スリープタイマ データシート SleepTimer V 1.0

Copyright © 2001-2011 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	
CY8C29/27/24/22/21/20xxx, CY8C23x33, CY8CLED02/04/08/16, CY7C64215, CY7C64343, CY7C60413, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8CTST200, CY8CTMG2xx, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C21x12						
	0	0	0	150	7	0

このユーザ モジュールを使用する単一あるいはすべてを設定するサンプルプログラムについては、以下を参照してください。www.cypress.com/psocexampleprojects

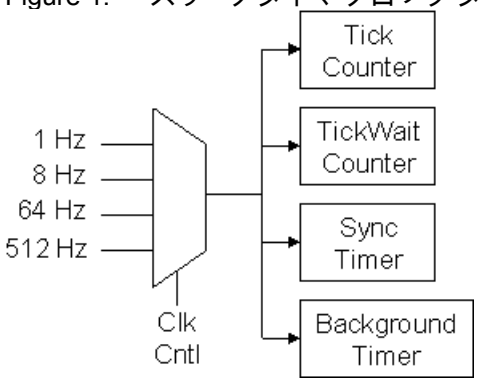
機能および概要

- デジタルブロックは不要です。
- 8、16、32-bit の刻みカウンタから選択可能
- 3 種類のタイマ機能

スリープタイマ ユーザ モジュールは、貴重なデジタルブロックを使用することなく、基本的なタイミング機能を提供します。このユーザ モジュールは標準搭載のスリープ タイマの機能を使用して、プロジェクトで利用価値の高い様々なタイミング機能を作り出します。これらの機能には以下のようなものがあります。

- バックグラウンドの刻みカウンタは、1、8、64、512 Hz で稼動
- プログラム実行を遅延するシンプルなウエイト機能
- 設定可能なカウントダウンタイマ
- ループ ガバナによるループタイマの安定な制御

Figure 1. スリープタイマ ブロック ダイアグラム



機能説明

スリープタイマ ユーザ モジュールは、標準的な PSoC システムのスリープ タイマをベースにしています。割り込み周波数は 1、8、64、512 Hz の中からユーザが選択できます。スリープ タイマ割り込みが発生するたびに、刻みカウンタが 1 カウント増加します。刻みカウンタの長さも、タイミングニーズ、割り込み周波数、RAM 使用を最適化するために、ユーザが選択できます（8、16、32 ビット）。刻みカウンタは、提供されている API を使うことでいつでも読むことができます。

すべてのタイマ機能は、刻みの増加に基づいて行われます。刻み周期は、割り込み周波数と反比例します。1、8、64、512 Hz の割り込み周波数の場合、周期はそれぞれ、1000、128、15.6、1.9 ミリ秒となります。

スリープ タイマの精度については、デバイスの Data Sheet を参照してください。

配置

スリープタイマ ユーザ モジュールは、1 つのプロジェクトに 1 つだけ配置できます。PSoC のアナログやデジタルブロックは不要ですが、PSoC スリープ タイマハードウェアと割り込みベクタを使用します。

パラメータおよびリソース

TickCounterSize

このパラメータは、刻みカウンタのサイズを選択します。利用可能なオプションは、1、2、4 バイトです。このパラメータの選択は、ユーザが割り込み周波数、タイミング分解能、測定が必要な最大周期に基づいて行います。

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) ルーチンは設計者がより高度なレベルでモジュールを処理できるようにユーザ モジュールの一部として提供されます。このセクションでは、それぞれの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定数とともに示します。

Note ここでは、全てのユーザ モジュール API と同じように、API 関数を呼び出すことで A と X レジスタの値が変更されます。API をコールした後でも A と X の前値を保持したいときは、API をコールするファンクションで A と X の値を保持する必要があります。PSoC Designer のバージョン 1.0 以降、効率性の観点から、この「registers are volatile（レジスタの揮発性）」ポリシーが採用されています。C コンパイラは自動的にこの条件を処理します。アセンブラ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しなければなりません。一部のユーザー 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。

SleepTimer_EnableInt

説明：

割り込みモード動作を有効にします。注：ただし、グローバル割り込みも、割り込みが実際に実行される前に有効にされなければなりません。スリープタイマ ユーザ モジュールが作動するには、この関数を呼び出さなければなりません。

C プロトタイプ：

```
void  SleepTimer_EnableInt(void);
```

アセンブラ：

```
lcall  SleepTimer_EnableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

このルーチンは、IO 空間に置かれている適切な割り込みエネーブルレジスタを変更します。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) の、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_DisableInt

説明：

割り込みモード動作を無効にします。

C プロトタイプ：

```
void  SleepTimer_DisableInt(void);
```

アセンブラ：

```
lcall  SleepTimer_DisableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

このルーチンは、IO 空間の適切な割り込み有効化レジスタを調整します。このルーチンは、IO 空間に置かれている適切な割り込みエネーブルレジスタを変更します。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_Start

説明：

SleepTimer 操作を開始します。刻みカウンタを許可するには、グローバル割り込みと SleepTimer 割り込みが事前に有効になっている必要があります。

C プロトタイプ：

```
void  SleepTimer_Start(void);
```

アセンブラ：

```
lcall  SleepTimer_Start
```

パラメータ：

なし

戻り値：

なし

特殊作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_Stop

説明：

この関数は、スリープ タイマの動作に影響しません。SleepTimer_DisableInt() を呼び出す、またはグローバル割り込みを無効にすると、タイマ動作が止まります。

C プロトタイプ：

```
void  SleepTimer_Stop(void);
```

アセンブラ：

```
lcall  SleepTimer_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

出力がローにセットされ、それに続く期間レジスタへの書き込みにより、カウント レジスタが新しい期間値で更新されます。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

Sleeptimer_SetInterval

説明：

この関数は割り込みの周期を設定し、それにより、すべての SleepTimer ユーザ モジュールタイミング機能用の刻み周期も設定します。

C プロトタイプ：

```
void  SleepTimer_SetInterval(BYTE bInterval);
```

アセンブラ：

```
mov   A, [bInterval]           ; Place interval in A
lcall  SleepTimer_SetInterval
```

パラメータ：

bInterval: このパラメータにより、ユーザは 1、8、64、512 Hz の 4 つの中から割り込み周波数を選択できます。

定数	値	割り込み周期
SleepTimer_1_Hz	0x18	1 Hz
SleepTimer_8_Hz	0x10	8 Hz
SleepTimer_64_Hz	0x08	64 Hz
SleepTimer_512_Hz	0x00	512 Hz

戻り値：

なし

副作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_SyncWait

説明：

この関数は、イベントの同期化に使用されます。異なるパスを持つループを、同一周期で動作するよう強制するために使用できます。この関数には 2 つのモードがあります。1 つ目のモードでは、タイマが新しい値を強制すると、関数をすぐに終了します。2 つ目のモードでは、プログラムフローが前回の遅延が過ぎるのを待ち、新しい遅延をセットしてから終了します。

C プロトタイプ：

```
void  SleepTimer_SyncWait(BYTE bTicks, BYTE bMode);
```

アセンブラ：

```
mov   A, [bTicks]                ; Load tick wait period in A
mov   X, SleepTimer_WAIT_RELOAD  ; Load mode in X
lcall  SleepTimer_SyncWait
```

パラメータ :

bTicks: 設定された周期の時間をティック数で表したもの。SleepTimer_SetInterval 関数によって間隔が 64Hz に設定されている場合、同期遅延は bTicks x 1/64 となります。bTicks が 8 に設定されている場合、同期時間は 125 ミリ秒になります。bMode: このパラメータは、この関数のモードを設定します。1 に設定されている場合、新しい時間が設定され、プログラムフローがすぐに関数を終了します。bMode = 0 の場合、前回の遅延時間の残りが終了するまでプログラム実行は延期されます。新しいティックカウントが読み込まれ、関数が終了します。

bMode	値	コメント
SleepTimer_FORCE_RELOAD	0x01	タイマをすぐに再ロードします。
SleepTimer_WAIT_RELOAD	0x00	タイマが終了するのを待って、タイマを再ロードします。

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_TickWait

説明 :

この関数は、プログラムの実行を n 回のティック単位で遅延するために使用します。

C プロトタイプ :

```
void  SleepTimer_TickWait(BYTE bTicks);
```

アセンブラ :

```
mov   A, [bTicks]                ; Load tick wait period in A
lcall  SleepTimer_TickWait
```

パラメータ :

bTicks: プログラム実行を遅延する時間の長さをティック数で表したもの。この関数は、刻みカウンタがゼロにカウントダウンするまで完了しません。

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_SetTimer

説明 :

この関数はタイマを、SleepTimer 割り込みが実行されるたびに 1 (ティック) 少ない値に設定します。このレートは SleepTimer_SetInterval() 関数によって設定されます。この関数は、カウントダウンタイマを設定し、すぐに終了します。

C プロトタイプ :

```
void  SleepTimer_SetTimer(BYTE bTicks);
```

アセンブラ :

```
mov   A, [bTicks]                ; Load timer period in A
lcall  SleepTimer_TickWait
```

パラメータ :

bTicks: タイマがゼロにカウントダウンされるまでのタイマの量をティック数で表したもの。

戻り値 :

なし

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_bGetTimer

説明 :

カウントダウンタイマの残り時間を読み込みます (ティック数)。

C プロトタイプ :

```
WORD  SleepTimer_bGetTimer(void);
```

アセンブラ :

```
lcall  SleepTimer_bGetTimer
mov   [bTickRemaining], A         ; Returns remain tim in A
```

パラメータ :

なし

戻り値 :

bTimeLeft: カウントダウンが終了するまでの残りティック数。戻り値がゼロの場合、タイマは関数呼び出しの前に終了します。

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_bGetTickCntr

説明：

バックグラウンド刻みカウンタの最下位 8 ビットを返します。

C プロトタイプ：

```
BYTE  SleepTimer_bGetTickCntr(void);
```

アセンブラ：

```
lcall  SleepTimer_bGetTickCntr
mov   [bTickCount], A         ; Returns LSB of tick counter in A
```

パラメータ：

なし

SleepTimer_iGetTickCntr

説明：

バックグラウンド刻みカウンタの最下位 16bits ビットを返します。

C プロトタイプ：

```
unsigned int SleepTimer_iGetTickCntr(void);
```

アセンブラ：

```
lcall  SleepTimer_iGetTickCntr
mov   [iTickCounter], X           ; MSB returned in X
mov   [iTickCounter+1], A         ; LSB returned in A
```

パラメータ：

なし

戻り値：

bTicks: 刻みカウンタの現在の 16LSB を返します。

副作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

SleepTimer_lGetTickCntr

説明：

32-bit バックグラウンド刻みカウンタの現在値を返します。

C プロトタイプ：

```
unsigned long * SleepTimer_lGetTickCntr(unsigned long * lPtr);
```

アセンブラ：

```
mov   A, >lTickCount
mov   X, <lTickCount              ; A:X points to the return buffer
lcall  SleepTimer_lGetTickCntr
```

パラメータ :

IPtr: 返される 32-bit 刻みカウンタ変数のポインタ。

戻り値 :

ITimeLeft: カウントダウンが終了するまでの残りティック数。戻り値がゼロの場合、タイマは関数呼び出しの前に終了します。

副作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリ モデル (CY8C29xxx) では、すべての RAM ページ ポインタ レジスタでも同じです。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

ファームウェア ソースコードの例

ループ速度を制御する簡単な C 言語の例をここに示します。

```
//-------------------------------------------------------------------
// Sample Code for the SleepTimer User Module
// In this example, the sleep timer interrupt rate is set to 64Hz.
// The loop is controlled to loop no faster than 8Hz.
//-------------------------------------------------------------------


#include "m8c.h"
#include "PSoCAPI.h"

void main(void)
{
   M8C_EnableGInt ;                              // Turn on interrupts
   SleepTimer_Start();
   SleepTimer_SetInterval(SleepTimer_64_HZ);    // Set interrupt to a
   SleepTimer_EnableInt();                       // 64 Hz rate


   // The following loop should loop no faster than 8 Hz (64/8)
   while(1) {
      SleepTimer_SyncWait(8, SleepTimer_WAIT_RELOAD);


   // …Other code in loop
   }
}
```

ループ速度を制御する簡単なアセンブリ言語の例をここに示します。

```
;-------------------------------------------------------------------------
; Sample Code for the SleepTimer User Module
; In this example, the sleep timer interrupt rate is set to 64Hz.
; The loop is controlled to loop no faster than 8Hz.
;-------------------------------------------------------------------------

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

export _main

_main:

   M8C_EnableGInt               ; Turn on interrupts
   lcall SleepTimer_Start       ; Start SleepTimer
   mov   A,SleepTimer_64_HZ     ; Use 64 Hz interrupt rate
   lcall SleepTimer_SetInterval ;
   lcall SleepTimer_EnableInt   ; Enable SleepTimer Interrupt

.Loop:
   mov   A,8                     ; Set tick delay to 8 ticks
   mov   X,SleepTimer_WAIT_RELOAD ; Wait for delay then reload timer
   lcall SleepTimer_SyncWait     ;

   ;  Put main loop user code here

   jmp .Loop
```

コンフィグレーション レジスタ

このレジスタは、初期化および API ライブラリによって構成されます。ユーザが、このレジスタを直接変更または読み取る必要はありません。このセクションは参考用です。

Table 1. レジスタ OSC_CR

ビット	7	6	5	4	3	2	1	0
値	32k 選択	PLL モード	No Buzz (ブザーなし)	Sleep[1]	Sleep[0]	CPU Speed[2]	CPU Speed[1]	CPU Speed[0]

ビット 3 と 4 (Sleep[1:0]) のみが SleepTimer ユーザ モジュール中で変更されます。

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 © 2001-2011 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.