お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

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。

User's Manual

16

M16C/62Pグループ
Renesas Embedded Application Programming Interface
リファレンスマニュアル

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2007.11

Renesas Embedded
Application Programming Interface
リファレンス マニュアル
<M16C/62P グループ>

Rev. 1.00

目次

1. はじめに

Renessas Embedded Application Programming Interface (API) は、株式会社ルネサステクノロジ製マイクロコンピュータ用統一 API です。

2. ドライバ

2.1 概要

ここで説明するライブラリは､マイクロコンピュータの周辺機能制御プログラム（ペリフェラルドライバ）を提供するものです。Renesas API を使用することによって、ペリフェラルドライバをユーザ プログラムに埋め込むことができます。

2.2 ドライバの機能

本ライブラリでは、ペリフェラル ドライバとして次の機能を用意しています。

(1) シリアル I/O 制御機能
通信データの送信/受信を制御および管理するとともに、シリアル通信条件を設定・解除するシリアル I/F ドライバを構成します。

(2) タイマ制御機能
タイマの動作を制御するとともに、タイマの動作条件を設定・解除するタイマ ドライバを構成します。

(3) I/O ポート制御機能
データの読み込み/書き込み操作を制御するとともに、I/O ポートの使用条件を設定・解除する I/O ポート ドライバを構成します。

(4) 外部割り込み制御機能
割り込み動作を制御するとともに、外部割り込みの使用条件を設定・解除する外部割り込みドライバを構成します。

(5) A/D 変換器制御機能
A/D 変換器の動作を制御するとともに､A/D 変換器の使用条件を設定･解除する A/D 変換器ドライバを構成します。

2.3 シリアル I/F ドライバ

シリアル I/F ドライバは、シリアル通信の設定、設定のクリア、データの送受信、およびシリアル通信の状態の制御を行います。

シリアル I/F ドライバには、シングルデータ送受信 API と、マルチデータ送受信 API の 2 種類があります。

2.4 タイマ ドライバ

タイマ ドライバは、以下のモードに応じて、タイマの設定、タイマ設定のクリア、タイマの動作の制御、およびカウンタ値の取得を行います。

- タイマ モード
- イベント カウンタ モード
- パルス幅変調モード (PWM モード)
- パルス周期測定モード
- パルス幅測定モード

2.4.1 タイマ モード

このモードでは、タイマは内部で生成されたカウント ソースをカウントします。アンダーフローまたはオーバーフロー割り込みが発生したときは、設定したコールバック関数をコールします。

2.4.2 イベント カウンタ モード

このモードでは、タイマは入力ピンから供給されるか、他のタイマからオーバーフローもしくはアンダーフローした外部信号をカウントします。アンダーフローまたはオーバーフロー割り込みが発生したときは、設定したコールバック関数をコールします。

2.4.3 パルス幅変調モード (PWM モード)

このモードでは、タイマは指定された幅のパルスを連続して出力します。アンダーフローまたはオーバーフロー割り込みが発生したときは、設定したコールバック関数をコールします。

2.4.4 パルス周期測定モード

このモードでは、タイマは入力ピンから供給される外部信号のパルス周期を測定します。アンダーフローまたはオーバーフロー割り込みが発生したときは、設定したコールバック関数をコールします。

2.4.5 パルス幅測定モード

このモードでは、タイマは入力ピンから供給される外部信号のパルス幅を測定します。アンダーフローまたはオーバーフロー割り込みが発生したときは、設定したコールバック関数をコールします。

2.5 I/O ポート ドライバ

I/O ポート ドライバは、入力または出力の I/O ポートの設定、I/O ポートへのデータの書き込み、および I/O ポートからのデータの読み出しを行います。

2.6 外部割り込みドライバ

外部割り込みドライバは、外部割り込みの設定、外部割り込みの制御、外部割り込みフラグの取得、および外部割り込みフラグのクリアを行います。

2.7 A/D 変換器ドライバ

A/D 変換器ドライバは、A/D 変換器の設定、A/D 変換器の制御、A/D 変換器の設定破棄、A/D 変換器の値の取得、A/D 変換器の状態取得、および A/D 変換器の状態クリアを行います。

3. 標準の型

このセクションでは、ライブラリで定義されている標準の型について説明します。型の設定の詳細については、各 API の説明を参照してください。

標準の型	説明
Boolean	Boolean 型は、成功 (RAPI_TRUE (= 1)) または失敗 (RAPI_FALSE (= 0)) のいずれかを示す enum 型データを表します。
VoidFuncNotify	VoidFuncNotify 型は、登録される通知関数の型を表します。

4. ライブラリ リファレンス

4.1 周辺機器機能別の API 一覧

下記の表は、ルネサス埋め込み API を周辺機器の機能別に分類したものです。

番号	機能分類	API	API の動作
1	シングルデータ シリアル I/O	R_BasicOpenSerialDriver	シリアル ポートのオープン
2		R_BasicCloseSerialDriver	シリアル ポートのクローズ
3		R_BasicSetSerialFormat	シリアル通信設定
4		R_BasicStartSerialReceiving	1 データの受信
5		R_BasicStartSerialSending	1 データの送信
6		R_BasicReceivingStatusRead	受信状態の読み込み
7		R_BasicSendingStatusRead	送信状態の読み込み
8		R_BasicStopSerialReceiving	受信の停止
9		R_BasicStopSerialSending	送信の停止
10	マルチデータ シリアル I/O	R_OpenSerialDriver	シリアル ポートのオープン
11		R_CloseSerialDriver	シリアル ポートのクローズ
12		R_ConfigSerialDriverNotify	通知関数の登録
13		R_SetSerialFormat	シリアル通信設定
14		R_SetSerialInterrupt	送受信割り込み設定
15		R_StartSerialReceiving	受信の開始
16		R_StartSerialSending	送信の開始
17		R_StopSerialReceiving	受信の停止
18		R_StopSerialSending	送信の停止
19		R_PollingSerialReceiving	ポーリング受信
20		R_PollingSerialSending	ポーリング送信
21	タイマ	R_CreateTimer	タイマ モード設定
22		R_EnableTimer	タイマ モード動作制御
23		R_DestroyTimer	タイマ モード設定破棄
24		R_CreateEventCounter	イベント カウンタ モード設定
25		R_EnableEventCounter	イベント カウンタ モード動作制御
26		R_DestroyEventCounter	イベント カウンタ モード設定破棄
27		R_GetEventCounter	イベント カウンタ モードのカウンタ値取得
28		R_CreatePulseWidthModulationMode	パルス幅変調モード設定
29		R_EnablePulseWidthModulationMode	パルス幅変調モード動作制御
30		R_DestroyPulseWidthModulationMode	パルス幅変調モード設定破棄
31		R_CreatePulsePeriodMeasurementMode	パルス周期測定モード設定
32		R_EnablePulsePeriodMeasurementMode	パルス周期測定モード動作制御

33		__DestroyPulsePeriodMeasurementMode	パルス幅測定モード設定破棄
34		__GetPulsePeriodMeasurementMode	パルス周期測定モードの測定値取得
35		__CreatePulseWidthMeasurementMode	パルス幅測定モード設定
36		__EnablePulseWidthMeasurementMode	パルス幅測定モード動作制御
37	タイマ	__DestroyPulseWidthMeasurementMode	パルス幅測定モード設定破棄
38		__GetPulseWidthMeasurementMode	パルス幅測定モードの測定値取得
39		__SetTimerRegister	タイマ レジスタ設定
40		__EnableTimerRegister	タイマ レジスタ動作制御
41		__ClearTimerRegister	タイマ レジスタ クリア
42		__GetTimerRegister	タイマ レジスタ値取得
43	I/O ポート	__SetIOPort	I/O ポートの設定
44		__ReadIOPort	I/O ポートからの読み出し
45		__WriteIOPort	I/O ポートへの書き込み
46		__SetIOPortRegister	I/O ポート レジスタの設定
47		__ReadIOPortRegister	I/O ポート レジスタからの読み出し
48		__WriteIOPortRegister	I/O ポート レジスタへの書き込み
49	外部割り込み	__SetInterrupt	外部割り込みの設定
50		__EnableInterrupt	外部割り込みの制御
51		__GetInterruptFlag	外部割り込みフラグの状態取得
52		__ClearInterruptFlag	外部割り込みフラグのクリア
53	A/D 変換器	__CreateADC	A/D 変換器の設定
54		__EnableADC	A/D 変換器の動作制御
55		__DestroyADC	A/D 変換器の設定破棄
56		__GetADC	A/D 変換値の取得（レジスタ指定）
67		__GetADCAll	A/D 変換値の取得（全レジスタ）

4.2 各 API の説明

このセクションでは、各 API について説明し、その使用方法とそれぞれのプログラム例を示します。

各 API の説明は、以下の項目で構成されています。

- **概要** ： その関数で実行される処理の内容の概要。関数の構文の後に、引数について簡単に説明しています。
- **説明** ： 関数について説明し、使用方法を詳細に示します。
- **戻り値** ： 関数の戻り値について説明します。
- **機能** ： 関数の機能分類について示します。
- **参考** ： 関連する関数を示します。
- **備考** ： この API を使用する際の注意事項について説明します。
- **プログラム例** ： 関数の使用方法を示すプログラムを例示します。

4.2.1 シリアル I/O
__BasicOpenSerialDriver

概要

<シリアル ポートのオープン>

Boolean __BasicOpenSerialDriver(unsigned long data)

data	設定データ

説明

指定されたシリアル ポートのオープンと初期化をおこないます。

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

__BasicCloseSerialDriver

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル ドライバを開きます */
return __BasicOpenSerialDriver( RAPI_COM1 );
}
```

__BasicCloseSerialDriver

概要

<シリアル ポートのクローズ>

Boolean __BasicCloseSerialDriver(unsigned long data)

data	設定データ

説明

指定されたシリアル ポートをクローズします。data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

__BasicOpenSerialDriver

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル ドライバを閉じます */
return __BasicCloseSerialDriver( RAPI_COM1 );
}
```

__BasicSetSerialFormat

概要

<シリアル通信設定>

Boolean __BasicSetSerialFormat(unsigned long data1, unsigned char data2)

data1	設定データ 1
data2	設定データ 2

説明

指定されたパラメータに従って、シリアル通信を設定します。

[data1]

data 1 には次の値を設定することができます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

シリアル通信モードについては、以下の値を設定できます。

(UART0、UART1、UART2)

RAPI_SM_SYNC	クロック同期シリアル通信モード
RAPI_SM_ASYNC	クロック非同期シリアル通信モード

(SI/O3、SI/O4)

RAPI_SIO_SM_SYNC	クロック同期シリアル通信モード

クロック非同期シリアル通信のデータ長フォーマットについては、以下の値が設定できます。

クロック同期シリアル通信モードで API を使用する場合には、これらの値を設定しないでください。

(UART0、UART1、UART2)

RAPI_BIT_7	データ長 7 ビットで転送	RAPI_BIT_8	データ長 8 ビットで転送
RAPI_BIT_9	データ長 9 ビットで転送		

シリアル通信のクロック ソースについては、以下の値を設定できます。

(UART0、UART1、UART2)

RAPI_CKDIR_INT	シリアル通信のクロック ソースとして内部クロックを使用します。
RAPI_CKDIR_EXT	シリアル通信のクロック ソースとして外部クロックを使用します。

(SI/O3、SI/O4)

RAPI_SIO_CKDIR_INT	シリアル通信のクロック ソースとして内部クロックを使用します。
RAPI_SIO_CKDIR_EXT	シリアル通信のクロック ソースとして外部クロックを使用します。

クロック非同期シリアル通信のストップ ビット長については、以下の値が設定できます。

クロック同期シリアル通信モードで API を使用する場合には、これらの値を設定しないでください。

(UART0、UART1、UART2)

RAPI_STPB_1	1 ストップ ビット	RAPI_STPB_2	2 ストップ ビット

クロック非同期シリアル通信のパリティ長については、以下の値が設定できます。
クロック同期シリアル通信モードで API を使用する場合には、これらの値を設定しないでください。
(UART0、UART1、UART2)

RAPI_PARITY_NON	パリティ ビットなし	RAPI_PARITY_EVEN	偶数パリティ ビット
RAPI_PARITY_ODD	奇数パリティ ビット		

シリアル通信のクロック極性については、以下の値を設定できます。
クロック非同期シリアル通信モードで API を使用する場合には、これらの値を設定しないでください。
(UART0、UART1、UART2)

RAPI_DPOL_NON	極性反転なし	RAPI_DPOL_INV	極性反転あり

(SI/O3、SI/O4)

RAPI_SIO_DPOL_NON	極性反転なし	RAPI_SIO_DPOL_INV	極性反転あり

内蔵ボー レート ジェネレータのカウント ソースについては、以下の値が設定できます。
(UART0、UART1、UART2)

RAPI_BCSS_F1	f1SIO	RAPI_BCSS_F2	f2SIO
RAPI_BCSS_F8	f8SIO	RAPI_BCSS_F32	f32SIO

(SI/O3、SI/O4)

RAPI_SIO_BCSS_F1	f1SIO	RAPI_SIO_BCSS_F2	f2SIO
RAPI_SIO_BCSS_F8	f8SIO	RAPI_SIO_BCSS_F32	f32SIO

_CTS/_RTS 関数については、以下の値を設定できます。
クロック同期シリアル通信モードで内部クロックを使用するように選択している場合には、_RTS 関数は無効になります。
(UART0、UART1、UART2)

RAPI_CTSRTS_DIS	_CTS/_RTS 関数は使用されません。
RAPI_CTS_SEL	_CTS 関数が選択されます。
RAPI_RTS_SEL	_RTS 関数が選択されます。

転送フォーマットについては、以下の値を設定できます。
クロック非同期シリアル通信モードで使用するデータ長として 7 ビットまたは 9 ビットを選択している場合は、この値を設定しないでください。
(UART0、UART1、UART2)

RAPI_LSB_SEL	LSB 先行	RAPI_MSB_SEL	MSB 先行

(SI/O3、SI/O4)

RAPI_SIO_LSB_SEL	LSB 先行	RAPI_SIO_MSB_SEL	MSB 先行

シリアル データ ロジックの切り替えについては、以下の値を設定できます。
(UART0、UART1、UART2)

RAPI_LOGIC_NO_REV	送信バッファ レジスタに書き込まれた値は、ロジックが反転されません。
RAPI_LOGIC_REV	送信バッファ レジスタに書き込まれた値は、送信前に反転されます。

[data2]

通信速度の N による除算値を設定します。

戻り値

シリアル通信が正常に設定されると RAPI_TRUE を返し、設定に失敗した場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

Boolean func( void )
{
/* RAPI_COM1 のデータをシリアル ドライバに設定します */
Return _BasicSetSerialFormat(RAPI_COM1 | RAPI_SM_SYNC | RAPI_CKDIR_INT |
             RAPI_BCSS_F1 | RAPI_DPOL_NON | RAPI_LSB_SEL, 20);
}
```

__BasicStartSerialReceiving

概要

<1 データの受信>

Boolean __BasicStartSerialReceiving(unsigned long data)

data	設定データ

説明

シリアル通信の 1 データ受信を開始します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル通信のデータ受信が正常に開始されると RAPI_TRUE を返し、開始しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__BasicReceivingStatusRead、__BasicStopSerialReceiving

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    ..........
__BasicStartSerialReceiving( RAPI_COM1 );
    ..........
}
```

__BasicStartSerialSending

概要

<1 データの送信>

Boolean _BasicStartSerialSending(unsigned long data1, unsigned int data2)

data	設定データ
data	送信データ

説明

シリアル通信の 1 データ送信を開始します。

data 1 には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル通信のデータ送信が正常に開始されると RAPI_TRUE を返し、開始しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__BasicSendingStatusRead、__BasicStopSerialSending

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

Void func( void )
{
    ..........
__BasicStartSerialSending( RAPI_COM1, 0x00AA );
    ..........
}
```

__BasicReceivingStatusRead

概要

<受信状態の読み込み>

unsigned int __BasicReceivingStatusRead(unsigned long data)

data	設定データ

説明

シリアル通信の受信状態を返します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル通信の受信状態を返します。戻り値は以下のいずれかになります。

(UART0、UART1、UART2)

RAPI_RX_INCOMPLETE	受信が完了していません。
上記以外	受信が完了しました。値は、UARTi 受信バッファ レジスタ (i = 0～2) から読み取ります。

(SI/O3、SI/O4)

RAPI_RX_INCOMPLETE	受信が完了していません。
上記以外	受信が完了しました。低次 8 ビット:値は、SI/Oi 送受信レジスタ (i = 3、4) から読み取ります。

機能

シリアル I/O

参考

__BasicStartSerialReceiving、__BasicStopSerialReceiving

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
unsigned int rcv_data;

    ..........
rcv_data = __BasicReceivingStatusRead( RAPI_COM1 );
    ..........
}
```

__BasicSendingStatusRead

概要

<送信状態の読み込み>

Boolean __BasicSendingStatusRead(unsigned long data)

data	設定データ

説明

シリアル通信の送信状態を返します。data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

データが送信バッファにない場合には RAPI_TRUE を返し、データがある場合には RAPI_FALSE を返します。

機能

シリアル I/O

参考

__BasicStartSerialSending、__BasicStopSerialSending

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    ..........
if (__BasicSendingStatusRead( RAPI_COM1 ) == RAPI_TRUE) {
/* 送信完了 */
    }
    ..........
}
```

__BasicStopSerialReceiving

概要

<受信の停止>

Boolean Rapi_BasicStopSerialReceiving(unsigned long data)

data	設定データ

説明

シリアル通信のデータ受信を停止します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

戻り値

シリアル通信のデータ受信が正常に停止されると RAPI_TRUE を返し、停止しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__BasicStartSerialReceiving

備考

- M16C SI/03 および SI/04 では、この API は使用できません。
- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル通信でのデータ受信を停止します */
    __BasicStopSerialReceiving ( RAPI_COM1 );
}
```

__BasicStopSerialSending

概要

<送信の停止>

Boolean __BasicStopSerialSending(unsigned long data)

data	設定データ

説明

シリアル通信のデータ送信を停止します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

戻り値

シリアル通信のデータ送信が正常に停止されると RAPI_TRUE を返し、停止しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__BasicStartSerialSending

備考

- M16C SI/03 および SI/04 では、この API は使用できません。
- クロック同期シリアル通信モードで動作しているときは、この API によってデータ受信も同時に停止されます。
- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル通信でのデータ送信を停止します */
    __BasicStopSerialSending ( RAPI_COM1 );
}
```

__OpenSerialDriver

概要

<シリアル ポートのオープン>

Boolean __OpenSerialDriver(unsigned long data)

data	設定データ

説明

指定されたシリアル ポートのオープンと初期化を行います。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

__CloseSerialDriver

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル ドライバを開きます */
    return __OpenSerialDriver( RAPI_COM1 );
}
```

__CloseSerialDriver

概要

<シリアル ポートのクローズ>

Boolean __CloseSerialDriver(unsigned long data)

data	設定データ

説明

指定されたシリアル ポートをクローズします。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

__OpenSerialDriver

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル ドライバを閉じます */
    return __CloseSerialDriver( RAPI_COM1 );
}
```

__ConfigSerialDriverNotify

概要

<通知関数の登録>

Boolean __ConfigSerialDriverNotify(unsigned long data, VoidFuncNotify *func)

data	設定データ
func	登録する関数のポインタ

説明

シリアル通信のさまざまな送受信情報を取得するのに必要な通知関数を登録します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

[func]

func で登録する関数は、ユーザがシリアル I/O ドライバに対して提供する必要があります。

シリアル I/O ドライバは、func に登録された関数をコールします。

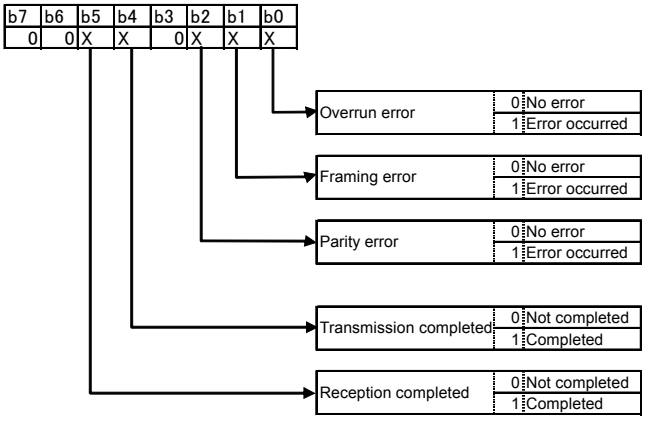
シリアル I/O ドライバは、引数によって送受信状態をユーザに通知します。

登録する関数のタイプについては下記に示します。

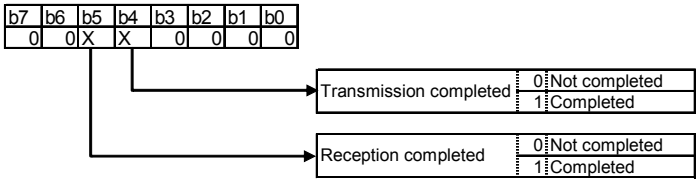
void “任意の関数名” (unsigned char notify);

引数の詳細は下記に示します。

(UART0、UART1、UART2)



(SI/O3、SI/O4)



戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

__StartSerialReceiving、__StartSerialSending

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void Notify(unsigned char result) {
    if ((result&RAPI_OVER_ERR) == RAPI_OVER_ERR) {
/* オーバーラン エラー */
    }
    if ((result&RAPI_FRAMING_ERR) == RAPI_FRAMING_ERR) {
/* フレーミング エラー */
    }
    if ((result&RAPI_PARITY_ERR) == RAPI_PARITY_ERR) {
/* パリティ エラー */
    }
    if ((result&RAPI_TX_END) == RAPI_TX_END) {
/* 送信完了 */
    }
    if ((result&RAPI_RX_END) == RAPI_RX_END) {
/* 受信完了 */
    }
}

Boolean func( void )
{
/* RAPI_COM1 のコールバック関数をシリアル ドライバに設定します */
    return __ConfigSerialDriverNotify( RAPI_COM1, Notify );
}
```

__SetSerialFormat

概要

<シリアル通信設定>

Boolean __SetSerialFormat(unsigned long data1, unsigned char data2)

data1	設定データ 1
data2	設定データ 2

説明

指定されたパラメータに従って、シリアル通信を設定します。
パラメータの詳細については、__BasicSetSerialFormat の説明を参照してください。

戻り値

シリアル通信が正常に設定されると RAPI_TRUE を返し、設定に失敗した場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

Boolean func( void )
{
/* RAPI_COM1 のデータをシリアル ドライバに設定します */
   return __SetSerialFormat(RAPI_COM1 | RAPI_SM_SYNC | RAPI_CKDIR_INT |
RAPI_BCSS_F1 | RAPI_DPOL_NON | RAPI_LSB_SEL, 20);
}
```

__SetSerialInterrupt

概要

<シリアル割り込み設定>

Boolean __SetSerialInterrupt(unsigned long data)

data	設定データ

説明

指定されたパラメータに従って、シリアル割り込みを設定します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

割り込み設定については、以下の値を設定できます。

(UART0、UART1、UART2)

RAPI_INT_TX_DIS	送信割り込み無効
RAPI_INT_TX_LV_1	送信割り込み優先順位レベル 1
RAPI_INT_TX_LV_2	送信割り込み優先順位レベル 2
RAPI_INT_TX_LV_3	送信割り込み優先順位レベル 3
RAPI_INT_TX_LV_4	送信割り込み優先順位レベル 4
RAPI_INT_TX_LV_5	送信割り込み優先順位レベル 5
RAPI_INT_TX_LV_6	送信割り込み優先順位レベル 6
RAPI_INT_TX_LV_7	送信割り込み優先順位レベル 7
RAPI_INT_RX_DIS	受信割り込み無効
RAPI_INT_RX_LV_1	受信割り込み優先順位レベル 1
RAPI_INT_RX_LV_2	受信割り込み優先順位レベル 2
RAPI_INT_RX_LV_3	受信割り込み優先順位レベル 3
RAPI_INT_RX_LV_4	受信割り込み優先順位レベル 4
RAPI_INT_RX_LV_5	受信割り込み優先順位レベル 5
RAPI_INT_RX_LV_6	受信割り込み優先順位レベル 6
RAPI_INT_RX_LV_7	受信割り込み優先順位レベル 7

(SI/O3、SI/O4)

RAPI_INT_SIO_DIS	SI/O 割り込み無効
RAPI_INT_SIO_LV_1	SI/O 割り込み優先順位レベル 1
RAPI_INT_SIO_LV_2	SI/O 割り込み優先順位レベル 2
RAPI_INT_SIO_LV_3	SI/O 割り込み優先順位レベル 3
RAPI_INT_SIO_LV_4	SI/O 割り込み優先順位レベル 4
RAPI_INT_SIO_LV_5	SI/O 割り込み優先順位レベル 5
RAPI_INT_SIO_LV_6	SI/O 割り込み優先順位レベル 6
RAPI_INT_SIO_LV_7	SI/O 割り込み優先順位レベル 7

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能　シリアル I/O

参考

備考
- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

Boolean func( void )
{
/* RAPI_COM1 の割り込みをシリアル ドライバに設定します */
   return    __SetSerialInterrupt(    RAPI_COM1    |    RAPI_INT_TX_LV_1    |
RAPI_INT_RX_LV_2 );
}
```

__StartSerialReceiving

概要

<受信の開始>

Boolean __StartSerialReceiving(unsigned long data, unsigned char wordNum, unsigned int *RcvDtBuf)

data	設定データ
wordNum	受信したワード数
RcvDtBuf	受信したデータが格納されるバッファへのポインタ

説明

シリアル通信の受信を開始し、受信したデータを指定されたワード数だけ取得します。受信データの取得が完了すると、この API は通知関数 (通知関数が登録されている場合) をコールします。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル通信の受信が正常に開始されると RAPI_TRUE を返し、開始しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__ConfigSerialDriverNotify、__StopSerialReceiving

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];
void func( void )
{
/* シリアル通信で受信した 5 ワードのデータを取得します */
    __StartSerialReceiving( RAPI_COM1, 5, buffer );
}
```

__StartSerialSending

概要

<送信の開始>

Boolean __StartSerialSending(unsigned long data, unsigned char wordNum, unsigned int *SndDtBuf)

data	設定データ
wordNum	送信したワード数
SndDtBuf	送信データへのポインタ

説明

シリアル通信の送信を開始し、送信データを指定されたワード数だけ送信バッファに書き込みます。すべての送信データの送信が完了すると、この API は通知関数 (通知関数が登録されている場合) をコールします。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル通信の送信が正常に開始されると RAPI_TRUE を返し、開始しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__ConfigSerialDriverNotify、__StopSerialSending

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];
void func( void )
{
/* シリアル通信の送信バッファに 5 ワードのデータを設定します */
    __StartSerialSending( RAPI_COM1, 5, buffer );
}
```

__StopSerialReceiving

概要

<受信の停止>

Boolean __StopSerialReceiving(unsigned long data)

data	設定データ

説明

シリアル通信の受信を停止します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

戻り値

シリアル通信の受信が正常に停止されると RAPI_TRUE を返し、停止しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__StartSerialReceiving

備考

- M16C SI/03 および SI/04 では、この API は使用できません。
- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル通信でのデータ受信を停止します */
    __StopSerialReceiving ( RAPI_COM1 );
}
```

__StopSerialSending

概要

<送信の停止>

Boolean __StopSerialSending(unsigned long data)

data	設定データ

説明

シリアル通信の送信を停止します。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

戻り値

シリアル通信の送信が正常に停止されると RAPI_TRUE を返し、停止しなかった場合は RAPI_FALSE を返します。

機能

シリアル I/O

参考

__StartSerialReceiving

備考

- M16C SI/03 および SI/04 では、この API は使用できません。
- クロック同期シリアル通信モードで動作しているときは、この API によってデータ受信も同時に停止されます。
- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
/* シリアル通信でのデータ送信を停止します */
    __StopSerialSending ( RAPI_COM1 );
}
```

__PollingSerialReceiving

概要

<ポーリング受信>

Boolean __PollingSerialReceiving(unsigned long data)

data	設定データ

説明

ポーリングによるシリアル通信の受信を実行します。この API は、__StartSerialReceiving で指定された量のデータを受信します。受信データの取得が完了すると、通知関数 (通知関数が登録されている場合) をコールします。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

__ConfigSerialDriverNotify, __SetSerialInterrupt, __StartSerialReceiving

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];

void func( void )
{
/* 受信割り込み無効 */
    __SetSerialInterrupt( RAPI_COM1 | RAPI_INT_TX_DIS | RAPI_INT_RX_DIS );
/* 受信開始 */
    __StartSerialReceiving( RAPI_COM1, 5, buffer );
    while(1){
        __PollingSerialReceiving( RAPI_COM1 );
    }
}
```

__PollingSerialSending

概要

<ポーリング送信>

Boolean __PollingSerialSending(unsigned long data)

data	設定データ

説明

ポーリングによるシリアル通信の送信を実行します。この API は、__StartSerialSending で指定された量のデータを __StartSerialSending で指定された送信データ バッファから送信します。すべての送信データの送信が完了すると、通知関数 (通知関数が登録されている場合) をコールします。

[data]

data には次の値を設定することができます。

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

戻り値

シリアル ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

シリアル I/O

参考

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];
void func( void )
{
/* 送信割り込み無効 */
    __SetSerialInterrupt( RAPI_COM1 | RAPI_INT_TX_DIS | RAPI_INT_RX_DIS );
/* 送信開始 */
    __StartSerialSending( RAPI_COM1, 5, buffer );
    while(1){
        __PollingSerialSending( RAPI_COM1 );
    }
}
```

4.2.2 タイマ
__CreateTimer

概要

<タイマ モード設定>

Boolean __CreateTimer(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	設定データ 1
data2	設定データ 2
data3	設定データ 3
data4	設定データ 4
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

指定したタイマのタイマ モードを設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を使用します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を使用します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を使用します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を使用します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を使用します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。
RAPI_F1	カウント ソースに f_1 を選択します。
RAPI_F2	カウント ソースに f_2 を選択します。
RAPI_F8	カウント ソースに f_8 を選択します。
RAPI_F32	カウント ソースに f_{32} を選択します。
RAPI_FC32	カウント ソースに f_{c32} を選択します。
RAPI_TIMER_ON	タイマが __CreateTimer で動作を開始するように設定します。
RAPI_TIMER_OFF	タイマが __CreateTimer で動作を停止するように設定します。
RAPI_GATE_L	TA_{iIN} ピンの入力が低いままになっている時間をカウントするゲート機能を選択します。
RAPI_GATE_H	TA_{iIN} ピンの入力が高いままになっている時間をカウントするゲート機能を選択します。
RAPI_PULSE_ON	TA_{iIN} ピンからパルスを出力することを選択します。
RAPI_PULSE_OFF	TA_{iIN} ピンからパルスを出力しないことを選択します。

• タイマ A を使用するときに指定できる定義値 (RAPI_TIMER_A0 ～ RAPI_TIMER_A4 が指定されているとき)

(カウント ソース)　{ RAPI_F1、RAPI_F2、RAPI_F8、RAPI_F32、RAPI_FC32 } のいずれかを指定します。デフォルト値は RAPI_F2 です。

(動作状態設定)　{ RAPI_TIMER_ON、RAPI_TIMER_OFF } のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

(パルス出力状態)　{ RAPI_PULSE_ON、RAPI_PULSE_OFF } のいずれかを指定します。デフォルト値は RAPI_PULSE_OFF です。

(ゲート機能)　{ RAPI_GATE_L、RAPI_GATE_H } のいずれかを指定します。省略した場合には「ゲート機能なし」に設定されます。

• タイマ B を使用するときに指定できる定義値 (RAPI_TIMER_B0 ～ RAPI_TIMER_B5 が指定されているとき)

(カウント ソース)　{ RAPI_F1、RAPI_F2、RAPI_F8、RAPI_F32、RAPI_FC32 } のいずれかを指定します。デフォルト値は RAPI_F2 です。

(動作状態設定)　{ RAPI_TIMER_ON、RAPI_TIMER_OFF } のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

[data3]

タイマ レジスタに設定する値を 16 ビットで指定します。

[data4]

0 を指定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (タイマ モード)

参考

__EnableTimer、__DestroyTimer

備考

• 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
/* タイマ A0 をタイマ モードに設定します */
    __CreateTimer(  RAPI_TIMER_A0|RAPI_TIMER_ON|RAPI_F8,  5,  0x80,  0,
TimerIntFunc );
}
```

__EnableTimer

概要

<タイマ レジスタ動作制御>

Boolean __EnableTimer(unsigned long data)

data	設定データ

説明

指定したタイマ モードに設定されているタイマの動作を、タイマを開始または停止して操舵します。

[data]

data については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。
RAPI_TIMER_ON	タイマ モードに設定されたタイマの動作を開始します。
RAPI_TIMER_OFF	タイマ モードに設定されたタイマの動作を停止します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (タイマ モード)

参考

__CreateTimer、__DestroyTimer

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* タイマ モードのタイマ A1 を無効にします */
    __EnableTimer( RAPI_TIMER_A1 | RAPI_TIMER_OFF );
}
```

__DestroyTimer

概要

<タイマ モード設定破棄>

Boolean __DestroyTimer(unsigned long data)

data	設定データ

説明

指定したタイマ モードに設定されているタイマの設定を破棄します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (タイマ モード)

参考

__CreateTimer、__EnableTimer

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* タイマ モードのタイマ A2 の設定を破棄します */
    __DestroyTimer( RAPI_TIMER_A2 );
}
```

__CreateEventCounter

概要

<イベント カウンタ モード設定>

Boolean __CreateEventCounter(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	設定データ 1
data2	設定データ 2
data3	設定データ 3
data4	設定データ 4
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

指定したタイマのイベント タイマ モードを設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を使用します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を使用します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を使用します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を使用します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を使用します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。
RAPI_EV_EXTERNAL	カウント ソースについて、TA_{iIN} ピン (タイマ Ai を使用する場合) または TB_{iIN} ピン (タイマ Bi を使用する場合) の外部信号入力を選択します。
RAPI_EV_TIMER_AJ	カウント ソースについて、タイマ Aj (j = i-1、ただし i = 0 の場合は j = 4) のオーバーフローまたはアンダーフローを選択します。
RAPI_EV_TIMER_AK	カウント ソースについて、タイマ Ak (k = i+1、ただし k = 4 の場合は j = 0) のオーバーフローまたはアンダーフローを選択します。
RAPI_EV_TIMER_B2	カウント ソースについて、タイマ B2 のオーバーフローまたはアンダーフローを選択します。
RAPI_EV_TIMER_BJ	カウント ソースについて、タイマ Bj (j = i-1、ただし j = 2 の場合は i = 0、i = 3 の場合は j = 5) のオーバーフローまたはアンダーフローを選択します。
RAPI_TIMER_ON	タイマが __CreateEventCounter で動作を開始するように設定します。

RAPI_TIMER_OFF	タイマが CreateEventCounter で動作を停止するように設定します。
RAPI_PULSE_ON	TA_{iIN} ピンからパルスを出力することを選択します。
RAPI_PULSE_OFF	TA_{iIN} ピンからパルスを出力しないことを選択します。
RAPI_AUTO_RELOAD	カウント タイプのリロード タイプを選択します。
RAPI_FREE_RUN	カウント タイプのフリーラン タイプを選択します。
RAPI_UP_COUNT	カウント動作のアップカウントを選択します。
RAPI_DOWN_COUNT	カウント動作のダウンカウントを選択します。
RAPI_UDF_REGISTER	アップ/ダウン切り替えを発生させる UDF レジスタを選択します。
RAPI_TAIOUT	アップ/ダウン切り替えを発生させる TA_{iOUT} ピンの入力信号を選択します。
RAPI_RISING	アクティブ エッジとして、カウント ソースの立ち上がりエッジを選択します。
RAPI_FALLING	アクティブ エッジとして、カウント ソースの立ち下がりエッジを選択します。
RAPI_BOTH	アクティブ エッジとして、カウント ソースの立ち上がりおよび立ち下がりエッジの両方を選択します。

• タイマ A を使用するときに指定できる定義値 (RAPI_TIMER_A0 ～ RAPI_TIMER_A4 が指定されているとき)

(カウント ソース) { RAPI_EV_EXTERNAL、RAPI_EV_TIMER_AJ、RAPI_EV_TIMER_AK、RAPI_EV_TIMER_B2 } のいずれかを指定します。デフォルト値は RAPI_EV_EXTERNAL です。

(動作状態設定) { RAPI_TIMER_ON、RAPI_TIMER_OFF } のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

(パルス出力機能) { RAPI_PULSE_ON、RAPI_PULSE_OFF } のいずれかを指定します。デフォルト値は RAPI_PULSE_OFF です。

(ゲート機能) { RAPI_GATE_L、RAPI_GATE_H } のいずれかを指定します。省略した場合には「ゲート機能なし」に設定されます。

(カウント タイプ) { RAPI_AUTO_RELOAD、RAPI_FREE_RUN } のいずれかを指定します。デフォルト値は RAPI_AUTO_RELOAD です。

(カウント方向) { RAPI_UP_COUNT、RAPI_DOWN_COUNT } のいずれかを指定します。デフォルト値は RAPI_DOWN_COUNT です。カウント方向は、UDF レジスタを使用するときにのみ設定できます。

(カウント方向切り替え) { RAPI_UDF_REGISTER、RAPI_TAIOUT } のいずれかを指定します。デフォルト値は RAPI_UDF_REGISTER です。

(カウント エッジ) { RAPI_RISING、RAPI_FALLING } のいずれかを指定します。デフォルト値は RAPI_FALLING です。

• タイマ B を使用するときに指定できる定義値 (RAPI_TIMER_B0 ～ RAPI_TIMER_B4 が指定されているとき)

(カウント ソース) { RAPI_EV_EXTERNAL、RAPI_EV_TIMER_BJ } のいずれかを指定します。デフォルト値は RAPI_EV_EXTERNAL です。

(動作状態設定) { RAPI_TIMER_ON、RAPI_TIMER_OFF } のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

(カウント エッジ)　{ RAPI_RISING、RAPI_FALLING, RAPI_BOTH } のいずれかを指定します。デフォルト値は RAPI_FALLING です。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

[data3]

タイマ レジスタに設定する値を 16 ビットで指定します。

[data4]

0 を指定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (イベント カウンタ モード)

参考

__EnableEventCounter、__DestroyEventCounter、__GetEventCounter

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
/* タイマ B0 をイベント カウンタ モードに設定します */
    __CreateEventCounter( RAPI_TIMER_B0|RAPI_TIMER_ON|RAPI_FALLING, 5,
                        0x80, 0, TimerIntFunc );
}
```

__EnableEventCounter

概要

<イベント カウンタ モード動作制御>

Boolean __EnableEventCounter(unsigned long data)

data	設定データ

説明

指定したタイマ モードに設定されているタイマの動作開始/動作停止停止を制御します。

[data]

data については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。
RAPI_TIMER_ON	イベント モードに設定されたタイマの動作を開始します。
RAPI_TIMER_OFF	イベント モードに設定されたタイマの動作を停止します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (イベント カウンタ モード)

参考

__CreateEventCounter、__DestroyEventCounter、__GetEventCounter

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* イベント カウンタ モードのタイマ B1 を無効にします */
    __EnableEventCounter( RAPI_TIMER_B1|RAPI_TIMER_OFF );
}
```

__DestroyEventCounter

概要

<イベント カウンタ モード設定破棄>

Boolean __DestroyEventCounter(unsigned long data)

data	設定データ

説明

指定したタイマ モードに設定されているタイマの設定を破棄します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (イベント カウンタ モード)

参考

__CreateEventCounter、__EnableEventCounter、__GetEventCounter

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* イベント カウンタ モードのタイマ B2 の設定を破棄します */
    __DestroyEventCounter( RAPI_TIMER_B2 );
}
```

__GetEventCounter

概要

<イベント カウンタ モードのカウンタ値取得>

Boolean __GetEventCounter(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	カウンタ値が格納されるバッファへのポインタ

説明

指定したイベント カウンタ モードに設定されているタイマのカウンタ値を取得します。

[data1]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (イベント カウンタ モード)

参考

__CreateEventCounter、__EnableEventCounter、__DestroyEventCounter

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    unsigned int data[2];

/* イベント カウンタ モードのタイマ B3 のカウンタを取得します */
    __GetEventCounter(RAPI_TIMER_B3, data );
}
```

__CreatePulseWidthModulationMode

概要

<パルス幅変調モード設定>

Boolean __CreatePulseWidthModulationMode(unsigned long data1, unsigned int data2, unsigned int* data3, void* data4)

data1	設定データ 1
data2	設定データ 2
data3	設定データ 3
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

タイマをパルス幅変調モードに設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を使用します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を使用します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を使用します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を使用します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を使用します。
RAPI_F1	カウント ソースに f_1 を選択します。
RAPI_F2	カウント ソースに f_2 を選択します。
RAPI_F8	カウント ソースに f_8 を選択します。
RAPI_F32	カウント ソースに f_{32} を選択します。
RAPI_FC32	カウント ソースに f_{C32} を選択します。
RAPI_TIMER_ON	タイマが __CreatePulseWidthModulationMode で動作を開始するように設定します。
RAPI_TIMER_OFF	タイマが __CreatePulseWidthModulationMode で動作を停止するように設定します。
RAPI_TG_TAIIN	カウント開始条件に TA_{iIN} ピンからの外部トリガ入力を選択します。
RAPI_EV_TIMER_AJ	タイマのカウント開始のトリガとして、タイマ Aj (j = i-1、ただし i = 0 の場合は j = 4) のオーバーフローまたはアンダーフローを選択します。
RAPI_EV_TIMER_AK	タイマのカウント開始のトリガとして、タイマ Ak (k = i+1、ただし i = 4 の場合は k = 0) のオーバーフローまたはアンダーフローを選択します。
RAPI_EV_TIMER_B2	タイマのカウント開始のトリガとして、タイマ B2 のオーバーフローまたはアンダーフローを選択します。
RAPI_TG_TAIS	TABSR レジスタの TA_iS ビットに 1 を書き込んだ場合にのみ、タイマのカウントが開始されます。
RAPI_PULSE_ON	TA_{iIN} ピンからパルスを出力することを選択します。タイマ Ai を使用するときにのみ選択できます。

RAPI_PULSE_OFF	TA_{iIN} ピンからパルスを出力しないことを選択します。タイマ Ai を使用するときにのみ選択できます。
RAPI_PWM_16	16 ビット パルス幅モジュレータとして動作することを選択します。
RAPI_PWM_8	8 ビット パルス幅モジュレータとして動作することを選択します。
RAPI_RISING	アクティブ エッジとして、TA_{iIN} ピン入力信号の立ち上がりエッジを選択します。
RAPI_FALLING	アクティブ エッジとして、TA_{iIN} ピン入力信号の立ち下がりエッジを選択します。

• タイマ A を使用するときに指定できる定義値 (RAPI_TIMER_A0 ～ RAPI_TIMER_A4 が指定されているとき)

(カウント ソース)　{ RAPI_F1、RAPI_F2、RAPI_F32、RAPI_FC32 } のいずれかを指定します。デフォルト値は RAPI_F2 です。

(動作状態設定)　{ RAPI_TIMER_ON、RAPI_TIMER_OFF } のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

(カウント開始条件)　{ RAPI_TG_TAIS、RAPI_TG_TAIIN、RAPI_EV_TIMER_AJ、RAPI_EV_TIMER_AK、RAPI_EV_TIMER_B2 } のいずれかを指定します。デフォルト値は RAPI_TG_TAIIN です。

(パルス出力機能)　{ RAPI_PULSE_ON、RAPI_PULSE_OFF } のいずれかを指定します。デフォルト値は RAPI_PULSE_OFF です。

(変調)　{ RAPI_PWM_16、RAPI_PWM_8 } のいずれかを指定します。デフォルト値は RAPI_PWM_16 です。

(TA_{iIN} ピン入力)　{ RAPI_RISING、RAPI_FALLING } のいずれかを指定します。デフォルト値は RAPI_FALLING です。TA_{iIN} ピン入力のアクティブ エッジは、RAPI_TG_TAIIN が選択されているときにのみ設定できます。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

[data3]

タイマ レジスタの設定値が格納される 16 ビット変数へのポインタを指定します。

16 ビット PWM の場合は、「高レベル幅 n/f_j、間隔 $65535/f_j$」の値 n を 16 ビットで指定します。

8 ビット PWM の場合は、「高レベル幅 $n (m + 1)/f$、間隔 $255 (m + 1)/f_j$」の n および m の値をそれぞれ高次 8 ビットおよび低次 8 ビットに指定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅変調モード (PWM モード))

参考

__EnablePulseWidthModulationMode、__DestroyPulseWidthModulationMode

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
   unsigned int p_tim[] =  {0xAA, 0xBB, 0xCC};

/* タイマ A3 をパルス幅変調モードに設定します */

__CreatePulseWidthModulationMode( RAPI_TIMER_A3|RAPI_TIMER_ON|RAPI_F8,
                                  5, p_tim, TimerIntFunc);
}
```

__EnablePulseWidthModulationMode

概要

<パルス幅変調モード動作制御>

Boolean __EnablePulseWidthModulationMode(unsigned long data)

data	設定データ

説明

指定したパルス幅変調モードに設定されているタイマの動作開始/動作停止を制御します。

[data]

data については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_ON	パルス幅変調モードに設定されたタイマの動作を開始します。
RAPI_TIMER_OFF	パルス幅変調モードに設定されたタイマの動作を停止します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅変調モード (PWM モード))

参考

__CreatePulseWidthModulationMode、__DestroyPulseWidthModulationMode

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* パルス幅変調モードのタイマ A3 を有効にします */
    __EnablePulseWidthModulationMode( RAPI_TIMER_A2|RAPI_TIMER_ON );
}
```

__DestroyPulseWidthModulationMode

概要

<パルス幅変調モード設定破棄>

Boolean __DestroyPulseWidthModulationMode(unsigned long data)

data	設定データ

説明

指定したパルス幅変調モードに設定されているタイマの設定を破棄します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅変調モード (PWM モード))

参考

__CreatePulseWidthModulationMode、__EnablePulseWidthModulationMode

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* パルス幅変調モードのタイマ A1 の設定を破棄します */
    __DestroyPulseWidthModulationMode( RAPI_TIMER_A1 );
}
```

__CreatePulsePeriodMeasurementMode

概要

<パルス周期測定モード設定>

Boolean __CreatePulsePeriodMeasurementMode(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	設定データ 1
data2	設定データ 2
data3	設定データ 3
data4	設定データ 4
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

タイマをパルス周期測定モードに設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。
RAPI_F1	カウント ソースに f_1 を選択します。
RAPI_F2	カウント ソースに f_2 を選択します。
RAPI_F8	カウント ソースに f_8 を選択します。
RAPI_F32	カウント ソースに f_{32} を選択します。
RAPI_FC32	カウント ソースに f_{C32} を選択します。
RAPI_TIMER_ON	タイマが __CreatePulsePeriodMeasurementMode で動作を開始するように設定します。
RAPI_TIMER_OFF	タイマが __CreatePulsePeriodMeasurementMode で動作を停止するように設定します。
RAPI_RISING_RISING	測定パルスの立ち上がりから次の立ち上がりの間隔を測定することを選択します。
RAPI_FALLING_FALLING	測定パルスの立ち下がりから次の立ち下がりの間隔を測定することを選択します。

• タイマ B を使用するときに指定できる定義値 (RAPI_TIMER_B0 ～ RAPI_TIMER_B5 が指定されているとき)

(カウントソース) {RAPI_F1、RAPI_F2、RAPI_F8、RAPI_F32、RAPI_FC32} のいずれかを指定します。デフォルト値は RAPI_F2 です。

(動作状態設定) {RAPI_TIMER_ON、RAPI_TIMER_OFF} のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

(測定パルス) {RAPI_RISING_RISING、RAPI_FALLING_FALLING} のいずれかを選択します。デフォルト値は RAPI_FALLING_FALLING です。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

[data3]

0 を指定します。

[data4]

0 を指定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス周期測定モード)

参考

__EnablePulsePeriodMeasurementMode、__DestroyPulsePeriodMeasurementMode, __GetPulsePeriodMeasurementMode

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
/* タイマ B0 をパルス周期測定モードに設定します */
    __CreatePulsePeriodMeasurementMode(
        RAPI_TIMER_B0|RAPI_TIMER_ON|RAPI_FALLING_FALLING|RAPI_F8,
        5, 0, 0, TimerIntFunc);
}
```

__EnablePulsePeriodMeasurementMode

概要

<パルス周期測定モード動作制御>

Boolean __EnablePulsePeriodMeasurementMode(unsigned long data)

data	設定データ

説明

指定したパルス周期測定モードに設定されているタイマの動作開始/動作停止を制御します。

[data]

data については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。
RAPI_TIMER_ON	パルス周期測定モードに設定されたタイマの動作を開始します。
RAPI_TIMER_OFF	パルス周期測定モードに設定されたタイマの動作を終了します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス周期測定モード)

参考

__CreatePulsePeriodMeasurementMode, __DestroyPulsePeriodMeasurementMode, __GetPulsePeriodMeasurementMode

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* パルス周期測定モードのタイマ B1 を有効にします */
    __EnablePulsePeriodMeasurementMode( RAPI_TIMER_B1|RAPI_TIMER_ON );
}
```

__DestroyPulsePeriodMeasurementMode

概要

<パルス周期測定モード設定破棄>

Boolean __DestroyPulsePeriodMeasurementMode(unsigned long data)

data	設定データ

説明

指定したパルス周期測定モードに設定されているタイマの設定を破棄します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス周期測定モード)

参考

__CreatePulsePeriodMeasurementMode、__EnablePulsePeriodMeasurementMode, __GetPulsePeriodMeasurementMode

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* パルス周期測定モードのタイマ B2 の設定を破棄します */
    __DestroyPulsePeriodMeasurementMode( RAPI_TIMER_B2 );
}
```

__GetPulsePeriodMeasurementMode

概要

<パルス周期測定モードの測定値取得>

Boolean __GetPulsePeriodMeasurementMode(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	カウンタ値が格納されるバッファへのポインタ

説明

指定したパルス周期測定モードに設定されているタイマのカウンタ値を取得します。

[data1]

data については、以下の定義値を設定できます。

RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス周期測定モード)

参考

__CreatePulsePeriodMeasurementMode、__EnablePulsePeriodMeasurementMode, __DestroyPulsePeriodMeasurementMode

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    unsigned int data[2];

/* パルス周期測定モードのタイマ B3 の測定値を取得します */
    __GetPulsePeriodMeasurementMode( RAPI_TIMER_B3, data );
}
```

__CreatePulseWidthMeasurementMode

概要

<パルス幅測定モード設定>

Boolean __CreatePulseWidthMeasurementMode(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	設定データ 1
data2	設定データ 2
data3	設定データ 3
data4	設定データ 4
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

タイマをパルス幅測定モードに設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。
RAPI_F1	カウント ソースに f_1 を選択します。
RAPI_F2	カウント ソースに f_2 を選択します。
RAPI_F8	カウント ソースに f_8 を選択します。
RAPI_F32	カウント ソースに f_{32} を選択します。
RAPI_FC32	カウント ソースに f_{C32} を選択します。
RAPI_TIMER_ON	タイマが __CreatePulseWidthMeasurementMode で動作を開始するように設定します。
RAPI_TIMER_OFF	タイマが __CreatePulseWidthMeasurementMode で動作を停止するように設定します。

• タイマ B を使用するときに指定できる定義値 (RAPI_TIMER_B0 ～ RAPI_TIMER_B2 が指定されているとき)

(カウント ソース)　{ RAPI_F1、RAPI_F2、RAPI_F8、RAPI_F32、RAPI_FC32 } のいずれかを指定します。デフォルト値は RAPI_F2 です。

(動作状態設定)　{ RAPI_TIMER_ON、RAPI_TIMER_OFF } のいずれかを指定します。デフォルト値は RAPI_TIMER_OFF です。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

[data3]

0 を指定します。

[data4]

0 を指定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅測定モード)

参考

__EnablePulseWidthMeasurementMode、__DestroyPulseWidthMeasurementMode, __GetPulseWidthMeasurementMode

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
/* タイマ B4 をパルス幅測定モードに設定します */
    __CreatePulseWidthMeasurementMode(
        RAPI_TIMER_B4|RAPI_TIMER_ON|RAPI_RISING_FALLING|RAPI_F8,
        5, 0, 0, TimerIntFunc);
}
```

__EnablePulseWidthMeasurementMode

概要

<パルス幅測定モード動作制御>

Boolean __EnablePulseWidthMeasurementMode(unsigned long data)

data	設定データ

説明

指定したパルス幅測定モードに設定されているタイマの動作を制御します。

[data]

data については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。
RAPI_TIMER_ON	パルス幅測定モードに設定されたタイマの動作を開始します。
RAPI_TIMER_OFF	パルス幅測定モードに設定されたタイマの動作を停止します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅測定モード)

参考

__CreatePulseWidthMeasurementMode, __DestroyPulseWidthMeasurementMode, __GetPulseWidthMeasurementMode

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* パルス幅測定モードのタイマ B5 を無効にします */
    __EnablePulseWidthMeasurementMode( RAPI_TIMER_B5|RAPI_TIMER_OFF );
}
```

__DestroyPulseWidthMeasurementMode

概要

<パルス幅測定モード設定破棄>

Boolean __DestroyPulseWidthMeasurementMode(unsigned long data)

data	設定データ

説明

指定したパルス幅測定モードに設定されているタイマの設定を破棄します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅測定モード)

参考

__CreatePulseWidthMeasurementMode、__EnablePulseWidthMeasurementMode, __GetPulseWidthMeasurementMode

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
/* パルス幅測定モードのタイマ B0 の設定を破棄します */
    __DestroyPulseWidthMeasurementMode( RAPI_TIMER_B0 );
}
```

__GetPulseWidthMeasurementMode

概要

<パルス幅測定モードの測定値取得>

Boolean __GetPulseWidthMeasurementMode(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	カウンタ値が格納されるバッファへのポインタ

説明

指定したパルス幅測定モードに設定されているタイマのカウンタ値を取得します。

[data1]

data については、以下の定義値を設定できます。

RAPI_TIMER_B0	タイマ B チャンネル 0 を使用します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を使用します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を使用します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を使用します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を使用します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を使用します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (パルス幅測定モード)

参考

__CreatePulseWidthMeasurementMode、__EnablePulseWidthMeasurementMode, __DestroyPulseWidthMeasurementMode

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    unsigned int data[2];

/* パルス幅測定モードのタイマ B1 の測定値を取得します */
    __GetPulseWidthMeasurementMode( RAPI_TIMER_B1, data );
}
```

__SetTimerRegister

概要

<タイマ レジスタ設定>

Boolean __SetTimerRegister(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	レジスタ値が格納されるバッファへのポインタ

説明

指定したタイマのレジスタを設定します。

[data1]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。

[data2]

レジスタ値が格納されるバッファへのポインタの内容は、以下のように指定する必要があります。値は、各レジスタにバッファ ポインタ要素の順に設定されます。

- **タイマ A (RAPI_TIMER_A0 to RAPI_TIMER_A4 を指定) を使用する場合**

[0]: タイマ Ai のモード レジスタの設定値を指定します (i = 0～4)。
[1]: タイマ Ai のレジスタの設定値を指定します (i = 0～4)。
[2]: アップ/ダウン フラグ レジスタの設定値を指定します。
[3]: ワンショット開始フラグ レジスタの設定値を指定します。
[4]: トリガ選択レジスタの設定値を指定します。
[5]: 時刻クロック プリスケーラー リセット レジスタの設定値を指定します。
[6]: カウント開始フラグ レジスタの設定値を指定します。

- **タイマ B (RAPI_TIMER_B0 to RAPI_TIMER_B5) を使用する場合**

[0]: タイマ Bi のモード レジスタの設定値を指定します (i = 0～5)。
[1]: タイマ Bi のレジスタの設定値を指定します (i = 0～5)。
[3]: 時刻クロック プリスケーラー リセット レジスタの設定値を指定します。
[4]: カウント開始フラグ レジスタの設定値を指定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (レジスタ操作)

参考

__EnableTimerRegister、__ClearTimerRegister, __GetTimerRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

- 指定可能なタイマは、使用する CPU によって異なります。

プログラム例

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
   unsigned char data[] =  {0,0,0,0,0,0,0};

/* タイマ A0 のレジスタを設定します */
   __SetTimerRegister( RAPI_TIMER_A0, data );
}
```

__EnableTimerRegister

概要

<タイマ レジスタ動作制御>

Boolean __EnableTimerRegister(unsigned long data)

data	設定データ

説明

指定したタイマの動作を、タイマを開始または停止して制御します。

[data]

data については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。
RAPI_TIMER_ON	選択したタイマの動作を開始するよう設定します。
RAPI_TIMER_OFF	選択したタイマの動作を停止するよう設定します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (レジスタ操作)

参考

__SetTimerRegister, __ClearTimerRegister, __GetTimerRegister

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
/* タイマ A1 を起動します */
    __EnableTimerRegister( RAPI_TIMER_A1|RAPI_TIMER_ON );
}
```

__ClearTimerRegister

概要

<タイマ レジスタ クリア>

Boolean __ClearTimerRegister(unsigned long data)

data	設定データ

説明

指定したタイマのタイマレジスタを、リセットした後に初期値に設定します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (レジスタ操作)

参考

__SetTimerRegister、__EnableTimerRegister, __GetTimerRegister

備考

- 引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
/* タイマ A2 の設定をクリアします */
    __ClearTimerRegister( RAPI_TIMER_A2 );
}
```

__GetTimerRegister

概要

<タイマ レジスタ値取得>

Boolean __GetTimerRegister(unsigned long data1, unsigned int *data2)

data1	設定データ
data2	レジスタ値が格納されるバッファへのポインタ

説明

指定したタイマのカウンタ値を取得します。

[data]

data については、以下の定義値を設定できます。

RAPI_TIMER_A0	タイマ A チャンネル 0 を選択します。
RAPI_TIMER_A1	タイマ A チャンネル 1 を選択します。
RAPI_TIMER_A2	タイマ A チャンネル 2 を選択します。
RAPI_TIMER_A3	タイマ A チャンネル 3 を選択します。
RAPI_TIMER_A4	タイマ A チャンネル 4 を選択します。
RAPI_TIMER_B0	タイマ B チャンネル 0 を選択します。
RAPI_TIMER_B1	タイマ B チャンネル 1 を選択します。
RAPI_TIMER_B2	タイマ B チャンネル 2 を選択します。
RAPI_TIMER_B3	タイマ B チャンネル 3 を選択します。
RAPI_TIMER_B4	タイマ B チャンネル 4 を選択します。
RAPI_TIMER_B5	タイマ B チャンネル 5 を選択します。

[data2]

取得したレジスタ値を格納する配列へのポインタを指定します。

配列の内容は以下の通りです。

- **タイマ A (RAPI_TIMER_A0 to RAPI_TIMER_A4 を指定) を使用する場合**

[0]: タイマ Ai のモード レジスタの値を格納します (i = 0～4)。

[1]: タイマ Ai のレジスタの値を格納します (i = 0～4)。

[2]: アップ/ダウン フラグ レジスタの値を格納します。

[3]: ワンショット開始フラグ レジスタの値を格納します。

[4]: トリガ選択レジスタの値を格納します。

[5]: 時刻クロック プリスケーラー リセット フラグ レジスタの値を格納します。

[6]: カウント開始フラグ レジスタの値を格納します。

- **タイマ B (RAPI_TIMER_B0 to RAPI_TIMER_B5) を使用する場合**

[0]: タイマ Bi のモード レジスタの値を格納します (i = 0～5)。

[1]: タイマ Bi のレジスタの値を格納します (i = 0～5)。

[2]: 時刻クロック プリスケーラー リセット フラグ レジスタの値を格納します。

[3]: カウント開始フラグ レジスタの値を格納します。

戻り値

タイマの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

タイマ (レジスタ操作)

参考　__SetTimerRegister、__EnableTimerRegister、__ClearTimerRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
   unsigned int data[7];

/* タイマ A3 のレジスタの値を取得します */
    __GetTimerRegister( RAPI_TIMER_A3, data );
}
```

4.2.3 I/O ポート
__SetIOPort

概要

<I/O ポートの設定>

Boolean __SetIOPort(unsigned long data1, unsigned int data2)

data1	設定データ 1
data2	設定データ 2

説明

指定した I/O ポートの動作条件を設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。ただし、複数のポートを同時に指定することはできませんのでご注意ください。

各 I/O ポートに対応する定義値を以下に示します。

RAPI_PORT_0_0	ポート $P0_0$	RAPI_PORT_0_1	ポート $P0_1$
RAPI_PORT_0_2	ポート $P0_2$	RAPI_PORT_0_3	ポート $P0_3$
RAPI_PORT_0_4	ポート $P0_4$	RAPI_PORT_0_5	ポート $P0_5$
RAPI_PORT_0_6	ポート $P0_6$	RAPI_PORT_0_7	ポート $P0_7$
RAPI_PORT_1_0	ポート $P1_0$	RAPI_PORT_1_1	ポート $P1_1$
RAPI_PORT_1_2	ポート $P1_2$	RAPI_PORT_1_3	ポート $P1_3$
RAPI_PORT_1_4	ポート $P1_4$	RAPI_PORT_1_5	ポート $P1_5$
RAPI_PORT_1_6	ポート $P1_6$	RAPI_PORT_1_7	ポート $P1_7$
RAPI_PORT_2_0	ポート $P2_0$	RAPI_PORT_2_1	ポート $P2_1$
RAPI_PORT_2_2	ポート $P2_2$	RAPI_PORT_2_3	ポート $P2_3$
RAPI_PORT_2_4	ポート $P2_4$	RAPI_PORT_2_5	ポート $P2_5$
RAPI_PORT_2_6	ポート $P2_6$	RAPI_PORT_2_7	ポート $P2_7$
RAPI_PORT_3_0	ポート $P3_0$	RAPI_PORT_3_1	ポート $P3_1$
RAPI_PORT_3_2	ポート $P3_2$	RAPI_PORT_3_3	ポート $P3_3$
RAPI_PORT_3_4	ポート $P3_4$	RAPI_PORT_3_5	ポート $P3_5$
RAPI_PORT_3_6	ポート $P3_6$	RAPI_PORT_3_7	ポート $P3_7$
RAPI_PORT_4_0	ポート $P4_0$	RAPI_PORT_4_1	ポート $P4_1$
RAPI_PORT_4_2	ポート $P4_2$	RAPI_PORT_4_3	ポート $P4_3$
RAPI_PORT_4_4	ポート $P4_4$	RAPI_PORT_4_5	ポート $P4_5$
RAPI_PORT_4_6	ポート $P4_6$	RAPI_PORT_4_7	ポート $P4_7$
RAPI_PORT_5_0	ポート $P5_0$	RAPI_PORT_5_1	ポート $P5_1$
RAPI_PORT_5_2	ポート $P5_2$	RAPI_PORT_5_3	ポート $P5_3$
RAPI_PORT_5_4	ポート $P5_4$	RAPI_PORT_5_5	ポート $P5_5$
RAPI_PORT_5_6	ポート $P5_6$	RAPI_PORT_5_7	ポート $P5_7$
RAPI_PORT_6_0	ポート $P6_0$	RAPI_PORT_6_1	ポート $P6_1$
RAPI_PORT_6_2	ポート $P6_2$	RAPI_PORT_6_3	ポート $P6_3$
RAPI_PORT_6_4	ポート $P6_4$	RAPI_PORT_6_5	ポート $P6_5$
RAPI_PORT_6_6	ポート $P6_6$	RAPI_PORT_6_7	ポート $P6_7$
RAPI_PORT_7_0	ポート $P7_0$	RAPI_PORT_7_1	ポート $P7_1$

RAPI_PORT_7_2	ポート $P7_2$	RAPI_PORT_7_3	ポート $P7_3$
RAPI_PORT_7_4	ポート $P7_4$	RAPI_PORT_7_5	ポート $P7_5$
RAPI_PORT_7_6	ポート $P7_6$	RAPI_PORT_7_7	ポート $P7_7$
RAPI_PORT_8_0	ポート $P8_0$	RAPI_PORT_8_1	ポート $P8_1$
RAPI_PORT_8_2	ポート $P8_2$	RAPI_PORT_8_3	ポート $P8_3$
RAPI_PORT_8_4	ポート $P8_4$	RAPI_PORT_8_5	ポート $P8_5$
RAPI_PORT_8_6	ポート $P8_6$	RAPI_PORT_8_7	ポート $P8_7$
RAPI_PORT_9_0	ポート $P9_0$	RAPI_PORT_9_1	ポート $P9_1$
RAPI_PORT_9_2	ポート $P9_2$	RAPI_PORT_9_3	ポート $P9_3$
RAPI_PORT_9_4	ポート $P9_4$	RAPI_PORT_9_5	ポート $P9_5$
RAPI_PORT_9_6	ポート $P9_6$	RAPI_PORT_9_7	ポート $P9_7$
RAPI_PORT_10_0	ポート $P10_0$	RAPI_PORT_10_1	ポート $P10_1$
RAPI_PORT_10_2	ポート $P10_2$	RAPI_PORT_10_3	ポート $P10_3$
RAPI_PORT_10_4	ポート $P10_4$	RAPI_PORT_10_5	ポート $P10_5$
RAPI_PORT_10_6	ポート $P10_6$	RAPI_PORT_10_7	ポート $P10_7$
RAPI_PORT_11_0	ポート $P11_0$	RAPI_PORT_11_1	ポート $P11_1$
RAPI_PORT_11_2	ポート $P11_2$	RAPI_PORT_11_3	ポート $P11_3$
RAPI_PORT_11_4	ポート $P11_4$	RAPI_PORT_11_5	ポート $P11_5$
RAPI_PORT_11_6	ポート $P11_6$	RAPI_PORT_11_7	ポート $P11_7$
RAPI_PORT_12_0	ポート $P12_0$	RAPI_PORT_12_1	ポート $P12_1$
RAPI_PORT_12_2	ポート $P12_2$	RAPI_PORT_12_3	ポート $P12_3$
RAPI_PORT_12_4	ポート $P12_4$	RAPI_PORT_12_5	ポート $P12_5$
RAPI_PORT_12_6	ポート $P12_6$	RAPI_PORT_12_7	ポート $P12_7$
RAPI_PORT_13_0	ポート $P13_0$	RAPI_PORT_13_1	ポート $P13_1$
RAPI_PORT_13_2	ポート $P13_2$	RAPI_PORT_13_3	ポート $P13_3$
RAPI_PORT_13_4	ポート $P13_4$	RAPI_PORT_13_5	ポート $P13_5$
RAPI_PORT_13_6	ポート $P13_6$	RAPI_PORT_13_7	ポート $P13_7$
RAPI_PORT_14_0	ポート $P14_0$	RAPI_PORT_14_1	ポート $P14_1$

ポート設定に関連する定義値について以下に説明します。

RAPI_PORT_INPUT	選択したポートを入力用に設定します。
RAPI_PORT_OUTPUT	選択したポートを出力用に設定します。
RAPI_PULLED_HIGH	選択したポートをハイにプルするように設定します。
RAPI_NOT_PULLED_HIGH	選択したポートをハイにプルしないように設定します。
RAPI_LATCH	選択したポートを、入力または出力のいずれに設定されているかにかかわらずポート ラッチを読み込むように設定します。ポート P1 を使用しているときにのみ指定できます。

[data2]

0 を指定します。

戻り値

I/O ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能　I/O ポート

参考　__ReadIOPort、__WriteIOPort、__SetIOPortRegister、__ReadIOPortRegister, __WriteIOPortRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
/* ポート P03 を入力ポートに設定します */
    __SetIOPort(RAPI_PORT_0_3| RAPI_PORT_INPUT| RAPI_PULLED_HIGH, 0 );
}
```

__ReadIOPort

概要

<I/O ポートからの読み出し>

Boolean __ReadIOPort(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	I/O ポートから読み込んだ値を格納する変数へのポインタ

説明

指定した I/O ポートの値を取得します。

[data1]

データを読み込む I/O ポートを指定します。各 I/O ポートに対応する定義値を以下に示します。

RAPI_PORT_0_0	ポート $P0_0$	RAPI_PORT_0_1	ポート $P0_1$
RAPI_PORT_0_2	ポート $P0_2$	RAPI_PORT_0_3	ポート $P0_3$
RAPI_PORT_0_4	ポート $P0_4$	RAPI_PORT_0_5	ポート $P0_5$
RAPI_PORT_0_6	ポート $P0_6$	RAPI_PORT_0_7	ポート $P0_7$
RAPI_PORT_1_0	ポート $P1_0$	RAPI_PORT_1_1	ポート $P1_1$
RAPI_PORT_1_2	ポート $P1_2$	RAPI_PORT_1_3	ポート $P1_3$
RAPI_PORT_1_4	ポート $P1_4$	RAPI_PORT_1_5	ポート $P1_5$
RAPI_PORT_1_6	ポート $P1_6$	RAPI_PORT_1_7	ポート $P1_7$
RAPI_PORT_2_0	ポート $P2_0$	RAPI_PORT_2_1	ポート $P2_1$
RAPI_PORT_2_2	ポート $P2_2$	RAPI_PORT_2_3	ポート $P2_3$
RAPI_PORT_2_4	ポート $P2_4$	RAPI_PORT_2_5	ポート $P2_5$
RAPI_PORT_2_6	ポート $P2_6$	RAPI_PORT_2_7	ポート $P2_7$
RAPI_PORT_3_0	ポート $P3_0$	RAPI_PORT_3_1	ポート $P3_1$
RAPI_PORT_3_2	ポート $P3_2$	RAPI_PORT_3_3	ポート $P3_3$
RAPI_PORT_3_4	ポート $P3_4$	RAPI_PORT_3_5	ポート $P3_5$
RAPI_PORT_3_6	ポート $P3_6$	RAPI_PORT_3_7	ポート $P3_7$
RAPI_PORT_6_0	ポート $P6_0$	RAPI_PORT_6_1	ポート $P6_1$
RAPI_PORT_6_2	ポート $P6_2$	RAPI_PORT_6_3	ポート $P6_3$
RAPI_PORT_6_4	ポート $P6_4$	RAPI_PORT_6_5	ポート $P6_5$
RAPI_PORT_6_6	ポート $P6_6$	RAPI_PORT_6_7	ポート $P6_7$
RAPI_PORT_7_0	ポート $P7_0$	RAPI_PORT_7_1	ポート $P7_1$
RAPI_PORT_7_2	ポート $P7_2$	RAPI_PORT_7_3	ポート $P7_3$
RAPI_PORT_7_4	ポート $P7_4$	RAPI_PORT_7_5	ポート $P7_5$
RAPI_PORT_7_6	ポート $P7_6$	RAPI_PORT_7_7	ポート $P7_7$
RAPI_PORT_8_0	ポート $P8_0$	RAPI_PORT_8_1	ポート $P8_1$
RAPI_PORT_8_2	ポート $P8_2$	RAPI_PORT_8_3	ポート $P8_3$
RAPI_PORT_8_4	ポート $P8_4$	RAPI_PORT_8_5	ポート $P8_5$
RAPI_PORT_8_6	ポート $P8_6$	RAPI_PORT_8_7	ポート $P8_7$
RAPI_PORT_9_0	ポート $P9_0$	RAPI_PORT_9_1	ポート $P9_1$
RAPI_PORT_9_2	ポート $P9_2$	RAPI_PORT_9_3	ポート $P9_3$
RAPI_PORT_9_4	ポート $P9_4$	RAPI_PORT_9_5	ポート $P9_5$

RAPI_PORT_9_6	ポート $P9_6$	RAPI_PORT_9_7	ポート $P9_7$
RAPI_PORT_10_0	ポート $P10_0$	RAPI_PORT_10_1	ポート $P10_1$
RAPI_PORT_10_2	ポート $P10_2$	RAPI_PORT_10_3	ポート $P10_3$
RAPI_PORT_10_4	ポート $P10_4$	RAPI_PORT_10_5	ポート $P10_5$
RAPI_PORT_10_6	ポート $P10_6$	RAPI_PORT_10_7	ポート $P10_7$
RAPI_PORT_11_0	ポート $P11_0$	RAPI_PORT_11_1	ポート $P11_1$
RAPI_PORT_11_2	ポート $P11_2$	RAPI_PORT_11_3	ポート $P11_3$
RAPI_PORT_11_4	ポート $P11_4$	RAPI_PORT_11_5	ポート $P11_5$
RAPI_PORT_11_6	ポート $P11_6$	RAPI_PORT_11_7	ポート $P11_7$
RAPI_PORT_12_0	ポート $P12_0$	RAPI_PORT_12_1	ポート $P12_1$
RAPI_PORT_12_2	ポート $P12_2$	RAPI_PORT_12_3	ポート $P12_3$
RAPI_PORT_12_4	ポート $P12_4$	RAPI_PORT_12_5	ポート $P12_5$
RAPI_PORT_12_6	ポート $P12_6$	RAPI_PORT_12_7	ポート $P12_7$
RAPI_PORT_13_0	ポート $P13_0$	RAPI_PORT_13_1	ポート $P13_1$
RAPI_PORT_13_2	ポート $P13_2$	RAPI_PORT_13_3	ポート $P13_3$
RAPI_PORT_13_4	ポート $P13_4$	RAPI_PORT_13_5	ポート $P13_5$
RAPI_PORT_13_6	ポート $P13_6$	RAPI_PORT_13_7	ポート $P13_7$
RAPI_PORT_14_0	ポート $P14_0$	RAPI_PORT_14_1	ポート $P14_1$

戻り値

I/O ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

I/O ポート

参考

__SetIOPort、__WriteIOPort、__SetIOPortRegister、__ReadIOPortRegister, __WriteIOPortRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
/* ポート P12 の値を取得します */
    __ReadIOPort(RAPI_PORT_1_2, &data );
}
```

__WriteIOPort

概要

<I/O ポートへの書き込み>

Boolean __WriteIOPort(unsigned long data1, unsigned int data2)

data1	設定データ 1
data2	I/O ポートに書き込むデータ

説明

指定した I/O ポートにデータを書き込みます。

[data1]

データを書き込む I/O ポートを指定します。各 I/O ポートに対応する定義値を以下に示します。

RAPI_PORT_0_0	ポート $P0_0$	RAPI_PORT_0_1	ポート $P0_1$
RAPI_PORT_0_2	ポート $P0_2$	RAPI_PORT_0_3	ポート $P0_3$
RAPI_PORT_0_4	ポート $P0_4$	RAPI_PORT_0_5	ポート $P0_5$
RAPI_PORT_0_6	ポート $P0_6$	RAPI_PORT_0_7	ポート $P0_7$
RAPI_PORT_1_0	ポート $P1_0$	RAPI_PORT_1_1	ポート $P1_1$
RAPI_PORT_1_2	ポート $P1_2$	RAPI_PORT_1_3	ポート $P1_3$
RAPI_PORT_1_4	ポート $P1_4$	RAPI_PORT_1_5	ポート $P1_5$
RAPI_PORT_1_6	ポート $P1_6$	RAPI_PORT_1_7	ポート $P1_7$
RAPI_PORT_2_0	ポート $P2_0$	RAPI_PORT_2_1	ポート $P2_1$
RAPI_PORT_2_2	ポート $P2_2$	RAPI_PORT_2_3	ポート $P2_3$
RAPI_PORT_2_4	ポート $P2_4$	RAPI_PORT_2_5	ポート $P2_5$
RAPI_PORT_2_6	ポート $P2_6$	RAPI_PORT_2_7	ポート $P2_7$
RAPI_PORT_3_0	ポート $P3_0$	RAPI_PORT_3_1	ポート $P3_1$
RAPI_PORT_3_2	ポート $P3_2$	RAPI_PORT_3_3	ポート $P3_3$
RAPI_PORT_3_4	ポート $P3_4$	RAPI_PORT_3_5	ポート $P3_5$
RAPI_PORT_3_6	ポート $P3_6$	RAPI_PORT_3_7	ポート $P3_7$
RAPI_PORT_6_0	ポート $P6_0$	RAPI_PORT_6_1	ポート $P6_1$
RAPI_PORT_6_2	ポート $P6_2$	RAPI_PORT_6_3	ポート $P6_3$
RAPI_PORT_6_4	ポート $P6_4$	RAPI_PORT_6_5	ポート $P6_5$
RAPI_PORT_6_6	ポート $P6_6$	RAPI_PORT_6_7	ポート $P6_7$
RAPI_PORT_7_0	ポート $P7_0$	RAPI_PORT_7_1	ポート $P7_1$
RAPI_PORT_7_2	ポート $P7_2$	RAPI_PORT_7_3	ポート $P7_3$
RAPI_PORT_7_4	ポート $P7_4$	RAPI_PORT_7_5	ポート $P7_5$
RAPI_PORT_7_6	ポート $P7_6$	RAPI_PORT_7_7	ポート $P7_7$
RAPI_PORT_8_0	ポート $P8_0$	RAPI_PORT_8_1	ポート $P8_1$
RAPI_PORT_8_2	ポート $P8_2$	RAPI_PORT_8_3	ポート $P8_3$
RAPI_PORT_8_4	ポート $P8_4$	RAPI_PORT_8_5	ポート $P8_5$
RAPI_PORT_8_6	ポート $P8_6$	RAPI_PORT_8_7	ポート $P8_7$
RAPI_PORT_9_0	ポート $P9_0$	RAPI_PORT_9_1	ポート $P9_1$
RAPI_PORT_9_2	ポート $P9_2$	RAPI_PORT_9_3	ポート $P9_3$
RAPI_PORT_9_4	ポート $P9_4$	RAPI_PORT_9_5	ポート $P9_5$

RAPI_PORT_9_6	ポート $P9_6$	RAPI_PORT_9_7	ポート $P9_7$
RAPI_PORT_10_0	ポート $P10_0$	RAPI_PORT_10_1	ポート $P10_1$
RAPI_PORT_10_2	ポート $P10_2$	RAPI_PORT_10_3	ポート $P10_3$
RAPI_PORT_10_4	ポート $P10_4$	RAPI_PORT_10_5	ポート $P10_5$
RAPI_PORT_10_6	ポート $P10_6$	RAPI_PORT_10_7	ポート $P10_7$
RAPI_PORT_11_0	ポート $P11_0$	RAPI_PORT_11_1	ポート $P11_1$
RAPI_PORT_11_2	ポート $P11_2$	RAPI_PORT_11_3	ポート $P11_3$
RAPI_PORT_11_4	ポート $P11_4$	RAPI_PORT_11_5	ポート $P11_5$
RAPI_PORT_11_6	ポート $P11_6$	RAPI_PORT_11_7	ポート $P11_7$
RAPI_PORT_12_0	ポート $P12_0$	RAPI_PORT_12_1	ポート $P12_1$
RAPI_PORT_12_2	ポート $P12_2$	RAPI_PORT_12_3	ポート $P12_3$
RAPI_PORT_12_4	ポート $P12_4$	RAPI_PORT_12_5	ポート $P12_5$
RAPI_PORT_12_6	ポート $P12_6$	RAPI_PORT_12_7	ポート $P12_7$
RAPI_PORT_13_0	ポート $P13_0$	RAPI_PORT_13_1	ポート $P13_1$
RAPI_PORT_13_2	ポート $P13_2$	RAPI_PORT_13_3	ポート $P13_3$
RAPI_PORT_13_4	ポート $P13_4$	RAPI_PORT_13_5	ポート $P13_5$
RAPI_PORT_13_6	ポート $P13_6$	RAPI_PORT_13_7	ポート $P13_7$
RAPI_PORT_14_0	ポート $P14_0$	RAPI_PORT_14_1	ポート $P14_1$

戻り値

I/O ポートの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

I/O ポート

参考

__SetIOPort、__ReadIOPort、__SetIOPortRegister、__ReadIOPortRegister, __WriteIOPortRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
    unsigned int data;

/* ポート P05 にデータを設定します */
    __WriteIOPort( RAPI_PORT_0_5, 0 );
}
```

__SetIOPortRegister

概要

<I/O ポート レジスタの設定>

Boolean __SetIOPortRegister(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4)

data1	設定データ 1
data2	設定データ 2
data3	設定データ 3
data4	設定データ 4

説明

[data1]

対応する各レジスタに指定した I/O ポートの動作条件を設定します。

各 I/O ポート レジスタに対応する定義値を以下に示します。

RAPI_PORT_0	ポート P0 レジスタ	RAPI_PORT_1	ポート P1 レジスタ
RAPI_PORT_2	ポート P2 レジスタ	RAPI_PORT_3	ポート P3 レジスタ
RAPI_PORT_4	ポート P4 レジスタ	RAPI_PORT_5	ポート P5 レジスタ
RAPI_PORT_6	ポート P6 レジスタ	RAPI_PORT_7	ポート P7 レジスタ
RAPI_PORT_8	ポート P8 レジスタ	RAPI_PORT_9	ポート P9 レジスタ
RAPI_PORT_10	ポート P10 レジスタ	RAPI_PORT_11	ポート P11 レジスタ
RAPI_PORT_12	ポート P12 レジスタ	RAPI_PORT_13	ポート P13 レジスタ
RAPI_PORT_14	ポート P14 レジスタ		

ポート設定に関連する定義値について以下に説明します。

RAPI_LATCH	入力または出力のいずれに設定されているかにかかわらず、ポートがポートラッチを読み込むように設定します。ポート P1 を使用しているときにのみ指定できます。

[data2]

選択したポートに対応するポート方向レジスタの設定値を指定します。

[data3]

選択したポートに対応するプルアップ制御レジスタの設定値を指定します。

[data4]

0 を指定します。

戻り値

I/O ポート レジスタの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

I/O ポート

参考

__SetIOPort, __ReadIOPort、__WriteIOPort、__ReadIOPortRegister、__WriteIOPortRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
/* ポート P1 レジスタの入出力を設定します */
    __SetIOPortRegister(RAPI_PORT_1, 0xAA, 0, 0 );
}
```

__ReadIOPortRegister

概要

<I/O ポート レジスタからの読み出し>

Boolean __ReadIOPortRegister(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	I/O ポート レジスタから読み込んだ値を格納する変数へのポインタ

説明

対応する各レジスタから、指定した I/O ポートの値を取得します。

[data1]

データを読み込む I/O ポート レジスタを指定します。各 I/O ポート レジスタに対応する定義値を以下に示します。

RAPI_PORT_0	ポート P0 レジスタ	RAPI_PORT_1	ポート P1 レジスタ
RAPI_PORT_2	ポート P2 レジスタ	RAPI_PORT_3	ポート P3 レジスタ
RAPI_PORT_4	ポート P4 レジスタ	RAPI_PORT_5	ポート P5 レジスタ
RAPI_PORT_6	ポート P6 レジスタ	RAPI_PORT_7	ポート P7 レジスタ
RAPI_PORT_8	ポート P8 レジスタ	RAPI_PORT_9	ポート P9 レジスタ
RAPI_PORT_10	ポート P10 レジスタ	RAPI_PORT_11	ポート P11 レジスタ
RAPI_PORT_12	ポート P12 レジスタ	RAPI_PORT_13	ポート P13 レジスタ
RAPI_PORT_14	ポート P14 レジスタ		

戻り値

I/O ポート レジスタの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

I/O ポート

参考

__SetIOPort、__ReadIOPort、__WriteIOPort、__SetIOPortRegister、__WriteIOPortRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
    unsigned int data;

/* ポート P1 レジスタの値を取得します */
    __ReadIOPortRegister( RAPI_PORT_1, &data );
}
```

__WriteIOPortRegister

概要

<I/O ポート レジスタへの書き込み>

Boolean __WriteIOPortRegister(unsigned long data1, unsigned int data2)

data1	設定データ 1
data2	I/O ポート レジスタに書き込むデータ

説明

対応する各レジスタに、指定した I/O ポートの値を書き込みます。

[data1]

データを書き込む I/O ポート レジスタを指定します。各 I/O ポート レジスタに対応する定義値を以下に示します。

RAPI_PORT_0	ポート P0 レジスタ	RAPI_PORT_1	ポート P1 レジスタ
RAPI_PORT_2	ポート P2 レジスタ	RAPI_PORT_3	ポート P3 レジスタ
RAPI_PORT_4	ポート P4 レジスタ	RAPI_PORT_5	ポート P5 レジスタ
RAPI_PORT_6	ポート P6 レジスタ	RAPI_PORT_7	ポート P7 レジスタ
RAPI_PORT_8	ポート P8 レジスタ	RAPI_PORT_9	ポート P9 レジスタ
RAPI_PORT_10	ポート P10 レジスタ	RAPI_PORT_11	ポート P11 レジスタ
RAPI_PORT_12	ポート P12 レジスタ	RAPI_PORT_13	ポート P13 レジスタ
RAPI_PORT_14	ポート P14 レジスタ		

戻り値

I/O ポート レジスタの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

I/O ポート

参考

__SetIOPort、__ReadIOPort、__WriteIOPort、__SetIOPortRegister、__ReadIOPortRegister

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
/* ポート P2 レジスタにデータを設定します */
    __WriteIOPortRegister( RAPI_PORT_2, 0xFF );
}
```

4.2.4 外部割り込み
__SetInterrupt

概要

<外部割り込みの設定>

Boolean __SetInterrupt(unsigned long data1, unsigned int data2, void* func)

data1	設定データ 1
data2	設定データ 2
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

指定した外部割り込みの設定を行います。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。ただし、複数の外部割り込みを同時に指定することはできませんのでご注意ください。

RAPI_INT0	_INT0 割り込みを使用します。
RAPI_INT1	_INT1 割り込みを使用します。
RAPI_INT2	_INT2 割り込みを使用します。
RAPI_INT3	_INT3 割り込みを使用します。
RAPI_INT4	_INT4 割り込みを使用します。
RAPI_INT5	_INT5 割り込みを使用します。
RAPI_KEY	キー入力割り込みを使用します。
RAPI_INT_RISING	選択した外部割り込みのアクティブ エッジとして立ち上がりエッジを指定します。
RAPI_INT_FALLING	選択した外部割り込みのアクティブ エッジとして立ち下がりエッジを指定します。
RAPI_INT_BOTH	選択した外部割り込みのアクティブ エッジとして両方のエッジを指定します。
RAPI_KI0_ENABLE	_KI0 ピン入力を使用します。
RAPI_KI1_ENABLE	_KI1 ピン入力を使用します。
RAPI_KI2_ENABLE	_KI2 ピン入力を使用します。
RAPI_KI3_ENABLE	_KI3 ピン入力を使用します。

• _INT0–5 割り込みを使用しているとき (RAPI_INT0 to RAPI_INT5 を指定) に指定できる定義値

(極性)　{ RAPI_INT_RISING、RAPI_INT_FALLING、RAPI_INT_BOTH } のいずれかを指定します。デフォルト値は RAPI_INT_FALLING です。

• キー入力割り込みを使用している (RAPI_KEY を指定) ときに指定できる定義値

(入力ピン)　_KI0、_KI1、_KI2、または _KI3 ピン入力を使用するには、それぞれ RAPI_KI0_ENABLE、RAPI_KI1_ENABLE、RAPI_KI2_ENABLE、または RAPI_KI3_ENABLE を指定します。デフォルト値は RAPI_INT_FALLING です。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

戻り値

外部割り込みの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

外部割り込み

参考

__EnableInterrupt, __GetInterruptFlag, __ClearInterruptFlag

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_interrupt_m16c_62p.h"

void IntFunc( void ){}

void func( void )
{
/* _INT0 割り込みを設定します */
    __SetInterrupt( RAPI_INT0|RAPI_INT_FALLING, 0, IntFunc );
}
```

__EnableInterrupt

概要

<外部割り込みの制御>

Boolean __EnableInterrupt(unsigned long data1, unsigned int data2)

data1	設定データ 1
data2	設定データ 2

説明

指定した外部割り込みの動作条件を変更します。

[data1]

RAPI_INT0	_INT0 割り込みを使用します。
RAPI_INT1	_INT1 割り込みを使用します。
RAPI_INT2	_INT2 割り込みを使用します。
RAPI_INT3	_INT3 割り込みを使用します。
RAPI_INT4	_INT4 割り込みを使用します。
RAPI_INT5	_INT5 割り込みを使用します。
RAPI_KEY	キー入力割り込みを使用します。

[data2]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

戻り値

外部割り込みの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

外部割り込み

参考

__SetInterrupt、__GetInterruptFlag、__ClearInterruptFlag

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_interrupt_m16c_62p.h"

void func( void )
{
/* _INT1 割り込み ( 割り込み優先順位レベル 5 ) を有効にします */
    __EnableInterrupt( RAPI_INT1, 5 );
}
```

__GetInterruptFlag

概要

<外部割り込みフラグの状態取得>

Boolean __GetInterruptFlag(unsigned long data1, unsigned int *data2))

data1	設定データ 1
data2	取得したフラグ データが格納されるバッファへのポインタ

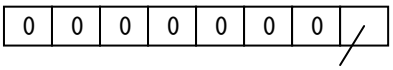
説明

指定した外部割り込みの割り込み要求フラグの値を取得します。

[data1]

RAPI_INT0	_INT0 割り込みを使用します。
RAPI_INT1	_INT1 割り込みを使用します。
RAPI_INT2	_INT2 割り込みを使用します。
RAPI_INT3	_INT3 割り込みを使用します。
RAPI_INT4	_INT4 割り込みを使用します。
RAPI_INT5	_INT5 割り込みを使用します。
RAPI_KEY	キー入力割り込みを使用します。

[data2]



割り込み要求フラグの値 (0: 割り込みが要求されていない; 1: 割り込みが要求された)

戻り値

外部割り込みの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

外部割り込み

参考

__SetInterrupt、__EnableInterrupt、__ClearInterruptFlag

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_interrupt_m16c_62p.h"

void func( void )
{
    unsigned char data;

/* _INT2 割り込みのフラグを取得します */
    __GetInterruptFlag( RAPI_INT2, &data );
}
```

__ClearInterruptFlag

概要

<外部割り込みフラグのクリア>

Boolean __ClearInterruptFlag(unsigned long data)

data	設定データ

説明

指定した外部割り込みの割り込み要求フラグをクリアします。

[data]

RAPI_INT0	_INT0 割り込みを使用します。
RAPI_INT1	_INT1 割り込みを使用します。
RAPI_INT2	_INT2 割り込みを使用します。
RAPI_INT3	_INT3 割り込みを使用します。
RAPI_INT4	_INT4 割り込みを使用します。
RAPI_INT5	_INT5 割り込みを使用します。
RAPI_KEY	キー入力割り込みを使用します。

戻り値

外部割り込みの指定が正しくない場合には RAPI_FALSE を返し、それ以外は RAPI_TRUE を返します。

機能

外部割り込み

参考

__SetInterrupt、__EnableInterrupt、__GetInterruptFlag

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_interrupt_m16c_62p.h"

void func( void )
{
/* _INT0 割り込みの状態をクリアします */
    __ClearInterruptFlag( RAPI_INT0 );
}
```

4.2.5 A/D 変換器

__CreateADC

概要

<A/D 変換器の設定>

Boolean __CreateADC(unsigned long data1, unsigned int data2, unsigned int data3, void* func)

data1	設定データ 1
data2	A/D 変換器が使用するアナログ入力ピンの数
data3	設定データ 3
func	コールバック関数のポインタ (コールバック関数を設定しない場合は 0 を指定します)

説明

A/D 変換器を指定したモードおよび動作条件に設定します。

[data1]

data 1 については、以下の定義値を設定できます。複数の定義値を同時に設定する場合には、記号 “|” を使用して各指定値を区切ります。ただし、同時に複数のアナログ入力ピン記号を指定することはできませんのでご注意ください。

RAPI_ONE_SHOT	ワンショット モードを選択します。
RAPI_REPEAT	リピート モードを選択します。
RAPI_SINGLE_SWEEP	シングル スイープ モードを選択します。
RAPI_REPEAT_SWEEP0	リピート スイープ モード 0 を選択します。
RAPI_REPEAT_SWEEP1	リピート スイープ モード 1 を選択します。
RAPI_AN0	アナログ入力ピンに AN_0 ピンを使用します。
RAPI_AN1	アナログ入力ピンに AN_1 ピンを使用します。
RAPI_AN2	アナログ入力ピンに AN_2 ピンを使用します。
RAPI_AN3	アナログ入力ピンに AN_3 ピンを使用します。
RAPI_AN4	アナログ入力ピンに AN_4 ピンを使用します。
RAPI_AN5	アナログ入力ピンに AN_5 ピンを使用します。
RAPI_AN6	アナログ入力ピンに AN_6 ピンを使用します。
RAPI_AN7	アナログ入力ピンに AN_7 ピンを使用します。
RAPI_AN00	アナログ入力ピンに AN_{00} ピンを使用します。
RAPI_AN01	アナログ入力ピンに AN_{01} ピンを使用します。
RAPI_AN02	アナログ入力ピンに AN_{02} ピンを使用します。
RAPI_AN03	アナログ入力ピンに AN_{03} ピンを使用します。
RAPI_AN04	アナログ入力ピンに AN_{04} ピンを使用します。
RAPI_AN05	アナログ入力ピンに AN_{05} ピンを使用します。
RAPI_AN06	アナログ入力ピンに AN_{06} ピンを使用します。
RAPI_AN07	アナログ入力ピンに AN_{07} ピンを使用します。
RAPI_AN20	アナログ入力ピンに AN_{20} ピンを使用します。
RAPI_AN21	アナログ入力ピンに AN_{21} ピンを使用します。
RAPI_AN22	アナログ入力ピンに AN_{22} ピンを使用します。
RAPI_AN23	アナログ入力ピンに AN_{23} ピンを使用します。
RAPI_AN24	アナログ入力ピンに AN_{24} ピンを使用します。
RAPI_AN25	アナログ入力ピンに AN_{25} ピンを使用します。
RAPI_AN26	アナログ入力ピンに AN_{26} ピンを使用します。

RAPI_AN27	アナログ入力ピンに AN_{27} ピンを使用します。
RAPI_P0_GROUP	アナログ入力ピンに port P_0 グループを使用します。
RAPI_P10_GROUP	アナログ入力ピンに port P_{10} グループを使用します。
RAPI_P2_GROUP	アナログ入力ピンに port P_2 グループを使用します。
RAPI_FAD	A/D 変換器の動作周波数を f_{AD} に設定します。
RAPI_FAD2	A/D 変換器の動作周波数を f_{AD} / 2 に設定します。
RAPI_FAD3	A/D 変換器の動作周波数を f_{AD} / 3 に設定します。
RAPI_FAD4	A/D 変換器の動作周波数を f_{AD} / 4 に設定します。
RAPI_FAD6	A/D 変換器の動作周波数を f_{AD} / 6 に設定します。
RAPI_FAD12	A/D 変換器の動作周波数を f_{AD} / 12 に設定します。
RAPI_10BIT	A/D 変換器の解像度を 10 ビットに設定します。
RAPI_8BIT	A/D 変換器の解像度を 8 ビットに設定します。
RAPI_AD_ON	A/D 変換器が動作を開始するよう設定します。
RAPI_AD_OFF	A/D 変換器が動作を停止するよう設定します。
RAPI_WITH_SAMPLE_HOLD	サンプル-アンド-ホールド動作を適用するように指定します。
RAPI_WITHOUT_SAMPLE_HOLD	サンプル-アンド-ホールド動作を適用しないように指定します。
RAPI_SOFTWARE_TRIGGER	ソフトウェア トリガを選択します。
RAPI_EXTERNAL_TRIGGER	外部トリガを選択します。
RAPI_ANEX_UNUSED	ANEX0 および ANEX1 を使用しません。
RAPI_ANEX0	ANEX0 入力は A/D 変換されます。
RAPI_ANEX1	ANEX1 入力は A/D 変換されます。
RAPI_EXTERNAL_OP_AMP	外部 OP アンプ接続モード

• ワンショット モードを使用している (RAPI_ONE_SHOT を指定) ときに指定できる定義値

(入力ピン) { RAPI_AN0、RAPI_AN1、RAPI_AN2、RAPI_AN3、RAPI_AN4、RAPI_AN5、RAPI_AN6、RAPI_AN7、RAPI_AN00、RAPI_AN01、RAPI_AN02、RAPI_AN03、RAPI_AN04、RAPI_AN05、RAPI_AN06、RAPI_AN07、RAPI_AN20、RAPI_AN21、RAPI_AN22、RAPI_AN23、RAPI_AN24、RAPI_AN25、RAPI_AN26、RAPI_AN27、RAPI_ANEX0、RAPI_ANEX1 } のいずれかを選択します。デフォルト値は RAPI_AN0 です。

(動作周波数) { RAPI_FAD、RAPI_FAD2、RAPI_FAD3、RAPI_FAD4、RAPI_FAD6、RAPI_FAD12 } のいずれかを指定します。デフォルト値は RAPI_FAD4 です。

(解像度) { RAPI_10BIT、RAPI_8BIT } のいずれかを指定します。デフォルト値は RAPI_8BIT です。

(動作状態設定) { RAPI_AD_ON、RAPI_AD_OFF } のいずれかを指定します。デフォルト値は RAPI_AD_OFF です。

(変換方法) { RAPI_WITH_SAMPLE_HOLD、RAPI_WITHOUT_SAMPLE_HOLD } のいずれかを指定します。デフォルト値は RAPI_WITHOUT_SAMPLE_HOLD です。

(トリガ) { RAPI_SOFTWARE_TRIGGER、RAPI_EXTERNAL_TRIGGER } のいずれかを指定します。
デフォルト値は RAPI_SOFTWARE_TRIGGER です。

(外部 OP アンプ接続) { RAPI_ANEX_UNUSED、RAPI_EXTERNAL_OP_AMP } のいずれかを指定します。
デフォルト値は RAPI_ANEX_UNUSED です。

• リピート モードを使用している (RAPI_REPEAT を指定) ときに指定できる定義値

(入力ピン) { RAPI_AN0、RAPI_AN1、RAPI_AN2、RAPI_AN3、RAPI_AN4、RAPI_AN5、RAPI_AN6、RAPI_AN7、RAPI_AN00、RAPI_AN01、RAPI_AN02、RAPI_AN03、RAPI_AN04、RAPI_AN05、RAPI_AN06、RAPI_AN07、RAPI_AN20、RAPI_AN21、RAPI_AN22、RAPI_AN23、RAPI_AN24、RAPI_AN25、RAPI_AN26、RAPI_AN27、RAPI_ANEX0、RAPI_ANEX1 } のいずれかを選択します。デフォルト値は RAPI_AN0 です。

(動作周波数) { RAPI_FAD、RAPI_FAD2、RAPI_FAD3、RAPI_FAD4、RAPI_FAD6、RAPI_FAD12 } のいずれかを指定します。デフォルト値は RAPI_FAD4 です。

(解像度) { RAPI_10BIT、RAPI_8BIT } のいずれかを指定します。デフォルト値は RAPI_8BIT です。

(動作状態設定) { RAPI_AD_ON、RAPI_AD_OFF } のいずれかを指定します。デフォルト値は RAPI_AD_OFF です。

(変換方法) { RAPI_WITH_SAMPLE_HOLD、RAPI_WITHOUT_SAMPLE_HOLD } のいずれかを指定します。
デフォルト値は RAPI_WITHOUT_SAMPLE_HOLD です。

(トリガ) { RAPI_SOFTWARE_TRIGGER、RAPI_EXTERNAL_TRIGGER } のいずれかを指定します。
デフォルト値は RAPI_SOFTWARE_TRIGGER です。

(外部 OP アンプ接続) { RAPI_ANEX_UNUSED、RAPI_EXTERNAL_OP_AMP } のいずれかを指定します。
デフォルト値は RAPI_ANEX_UNUSED です。

• シングル スイープ モードを使用している (RAPI_SINGLE_SWEEP を指定) ときに指定できる定義値

(入力ピン) { RAPI_P0_GROUP、RAPI_P10_GROUP、RAPI_P2_GROUP } のいずれかを指定します。デフォルト値は RAPI_P10_GROUP です。

(動作周波数) { RAPI_FAD、RAPI_FAD2、RAPI_FAD3、RAPI_FAD4、RAPI_FAD6、RAPI_FAD12 } のいずれかを指定します。デフォルト値は RAPI_FAD4 です。

(解像度) { RAPI_10BIT、RAPI_8BIT } のいずれかを指定します。デフォルト値は RAPI_8BIT です。

(動作状態設定) { RAPI_AD_ON、RAPI_AD_OFF } のいずれかを指定します。デフォルト値は RAPI_AD_OFF です。

(変換方法) { RAPI_WITH_SAMPLE_HOLD、RAPI_WITHOUT_SAMPLE_HOLD } のいずれかを指定します。
デフォルト値は RAPI_WITHOUT_SAMPLE_HOLD です。

(トリガ) { RAPI_SOFTWARE_TRIGGER、RAPI_EXTERNAL_TRIGGER } のいずれかを指定します。
デフォルト値は RAPI_SOFTWARE_TRIGGER です。

(外部 OP アンプ接続) { RAPI_ANEX_UNUSED、RAPI_EXTERNAL_OP_AMP } のいずれかを指定します。
デフォルト値は RAPI_ANEX_UNUSED です。

• リピート スイープ モード 0 を使用している (RAPI_REPEAT_SWEEP0 を指定) ときに指定できる定義値

(入力ピン)　{ RAPI_P0_GROUP、RAPI_P10_GROUP、RAPI_P2_GROUP } のいずれかを指定します。デフォルト値は RAPI_P10_GROUP です。

(動作周波数)　{ RAPI_FAD、RAPI_FAD2、RAPI_FAD3、RAPI_FAD4、RAPI_FAD6、RAPI_FAD12 } のいずれかを指定します。デフォルト値は RAPI_FAD4 です。

(解像度)　{ RAPI_10BIT、RAPI_8BIT } のいずれかを指定します。デフォルト値は RAPI_8BIT です。

(動作状態設定)　{ RAPI_AD_ON、RAPI_AD_OFF } のいずれかを指定します。デフォルト値は RAPI_AD_OFF です。

(変換方法)　{ RAPI_WITH_SAMPLE_HOLD、RAPI_WITHOUT_SAMPLE_HOLD } のいずれかを指定します。デフォルト値は RAPI_WITHOUT_SAMPLE_HOLD です。

(トリガ)　{ RAPI_SOFTWARE_TRIGGER、RAPI_EXTERNAL_TRIGGER } のいずれかを指定します。デフォルト値は RAPI_SOFTWARE_TRIGGER です。

(外部 OP アンプ接続)　{ RAPI_ANEX_UNUSED、RAPI_EXTERNAL_OP_AMP } のいずれかを指定します。
デフォルト値は RAPI_ANEX_UNUSED です。

• リピート スイープ モード 1 を使用している (RAPI_REPEAT_SWEEP1 を指定) ときに指定できる定義値

(入力ピン)　{ RAPI_P0_GROUP、RAPI_P10_GROUP、RAPI_P2_GROUP } のいずれかを指定します。デフォルト値は RAPI_P10_GROUP です。

(動作周波数)　{ RAPI_FAD、RAPI_FAD2、RAPI_FAD3、RAPI_FAD4、RAPI_FAD6、RAPI_FAD12 } のいずれかを指定します。デフォルト値は RAPI_FAD4 です。

(解像度)　{ RAPI_10BIT、RAPI_8BIT } のいずれかを指定します。デフォルト値は RAPI_8BIT です。

(動作状態設定)　{ RAPI_AD_ON、RAPI_AD_OFF } のいずれかを指定します。デフォルト値は RAPI_AD_OFF です。

(変換方法)　{ RAPI_WITH_SAMPLE_HOLD、RAPI_WITHOUT_SAMPLE_HOLD } のいずれかを指定します。
デフォルト値は RAPI_WITHOUT_SAMPLE_HOLD です。

(トリガ)　{ RAPI_SOFTWARE_TRIGGER、RAPI_EXTERNAL_TRIGGER } のいずれかを指定します。
デフォルト値は RAPI_SOFTWARE_TRIGGER です。

(外部 OP アンプ接続)　{ RAPI_ANEX_UNUSED、RAPI_EXTERNAL_OP_AMP } のいずれかを指定します。
デフォルト値は RAPI_ANEX_UNUSED です。

[data2]

設定する値は使用する A/D 変換モードによって異なります。

ワンショット モード リピート モード	1 を指定します。
シングル スイープ モード リピート スイープ モード 0	2、4、6、または 8 を指定します。
リピート スイープ モード 1	1、2、3、または 4 を指定します。

[data3]

割り込み制御レジスタに設定する割り込み優先順位レベル (0～7) を指定します。

戻り値

A/D 変換器が正常に設定されると RAPI_TRUE を返し、失敗した場合は RAPI_FALSE を返します。

機能

A/D 変換器

参考

__EnableADC、__DestroyADC、__GetADC, __GetADCAll

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include " rapi_ad_m16c_62p.h"

void AdIntFunc( void ){}

void func( void )
{
/* A/D 変換器をワンショット モードに設定します */
    __CreateADC( RAPI_ONE_SHOT|RAPI_AN2|RAPI_FAD2| RAPI_WITH_SAMPLE_HOLD
| RAPI_EXTERNAL_TRIGGER |RAPI_AD_ON|RAPI_10BIT| RAPI_ANEX_UNUSED, 1, 5,
AdIntFunc );
}
```

__EnableADC

概要

<A/D 変換器の動作制御>

Boolean __EnableADC (unsigned long data1, unsigned int data2)

data1	設定データ 1
data2	A/D 変換器が使用するアナログ入力ピンの数

説明

指定した A/D 変換器の動作を、タイマを開始または停止して制御します。

[data1]

RAPI_AN0	アナログ入力ピントして AN_0 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN1	アナログ入力ピントして AN_1 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN2	アナログ入力ピントして AN_2 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN3	アナログ入力ピントして AN_3 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN4	アナログ入力ピントして AN_4 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN5	アナログ入力ピントして AN_5 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN6	アナログ入力ピントして AN_6 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN7	アナログ入力ピントして AN_7 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN00	アナログ入力ピントして AN_{00} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN01	アナログ入力ピントして AN_{01} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN02	アナログ入力ピントして AN_{02} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN03	アナログ入力ピントして AN_{03} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN04	アナログ入力ピントして AN_{04} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN05	アナログ入力ピントして AN_{05} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN06	アナログ入力ピントして AN_{06} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN07	アナログ入力ピントして AN_{07} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN20	アナログ入力ピントして AN_{20} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。

RAPI_AN21	アナログ入力ピントして AN_{21} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN22	アナログ入力ピントして AN_{22} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN23	アナログ入力ピントして AN_{23} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN24	アナログ入力ピントして AN_{24} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN25	アナログ入力ピントして AN_{25} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN26	アナログ入力ピントして AN_{26} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_AN27	アナログ入力ピントして AN_{27} ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_ANEX0	アナログ入力ピントして ANEX0 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_ANEX1	アナログ入力ピントして ANEX1 ピンを使用します。 ワンショット モードまたはリピート モード使用時に選択できます。
RAPI_P0_GROUP	アナログ入力ピントして P_0 グループを使用します。スイープ モード、リピート スイープ モード 0、またはリピート スイープ モード 1 の使用時に選択できます。
RAPI_P10_GROUP	アナログ入力ピントして P_{10} グループを使用します。スイープ モード、リピート スイープ モード 0、またはリピート スイープ モード 1 の使用時に選択できます。
RAPI_P2_GROUP	アナログ入力ピントして P_2 グループを使用します。スイープ モード、リピート スイープ モード 0、またはリピート スイープ モード 1 の使用時に選択できます。
RAPI_AD_ON	A/D 変換器が動作を開始するよう設定します。
RAPI_AD_OFF	A/D 変換器が動作を停止するよう設定します。

[data2]

設定する値は使用する A/D 変換モードによって異なります。

ワンショット モード リピート モード	1 を指定します。
シングル スイープ モード リピート スイープ モード 0	2、4、6、または 8 を指定します。
リピート スイープ モード 1	1、2、3、または 4 を指定します。

戻り値

A/D 変換器が正常に制御されると RAPI_TRUE を返し、失敗した場合は RAPI_FALSE を返します。

機能

A/D 変換器

参考

CreateADC、DestroyADC、GetADC、GetADCAll

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
/* A/D 変換器を無効にします */
    __EnableADC( RAPI_AN0|RAPI_AD_OFF, 1 );
}
```

__DestroyADC

概要

<A/D 変換器の設定破棄>

Boolean __DestroyADC(void)

説明

指定した A/D 変換器の設定を破棄します。

戻り値

A/D 変換器の設定が正常に破棄されると RAPI_TRUE を返し、失敗した場合は RAPI_FALSE を返します。

機能

A/D 変換器

参考

__CreateADC、__EnableADC、__GetADC、__GetADCAll

プログラム例

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
/* A/D 変換器を破棄にします */
    __DestroyADC();
}
```

__GetADC

概要

<A/D 変換値の取得 (レジスタ指定)>

Boolean __GetADC(unsigned long data1, unsigned int *data2)

data1	設定データ 1
data2	A/D 変換された値が格納されるバッファへのポインタ

説明

指定した A/D レジスタから A/D 変換された値を取得します。

data 1 には次の値を設定することができます。

RAPI_AD0	A/D レジスタ 0 を選択します。
RAPI_AD1	A/D レジスタ 1 を選択します。
RAPI_AD2	A/D レジスタ 2 を選択します。
RAPI_AD3	A/D レジスタ 3 を選択します。
RAPI_AD4	A/D レジスタ 4 を選択します。
RAPI_AD5	A/D レジスタ 5 を選択します。
RAPI_AD6	A/D レジスタ 6 を選択します。
RAPI_AD7	A/D レジスタ 7 を選択します。

戻り値

A/D 変換された値が正常に取得されると RAPI_TRUE を返し、失敗した場合は RAPI_FALSE を返します。

機能

A/D 変換器

参考

__CreateADC、__EnableADC、__DestroyADC、__GetADCAll

備考

- 1 つめの引数で未定義の値が指定された場合には、API の動作は保証できません。

プログラム例

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
    unsigned int data;

/* A/D レジスタ 0 の A/D 変換された値を取得します */
    __GetADC( RAPI_AD0, &data );
}
```

__GetADCAll

概要

<A/D 変換値の取得 (全レジスタ)>

Boolean __GetADCAll(unsigned int *data)

data	A/D 変換された値が格納されるバッファへのポインタ

説明

すべての A/D レジスタから A/D 変換された値を取得します。

取得する A/D 変換された値がある A/D レジスタを以下に示します。

[M16C]	:	[0] A/D レジスタ 0
(16 バイト)		[1] A/D レジスタ 1
		[2] A/D レジスタ 2
		[3] A/D レジスタ 3
		[4] A/D レジスタ 4
		[5] A/D レジスタ 5
		[6] A/D レジスタ 6
		[7] A/D レジスタ 7

戻り値

A/D 変換された値が正常に取得されると RAPI_TRUE を返し、失敗した場合は RAPI_FALSE を返します。

機能

A/D 変換器

参考

__CreateADC、__EnableADC、__DestroyADC、__GetADC

プログラム例

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
    unsigned int data[8];

/* A/D レジスタの A/D 変換された値を取得します */
    __GetADCAll( data );
}
```

M16C/62P グループ
Renesas Embedded Application Programming Interface
リファレンスマニュアル

ルネサス エレクトロニクス株式会社
神奈川県川崎市中原区下沼部1753 〒211-8668

RJJ10J2099-0100