



# Application Note

# AN2266

## 16-Bit PWM/PWM-DACs using One Digital PSoC™ Block

**Author:** Brian Millier  
**Associated Project:** Yes  
**Associated Part Family:** CY8C27xxx  
**PSoC Designer Version:** 4.2  
**Associated Application Notes:** AN2199

### Abstract

Sometimes an 8-bit PWM does not provide you with enough resolution. The PWM16 User Module is an option, but it takes an extra digital block, and produces a PWM waveform of a much lower frequency. This Application Note describes an alternative that uses only one digital block, and includes some practical ways to use it to implement a high-resolution DAC.

### Introduction

A Pulse Width Modulator (PWM) is basically a circuit that repeatedly measures out a fixed period of time, and outputs a signal that is active for some percentage of that time (termed the duty cycle). That signal is generally used to provide power to some form of load, thereby providing it with a variable amount of power. By its nature, the PWM provides power in discrete pulses, so to be useful, the load itself must be capable of storing, or integrating much of that power, from one pulse to the next. This will generally limit the lower frequency limit at which a PWM circuit can operate, while still delivering useable power to the load. This depends upon your load, of course. For example, an electric stovetop range uses an electro-mechanical PWM controller with a period of about 10 seconds. The common lamp dimmer basically uses a 60 Hz PWM (actually a phase controller, but the effect is very similar). This Application Note describes ways to achieve PWM resolutions higher than 8-bits, at reasonably high PWM frequencies, with the added bonus of using only one digital block.

### PWM Theory

A pulse width modulator consists of an n-bit digital counter with additional circuitry to signal when its count equals or exceeds some user-defined count (which must be less than  $2^N - 1$ ). It is this latter signal, which controls the flow of power to the load. This counter is clocked by a separate clock generator. We will define this clock frequency  $F$ .

The relationship between the PWM's operating frequency, its resolution, and the clock frequency  $F$ , is defined as:

$$\text{PWM frequency} = \frac{F}{2^N} \quad (1)$$

$N$  is the number of stages in the PWM counter chain and is also the resolution of the PWM in bits.

Used as a PWM, the PSoC™ device's digital blocks can be clocked using any of the available internal clock sources, from SysClk\*2 (nominally 48 MHz) down to 366 Hz (using VC3 and the maximum divisor rates for VC3, 2 and 1). Therefore, from Equation (1), it can be seen that an 8-bit PWM User Module can achieve an operating frequency up to 187,500 Hz. This is plenty fast for most applications.

Things start to become less-than-ideal, when we require a much higher resolution. For example, if we wanted a 14-bit resolution PWM, the operating frequency would now be reduced to 2930 Hz. For heating and incandescent lighting loads, this would still be adequate. However, allow us to assume that we want to use this PWM signal to implement a DAC. In such a case, we are generally looking to provide a smooth DC voltage proportional to the duty cycle of the PWM. To achieve this, we must follow the PWM's output signal with a low-pass filter to smooth out the pulses. This can generally be in the form of a simple RC filter.

However, the time constant of the RC filter must be hundreds of times greater than the operating period of the PWM, to properly smooth out the pulses into a stable voltage. Exactly how long this time constant must be, depends upon how much ripple you can tolerate in the output voltage signal. You may be able to tolerate more ripple than you think, if you are measuring this signal later on using some form of integrating ADC.

Nevertheless, the 2930 Hz 14-bit PWM mentioned earlier will generally have its response time slowed down to less than 100 Hz, if followed by an appropriate RC filter, and used as a DAC. This response time may be too low for your application.

In discussing PWM controllers to this point, I have only mentioned that the power that they deliver to the load is proportional to the PWM's duty cycle. Because PWMs merely switch the power to the load, the voltage stability of this power source also contributes heavily to the stability of the power delivered to the load. Actually, the load power is proportional to the square of this supply voltage, so its stability is even more important.

In some cases involving high-resolution PWM, the absolute accuracy of this power supply is not too important. For example, a high-resolution PWM circuit might supply power to a DC motor, with a tachometer signal used in a feedback loop to accurately control motor speed. Here the high resolution of the PWM is necessary, but the absolute accuracy of the motor power supply voltage is not so important, as a feedback loop will ultimately act to control the speed, which is the critical parameter.

However, when implementing a PWM-based DAC, the accuracy of the DAC itself is commensurate with the absolute accuracy of the reference power supply. Therefore, for a higher resolution PWM-based DAC, one cannot simply use the PSoC's PWM block output signal, after RC filtering, to provide the DAC signal. The reason for this is that the PSoC device's PWM block simply provides a signal that switches between 0 and  $V_{dd}$ . The stability of the  $V_{dd}$  supply will not be sufficient, and it will likely contain digital switching noise as well.

To overcome this, Application Note AN2199 – Standard – “DAC with Analog Modulator” outlines a high-resolution PWM DAC solution that makes use of the PSoC device's analog modulator block. In this implementation, the PWM block acts as the switching source for an analog modulator.

The analog modulator is fed by the internal voltage reference present in the PSoC device; thereby achieving much better accuracy than what can be achieved using only the PWM block. Unfortunately, while that Application Note implies that one can achieve high PWM frequencies by using the SysClk\*2 signal for the PWM clock, in reality, the analog modulator cannot switch any faster than about 250 KHz. Therefore, using a PWM16 block clocked at 250 KHz, you are only going to end up with a PWM frequency of 3.8 Hz.

## An Alternate Form of PWM

From Equation (1) in the earlier section, it seems that you can have high speed or high resolution, but not both, unless you are prepared to clock the PWM at a very high rate. This is fine for the PSoC device's digital blocks, but is not compatible with the PSoC device's analog blocks, thus limiting the usefulness of the scheme outlined in Application Note AN2199.

As an alternate way of extending resolution, consider the following. Imagine that your 8-bit PWM block has the ability to increase its user-specified duty cycle value by 1, under certain conditions. To simplify the explanation, allow us to assume that every time the PWM block overflows (beyond count 255), it toggles the user-specified duty-cycle value (M) between M and M+1. Averaged over time, this will result in an effective duty cycle of  $M + \frac{1}{2}$ . Already we have achieved 9-bit resolution from the PWM8 block. Carrying this one step further, allows us to assume that we maintain an 8-bit variable “Count” and another one called “DutyCycleLSB.” Then we set up the PWM8 block to provide an interrupt when it overflows.

In the interrupt service routine (ISR) for the PWM8, we can implement a short routine. Its flowchart is shown in Figure 1. The net effect of this routine is that for any specific duty cycle that we set up on the PWM8 block, there are 256 small increments between this value and the next consecutive value that we can achieve by merely setting the value of variable “DutyCycleLSB” accordingly. This achieves the equivalent of a 16-bit PWM using only one PWM8 block. To make implementation of this algorithm a bit simpler, you will note that the PSoC device's PWM block has a parameter labeled **Compare Type** which can be set for either “Less than” or “Less than or Equal to.” By simply changing the value of one bit in one of the PWM's control registers, you can switch between these two comparison types, which is equivalent to changing the PWM duty cycle value between any value M and its successor, M+1.

This is how the PWM ISR actually implements the change from M to M+1, and back again, when necessary. Actual ISR code is shown in Code 1.

If you analyze this operation at all, you will quickly see that the MSB of the 16-bit duty-cycle value is being updated at the effective PWM rate of the PWM8 block itself, which is PWM clock/256. The LSB is being updated at 1/256th of this rate. In fact, for those cases where the MSB is equal to zero, the output of this PWM is the same as what you would get if you used a PWM16 User Module instead of the PWM8. However, for the vast majority of the values that are not so close to zero, the PWM waveform that this circuit provides requires a much lower time-constant RC filter to achieve a given ripple, than does the straight PWM16.

The net result is that you can achieve a high resolution PWM signal using only one digital block, as well as require a lower PWM clock rate. This in turn makes the use of the PSoC device's analog modulator a possibility. In exchange for this, you do incur some MCU overhead servicing the PWM8 ISR. You have to keep the PWM8's clock rate low enough, so that the ISR routine can execute in less time than it takes for the PWM block to advance by 1 count. For example, if you clock the PWM8 block at a SysClk/64 rate, (assuming that you are running the M8C core at the SysClk rate), you have 64 cycles to perform the ISR routine shown in Code 1, which is plenty of time.

The technique just described is not original. Similar techniques have been used in PLL "pulse-swallowing" counters for many years. In this context though, I was reminded of the idea by Victor Kremin, who implemented it in hardware using several PSoC device blocks, rather than using an ISR, as I have done.

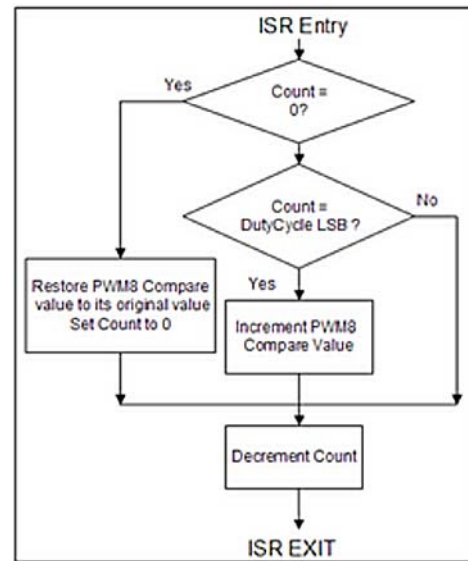


Figure 1. ISR Flowchart

```

_PWM8_1_ISR:

;@PSoC_UserCode_BODY@ (Do not change this
line.)
;-----
; Insert your custom code below this banner
;-----
; NOTE: interrupt service routines must
preserve
; the values of the A and X CPU registers.
push A
mov A,[PWM8_1_Count]
cmp A,0
jnz Case2
M8C_SetBank1
or REG[PWM8_1_FUNC_REG], 10h
; compare type = LT
M8C_SetBank0
mov [PWM8_1_Count],0
jmp Ex
Case2:
cmp A,[PWM8_1_DutyCycle]
jnz Ex
M8C_SetBank1
and REG[PWM8_1_FUNC_REG], EFh
;compare type = LT or EQ
M8C_SetBank0
Ex:
dec [PWM8_1_Count]
pop A
reti
;-----
; Insert your custom code above this banner
;-----
;@PSoC_UserCode_END@ (Do not change this
line.)

reti

```

Code 1. PWM8 ISR Code to Implement 16-Bit PWM

## A Practical Example

Allow us to go through the steps necessary to implement such a PWM circuit in a PSoC Designer project. Place a PWM8 as shown in Figure 2. I have also placed a UART in my project, to enable you to experiment with the circuit, by entering various PWM values using the serial port, but this is optional. Set the PWM8 parameters as shown in Figure 3, noting in particular that **Interrupt Type** is set to Terminal Count.

Merely setting this parameter does not enable interrupts from the PWM; that must be done in the application code. But it does tell the Application Generator to include a “stub” module called *PWM8\_1int.asm* (found in the Library Source folder), which is all set up and ready for you to manually insert the ISR routine shown in Code 1. The digital output of the PWM8 signal is routed to P0[1], where it can be observed on a scope as a 5V CMOS level signal. Note that it is an inverted signal, as the RO0[1] bus is routed to the Global OutEven Bus via an inverter in the LUT. This is done to produce the proper analog signal output polarity from the DAC\_Modulator block.

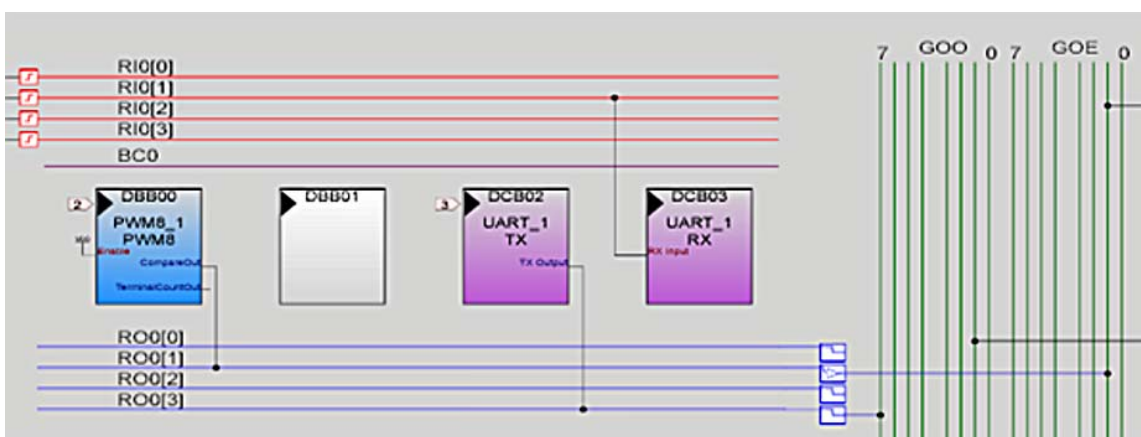


Figure 2. Digital Block Placement

Next, place an SC block (re-named DAC\_Modulator for clarity) as shown in Figure 4, setting its parameters as shown in Figure 5. See Application Note AN2199 for a full explanation of the function of the analog modulator. The DAC\_Modulator’s output is routed to P0[3], where it can be observed on the scope as an analog PWM signal: its ON state equal to  $[Bandgap*2 + Bandgap]$  and its OFF state equal to  $[Bandgap*2 - Bandgap]$ . Expressed differently, it is an analog signal with a range  $\pm Bandgap$ , riding on an offset of  $[2*Bandgap]$ . The PSoC device’s internal Bandgap reference is nominally 1.3 volts.

For convenience, I have placed a RefMux User Module at ACB01, and set it to AGND. This is routed to P0[5], where it provides a level equal to the DAC\_Modulator’s offset:  $[2*Bandgap]$ . Therefore, the DAC voltage exists between P0[3] and P0[5], and rides on the accurate internal Bandgap reference, free of ground noise. All that remains is to place an RC filter between these two terminals to obtain the filtered DAC voltage, referenced to  $2*Bandgap$ .

The initialization code fragment for this project is shown in Code 2. Apart from that, you must manually enter the ISR code in Code 1 into the *PWM8\_1int.asm* at the position shown in the “stub” file created by the Application Generator. To set a 16-bit PWM value, call the PWM8\_1WritePulseWidth with the MSB, and load variable “PWM8\_1\_DutyCycle” with the LSB.

User Module Parameters	
Clock	VC2
Enable	High
CompareOut	Row_0_Output_1
TerminalCountOut	?
Period	255
PulseWidth	25
CompareType	Less Than
InterruptType	Terminal Count
ClockSync	Sync to SysClk
InvertEnable	Normal

Figure 3. PWM8 Configuration

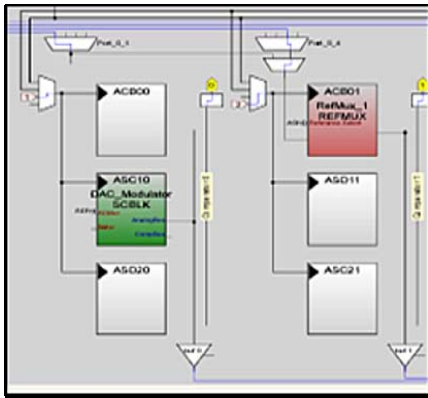


Figure 4. Analog Block Placement

User Module Parameters	
FCap	16
ClockPhase	Norm
ASign	Pos
ACap	16
ACMux	REFHI
BCap	0
AnalogBus	AnalogOutBus_0
CompBus	Disable
AutoZero	On
CCap	0
ARefMux	AGND
FSW1	On
FSW0	On
BMux	?
Power	High

Figure 5. DAC\_Modulator Parameters

Global Resources	
CPU_Clock	24_MHz (SysClk/1)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	16
VC2= VC1/N	4
VC3 Source	SysClk/1
VC3 Divider	52
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref Low
Ref Mux	(2 BandGap)+/BandGap
AGndBypass	Disable
OpAmp Bias	Low
A_Buff_Power	Low
SwitchModePump	OFF
Trip Voltage [LVD (SMP)]	4.64V (5.00V)
LVDThrottleBack	Disable
Supply Voltage	5.0V
Watchdog Enable	Disable

Figure 6. Global Resource Parameters

The Project included with this note is written in assembly language, and also contains routines to allow the user to load PWM values via the UART (at 57,600 Baud).

Call this routine to setup and start the 14-bit PWM DAC circuit:

```

init:
    mov A,128
    mov [PWM8_1_DutyCycle],A ; Set LSB
of PWM duty cycle to 128 ( just an example)
    call PWM8_1_Start
    call PWM8_1_EnableInt
    mov A,40
    call PWM8_1_WritePulseWidth ; Set MSB
of PWM duty cycle to 40 ( just an example)
    MOV A,3
    call RefMux_1_Start
    ; Start RefMux at high power
    mov A,DAC_Modulator_HIGHPOWER
    call DAC_Modulator_Start
    M8C_SetBank1
    mov A,0x01
    mov REG[AMD_CR0],A ; set
DAC_Modulator switching source to
Global_Out_Even1
    M8C_SetBank0
    ; enable interrupts
    M8C_EnableGInt

```

**Code 2. Set up and Start the 14-bit PWM DAC circuit**

Figure 7 represents a PWM value of 0x0380, (halfway between 0x0300 and 0x0400) so the width of the pulse alternates at a 50% duty cycle between 9.5 and 13 units on the scope.

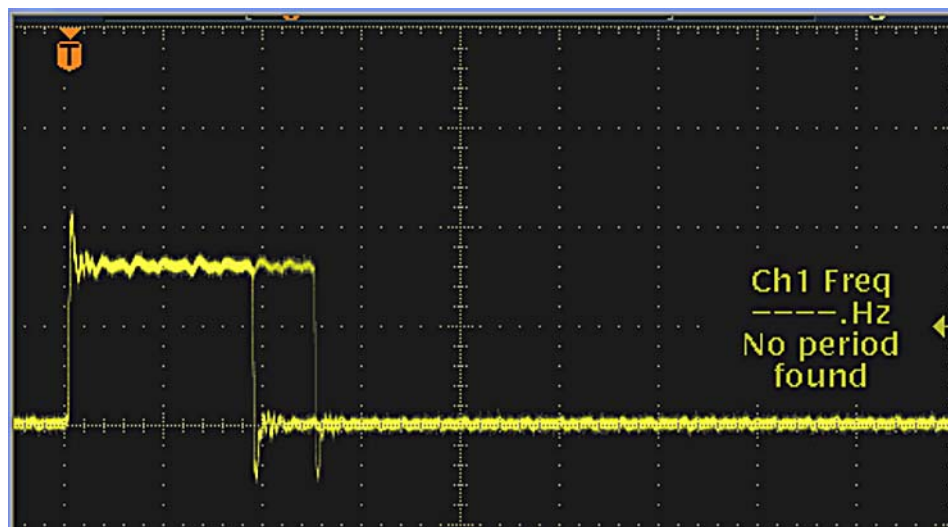


Figure 7. Scope Trace of PWM Output

## About the Author

**Name:** Brian Millier

**Title:** Technical Manager

**Background:** Brian has 23 years designing custom instrumentation in the Chemistry Dept. of Dalhousie University, and is a regular writer for Circuit Cellar Magazine.

**Contact:** [Brian.millier@dal.ca](mailto:Brian.millier@dal.ca)

Cypress Semiconductor  
2700 162<sup>nd</sup> Street SW, Building D  
Lynnwood, WA 98037  
Phone: 800.669.0557  
Fax: 425.787.4641

<http://www.cypress.com/>

Copyright © 2005 Cypress Semiconductor Corporation. All rights reserved.  
PSoC™, Programmable System-on-Chip™, and PSoC Designer™ are PSoC-related trademarks of Cypress.  
All other trademarks or registered trademarks referenced herein are the property of their respective owners.  
The information contained herein is subject to change without notice. Made in the U.S.A.