

## *Three-Phase Sine Wave Generator*

By: Uroš Platiše

**Associated Project:** Yes

**Associated Part Family:** CY8C25xxx, CY8C26xxx, CY8C27xxx

**PSoC Designer Version:** 4.0

**Associated Application Notes:** AN2141

### Summary

This Application Note provides and describes an implementation of the symmetric and glitch-free, three-phase sine generator using three 8-bit PWM User Modules. This generator may be used to drive three-phase inverters and three-phase electric drives, such as, for example, the popular AC and PMSM drives.

### Introduction

The need for a three-phase sine wave increases with the popularization of three-phase electric drives. Usually, stronger DSP machines play a part in such applications but in this Application Note, it is shown that PSoC, with its unique structure, can generate such three-phase sine waves, which then directly drive electric drives and invertors.

This note focuses on generation of the three-phase sine waves with the same characteristics as one generated by the special Motor PWM blocks in DSP machines. Detailed information is provided on stability issues, glitch-free operation and symmetric sine function computation requirements and limitations.

### Key Features

- Three-Phase Sine Wave Generator
- Symmetric PWM Outputs
- Glitch Free
- PWM Resolution from 8 to 16 bit
- PWM frequency above 20 kHz
- Direct drive of the three-phase, full-bridge power-section when using the PWM8DB blocks
- Lowest-Cost, MCU-based Three-Phase Generator

### Applications

- Single-Phase Inverters with Symmetric PWM Outputs such as UPS
- Three-Phase Industry Inverters
- AC Motor Drives with the V/Hz control
- Permanent Magnet Synchronous Motor (PMSM) Drives
- Power Factor Correction (PFC)
- Robotics and Industrial Control

## Design Structure

The three-phase design incorporates three PWM User Modules, which may be either 8 or 16 bit. To save PSoC digital resources, only the 8-bit PWM modules are described. However, the theory may be applied to 16-bit PWM modules without additional considerations. Additionally, 8-bit PWM modules with dead-time units may be used to directly interface to the inverter's power section.

## Building Blocks

The 8-bit PWM block is known as PWM8 User Module and is described in detail in its data sheet. Here, we review the most important features of the three-phase design.

The PWM8 User Module consists of three main registers:

- Down counter (CNT). Value is expressed with the  $c(i)$  symbol.
- Period register (PERIOD). Value is expressed with the  $p(i)$  symbol and actual PWM period with the  $P$  symbol.
- Compare register (COMPARE). Value is expressed with the  $w(i)$  symbol.

$i$  represents  $i$ -th PWM cycle. When  $i$  is not given, it means an arbitrary PWM cycle.

The down counter (CNT) counts down continuously to zero from the value specified in the PERIOD register, where it reloads the CNT register with the value stored in the PERIOD register again and again.

Meanwhile, the value of the down counter is compared with the compare value stored in the COMPARE register. When the value of the CNT is less-than or less-than-or-equal-to the value stored in the COMPARE register, the PWM output is switched into HIGH state. Otherwise, it is in LOW state.

This three-phase design recommends using the less-than comparison and that the PERIOD register have an odd number less than 255. When these two conditions are met, full symmetry between the COMPARE value and output pulse width may be obtained. Thus, PWM output may remain in:

- HIGH state when  $w > p$  (i.e.  $w=254$ ,  $p=253$ )
- LOW state when  $w=0$

The mean value is equal to  $(p+1)/2$ . The duty cycle is equal to:

$$d = w / (P+1) \quad (1)$$

The COMPARE register actually holds the pulse width value.

PWM frequency  $f_{\text{pwm}}$  is equal to:

$$f_{\text{pwm}} = f_{\text{clk}} / (P+1) \quad (2)$$

$f_{\text{clk}}$  is the system clock for all three PWM modules, which has direct influence on CPU usage, described later.

Figure 1 illustrates a PWM with the two aforementioned conditions met. It has the following parameters:

- $P=3$
- $w$  has values 0,1,2 and 4
- System clock  $f_{\text{clk}} = 100$  kHz.

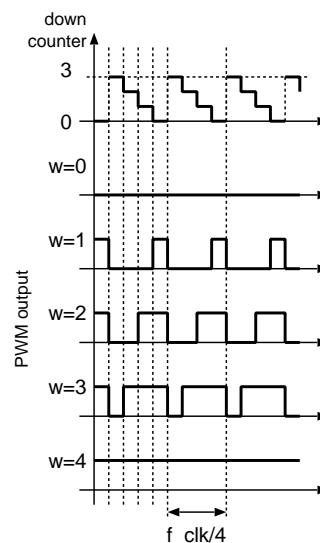


Figure 1. PWM

The symmetry is clearly seen in this figure. Note that the symmetry plays an important role in the power section to avoid unwanted saturations in ferromagnetic cores.

## Three-Phase Composition, Requirements and Synchronization

Composition of the three-phase design is shown in Figure 2.

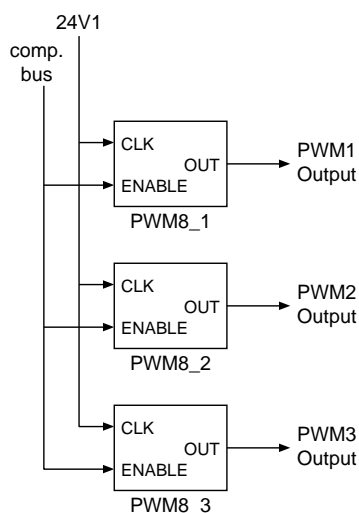


Figure 2. Phase Design Composition

The design requires three PWM modules to be placed in adjacent digital blocks with the highest interrupt priorities. All of them must use the same system clock  $f_{clk}$ , such as 24V1, 24V2 or other. PWM8 ENABLE input of all modules must be tied to the same controllable source, which may either be a compare bus, global input or a broadcast signal. This source is needed during start-up synchronization in order to correctly position the centers of all three PWM outputs. PWM8 blocks must be set to trigger interrupt on Terminal Count and comparison is set to Less-Than.

### Symmetric Sine Wave

The PWM8 and PWM16 blocks are not able to generate center-based PWM outputs as shown below, but are aligned to the right-edge.

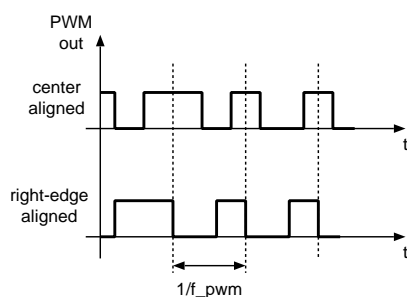


Figure 3. PWM Outputs

Symmetric PWM output, in comparison to the edge-aligned, does not generate even harmonics at the output and is more suitable for inverters.

The latest control algorithms use a double-update technique to add asymmetry by setting different times for the left and right edge of the PWM output, according to the center. With little modification of the algorithm presented in this note, such signals may be generated.

Symmetric sine wave can be generated with the PWM8 module by simultaneous modification of the PERIOD register and the COMPARE register. Changing the value of the PERIOD register will effectively shift the center of the PWM pulse to the left, when decreased, or to the right, when increased, on the time axis.

First, we shall discuss the idea of center-based PWM signals and then additional considerations are given for use of the PWM8 module.

### Center-Based PWM

From Figure 4, Equation (3) for any arbitrary  $p(i)$  may be derived.

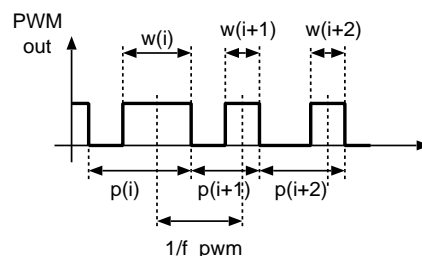


Figure 4.

$$p(i+1) = P + (w(i+1) - w(i)) / 2 \quad (3)$$

$w(i+1)$  and  $w(i)$  are pulse widths of the  $i$ -th and  $(i+1)$ -th PWM cycles,  $p(i+1)$  is the new period for the  $(i+1)$ -th PWM cycle, and  $P$  is the actual PWM period. We see that the period value for the  $i$ -th PWM pulse is determined from two adjacent pulse widths.

Equation (3) shows the following important properties of this algorithm:

The values of the pulse widths  $w(i)$  must be even. Otherwise, error might occur when dividing by two. It is possible to implement such an algorithm where  $w(i)$  may be of any value, but such a signal would no longer be symmetric.

$p(i)$  must always stay under 256 and above 0 when the PWM8 block is used. Compliance derives the maximum allowable change of pulse width and consequently, the maximum dynamic range of the generator:

$$0 < [P + (w(i) - w(i+1)) / 2] < 256 \quad (4)$$

Equation (4) is described in the “Period Range” section of this note.

The same equation also implies the maximum allowable execution time ( $t_{\text{irqexec\_max}}$ ) of the PWM interrupt, since periods must be updated continuously, one after another. The time equals...

$$t_{\text{irqexec\_max}} = (p(i) + 1) / f_{\text{clk}} \quad (5)$$

for each period  $p(i)$ . The time further equals:

$$t_{\text{irqexec\_max}} = t_{\text{irq\_latency}} + t_{\text{irqexec}} \quad (6)$$

$t_{\text{irq\_latency}}$  is the maximum system interrupt latency time and  $t_{\text{irqexec}}$  is the time needed to execute the PWM interrupt.

Note that when the pulse width changes, the PERIOD register is updated two times. To illustrate this we set:

$$\begin{aligned} P &= 59 \\ w(i) &= 30; i < 0 \\ w(0) &= 32 \\ w(i) &= 32; i > 0 \end{aligned}$$

Substituting the values in Equation (3), we get:

$$\begin{aligned} p(-1) &= 59 + (30 - 30)/2 = 59 \\ p(0) &= 59 + (32 - 30)/2 = 60 \\ p(1) &= 59 + (30 - 32)/2 = 58 \\ p(2) &= 59 + (30 - 30)/2 = 59 \end{aligned}$$

The pulse width is updated on the zero cycle only when  $i=0$ , but the period is updated on the zero and the first PWM cycles.

### Sine-Wave Generation

The width of the PWM pulse may be modulated by the sine function as:

$$w(i) = (P+1) (1 + A(i) \sin[f(i)]) / 2 \quad (7)$$

Now expressed with a duty cycle:

$$d(i) = (1 + A(i) \sin[f(i)]) / 2 \quad (8)$$

$A(i)$  is the current amplitude in the range  $[0, 1]$  and  $f(i)$  is the current phase of the sine function. Note that the neutral value, when  $A(i)=0$  or  $f(i)=n\pi$ , has a duty cycle of 50%, maximum value 100%, and minimum value 0%. To increase performance and avoid extensive computation of the sine function, at least  $\frac{1}{4}$  of the sine function is given as a look-up table.

### PWM8 Operation

When the new value is written into the PERIOD register, it takes effect after the down counter reaches zero and reloads from the PERIOD register. When the new value is written into the COMPARE register, it takes effect immediately.

The PERIOD and COMPARE registers must be updated in such a way that no glitch occurs at the output, and output PWM frequency is fixed (actual period equals  $P$ ), providing no deviation from the specified value  $f_{\text{pwm}}$ . This last property is especially important when three-phase sine waves are generated, where all three PWM centers must stay aligned forever. That is why the last property represents general Stability Criteria of the sine-wave generator and the first property represents Glitch-Free Criteria.

Single center-based sine wave output may be obtained using the Glitch Free Generator described in Application Note AN2141, with a little modification. The PWM8 interrupt must update the PERIOD register in addition to the COMPARE register.

The Stability Criteria is derived by the  $t_{\text{irqexecmax}}$  parameter given in Equation (5) and the PERIOD( $i$ ) range defined in Equation (4). The Glitch-Free Criteria conforms to the Glitch-Free PWM as described in AN2141.

### Three-Phase Sine Wave Generation

The beauty of the three-phase system as proposed by Nikola Tesla is that everything is simplified. It is also true in our case, where three-phase sine wave, glitch-free implementation is less complicated than the single sine wave, glitch-free implementation described above.

Implementation of the three-phase, sine generator uses three PWM8 modules and a single Interrupt Service Routine (ISR) to handle interrupts from all three modules. All PWM8 modules must be set to trigger on Terminal Count (TC).

The key idea is to sort interrupts from all three blocks in such a way that the best interrupt is used to update all PWM PERIOD registers and the PWM widths (COMPARE registers).

### Symbols and Definitions

Let us first redefine some symbols. PWM width  $w$ , period  $p$ , duty cycle  $d$  and phase  $f$  are modified into  $w_{-1}$ ,  $p_{-1}$ ,  $d_{-1}$  and  $f_{-1}$  for the PWM1, PWM2, PWM3, and so on.

Arbitrary, even periodic, functions may be used to modulate the pulse width, but here we shall limit it to the sine function. Therefore, pulse widths  $w_1(i)$ ,  $w_2(i)$  and  $w_3(i)$  must be calculated using Equation (7), shifting phases by 120 degrees:

$$f_2(i) = f_1(i) + 2\pi/3$$

$$f_3(i) = f_1(i) + 4\pi/3$$

All three PWMs must be center-aligned and their effective period must remain the same, equal to the P. Note that the mean period is always considered the period of the PWM cycle and not the modulated sine wave.

### Interrupt Sorting and PWM Updating

Interrupt vectors from all PWM blocks should point to a single ISR called `tfpwm_isr()`. The ISR counts interrupts. Within each PWM cycle, three interrupts must be generated. If they are not, the system is no longer stable because it fails to meet the Stability Criteria, which is described ahead. That is why it is important for all three PWM modules to have the highest interrupt priorities.

Interrupts sort automatically, due to the nature of the three-phase system, where one is always the first, one is always the second, and one is always the third. No other options are possible. All three PWM widths are updated when the first interrupt is triggered. The PWM periods are updated on the last interrupt. The last interrupt is also used to latch new widths and periods of all three PWM8 modules. It also provides a unit of delay for the pulse widths, which would otherwise be updated one cycle in advance of the period updates.

The system is stable when it meets the Stability Criteria. The output is glitch-free when it meets the Glitch-Free Criteria. Both criteria limit the dynamic range of the modulated sine function.

### Arrangement of the TC Interrupts

The first TC interrupt is triggered by the PWM8 module, which has the shortest PWM width and the last TC interrupt by the PWM8, which has the largest PWM width.

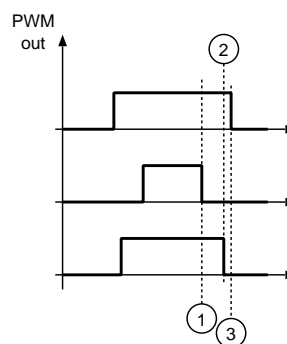


Figure 5.

So we may write:

$$w_m \leq w_n \leq w_l \quad (9)$$

$m$ ,  $n$  and  $l$  represent the first, second and third TC interrupts. For arbitrary  $m$ ,  $n$  and  $l$ , two widths may be the same but the third may not:  $w_m = w_n \neq w_l$ .

Arrangement of the Terminal Count interrupts in a center-based PWM system is shown in the following figure:

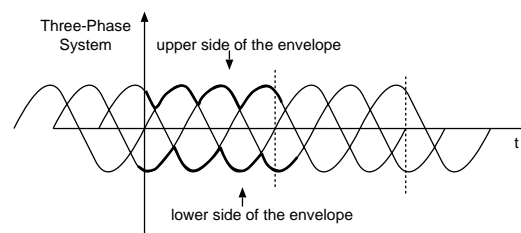


Figure 6.

The first TC interrupt,  $w_m$ , follows the lower side of the envelope of the three-phase system. The last (third) TC interrupt,  $w_l$ , follows the upper side of the envelope of the three-phase system. The second TC interrupt is always triggered between the first and last and is represented by the middle sine between the upper and lower envelopes.

### Pulse Width Update

The first TC interrupt is triggered by the PWM, which has the shortest pulse width. Since the COMPARE registers are not latched or synchronized in hardware, it is necessary to take extra care when these registers are updated.

The following figure defines the setup time.

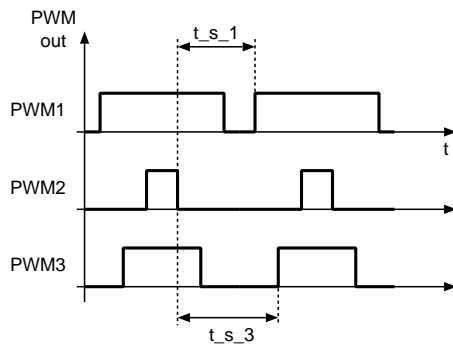


Figure 7.

The first TC interrupt provides the longest setup time to all three PWM output rising edges and, as such, represents an ideal source for updating the PWM COMPARE registers. Note that the new pulse width written in the  $i$ -th cycle will take effect in the next cycle,  $i+1$ .

On the other hand, it does require a longer hold period before the COMPARE register can be written with the new pulse width value. The hold time period is defined in the next figure:

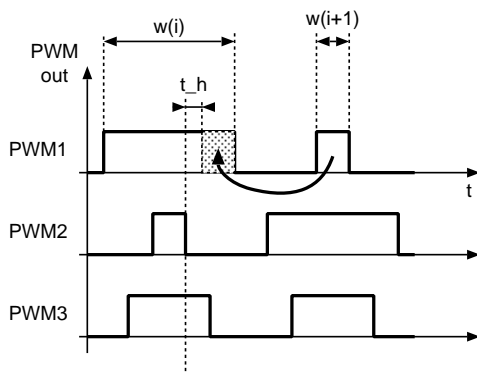


Figure 8.

For example, if one's COMPARE register is updated with a very short width before its PWM output signal actually switched into the LOW state, the output will generate a glitch at the end of the PWM cycle. (See Figure 9.)

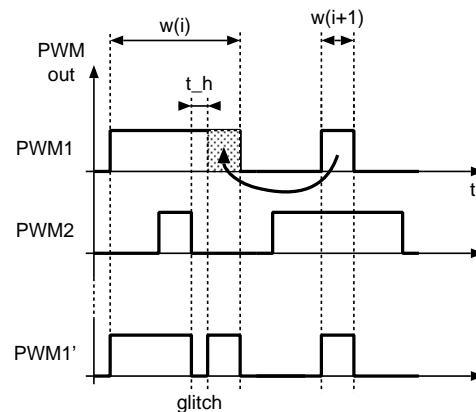


Figure 9.

The setup time  $t_s$  and hold time  $t_h$  have a strong effect on the output and must be considered in order to provide glitch-free output.

### Period Update

The last TC interrupt is triggered after all down counters have reloaded their values from the PERIOD registers. Thus, all current values from all PERIOD registers have been latched. That is why the last TC interrupt represents an ideal moment for updating the PERIOD registers for all PWM generators.

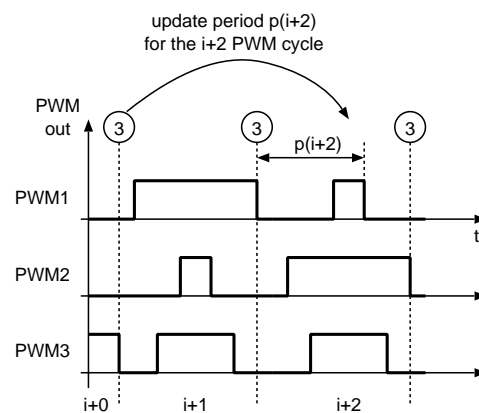


Figure 10.

New values written to the PERIOD registers in cycle  $i$  will take effect in the next cycle,  $i+1$ . There is a shift of one cycle between width and period updates. Pulse widths are updated on the first interrupt of the PWM cycle  $i$  in order to update pulse width in cycle  $i+1$ . But periods are updated after the last TC interrupt is triggered at which time the  $i+1$  PWM cycle is already in progress. That is why periods will effectively be updated in the  $i+2$  PWM cycle.

Hence, to synchronize operation, pulse width values must be delayed for one PWM cycle.

## Stability Criteria

The Stability Criteria defines a general rule, which guarantees proper and stable operation of the three-phase, sine-wave generator. If the Stability Criteria cannot be met, the three-phase generator will malfunction and the centers of the PWM pulses will become randomly positioned.

All three TC interrupts must execute within the  $t_{irqexec\_max}$  time, which is the time from one period update to the second period update. Otherwise,  $i$ -th period cannot be updated. A similar condition was already described with Equation (5). For the three-phase generator, Equation (5) becomes a little bit different since three TC interrupts must be handled within the same time and period according to the phase of the modulated sine wave.

The worst case represents the minimum time between two period updates. For arbitrary signals, extreme points are definitely the largest PWM width and the shortest PWM width, as shown below:

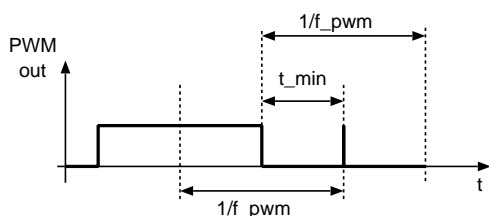


Figure 11.

In this case, the maximum time given to service all three interrupts equals:

$$t_{irqexec\_max} = (P+1) / (2f\_clk) \quad (10)$$

Again, note that this time is the absolute maximum time it takes to service all interrupts, which includes maximum system interrupt latency and the full time needed for execution of the three interrupts. Take care on the CPU stalls produced by the D/A converters, segments of code which disable interrupts for a time, and all other things that may lengthen the interrupt latency time.

The three-phase system can never exhibit such extreme conditions because the third TC interrupt always follows the upper side of the envelope of the system, as described before. The maximum possible width for the third TC interrupt equals 100% duty cycle, when  $A(i)=1$ . The minimum duty cycle for the PWM that generates the third interrupt is 50%, when  $A(i)=0$ .

If amplitude does not change, the minimum possible duty cycle is equal to the intersection point of the adjacent sine waves, with  $\sin/\cos$  value of 0.5. Recalling Equation (8) and for maximum amplitude, the duty cycle is:

$$d(i) = (1 + A(i) \sin[f(i)]) / 2 = 0.75 \quad (11)$$

But this is not the worst-case condition, which is what we are looking for.

Further evaluation of the equations shows that the change of the duty cycle is directly proportional to the change of time  $t_{irqexec\_max}$ , by the following relation:

$$t_{irqexec\_max} = (P+1) / (f\_clk) (1 - \frac{1}{2}(d(i) - d(i+1))) \quad (12)$$

Substituting  $d(i)=0.1$  with  $d(i+1)=0.5$ , we obtain the worst-case condition of stability for the generation of the three-phase sine wave.

$$t_{irqexec\_max} = 0.75 (P+1) / (f\_clk) \quad (13)$$

Equation (13) gives 50% more time than Equation (10) as calculated for an arbitrary function. This equation also defines the theoretical maximum possible PWM frequency, which is approximately 51 kHz when PSoC is running at 24 MHz with CPU load of 75%. All three TC interrupts together require approximately 350 CPU cycles and there are no other interrupts in the system.

Practical applications require the PWM frequency to be between 6 kHz and up to 20 kHz. Using the same parameters as above, the CPU load would range from 9% to 30%. Note that this CPU load usage includes the ISR only. Extra load is required for the sine function look-up routines as well as other functions.

## Glitch-Free Criteria

The Glitch-Free Criteria define constraints for the setup and hold times, which must be met to provide glitch-free output. If these time constraints are not met, a glitch might occur at the beginning of the next pulse if setup time is not met, or at the end of current pulse if hold time is not met. Both are analyzed separately.

### Setup Time

The setup time was already defined under the "Pulse Width Update" subsection. With the following figure, a general rule that applies to an arbitrary signal is derived.

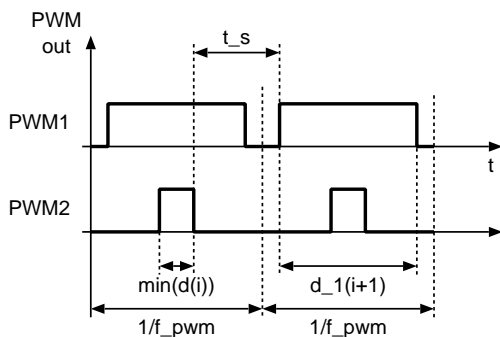


Figure 12.

$$t_s(i) = (P+1) / (2f_{clk}) \left( [2 \ 2 \ 2]' - \min(d(i)) [1 \ 1 \ 1]' - d(i+1) \right) \quad (14)$$

$t_s(i)$  is the vector for all setup times for all three PWM signals.  $d(i)$  is the vector of all three duty cycles and  $\min(d(i))$  returns the minimum duty cycle from the  $d(i)$  vector, which represents the first TC interrupt.

In the three-phase system the first TC interrupt always follows the lower side of the envelope of the system, which gives the maximum possible duty cycle for the first TC interrupt. But with limiting the amplitude to zero, the duty cycle approaches 50%, and consequently  $d(i)=0.5$ .

For the shortest setup time we set  $d(i+1)=1$ . What will happen when phase is 90 degrees and amplitude steps from 0 to 1? Substituting these two values into (14), we obtain the worst case.

$$t_s(i) = (P+1) / (2f_{clk}) (2 - 0.5 - 1) = (P+1) / (4f_{clk}) = 1 / (4f_{pwm}) \quad (15)$$

The setup time includes maximum possible latency for the first TC interrupt and its execution time. Assuming that the first TC interrupt takes 120 CPU cycles and output frequency is approximately 16 kHz, the required minimum setup time according to Equation (15) is 15  $\mu$ s, and practical implementation achieves 5  $\mu$ s.

This means that no glitch will occur at the beginning of any PWM pulse with any change in amplitude or phase.

Whenever setup time cannot be achieved according to the worst-case Equation (15), it does not mean that the system will always produce glitches but rather that the dynamic range of the sine wave is limited. Glitch-free, sine-wave bandwidth may be obtained from the general rule, Equation (14), and duty cycle, Equation (8), for the given setup time.

Take extra care when working with glitches. They may cause system instability along the path from PWM output to the full bridge in the power section.

Generally, inverters will not suffer from random glitches that might occur due to the change of amplitude or phase.

### Hold Time

Hold time has been defined under the "Pulse Width Update" subsection. We find the hold time for an arbitrary signal in the figure below:

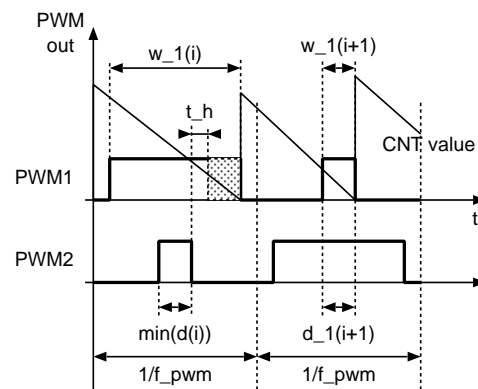


Figure 13.

$$t_h(i) = (P+1) / (2f_{clk}) (d(i) - 2d(i+1) - \min(d(i)) [1 \ 1 \ 1]') \quad (16)$$

$t_h(i)$  is the vector for all hold times for all three PWM signals.  $d(i)$  is the vector of all three duty cycles and  $\min(d(i))$  returns the minimum duty cycle from the  $d(i)$  vector, which represents the first TC interrupt.

A general rule (constraint) that must be met to avoid glitches at the end of any PWM signal is:

$$t_h(i) < 0 \quad (17)$$

Assuming that the first interrupt is always triggered by the PWM with a duty cycle of 0%, the last part of Equation (17) is 0. For worst-case calculation, we assume maximum hold time to be  $t_h=0$ , then we obtain:

$$d(i+1) > \frac{1}{2}d(i) \quad (18)$$

This implies that the arbitrary signal, which successive widths are never less than a half of the previous width, may be produced without a glitch. Note that condition (18) assumes that one's PWM signal has duty cycle of 0, for which this condition is *not* true.

For an arbitrary signal, glitch-free operation may be approved using equations (16) and (17). The simplified condition in Equation (18) shall be used only to find the maximum bandwidth of the sine function in the three-phase system.

Substituting (8) into (18) we get:

$$(1 + A(i+1) \sin[f(i+1)]) > \frac{1}{2} (1 + A(i) \sin[f(i)]) \quad (19)$$

The first derivative of the sine function has its maximum at 0; hence, we shift (19) into that area and seek for maximum allowable change in the phase for the given amplitude. That is when the left part of the equation has the minimum value and the right-hand part the maximum value around the 0.

$$(1 + A(i+1) \sin[-f_{\max}]) > \frac{1}{2} (1 + A(i) \sin[+f_{\max}]) \quad (20)$$

We further simplify (20) by setting the amplitude to its maximum 1, since it yields the lowest left-hand part of the equation.

$$\frac{1}{2} + \sin[-f_{\max}] > \frac{1}{2} \sin[+f_{\max}] \quad (21)$$

Equation (21) yields a maximum change in the phase of approximately 40 degrees ( $2 f_{\max}$ ) for glitch-free operation. Furthermore, it introduces the maximum bandwidth of a three-phase sine system at full load (maximum amplitude), which is equal to:

$$BW = f_{\text{pwm}} 40/360 = f_{\text{pwm}} / 9 \quad (22)$$

Reducing the amplitude will, of course, increase the bandwidth in Equation (22).

## Period Range

We recall Equation (4) to confirm the maximum dynamic range (bandwidth) of the sine function.

$$0 < [P + (w(i) - w(i+1)) / 2] < 256$$

The given maximum step of 40 degrees yields a maximum duty-cycle change of 34%. The relationship between the width and duty cycle was given in Equation (1).

$$d = w / (P+1)$$

Substituting Equation (1) into Equation (4) gives:

$$0 < [P + (P+1) (d(i) - d(i+1)) / 2] < 256 \quad (23)$$

It can clearly be seen that the condition on the left is always true. Hence, we must only confirm the condition on the right. For the given maximum duty-cycle change of 34%, we get:

$$P < 218 \quad (24)$$

In the previous example,  $P$  equals 199 and the PERIOD register will not suffer from overflow.

## Summary

This note describes a general algorithm for generating an arbitrary function, with some limitations, through a three-phase generator. A detailed analysis is also provided for the sine modulated PWM output.

Every application that uses this three-phase generator must be checked for the following:

### Stable Operation:

Period Range Overflow/Underflow: Equation (4)

Stability Criteria: Equation (13)

### Glitch-Free Output:

Setup Time: Equation (14) or (15)

Hold Time: Equation (17) or (22)

When two equations are given, the first applies to the general rule and the second to the sine. Note that P, PI, PID, and other regulators may produce high-bandwidth signals (phase, amplitude). In such cases, the output from these regulators must be filtered - bandwidth limited, so to confirm all criteria above.

## Example Code

The example code in the Appendix shows a three-phase sine generator with PWM output frequency of 17 kHz. The entire sine period counts 150 samples, and  $P = 199$ . This gives a three-phase output frequency of 114 Hz. All PWM8 User Modules use a 24V1 clock, which has divisor=7. Setting the divisor to 16, the output is 50 Hz and the PWM frequency is 7.5 kHz.

To see the three-phase output, simply apply RC-networks to pins P1[0], P1[1], and P1[2] with the time constant around  $1/f\_pwm$ .

The code is divided into:

- Three-phase module header file (*tfpwm.h*)
- Three-phase module source code (*tfpwm.c*)
- Three-phase main (*main.c*)

The three-phase module provides all necessary API functions to correctly drive the three-phase generator. These are:

```
tfpwm_start()
tfpwm_stop()
tfpwm_set()
```

The first two functions are used to start and stop the module while the last is used to set phase and amplitude.

Note that *main.c* is simple as there is no other code besides this module. According to the Stability Criteria period of the PWM, the pulse must be updated at least two times when a change in pulse width occurs. If the Stability Criteria is met but the code is not capable of providing the next period and width values for all three-modules, the three-phase generator will not malfunction, but the actual PWM period will be invalid for two cycles.

The example code uses the comparator to synchronize all three PWM8 generators. Interrupts must be disabled during start-up configuration and re-enabled before the synchronization pulse occurs.

Additional synchronization is done between the API functions and the *tfpwm\_isr()* ISR to correctly latch all widths and periods of all PWMs at once.

**Note:** If an application starts and stops the three-phase module, pending interrupts can make the system unstable. That is why, after the three-phase module is stopped, interrupts must stay enabled. Otherwise, all pending interrupts from the three PWM8 generators must be cleared by writing to the interrupt vector register.

## Appendix A: TFPWM Header File

```
/* tfpwm.h, Uros Platise <uros.platise@ijs.si> */

#ifndef __TFPWM_H__
#define __TFPWM_H__

/* Table Length/3, Length/2 and total Length */
#define SIN_TABLE_N3      50
#define SIN_TABLE_N2      75
#define SIN_TABLE_N      (SIN_TABLE_N2 * 2)

/* PWM Range:
   PWM_PERIOD = full_range - 1,
   PWM_HPERIOD = full_range / 2
*/
#define PWM_PERIOD        199
#define PWM_HPERIOD       100

/* PWM Sync. semaphore. When zero, PWM variables are ready to accept new data
   and tfpwm_set() function will execute immediately.
*/ extern unsigned char pwm_load_flag;

/* function prototypes */

/* Initializes PWMs and performs synchronous start of all PWMs */
void tfpwm_start(unsigned char sine_phase, unsigned char amplitude);

/* Stop all three PWMs */
void tfpwm_stop(void);

/* Set output phase of the sine wave, requires global interrupt enabled. */
void tfpwm_set(unsigned char sine_phase, unsigned char amplitude);

#endif
```

## Appendix B: TFPWM Source Code

```

/* tfpwm.c
   Three Phase Sine Generator
   Uros Platise (c) 2003, <uros.platise@ijs.si>
*/

#include <tfpwmapi.h>
#include "tfpwm.h"

/* to provide debug sync. output so you may see the center-based pulses on the scope */
//#define DEBUG

#define PSoC_Designer_V4

/* fix the differences between the PSoC builder version 3 and 4 */
#ifndef PSoC_Designer_V4
#define PWM8_1_PWIDITH_REG    PWM8_1_COMPARE_REG
#define PWM8_2_PWIDITH_REG    PWM8_2_COMPARE_REG
#define PWM8_3_PWIDITH_REG    PWM8_3_COMPARE_REG
#endif

/* Table of Half-Wave Symmetric Sinus Function */
const char sin_table[SIN_TABLE_N2] =
{
    0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28,
    29, 31, 33, 34, 36, 37, 39, 40, 41, 42, 43, 44, 45, 46,
    47, 48, 48, 49, 49, 49, 50, 50, 50, 50, 50, 50, 49, 49,
    49, 48, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 37, 36,
    34, 33, 31, 29, 28, 26, 24, 22, 20, 18, 16, 14, 12, 10,
    8, 6, 4, 2
};

/* global variables for synchronous load */
unsigned char pwm1, period1, pwm2, period2, pwm3, period3;

/* pwm width are delayed for one period */
unsigned char bpwm1, bpwm2, bpwm3;

/* semaphore */
unsigned char pwm_load_flag = 0;

/* test sync. output */
unsigned char debug_sync_out = 0;

/*
   Sine Retrieve function
*/
unsigned char sine_calwidth(unsigned char phase, unsigned char amplitude)
{
    unsigned char tval;

    if (phase >= SIN_TABLE_N) phase -= SIN_TABLE_N;

    if (phase < SIN_TABLE_N2)
    {
        tval = sin_table[phase];
        return PWM_HPERIOD + (tval << 1);    // make it even, add a zero on right side
    }
    else
    {
        tval = sin_table[phase - SIN_TABLE_N2];
        return PWM_HPERIOD - (tval << 1);    // make it even, add a zero on right side
    }
}

/*
   Three Phase PWM Fast Interrupt

   Note: C-compiler compiles the following ISR so good,
   as it were almost written in assembler. Minor optimizations
   may still be applied. Check the output .lst file for details.

   Placement of the PWM8 blocks _must_ be such that they are
   assigned highest priority interrupts - all three!

   Approximate cycle analyzed:

   IRQ_COUNT PWM_UPLOAD    CYCLES
   -----
   0          -            -
   1          -            27+35+43 = 105
   2          YES          27+43    = 70
                          27+73+43 = 143

```

```

2 ----- NO ----- 27+24+43 = 94
-----
WORST CASE TOTAL: 318

Details:
-----
IRQ_Entry: 27
IRQ_Count_0: 35
IRQ_Count_1: 0
IRQ_Count_2: 24 / 73
IRQ_Exit: 43
*/

#pragma interrupt_handler tfpwm_isr

void tfpwm_isr()
{
    static unsigned char irq_count=0;

    if (irq_count==0)
    {
        PWM8_1_PWIDTH_REG = bpwm1;
        PWM8_2_PWIDTH_REG = bpwm2;
        PWM8_3_PWIDTH_REG = bpwm3;
    }
    else if (irq_count==2)
    {
        if (pwm_load_flag)
        {
            PWM8_1_PERIOD_REG = period1;
            PWM8_2_PERIOD_REG = period2;
            PWM8_3_PERIOD_REG = period3;
            bpwm1 = pwm1;
            bpwm2 = pwm2;
            bpwm3 = pwm3;

#ifdef DEBUG
            PRT1DR = debug_sync_out;
#endif

            pwm_load_flag = 0;
        }
        else
        {
            /* if load was unsuccessful, next period has no change! */
            PWM8_1_PERIOD_REG = PWM_PERIOD;
            PWM8_2_PERIOD_REG = PWM_PERIOD;
            PWM8_3_PERIOD_REG = PWM_PERIOD;
        }
    }

    irq_count++;
    if (irq_count==3) irq_count=0;
#ifdef DEBUG
    PRT1DR = 0;
#endif
}

/*
Three Phase API
*/

void tfpwm_start(unsigned char sine_phase, unsigned char amplitude)
{
    M8C_DisableGInt;

    /* Set comparator bus to low (default) to perform sync. start */
    CMPPRG_1_Start(CMPPRG_1_LOWPOWER);
    CMPPRG_1_SetRef(CMPPRG_1_REF1_000);

    /* Set PWM start values / period and offset
    Default counter value is updated while counter is stopped.
    */

    pwm1 = bpwm1 = sine_calwidth(sine_phase, amplitude);
    PWM8_1_WritePeriod((bpwm1>>1)-1);
    PWM8_1_Start();
    PWM8_1_WritePeriod(PWM_PERIOD);
    PWM8_1_WritePulseWidth(bpwm1);
    PWM8_1_EnableInt();

    pwm2 = bpwm2 = sine_calwidth(sine_phase + SIN_TABLE_N3, amplitude);
    PWM8_2_WritePeriod((bpwm2>>1)-1);
    PWM8_2_Start();
    PWM8_2_WritePeriod(PWM_PERIOD);
    PWM8_2_WritePulseWidth(bpwm2);
    PWM8_2_EnableInt();

    pwm3 = bpwm3 = sine_calwidth(sine_phase + (2*SIN_TABLE_N3), amplitude);
    PWM8_3_WritePeriod((bpwm3>>1)-1);
    PWM8_3_Start();
}

```

```

PWM8_3_WritePeriod(PWM_PERIOD);
PWM8_3_WritePulseWidth(bpwm3);
PWM8_3_EnableInt();

/* new settings are ok */
pwm_load_flag = 0;

/* Run All - Sync. Start */
CMPPRG_1_SetRef(CMPPRG_1_REF0_062);
M8C_EnableGInt;

/* After this point, comparator may be reused for other purposes ...
(do not forget to set pwm enable bits to high)
*/
}

void tfpwm_stop(void)
{
    PWM8_1_Stop();
    PWM8_1_DisableInt();
    PWM8_2_Stop();
    PWM8_2_DisableInt();
    PWM8_3_Stop();
    PWM8_3_DisableInt();
}

void tfpwm_set(unsigned char sine_phase, unsigned char amplitude)
{
    /* wait until pwm isr loads new settings */
    while(pwm_load_flag);

#ifdef DEBUG
    if (sine_phase==0) debug_sync_out = 0x08; else debug_sync_out = 0x00;
#endif

    /* calculate new values for the pwm */
    pwm1  >>= 1;
    period1 = PWM_PERIOD - pwm1;
    pwm1    = sine_calwidth(sine_phase, amplitude);
    period1 += (pwm1 >> 1);

    pwm2  >>= 1;
    period2 = PWM_PERIOD - pwm2;
    pwm2    = sine_calwidth(sine_phase + SIN_TABLE_N3, amplitude);
    period2 += (pwm2 >> 1);

    pwm3  >>= 1;
    period3 = PWM_PERIOD - pwm3;
    pwm3    = sine_calwidth(sine_phase + (2*SIN_TABLE_N3), amplitude);
    period3 += (pwm3 >> 1);

    /* request pwm update */
    pwm_load_flag = 1;
}

```

## Appendix C: Example Main

```

// main.c
// Uros Platise (c) 2003 July <uros.platise@ijs.si>

#include "tfpwm.h"

void main()
{
    unsigned char phase = 0;

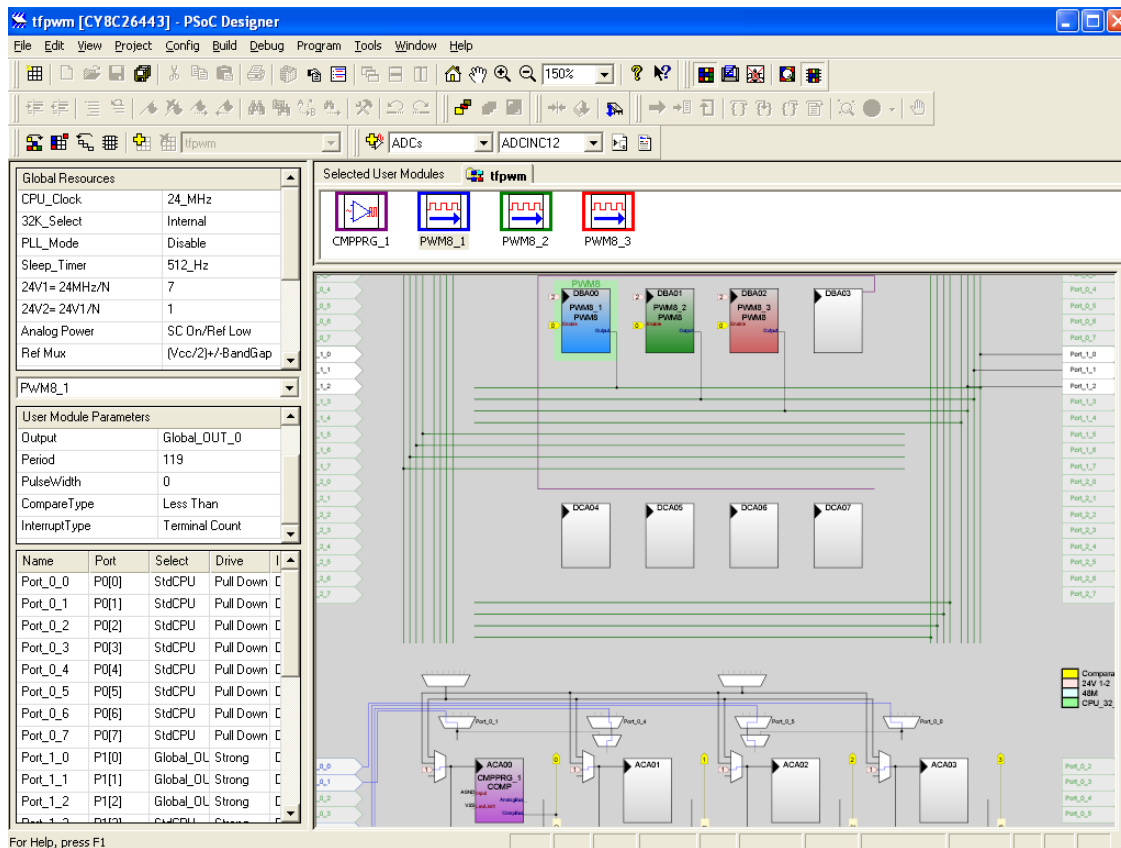
    tfpwm_start(phase, 1);

    /* Demonstrate three phase sine output */
    while(1)
    {
        tfpwm_set(phase, 1);
        phase += 1;
        if (phase >= SIN_TABLE_N) phase -= SIN_TABLE_N;
    }

    tfpwm_stop();
}

```

## Appendix D: PSoC Configuration



### About the Author

**Name:** Uroš Platiše

**Title:** Research & Development

**Background:** Mixed-signal designs, computer and CPLD/FPGA programming, control, sensors, measurement...

**Contact:** Uros Platise

Seljakovo naselje 45

SI-4000 Kranj

Slovenia

Email: [uros.platise@ijs.si](mailto:uros.platise@ijs.si)

Web: <http://www.andeuos.org>

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

<http://www.cypress.com/> / <http://www.cypress.com/support/mysupport.cfm>

Copyright © 2004 Cypress MicroSystems, Inc. All rights reserved.

PSoC™ (Programmable System-on-Chip) and PSoC Designer are trademarks of Cypress MicroSystems, Inc.

All other trademarks or registered trademarks referenced herein are property of the respective corporations.

The information contained herein is subject to change without notice.