Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN2141

PSoC® 1 Glitch Free PWM

Author: Uroš Platiše

Associated Project: Yes

Associated Part Family: CY8C27xxx, CY8C29xxx

Software Version: PSoC® Designer™ 5.4

Related Application Notes: None

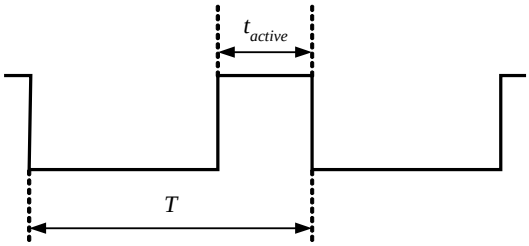
To get the latest version of this application note, or the associated project file, please visit http://www.cypress.com/go/AN2141.

PSoC® digital blocks can operate as 8-bit pulse width modulators (PWM). Due to the special design of the PSoC, these modules do not incorporate glitch-free hardware logic but instead logic must be considered in software. AN2141 presents a solution with a first-order predictor adequate to precisely generate any function.

Introduction

Pulse width modulation represents one of the most versatile ways of expressing signals in a digital control. Such a signal can be amplified easily and, consequently, is the key element of all power supplies and power converters. Discrete values that are referred to as the duty cycle of some period are represented as a ratio of the time when output is active, t_{active}, to the whole period, T. This is presented in Figure 1 and Equation (1).

Figure 1. Duty Cycle



$$duty = \frac{t_{active}}{T} \qquad \text{Equation 1}$$

The values are transmitted in counts with a period of T. Normally, T and t_{active} are of forms:

$$T = (N+1)dt \qquad \text{Equation 2}$$

and

$$t_{active} = ndt \qquad \text{Equation 3}$$

dt is the smallest time step equal to the PWM clock frequency, f_{clk}. The number of active steps, usually called pulse width, is n and $N+1$ is the total number of steps per period.

Both n and N are referenced from zero because of the hardware implementation.

The PSoC PWM is implemented as a down counter with a period register that holds the N value and a compare register (also called pulse width register) that holds the n value. The PSoC block runs down to zero from the maximum value defined in the period register. At the zero point, the down counter reloads the content from the period register and repeats this cycle. Output is set high whenever the down counter is less than (LESS) the value stored in the compare register or less-than-or-equal-to (LESS-EQUAL) the value stored in the compare register. Otherwise, it is cleared.

The length of the period determines the resolution of the PWM that is identical to the effective number of bits:

$$B = \log_2(N+1) \qquad \text{Equation 4}$$

N, n, dt, and type of comparison (LESS or LESS-EQUAL) parameters completely define the PWM output.

PWM output frequency is defined as:

$$f_{pwm} = \frac{f_{clk}}{N+1} \qquad \text{Equation 5}$$

The type of comparison defines the output according to n.

When the comparison is set to LESS-EQUAL (i.e., *down_counter* ≤ *compare_register*, the higher the value in the compare register brings out a higher value of duty cycle), n is at least 1 and ranges from 1 to maximum value. Using the LESS comparison (*down_counter* < *compare_register*), n ranges from 0 to maximum value-1. The maximum value represents the highest possible number to be stored in the compare register. In the case of an 8-bit PWM, this value is 255. When the period and compare values are both set to 255, the period will take 256 cycles and the output asserts for 255 cycles.

Finally, it is important to notice how the PWM block generates interrupts. Interrupts can be generated by terminal count (TC). TC is when the counter reaches zero and reloads with the contents of the period register, or when output rising edge (RE) occurs. Note that the output may remain in high (active) state or low (inactive) state for several periods, skipping the generation of an RE interrupts.

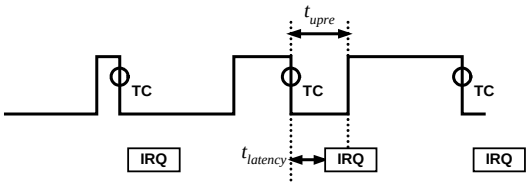
Changing the Pulse Width (n)

PWM steady conditions do not require special attention. Users can simply modify the pulse width once in a while. Possible glitches would be insignificant over several tens or hundreds of periods.

We shall focus on cases where the pulse width is changed constantly, from period to period. Because the PSoC PWM does not provide extra buffering on the pulse width register, writing a new value to it may cause an immediate change in the result of the down counter for compare/pulse width register comparison.

Proper modifications to the compare register can be achieved through interrupt handlers or by observing interrupt status flags. Indeed, many power converters require maximum pulse widths equal to 50% of the period (duty cycle = 50% ≥ n = N/2). The TC interrupt should be used in these cases, as shown in Figure 2.

Figure 2. Terminal Count Interrupt

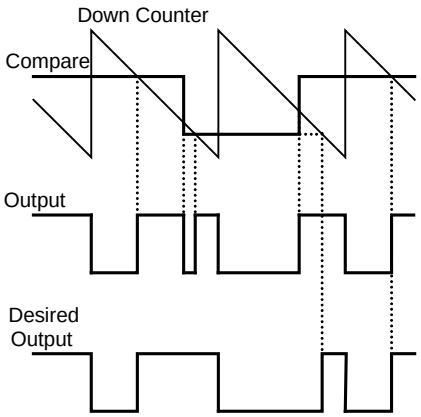


Note Changes written to the compare register in the interrupt handler IRQ caused by the TC interrupt of the current (i^{th}) period (past period) are valid for the following (i+1) period. Maximum glitch-free pulse width is limited by interrupt latency time plus the time needed to update the compare register. In Figure 2, t_{upre} represents a maximum update-to-rising edge window in which the compare register is safely updated.

Generating Glitches

The story becomes more interesting when the duty cycle is in the upper range close to 100% or when it varies over all ranges from 0 to 100%. The following figure shows an example of how easily a glitch may be generated.

Figure 3. Glitch Example



In general, we may conclude that the PWM compare register of lower duty cycle signals (up to approximately 70% depending on f_{pwm} and CPU speed) may safely be updated on TC interrupts and on RE interrupts with higher duty cycles. For perfect operation, updates on RE interrupts need an additional delay estimator, as shown later.

Implementation of a PWM with a full range (0 to 100%) of duty cycles is required to use a combination of RE and TC interrupts.

Here, we shall propose a solution with a first-order predictor. A predictor is used to estimate the optimal interrupt source in subsequent periods. A first-order predictor fulfills the requirements of most applications.

Solution with First-Order Predictor

Before exploring an example pattern of a PWM signal, rules, notes and basic features of the first-order predictor are described.

Let us assume a PWM signal that is over several periods. Each period is assigned a pulse width, $W(i)$, where $W(0)$ is the width of the current period (i=0), $W(1)$ is width of the next period (i=1), $W(2)$ is the width of the second period, etc. Our purpose is to find an algorithm that implements:

$$W(i) = func(i) \qquad \text{Equation 6}$$

func(i) is an arbitrary function, such as sin-wave, triangle, saw-tooth, etc.

The pulse width must be updated at least once per period to obtain the desired stream. This update is, of course, always done inside the interrupt service routine (ISR), also referred to as the interrupt handler. The purpose of the predictor is to best select the interrupt source so the pulse width is updated in time.

Predictor Rules

The first-order predictor uses a current pulse width $W(0)$ and one pulse width in advance $W(1)$ to estimate the optimal interrupt source. As said, there are two different interrupt sources but three different cases:

- TC: terminal count interrupt source
- RE: rising edge interrupt source
- TCRE: terminal count instead of rising edge when RE cannot be generated

TCRE, the last interrupt source, uses a TC interrupt whenever an RE interrupt cannot be generated (when output is always high). It is identified as a different interrupt source for easier understanding.

As we have three different cases, the predictor has nine transition possibilities. Predictor rules are summarized in the following table.

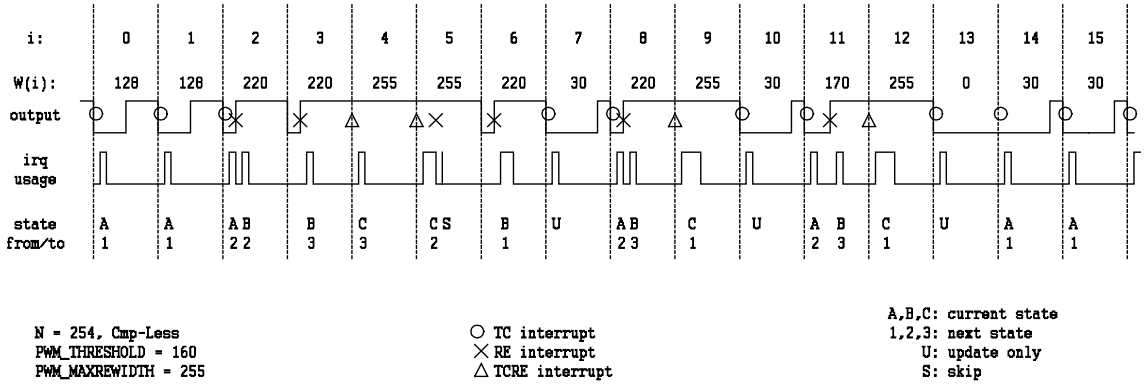
States A, B, and C represent the current state in period i, and states 1, 2, and 3 represent a new optimal interrupt source for the next period, i+1. Hereinafter, we assume i=0, unless otherwise specified. Later, we shall use these state signatures in describing transitions.

Table 1. Predictor Transition Possibilities

A: TC	B: RE	C: TCRE	
W ≤ T [Update]	W ≤ T [SkipUp] [ParUp]	W ≤ T [SkipUp] [ParUp]	1: TC
M > W > T [Update]	M > W > T [Wait] [Update]	M > W > T [Wait] [Update] [SkipNxtRE]	2: RE
INVALID	W ≥ M [Update]	W ≥ M [Update*]	3: TCRE

- W stands for pulse width $W(1)$.
- T is equal to PWM_THRESHOLD and defines the transition point of TCRE interrupt sources. It is in the range [0,N].
- M is equal to PWM_MAXREWIDTH and defines the maximum width at which output is always high. For example, if N=255, n=254, and less comparison is selected, then RE is never generated since the output is always set. In this case M=255.
- [Update] command instructs the pulse width to be updated and the next $W(1)$ to be fetched.
- [Update*] command may be skipped because the output state is always high (this is just a note).
- [SkipUp] command skips the pulse width update of value $W(1)$ in this interrupt and performs it in the next interrupt.
- [ParUp] command performs a partial update. It changes the current width to the width of value PWM_THRESHOLD. This gives extra time to the TC interrupt executed at the end of the period to properly modify the pulse width. Note that this command waits for a specific amount of time before the width is updated. If this amount of time is too little, several glitches may occur.
- [Wait] command adds delay of form T_{delay} = [W(0)-W(1)]k – n only if T_{delay} > 0. The Wait function is described later.
- [SkipNxtRE] command skips the next RE interrupt.

Figure 4. PWM Pattern Example



The first-order predictor shall fail only in two cases:

- Accurate transition from pulse width under PWM_THRESHOLD to pulse widths close to the maximum pulse width. This is due to the TC interrupt latency and time it takes to update the compare register.
- Transition from pulse width under PWM_THRESHOLD to maximum pulse width. This is an INVALID transition.

Description of Transitions/Example Pattern

Figure 4 illustrates all nine transition possibilities, places of pulse width updates, and all three types of interrupts. The represented sequence has 16 periods. Each transition shall be described separately.

Transition A1

State A (TC) to 1 (TC) is the default state after power up. Thus, the initial value must be under PWM_THRESHOLD for proper operation. Transition A1 means no change in interrupt source as seen in cycles 0, 1, 14 and 15.

The pulse width is updated at the very beginning of the ISR to support the highest possible frequencies. The update is marked with the [Update] command.

Transition A2

State A (TC) to 2 (RE) occurs whenever the pulse width $W(1)$ is over PWM_THRESHOLD and below PWM_MAXREWIDTH. This condition occurs in cycles 2, 8 and 11.

Two interrupts are ALWAYS generated per period; one TC from the previous period (State A) to set up the width of this period, and one RE to set up the next period. Note that one RE interrupt is already represented by next State B.

The pulse width is updated at the very beginning of the ISR to support the highest possible frequencies. The update is marked with the [Update] command.

Transition A2 has one weakness as it can be seen in cycles 2 and 8. Whenever $W(1)$ is close to the PWM_MAXREWIDTH and, of course, over PWM_THRESHOLD, the ISR has little time to write a new pulse width to the compare register on TC interrupt at the beginning of cycles 2 and 8. This problem is represented in this case. It is likely that the pulse width in periods 2 and 8 will not be 220 but a little less. A higher-order predictor can help solve this case.

From this we can also find the maximum possible value of the PWM_THRESHOLD. This was partly described previously under “Changing the Pulse Width (n)” and shall be described in greater detail later.

Transition A3

This is an INVALID transition and is not shown in the figure. This transition could be supported by the first-order predictor but:

- 100% duty cycle cannot be achieved even if a predictor supports that feature.
- The transition can be supported by higher-order predictors only – which look at least two cycles forward.
- There are not many applications that require accurate transition from below PWM_THRESHOLD to 100% PWM duty cycle.

Transition B1

State B (RE) to 1 (TC) occurs whenever the pulse width of the next period is under PWM_THRESHOLD. The steps to shorten the width are represented by the following:

1. Shorten the pulse width to PWM_THRESHOLD only, referenced as the [ParUp] command.

2. Update pulse width to the desired value, referenced as the [SkipUp] command.

The first step is done in the RE interrupt generated by State B. The second step is done in the first TC interrupt marked as U in Figure 4. High frequency designs may combine the two steps into a single step to reduce interrupt count.

Transition B2

State B (RE) to 2 (RE) occurs whenever the pulse width remains in the range above PWM_THRESHOLD and below PWM_MAXREWIDTH.

It should be stressed that before shortening the pulse width register, an extra delay is required before the update, otherwise a glitch may occur. This is marked as the [Wait] command followed by the [Update] command. The delay must be estimated in advance and is described in greater detail ahead.

Transition B3

State B (RE) to 3 (TCRE) occurs whenever the pulse width $W(1)$ is to be set to its maximum. Transition is smooth and is represented in cycles 3, 8 and 11.

Transition C1

State C (TCRE) to 1 (TC) occurs whenever the pulse width changes from its maximum value to a value below PWM_THRESHOLD. The sequence to achieve accurate results is the same as described under Transition B1.

Transition C2

State C (TCRE) to 2 (RE) occurs whenever the pulse width changes from its maximum value to a value above PWM_THRESHOLD. The sequence is generally the same as Transition B2 except that an additional RE interrupt is generated.

The additional RE interrupt is skipped as referenced by the [SkipNxtRE] command.

Transition C3

State C (TCRE) to 3 (TCRE) occurs whenever the pulse width remains at its maximum value. The pulse width register does not need to be updated.

Changing the Interrupt Source

Changing the interrupt source is done by setting the proper bit in the PWM register. Whenever the interrupt source is changed from TC to RE and output is high or becomes high, the RE interrupt is **always** generated! This includes the transition from maximum pulse width, where the output is high and the RE interrupt source is selected.

RE-TC Threshold Parameter

Let us focus on the threshold that defines the border above which the RE interrupt source is selected and below or equal to the selected TC interrupt source. The RE interrupt sources are less welcome. They require additional CPU time, using delay loops when the pulse width is shortened. Let us define:

$$IRQ_CYCLES = (T_{irqup} + T_{irqlatency}) / dt$$

Equation 7

T_{irqup} is the maximum time required by the interrupt handler without delay loops when LESS comparison is chosen and:

$$PWM_MAXREWIDTH = N$$

Equation 8

…when LESS-EQUAL comparison is chosen.

Wait Function

The WAIT function is represented by the [Wait] command in Table 1q. It removes possible glitches that could occur when the pulse width is decreased. The WAIT function is called before the update to the compare register is done in RE and TCRE interrupts.

The $WAIT(dn)$ function declares the time to loop. It may be written as follows:

$$T_{waitloop} = (W(0) - W(1) - IRQ_CYCLES)dt$$

Equation 9

Usually smooth functions $func(i)$ have low first derivates, $W(0)$-$W(1) < dW$. In such cases, computation of $T_{wait\ loop}$ is too time consuming.

It is easier to declare constant $T_{wait\ loop}$ calculated or estimated for worst case using only dW to update the pulse width register. $T_{irq\ latency}$ is the maximum possible interrupt invocation latency and IRQ_CYCLES referenced to the PWM clock frequency and not to the CPU clock frequency. The value $T_{irqup} + T_{irq\ latency}$ is also equal to the t_{upre}, as described earlier.

Therefore, the PWM_THRESHOLD variable should be set as:

$$PWM_THRESHOLD = N - IRQ_CYCLES$$

Equation 10

Under these considerations, the interrupt handler CPU consumption is the lowest.

Maximum Pulse Width Parameter

This important parameter defines the most upper border when the RE interrupt cannot be used, as it is not generated if the output does not change.

$$PWM_MAXREWIDTH = N + 1$$ Equation 11

Example of Implementation

The associated project provides source code of the first-order implementation. ‘C’ language was used to describe the algorithm. In an effort to be clear and easy to understand, the code is not completely optimized.

There are two API functions; one to start the operation and a second to modify the pulse width from anywhere in the program. The core of the algorithm resides in the ISR gfpwm_update(). This ISR should have the highest interrupt priority. Lower priority increases maximum latency time, which influences PWM_THRESHOLD. Consequently, the ISR uses more CPU power due to the [Wait] command.

Applications that update pulse widths in every period should modify the gfpwm_w2 variable directly in the ISR gfpwm_update() as shown in the source code by reading the foo_table[0].

Example source generates the same stream as shown under the example pattern. The time consumed by the ISR is set as HIGH state on output pin P1[3]. Additional synchronization is provided on output pin P1[1]. It is set HIGH only when the first sample is fetched from the foo_table[0].

Summary

Applications rarely require an accurate, fast transition from 0% duty cycle to 99% or 100%; the first-order predictor cannot accurately handle such situations. Nevertheless, such functions as:

- Sine wave
- Triangular
- Saw-tooth (with positive gradient only)
- Other smooth functions

…are perfectly generated using the first-order predictor without a single glitch.

Implementation may differ from application to application. Sometimes a single interrupt is used for timer and PWM updates and some applications require very fast PWM updates. The last case requires good examination of the algorithm. Some parts of the code may be rearranged or changed, as for example; double update in states B1 or C1 may be replaced by a single update to reduce interrupts.

The algorithm described here is general purpose and handles all possible cases and states of the first-order predictor.

Document History

Document Title: PSoC® 1 Glitch Free PWM - AN2141

Document Number: 001-34540

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1505943	SWU	09/26/2007	New application note.
*A	3252272	OWEN	05/09/2011	Updated document title. Updated project and software version to PSoC® Designer™ 5.1. Updated template to current Cypress standards.
*B	4391945	KRIS	05/28/2014	Updated project and software version to PSoC® Designer™ 5.4. Updated in new template. Completing Sunset Review.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

PSoC is a registered trademark of Cypress Semiconductor Corp. "Programmable System-on-Chip," PSoC Designer, and PSoC Express are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2007-2014. 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.
This 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.