AN2266

PSoC® 1 - 16-Bit PWM/PWM-DACs using One Digital PSoC® Block

Author: Brian Millier
Associated Project: Yes
Associated Part Family: CY8C27xxx
Software Version: PSoC® Designer™ 5.4
Related Application Notes: None

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. AN2266 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 s. 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 PWM 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:

$$\textbf{PWM frequency} = \frac{F}{2^N} \qquad \text{Equation 1}$$

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 Analog DAC with Analog Modulator – AN2199 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.

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 + ½. 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 are equivalent to changing the PWM duty cycle value between any value M and its successor, M+1.

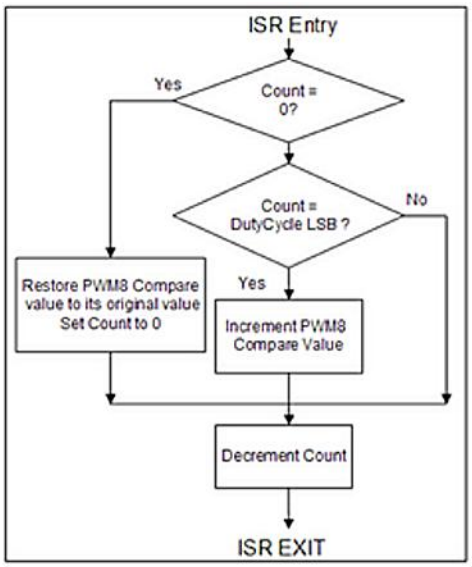
This is how the PWM ISR actually implements the change from M to M+1, and back again, when necessary. Actual ISR code is shown in Code 1.

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



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

```
_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.)

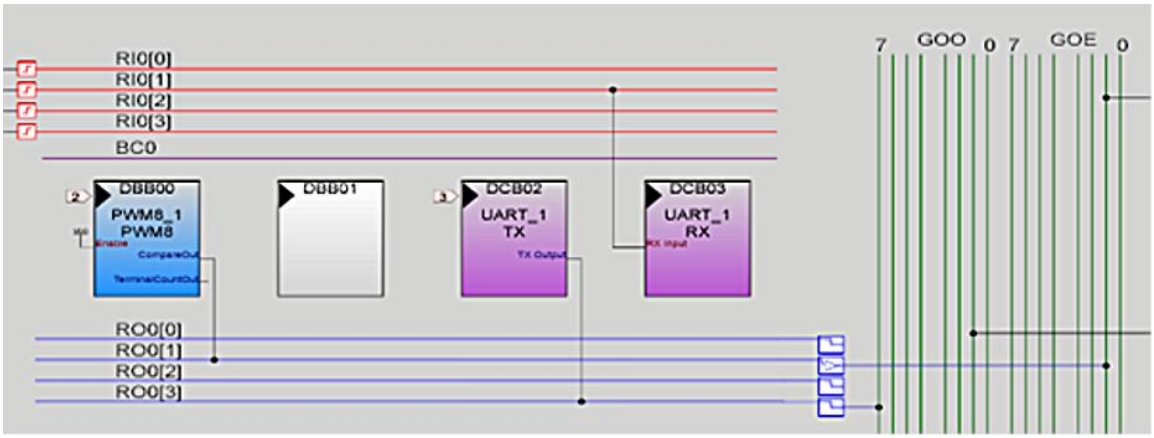
   reti
```

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 5 V 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 lookup table (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 +/- Bandgap, riding on an offset of [2*Bandgap]. The PSoC device's internal Bandgap reference is nominally 1.3 V.

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.

Figure 3. PWM8 Configuration

PWM8_1

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 4. Analog Block Placement

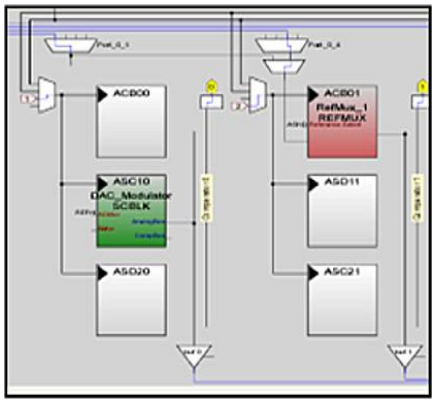


Figure 5. DAC_Modulator Parameters

DAC_Modulator

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 6. Global Resource 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
Op-Amp 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

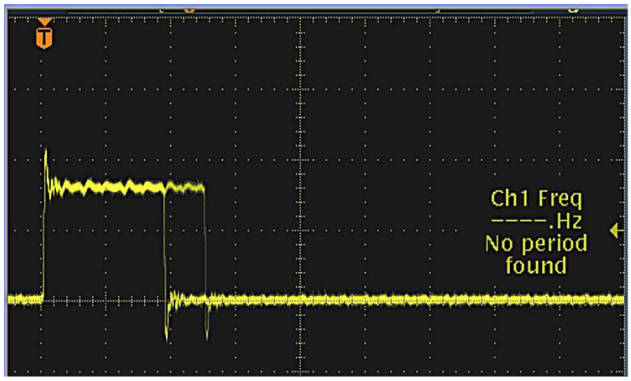
The Project included with this note is written in assembly language. The project starts by initializing the different user modules and starting the PWM code in the init routine as shown below.

Code 2. Set up 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
```

The project contains routines to allow the user to load PWM values via the UART (at 57,600 Baud). The user can update the duty cycle by first sending character ‘P’ to the PSoC1 through the UART interface. Once the character ‘P’ is recognized the PSoC will wait for the 2 byte duty cycle value. This is taken care of by the routine SetFreq, 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



Summary

The application note explains how you can build a 16-bit PWM from single PSoC Digital block. It also explains a practical method to implement a high resolution DAC using the block. This enables the user to build a high resolution DAC in his design with just one digital block, one switched capacitor block and a continuous time block.

Document History

Document Title: AN2266 – PSoC® 1 – 16-Bit PWM/PWM-DACs using One Digital PSoC® Block

Document Number: 001-32405

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1491203	QVS	09/20/2007	OLD APP. NOTE: Obtain spec. # for note to be added to spec. system. Update copyright. Add source disclaimer, revision disclaimer, Samples Request Form link, PSoC App. Note Index link. .pdf has been stamped. **This note had no technical updates. There is an associated project but it was not updated.**
*A	3175061	QVS	02/22/2011	Updated title to read “16-Bit PWM/PWM-DACs using One Digital PSoC Block”. Updated Software Version as PSoC® Designer™ 5.0 or Newer. Updated the section A Practical Example.
*B	3277483	QVS	06/08/2011	No technical updates.
*C	4422515	MSUR	06/27/2014	Updated to new template. Completing Sunset Review.
*D	4622480	ASRI	01/13/2015	Updated Software Version as “PSoC® Designer™ 5.4” in page 1. Updated attached associated project to PSoC Designer 5.4.
*E	5687753	AESATMP8	04/19/2017	Updated logo and Copyright.

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

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

© Cypress Semiconductor Corporation, 2007-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.