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

78K0R マイクロコントローラ

資料番号　ZUD-CD-10-0164（第 1 版）
発行年月　July 2010

© RENESAS Electronics Corporation 2010

［メ　　モ］

Applilet および MINICUBE は，ルネサス エレクトロニクス株式会社の日本およびその他の国における登録商標または商標です。
IAR Embedded Workbench は，IAR Systems AB の登録商標または商標です。
MULTI は，米国 Green Hill Software, Inc. の米国およびその他の国における登録商標または商標です。
Intel および Pentium は，米国 Intel Corporation の米国およびその他の国における登録商標または商標です。
Microsoft，Windows，Windows Vista および.NET Framework は，米国 Microsoft Corporation の米国，日本およびその他の国における登録商標または商標です。
その他，この資料に記載されている会社名，製品名などは，各社の商標または登録商標です。

- 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
	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	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_system_user.c	CLOCK_UserInit CG_ReadResetSource
	CG_system.h	―
外部バス	CG_bus.c	BUS_Init

周辺機能	ファイル名	含まれる API 関数名
外部バス	CG_bus.c	BUS_PowerOff
	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_INTP*n* MD_INTKR INTP_UserInit KEY_UserInit
	CG_int.h	–
シリアル	CG_serial.c	SAUm_Init SAUm_PowerOff UARTn_Init UARTn_Start UARTn_Stop UARTn_SendData UARTn_ReceiveData CSImn_Init CSImn_Start CSImn_Stop CSImn_SendData CSImn_ReceiveData CSImn_SendReceiveData IICmn_Init IICmn_Stop IICmn_MasterSendStart IICmn_MasterReceiveStart IICmn_StartCondition IICmn_StopCondition UARTFn_Init UARTFn_PowerOff UARTFn_Start UARTFn_Stop

周辺機能	ファイル名	含まれる API 関数名
シリアル	CG_serial.c	UARTFn_SendData UARTFn_ReceiveData UARTFn_SetComparisonData UARTFn_DataComparisonEnable UARTFn_DataComparisonDisable IICA_Init IICA_PowerOff IICA_Stop IICA_MasterSendStart IICA_MasterReceiveStart IICA_StopCondition IICA_SlaveSendStart IICA_SlaveReceiveStart IICn_Init IICn_Stop IICn_MasterSendStart IICn_MasterReceiveStart IICn_SlaveSendStart IICn_SlaveReceiveStart
	CG_serial_user.c	MD_INTSR*n* MD_INTSRE*n* MD_INTST*n* MD_INTCSI*mn* MD_INTIIC*mn* MD_INTLT*n* MD_INTLR*n* MD_INTLS*n* MD_INTIIC*n* MD_INTIICA SAUm_UserInit UARTn_SendEndCallback UARTn_ReceiveEndCallback UARTn_SoftOverRunCallback UARTn_ErrorCallback CSImn_SendEndCallback CSImn_ReceiveEndCallback CSImn_ErrorCallback IICmn_MasterSendEndCallback IICmn_MasterReceiveEndCallback IICmn_MasterErrorCallback UARTFn_SendEndCallback UARTFn_ReceiveEndCallback UARTFn_SoftOverRunCallback UARTFn_ExpBitCetectCallback UARTFn_IDMatchCallback

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	UARTFn_ErrorCallback IICA_UserInit IICA_MasterSendEndCallback IICA_MasterReceiveEndCallback IICA_MasterErrorCallback IICA_SlaveSendEndCallback IICA_SlaveReceiveEndCallback IICA_SlaveErrorCallback IICA_GetStopConditionCallback IICn_UserInit IICn_MasterSendEndCallback IICn_MasterReceiveEndCallback IICn_MasterErrorCallback IICn_SlaveSendEndCallback IICn_SlaveReceiveEndCallback IICn_SlaveErrorCallback IICn_GetStopConditionCallback
	CG_serial.h	–
オペアンプ	CG_opamp.c	OPAMP_Init AMPn_Start AMPn_Stop
	CG_opamp_user.c	OPAMP_UserInit
	CG_opamp.h	–
コンパレータ／PG アンプ	CG_cmppga.c	CMPPGA_Init CMPPGA_PowerOff CMPPGA_Start CMPPGA_Stop CMPPGA_ChangeCMPnRefVoltage CMPPGA_ChangePGAFactor
	CG_cmppga_user.c	MD_INTCMP*n* CMPPGA_UserInit
	CG_cmppga.h	–
A/D コンバータ	CG_ad.c	AD_Init AD_UserInit AD_PowerOff AD_ComparatorOn AD_ComparatorOff AD_Start AD_Stop AD_SelectADChannel AD_Read AD_ReadByte
	CG_ad_user.c	MD_INTAD

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_user.c	AD_UserInit
	CG_ad.h	—
D/A コンバータ	CG_da.c	DA_Init DA_PowerOff DAn_Start DAn_Stop DAn_SetValue DAn_Set8BitsValue DAn_Set12BitsValue
	CG_da_user.c	DA_UserInit
	CG_ds.h	—
タイマ	CG_timer.c	TAUm_Init TAUm_PowerOff TAUm_Channeln_Start TAUm_Channeln_Stop TAUm_Channeln_ChangeCondition TAUm_Channeln_ChangeTimerCondition TAUm_Channeln_GetPulseWidth TAUm_Channeln_ChangeDuty TAUm_Channeln_SoftWareTriggerOn
	CG_timer_user.c	MD_INTTM*mn* TAUm_UserInit
	CG_timer.h	—
ウォッチドッグ・タイマ	CG_wdt.c	WDT_Init WDT_Restart
	CG_wdt_user.c	WDT_UserInit
	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 RTC_AlarmGet RTC_AlarmInterruptCallback RTC_IntervalStart

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_IntervalStop RTC_IntervalInterruptEnable RTC_IntervalInterruptDisable RTC_RTC1HZ_OutputEnable RTC_RTC1HZ_OutputDisable RTC_RTCCL_OutputEnable RTC_RTCCL_OutputDisable RTC_RTCCL_OutputDisable RTC_RTCDIV_OutputEnable RTC_RTCDIV_OutputDisable
	CG_rtc_user.c	MD_INTRTC MD_INTRTCI RTC_UserInit RTC_ConstPeriodInterruptCallback RTC_ChangeCorrectionValue
	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	–
クロック出力／ブザー出力	CG_pclbuz.c	PCLBUZn_Init PCLBUZn_Start PCLBUZn_Stop PCLBUZn_ChangeFreq
	CG_pclbuz_user.c	PCLBUZn_UserInit
	CG_pclbuz.h	–
LCD コントローラ／ドライバ	CG_lcd.c	LCD_Init LCD_DisplayOn LCD_DisplayOff LCD_VoltageOn LCD_VoltageOff
	CG_lcd_user.c	LCD_UserInit
	CG_lcd.h	–
DMA コントローラ	CG_dma.c	DMAn_Init DMAn_Enable DMAn_Disable DMAn_Hold DMAn_Restart DMAn_CheckStatus DMAn_SetData

周辺機能	ファイル名	含まれる API 関数名
DMA コントローラ	CG_dma.c	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_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。
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。
	BUS_PowerOff	外部バス・インタフェースに対するクロック供給を停止します。
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	SAUm_Init	シリアル・アレイ・ユニット，およびシリアル・インタフェー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。

周辺機能	API 関数名	機能概要
シリアル	SAUm_UserInit	シリアル・アレイ・ユニット，およびシリアル・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。
	SAUm_PowerOff	シリアル・アレイ・ユニットに対するクロック供給を停止します。
	UARTn_Init	シリアル・インタフェース（UART）用チャネルの初期化処理を行います。
	UARTn_Start	UART 通信を待機状態にします。
	UARTn_Stop	UART 通信を終了します。
	UARTn_SendData	データの UART 送信を開始します。
	UARTn_ReceiveData	データの UART 受信を開始します。
	UARTn_SendEndCallback	UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。
	UARTn_ReceiveEndCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
	UARTn_SoftOverRunCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
	UARTn_ErrorCallback	UART 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
	CSImn_Init	シリアル・インタフェース（CSI）用チャネルの初期化処理を行います。
	CSImn_Start	CSI 通信を待機状態にします。
	CSImn_Stop	CSI 通信を終了します。
	CSImn_SendData	データの CSI 送信を開始します。
	CSImn_ReceiveData	データの CSI 受信を開始します。
	CSImn_SendReceiveData	データの CSI 送受信を開始します。
	CSImn_SendEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
	CSImn_ReceiveEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
	CSImn_ErrorCallback	CSI 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
	IICmn_Init	シリアル・インタフェース（簡易 IIC）用チャネルの初期化処理を行います。
	IICmn_Stop	簡易 IIC 通信を終了します。
	IICmn_MasterSendStart	簡易 IIC マスタ送信を開始します。
	IICmn_MasterReceiveStart	簡易 IIC マスタ受信を開始します。
	IICmn_StartCondition	スタート・コンディションを発生します。
	IICmn_StopCondition	ストップ・コンディションを発生します。

周辺機能	API 関数名	機能概要
シリアル	IICmn_MasterSendEndCallback	簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
	IICmn_MasterReceiveEndCallback	簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
	IICmn_MasterErrorCallback	簡易 IIC 通信におけるパリティ・エラー（ACK エラー）の検出に伴う処理を行います。
	UARTFn_Init	シリアル・インタフェース（UARTF*n*）の初期化処理を行います。
	UARTFn_PowerOff	シリアル・インタフェース（UARTF*n*）に対するクロック供給を停止します。
	UARTFn_Start	UARTF 通信を待機状態にします。
	UARTFn_Stop	UARTF 通信を終了します。
	UARTFn_SendData	データの UARTF 送信を開始します。
	UARTFn_ReceiveData	データの UARTF 受信を終了します。
	UARTFn_SetComparisonData	受信データと比較するデータを設定します。
	UARTFn_DataComparisonEnable	データの比較を開始します。
	UARTFn_DataComparisonDisable	データの比較を終了します。
	UARTFn_SendEndCallback	送信割り込み INTLT*n* の発生に伴う処理を行います。
	UARTFn_ReceiveEndCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
	UARTFn_SoftOverRunCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
	UARTFn_ExpBitCetectCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
	UARTFn_IDMatchCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
	UARTFn_ErrorCallback	ステータス割り込み INTLS*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_PowerOff	シリアル・インタフェース（IICA）に対するクロック供給を停止します。
	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。

周辺機能	API 関数名	機能概要
シリアル	IICA_MasterSendEndCallback	IICA 通信完了割り込み INTIICA の発生に伴う処理を行います。
	IICA_MasterReceiveEndCallback	IICA 通信完了割り込み INTIICA 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 INTIICA の発生に伴う処理を行います。
	IICA_SlaveReceiveEndCallback	IICA 通信完了割り込み INTIICA 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。
	IICn_Init	シリアル・インタフェース（IIC*n*）の初期化処理を行います。
	IICn_UserInit	シリアル・インタフェース（IIC*n*）に関するユーザ独自の初期化処理を行います。
	IICn_Stop	IIC*n* 通信を終了します。
	IICn_MasterSendStart	IIC*n* マスタ送信を開始します。
	IICn_MasterReceiveStart	IIC*n* マスタ受信を開始します。
	IICn_MasterSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_MasterReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_MasterErrorCallback	IIC*n* 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。
	IICn_SlaveSendStart	IIC*n* スレーブ送信を開始します。
	IICn_SlaveReceiveStart	IIC*n* スレーブ受信を開始します。
	IICn_SlaveSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_SlaveReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
	IICn_SlaveErrorCallback	IIC*n* 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。
	IICn_GetStopConditionCallback	IIC*n* 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。

周辺機能	API 関数名	機能概要
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。
	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／PG アンプ	CMPPGA_Init	コンパレータ／プログラマブル・ゲイン・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。
	CMPPGA_UserInit	コンパレータ／プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。
	CMPPGA_PowerOff	コンパレータ／プログラマブル・ゲイン・アンプに対するクロック供給を停止します。
	CMPPGA_Start	コンパレータ／プログラマブル・ゲイン・アンプの動作を開始します。
	CMPPGA_Stop	コンパレータ／プログラマブル・ゲイン・アンプの動作を停止します。
	CMPPGA_ChangeCMPnRefVoltage	コンパレータ *n* の内蔵基準電圧を設定します。
	CMPPGA_ChangePGAFactor	プログラマブル・ゲイン・アンプにおける入力電圧の増幅率を設定します。
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_PowerOff	A/D コンバータに対するクロック供給を停止します。
	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。
D/A コンバータ	DA_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。

周辺機能	API 関数名	機能概要
D/A コンバータ	DA_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。
	DA_PowerOff	D/A コンバータに対するクロック供給を停止します。
	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。
	DAn_Set8BitsValue	ANO*n* 端子に出力するアナログ電圧値（8 ビット）を設定します。
	DAn_Set12BitsValue	ANO*n* 端子に出力するアナログ電圧値（12 ビット）を設定します。
タイマ	TAUm_Init	タイマ・アレイ・ユニッ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。
	TAUm_UserInit	タイマ・アレイ・ユニットに関するユーザ独自の初期化処理を行います。
	TAUm_PowerOff	タイマ・アレイ・ユニットに対するクロック供給を停止します。
	TAUm_Channeln_Start	チャネル *n* のカウントを開始します。
	TAUm_Channeln_Stop	チャネル *n* のカウントを終了します。
	TAUm_Channeln_ChangeCondition	カウント値を変更します。
	TAUm_Channeln_ChangeTimerCondition	カウント値を変更します。
	TAUm_Channeln_GetPulseWidth	TI*mn* 端子に対する入力信号（入力パルス）のパルス間隔，またはハイ／ロウ・レベルの測定幅を獲得します。
	TAUm_Channeln_ChangeDuty	TO*mn* 端子に出力する PWM 信号のデューティ比を変更します。
	TAUm_Channeln_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。
ウォッチドッグ・タイマ	WDT_Init	ウォッチドッグ・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。
	WDT_UserInit	ウォッチドッグ・タイマに関するユーザ独自の初期化処理を行います。
	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。

周辺機能	API 関数名	機能概要
リアルタイム・カウンタ	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。
	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。

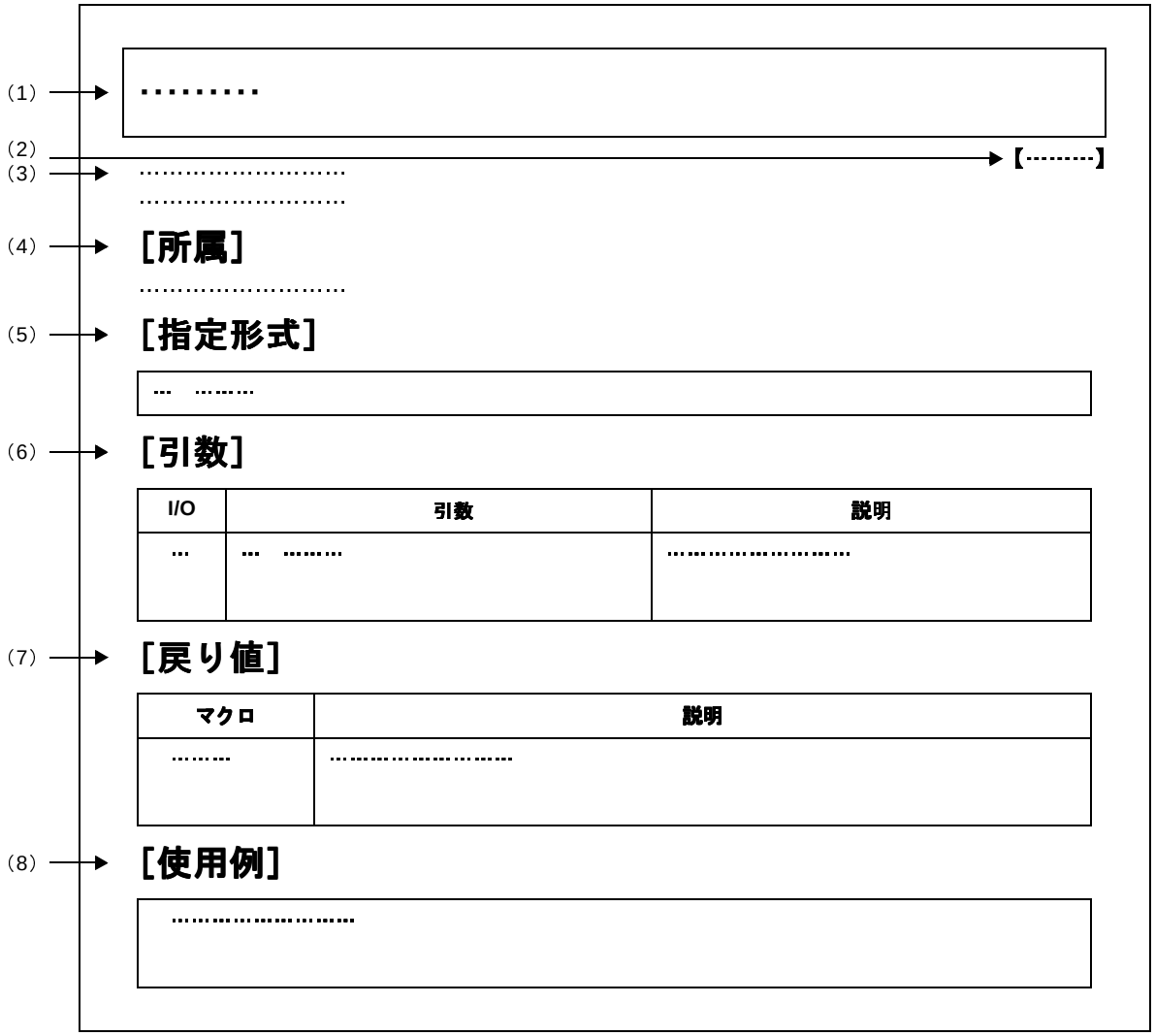
周辺機能	API 関数名	機能概要
リアルタイム・カウンタ	RTC_RTCDIV_OutputDisable	RTCDIV 端子に対するリアルタイム・カウンタ・クロック（32 kHz 分周）の出力を禁止します。
	RTC_ChangeCorrectionValue	時計誤差を補正するタイミング，および補正値を変更します。
クロック出力	PCL_Init	クロック出力制御回路の機能を制御するうえで必要となる初期化処理を行います。
	PCL_UserInit	クロック出力制御回路に関するユーザ独自の初期化処理を行います。
	PCL_Start	クロック出力を開始します。
	PCL_Stop	クロッ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。
クロック出力／ブザー出力	PCLBUZn_Init	クロック出力／ブザー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。
	PCLBUZn_UserInit	クロック出力／ブザー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。
	PCLBUZn_Start	クロック出力／ブザー出力を開始します。
	PCLBUZn_Stop	クロック出力／ブザー出力を停止します。
	PCLBUZn_ChangeFreq	PCLBUZ*n* 端子への出力クロックを変更します。
LCD コントローラ／ドライバ	LCD_Init	LCD コントローラ／ドライバ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。
	LCD_UserInit	LCD コントローラ／ドライバに関するユーザ独自の初期化処理を行います。
	LCD_DisplayOn	LCD コントローラ／ドライバを表示オン状態にします。
	LCD_DisplayOff	LCD コントローラ／ドライバを表示オフ状態にします。
	LCD_VoltageOn	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作許可としたのち，非選択信号をセグメント端子から出力します。
	LCD_VoltageOff	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，グランド・レベルの信号をセグメント端子／コモン端子に出力します。
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_Hold	DMA 起動要求を保留します。

周辺機能	API 関数名	機能概要
DMA コントローラ	DMAn_Restart	DMA 起動要求の保留を解除します。
	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 動作許可状態の際，DMA 転送を開始します。
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_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。

CLOCK_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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：WDRF 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 */
    } else if ( flag & 0x80 ) { /* 発生要因の判別：TRAP フラグのチェック */
        ……    /* 不正命令の実行による内部リセット要求に対応した処理 */
    }
```

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

CG_ChangeClockMode

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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／周辺ハードウエア・クロックの種類 【Fx3】 HIOCLK : 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック FILCLK : 低速内蔵発振クロック 【Ix3】 HIOCLK : 高速内蔵発振クロック HIO40CLK : 40MHz 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBCLK : サブシステム・クロック 【Kx3】 HIOCLK : 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBCLK : サブシステム・クロック 【Kx3-A】【Kx3-L】【Lx3】 HIOCLK : 高速内蔵発振クロック HIO20CLK : 20MHz 高速内蔵発振クロック SYSX1CLK : X1 クロック SYSEXTCLK : 外部メイン・システム・クロック SUBCLK : サブシステム・クロック

[戻り値]

マクロ	説明
MD_OK	正常終了
MD_ERROR1	異常終了【Fx3】【Kx3】 - X1 クロックへの変更はできません。 異常終了【Ix3】 - 40MHz 高速内蔵発振クロックへの変更はできません。 異常終了【Kx3-A】【Kx3-L】【Lx3】 - 20MHz 高速内蔵発振クロックへの変更はできません。
MD_ERROR2	異常終了【Fx3】【Kx3】 - 外部メイン・システム・クロックへの変更はできません。 異常終了【Ix3】【Kx3-A】【Kx3-L】【Lx3】 - X1 クロックへの変更はできません。
MD_ERROR3	異常終了【Fx3】 - 低速内蔵発振クロックへの変更はできません。 異常終了【Ix3】【Kx3-A】【Kx3-L】【Lx3】 - 外部メイン・システム・クロックへの変更はできません。 異常終了【Kx3】 - XT1，XT2 端子が入力モードのため，サブシステム・クロックへの変更はできません。
MD_ERROR4	異常終了【Ix3】【Kx3-A】【Kx3-L】【Lx3】 - サブシステム・クロックへの変更はできません。
MD_ARGERROR	引数の指定が不正

CG_ChangeFrequency

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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 【Fx3】 SYSTEMCLOCK : fPLL SYSONEHALF : fPLL/2 SYSONEFOURTH : fPLL/4 SYSONEEIGHTH : fPLL/8 SYSONESIXTEENTH : fPLL/16 SYSONETHIRTYSECOND : fPLL/32 【Ix3】【Kx3】【Kx3-L】 SYSTEMCLOCK : fMAIN SYSONEHALF : fMAIN/2 SYSONEFOURTH : fMAIN/4 SYSONEEIGHTH : fMAIN/8 SYSONESIXTEENTH : fMAIN/16 SYSONETHIRTYSECOND : fMAIN/32 【Kx3-A】【Lx3】 SYSTEMCLOCK : fMAIN SYSONEHALF : fMAIN/2 SYSONEFOURTH : fMAIN/4 SYSONEEIGHTH : fMAIN/8 SYSONESIXTEENTH : fMAIN/16 SYSONETHIRTYSECOND : fMAIN/32 SUB : fSUB SUBONEHALF : fSUB/2

備考 fPLL は PLL クロックの周波数を，fMAIN 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

CG_SelectPowerSaveMode

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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 - CPU がサブシステム・クロック（XT1 発振回路）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

[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;
```

```
    ……
    TAU0_PowerOff ();                           /* クロック供給の停止 */
    ret = CG_SelectPowerSaveMode ( PSSTOP );    /* STOP モードへの移行 */
    if ( ret != MD_OK ) {
        while ( 1 );
    }
    TAU0_Init ();                               /* タイマ・アレイ・ユニットの初期化 */
    TAU0_Channel0_Start ();                     /* チャネル 0 のカウント開始 */
    ……
}
```

CG_SelectStabTime

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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 : 2^8/fx STLEVEL1 : 2^9/fx STLEVEL2 : 2^10/fx STLEVEL3 : 2^11/fx STLEVEL4 : 2^13/fx STLEVEL5 : 2^15/fx STLEVEL6 : 2^17/fx STLEVEL7 : 2^18/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

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_PowerOff	外部バス・インタフェースに対するクロック供給を停止します。

BUS_Init

【Kx3】

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

【Kx3】

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

BUS_PowerOff

【Kx3】

外部バス・インタフェース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，外部バス・インタフェース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（メモリ拡張モード制御レジスタ：MEM など）への書き込みは無視されます。

[所属]

CG_bus.c

[指定形式]

```
void    BUS_PowerOff ( 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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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／ TTL 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，TTL 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，TTL 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，TTL 入力バッファ：あり

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

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 *enablettl*;	入力バッファの種類 MD_TRUE： TTL 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，TTL 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，TTL 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，TTL 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： TTL 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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

対象デバイスが 78K0R/Fx3 の場合，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，スロー・モードの指定を行う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，スロー・モード：なし【Fx3】

```
#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，スロー・モード：なし【Fx3】

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

- N-ch オープン・ドレーン出力：なし，スロー・モード：あり【Fx3】

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

- N-ch オープン・ドレーン出力：あり，スロー・モード：あり【Fx3】

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

また，対象デバイスが 78K0R/Ix3，78K0R/Kx3，78K0R/Kx3-A，78K0R/Kx3-L，または 78K0R/Lx3 の場合，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【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

```
#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【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_macrodriver.h"
void    PORT_ChangePmnOutput ( 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（VDD 耐圧）モード MD_FALSE： 通常出力モード
I	BOOL *enableslow*;	出力モードの種類 MD_TRUE： スロー・モード MD_FALSE： 通常モード
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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Kx3】【Kx3-L】【Lx3】

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

【Fx3】【Kx3】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

```
#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]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

  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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Kx3】【Kx3-L】【Lx3】

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

【Fx3】【Kx3】【Kx3-L】【Lx3】

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
SAUm_Init	シリアル・アレイ・ユニット，およびシリアル・インタフェー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。
SAUm_UserInit	シリアル・アレイ・ユニット，およびシリアル・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。
SAUm_PowerOff	シリアル・アレイ・ユニットに対するクロック供給を停止します。
UARTn_Init	シリアル・インタフェース（UART）用チャネルの初期化処理を行います。
UARTn_Start	UART 通信を待機状態にします。
UARTn_Stop	UART 通信を終了します。
UARTn_SendData	データの UART 送信を開始します。
UARTn_ReceiveData	データの UART 受信を開始します。
UARTn_SendEndCallback	UART 送信完了割り込み INTST*n* の発生に伴う処理を行います。
UARTn_ReceiveEndCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
UARTn_SoftOverRunCallback	UART 受信完了割り込み INTSR*n* の発生に伴う処理を行います。
UARTn_ErrorCallback	UART 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
CSImn_Init	シリアル・インタフェース（CSI）用チャネルの初期化処理を行います。
CSImn_Start	CSI 通信を待機状態にします。
CSImn_Stop	CSI 通信を終了します。
CSImn_SendData	データの CSI 送信を開始します。
CSImn_ReceiveData	データの CSI 受信を開始します。
CSImn_SendReceiveData	データの CSI 送受信を開始します。
CSImn_SendEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
CSImn_ReceiveEndCallback	CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。
CSImn_ErrorCallback	CSI 通信におけるエラー割り込み INTSRE*n* の発生に伴う処理を行います。
IICmn_Init	シリアル・インタフェース（簡易 IIC）用チャネルの初期化処理を行います。
IICmn_Stop	簡易 IIC 通信を終了します。
IICmn_MasterSendStart	簡易 IIC マスタ送信を開始します。
IICmn_MasterReceiveStart	簡易 IIC マスタ受信を開始します。
IICmn_StartCondition	スタート・コンディションを発生します。
IICmn_StopCondition	ストップ・コンディションを発生します。
IICmn_MasterSendEndCallback	簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
IICmn_MasterReceiveEndCallback	簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。
IICmn_MasterErrorCallback	簡易 IIC 通信におけるパリティ・エラー（ACK エラー）の検出に伴う処理を行います。

API 関数名	機能概要
UARTFn_Init	シリアル・インタフェース（UARTF*n*）の初期化処理を行います。
UARTFn_PowerOff	シリアル・インタフェース（UARTF*n*）に対するクロック供給を停止します。
UARTFn_Start	UARTF 通信を待機状態にします。
UARTFn_Stop	UARTF 通信を終了します。
UARTFn_SendData	データの UARTF 送信を開始します。
UARTFn_ReceiveData	データの UARTF 受信を終了します。
UARTFn_SetComparisonData	受信データと比較するデータを設定します。
UARTFn_DataComparisonEnable	データの比較を開始します。
UARTFn_DataComparisonDisable	データの比較を終了します。
UARTFn_SendEndCallback	送信割り込み INTLT*n* の発生に伴う処理を行います。
UARTFn_ReceiveEndCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
UARTFn_SoftOverRunCallback	受信完了割り込み INTLR*n* の発生に伴う処理を行います。
UARTFn_ExpBitCetectCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
UARTFn_IDMatchCallback	ステータス割り込み INTLS*n* の発生に伴う処理を行います。
UARTFn_ErrorCallback	ステータス割り込み INTLS*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_PowerOff	シリアル・インタフェース（IICA）に対するクロック供給を停止します。
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 INTIICA の発生に伴う処理を行います。
IICA_MasterReceiveEndCallback	IICA 通信完了割り込み INTIICA 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 INTIICA の発生に伴う処理を行います。
IICA_SlaveReceiveEndCallback	IICA 通信完了割り込み INTIICA 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。
IICn_Init	シリアル・インタフェース（IIC*n*）の初期化処理を行います。
IICn_UserInit	シリアル・インタフェース（IIC*n*）に関するユーザ独自の初期化処理を行います。
IICn_Stop	IIC*n* 通信を終了します。

API 関数名	機能概要
IICn_MasterSendStart	IIC*n* マスタ送信を開始します。
IICn_MasterReceiveStart	IIC*n* マスタ受信を開始します。
IICn_MasterSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_MasterReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_MasterErrorCallback	IIC*n* 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。
IICn_SlaveSendStart	IIC*n* スレーブ送信を開始します。
IICn_SlaveReceiveStart	IIC*n* スレーブ受信を開始します。
IICn_SlaveSendEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_SlaveReceiveEndCallback	IIC*n* 通信完了割り込み INTIIC*n* の発生に伴う処理を行います。
IICn_SlaveErrorCallback	IIC*n* 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。
IICn_GetStopConditionCallback	IIC*n* 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。

SAU*m*_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

シリアル・アレイ・ユニット，およびシリアル・インタフェー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    SAUm_Init ( void );
```

備考 *m* は，ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

SAU*m*_UserInit

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

シリアル・アレイ・ユニット，およびシリアル・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，SAUm_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    SAUm_UserInit ( void );
```

備考 *m* は，ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

SAU*m*_PowerOff

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

シリアル・アレイ・ユニット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，シリアル・アレイ・ユニット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（シリアル・クロック選択レジスタ *n*：SPS*n* など）への書き込みは無視されます。

[所属]

CG_serial.c

[指定形式]

```
void    SAUm_PowerOff ( void );
```

備考 *m* は，ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

UART*n*_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

シリアル・インタフェース（UART）用チャネルの初期化処理を行います。

備考 本 API 関数は，SAUm_Init の内部関数として位置づけられているため，通常，ユーザの処理プログラムから呼び出す必要はありません。

[所属]

CG_serial.c

[指定形式]

```
void    UARTn_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

UART*n*_Start

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

UART 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    UARTn_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

UART*n*_Stop

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

UART 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    UARTn_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

UART*n*_SendData

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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 UARTn_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   UARTn_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

[使用例]

以下に，チャネル 0 から 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 */
    ……
    UART0_Start ();                     /* UART 通信の開始 */
    UART0_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_INTST0 ( void ) {   /* 割り込み INTST0 発生時の割り込み処理 */
    if ( gUart0TxCnt > 0 ) {
        ……
    } else {
        UART0_SendEndCallback ();       /* コールバック・ルーチンの呼び出し */
    }
}

void UART0_SendEndCallback ( void ) {   /* 割り込み INTST0 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 */
}
```

UART*n*_ReceiveData

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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，UARTn_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   UARTn_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 0 から 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 */
    ……
    UART0_ReceiveData ( &rxbuf, rxnum );    /* UART 受信の開始 */
    UART0_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_INTSR0 ( void ) {       /* 割り込み INTSR0 発生時の割り込み処理 */
    ……
    if ( gUart0RxLen > gUart0RxCnt ) {
        ……
        if ( gUart0RxLen == gUart0RxCnt ) {
            UART0_ReceiveEndCallback ();    /* コールバック・ルーチンの呼び出し */
        }
    }
}

void UART0_ReceiveEndCallback ( void ) {    /* 割り込み INTSR0 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 */
}
```

UART*n*_SendEndCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

UART 送信完了割り込み INTST*n* 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 INTST*n* に対応した割り込み処理 MD_INTST*n* のコールバック・ルーチン（UARTn_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    UARTn_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

UART*n*_ReceiveEndCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

UART 受信完了割り込み INTSR*n* 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 INTSR*n* に対応した割り込み処理 MD_INTSR*n* のコールバック・ルーチン（UARTn_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    UARTn_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

UART*n*_SoftOverRunCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

UART 受信完了割り込み INTSR*n* 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 INTSR*n* に対応した割り込み処理 MD_INTSR*n* のコールバック・ルーチン（UARTn_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]

- 【Fx3】

```
void    UARTn_SoftOverRunCallback ( void );
```

- 【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_ad.h"
void    UARTn_SoftOverRunCallback ( UCHAR rx_data );
```

備考 *n* は，チャネル番号を意味します。

[引数]

- 【Fx3】

なし

- 【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

I/O	引数	説明
O	UCHAR *rx_data*;	受信したデータ（UARTn_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

UART*n*_ErrorCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

UART 通信におけるエラー割り込み INTSRE*n* 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 INTSRE*n* に対応した割り込み処理 MD_INTSRE*n* 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    UARTn_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：パリティ・エラー 000001*xx*B：フレーミング・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_INTSRE0 ( void ) {        /* 割り込み INTSRE0 発生時の割り込み処理 */
    UCHAR   err_type;
    ……
    UART0_ErrorCallback ( err_type );         /* コールバック・ルーチンの呼び出し */
```

```
}


void UART0_ErrorCallback ( UCHAR err_type ) { /* 割り込み INTSRE0 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 */
        ……   /* パリティ・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 */
        ……   /* フレーミング・エラーが発生した際のコールバック処理 */
    }
}
```

CSI*mn*_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

シリアル・インタフェース（CSI）用チャネルの初期化処理を行います。

備考 本 API 関数は，SAUm_Init の内部関数として位置づけられているため，通常，ユーザの処理プログラムから呼び出す必要はありません。

[所属]

CG_serial.c

[指定形式]

```
void    CSImn_Init ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSI*mn*_Start

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

CSI 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    CSImn_Start ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSI*mn*_Stop

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

CSI 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    CSImn_Stop ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSI*mn*_SendData

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

データの CSI 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 CSI 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. CSI 送信を行う際には，本 API 関数の呼び出し以前に CSImn_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   CSImn_SendData ( UCHAR *txbuf, USHORT txnum );
```

備考 *m* はユニット番号を，*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

[使用例]

以下に，チャネル 00 から 4 バイトの固定長データを 1 回だけ CSI 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 */
    ……
    CSI00_Start ();                     /* CSI 通信の開始 */
    CSI00_SendData ( &txbuf, txnum );   /* CSI 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_INTCSI00 ( void ) {   /* 割り込み INTCSI00 発生時の割り込み処理 */
    if ( gCsi00TxCnt > 0 ) {
        ……
    } else {
        CSI00_SendEndCallback ();       /* コールバック・ルーチンの呼び出し */
    }
}

void CSI00_SendEndCallback ( void ) {     /* 割り込み INTCSI00 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 */
}
```

CSI*mn*_ReceiveData

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

データの CSI 受信を開始します。

備考 1. 本 API 関数では，1 バイト単位の CSI 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. CSI 受信を行う際には，本 API 関数の呼び出し以前に CSImn_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   CSImn_ReceiveData ( UCHAR *rxbuf, USHORT rxnum );
```

備考 *m* はユニット番号を，*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 00 から 4 バイトの固定長データを 1 回だけ CSI 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 */
    ……
    CSI00_Start ();                         /* CSI 通信の開始 */
    CSI00_ReceiveData ( &rxbuf, rxnum );    /* CSI 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_INTCSI00 ( void ) {     /* 割り込み INTCSI00 発生時の割り込み処理 */
    if ( gCsi00RxCnt < gCsi00RxLen ) {
        ……
        if ( gCsi00RxCnt == gCsi00RxLen ) {
            CSI00_ReceiveEndCallback ();    /* コールバック・ルーチンの呼び出し */

        } else {
            ……
        }
    }
}

void CSI00_ReceiveEndCallback ( void ) {    /* 割り込み INTCSI00 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 */
}
```

CSI*mn*_SendReceiveData

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

データの CSI 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 CSI 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 CSI 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. CSI 送受信を行う際には，本 API 関数の呼び出し以前に CSImn_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   CSImn_SendReceiveData ( UCHAR *txbuf, USHORT txnum, UCHAR *rxbuf );
```

備考 *m* はユニット番号を，*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 00 から 4 バイトの固定長データを 1 回だけ CSI 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 */
    ……
    CSI00_Start ();                                      /* CSI 通信の開始 */
    CSI00_SendReceiveData ( &txbuf, txnum, &rxbuf );     /* CSI 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_INTCSI00 ( void ) {     /* 割り込み INTCSI00 発生時の割り込み処理 */
   if ( gCsi00TxCnt > 0 ) {
        ……
    } else {
        ……
        CSI00_SendEndCallback ();           /* コールバック・ルーチンの呼び出し */
    }
}

void CSI00_SendEndCallback ( void ) {       /* 割り込み INTCSI00 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 */
}
```

CSI*mn*_SendEndCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。

備考 本 API 関数は，CSI 通信完了割り込み INTCSI*mn* に対応した割り込み処理 MD_INTCSI*mn* のコールバック・ルーチン（CSImn_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    CSImn_SendEndCallback ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSI*mn*_ReceiveEndCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

CSI 通信完了割り込み INTCSI*mn* の発生に伴う処理を行います。

備考 本 API 関数は，CSI 通信完了割り込み INTCSI*mn* に対応した割り込み処理 MD_INTCSI*mn* のコールバック・ルーチン（CSImn_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    CSImn_ReceiveEndCallback ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

CSI*mn*_ErrorCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

CSI 通信におけるエラー割り込み INTSRE*n* 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 INTSRE*n* に対応した割り込み処理 MD_INTSRE*n* 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    CSImn_ErrorCallback ( UCHAR err_type );
```

備考 *m* はユニット番号を，*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

[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_INTSRE0 ( void ) {        /* 割り込み INTSRE0 発生時の割り込み処理 */
    UCHAR   err_type;
    ……
    CSI00_ErrorCallback ( err_type );         /* コールバック・ルーチンの呼び出し */
}
```

```
void CSI00_ErrorCallback ( UCHAR err_type ) { /* 割り込み INTSRE0 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 */
    }
}
```

IIC*mn*_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

シリアル・インタフェース（簡易 IIC）用チャネルの初期化処理を行います。

備考 本 API 関数は，SAUm_Init の内部関数として位置づけられているため，通常，ユーザの処理プログラムから呼び出す必要はありません。

[所属]

CG_serial.c

[指定形式]

```
void    IIC*mn*_Init ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_Stop

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

簡易 IIC 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    IIC*mn*_Stop ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_MasterSendStart

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

簡易 IIC 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 バイト単位の簡易 IIC 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]

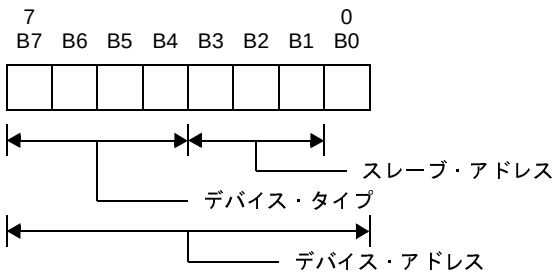
```
#include    "CG_macrodriver.h"
void    IICmn_MasterSendStart ( UCHAR adr, UCHAR *txbuf, USHORT txnum );
```

備考 *m* はユニット番号を，*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

備考 以下に，デバイス・アドレス *adr* の指定形式を示します。



[戻り値]

なし

IIC*mn*_MasterReceiveStart

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

簡易 IIC マスタ受信を開始します。

備考 本 API 関数では，1 バイト単位の簡易 IIC 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]

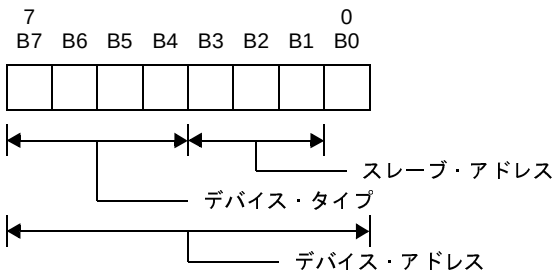
```
#include    "CG_macrodriver.h"
void    IICmn_MasterReceiveStart ( UCHAR adr, UCHAR *rxbuf, USHORT rxnum );
```

備考 *m* はユニット番号を，*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

備考 以下に，デバイス・アドレス *adr* の指定形式を示します。



[戻り値]

なし

IIC*mn*_StartCondition

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

スタート・コンディションを発生します。

備考 本 API 関数は，IICmn_MasterSendStart，および IICmn_MasterReceiveStart の内部関数として位置づけられているため，通常，ユーザの処理プログラムから呼び出す必要はありません。

[所属]

CG_serial.c

[指定形式]

```
void    IIC mn_StartCondition ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_StopCondition

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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    IIC*mn*_StopCondition ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_MasterSendEndCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。

備考 本 API 関数は，簡易 IIC 通信完了割り込み INTIIC*mn* に対応した割り込み処理 MD_INTIIC*mn* のコールバック・ルーチン（IICmn_MasterSendStart 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    IIC*mn*_MasterSendEndCallback ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_MasterReceiveEndCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

簡易 IIC 通信完了割り込み INTIIC*mn* の発生に伴う処理を行います。

備考 本 API 関数は，簡易 IIC 通信完了割り込み INTIIC*mn* に対応した割り込み処理 MD_INTIIC*mn* のコールバック・ルーチン（IICmn_MasterReceiveStart の引数 *rxnum* 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    IIC*mn*_MasterReceiveEndCallback ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

IIC*mn*_MasterErrorCallback

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

簡易 IIC 通信におけるパリティ・エラー（ACK 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    IICmn_MasterErrorCallback ( MD_STATUS flag );
```

備考 *m* はユニット番号を，*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*;	通信エラーの発生要因 MD_NACK：アクノリッジの未検出

[戻り値]

なし

UARTF*n*_Init

【Fx3】

シリアル・インタフェース（UARTF*n*）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    UARTFn_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

UARTF*n*_PowerOff

【Fx3】

シリアル・インタフェース（UARTF*n*）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，シリアル・インタフェース（UARTF*n*）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（LIN-UART*n* 状態レジスタ：UF*n*STR など）への書き込みは無視されます。

[所属]

CG_serial.c

[指定形式]

```
void    UARTFn_PowerOff ( 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

UARTF*n*_Start

【Fx3】

UARTF 通信を待機状態にします。

[所属]

CG_serial.c

[指定形式]

```
void    UARTFn_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

UARTF*n*_Stop

【Fx3】

UARTF 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    UARTFn_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

UARTF*n*_SendData

【Fx3】

データの UARTF 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 UARTF 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. UARTF 送信を行う際には，本 API 関数の呼び出し以前に UARTFn_Start を呼び出す必要があります。

3. シリアル・インタフェース（UARTF*n*）を拡張ビット・モードで使用する場合，引数 *txbuf* で指定されたバッファには，送信するデータを以下の形式で格納します。

“8 ビット・データ”，“拡張ビット”，“8 ビット・データ”，“拡張ビット”，…

[所属]

CG_serial.c

[指定形式]

```
#include    “CG_macrodriver.h“
MD_STATUS   UARTFn_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	送信処理を実行中

UARTF*n*_ReceiveData

【Fx3】

データの UARTF 受信を終了にします。

備考 1. 本 API 関数では，1 バイト単位の UARTF 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 UARTF 受信は，本 API 関数の呼び出し後，UARTFn_Start を呼び出すことにより開始されます。

3. シリアル・インタフェース（UARTF*n*）を拡張ビット・モードで使用する場合，引数 *rxbuf* で指定されたバッファには，受信したデータが以下の形式で格納されます。

“8 ビット・データ”，“拡張ビット”，“8 ビット・データ”，“拡張ビット”，…

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
MD_STATUS   UARTFn_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

UARTF*n*_SetComparisonData

【Fx3】

受信データと比較するデータを設定します。

備考 引数 *comdata* に指定された値は，LIN-UART*n* ID 設定レジスタ（UF*n*ID）に設定されます。

[所属]

CG_serial.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    UARTFn_SetComparisonData ( UCHAR comdata );
```

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 *comdata*;	比較するデータ

[戻り値]

なし

UARTF*n*_DataComparisonEnable

【Fx3】

データの比較を開始します。

備考 本 API 関数を呼び出すことにより，シリアル・インタフェース（UARTF*n*）は，拡張ビット・モード（データ比較あり）へと移行します。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_DataComparisonEnable ( 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

UARTF*n*_DataComparisonDisable

【Fx3】

データの比較を終了します。

備考 本 API 関数を呼び出すことにより，シリアル・インタフェース（UARTF*n*）は，拡張ビット・モード（データ比較なし）へと移行します。

[所属]

CG_serial_user.c

[指定形式]

```
void    UARTFn_DataComparisonDisable ( 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

UARTF*n*_SendEndCallback

【Fx3】

送信割り込み INTLT*n* の発生に伴う処理を行います。

備考 本 API 関数は，送信割り込み INTLT*n* に対応した割り込み処理 MD_INTLT*n* のコールバック・ルーチン（UARTFn_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    UARTFn_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

UARTF*n*_ReceiveEndCallback

【Fx3】

受信完了割り込み INTLR*n* の発生に伴う処理を行います。

備考 本 API 関数は，受信完了割り込み INTLR*n* に対応した割り込み処理 MD_INTLR*n* のコールバック・ルーチン（UARTFn_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    UARTFn_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

UARTF*n*_SoftOverRunCallback

【Fx3】

受信完了割り込み INTLR*n* の発生に伴う処理を行います。

備考 本 API 関数は，受信完了割り込み INTLR*n* に対応した割り込み処理 MD_INTLR*n* のコールバック・ルーチン（UARTFn_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]

```
void    UARTFn_SoftOverRunCallback ( 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

UARTF*n*_ExpBitCetectCallback

【Fx3】

ステータス割り込み INTLS*n* の発生に伴う処理を行います。

備考 本 API 関数は，ステータス割り込み INTLS*n* に対応した割り込み処理 MD_INTLS*n* のコールバック・ルーチン（拡張ビット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]

```
void    UARTFn_ExpBitCetectCallback ( 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

UARTF*n*_IDMatchCallback

【Fx3】

ステータス割り込み INTLS*n* の発生に伴う処理を行います。

備考 本 API 関数は，ステータス割り込み INTLS*n* に対応した割り込み処理 MD_INTLS*n* 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    UARTFn_IDMatchCallback ( 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

UARTF*n*_ErrorCallback

【Fx3】

ステータス割り込み INTLS*n* の発生に伴う処理を行います。

備考 本 API 関数は，ステータス割り込み INTLS*n* に対応した割り込み処理 MD_INTLSn 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    UARTFn_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*;	ステータス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

IICA_Init

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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_PowerOff

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

シリアル・インタフェース（IICA）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，シリアル・インタフェース（IICA）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（IICA コントロール・レジスタ *n*：IICCTL*n* など）への書き込みは無視されます。

[所属]

CG_serial.c

[指定形式]

```
void    IICA_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

IICA_Stop

【IC3（48 ピン)】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く)】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【LX3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

IICA 通信完了割り込み INTIICA 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 INTIICA に対応した割り込み処理 MD_INTIICA 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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

IICA 通信完了割り込み INTIICA 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 INTIICA に対応した割り込み処理 MD_INTIICA 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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

IICA 通信完了割り込み INTIICA 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 INTIICA に対応した割り込み処理 MD_INTIICA 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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

IICA 通信完了割り込み INTIICA 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 INTIICA に対応した割り込み処理 MD_INTIICA 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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

【IC3（48 ピン）】【ID3】【IE3】【Kx3-A】【Kx3-L】【Lx3（LF3 を除く）】

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

IIC*n*_Init

【Kx3】

シリアル・インタフェース（IIC*n*）の初期化処理を行います。

[所属]

CG_serial.c

[指定形式]

```
void    IICn_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

IIC*n*_UserInit

【Kx3】

シリアル・インタフェース（IIC*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，IICn_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    IIC*n*_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

IIC*n*_Stop

【Kx3】

IIC*n* 通信を終了します。

[所属]

CG_serial.c

[指定形式]

```
void    IICn_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

IIC*n*_MasterSendStart

【Kx3】

IIC*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 IIC*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   IICn_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*;	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_ERROR	異常終了

IIC*n*_MasterReceiveStart

【Kx3】

IIC*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 IIC*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   IICn_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*;	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_ERROR	異常終了

IIC*n*_MasterSendEndCallback

【Kx3】

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

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_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

IIC*n*_MasterReceiveEndCallback

【Kx3】

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

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_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

IIC*n*_MasterErrorCallback

【Kx3】

IIC*n* 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    IICn_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
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

IIC*n*_SlaveSendStart

【Kx3】

IIC*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 IIC*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    IICn_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

IIC*n*_SlaveReceiveStart

【Kx3】

IIC*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 IIC*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    IICn_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

IIC*n*_SlaveSendEndCallback

【Kx3】

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

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_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

IIC*n*_SlaveReceiveEndCallback

【Kx3】

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

備考 本 API 関数は，IIC*n* 通信完了割り込み INTIIC*n* に対応した割り込み処理 MD_INTIIC*n* のコールバック・ルーチンとして呼び出されます。

[所属]

CG_serial_user.c

[指定形式]

```
void    IICn_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

IIC*n*_SlaveErrorCallback

【Kx3】

IIC*n* 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    IICn_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
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

IIC*n*_GetStopConditionCallback

【Kx3】

IIC*n* 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    IICn_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 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—7 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。
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

【Kx3-A】【Lx3】

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

【Kx3-A】【Lx3】

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

AMP*n*_Start

【Kx3-A】【Lx3】

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

【Kx3-A】【Lx3】

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. 7 コンパレータ／PG アンプ

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

表 3—8 コンパレータ／PG アンプ用 API 関数

API 関数名	機能概要
CMPPGA_Init	コンパレータ／プログラマブル・ゲイン・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。
CMPPGA_UserInit	コンパレータ／プログラマブル・ゲイン・アンプに関するユーザ独自の初期化処理を行います。
CMPPGA_PowerOff	コンパレータ／プログラマブル・ゲイン・アンプに対するクロック供給を停止します。
CMPPGA_Start	コンパレータ／プログラマブル・ゲイン・アンプの動作を開始します。
CMPPGA_Stop	コンパレータ／プログラマブル・ゲイン・アンプの動作を停止します。
CMPPGA_ChangeCMPnRefVoltage	コンパレータ *n* の内蔵基準電圧を設定します。
CMPPGA_ChangePGAFactor	プログラマブル・ゲイン・アンプにおける入力電圧の増幅率を設定します。

CMPPGA_Init

【Ix3】【Kx3-L】

コンパレータ／プログラマブル・ゲイン・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_cmppga.c

[指定形式]

```
void    CMPPGA_Init ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_UserInit

【Ix3】【Kx3-L】

コンパレータ／プログラマブル・ゲイン・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，CMPPGA_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_cmppga_user.c

[指定形式]

```
void    CMPPGA_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_PowerOff

【Ix3】【Kx3-L】

コンパレータ／プログラマブル・ゲイン・アンプ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，コンパレータ／プログラマブル・ゲイン・アンプ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（プログラマブル・ゲイン・アンプ制御レジスタ：OAM など）への書き込みは無視されます。

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_Start

【Ix3】【Kx3-L】

コンパレータ／プログラマブル・ゲイン・アンプの動作を開始します。

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_Start ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_Stop

【Ix3】【Kx3-L】

コンパレータ／プログラマブル・ゲイン・アンプの動作を停止します。

[所属]

CG_cmppga.c

[指定形式]

```
void    CMPPGA_Stop ( void );
```

[引数]

なし

[戻り値]

なし

CMPPGA_ChangeCMP*n*RefVoltage

【Ix3】【Kx3-L】

コンパレータ *n* の内蔵基準電圧を設定します。

備考 引数 *voltage* に指定された値は，コンパレータ *n* 内蔵基準電圧選択レジスタ（C*n*RVM）に設定されます。

[所属]

CG_cmppga.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_ChangeCMPnRefVoltage ( enum CMPRefVoltage voltage );
```

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 CMPRefVoltage *voltage*;	コンパレータ *n* の内蔵基準電圧 【*n* = 0：チャネル 0 の場合】 CMPREFVOL0： 2AVREF/16 CMPREFVOL1： 4AVREF/16 CMPREFVOL2： 6AVREF/16 CMPREFVOL3： 8AVREF/16 CMPREFVOL4： 10AVREF/16 CMPREFVOL5： 12AVREF/16 【*n* = 1：チャネル 1 の場合】 CMPREFVOL0： 3AVREF/16 CMPREFVOL1： 5AVREF/16 CMPREFVOL2： 7AVREF/16 CMPREFVOL3： 9AVREF/16 CMPREFVOL4： 11AVREF/16 CMPREFVOL5： 13AVREF/16

[戻り値]

マクロ	説明
MD_OK	正常終了

マクロ	説明
MD_ARGERROR	引数の指定が不正

CMPPGA_ChangePGAFactor

【Ix3】【Kx3-L】

プログラマブル・ゲイン・アンプにおける入力電圧の増幅率を設定します。

備考 引数 *voltage* に指定された値は，プログラマブル・ゲイン・アンプ制御レジスタ（OAM）に設定されます。

[所属]

CG_cmppga.c

[指定形式]

```
#include    "CG_macrodriver.h"
#include    "CG_cmppga.h"
MD_STATUS   CMPPGA_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： 6 倍 PGAFACTOR2： 8 倍 PGAFACTOR3： 10 倍 PGAFACTOR4： 12 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

3. 3. 8 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—9 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_PowerOff	A/D コンバータに対するクロック供給を停止します。
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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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_PowerOff

【Fx3】【Ix3】【Kx3】【Kx3-L】

A/D コンバー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，A/D コンバー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（A/D コンバータ・モード・レジスタ：ADCM など）への書き込みは無視されます。

[所属]

CG_ad.c

[指定形式]

```
void    AD_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

AD_ComparatorOn

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

電圧コンパレータを動作許可状態に設定します。

備考 1. 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。

2. ［A/D コンバータ］の［コンパレータの動作設定］エリアで“許可”を選択した場合，電圧コンパレータは“常時 ON”となるため，本 API 関数の呼び出しは不要となります。

[所属]

CG_ad.c

[指定形式]

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

[引数]

なし

[戻り値]

なし

AD_ComparatorOff

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

[使用例 1]

以下に，［A/D コンバータ］で選択された変換開始端子からのアナログ電圧を A/D 変換したのち，入力端子 ANI1 からのアナログ電圧を A/D 変換する際の例を示します。

なお，下記は，［A/D コンバータ］の［コンパレータの動作設定］エリアで，“停止”が選択された場合（AD_ComparatorOn の呼び出しを行う場合）の例となっています。

【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_ComparatorOn ();                 /* 動作許可状態への移行 */
    while ( wait );                     /* 安定時間の確保（1μ 秒以上）*/
```

```
    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_SelectADChannel ( ADCHANNEL1 );    /* 入力端子の切り替え */
    gFlag = 1;                            /* A/D 変換完了フラグの初期化 */
    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 */
    AD_ComparatorOff ();                  /* 動作停止状態への移行 */
    ……
}
```

【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 */
}
```

［使用例 2］

以下に，［A/D コンバータ］で選択された変換開始端子からのアナログ電圧を A/D 変換したのち，入力端子 ANI1 からのアナログ電圧を A/D 変換する際の例を示します。

なお，下記は，［A/D コンバータ］の［コンパレータの動作設定］エリアで，“許可”が選択された場合（AD_ComparatorOn の呼び出しを行わない場合）の例となっています。

【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;
    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_SelectADChannel ( ADCHANNEL1 );    /* 入力端子の切り替え */
    gFlag = 1;                            /* A/D 変換完了フラグの初期化 */
    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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

```
#include    "CG_ad.h"
void    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]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

なし

AD_Read

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

A/D 変換結果を読み出します。

備考 読み出す A/D 変換結果は，対象デバイスが Fx3，78K0R/Kx3，78K0R/Kx3-L の場合は 10 ビット，78K0R/Kx3-A，78K0R/Lx3 の場合は 12 ビットとなります。

[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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

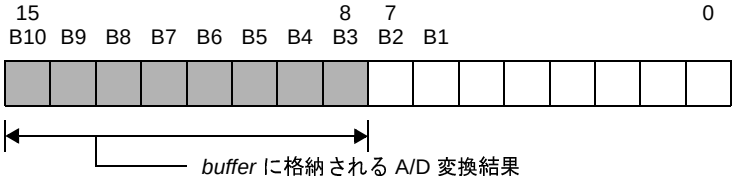
```
#include    "CG_macrodriver.h"
void    AD_ReadByte ( UCHAR *buffer );
```

[引数]

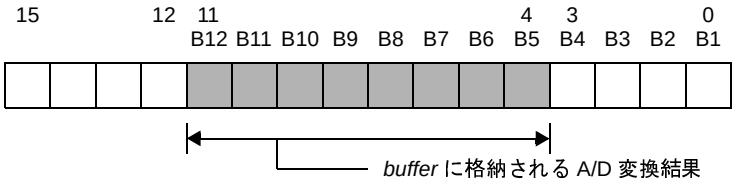
I/O	引数	説明
O	UCHAR *buffer;	読み出した A/D 変換結果（8 ビット：10 ビット分解能の上位 8 ビット）を格納する領域へのポインタ

備考 以下に，*buffer* に格納される A/D 変換結果を示します。

- 【Fx3】【Ix3】【Kx3】【Kx3-L】



- 【Kx3-A】【Lx3】



[戻り値]

なし

3. 3. 9 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—10 D/A コンバータ用 API 関数

API 関数名	機能概要
DA_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。
DA_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。
DA_PowerOff	D/A コンバータに対するクロック供給を停止します。
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。
DAn_Set8BitsValue	ANO*n* 端子に出力するアナログ電圧値（8 ビット）を設定します。
DAn_Set12BitsValue	ANO*n* 端子に出力するアナログ電圧値（12 ビット）を設定します。

DA_Init

【Kx3】【Kx3-A】【Lx3】

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    DA_Init ( void );
```

[引数]

なし

[戻り値]

なし

DA_UserInit

【Kx3】【Kx3-A】【Lx3】

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，DA_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    DA_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

DA_PowerOff

【Kx3】【Kx3-A】【Lx3】

D/A コンバー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，D/A コンバー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（D/A コンバータ・モード・レジスタ：DAM など）への書き込みは無視されます。

[所属]

CG_da.c

[指定形式]

```
void    DA_PowerOff ( void );
```

[引数]

なし

[戻り値]

なし

DA*n*_Start

【Kx3】【Kx3-A】【Lx3】

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

【Kx3】【Kx3-A】【Lx3】

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

【Kx3】

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_timer_user.c】

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* 割り込み INTTM05 発生時の割り込み処理 */
    DA1_SetValue ( gValue++ );          /* アナログ電圧値の設定 */
}
```

DA*n*_Set8BitsValue

【Kx3-A】【Lx3】

ANO*n* 端子に出力するアナログ電圧値（8 ビット）を設定します。

[所属]

CG_da.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    DAn_Set8BitsValue ( 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 ～ 0x1f）

[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_Set8BitsValue ( 0x7f );         /* アナログ電圧値の設定 */
    ……
}
```

【CG_timer_user.c】

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* 割り込み INTTM05 発生時の割り込み処理 */
    DA1_Set8BitsValue ( gValue++ );     /* アナログ電圧値の設定 */
}
```

DA*n*_Set12BitsValue

【Kx3-A】【Lx3】

ANO*n* 端子に出力するアナログ電圧値（12 ビット）を設定します。

[所属]

CG_da.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    DAn_Set12BitsValue ( 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 ～ 0xfff）

[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_Set12BitsValue ( 0x1ff );       /* アナログ電圧値の設定 */
    ……
}
```

【CG_timer_user.c】

```
#include    "CG_macrodriver.h"
UCHAR   gValue = 0;
__interrupt void MD_INTTM05 ( void ) {  /* 割り込み INTTM05 発生時の割り込み処理 */
    DA1_Set12BitsValue ( gValue++ );    /* アナログ電圧値の設定 */
}
```

3. 3. 10 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—11 タイマ用 API 関数

API 関数名	機能概要
TAUm_Init	タイマ・アレイ・ユニッ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。
TAUm_UserInit	タイマ・アレイ・ユニットに関するユーザ独自の初期化処理を行います。
TAUm_PowerOff	タイマ・アレイ・ユニットに対するクロック供給を停止します。
TAUm_Channeln_Start	チャネル *n* のカウントを開始します。
TAUm_Channeln_Stop	チャネル *n* のカウントを終了します。
TAUm_Channeln_ChangeCondition	カウント値を変更します。
TAUm_Channeln_ChangeTimerCondition	カウント値を変更します。
TAUm_Channeln_GetPulseWidth	TI*mn* 端子に対する入力信号（入力パルス）のパルス間隔，またはハイ／ロウ・レベルの測定幅を獲得します。
TAUm_Channeln_ChangeDuty	TO*mn* 端子に出力する PWM 信号のデューティ比を変更します。
TAUm_Channeln_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。

TAU*m*_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

タイマ・アレイ・ユニッ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_timer.c

[指定形式]

```
void    TAUm_Init ( void );
```

備考 *m* は，ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

TAU*m*_UserInit

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

タイマ・アレイ・ユニッ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，TAUm_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    TAUm_UserInit ( void );
```

備考 *m* は，ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

TAU*m*_PowerOff

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

タイマ・アレイ・ユニット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，タイマ・アレイ・ユニット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（タイマ・クロック選択レジスタ 0：TPS0 など）への書き込みは無視されます。

[所属]

CG_timer.c

[指定形式]

```
void    TAUm_PowerOff ( void );
```

備考 *m* は，ユニット番号を意味します。

[引数]

なし

[戻り値]

なし

TAU*m*_Channel*n*_Start

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

チャネル *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，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    TAUm_Channeln_Start ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

TAU*m*_Channel*n*_Stop

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

チャネル *n* のカウントを終了します。

[所属]

CG_timer.c

[指定形式]

```
void    TAUm_Channeln_Stop ( void );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

なし

[戻り値]

なし

TAU*m*_Channel*n*_ChangeCondition

【Kx3】

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，タイマ・データ・レジスタ *mn*（TDR*mn*）に設定されます。

2. 本 API 関数の呼び出しタイミングは，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）により以下のように異なります。

機能の種類	呼び出しタイミング
インターバル・タイマ	任意のタイミングで可
方形波出力	任意のタイミングで可
分周器機能	任意のタイミングで可
外部イベント・カウンタ	任意のタイミングで可
入力パルス間隔測定	呼び出し不可
入力信号のハイ／ロウ・レベル幅測定	呼び出し不可
PWM 出力	呼び出し不可
ワンショット・パルス出力	動作中は呼び出し不可
多重 PWM 出力	呼び出し不可

[所属]

CG_timer.c

[指定形式]

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

備考 *m* はユニット番号を，*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

［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 ) {
    USHORT  value = TAU_TDR00_VALUE >> 1;       /* TAU_TDR00_VALUE：現在のインターバル時間 */
    ……
    TAU0_Channel0_Start ();                     /* カウントの開始 */
    ……
    TAU0_Channel0_ChangeCondition ( value );    /* カウント値の変更 */
   ……
}
```

TAU*m*_Channel*n*_ChangeTimerCondition

【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

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，タイマ・データ・レジスタ *mn*（TDR*mn*）に設定されます。

2. 本 API 関数の呼び出しタイミングは，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）により以下のように異なります。

機能の種類	呼び出しタイミング
インターバル・タイマ	任意のタイミングで可
方形波出力	任意のタイミングで可
分周器機能	任意のタイミングで可
外部イベント・カウンタ	任意のタイミングで可
入力パルス間隔測定	呼び出し不可
入力信号のハイ／ロウ・レベル幅測定	呼び出し不可
PWM 出力	呼び出し不可
ワンショット・パルス出力	動作中は呼び出し不可
多重 PWM 出力	呼び出し不可

[所属]

CG_timer.c

[指定形式]

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

備考 *m* はユニット番号を，*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

［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 ) {
    USHORT  value = TAU_TDR00_VALUE >> 1;          /* TAU_TDR00_VALUE：現在のインターバル時間 */
    ……
    TAU0_Channel0_Start ();                        /* カウントの開始 */
    ……
    TAU0_Channel0_ChangeTimerCondition ( value ); /* カウント値の変更 */
   ……
}
```

TAU*m*_Channel*n*_GetPulseWidth

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

TI*mn* 端子に対する入力信号（入力パルス）のパルス間隔，またはハイ／ロウ・レベルの測定幅を獲得します。

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_GetPulseWidth ( ULONG *width );
```

備考 *m* はユニット番号を，*n* はチャネル番号を意味します。

[引数]

I/O	引数	説明
O	ULONG *width;	測定幅（0x0 ～ 0x1ffff）を格納する領域へのポインタ

[戻り値]

なし

TAU*m*_Channel*n*_ChangeDuty

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

TO*mn* 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 関数の呼び出しタイミングは，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）により以下のように異なります。

機能の種類	呼び出しタイミング
インターバル・タイマ	呼び出し不可
方形波出力	呼び出し不可
分周器機能	呼び出し不可
外部イベント・カウンタ	呼び出し不可
入力パルス間隔測定	呼び出し不可
入力信号のハイ／ロウ・レベル幅測定	呼び出し不可
PWM 出力	マスタ・チャネルで割り込み INTTM*mn* が発生した後
ワンショット・パルス出力	呼び出し不可
多重 PWM 出力	マスタ・チャネルで割り込み INTTM*mn* が発生した後

[所属]

CG_timer.c

[指定形式]

```
#include    "CG_macrodriver.h"
void    TAUm_Channeln_ChangeDuty ( UCHAR ratio );
```

備考 *m* はユニット番号を，*n* はスレーブ側の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。

なお，下記は，チャネル 0，1 が PWM 出力，または多重 PWM 出力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 ) {
    UCHAR   ratio = 25;
    ……
    TAU0_Channel0_Start ();             /* カウントの開始 */
    ……
    TAU0_Channel1_ChangeDuty ( ratio ); /* デューティ比の変更 */
    ……
}
```

TAU*m*_Channel*n*_SoftWareTriggerOn

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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。

[所属]

CG_timer.c

[指定形式]

```
void    TAUm_Channeln_SoftWareTriggerOn ( void );
```

備考 *m* はユニット番号を，*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. 11 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—12 ウォッチドッグ・タイマ用 API 関数

API 関数名	機能概要
WDT_Init	ウォッチドッグ・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。
WDT_UserInit	ウォッチドッグ・タイマに関するユーザ独自の初期化処理を行います。
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_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

ウォッチドッグ・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_wdt.c

[指定形式]

```
void    WDT_Init ( void );
```

[引数]

なし

[戻り値]

なし

WDT_UserInit

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

ウォッチドッグ・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，WDT_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_wdt_user.c

[指定形式]

```
void    WDT_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

WDT_Restart

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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. 12 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—13 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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

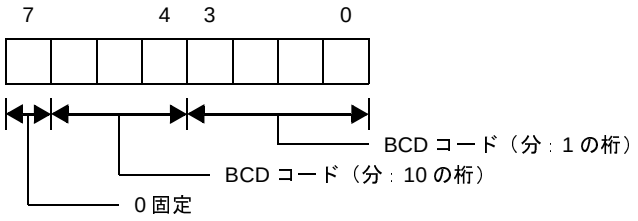
I/O	引数	説明
I	struct RTCAlarmValue *alarmval*;	アラームの発生条件（曜日，時，分）

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。



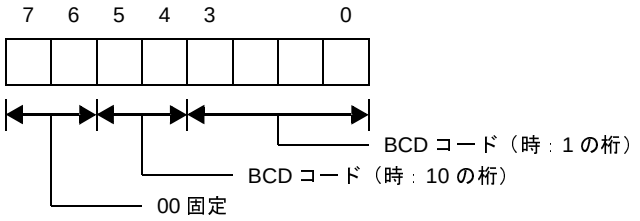
- Alarmwh（時）

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。

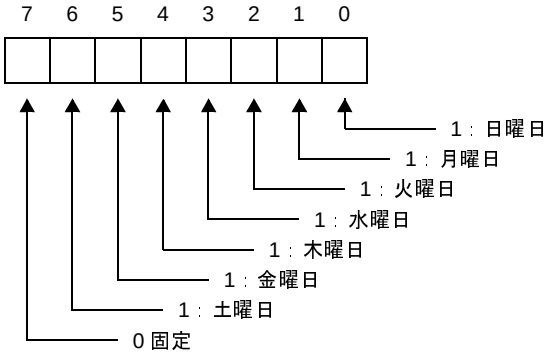
0： 午前

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

- 【Ix3（IB3 を除く）】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

```
#include    "CG_rtc.h"
void    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/fXT INTERVAL1： 2^7/fXT INTERVAL2： 2^8/fXT INTERVAL3： 2^9/fXT INTERVAL4： 2^10/fXT INTERVAL5： 2^11/fXT INTERVAL6： 2^12/fXT

備考 fXT は，サブシステム・クロックの周波数を意味します。

[戻り値]

- 【Ix3（IB3 を除く）】【Kx3-A】【Kx3-L】【Lx3】

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

- 【Kx3】

なし

[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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Ix3（IB3 を除く）】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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. 13 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—14 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

【Fx3】

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

【Fx3】

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

【Fx3】

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

【Fx3】

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

【Fx3】

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"
void    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 MAINCLOCK : fMAIN MAIN2 : fMAIN/2 MAIN4 : fMAIN/4 MAIN8 : fMAIN/8 MAIN16 : fMAIN/16 MAIN2048 : fMAIN/2048 MAIN4096 : fMAIN/4096 MAIN8192 : fMAIN/8192 PLLCLOCK : fPLL PLL2 : fPLL/2 PLL4 : fPLL/4 PLL8 : fPLL/8 PLL16 : fPLL/16 PLL2048 : fPLL/2048 PLL4096 : fPLL/4096 PLL8192 : fPLL/8192 ILCLOCK : fIL SUBCLOCK : fSUB

備考 fMAIN はメイン・システム・クロックの周波数を，fPLL は PLL クロックの周波数を，fIL は低速内蔵発振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]

なし

3. 3. 14 クロック出力／ブザー出力

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

表 3—15 クロック出力／ブザー出力用 API 関数

API 関数名	機能概要
PCLBUZn_Init	クロック出力／ブザー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。
PCLBUZn_UserInit	クロック出力／ブザー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。
PCLBUZn_Start	クロック出力／ブザー出力を開始します。
PCLBUZn_Stop	クロック出力／ブザー出力を停止します。
PCLBUZn_ChangeFreq	PCLBUZ*n* 端子への出力クロックを変更します。

PCLBUZ*n*_Init

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

クロック出力／ブザー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_pclbuz.c

[指定形式]

```
void    PCLBUZn_Init ( void );
```

備考 *n* は，出力端子を意味します。

[引数]

なし

[戻り値]

なし

PCLBUZ*n*_UserInit

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

クロック出力／ブザー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，PCLBUZn_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_pclbuz_user.c

[指定形式]

```
void    PCLBUZn_UserInit ( void );
```

備考 *n* は，出力端子を意味します。

[引数]

なし

[戻り値]

なし

PCLBUZ*n*_Start

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

クロック出力／ブザー出力を開始します。

[所属]

CG_pclbuz.c

[指定形式]

```
void    PCLBUZn_Start ( void );
```

備考 *n* は，出力端子を意味します。

[引数]

なし

[戻り値]

なし

PCLBUZ*n*_Stop

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

クロック出力／ブザー出力を停止します。

[所属]

CG_pclbuz.c

[指定形式]

```
void    PCLBUZn_Stop ( void );
```

備考 *n* は，出力端子を意味します。

[引数]

なし

[戻り値]

なし

PCLBUZ*n*_ChangeFreq

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

PCLBUZ*n* 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 *n*（CKS*n*）に設定されます。

[所属]

CG_pclbuz.c

[指定形式]

```
#include    "CG_pclbuz.h"
MD_STATUS   PCLBUZn_ChangeFreq ( enum PCLBUZclock clock );
```

備考 *n* は，出力端子を意味します。

[引数]

I/O	引数	説明
I	enum PCLBUZclock *clock*;	出力クロックの種類 MAINCLOCK：fMAIN MAIN2：fMAIN/2 MAIN4：fMAIN/4 MAIN8：fMAIN/8 MAIN16：fMAIN/16 MAIN2048：fMAIN/2048 MAIN4096：fMAIN/4096 MAIN8192：fMAIN/8192 SUBCLOCK：fSUB SUB2：fSUB/2 SUB4：fSUB/4 SUB8：fSUB/8 SUB16：fSUB/16 SUB32：fSUB/32 SUB64：fSUB/64 SUB128：fSUB/128

備考 fMAIN 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

[使用例]

以下に，外部接続のスイッチ SW を押下した際に割り込み INTP0 が発生するシステムにおいて，スイッチを押下するたびに PCLBUZ0 端子への出力クロックを変更する例を示します。

【CG_main.c】

```
#include    "CG_macrodriver.h"
#include    "CG_pclbuz.h"
BOOL    gFlag;                                  /* SW 押下フラグ */
#define PCLBUZ_FREQUENCY    8                   /* 出力クロックの種類総数 */
const   enum    PCLBUZclock gClock[PCLBUZ_FREQUENCY] = {    /* 出力クロックの種類 */
    PCLBUZ_OUTCLK_fSUB0, PCLBUZ_OUTCLK_fSUB1, PCLBUZ_OUTCLK_fSUB2,
    PCLBUZ_OUTCLK_fSUB3, PCLBUZ_OUTCLK_fSUB4, PCLBUZ_OUTCLK_fSUB5,
    PCLBUZ_OUTCLK_fSUB6, PCLBUZ_OUTCLK_fSUB7
};
void main ( void ) {
    int     index = 0;
    gFlag = 0;                                  /* SW 押下フラグの初期化 */
    ……
    PCLBUZ0_Start ();                           /* クロック出力／ブザー出力の開始 */
    while ( 1 ) {
        if ( gFlag ) {                          /* 割り込み INTP0 の発生待ち */
            PCLBUZ_ChangeFreq ( gClock[index++] );  /* 出力クロックの変更 */
            if ( index >= PCLBUZ_FREQUENCY ) {
                index = 0;
            }
            gFlag = 0;                          /* SW 押下フラグのクリア */
        }
    }
}
```

【CG_int_user.c】

```
#include    "CG_macrodriver.h"
extern  BOOL    gFlag;                      /* SW 押下フラグ */
__interrupt void MD_INTP0 ( void ) {        /* 割り込み INTP0 発生時の割り込み処理 */
    gFlag = 1;                              /* SW 押下フラグの設定 */
}
```

3. 3. 15 LCD コントローラ／ドライバ

以下に，Applilet3 が LCD コントローラ／ドライバ用として出力する API 関数の一覧を示します。

表 3—16 LCD コントローラ／ドライバ用 API 関数

API 関数名	機能概要
LCD_Init	LCD コントローラ／ドライバ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。
LCD_UserInit	LCD コントローラ／ドライバに関するユーザ独自の初期化処理を行います。
LCD_DisplayOn	LCD コントローラ／ドライバを表示オン状態にします。
LCD_DisplayOff	LCD コントローラ／ドライバを表示オフ状態にします。
LCD_VoltageOn	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作許可としたのち，非選択信号をセグメント端子から出力します。
LCD_VoltageOff	LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，グランド・レベルの信号をセグメント端子／コモン端子に出力します。

LCD_Init

【Lx3】

LCD コントローラ／ドライバ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_lcd.c

[指定形式]

```
void    LCD_Init ( void );
```

[引数]

なし

[戻り値]

なし

LCD_UserInit

【Lx3】

LCD コントローラ／ドライバ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，LCD_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_lcd_user.c

[指定形式]

```
void    LCD_UserInit ( void );
```

[引数]

なし

[戻り値]

なし

LCD_DisplayOn

【Lx3】

LCD コントローラ／ドライバを表示オン状態にします。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_DisplayOn ( void );
```

[引数]

なし

[戻り値]

なし

LCD_DisplayOff

【Lx3】

LCD コントローラ／ドライバを表示オフ状態にします。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_DisplayOff ( void );
```

[引数]

なし

[戻り値]

なし

LCD_VoltageOn

【Lx3】

LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，非選択信号をセグメント端子から出力します。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_VoltageOn ( void );
```

[引数]

なし

[戻り値]

なし

LCD_VoltageOff

【Lx3】

LCD コントローラ／ドライバの昇圧回路，および容量分割回路を動作停止としたのち，グランド・レベルの信号をセグメント端子／コモン端子から出力します。

[所属]

CG_lcd.c

[指定形式]

```
void    LCD_VoltageOff ( void );
```

[引数]

なし

[戻り値]

なし

3. 3. 16 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—17 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_Hold	DMA 起動要求を保留します。
DMAn_Restart	DMA 起動要求の保留を解除します。
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 動作許可状態の際，DMA 転送を開始します。

DMA*n*_Init

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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*_Hold

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

DMA 起動要求を保留します。

備考 DMA 起動要求の保留を解除する際は，DMAn_Restart を呼び出します。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Hold ( 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*_Restart

【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

DMA 起動要求の保留を解除します。

[所属]

CG_dma.c

[指定形式]

```
void    DMAn_Restart ( 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*_CheckStatus

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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]

- 【Fx3】【Ix3】【Kx3-A】【Kx3-L】【Lx3】

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( UCHAR sfraddr, USHORT ramaddr, USHORT count );
```

- 【Kx3】

```
#include    "CG_macrodriver.h"
MD_STATUS   DMAn_SetData ( USHORT ramaddr, USHORT 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	UCHAR *sfraddr*;	転送先／転送元の SFR アドレス
I	USHORT *ramaddr*;	転送先／転送元の RAM アドレス
I	USHORT *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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

DMA 動作許可状態の際，DMA 転送を開始します。

[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，ソフトウエア・トリガを DMA 転送の起動要因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. 17 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—18 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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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_Stop

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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

【Fx3】【Ix3】【Kx3】【Kx3-A】【Kx3-L】【Lx3】

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 対象デバイスが 78K0R/Fx3，または 78K0R/Ix3 の場合，*level* に指定可能な値は LVILEVEL0 ～ LVILEVEL9 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
MD_ARGERROR	引数の指定が不正

付録 A 索 引

【I】

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

© 2010 Renesas Electronics Corporation. All rights reserved.
Colophon 1.0