リアルタイムクロック データシート RTC V 1.0

Copyright © 2009-2011 Cypress Semiconductor Corporation. All Rights Reserved.

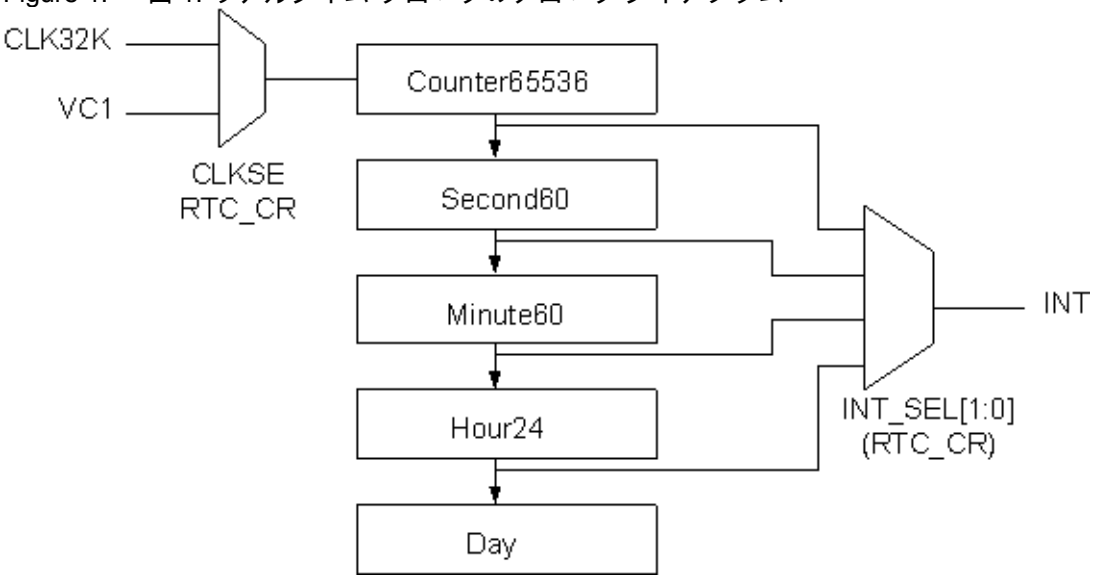
リソース	PSoC® ブロック			API メモリ （バイト数）		ピン （外部入出力ごと）
	デジタル	アナログ CT	アナログ SC	フラッシュ	RAM	
CY8C21x45, CY8C22x45, CY8C28x45, CY8C28xxx						
RTC	0	0	0	166	0	0

特性および概要

- 外付け 32K 水晶振動子を使用したリアルタイムクロック
- 秒、分、時間、日の間にある柔軟な割り込みソース
- BCD フォーマットの時間、分、秒の時間読み取りと書き込み
- VC1 をクロックソースとして使用する場合は標準タイマ
- 32K の内蔵または外付けクロックソースを用いたスリープ モード
- PPOR、IPOR、ウォッチドッグ リセットによるリセット

リアルタイムクロック ユーザ モジュールは、ファームウェアのメンテナンスなしでタイマを提供します。このユーザ モジュールは、時間 : 分 : 秒のフォーマットをサポートします。関連レジスタからデータを読むことで、時間表示が得られます。割り込みは、構成可能なパラメータの値に基づいて生成することもできます。リアルタイムクロック ユーザ モジュールは、一般タイマとリアルタイムタイマという 2 つのモードをサポートしますが、これらはクロックソースによって異なります。

Figure 1. 図 1. リアルタイム クロックのブロック ダイアグラム



機能説明

リアルタイムクロック ユーザ モジュールは、ハードウェアの RTC を使用します。RTC ブロックには 3 つのカウンタがあり、BCD フォーマットで、時間、分、秒をカウントします。秒カウンタは、65536 クロック周期ごとに 1 増えます。分カウンタは、60 秒ごとに 1 増えます。時間カウンタは、60 分ごとに 1 増えます。24 時間ごとに日の割り込みが発生します（オプション）。リアルタイムブロックには、VC1 と CLK32K という 2 つのクロック源があります。VC1 がクロック入力として選択されているときは、リアルタイム クロックブロックは、VC1 周期に基づく固定周期タイマとして使用できます。

時間、分、秒に対応するリアルタイム クロックブロックには、RTC_HRTC_M、RTC_S. という 3 つのレジスタがあります。これらはすべて読み取り· 書き込みが可能です。ファームウェアで、時間、分、秒の BCD 値を書くことができます。RTC_H の合法な範囲は 0 ~ 23、RTC_M と RTC_S では 0 ~ 59 です。

レジスタからの時間の読み取りは、同期モードまたは非同期モードでできます。

- 同期読み取りでは、HH:MM:SS データが RTC_H の読み取り動作の時間と同期されます。同期読み取りが有効の場合、MM:SS が RTC_H の読み取り動作の時間と確実に同期されるように、RTC_H の読み取りが現在の分値と秒値がバッファにラッチングされます。
- 同期読み取りが無効の場合、HH:MM:SS の読み値はそれぞれ独立して、各レジスタの時間の読み値に適合します。これはすなわち、時間レジスタの読み値が 23:59:59 の場合、次の指示で読むときには分レジスタが 00 となっているかも知れないということです。

配置

リアルタイム クロック ユーザ モジュールは RTC ブロックを独占的に使用するため、複数の場所に配置することは不可能です。

パラメータおよびリソース

クロック

クロックのパラメータは、現在のクロックソースを設定します。このパラメータの選択肢は次のとおりです。

クロック	説明
VC1	リアルタイム クロック ユーザ モジュールは、一般タイマとして使用されます。
32KHz	リアルタイム クロック ユーザ モジュールは、リアルタイム クロック ユーザ モジュールとして使用されます。

InterruptType (割り込みタイプ)

InterruptType パラメータは、割り込み周期を設定します。このパラメータの選択肢は次のとおりです。

[+] Feedback

InterruptType (割り込みタイプ)	説明
1 秒	割り込みが 1 秒に 1 回発生します。
1 分	割り込みが 1 分に 1 回発生します。
1 時間	割り込みが 1 時間に 1 回発生します。
1 日	割り込みが 1 日に 1 回発生します。

SynchronousRead

SynchronousRead パラメータは、現在の読み取りメソッドを設定します。このパラメータの選択肢は次のとおりです。

SynchronousRead	説明
ディスエーブル	HH:MM:SS の読み値はそれぞれ独立して、各レジスタの時間の読み値に適合します。
イネーブル	MM:SS が RTCH_REG の読み取り動作の時間と確実に同期されるように、RTCH_REG の読み取りが現在の分値と秒値がバッファにラッチングされます。

アプリケーション プログラミング インタフェース

アプリケーション プログラミング インタフェース (API) 関数は、高レベルでモジュールを扱うことができるように、ユーザ モジュールの一部として提供されています。このセクションでは、各機能に対するインタフェースを include ファイルによって提供される関連定数とともに示します。

ユーザ モジュールを配置するたびに、インスタンス名が割り当てられます。デフォルトでは、PSoC デザイナが、そのプロジェクトのユーザ モジュールの第 1 インスタンスに RTC_1 を割り当てます。これは識別子の構文ルールに従った一意の値に変更できます。指定されたサンプルの名前は、すべてのグローバル機能名、変数、定数記号の接頭辞になります。次の説明では、簡単にするために、インスタンス名は省略されて単に 「RTC」となっています。

Note ** ここでは、全てのユーザー モジュールの API では、A と X レジスタの値は、API 関数を呼び出すことによって変更することができます。関数を呼び出す場合、呼出し後に A と X の値が必要になるならば、必ず呼び出し前に A と X の値を保存してください。PSoC Designer のバージョン 1.0 以降では、効率を高めるために、この 「registers are volatile (レジスタの揮発性)」ポリシーが選択され、実施されています。C コンパイラは自動的にこの条件を処理します。アセンブリ言語のプログラマは、コードがこのポリシーを遵守していることも確認する必要があります。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。

RTC_Start

説明

RTC ブロックを有効にして、リアルタイムクロックを起動します。

C プロトタイプ

```
void  RTC_Start(void);
```


[+] Feedback

アセンブラ
```
lcall  RTC_Start
```
パラメータ
なし
戻り値
なし
副作用
API セクションの冒頭にある注意事項 ** を参照してください。

RTC_Stop

説明
RTC ブロックを無効にして、リアルタイムクロックを停止します。
C プロトタイプ
```
void  RTC_Stop(void);
```
アセンブラ
```
call  RTC_Stop
```
パラメータ
なし
戻り値
なし
副作用
API セクションの冒頭にある注意事項 ** を参照してください。

RTC_EnableInt

説明
RTC ブロックの割り込みを有効にします。
C プロトタイプ
```
void  RTC_EnableInt(void);
```
アセンブラ
```
lcall  RTC_EnableInt
```
パラメータ
なし
戻り値
なし
副作用
API セクションの冒頭にある注意事項 ** を参照してください。

RTC_DisableInt

説明

RTC ブロックの割り込みを無効にします。

C プロトタイプ

```
void  RTC_DisableInt(void);
```

アセンブラ

```
lcall  RTC_DisableInt
```

パラメータ

なし

戻り値

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_ClearInt

説明

指定されている RTC 割り込みをクリアします。

C プロトタイプ

```
void  RTC_ClearInt(void);
```

アセンブラ

```
lcall  RTC_ClearInt
```

パラメータ

なし

戻り値

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_SetIntPeriod

説明

リアルタイム クロック モジュールの割り込み周期を構成します。

C プロトタイプ

```
void  RTC_SetIntPeriod(BYTE bConfiguration);
```

アセンブラ

```
mov   A,bConfiguration
lcall  RTC_SetIntPeriod
```

パラメータ

bConfiguration は割り込み周期です。シンボル名は、C 言語およびアセンブリ言語で記載されています。これらに関連する値は、以下の表に記載されています。

記号名	値	説明
RTC_INT_SEC	0x00	割り込み周期を 1 秒に設定します。
RTC_INT_MIN	0x04	割り込み周期を 1 分に設定します。
RTC_INT_HOUR	0x08	割り込み周期を 1 時間に設定します。
RTC_INT_DAY	0x0C	割り込み周期を 1 日に設定します。

戻り値

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_bReadSecond

説明

秒の値を BCD フォーマットで読み取ります。

C プロトタイプ

```
BYTE  RTC_bReadSecond(void);
```

アセンブラ

```
lcall  RTC_bReadSecond
```

パラメータ

なし

戻り値

秒のデータを BCD フォーマットで返します。

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_bReadMinute

説明

分の値を BCD フォーマットで読み取ります。

C プロトタイプ

```
BYTE  RTC_bReadMinute(void);
```

アセンブラ

```
lcall  RTC_bReadMinute
```

パラメータ

なし

戻り値

分のデータを BCD フォーマットで返します。

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_bReadHour

説明

時間の値を BCD フォーマットで読み取ります。

C プロトタイプ

```
BYTE  RTC_bReadHour(void);
```

アセンブラ

```
lcall  RTC_bReadHour
```

パラメータ

なし

戻り値

時間のデータを BCD フォーマットで返します。

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_SetSecond

説明

秒の値を設定します。

C プロトタイプ

```
void  RTC_SetSecond(BYTE bSecond);
```

アセンブラ

```
mov A,bSecond
lcall  RTC_SetSecond
```

パラメータ

bSecond は秒のデータです。秒の値書き込みの合法範囲は 0 ~ 59 です。このデータは BCD フォーマットでなければなりません。

戻り値

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_SetMinute

説明

分の値を設定します。

C プロトタイプ

```
void  RTC_SetMinute(BYTE bMinute);
```

アセンブラ

```
mov A,bMinute
lcall  RTC_SetMinute
```

パラメータ

bMinute は分のデータです。分の値書き込みの合法範囲は 0 ~ 59 です。このデータは BCD フォーマットでなければなりません。

戻り値

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

RTC_SetHour

説明

時間の値を設定します。

C プロトタイプ

```
void  RTC_SetHour(BYTE bHour);
```

アセンブラ

```
mov A,bHour
lcall  RTC_SetHour
```

パラメータ

bHour は時間のデータです。時間の値書き込みの合法範囲は 0 ~ 23 です。このデータは BCD フォーマットでなければなりません。

戻り値

なし

副作用

API セクションの冒頭にある注意事項 ** を参照してください。

ファームウェア ソースコードの例

この C 言語のコード図は、RTC ユーザ モジュールの使用方法を説明しています。

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

BYTE abRTCData[3];
void main(void)
{

RTC_SetHour(0x11); // Set 11 hour as initial hour value
RTC_SetMinute(0x20); // Set 20 minutes as initial minute value
RTC_SetSecond(0x0); // Set 0 seconds as initial second value
RTC_SetIntPeriod(RTC_INT_SEC); // Set 1 sec as interrupt period
RTC_EnableInt(); // Enable RTC interrupt
RTC_Start(); // Start RTC block

M8C_EnableGInt; // Enable global interrupt

while(1)
{
abRTCData[0] = RTC_bReadHour(); // Read current hour value in BCD
abRTCData[1] = RTC_bReadMinute(); //Read current minute value in BCD
abRTCData[2] = RTC_bReadSecond(); //Read current second value in BCD
}
}
```

アセンブリ言語での同じコードは次のようになります。

```
include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

_abRTCData:
abRTCData:               BLK   3

export  main
export _main

_main:
main:
mov A, 0x11
call RTC_SetHour ;Set 11 hour as initial hour value
mov A, 0x20
call RTC_SetMinute ;Set 20 minutes as initial minute value
mov A,0x0
call RTC_SetSecond ;Set 0 seconds as initial second value
mov A, RTC_INT_SEC
call RTC_SetIntPeriod ;Set 1 sec as interrupt period
call RTC_EnableInt ;Enable RTC interrupt
call RTC_Start ;Start RTC block

M8C_EnableGInt ;Enable global interrupt
.ReadRTCData:
call RTC_bReadHour ;Read current hour value in BCD
```

```
RAM_SETPAGE_CUR >abRTCData
mov [abRTCData],A
call RTC_bReadMinute ;Read current minute value in BCD
RAM_SETPAGE_CUR >abRTCData
mov [abRTCData+1],A
call RTC_bReadSecond ;Read current second value in BCD
RAM_SETPAGE_CUR >abRTCData
mov [abRTCData+2],A
jmp .ReadRTCData
```

コンフィグレーション レジスタ

このレジスタは、現在の時間値の読み出し・書き込みを BCD フォーマットで行います。HR_DEC ビットは、時間値のデカル番号を表します。HR_UNIT ビットは時間値のユニット番号を表します。このレジスタの値は、RTC_SetHourAPI を呼び出すことで変更できます。

Table 1. RTCM_REG

ビット	7	6	5	4	3	2	1	0
値	0	MIN_DEC			MIN_UNIT			

このレジスタは、現在の分値の読み出し・書き込みを BCD フォーマットで行います。MIN_DEC ビットは、分値のデカル番号を表します。MIN_UNIT ビットは分値のユニット番号を表します。このレジスタの値は、RTC_SetMinuteAPI を呼び出すことで変更できます。

Table 2. RTCS_REG

ビット	7	6	5	4	3	2	1	0
値	0	SEC_DEC			SEC_UNIT			

このレジスタは、現在の秒値の読み出し・書き込みを BCD フォーマットで行います。SEC_DEC ビットは、秒値のデカル番号を表します。SEC_UNIT ビットは秒値のユニット番号を表します。このレジスタの値は、RTC_SetSecondAPI を呼び出すことで変更できます。

Table 3. RTCCR_REG

ビット	7	6	5	4	3	2	1	0
値	0	0	InterruptEn	ClkSelect	InterruptSelect		Sync	Enable

この Enable ビットは、RTC_Start または RTC_Stop API ルーチンを呼び出すことで変更できます。

Sync ビットは読み取りメソッドを指定します。このビットの値は、デバイス エディタのユーザ モジュール パラメータの 「SynchronousRead」パラメータの選択肢によって決定します。

InterruptSelect ビットは割り込み周期を指定します。これらのビットの値は、デバイス エディタのユーザ モジュール パラメータの 「InterruptType」パラメータの選択肢によって決定します。この値も、RTC_SetIntPeriodAPI を呼び出すことによって変更できます。

ClkSelect ビットは、クロックソースを指定します。このビットの値は、デバイス エディタのユーザ モジュール パラメータの 「Clock」パラメータの選択肢によって決定します。

InterruptEn ビットは、RTC_EnableInt または RTC_DisableIntAPI ルーチンを呼び出すことで変更できます。

バージョン ヒストリー

バージョン	著者	説明
1.0	DHA	初期バージョン

Note PSoC Designer 5.1 は、すべてのユーザ モジュール データシートにおいてバージョン ヒストリーを導入しています。このセクションでは、ユーザ モジュールの過去のバージョンと現在のバージョンとの違いに関して高度な解説を掲載しています。

Copyright © 2009-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer ™ and Programmable System-on-Chip ™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.