Sensing - Measuring an Input PWM

AN2103

Author: Steve Gerber
Associated Project: Yes
Associated Part Family: CY8C25xxx, CY8C26xxx
Software Version: NA
Associated Application Notes: None

Application Note Abstract

The Cypress MicroSystems PSoC® has the flexibility to measure input Pulse-Width Modulated (PWM) signals using a variety of methods. This Application Note details several measurement approaches and their associated trade-offs. Two example project files accompany this Application Note.

Introduction

PWMs are traditionally the output of a microcontroller. However, as microcontrollers evolve, they are being utilized in portions of systems once dominated by discrete analog circuitry. One of those applications is PWM input measurement.

When a microcontroller accepts a PWM input for digital processing, there are two important characteristics that must be considered. The first characteristic is that of measuring the cycle duration. The cycle duration is often a fixed value that may be 'assumed' given the characteristics of the PWM source. The second characteristic is the width of the pulse itself.

The analog filter and ADC approach typically mirrors that of traditional PWM motor control applications. In those applications, precision, accuracy, and speed are of little consequence. This method can be easily implemented in PSoC and is not the primary goal of this Application Note. Rather, this Application Note focuses on the digital counter capture implementation of a PWM input measurement. It is these implementations that often require precision, accuracy, speed-of-response, and creativity.

The PSoC device contains configurable resources that can accomplish either an analog or digital input PWM-measurement technique. Because PSoC applications are widely varied, there are situations when particular resources become scarce. PSoC may have millions of possible configurations; however, there still remains a finite set of resources. A typical design may very well push the limits of a given set of resources, such as: memory, clock speeds, digital PSoC Blocks, analog PSoC Blocks, processor overhead, etc. There are inherent flexibilities within PSoC to accomplish feats that at first glance appear to be significant technical hurdles.

These inherent flexibilities of the architecture have provided several methods of implementing the counter/timer measurement option depending on the precision, accuracy, and speed requirements. This Application Note addresses those flexibilities and provides design and project information to assist the reader through implementation.
Input PWM Measurement with an ADC

For those systems that do not require appreciable precision, accuracy, or speed, a sensible implementation for measuring a PWM input involves the use of a filter and an analog to digital conversion.

Both PWM characteristics can be approximated by conversion through an Analog to Digital Converter (ADC). The ADC provides a voltage output (representative of the source's pulse width) relative to the ADCs rail voltage (representative of the source's cycle duration). The resultant voltage represents a communicated value somewhere between completely off and completely on.

The PSoC device offers a variety of ADC User Modules providing variations in sampling speed, resolution, and system overhead. The ADC approach may also include external filtering circuitry or internal PSoC analog filtering resources. To reiterate, this approach is most appropriate for slowly responding systems and is not the main topic of this Application Note. Additionally, the filter and ADC approach consumes precious resources including analog/digital blocks, processor overhead and interrupt latency.

Input Pulse Width-Only Measurement with a Single, Digital PSoC Block

Some implementations of PWMs only require the measurement of the incoming pulse width. The cycle time is either known, or inconsequential to the input measurement. For those applications, a configuration has been created that simply requires a Counter User Module. Depending on clock rates and required resolution, the implementation may require one or more digital blocks within PSoC. The configuration also requires the Interrupt Service Routine (ISR) to monitor the counter in order to return a width value prior to the next incoming pulse. While this is a very simple implementation, it may not suit all needs. The GPIO Interrupt is a low priority interrupt; therefore, as a part of a system of interrupts, it may be difficult to accommodate ISR latency that coincides with the measurement requirements of the incoming PWM. This becomes a concern with fast PWMs and near 100% pulse widths.

The Pulsewidth1 project included with this Application Note demonstrates a pulse width measurement technique. It allows demonstration with only the PSoC Pup board and the In-Circuit Emulator (ICE). Included in this project is an internal PWM generator for subsequent measurement with an 8-bit counter. The internal PWM generates a signal by which to trigger the counter capture. This could easily be modified such that the PWM signal comes in on an external pin with a GPIO Interrupt on the falling edge of that pin. For demonstration purposes, this project uses the falling edge interrupt of the PWM to avoid the necessity of external PSoC connections. Figure 2 depicts the chosen PSoC block placement and connections of the Counter and PWM generator.

You may monitor the Pulsewidth value by setting a break on the following line:

```asm
mov [NewData], 0h
```

You will need to have placed the four global values in the watch window to easily view the results.

To exercise the overflow monitoring, you should change the PWM Period and Pulsewidth to 255 and put a breakpoint on:

```asm
mov [PulseWidth], FFh
```

The `main.asm` file is as follows (see associated project file for complete `main.asm` file):

```asm
include "m8c.inc"
include "Counter8.inc"
include "pwm8.inc"

export Overflow
export PulseWidth
export NewData
export PulseData

area bss(RAM)
PulseData:    BLK 1
PulseWidth:   BLK 1
Overflow:     BLK 1
NewData:      BLK 1

area text(ROM)
exptot _main

_mian:
    M8C_SetBank0
    mov  [NewData], 0h
    mov  [Overflow], 0h

    ; Start the Counter
    call PWM8_Start
    call Counter8_Start

    ; Enable the interrupts
    call PWM8_EnableInt
    call Counter8_EnableInt

    ; clear pending interrupts
    mov  reg[INT_VC], 0
    M8C_EnableGInt

    loop:
        ; Wait for a measurement
        wait:
            cmp [NewData], 0
            jz  wait

            ; Check for valid measurement
            cmp [Overflow], 0
            jnz  bad_data
```

January 22, 2003
good_data:
mov A, FFh
; calculate the pulse width
sub A, [PulseData]
; save the new value
mov [PulseWidth], A
; clear the result flag
mov [NewData], 0h
; go back and wait for more
jmp loop

bad_data:
; set result to max
mov [PulseWidth], FFh
; clear the overflow flag
mov [Overflow], 0h
; clear the result flag
mov [NewData], 0h
; go back and wait for more
jmp loop

export PWM8INT

PWM8INT:
push X
push A
; Stop the counter and read the value
call Counter8_Stop
call bCounter8_ReadCounter
mov [PulseData], A
; Now reset the counter to FFh
mov A, FFh
call Counter8_WritePeriod
call Counter8_Start
; Set the new measurement received
; flag
mov [NewData], 01h
pop A
pop X
reti

export Counter8INT

Counter8INT:
; set the TC flag
mov [Overflow], 01h
reti

Input PWM Measurement with Two Digital Resources

When digital resources are readily available, the DigInv User Module (Digital Inverter) may be used in conjunction with a capture timer to measure both the pulse width and the cycle duration simultaneously.

The pulse to be measured enters the digital inverter via a Global Bus. The output of the digital inverter may feed the timer via either the Previous PSoC block connection [or via the GlobalOut Bus if the necessary digital resource placement prevents utilizing the Previous connection]. At this point, you are probably questioning what value a simple logic inversion provides. The unique register-based architecture of PSoC provides the ability to dynamically switch the inputs to the PSoC blocks. With this flexibility, we can dynamically adjust whether the capture timer receives its input from the output of the DigInv block or from the input of the DigInv block. The Global Input Bus that feeds the DigInv block (the non-inverted input) is also available to the timer. Essentially, we can perform a logic inversion immediately following every PWM input’s rising edge. Correspondingly, we can provide a capture timer input that is not inverted after every falling edge. The input to the Capture Timer is essentially a set of ‘markers’ in time (Figure 3).

Figure 3. Pulse Width and Cycle Duration

The Timer then captures the count values at each marker. The top waveform of Figure 3 depicts the input to the Digital Inverter digital PSoC block and the bottom waveform represents the ‘switched’ input to the timer. Notice that the time markers are slightly delayed due to latency; however, the integrity of the pulse width and cycle-duration measurements remains intact. The PSoC block placement is indicated in Figure 4.
Input PWM Measurement with Analog Resources

When digital resources become scarce, your success demands creativity. The creative solution for simultaneously measuring pulse width and cycle duration with only one digital resource requires the addition of at least one analog resource. The Type A Switched Capacitor block in Figure 5 provides a tool toward an elegant solution.

Within the switched capacitor block is a register that allows for a dynamic inversion to occur (A.SIGN in Figure X). This register is CR0 bit 5 (ASign) as indicated in Table 1.

ASign controls the switched capacitor matrix in the following manner:

- **Pos** – A.Cap connected to the A Input during \( \phi_1 \) and the Ref Input during \( \phi_2 \)
- **Neg** – A.Cap connected to the Ref Input during \( \phi_1 \) and the A Input during \( \phi_2 \)
As with the Digital Inverter approach, when placed in front of a timer block, the SC block allows the simultaneous measurement of both pulse width and cycle duration with a single timer. This is accomplished by changing the value of ASign after every rising and falling edge on the PWM input. As with the DigInv approach, the output of the Dynamic Inversion converts the PWM input to a set of ‘markers’ in time. The Timer then captures the timer values at each rising edge of the markers (See Figure 3). In this approach, the top waveform of Figure 3 depicts the input to the SC analog block and the bottom waveform represents the modified input to the timer. Also in this approach, the input to the timer does not need to be switched as it was in the Digital Inverter approach. The time markers are slightly delayed due to latency; however, the integrity of the pulse width and cycle-duration measurements remains intact.

Figure 6 depicts the placement of four User Modules. The PWM8 block is the optional PWM input generator (created internal to the device for simplicity of demonstration). Timer8 is an 8-bit down counter with selectable clocking and programmable period. At terminal count the output signal is held high for one-half clock cycle. An external event captures the current Timer8 count value by asserting the edge-sensitive capture input signal. A programmable interrupt can be set to interrupt when the counter reaches its terminal count. This is particularly useful when monitoring the incoming pulse width for 0 or 100% pulse widths (overflow conditions).

Figure 6 includes a PGA block. When the CTBLOCK above the SCBLOCK is available, it can be used to pass the internal or external signal to the SCBLOCK. If the CTBLOCK is being used for other resources, it is possible to accomplish this measurement without this module. The architecture permits both ASB13 and ASB20 to bring a PWM input into the A mux directly from an outside pin (P2[2] and P2[1] respectively). It is also possible to accomplish this measurement using the ASA10 and ASA23 blocks; however, they require the B mux input, which complicates the measurement further.

The key to the second example project exists within the interrupt structure. Reference Pulsewidth3 project file for details.

**Design Note** The example project leaves timer overflow / wrap-around protection up to the designer to implement. Since the timer does not restart at every PWM-measurement result (it keeps counting), there may need to be control added to handle wrap around (especially in 16-bit timer applications) and overflow protection. Each design should completely eliminate timer and counter overflow problems. Lastly, protection must be included to properly measure 0 and 100% input PWM duty cycles.

**Summary**

Table 2 lists the variety of methods and considerations when measuring a PWM input to PSoC. The Dynamic Inversion implementation is a creative solution that does not compromise precious resources. Whether implemented with the DigInv or an analog SC block, the Dynamic Inversion technique can provide a powerful and conservative approach to PSoC design.
Table 2. Resource Comparison of Various Input PWM Measurement Techniques
(Assumes 8-Bit Implementation)

<table>
<thead>
<tr>
<th>Implementation</th>
<th># Digital Blocks</th>
<th># Analog Blocks</th>
<th>Overflow Handling</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter and ADC Measurement</td>
<td>1</td>
<td>1-3</td>
<td>Not Required</td>
<td>May require external or internal filter and more digital blocks. Slowly responding.</td>
</tr>
<tr>
<td>Pulse-Width Only Measurement with Counter</td>
<td>1</td>
<td>0</td>
<td>Required</td>
<td>Requires servicing of ISR, which may be precluded by other system resources. Does not allow measurement of PWM cycle duration.</td>
</tr>
<tr>
<td>Pulse Width and Cycle Duration Measurement Using SC Block</td>
<td>1</td>
<td>2</td>
<td>Required</td>
<td>May be accomplished with only one analog block depending on resource availability.</td>
</tr>
<tr>
<td>Pulse Width and Cycle Duration Measurement with Dig-Invert</td>
<td>2</td>
<td>0</td>
<td>Required</td>
<td></td>
</tr>
</tbody>
</table>

In March of 2007, Cypress recataloged all of its Application Notes using a new documentation number and revision code. This new documentation number and revision code (001-xxxxx, beginning with rev. **), located in the footer of the document, will be used in all subsequent revisions.

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 Corporation, 2003-2007. 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.