16-Bit タイマ データシート Timer16 V 1.1

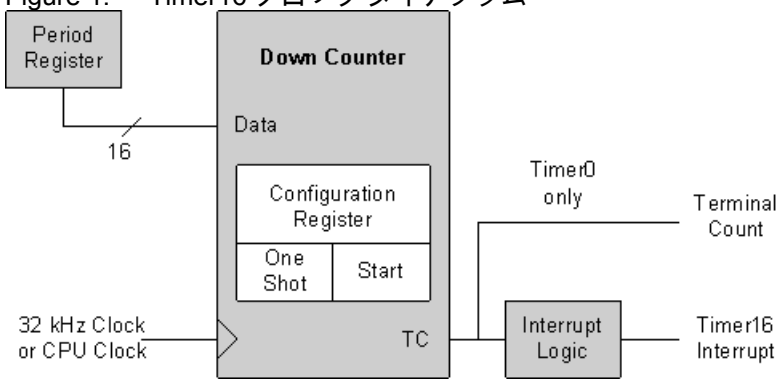
リソース	PSoC® ブロック				API メモリ（バイト数）		センサー当たりのピン数
	CapSense®	I2C/SPI	タイマ	コンパレータ	フラッシュ	RAM	
CY8C20x66, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20x46, CY8C20x96, CY7C64xxx, CY7C60413, CY7C60424, CY7C6053x, CYONS2010, CYONS2011, CYONSFN2051, CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161, CYONSFN2162, CY8CTST200, CY8CTMG2xx							
			1		36	0	なし

特長および概要

- 16-bit のプログラマブルなカウントダウンタイマ
- 期間値がターミナルカウントにリロードされないワンショットの ダウンカウントオプション
- 最終カウントで割り込みが発生
- 内部の 32 kHz クロックまたは CPU クロックを使用

このユーザ モジュールは、割り込みコールバックを伴う 16-bit のプログラマブルなタイマです。クロックは、内部の 32 kHz クロックまたは CPU クロックから供給されます。

Figure 1. Timer16 ブロック ダイアグラム



機能説明

Timer16 ユーザ モジュールは、ワンショット カウントダウンを実行する（ワンショットモード）か、一定期間にカウントダウンを連続的に繰り返す（連続モード）があります。デフォルトのモードと期間値は、デバイス エディタのパラメータセクションに設定するか、アプリケーション プログラミング インタフェース（API）セッションで SetMode() と SetPeriod() を使用してプログラム的に設定されます。

DC 電気的特性と AC 電気的特性

Table 1. Timer16 DC 電気的特性と AC 電気的特性

パラメータ	条件および注記	標準値	制限	単位
F_{32K1}		32	-	kHz

タイミング

開始すると、プログラマブルタイマが、データレジスタの値をロードし、最終カウントのゼロまでカウントダウンします。タイマは、最終カウントに到達すると 1 クロックサイクル分のアクティブ HIGH 最終カウントパルスを出力します。最終カウントパスルーティングの LOW 時間は、ロードされた 10 進のカウント値にクロック周期を掛けたものです。(TCpw = COUNT VALUEdecimal * CLKperiod) ． 最終カウント出力の期間は、最終カウントのパルス幅と 1 クロック周期を足したものです。(TCperiod = TCpw + CLKperiod).

TIMER0 だけがこの最終カウント出力を出力します。TIMER1 と TIMER2 は、最終カウント出力を出力しません。

Figure 2. Timer16 連続動作

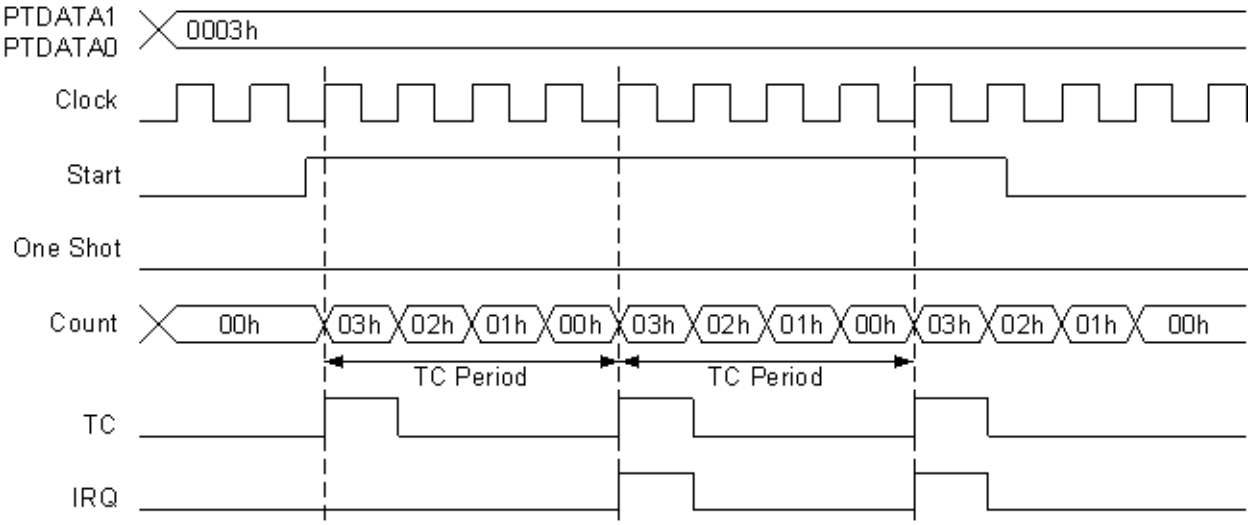
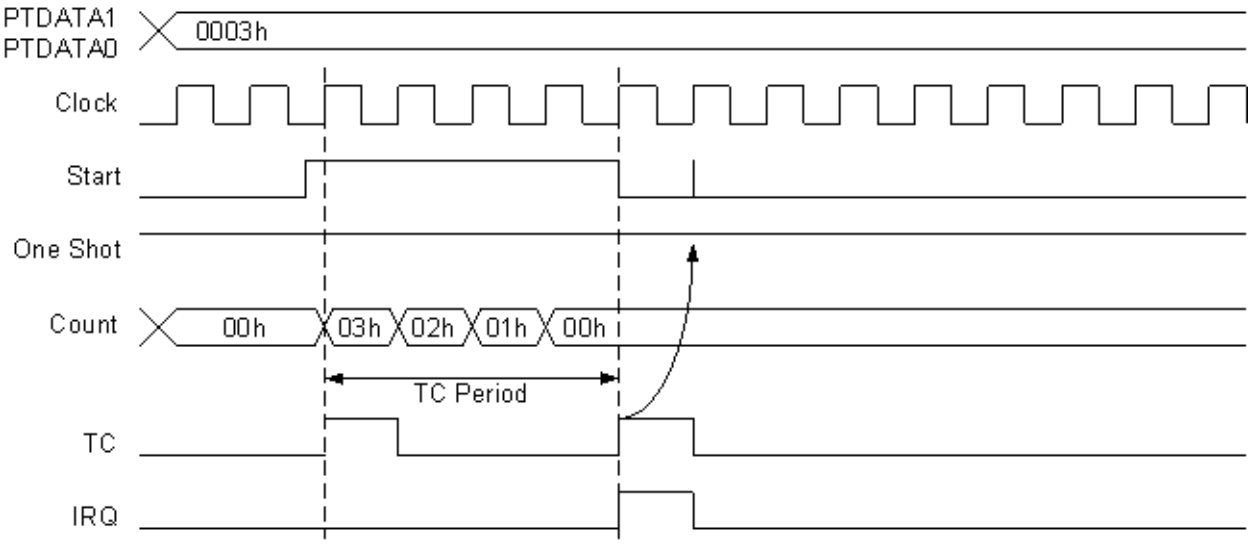


Figure 3. Timer16 ワンショット動作



配置

使用可能な TIMER ブロックの 1 つである Timer16 を配置します。注：TIMER0 だけが最終カウントに TC 信号を出力します。TIMER1 と TIMER2 は、これを出力しません。

パラメータおよびリソース

Period (期間)

1-65535 の範囲にある 16-bit 値は、カウントダウン期間を指定します。

モード

Timer16 のモードを設定します。「ワンショット」「連続モード」という 2 つのオプションがあります。ワンショットモードでは、タイマがフル カウントサイクルを完了して、動作を終了します。終了すると、このレジスタの START ビットがクリアされます。連続モードでは、タイマはカウントサイクルが終了するたびにカウント値をロードし、これを繰り返します。

クロック選択

入力クロックを選択します。使用可能なクロックから 1 つを選択します。

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、各関数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。

Note ここでは、API 機能を呼び出して、A および X レジスタの値を、すべてのユーザーのモジュールの API に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、そのコードが、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。

ここに、Timer16 提供により API 関数のリストを挙げます。

Timer16_Start

説明：

スタートビットに 1 を書いて Timer16 を有効にします。

C プロトタイプ：

```
void  Timer16_Start(void)
```

アセンブラ：

```
lcall  Timer16_Start
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数によって、A および X レジスタが変更される場合があります。

Timer16_Stop

説明：

スタートビットをクリアして、Timer16 を無効にします。

C プロトタイプ：

```
void  Timer16_Stop(void)
```

アセンブラ：

```
lcall  Timer16_Stop
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数によって、A および X レジスタが変更される場合があります。

Timer16_EnableInt

説明：

Timer16 最終カウント割り込みを有効にします。

C プロトタイプ：

```
void Timer16_EnableInt(void)
```

アセンブラ：

```
lcall  Timer16_EnableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

Timer16_DisableInt

説明：

Timer16 最終カウント割り込みを無効にします。

C プロトタイプ：

```
void  Timer16_DisableInt(void)
```

アセンブラ：

```
lcall  Timer16_DisableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

Timer16_SetMode

説明：

モードビット値を構成レジスタに書くことにより、モードを設定します。

C プロトタイプ：

```
void  Timer16_SetMode(BYTE bMode)
```

アセンブラ：

```
mov   A, [bMode] ; place bMode in A
lcall  Timer16_SetMode
```

パラメータ：

BYTE bMode: ワンショットモード（ビット値 1）または連続モード（ビット値 0）を 指定します。

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

Timer16_SetPeriod

説明：

期間レジスタに 16-bit 値を書き込むことによって、期間を指定します。

C プロトタイプ：

```
void  Timer16_SetPeriod(WORD wPeriod)
```

アセンブラ：

```
mov   X, [wPeriod+0] ; place MSB in X
mov   A, [wPeriod+1] ; place LSB in A
lcall  Timer16_SetPeriod
```

パラメータ：

wPeriod - 16-bit 期間値

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

ファームウェア ソースコードの例

次の C 言語とアセンブリ言語のサンプルコードでは、グローバル割り込みとタイマ割り込みが有効で、割り込みハンドラが実行されます。timer16int.asm の Timer16_ISR ルーチンは、ユーザコードバナー間に 「ljmp _myTimer_ISR_Handler」という文字列を加えることで、変更します。次にスタートルーチンが呼び出され、カウントダウン期間が開始します。タイマの刻みは、タイマが最終カウント（0）に到達するたびに増加します。254 回のタイマの刻みで、GPIO pin Port_1_0 が立ち上がり、下がります。

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

BYTE  timerTicks;

#pragma interrupt_handler myTimer_ISR_Handler;
void              myTimer_ISR_Handler( void );

void main(void)
{
    timerTicks = 0;
    M8C_EnableGInt;

    Timer16_EnableInt();
    Timer16_Start();

    while( 1 )
    {
        if (timerTicks > 254)
        {
            timerTicks = 0;
            //Port_1_0 is set to a Strong Drive
            Port_1_0_Data_ADDR |= Port_1_0_MASK;  //Raise Port_1_0
            Port_1_0_Data_ADDR &= ~Port_1_0_MASK; //Lower Port_1_0
        }
     }
}


//--------------------------------------------------------------------------
// myTimer_ISR_Handler    // The handler for the Timer ISR
//--------------------------------------------------------------------------
void myTimer_ISR_Handler(void)
{
timerTicks++;
}
```

アセンブリ言語で書かれた同等のコードの例を下記に示します。

```
include "PSoCAPI.inc"
include "m8c.inc"       ; part specific constants and macros
include "PSoCGPIOInt.inc"

area    bss     (RAM,REL)
timerTicks:                 blk     1

area text(ROM,REL)
export _main
export _myTimer_ISR_Handler

_main:
    mov   [timerTicks], 0
    M8C_EnableGInt
    lcall  _Timer16_EnableInt
    lcall  _Timer16_Start

loop:
    mov   A,254
    cmp   A,[timerTicks]
    jnc   loop
    mov   [timerTicks], 0
    ;Port_1_0 is set to a Strong Drive
    or    reg[Port_1_0_Data_ADDR],1   ;Raise Port_1_0
    and   reg[Port_1_0_Data_ADDR],254 ;Lower Port_1_0
    jmp   loop

;-----------------------------------------------------------------------
; myTimer_ISR_Handler    // The handler for the Timer ISR
;-----------------------------------------------------------------------
_myTimer_ISR_Handler:
    inc   [timerTicks]
    reti
```

コンフィグレーション レジスタ

このユーザ モジュールの構成に使用するタイマ PSoC ブロックレジスタに関する説明を示します。

Table 2. ブロック Timer16, Timer0 構成レジスタ PTx_CFG

ビット	7	6	5	4	3	2	1	0
値	予約済み					CLKSEL	ワンショット	START

CLKSEL - このビットは、タイマが 32kHz クロックまたは CPU クロックのどちらに基づくかを指定します。このビットが 1'b1 に設定されている場合はタイマは CPU クロックに基づき、それ以外の場合は、タイマは 32 kHz クロックに基づきます。

ワンショット - このビットは、タイマがワンショットモードと連続モードのうちどちらで作動するかを指定します。ワンショットモードでは、タイマがフル カウントサイクルを完了して、動作を終了します。終了すると、このレジスタの START ビットがクリアされます。連続モードでは、タイマはカウントサイクルが終了するたびにカウント値をロードし、これを繰り返します。

START - このビットは、タイマのカウントをフル カウントから開始します。フル カウントはデータレジスタにロードされた値で指定されます。このビットは、フル カウントサイクルの終了に伴い、タイマがワンショットモードで稼動している場合にクリアされます。

Table 3. ブロック Timer16、データレジスタ 1、PTx_DATA1

ビット	7	6	5	4	3	2	1	0
値	データ							

これらのビットは、タイマの 16-bit カウント値の上位 8 ビットを占めます。

Table 4. ブロック Timer16、データレジスタ 0、PTx_DATA0

ビット	7	6	5	4	3	2	1	0
値	データ							

これらのビットは、タイマの 16-bit カウント値の下位 8 ビットを占めます。

バージョン ヒストリー

バージョン	著者	説明
1.1	DHA	Timer16.h ファイルと Timer16.inc ファイルに、レジスタのエイリアスを追加。

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の違いの概要を説明しています。