13-Bit Timer Datasheet TIMER13V 1.1

Copyright © 2006-2015 Cypress Semiconductor Corporation. All Rights Reserved.

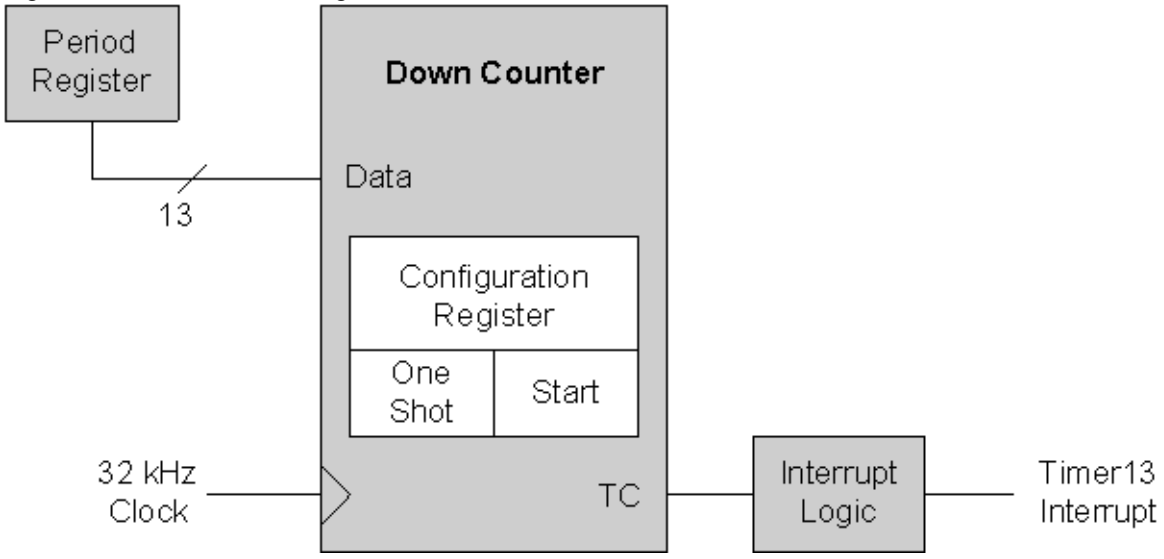
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	CapSense®	I²C/SPI	Timer	flash	RAM	
CY8C20x34, CY8C20x24			X	41	0	None

Features and Overview

- 13-bit programmable countdown timer
- One shot countdown option where the period is not reloaded on Terminal Count
- Interrupt occurs on Terminal Count
- Uses the internal 32 kHz clock

The user module is a 13-bit programmable timer with interrupt call back. It is clocked by the internal 32 kHz clock.

Figure 1. Timer Block Diagram



Functional Description

The Timer13 User Module either does a one shot countdown (One Shot mode) or continuously repeats the countdown (Continuous mode) given a certain period. The default Mode and Period values are set in the parameters section of the Device Editor, or programmatically using the SetMode() and SetPeriod() as described in the Application Programming Interface (API) section.

DC and AC Electrical Characteristics

Table 1. Timer13 DV and AC Electrical Characteristics

Parameter	Conditions and Notes	Typical	Limit	Units
F_{32K1}		32	-	kHz

Timing

When started, the programmable timer loads the value contained in its data registers and counts down to its terminal count of zero. The timer outputs an active high terminal count pulse for one clock cycle upon reaching the terminal count. The low time of the terminal count pulse is equal to the loaded decimal count value, multiplied by the clock period. (TCpw = COUNT VALUEdecimal * CLKperiod). The period of the terminal count output is the pulse width of the terminal count, plus one clock period. (TCperiod = TCpw + CLKperiod).

Figure 2. Timer13 Continuous Operation

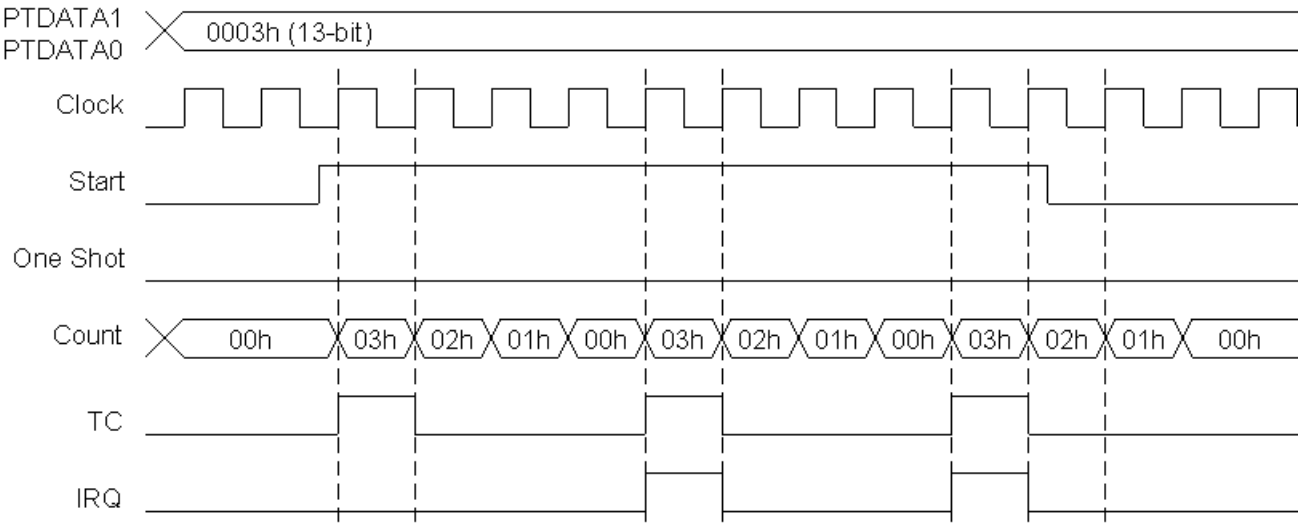
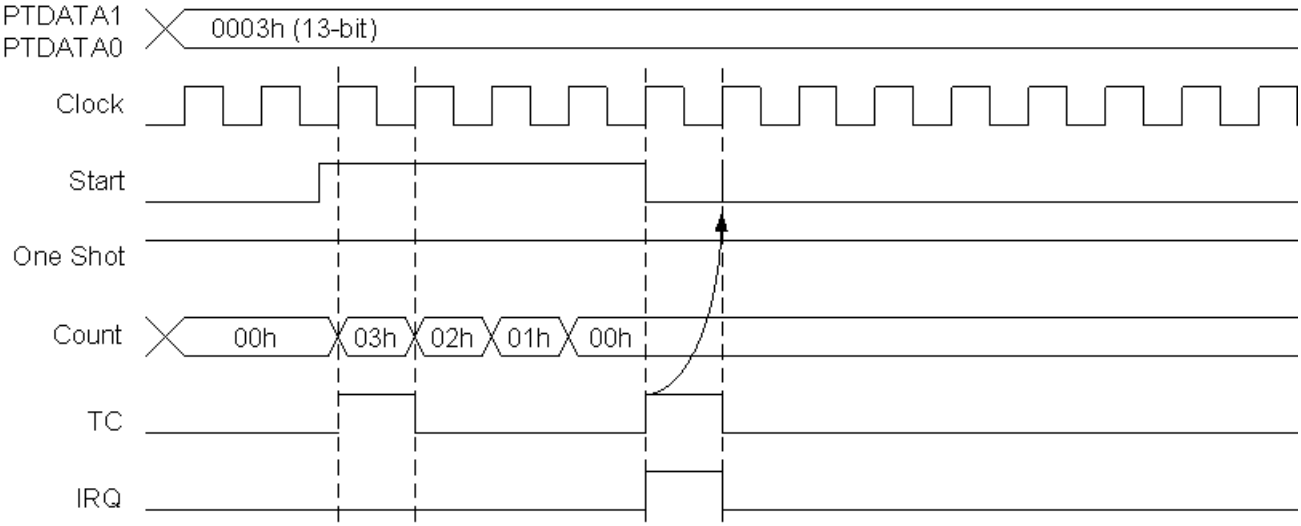


Figure 3. Timer13 One Shot Operation



Placement

Place Timer13 in the TIMER block.

Parameters and Resources

Period

This parameter is a 13-bit value that ranges from 1-8191 to determine the countdown period.

Mode

This parameter sets the mode of the Timer13. The two options are One Shot or Continuous mode. In One Shot mode, the timer completes one full count cycle and terminates. At termination, the START bit in this register is cleared. In Continuous mode, the timer reloads the count value each time at completion of its count cycle and repeats.

Interrupt Generation Control

The IntDispatchMode and InterruptAPI parameters are only accessible when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**.

InterruptAPI

The InterruptAPI parameter allows conditional generation of a user module's interrupt handler and interrupt vector table entry. Select **Enable** to generate the interrupt handler and interrupt vector table entry. Select **Disable** to bypass the generation of the interrupt handler and interrupt vector table entry. Properly selecting whether to generate an Interrupt API is recommended particularly with projects that have multiple overlays where a single block resource is used by the different overlays. By selecting only Interrupt API generation when it is necessary the need to generate an interrupt dispatch code is eliminated, reducing overhead.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting ActiveStatus causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting OffsetPreCalc causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Application Programming Interface

Note

In this, as in all user module APIs, the values of the A and X register are altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X before the call if those values are required after the call. This registers are volatile policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure that their code observes the policy, as well. Though some user module API functions may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

This is the list of Timer13 supplied API functions.

Timer13_Start

Description:

Enables the Timer13 by writing a one to the start bit.

C Prototype:

```
void  Timer13_Start(void)
```

Assembly:

```
lcall  Timer13_Start
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Timer13_Stop

Description:

Disables the Timer13 by clearing the start bit.

C Prototype:

```
void  Timer13_Stop(void)
```

Assembly:

```
lcall  Timer13_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Timer13_EnableInt

Description:

Enables the Timer13 terminal count interrupt.

C Prototype:

```
void Timer13_EnableInt(void)
```

Assembly:

```
lcall  Timer13_EnableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Timer13_DisableInt

Description:

Disables the Timer13 terminal count interrupt.

C Prototype:

```
void  Timer13_DisableInt(void)
```

Assembly:

```
lcall  Timer13_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Timer13_SetMode

Description:

Sets the mode by writing the mode bit value to the configuration register.

C Prototype:

```
void  Timer13_SetMode(BYTE bMode)
```

Assembly:

```
mov   A, [bMode] ; place bMode in A
lcall  Timer13_SetMode
```

Parameters:

BYTE bMode: The mode, OneShot (bit value 1) or Continuous (bit value 0). Defines to be provided.

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Timer13_SetPeriod

Description:

Sets the period by writing the 13-bit value to the period register.

C Prototype:

```
void  Timer13_SetPeriod(WORD wPeriod)
```

Assembly:

```
mov   X, [wPeriod+0] ; place MSB in X
mov   A, [wPeriod+1] ; place LSB in A
lcall  Timer13_SetPeriod
```

Parameters:

wPeriod - a 13-bit period value

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

Sample Firmware Source Code

In the following C and Assembly sample code, the global and timer interrupts are enabled to allow the interrupt handler to execute. The Timer13_ISR routine in timer13int.asm must be changed to call myTimer_ISR_Handler. The start routine is then called to start the countdown period. The timer ticks are incremented every time the timer reaches the Terminal Count (0). For every 254 timer ticks GPIO pin Output is raised and lowered.

Note Please rename the required output pin in the Chip Editor to "Output".

```
#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;

    Timer13_EnableInt();
    Timer13_Start();

while( 1 )
    {
        if (timerTicks > 254)
        {
            timerTicks = 0;
            //Output is set to a Strong Drive
```

```
            Output_Data_ADDR |= Output_MASK;  //Raise Output
            Output_Data_ADDR &= ~Output_MASK; //Lower Output
        }
     }
}


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

Here is an example of the equivalent code, written in Assembly language.

```
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  _Timer13_EnableInt
    lcall  _Timer13_Start

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

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

Configuration Registers

The Timer PSoC block registers used to configure this user module are described here.

Table 2. Block Timer13, Configuration Register

Bit	7	6	5	4	3	2	1	0
Value	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	One Shot	START

One Shot - This bit determines if the timer runs in one shot mode or continuous mode. In one shot mode the timer completes one full count cycle and terminates. At termination, the START bit in this register is cleared. In continuous mode, the timer reloads the count value each time upon completion of its count cycle and repeats.

START - This bit starts the timer counting from a full count. The full count is determined by the value loaded into the DATA registers. This bit is cleared when the timer is running in one shot mode upon completion of a full count cycle.

Table 3. Block Timer13, Data Register 1

Bit	7	6	5	4	3	2	1	0
Value	Reserved	Reserved	Reserved	Data				

These bits hold the upper 5 bits of the timer's 13-bit count value.

Table 4. Block Timer13, Data Register 0

Bit	7	6	5	4	3	2	1	0
Value	Data							

These bits hold the lower 8 bits of the timer's13-bit count value.

Version History

Version	Originator	Description
1.1	DHA	Added Version History

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2006-2015 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.