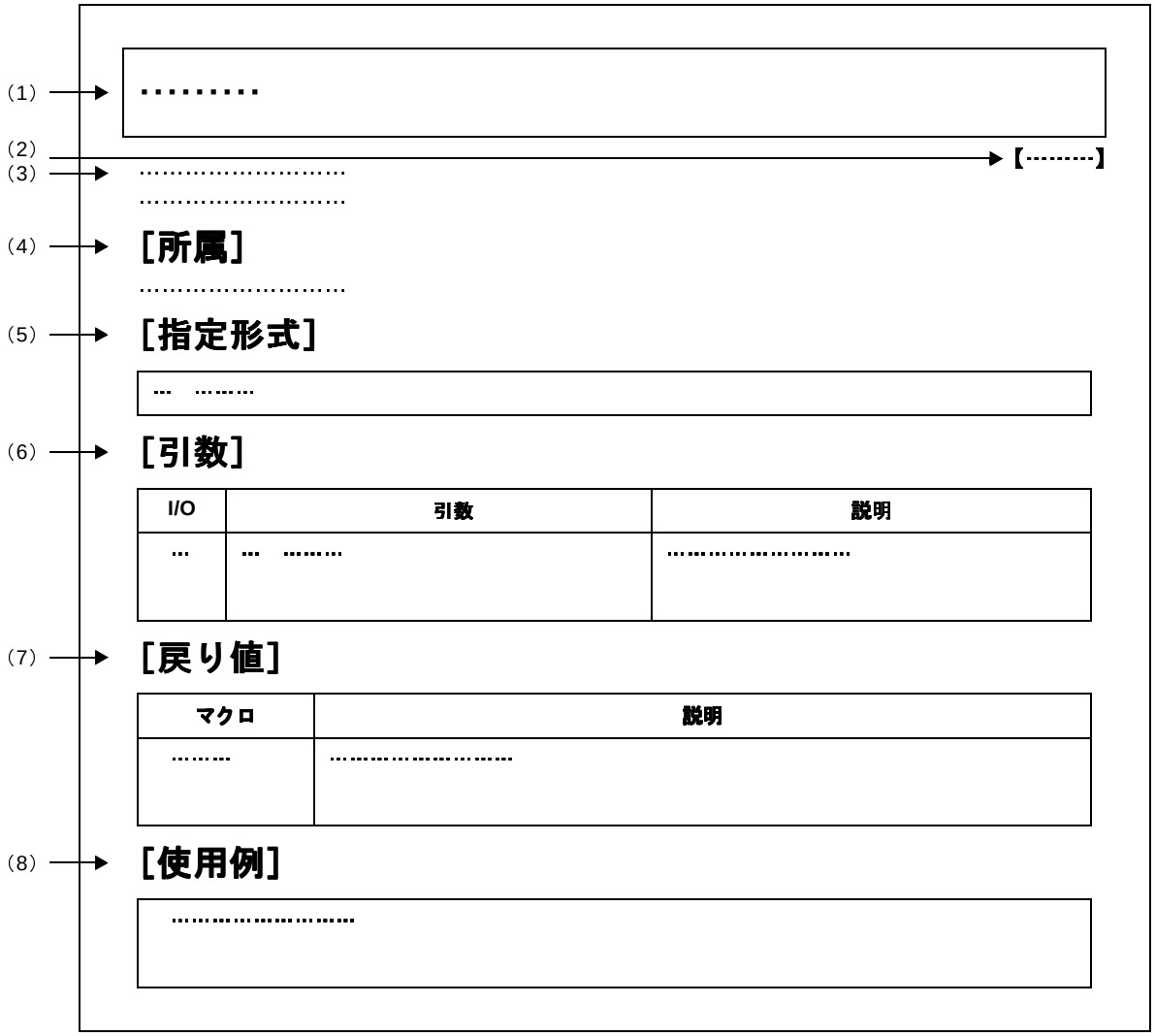
周辺機能	API 関数名	機能概要
リアルタイム・カウンタ	RTC_ConstPeriodInterruptCallback	定周期割り込み INTRTC の発生に伴う処理を行います。
	RTC_AlarmEnable	アラーム割り込み機能を開始します。
	RTC_AlarmDisable	アラーム割り込み機能を終了します。
	RTC_AlarmSet	アラームの発生条件（曜日，時，分）を設定します。
	RTC_AlarmGet	アラームの発生条件（曜日，時，分）を読み出します。
	RTC_AlarmInterruptCallback	アラーム割り込み INTRTC の発生に伴う処理を行います。
	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 INTRTCI 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_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。
クロック出力	PCL_Init	クロック出力制御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。
	PCL_UserInit	クロック出力制御回路に関するユーザ独自の初期化処理を行います。
	PCL_Start	クロック出力を開始します。
	PCL_Stop	クロック出力を停止します。
	PCL_ChangeFreq	PCL 端子への出力クロックを変更します。
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。

周辺機能	API 関数名	機能概要
低電圧検出回路	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_Stop	低電圧検出動作を停止します。
	LVI_SetLVILevel	低電圧検出レベルを設定します。

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) 対象デバイス

API 関数の出力対象となるデバイス名を示しています。

(3) 機能

API 関数の機能概要を示しています。

(4) [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。

(5)［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。

(6)［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

(7)［戻り値］

API 関数からの戻り値を示しています。

(8)［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	クロック発生回路の機能，オンチップ・デバッグ機能など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	クロック発生回路，オンチップ・デバッグなどに関するユーザ独自の初期化処理を行います。
CG_ReadResetSource	内部リセットの発生に伴う処理を行います。
CG_ChangeClockMode	CPU クロック／周辺ハードウエア・クロックを変更します。
CG_ChangeFrequency	CPU クロック／周辺ハードウエア・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	X1 クロックの発振安定時間を設定します。
CG_ChagngePllMode	PLL 機能の動作を制御します。

CLOCK_Init

【Ix2】【Kx2-L】

クロック発生回路の機能，オンチップ・デバッグ機能など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

【Ix2】【Kx2-L】

クロック発生回路，オンチップ・デバッグなど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

【Ix2】【Kx2-L】

内部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，リセット信号 RESET 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 ();      /* リセット信号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   flag = 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 ( flag & 0x1 ) {         /* 発生要因の判別：LVIRF フラグのチェック */
        ……    /* 低電圧検出回路による内部リセット要求に対応した処理 */
    } else if ( flag & 0x10 ) { /* 発生要因の判別：WDTRF フラグのチェック */
        ……    /* ウォッチドッグ・タイマによる内部リセット要求に対応した処理 */
    }
    ……
}
```

CG_ChangeClockMode

【Ix2】【Kx2-L】

CPU クロック／周辺ハードウエア・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（fCPU）／周辺ハードウエア・クロック（fPRS）の種類 HIOCLK： fCPU ／ fPRS → 高速内蔵発振クロック HIOSYSCLK： fCPU → 高速内蔵発振クロック，fPRS → 高速システム・クロック SYSX1CLK： fCPU ／ fPRS → X1 クロック SYSEXTCLK： fCPU ／ fPRS → 外部メイン・システム・クロック SUBXT1CLK： fCPU → XT1 クロック SUBEXTCLK： fCPU → 外部サブシステム・クロック

備考 SUBXT1CLK，SUBEXTCLK の指定は，対象デバイスが 78K0/KC2-L の場合に限られます。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了 - fCPU ／ fPRS を高速内蔵発振クロックへ変更することはできません。
MD_ERROR2	異常終了 - fCPU を高速内蔵発振クロックへ変更するこはできません。 - fPRS を高速システム・クロックへ変更することはできません。
MD_ERROR3	異常終了 - fCPU ／ fPRS を X1 クロックへ変更することはできません。

マクロ	説明
MD_ERROR4	異常終了 - fCPU ／ fPRS を外部メイン・システム・クロックへ変更することはできません。
MD_ERROR5	異常終了【Kx2-L】 - fCPU を XT1 クロックへ変更することはできません。
MD_ERROR6	異常終了【Kx2-L】 - fCPU を外部サブシステム・クロックへ変更することはできません。
MD_ARGERROR	引数の指定が不正

備考 戻り値 MD_ERROR5，MD_ERROR6 は，対象デバイスが 78K0/KC2-L の場合に限られます。

CG_ChangeFrequency

【Ix2】【Kx2-L】

CPU クロック／周辺ハードウエア・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 : fMAIN SYSONEHALF : fMAIN/2 SYSONEFOURTH : fMAIN/4 SYSONEEIGHTH : fMAIN/8 SYSONESIXTEENTH : fMAIN/16

備考 fMAIN は，メイン・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

CG_SelectPowerSaveMode

【Ix2】【Kx2-L】

CPU のスタンバイ・モードを設定します。

[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 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_ERROR	異常終了【Kx2-L】 - CPU が XT1 クロックで動作している場合，STOP モードを指定することはできません。
MD_ARGERROR	引数の指定が不正

備考 戻り値 MD_ERROR は，対象デバイスが 78K0/KC2-L の場合に限られます。

[使用例]

以下に，スタンバイ・モードを“STOP モード”へと移行させる際の例を示します。

【CG_main.c】

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

```
    MD_STATUS   ret;
    ……
    TM00_Stop ();                                   /* カウントの終了 */
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* STOP モードへの移行 */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
    TM00_Init ();                                   /* TM00 の初期化 */
    TM00_Start ();                                  /* カウントの開始 */
    ……
}
```

CG_SelectStabTime

【Ix2】【Kx2-L】

X1 クロックの発振安定時間を設定します。

[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 STLEVEL0：2^11/fx STLEVEL1：2^13/fx STLEVEL2：2^14/fx STLEVEL3：2^15/fx STLEVEL4：2^16/fx

備考 fx は，X1 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

CG_ChagngePllMode

【Ix2】

PLL 機能の動作を制御します。

[所属]

CG_system.c

[指定形式]

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

[引数]

I/O	引数	説明
I	enum PllMode *pllmode*;	動作の制御 SYSPLLON： 動作許可 SYSPLLOFF： 停止

[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. 2 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—3 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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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]

本 API 関数の指定形式は，対象端子に内蔵プルアップ抵抗／ SMBus 入力バッファが存在するか否かにより異なります。

- 内蔵プルアップ抵抗：なし，SMBus 入力バッファ：なし

```
void    PORT_ChangeP mn Input ( void );
```

- 内蔵プルアップ抵抗：あり，SMBus 入力バッファ：なし

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

- 内蔵プルアップ抵抗：あり，SMBus 入力バッファ：あり

```
#include    "CG_macrodriver.h"
void    PORT_ChangeP mn Input ( BOOL enablepu, BOOL enablesmbus );
```

備考 *mn* は，ポート番号を意味します。

[引数]

I/O	引数	説明
I	BOOL *enablepu*;	内蔵プルアップ抵抗の使用有無 MD_TRUE : 使用する MD_FALSE : 使用しない
I	BOOL *enablesmbus*;	入力バッファの種類 MD_TRUE : SMBus 入力バッファ MD_FALSE : 通常入力バッファ

[戻り値]

なし

[使用例 1]

以下に，P00 端子（内蔵プルアップ抵抗：あり，SMBus 入力バッファ：なし）を
　入出力モードの種類： 入力モード
　内蔵プルアップ抵抗の使用有無： 使用する
に変更する際の例を示します。

【CG_main.c】

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

[使用例 2]

以下に，P00 端子（内蔵プルアップ抵抗：あり，SMBus 入力バッファ：なし）を
　入出力モードの種類： 入力モード
　内蔵プルアップ抵抗の使用有無： 使用しない
に変更する際の例を示します。

【CG_main.c】

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

[使用例 3]

以下に，P04 端子（内蔵プルアップ抵抗：あり，SMBus 入力バッファ：あり）を
　入出力モードの種類： 入力モード
　内蔵プルアップ抵抗の使用有無： 使用しない
　入力バッファの種類： SMBus 入力バッファ
に変更する際の例を示します。

【CG_main.c】

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

```
    ……
    PORT_ChangeP04Input ( MD_FALSE, MD_TRUE );  /* 入出力モードの切り替え */
    ……
}
```

PORT_ChangeP*mn*Output

【Ix2】【Kx2-L】

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（V_{DD} 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. 3 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—4 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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

【KC2-L】

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

【KC2-L】

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

【Ix2】【Kx2-L】

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]

```
#include    "CG_macrodriver.h"
#include    "CG_int.h"
MD_STATUS   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]

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

[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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

【KC2-L】

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

【KC2-L】

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. 4 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—5 シリアル用 API 関数

API 関数名	機能概要
UART6_Init	シリアル・インタフェース（UART6）の初期化処理を行います。
UART6_UserInit	シリアル・インタフェース（UART6）に関するユーザ独自の初期化処理を行います。
UART6_Start	UART 通信を待機状態にします。
UART6_Stop	UART 通信を終了します。
UART6_SendData	データの UART 送信を開始します。
UART6_ReceiveData	データの UART 受信を開始します。
UART6_SendEndCallback	UART 送信完了割り込み INTST6 の発生に伴う処理を行います。
UART6_ReceiveEndCallback	UART 受信完了割り込み INTSR*6* の発生に伴う処理を行います。
UART6_SoftOverRunCallback	UART 受信完了割り込み INTSR6 の発生に伴う処理を行います。
UART6_ErrorCallback	UART 通信におけるエラー割り込み INTSRE6 の発生に伴う処理を行います。
CSI1n_Init	シリアル・インタフェース（CSI1*n*）の初期化処理を行います。
CSI1n_UserInit	シリアル・インタフェース（CSI1*n*）に関するユーザ独自の初期化処理を行います。
CSI1n_Start	CSI1*n* 通信を待機状態にします。
CSI1n_Stop	CSI1*n* 通信を終了します。
CSI1n_ReceiveData	データの CSI1*n* 受信を開始します。
CSI1n_SendReceiveData	データの CSI1*n* 送受信を開始します。
CSI1n_SendEndCallback	CSI1*n* 通信完了割り込み INTCSI1*n* の発生に伴う処理を行います。
CSI1n_ReceiveEndCallback	CSI1*n* 通信完了割り込み INTCSI1*n* の発生に伴う処理を行います。
IICA_Init	シリアル・インタフェース（IICA）の初期化処理を行います。
IICA_UserInit	シリアル・インタフェース（IICA）に関するユーザ独自の初期化処理を行います。
IICA_Stop	IICA 通信を終了します。
IICA_MasterSendStart	IICA マスタ送信を開始します。
IICA_MasterReceiveStart	IICA マスタ受信を開始します。
IICA_StopCondition	ストップ・コンディションを発生します。
IICA_MasterSendEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。
IICA_MasterReceiveEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。
IICA_MasterErrorCallback	IICA マスタ通信におけるエラーの検出に伴う処理を行います。
IICA_SlaveSendStart	IICA スレーブ送信を開始します。
IICA_SlaveReceiveStart	IICA スレーブ受信を開始します。
IICA_SlaveSendEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。

API 関数名	機能概要
IICA_SlaveReceiveEndCallback	IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。
IICA_SlaveErrorCallback	IICA スレーブ通信におけるエラーの検出に伴う処理を行います。
IICA_GetStopConditionCallback	IICA スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。

UART6_Init

【Ix2】【Kx2-L】

シリアル・インタフェース（UART6）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    UART6_Init ( void );
```

[引数]

なし

[戻り値]

なし

UART6_UserInit

【Ix2】【Kx2-L】

シリアル・インタフェース（UART6）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，UART6_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    UART6_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

UART6_Start

【Ix2】【Kx2-L】

UART 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    UART6_Start ( void );
```

[引数]

なし

[戻り値]

なし

UART6_Stop

【Ix2】【Kx2-L】

UART 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    UART6_Stop ( void );
```

[引数]

なし

[戻り値]

なし

UART6_SendData

【Ix2】【Kx2-L】

データの UART 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 UART 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. UART 送信を行う際には，本 API 関数の呼び出し以前に UART6_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   UART6_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

[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

備考 シリアル・インタフェース（UART6）が DALI モードで動作している場合，送信するデータの総数 *txnum* として設定可能な値は，1 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，4 バイトの固定長データを 1 回だけ UART 送信する際の例を示します。

【CG_main.c】

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

```
BOOL    gFlag;                          /* 送信完了フラグ */
void main ( void ) {
    UCHAR   txbuf[] = "ABCD";
    USHORT  txnum = 4;
    gFlag = 1;                          /* 送信完了フラグの初期化 */
    ……
    UART6_Start ();                     /* UART 通信の開始 */
    UART6_SendData ( &txbuf, txnum );   /* UART 送信の開始 */
    while ( gFlag );                    /* txnum 個の送信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                  /* 送信完了フラグ */
__interrupt void MD_INTST6 ( void ) {   /* 割り込み INTST6 発生時の割り込み処理 */
    if ( gUart6TxCnt > 0 ) {
        ……
    } else {
        UART6_SendEndCallback ();       /* コールバック・ルーチンの呼び出し */
    }
}

void UART6_SendEndCallback ( void ) {   /* 割り込み INTST6 発生時のコールバック・ルーチン */
    gFlag = 0;                          /* 送信完了フラグの設定 */
}
```

UART6_ReceiveData

【Ix2】【Kx2-L】

データの UART 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の UART 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 UART 受信は，本 API 関数の呼び出し後，UART6_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   UART6_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

[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

備考 シリアル・インタフェース（UART6）が DALI モードで動作している場合，受信するデータの総数 *rxnum* として設定可能な値は，2 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，4 バイトの固定長データを 1 回だけ UART 受信する際の例を示します。

【CG_main.c】

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

```
BOOL    gFlag;                                  /* 受信完了フラグ */
void main ( void ) {
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                                  /* 受信完了フラグの初期化 */
    ……
    UART6_ReceiveData ( &rxbuf, rxnum );        /* UART 受信の開始 */
    UART6_Start ();                             /* UART 通信の開始 */
    while ( gFlag );                            /* rxnum 個の受信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                          /* 受信完了フラグ */
__interrupt void MD_INTSR6 ( void ) {           /* 割り込み INTSR6 発生時の割り込み処理 */
    ……
    if ( gUart6RxLen > gUart6RxCnt ) {
        ……
        if ( gUart6RxLen == gUart6RxCnt ) {
            UART6_ReceiveEndCallback ();        /* コールバック・ルーチンの呼び出し */
        }
    }
}

void UART6_ReceiveEndCallback ( void ) {        /* 割り込み INTSR6 発生時のコールバック・ルーチン */
    gFlag = 0;                                  /* 受信完了フラグの設定 */
}
```

UART6_SendEndCallback

【Ix2】【Kx2-L】

UART 送信完了割り込み INTST6 の発生に伴う処理を行います。

備考　本 API 関数は，UART 送信完了割り込み INTST6 に対応した割り込み処理 MD_INTST6 のコールバック・ルーチン（UART6_SendData の引数 *txnum* で指定された数のデー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    UART6_SendEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

UART6_ReceiveEndCallback

【Ix2】【Kx2-L】

UART 受信完了割り込み INTSR6 の発生に伴う処理を行います。

備考 本 API 関数は，UART 受信完了割り込み INTSR6 に対応した割り込み処理 MD_INTSR6 のコールバック・ルーチン（UART6_ReceiveData の引数 *rxnum* で指定された数のデー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    UART6_ReceiveEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

UART6_SoftOverRunCallback

【Ix2】【Kx2-L】

UART 受信完了割り込み INTSR6 の発生に伴う処理を行います。

備考 本 API 関数は，UART 受信完了割り込み INTSR6 に対応した割り込み処理 MD_INTSR6 のコールバック・ルーチン（UART6_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]

- 【Ix2】

```
void    UART6_SoftOverRunCallback ( USHORT rx_data );
```

- 【Kx2-L】

```
void    UART6_SoftOverRunCallback ( UCHAR rx_data );
```

[引数]

I/O	引数	説明
O	USHORT *rx_data*;	受信したデータ（UART6_ReceiveData の引数 *rxnum* で指定された数以上に受信したデータ）
O	UCHAR *rx_data*;	受信したデータ（UART6_ReceiveData の引数 *rxnum* で指定された数以上に受信したデータ）

[戻り値]

なし

UART6_ErrorCallback

【Ix2】【Kx2-L】

UART 通信におけるエラー割り込み INTSRE6 の発生に伴う処理を行います。

備考 本 API 関数は，エラー割り込み INTSRE6 に対応した割り込み処理 MD_INTSRE6 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    UART6_ErrorCallback ( UCHAR err_type );
```

[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

[使用例]

以下に，エラー割り込みの発生要因別にコールバック処理を行う際の例を示します。

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
__interrupt void MD_INTSRE6 ( void ) {        /* 割り込み INTSRE6 発生時の割り込み処理 */
    UCHAR   err_type;
    ……
    UART6_ErrorCallback ( err_type );         /* コールバック・ルーチンの呼び出し */
}
```

```
void UART6_ErrorCallback ( UCHAR err_type ) { /* 割り込み INTSRE6 発生時のコールバック・ルーチン */
    if ( err_type & 0x1 ) {                   /* 発生要因の判別 */
        ……   /* オーバラン・エラーが発生した際のコールバック処理 */
    } else if ( err_type & 0x2 ) {            /* 発生要因の判別 */
        ……   /* フレーミング・エラーが発生した際のコールバック処理 */
    } else if ( err_type & 0x4 ) {            /* 発生要因の判別 */
        ……   /* パリティ・エラーが発生した際のコールバック処理 */
    }
}
```

CSI1*n*_Init

【IB2】【KB2-L】【KC2-L】

シリアル・インタフェース（CSI1*n*）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    CSI1n_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

CSI1*n*_UserInit

【IB2】【KB2-L】【KC2-L】

シリアル・インタフェース（CSI1*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，CSI1n_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    CSI1n_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

CSI1*n*_Start

【IB2】【KB2-L】【KC2-L】

CSI1*n* 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    CSI1n_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

CSI1*n*_Stop

【IB2】【KB2-L】【KC2-L】

CSI1*n* 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    CSI1n_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

CSI1*n*_ReceiveData

【IB2】【KB2-L】【KC2-L】

データの CSI1*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 CSI1*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. CSI1*n* 受信を行う際には，本 API 関数の呼び出し以前に CSI1n_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   CSI1n_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

[使用例]

以下に，チャネル 10 から 4 バイトの固定長データを 1 回だけ CSI10 受信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gFlag;                              /* 受信完了フラグ */
void main ( void ) {
```

```
    UCHAR   rxbuf[10];
    USHORT  rxnum = 4;
    gFlag = 1;                              /* 受信完了フラグの初期化 */
    ……
    CSI10_Start ();                         /* CSI10 通信の開始 */
    CSI10_ReceiveData ( &rxbuf, rxnum );    /* CSI10 受信の開始 */
    while ( gFlag );                        /* rxnum 個の受信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* 受信完了フラグ */
__interrupt void MD_INTCSI10 ( void ) {     /* 割り込み INTCSI10 発生時の割り込み処理 */
    if ( gCsi10RxCnt < gCsi10RxLen ) {
        ……
        if ( gCsi10RxCnt == gCsi10RxLen ) {
            CSI10_ReceiveEndCallback ();    /* コールバック・ルーチンの呼び出し */

        } else {
            ……
        }
    }
}

void CSI10_ReceiveEndCallback ( void ) {    /* 割り込み INTCSI10 発生時のコールバック・ルーチン */
    gFlag = 0;                              /* 受信完了フラグの設定 */
}
```

CSI1*n*_SendReceiveData

【IB2】【KB2-L】【KC2-L】

データの CSI1*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 CSI1*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 CSI1*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. CSI1*n* 送受信を行う際には，本 API 関数の呼び出し以前に CSI1n_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   CSI1n_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
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;	送受信するデータの総数
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

[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，チャネル 10 から 4 バイトの固定長データを 1 回だけ CSI10 送受信する際の例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
BOOL    gSflag;                                          /* 送信完了フラグ */
void main ( void ) {
    UCHAR   txbuf[] = "0123";
    USHORT  txnum = 4;
    UCHAR   rxbuf[10];
    gSflag = 1;                                          /* 送信完了フラグの初期化 */
    ……
    CSI10_Start ();                                      /* CSI10 通信の開始 */
    CSI10_SendReceiveData ( &txbuf, txnum, &rxbuf );     /* CSI10 送受信の開始 */
    while ( gSflag );                                    /* txnum 個の送受信待ち */
    ……
}
```

【CG_serial_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gSflag;                     /* 送信完了フラグ */
__interrupt void MD_INTCSI10 ( void ) {     /* 割り込み INTCSI10 発生時の割り込み処理 */
   if ( gCsi10TxCnt > 0 ) {
        ……
    } else {
        ……
        CSI10_SendEndCallback ();           /* コールバック・ルーチンの呼び出し */
    }
}

void CSI10_SendEndCallback ( void ) {       /* 割り込み INTCSI10 発生時のコールバック・ルーチン */
    gSflag = 0;                             /* 送信完了フラグの設定 */
}
```

CSI1*n*_SendEndCallback

【IB2】【KB2-L】【KC2-L】

CSI1*n* 通信完了割り込み INTCSI1*n* の発生に伴う処理を行います。

備考 本 API 関数は，CSI1*n* 通信完了割り込み INTCSI1*n* に対応した割り込み処理 MD_INTCSI1*n* のコールバック・ルーチン（CSI1n_SendReceiveData の引数 *txnum* で指定された数のデー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    CSI1n_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

CSI1*n*_ReceiveEndCallback

【IB2】【KB2-L】【KC2-L】

CSI1*n* 通信完了割り込み INTCSI1*n* の発生に伴う処理を行います。

備考 本 API 関数は，CSI1*n* 通信完了割り込み INTCSI1*n* に対応した割り込み処理 MD_INTCSI1*n* のコールバック・ルーチン（CSI1n_ReceiveData の引数 *rxnum* で指定された数のデー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    CSI1n_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

IICA_Init

【IA2】【IB2】【Kx2-L】

シリアル・インタフェース（IICA）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_Init ( void );
```

[引数]

なし

[戻り値]

なし

IICA_UserInit

【IA2】【IB2】【Kx2-L】

シリアル・インタフェース（IICA）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，IICA_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    IICA_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

IICA_Stop

【IA2】【IB2】【Kx2-L】

IICA 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterSendStart

【IA2】【IB2】【Kx2-L】

IICA 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 IICA 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   IICA_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum, UCHAR wait );
```

[引数]

I/O	引数	説明
I	UCHAR *adr*;	スレーブ・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

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バス通信状態
MD_ERROR2	バス未解放状態

IICA_MasterReceiveStart

【IA2】【IB2】【Kx2-L】

IICA 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 IICA 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   IICA_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum, UCHAR wait );
```

[引数]

I/O	引数	説明
I	UCHAR *adr*;	スレーブ・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

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	バス通信状態
MD_ERROR2	バス未解放状態

IICA_StopCondition

【IA2】【IB2】【Kx2-L】

ストップ・コンディションを発生します。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_StopCondition ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterSendEndCallback

【IA2】【IB2】【Kx2-L】

IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。

備考 本 API 関数は，IICA 通信完了割り込み INTIICA0 に対応した割り込み処理 MD_INTIICA0 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    IICA_MasterSendEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterReceiveEndCallback

【IA2】【IB2】【Kx2-L】

IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。

備考 本 API 関数は，IICA 通信完了割り込み INTIICA0 に対応した割り込み処理 MD_INTIICA0 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    IICA_MasterReceiveEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_MasterErrorCallback

【IA2】【IB2】【Kx2-L】

IICA マスタ通信におけるエラーの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

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

[引数]

I/O	引数	説明
I	MD_STATUS *flag*;	通信エラーの発生要因 MD_SPT : ストップ・コンディションの検出 MD_NACK : アクノリッジの未検出

[戻り値]

なし

IICA_SlaveSendStart

【IA2】【IB2】【Kx2-L】

IICA 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 IICA 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    IICA_SlaveSendStart ( UCHAR *txbuf, USHORT txnum );
```

[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

IICA_SlaveReceiveStart

【IA2】【IB2】【Kx2-L】

IICA 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 IICA 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    IICA_SlaveReceiveStart ( UCHAR *rxbuf, USHORT rxnum );
```

[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

IICA_SlaveSendEndCallback

【IA2】【IB2】【Kx2-L】

IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。

備考 本 API 関数は，IICA 通信完了割り込み INTIICA0 に対応した割り込み処理 MD_INTIICA0 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    IICA_SlaveSendEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_SlaveReceiveEndCallback

【IA2】【IB2】【Kx2-L】

IICA 通信完了割り込み INTIICA0 の発生に伴う処理を行います。

備考 本 API 関数は，IICA 通信完了割り込み INTIICA0 に対応した割り込み処理 MD_INTIICA0 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    IICA_SlaveReceiveEndCallback ( void );
```

[引数]

なし

[戻り値]

なし

IICA_SlaveErrorCallback

【IA2】【IB2】【Kx2-L】

IICA スレーブ通信におけるエラーの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

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

[引数]

I/O	引数	説明
I	MD_STATUS *flag*;	通信エラーの発生要因 MD_ERROR : アドレス不一致の検出 MD_NACK : アクノリッジの未検出

[戻り値]

なし

IICA_GetStopConditionCallback

【IA2】【IB2】【Kx2-L】

IICA スレーブ通信におけるストップ・コンディションの検出に伴う処理を行います。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICA_GetStopConditionCallback ( void );
```

[引数]

なし

[戻り値]

なし

3. 3. 5 オペアンプ

以下に，Applilet3 がオペアンプ用として出力する API 関数の一覧を示します。

表 3—6 オペアンプ用 API 関数

API 関数名	機能概要
OPAMP_Init	オペアンプ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。
OPAMP_UserInit	オペアンプに関するユーザ独自の初期化処理を行います。
PGA_Start	オペアンプ（PGA モード）の動作を開始します。
PGA_Stop	オペアンプ（PGA モード）の動作を停止します。
PGA_ChangePGAFactor	オペアンプ（PGA モード）における入力電圧の増幅率を設定します。
AMP_Start	オペアンプ（シングル・アンプ・モード）の動作を開始します。
AMP_Stop	オペアンプ（シングル・アンプ・モード）の動作を停止します。
AMPn_Start	オペアンプ*n*（シングル・アンプ・モード）の動作を開始します。
AMPn_Stop	オペアンプ*n*（シングル・アンプ・モード）の動作を停止します。

OPAMP_Init

【Ix2（オペアンプ搭載品）】【Kx2-L（オペアンプ搭載品）】

オペアンプ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_opamp.c

[指定形式]

```
void    OPAMP_Init ( void );
```

[引数]

なし

[戻り値]

なし

OPAMP_UserInit

【Ix2（オペアンプ搭載品）】【Kx2-L（オペアンプ搭載品）】

オペアンプ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，OPAMP_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_opamp_user.c

[指定形式]

```
void    OPAMP_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

PGA_Start

【Ix2（オペアンプ搭載品）】【Kx2-L（オペアンプ搭載品）】

オペアンプ（PGA モード）の動作を開始します。

[所属]

CG_opamp.c

[指定形式]

```
void    PGA_Start ( void );
```

[引数]

なし

[戻り値]

なし

PGA_Stop

【Ix2（オペアンプ搭載品）】【Kx2-L（オペアンプ搭載品）】

オペアンプ（PGA モード）の動作を停止します。

[所属]

CG_opamp.c

[指定形式]

```
void    PGA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

PGA_ChangePGAFactor

【Ix2（オペアンプ搭載品）】【Kx2-L（オペアンプ搭載品）】

オペアンプ（PGA モード）における入力電圧の増幅率を設定します。

備考 引数 *factor* に指定された値は，オペアンプ制御レジスタ（AMP0M）に設定されます。

[所属]

CG_opamp.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_opamp.h"
MD_STATUS   PGA_ChangePGAFactor ( enum PGAFactor factor );
```

[引数]

I/O	引数	説明
I	`enum PGAFactor factor;`	入力電圧の増幅率 PGAFACTOR0： 4 倍 PGAFACTOR1： 8 倍 PGAFACTOR2： 16 倍 PGAFACTOR3： 32 倍

[戻り値]

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

AMP_Start

【Ix2（オペアンプ搭載品）】

オペアンプ（シングル・アンプ・モード）の動作を開始します。

[所属]

CG_opamp.c

[指定形式]

```
void    AMP_Start ( void );
```

[引数]

なし

[戻り値]

なし

AMP_Stop

【Ix2（オペアンプ搭載品）】

オペアンプ（シングル・アンプ・モード）の動作を停止します。

[所属]

CG_opamp.c

[指定形式]

```
void    AMP_Stop ( void );
```

[引数]

なし

[戻り値]

なし

AMP*n*_Start

【Kx2-L（オペアンプ搭載品）】

オペアンプ *n*（シングル・アンプ・モード）の動作を開始します。

[所属]

CG_opamp.c

[指定形式]

```
void    AMPn_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

AMP*n*_Stop

【Kx2-L（オペアンプ搭載品）】

オペアンプ *n*（シングル・アンプ・モード）の動作を停止します。

[所属]

CG_opamp.c

[指定形式]

```
void    AMPn_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

3. 3. 6 コンパレータ

以下に，Applilet3 がコンパレータ用として出力する API 関数の一覧を示します。

表 3—7 コンパレータ用 API 関数

API 関数名	機能概要
Comparator_Init	コンパレ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。
Comparator_UserInit	コンパレータに関するユーザ独自の初期化処理を行います。
Comparatorn_Start	コンパレータ *n* の動作を開始します。
Comparatorn_Stop	コンパレータ *n* の動作を停止します。

Comparator_Init

【Ix2】

コンパレ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_comparator.c

[指定形式]

```
void    Comparator_Init ( void );
```

[引数]

なし

[戻り値]

なし

Comparator_UserInit

【lx2】

コンパレ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，Comparator_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_comparator_user.c

[指定形式]

```
void    Comparator_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

Comparator*n*_Start

【lx2】

コンパレータ *n* の動作を開始します。

[所属]

CG_comparator.c

[指定形式]

```
void    Comparatorn_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

Comparator*n*_Stop

【lx2】

コンパレータ *n* の動作を停止します。

[所属]

CG_comparator.c

[指定形式]

```
void    Comparatorn_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

3. 3. 7 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—8 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_ComparatorOn	電圧コンパレータを動作許可状態に設定します。
AD_ComparatorOff	電圧コンパレータを動作停止状態に設定します。
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_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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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_ComparatorOn

【Ix2】【Kx2-L】

電圧コンパレータを動作許可状態に設定します。

備考 電圧コンパレータが動作停止状態から動作許可状態へと移行した際，約 1μ 秒の安定時間を必要とします。
したがって，本 API 関数と AD_Start の間には，約 1μ 秒の時間を空ける必要があります。

[所属]

CG_ad.c

[指定形式]

```
void    AD_ComparatorOn ( void );
```

[引数]

なし

[戻り値]

なし

AD_ComparatorOff

【Ix2】【Kx2-L】

電圧コンパレータを動作停止状態に設定します。

[所属]

CG_ad.c

[指定形式]

```
void    AD_ComparatorOff ( void );
```

[引数]

なし

[戻り値]

なし

AD_Start

【Ix2】【Kx2-L】

A/D 変換を開始します。

備考 電圧コンパレータが動作停止状態から動作許可状態へと移行した際，約 1μ 秒の安定時間を必要とします。したがって，AD_ComparatorOn と本 API 関数の間には，約 1μ 秒の時間を空ける必要があります。

[所属]

CG_ad.c

[指定形式]

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

[引数]

なし

[戻り値]

なし

AD_Stop

【Ix2】【Kx2-L】

A/D 変換を終了します。

備考 電圧コンパレータは，本 API 関数の処理完了後も動作を継続しています。
したがって，電圧コンパレータの動作を停止する場合は，本 API 関数の処理完了後，AD_ComparatorOff を呼び出す必要があります。

[所属]

CG_ad.c

[指定形式]

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

[引数]

なし

[戻り値]

なし

AD_SelectADChannel

【Ix2】【Kx2-L】

A/D 変換するアナログ電圧の入力端子を設定します。

備考 引数 *channel* に指定された値は，アナログ入力チャネル指定レジスタ（ADS）に設定されます。

[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 ADCHANNELPGAIN : オペアンプ出力端子【Ix2】 ADCHANNEL12V : 内部電圧（1.2 V）【Ix2】

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_Read

【Ix2】【Kx2-L】

A/D 変換結果（10 ビット）を読み出します。

備考 10 ビット A/D 変換結果レジスタ（ADCR）の内容が引数 *buffer* で指定された領域に格納されます。

[所属]

CG_ad.c

[指定形式]

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

[引数]

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

[戻り値]

なし

AD_ReadByte

【Ix2】【Kx2-L】

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。

備考 8 ビット A/D 変換結果レジスタ H（ADCRH）の内容が引数 *buffer* で指定された領域に格納されます。

[所属]

CG_ad.c

[指定形式]

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

[引数]

I/O	引数	説明
O	UCHAR *buffer;	読み出した A/D 変換結果を格納する領域へのポインタ

[戻り値]

なし

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
TMX_Init	16 ビット・タイマ X*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。
TMXn_Start	16 ビット・タイマ X*n* のカウントを開始します。
TMXn_Stop	16 ビット・タイマ X*n* のカウントを終了します。
TMXn_ChangeDuty	TOX0*n* 端子にシングル出力する PWM 信号のデューティ比を変更します。
TMXn_ChangeDualDuty	TOX0*n* 端子にデュアル出力する PWM 信号のデューティ比を変更します。
TMX_EnableHighImpedanceState	16 ビット・タイマ X*n* のハイ・インピーダンス出力を開始します。
TMX_DisableHighImpedanceState	16 ビット・タイマ X*n* のハイ・インピーダンス出力を終了します。
TM00_Init	16 ビット・タイマ／イベント・カウンタ 00 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。
TM00_UserInit	16 ビット・タイマ／イベント・カウンタ 00 に関するユーザ独自の初期化処理を行います。
TM00_Start	16 ビット・タイマ／イベント・カウンタ 00 のカウントを開始します。
TM00_Stop	16 ビット・タイマ／イベント・カウンタ 00 のカウントを終了します。
TM00_ChangeTimerCondition	キャプチャ／コンペア・コントロール・レジスタ 00（CRC00）の内容を変更します。
TM00_GetFreeRunningValue	キャプチャ・レジスタ（CR0*n*0）の内容を獲得します。
TM00_SoftwareTriggerOn	ワンショット・パルス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。
TM00_ChangeDuty	TO00 端子に出力する信号のデューティ比を変更します。
TM00_GetPulseWidth	TI0*n*0 端子に対する入力信号（入力パルス）のハイ／ロウ・レベルの測定幅を獲得します。
TM5n_Init	8 ビット・タイマ／イベント・カウンタ 5*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。
TM5n_UserInit	8 ビット・タイマ／イベント・カウンタ 5*n* に関するユーザ独自の初期化処理を行います。
TM5n_Start	8 ビット・タイマ／イベント・カウンタ 5*n* のカウントを開始します。
TM5n_Stop	8 ビット・タイマ／イベント・カウンタ 5*n* のカウントを終了します。
TM5n_ChangeTimerCondition	8 ビット・タイマ・コンペア・レジスタ 5*n*（CR5*n*）の内容を変更します。
TM5n_ChangeDuty	TO5*n* 端子に出力する PWM 信号のデューティ比を変更します。
TMHn_Init	8 ビット・タイマ H*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。
TMHn_UserInit	8 ビット・タイマ H*n* に関するユーザ独自の初期化処理を行います。
TMHn_Start	8 ビット・タイマ H*n* のカウントを開始します。

API 関数名	機能概要
TMHn_Stop	8 ビット・タイマ H*n* のカウントを終了します。
TMHn_ChangeTimerCondition	8 ビット・タイマ H コンペア・レジスタ 0*n* ／ 1*n*（CMP0*n* ／ CMP1*n*）の内容を変更します。
TMHn_ChangeDuty	TOH*n* 端子に出力する PWM 信号のデューティ比を変更します。
TMH1_CarrierOutputEnable	8 ビット・タイマ H1（キャリア・ジェネレータ・モード）のキャリア・パルス出力を開始します。
TMH1_CarrierOutputDisable	8 ビット・タイマ H1（キャリア・ジェネレータ・モード）のキャリア・パルス出力を終了します。

TMX_Init

【Ix2】

16 ビット・タイマ X*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_timer.c

[指定形式]

```
void    TMX_Init ( void );
```

[引数]

なし

[戻り値]

なし

TMX*n*_Start

【Ix2】

16 ビット・タイマ X*n* 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（PWM 出力，A/D 変換スタート・タイミング信号出力など）により異なります。

[所属]

CG_timer.c

[指定形式]

```
void    TMXn_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

TMX*n*_Stop

【Ix2】

16 ビット・タイマ X*n* のカウントを終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TMXn_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

TMX*n*_ChangeDuty

【Ix2】

TOX*n* 端子にシングル出力する 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 X*n* をシングル出力用に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    TMXn_ChangeDuty ( UCHAR ratio );
```

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 *ratio*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *ratio* 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 ) {
    UCHAR   ratio = 25;
    ……
    TMX0_Start ();                  /* カウントの開始 */
    ……
    TMX0_ChangeDuty ( ratio );      /* デューティ比の変更 */
```

```
    ……
}
```

TMX*n*_ChangeDualDuty

【Ix2】

TOX*n* 端子にデュアル出力する 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 X*n* をデュアル出力用に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    TMXn_ChangeDualDuty ( UCHAR ratio, UCHAR ratio1, UCHAR delay );
```

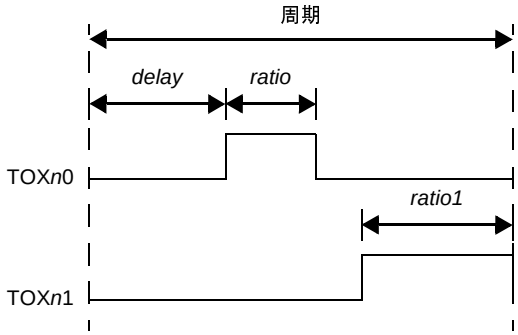
備考 *n* は，チャネル番号を意味します。

[引数]

I/O	引数	説明
I	UCHAR *ratio*;	TOX*n*0 のデューティ比（0 ～ 100，単位：%）
I	UCHAR *ratio1*;	TOX*n*1 のデューティ比（0 ～ 100，単位：%）
I	UCHAR *delay*;	TOX*n*0 の遅延間隔（0 ～ 100，単位：%）

備考 1. デューティ比 *ratio*，*ratio1*, *delay* に設定する値は，10 進数に限られます。

2. 各引数は，以下の意味を持ちます。



[戻り値]

なし

TMX_EnableHighImpedanceState

【Ix2】

16 ビット・タイマ X*n* のハイ・インピーダンス出力を開始します。

[所属]

CG_timer.c

[指定形式]

```
void    TMX_EnableHighImpedanceState ( void );
```

[引数]

なし

[戻り値]

なし

TMX_DisableHighImpedanceState

【Ix2】

16 ビット・タイマ X*n* のハイ・インピーダンス出力を終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TMX_DisableHighImpedanceState ( void );
```

[引数]

なし

[戻り値]

なし

TM00_Init

【Ix2】【Kx2-L】

16 ビット・タイマ／イベント・カウンタ 00 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    TM00_Init ( void );
```

[引数]

なし

[戻り値]

なし

TM00_UserInit

【Ix2】【Kx2-L】

16 ビット・タイマ／イベント・カウンタ 00 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，TM00_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    TM00_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

TM00_Start

【Ix2】【Kx2-L】

16 ビット・タイマ／イベント・カウンタ 00 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，方形波出力，外部イベント・カウンタなど）により異なります。

[所属]

CG_timer.c

[指定形式]

```
void    TM00_Start ( void );
```

[引数]

なし

[戻り値]

なし

TM00_Stop

【Ix2】【Kx2-L】

16 ビット・タイマ／イベント・カウンタ 00 のカウントを終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TM00_Stop ( void );
```

[引数]

なし

[戻り値]

なし

TM00_ChangeTimerCondition

【Ix2】【Kx2-L】

16 ビット・タイマ・キャプチャ／コンペア・コントロール・レジスタ 0*n*0（CR0*n*0）の内容を変更します。

備考 CR0*n*0 の内容を変更する際には，本 API 関数の呼び出し以前に TM00_Stop を呼び出す必要があります。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TM00_ChangeTimerCondition ( USHORT *array_reg, USHORT array_num );
```

[引数]

I/O	引数	説明
I	USHORT *array_reg;	変更対象レジスタに設定する値を格納した領域へのポインタ
I	USHORT array_num;	変更対象レジスタ 1： CR000 2： CR000，CR010

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	引数 *array_num* の指定が不正

TM00_GetFreeRunningValue

【Ix2】【Kx2-L】

キャプチャ・レジスタ（CR0*n*0）の内容を獲得します。

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 00 がフリー・ランニング・タイマ・モードで動作し，16 ビット・タイマ・キャプチャ／コンペア・コントロール・レジスタ 0*n*0（CRC0*n*0）がキャプチャ・レジスタとして使用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   TM00_GetFreeRunningValue ( ULONG *count, enum TMChannel channel );
```

[引数]

I/O	引数	説明
O	ULONG *count;	獲得した値を格納する領域へのポインタ
I	enum TMChannel channel;	獲得対象端子 TMCHANNEL0： TI000 端子 TMCHANNEL1： TI010 端子

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了 - CR0*n*0 がコンペア・レジスタとして動作しています。
MD_ARGERROR	引数の指定が不正

TM00_SoftwareTriggerOn

【Ix2】【Kx2-L】

ワンショット・パルス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 00 をワンショット・パルス出力用に使用している場合に限られます。

[所属]

CG_timer.c

[指定形式]

```
void    TM00_SoftwareTriggerOn ( void );
```

[引数]

なし

[戻り値]

なし

TM00_ChangeDuty

【Ix2】【Kx2-L】

TO00 端子に出力する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 00 を PPG 出力用に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    TM00_ChangeDuty ( UCHAR ratio );
```

[引数]

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

備考 デューティ比 *ratio* 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 ) {
    UCHAR   ratio = 25;
    ……
    TM00_Start ();                  /* カウントの開始 */
    ……
    TM00_ChangeDuty ( ratio );      /* デューティ比の変更 */
    ……
```

```
}
```

TM00_GetPulseWidth

【Ix2】【Kx2-L】

TI0*n*0 端子に対する入力信号（入力パルス）のハイ／ロウ・レベルの測定幅を獲得します。

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 00 をパルス幅測定用に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"
#include    "CG_timer.h"
void    TM00_GetPulseWidth ( ULONG *highwidth, ULONG *lowwidth, enum TMChannel channel );
```

[引数]

I/O	引数	説明
O	ULONG *highwidth;	ハイ・レベルの測定幅（0x0 ～ 0xffff）を格納する領域へのポインタ
O	ULONG *lowwidth;	ロウ・レベルの測定幅（0x0 ～ 0xffff）を格納する領域へのポインタ
I	enum TMChannel channel;	測定対象端子 TMCHANNEL0：TI000 端子 TMCHANNEL1：TI010 端子

[戻り値]

なし

TM5*n*_Init

【Ix2】【Kx2-L】

8 ビット・タイマ／イベント・カウンタ 5*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_timer.c

[指定形式]

```
void    TM5n_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

TM5*n*_UserInit

【Ix2】【Kx2-L】

8 ビット・タイマ／イベント・カウンタ 5*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，TM5n_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    TM5n_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

TM5*n*_Start

【Ix2】【Kx2-L】

8 ビット・タイマ／イベント・カウンタ 5*n* 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・カウンタなど）により異なります。

[所属]

CG_timer.c

[指定形式]

```
void    TM5n_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

TM5*n*_Stop

【Ix2】【Kx2-L】

8 ビット・タイマ／イベント・カウンタ 5*n* のカウントを終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TM5n_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

TM5*n*_ChangeTimerCondition

【Ix2】【Kx2-L】

8 ビット・タイマ・コンペア・レジスタ 5*n*（CR5*n*）の内容を変更します。

備考 CR5*n* の内容を変更する際には，本 API 関数の呼び出し以前に TM5n_Stop を呼び出す必要があります。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TM5n_ChangeTimerCondition ( 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*;	CR5*n* に設定する値

[戻り値]

なし

TM5*n*_ChangeDuty

【Kx2-L】

TO5*n* 端子に出力する 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，8 ビット・タイマ／イベント・カウンタ 5*n* を PWM 出力用に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    TM5n_ChangeDuty ( UCHAR ratio );
```

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 *ratio*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *ratio* 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 ) {
    UCHAR   ratio = 25;
    ……
    TM50_Start ();                  /* カウントの開始 */
    ……
```

```
    TM50_ChangeDuty ( ratio );      /* デューティ比の変更 */
    ……
}
```

TMH*n*_Init

【Ix2】【Kx2-L】

8 ビット・タイマ H*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_timer.c

[指定形式]

```
void    TMHn_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

TMH*n*_UserInit

【Ix2】【Kx2-L】

8 ビット・タイマ H*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，TMHn_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    TMHn_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

TMH*n*_Start

【Ix2】【Kx2-L】

8 ビット・タイマ H*n* 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，方形波出力，PWM 出力など）により異なります。

[所属]

CG_timer.c

[指定形式]

```
void    TMHn_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

TMH*n*_Stop

【Ix2】【Kx2-L】

8 ビット・タイマ H*n* のカウントを終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TMHn_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

TMH*n*_ChangeTimerCondition

【Ix2】【Kx2-L】

8 ビット・タイマ H コンペア・レジスタ 0*n* ／ 1*n*（CMP0*n* ／ CMP1*n*）の内容を変更します。

備考 CMP0*n* ／ CMP1*n* の内容を変更する際には，本 API 関数の呼び出し以前に TMHn_Stop を呼び出す必要があります。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_timer.h"
MD_STATUS   TMHn_ChangeTimerCondition ( UCHAR *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	UCHAR *array_reg;	変更対象レジスタに設定する値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： CMP0*n* 2： CMP0*n*，CMP1*n*

[戻り値]

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

TMH*n*_ChangeDuty

【Ix2】【Kx2-L】

TOH*n* 端子に出力する 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，8 ビット・タイマ H*n* を PWM 出力用に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    TMHn_ChangeDuty ( UCHAR ratio );
```

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 *ratio*;	デューティ比（0 ～ 100，単位：%）

備考 デューティ比 *ratio* 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 ) {
    UCHAR   ratio = 25;
    ……
    TMH0_Start ();                  /* カウントの開始 */
    ……
    TMH0_ChangeDuty ( ratio );      /* デューティ比の変更 */
```

```
    ……
}
```

TMH1_CarrierOutputEnable

【Ix2】【Kx2-L】

8 ビット・タイマ H1（キャリア・ジェネレータ・モード）のキャリア・パルス出力を開始します。

[所属]

CG_timer.c

[指定形式]

```
void    TMH1_CarrierOutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

TMH1_CarrierOutputDisable

【Ix2】【Kx2-L】

8 ビット・タイマ H1（キャリア・ジェネレータ・モード）のキャリア・パルス出力を終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TMH1_CarrierOutputDisable ( void );
```

[引数]

なし

[戻り値]

なし

3. 3. 9 ウォッチドッグ・タイマ

以下に，Applilet3 がウォッチドッグ・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 ウォッチドッグ・タイマ用 API 関数

API 関数名	機能概要
WDT_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。

WDT_Restart

【Ix2】【Kx2-L】

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_wdt.c

[指定形式]

```
void    WDT_Restart ( void );
```

[引数]

なし

[戻り値]

なし

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_PowerOff	リアルタイム・カウンタに対するクロック供給を停止します。
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 INTRTC 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_ConstPeriodInterruptCallback	定周期割り込み INTRTC の発生に伴う処理を行います。
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_AlarmInterruptCallback	アラーム割り込み INTRTC の発生に伴う処理を行います。
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 INTRTCI 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_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。

API 関数名	機能概要
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

【KC2-L】

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

【KC2-L】

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_PowerOff

【KC2-L】

リアルタイム・カウンタに対するクロック供給を停止します。

備考 本 API 関数の呼び出しにより，リアルタイム・カウンタはリセット状態へと移行します。
このため，本 API 関数の呼び出し後，制御レジスタ（リアルタイム・カウンタ・コントロール・レジスタ 0：RTCC0 など）への書き込みは無視されます。

[所属]

CG_rtc.c

[指定形式]

```
void    RTC_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

RTC_CounterEnable

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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

【KC2-L】

割り込み INTRTC 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 INTRTC 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 INTRTC 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

【KC2-L】

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_ConstPeriodInterruptCallback

【KC2-L】

定周期割り込み INTRTC の発生に伴う処理を行います。

備考 本 API 関数は，定周期割り込み INTRTC に対応した割り込み処理 MD_INTRTC 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_ConstPeriodInterruptCallback ( void );
```

[引数]

なし

[戻り値]

なし

RTC_AlarmEnable

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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]

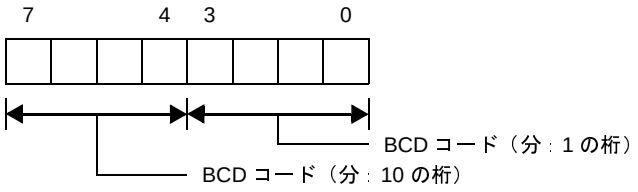
I/O	引数	説明
I	struct RTCAlarmValue *alarmval*;	アラームの発生条件（曜日，時，分）

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。



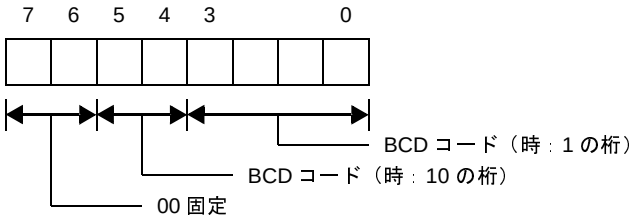
- Alarmwh（時）

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。

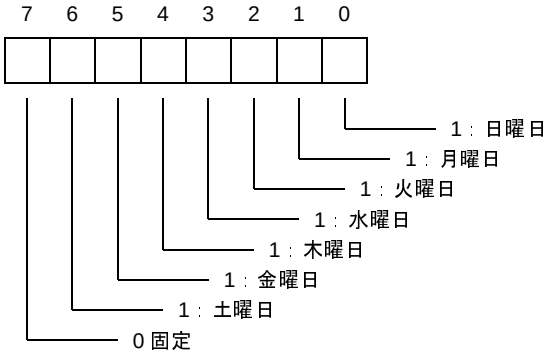
0： 午前

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

【KC2-L】

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_AlarmInterruptCallback

【KC2-L】

アラーム割り込み INTRTC の発生に伴う処理を行います。

備考 本 API 関数は，アラーム割り込み INTRTC に対応した割り込み処理 MD_INTRTC 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_AlarmInterruptCallback ( void );
```

[引数]

なし

[戻り値]

なし

RTC_IntervalStart

【KC2-L】

インターバル割り込み機能を開始します。

備考 割り込み INTRTCI 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

【KC2-L】

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

【KC2-L】

割り込み INTRTCI 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 INTRTCI 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 INTRTCI の発生周期 INTERVAL0： 2^6/fSUB INTERVAL1： 2^7/fSUB INTERVAL2： 2^8/fSUB INTERVAL3： 2^9/fSUB INTERVAL4： 2^10/fSUB INTERVAL5： 2^11/fSUB INTERVAL6： 2^12/fSUB

備考 fSUB 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

【KC2-L】

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_RTC1HZ_OutputEnable

【KC2-L】

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    RTC_RTC1HZ_OutputEnable ( void );
```

[引数]

なし

[戻り値]

なし

RTC_RTC1HZ_OutputDisable

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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

【KC2-L】

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 クロック出力

以下に，Applilet3 がクロック出力用として出力する API 関数の一覧を示します。

表 3—12 クロック出力用 API 関数

API 関数名	機能概要
PCL_Init	クロック出力制御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。
PCL_UserInit	クロック出力制御回路に関するユーザ独自の初期化処理を行います。
PCL_Start	クロック出力を開始します。
PCL_Stop	クロック出力を停止します。
PCL_ChangeFreq	PCL 端子への出力クロックを変更します。

PCL_Init

【KC2-L（48 ピン）】

クロック出力制御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_pcl.c

[指定形式]

```
void    PCL_Init ( void );
```

[引数]

なし

[戻り値]

なし

PCL_UserInit

【KC2-L（48 ピン）】

クロック出力制御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，PCL_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_pcl_user.c

[指定形式]

```
void    PCL_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

PCL_Start

【KC2-L（48 ピン）】

クロック出力を開始します。

[所属]

CG_pcl.c

[指定形式]

```
void    PCL_Start ( void );
```

[引数]

なし

[戻り値]

なし

PCL_Stop

【KC2-L（48 ピン）】

クロック出力を停止します。

[所属]

CG_pcl.c

[指定形式]

```
void    PCL_Stop ( void );
```

[引数]

なし

[戻り値]

なし

PCL_ChangeFreq

【KC2-L（48 ピン）】

PCL 端子への出力クロックを変更します。

備考 引数 *clock* に指定された値は，クロック出力選択レジスタ（CKS）に設定されます。

[所属]

CG_pcl.c

[指定形式]

```
#include    "CG_pclbuz.h"
MD_STATUS   PCL_ChangeFreq ( enum PCLclock clock );
```

[引数]

I/O	引数	説明
I	enum PCLclock *clock*;	出力クロックの種類 FPRS : fPRS FPRS2 : fPRS/2 FPRS4 : fPRS/4 FPRS8 : fPRS/8 FPRS16 : fPRS/16 FPRS32 : fPRS/2048 FPRS64 : fPRS/4096 FPRS128 : fPRS/8192 SUBCLOCK : fSUB

備考 fPRS はメイン・システム・クロックの周波数を，fSUB 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

3. 3. 12 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—13 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_Stop	低電圧検出動作を停止します。
LVI_SetLVILevel	低電圧検出レベルを設定します。

LVI_Init

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

【Ix2】【Kx2-L】

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

LVI_SetLVILevel

【Ix2】【Kx2-L】

低電圧検出レベルを設定します。

備考 1. 低電圧検出レベルを変更する際には，本 API 関数の呼び出し以前に LVI_Stop を呼び出す必要があります。

2. 引数 *level* に指定された値は，低電圧検出レベル選択レジスタ（LVIS）に設定されます。

[所属]

CG_lvi.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_lvi.h"
MD_STATUS   LVI_SetLVILevel ( enum LVILevel level );
```

[引数]

I/O	引数	説明
I	enum LVILevel *level*;	低電圧検出対象の電圧レベル LVILEVEL0： 4.22 V ± 0.1 V LVILEVEL1： 4.07 V ± 0.1 V LVILEVEL2： 3.92 V ± 0.1 V LVILEVEL3： 3.76 V ± 0.1 V LVILEVEL4： 3.61 V ± 0.1 V LVILEVEL5： 3.45 V ± 0.1 V LVILEVEL6： 3.30 V ± 0.1 V LVILEVEL7： 3.15 V ± 0.1 V LVILEVEL8： 2.99 V ± 0.1 V LVILEVEL9： 2.84 V ± 0.1 V LVILEVEL10： 2.68 V ± 0.1 V LVILEVEL11： 2.53 V ± 0.1 V LVILEVEL12： 2.38 V ± 0.1 V LVILEVEL13： 2.22 V ± 0.1 V LVILEVEL14： 2.07 V ± 0.1 V LVILEVEL15： 1.91 V ± 0.1 V

備考 LVILEVEL10 ～ LVILEVEL15 の指定は，対象デバイスが 78K0/Kx2-L の場合に限られます。

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR	異常終了 - 低電圧検出対象が外部入力端子からの外部入力電圧（EXLVI）
MD_ARGERROR	引数の指定が不正

備考 戻り値 MD_ERROR は，対象デバイスが 78K0/KB2-L，78K0/KC2-L の場合に限られます。

付録A 索 引

【わ行】

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2010.07.01	―	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