AN2157

Analog Three-Phase Sine Wave Generator

Author: Uroš Platiše

Associated Project: Yes

Associated Part Family: CY8C27xxx, CY8C29xxx

Software Version: PSoC® Designer™ 5.4

Related Application Notes: AN2141

This application note provides and describes an implementation of the symmetric and glitch-free three-phase sine wave generator using three 8-bit PWM User Modules. This generator can be used to drive three-phase inverters and three-phase electric drives, such as the popular AC and permanent magnet synchronous motor (PMSM) drives.

Contents

1 Introduction 1
2 PSoC Resources 2
2.1 PSoC Designer 2
2.2 Code Examples 3
2.3 Technical Support 4
3 Key Features 5
4 Applications 5
5 Design Structure 5
5.1 Building Blocks 5
6 Three-Phase Composition, Requirements, and Synchronization 7
7 Symmetric Sine Wave 7
8 Center-Based PWM 8
9 Sine Wave Generation 9
10 PWM8 Operation 9
11 Three-Phase Sine Wave Generation 9
12 Symbols and Definitions 10
13 Interrupt Sorting and PWM Updating 10
14 Arrangement of the TC Interrupts 10
15 Pulse Width Update 11
16 Period Update 12
17 Stability Criteria 13
18 Glitch-Free Criteria 14
18.1 Setup Time 14
18.2 Hold Time 15
18.3 Period Range 16
19 Example Code 16
20 Summary 17
A Appendix A: TFPWM Header File 18
B Appendix B: TFPWM Source Code 19
C Appendix C: Example Main 24
D Appendix D: PSoC Configuration 25
Document History 27
Worldwide Sales and Design Support 28

1 Introduction

As the popularity of three-phase electric drives increases, so does the need for a three-phase sine wave generator. Usually, DSP machines are used in such applications. However, this application note shows that PSoC® 1, with its unique structure, can generate three-phase sine waves, which then can directly drive electric drives and inverters.

This document focuses on the generation of three-phase sine waves with the same characteristics as those generated by the special motor PWM blocks in DSP machines. It provides detailed information on stability issues, glitch-free operation, and symmetric sine function computation requirements and limitations.

2 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and quickly and effectively integrate the device into your design. In this document, PSoC refers to the PSoC 1 family of devices. To learn more about PSoC 1, refer to the application note AN75320 - *Getting Started with PSoC 1*.

The following is an abbreviated list for PSoC 1:

- **Overview:** PSoC Portfolio, PSoC Roadmap
- **Product Selectors:** PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP. In addition, PSoC Designer includes a device selection tool.
- **Datasheets:** Describe and provide electrical specifications for the PSoC 1 device family.
- **Application Notes and Code Examples:** Cover a broad range of topics, from basic to advanced level. Many of the application notes include code examples.
- **Technical Reference Manuals (TRM):** Provide detailed descriptions of the internal architecture of the PSoC 1 devices.
- **Development Kits:**
  - CY3215A-DK In-Circuit Emulation Lite Development Kit includes an in-circuit emulator (ICE). While the ICE-Cube is primarily used to debug PSoC 1 devices, it can also program PSoC 1 devices using ISSP.
  - CY3210-PSOCEVAL1 Kit enables you to evaluate and experiment Cypress's PSoC 1 programmable system-on-chip design methodology and architecture.
  - CY8CKIT-001 is a common development platform for all PSoC family devices.
- The MiniProg1 and MiniProg3 devices provide an interface for flash programming.

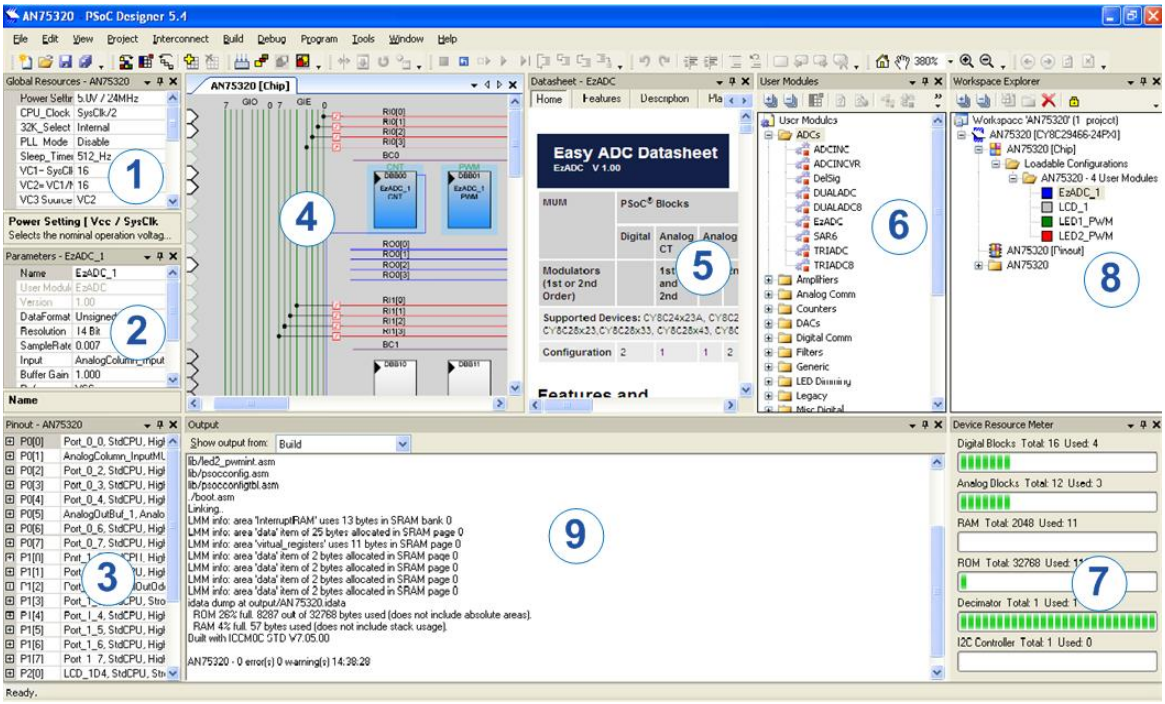
2.1 PSoC Designer

PSoC Designer is a free Windows-based Integrated Design Environment (IDE). Develop your applications using a library of pre-characterized analog and digital peripherals in a drag-and-drop design environment. Then, customize your design leveraging the dynamically generated API libraries of code. Figure 1 shows PSoC Designer windows. **Note:** This is not the default view.

1. **Global Resources –** all device hardware settings.
2. **Parameters –** the parameters of the currently selected User Modules.
3. **Pinout –** information related to device pins.
4. **Chip-Level Editor –** a diagram of the resources available on the selected chip.
5. **Datasheet –** the datasheet for the currently selected UM
6. **User Modules –** all available User Modules for the selected device.
7. **Device Resource Meter –** device resource usage for the current project configuration.
8. **Workspace –** a tree level diagram of files associated with the project.
9. **Output –** output from project build and debug operations.

Note: For detailed information on PSoC Designer, go to **PSoC® Designer** > **Help** > **Documentation** > **Designer Specific Documents** > **IDE User Guide**.

Figure 1. PSoC Designer Layout



2.2 Code Examples

The following webpage lists the PSoC Designer based Code Examples. These Code Examples can speed up your design process by starting you off with a complete design, instead of a blank page and also show how PSoC Designer User modules can be used for various applications.

http://www.cypress.com/go/PSoC1Code Examples

To access the Code Examples integrated with PSoC Designer, follow the path **Start Page** > **Design Catalog** > **Launch Example Browser** as shown in Figure 2.

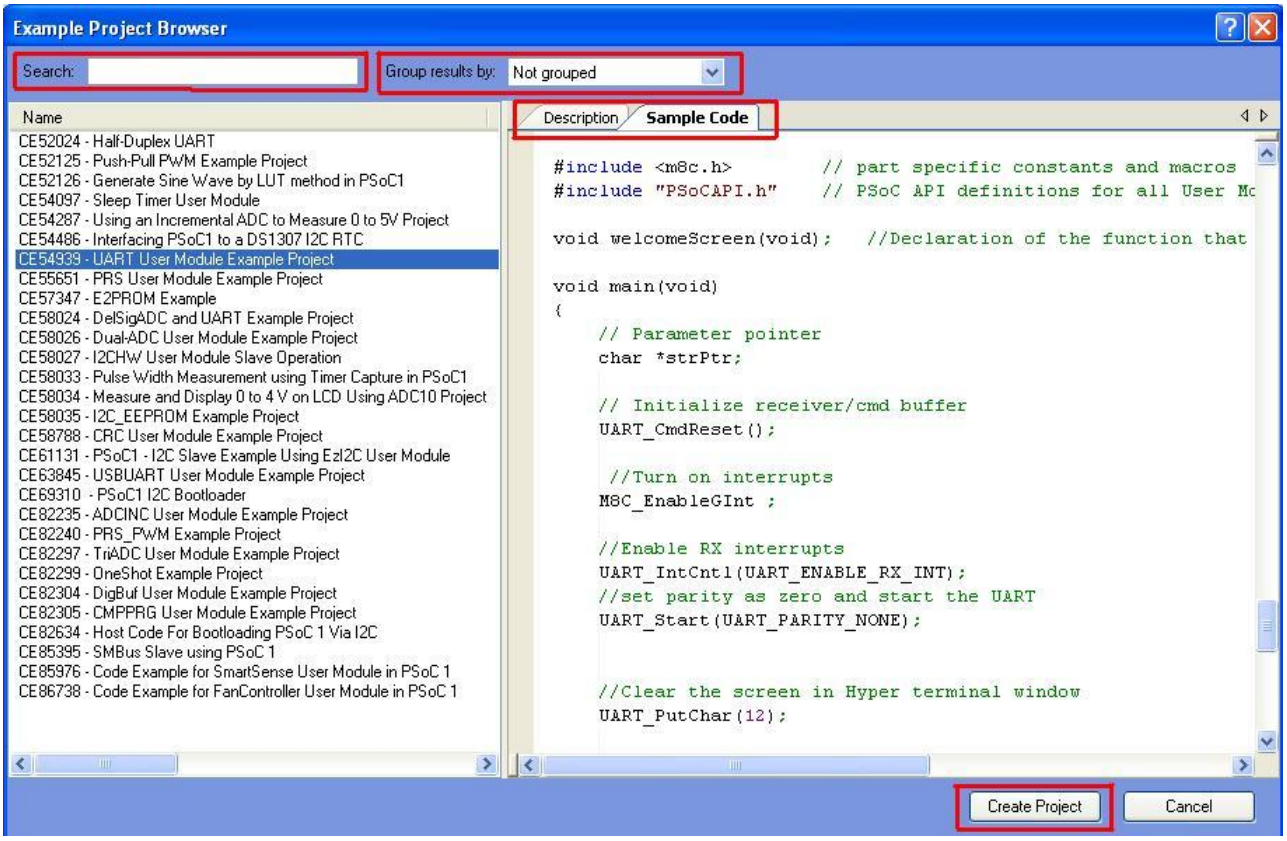
Figure 2. Code Examples in PSoC Designer



In the Example Projects Browser shown in Figure 3, you have the following options.

- Keyword search to filter the projects.
- Listing the projects based on Category.
- Review the datasheet for the selection (on the Description tab).
- Review the code example for the selection. You can copy and paste code from this window to your project, which can help speed up code development, or
- Create a new project (and a new workspace if needed) based on the selection. This can speed up your design process by starting you off with a complete, basic design. You can then adapt that design to your application.

Figure 3. Code Example Projects, with Sample Codes



2.3 Technical Support

If you have any questions, our technical support team is happy to assist you. You can create a support request on the Cypress Technical Support page.

You can also use the following support resources if you need quick assistance.

- Self-help
- Local Sales Office Locations

3 Key Features

This application note introduces PSoC1 glitch free three-phase sine wave generator. The key features of PSoC1 three-phase sine wave generator are following:

- Symmetric PWM outputs
- Glitch free
- PWM resolution from 8 bit 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

4 Applications

The PSoC 1 generated three-phase sine can be used in following applications:

- Single-phase inverters with symmetric PWM outputs such as uninterruptible power supplies (UPS)
- Three-phase industry inverters
- AC motor drives with V/Hz control
- Permanent magnet synchronous motor (PMSM) drives
- Power factor correction (PFC)
- Robotics and industrial control

5 Design Structure

The three-phase design incorporates three PWM User Modules, which may be either 8 bit 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 can be used to interface directly to the inverter's power section.

5.1 Building Blocks

The 8-bit PWM block is known as the PWM8 User Module. This section reviews 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 the *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 repeatedly.

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 equal to the value stored in the COMPARE register, the PWM output is switched into a HIGH state. Otherwise, it is in a LOW state.

This three-phase design recommends using the less-than comparison and that the PERIOD registers have an odd number of 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* (that is, *w*=254, *p*=253)
- LOW state when *w*=0

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

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

The COMPARE register holds the pulse width value.

PWM frequency, *f*_pwm, is equal to:

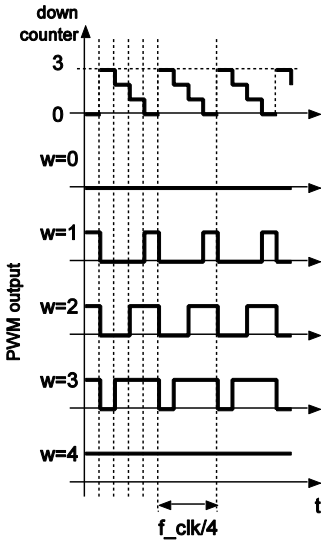
f_pwm = f_clk / (P+1) Equation 2

*f*_clk is the system clock for all three PWM modules, which has direct influence on CPU usage...

Figure 4 illustrates a PWM with the two aforementioned conditions met. It has the following parameters:

- *P*=3
- *w* has the values 0, 1, 2, 3, and 4.
- System clock *f*_clk =100 kHz

Figure 4. PWM

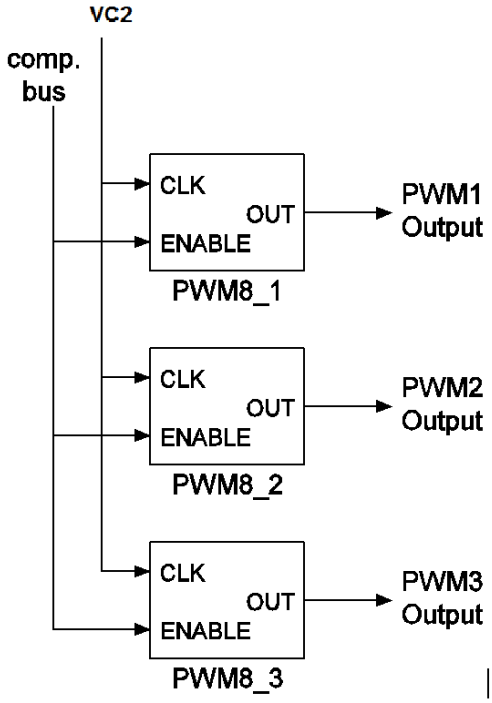


The figure displays the symmetry, which plays an important role in the power section to avoid unwanted saturations in ferromagnetic cores.

6 Three-Phase Composition, Requirements, and Synchronization

Figure 5 shows the composition of the three-phase design.

Figure 5. Phase Design Composition

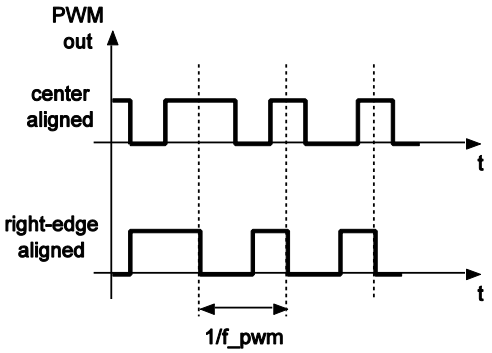


The design requires three PWM modules to be placed in adjacent digital blocks with the highest interrupt priorities. All must use the same system clock, *f*_clk, such as VC1, VC2, or other. The PWM8 ENABLE input of all modules must be tied to the same controllable source, which may be a compare bus, global input, or broadcast signal. This source is needed during startup synchronization to correctly position the centers of all three PWM outputs. The PWM8 blocks must be set to trigger an interrupt on Terminal Count (TC), and comparison is set to Less Than.

7 Symmetric Sine Wave

The PWM8 and PWM16 blocks are not able to generate symmetric PWM (center-aligned) outputs, but they are asymmetric and aligned to the right edge, as shown in Figure 6.

Figure 6. PWM Outputs



Symmetric PWM (center-aligned) output, in comparison to asymmetric PWM (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 can be generated.

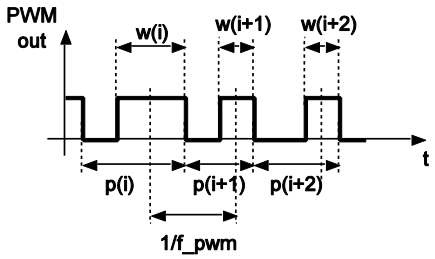
Symmetric sine wave can be generated with the PWM8 module by simultaneous modification of the PERIOD register and COMPARE register. Changing the value of the PERIOD register will effectively shift the center of the PWM pulse to the left, when decreased, or to the right, when increased, on the time axis.

The following sections discuss center-based PWM signals and give additional considerations for use of the PWM8 module.

8 Center-Based PWM

From Figure 7, you can derive Equation 3 for any arbitrary *p*(i).

Figure 7. Period and Pulse Width



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

w(i) and *w*(i+1) 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. You can see that the period value for the *i*-th PWM pulse is determined from the two adjacent pulse widths.

In this algorithm, the values of the pulse widths w(i) must be even; otherwise, an error may occur when dividing by two, as shown in Equation 3. 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:

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

Equation 4 is described in the Period Range section.

The same equation also implies the maximum allowable execution time (*t*_irqexec_max) of the PWM interrupt, because periods must be updated continuously, one after another. The time equals:

$t_irqexec_max = (p(i) + 1) / f_clk$ Equation 5

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

$t_irqexec_max = t_irq_latency + t_irqexec$ Equation 6

*t*_irq_latency is the maximum system interrupt latency time, and *t*_irqexec is the time needed to execute the PWM interrupt.

Note: When the pulse width changes, the PERIOD register is updated two times, as illustrated by the following settings:

- *P*=59
- w(i)=30; i < 0
- *w*(0)=32

- $w(i)=32; i > 0$

Substituting the values in Equation 3 results in the following:

- $p(-1) = 59 + (30 – 30)/2 = 59$
- $p(0) = 59 + (32 – 30)/2 = 60$
- $p(1) = 59 + (30 – 32)/2 = 58$
- $p(2) = 59 + (30 – 30)/2 = 59$

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.

9 Sine Wave Generation

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

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

Here expressed with a duty cycle:

$d(i) = (1 + A(i) \sin[f(i)]) / 2$ Equation 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 percent, a maximum value of 100 percent, and a minimum value of 0 percent. To increase the performance and avoid extensive computation of the sine function, at least one-quarter of the sine function is given as a lookup table.

10 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 the output PWM frequency is fixed (actual period equals *P*), providing no deviation from the specified value *f*_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 for the sine wave generator and the first property represents glitch-free criteria.

Single center-based sine wave output can be obtained using the glitch-free generator described in AN2141, with a little modification. The PWM8 interrupt must update the PERIOD register in addition to the COMPARE register.

The stability criteria are derived by the *t*_irqexecmax parameter given in Equation 5 and the PERIOD(i) range defined in Equation 4. The glitch-free criteria conform to the glitch-free PWM described in AN2141.

11 Three-Phase Sine Wave Generation

The beauty of the three-phase system proposed by Nikola Tesla is that everything is simplified. It is also true in this case, where the three-phase sine wave, glitch-free implementation is less complicated than the single sine wave, glitch-free implementation described in Three-Phase Composition, Requirements, and Synchronization.

Implementation of the three-phase sine generator uses three PWM8 modules and a single ISR to handle interrupts from all three modules. All PWM8 modules must be set to trigger on 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 the PWM PERIOD registers and the PWM widths (COMPARE registers).

12 Symbols and Definitions

This section redefines some symbols: PWM width *w*, period *p*, duty cycle *d,* and phase *f* are modified as *w*_j, *p*_j, *d*_j, and *f*_j for the PWMj.

Arbitrary, even periodic, functions can be used to modulate the pulse width, but here it will be limited 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)+ 2π/3

*f*_3(i) = *f*_1(i)+4π /3

All three PWMs must be center aligned, and their time 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.

13 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. That is why it is important for all three PWM modules to have the highest interrupt priorities.

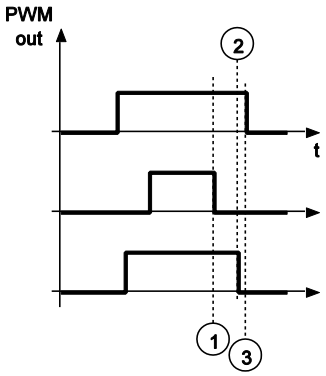
Interrupts are sorted automatically, due to the nature of the three-phase system, where one is always the first, one is always the second, and one is always the third. No other options are possible. All three PWM widths are updated when the first interrupt is triggered. The PWM periods are updated on the last interrupt. The last interrupt is also used to latch the 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.

14 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 is triggered by the PWM8, which has the longest PWM width, as shown in Figure 8.

Figure 8. Shortest and Longest PWM



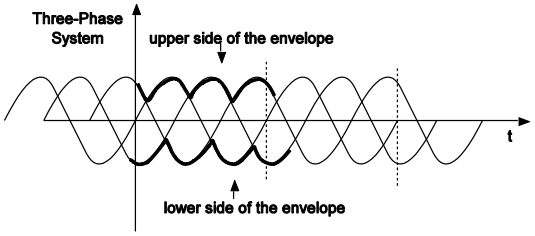
So you can write:

w_m ≤ w_n ≤ w_l Equation 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 != w_l*.

Figure 9 shows the arrangement of the TC interrupts in a center-based PWM system.

Figure 9. Three-Phase Sine Wave

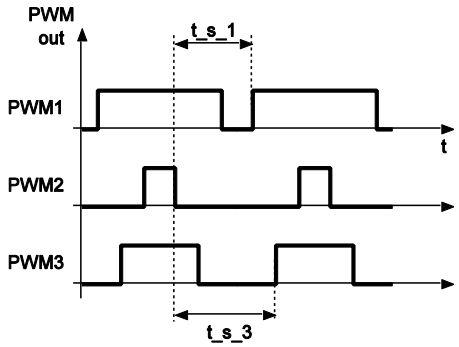


The first TC interrupt, *w_m*, follows the lower side of the envelope of the three-phase system. The last (third) TC interrupt, *w_l*, follows the upper side of the envelope of the three-phase system. The second TC interrupt is always triggered between the first and last and is represented by the middle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

15 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 the hardware, take extra care when these registers are updated. Figure 10 defines the setup time.

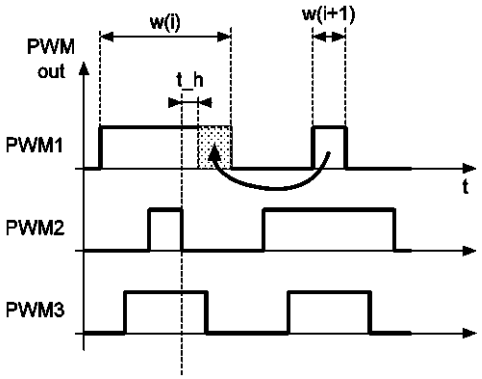
Figure 10. Setup Time



The first TC interrupt provides the longest setup time to all three PWM output rising edges and therefore 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.

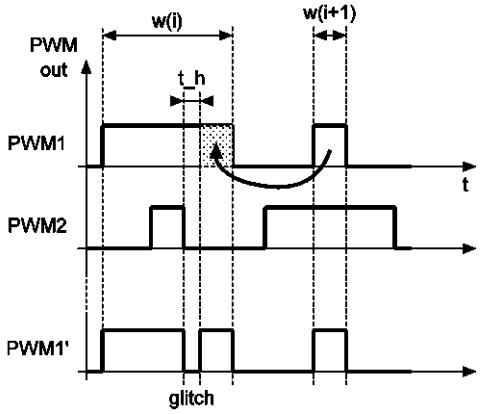
On the other hand, the first TC interrupt requires a longer hold time before the COMPARE register can be written with the new pulse width value. Figure 11 defines the hold time.

Figure 11. Hold Time



For example, if the COMPARE register is updated with a very short width before its PWM output signal is switched into the LOW state, the output will generate a glitch at the end of the PWM cycle, as shown in Figure 12.

Figure 12. Glitch

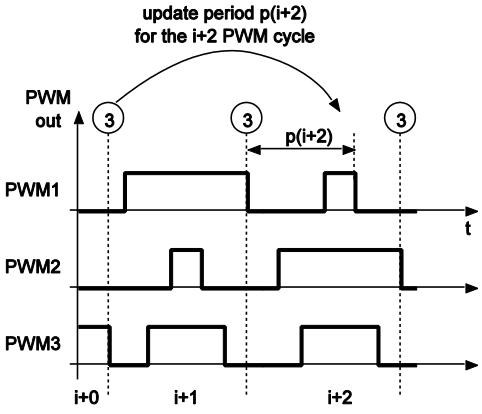


The setup time, *t*_s, and hold time, *t*_h, have a strong effect on the output and must be considered to provide glitch-free output.

16 Period Update

The last TC interrupt is triggered after all the 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 source for updating the PERIOD registers for all PWM generators, as shown in Figure 13.

Figure 13. Period Update



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,* to update the 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.

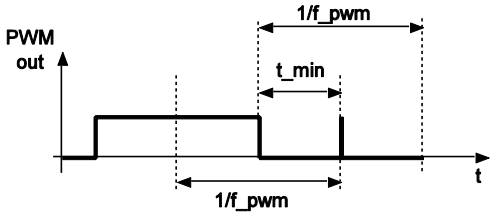
17 Stability Criteria

The stability criteria define a general rule, which guarantees proper and stable operation of the three-phase sine wave generator. If these 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 described with Equation 5. For the three-phase generator, Equation 5 becomes a 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 the largest PWM width and the shortest PWM width, as shown in Figure 14.

Figure 14. PWM Signals



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

t_irqexec_max = (P+1) / (2f_clk) Equation 10

Again, note that this time is the absolute maximum time it takes to service all interrupts, which includes the maximum system interrupt latency and the full time needed to execute the three interrupts. Take care with CPU stalls produced by the DACs, segments of code that disable interrupts for a time, and anything else 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 system envelope, as described in Arrangement of the TC Interrupts. The maximum possible width for the third TC interrupt equals 100 percent duty cycle when *A*(i)=1. The minimum duty cycle for the PWM that generates the third interrupt is 50 percent when *A*(i)=0.

If the amplitude does not change, the minimum possible duty cycle is equal to the intersection point of the adjacent sine waves, with a 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 Equation 11

But this is not the worst-case condition, which is what you want to see.

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 – ½(d(i) - d(i+1))) Equation 12

Substituting *d*(i)=0.1 with *d*(I+1)=0.5 provides the worst-case condition of stability for the generation of the three-phase sine wave.

t_irqexec_max = 0.75 (P+1) / (f_clk) Equation 13

Equation 13 gives 50 percent 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 the PSoC device is running at 24 MHz with a CPU load of 75 percent. 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 from 6 kHz up to 20 kHz. Using the same parameters as the previous ones, the CPU load would range from 9 percent to 30 percent. Note that this CPU load usage includes the ISR only. Extra load is required for the sine function lookup routines as well as other functions.

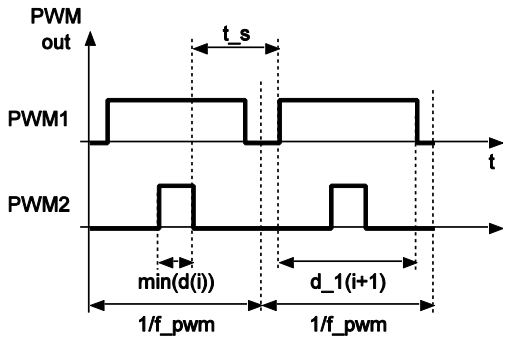
18 Glitch-Free Criteria

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

18.1 Setup Time

The Pulse Width Update section defines the setup time. You can derive a general rule that applies to an arbitrary signal with Figure 15.

Figure 15. Setup Time for Arbitrary Signals



$$t_s(i) = ((P+1) / (2\ f_clk))\ \ ([2\ 2\ 2]' - \min(d(i))\ [1\ 1\ 1]' - d(i+1)) \qquad \text{Equation 14}$$

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 when limiting the amplitude to zero, the duty cycle approaches 50 percent, and consequently d(*i*)=0.5.

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

$$t_s(i) = ((P+1) / (2\ f_clk))\ \ (2 - 0.5 - 1) = (P+1) / (4\ f_clk) = 1 / (4\ f_pwm) \qquad \text{Equation 15}$$

The setup time includes the maximum possible latency for the first TC interrupt and its execution time. Assuming that the first TC interrupt takes 120 CPU cycles and the output frequency is approximately 16 kHz, the required minimum setup time according to Equation 15 is 15 µs, and the practical implementation achieves 5 µs. This means that no glitch will occur at the beginning of any PWM pulse with any change in amplitude or phase.

Whenever the 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. A glitch-free sine wave bandwidth may be obtained from the general rule, Equation 14, and duty cycle, Equation 8, for the given setup time.

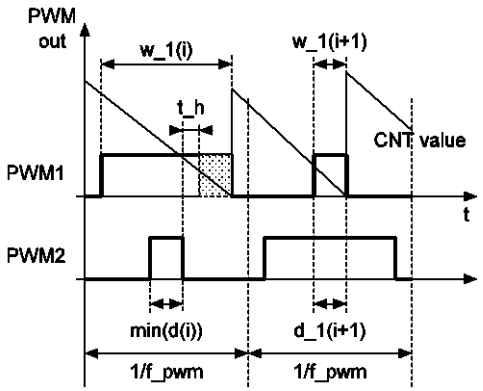
Take extra care when working with glitches. They may cause system instability along the path from PWM output to the full bridge in the power section due to setup time violation.

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

18.2 Hold Time

The Pulse Width Update section defines the hold time. You can find the hold time for an arbitrary signal with Figure 16.

Figure 16. Hold Time for Arbitrary Signals



$\boldsymbol{t_h}(i) = (P+1) / (2\ f_clk)\ \ (\boldsymbol{d}(i) - 2\boldsymbol{d}(i+1) - \min(\boldsymbol{d}(i))\ [1\ 1\ 1]')$ Equation 16

$\boldsymbol{t_h}(i)$ is the vector for all hold times for all three PWM signals. $\boldsymbol{d}(i)$ is the vector of all three duty cycles. $\min(\boldsymbol{d}(i))$ returns the minimum duty cycle from the $\boldsymbol{d}(i)$ 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:

$t_h(i) < 0$ Equation 17

Assuming that the first interrupt is always triggered by the PWM with a duty cycle of 0 percent, the last part of Equation 17 is 0. For the worst-case calculation, you assume maximum hold time to be $\boldsymbol{t}_\mathbf{h}=\mathbf{0}$ and then obtain:

$\boldsymbol{d}(i+1) > ½\boldsymbol{d}(i)$ Equation 18

This implies that the arbitrary signal, in which successive widths are never less than a half of the previous width, may be produced without a glitch. Note that Equation 18 assumes that the PWM signal has a 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 will be used only to find the maximum bandwidth of the sine function in the three-phase system.

Merging Equation 8 with Equation 18 yields:

$(1 + A(i+1) \sin[f(i+1)]) > ½ (1 + A(i) \sin[f(i)])$ Equation 19

The first derivate of the sine function has its maximum at 0; hence, you shift Equation 19 into that area and seek the 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 part the maximum value around 0.

$(1 + A(i+1) \sin[-f_max]) > ½ (1 + A(i) \sin[+f_max])$ Equation 20

You can further simplify Equation 20 by setting the amplitude to its maximum 1, since it yields the lowest left part of the equation.

$½ + \sin[-f_max] > ½ \sin[+f_max]$ Equation 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:

$BW = f_pwm\ 40/360 = f_pwm / 9$ Equation 22

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

18.3 Period Range

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 percent. The relationship between the width and duty cycle is given in Equation 1.

$d = w / (P+1)$

Merging Equation 1 with Equation 4 yields:

$0 < [P + (P+1) (d(i) – d(i+1)) / 2] < 256$ Equation 23

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

$P < 218$ Equation 24

In the Example Code, P equals 199 and the PERIOD register will not suffer from overflow.

19 Example Code

The example code in Appendix A: TFPWM Header File, Appendix B: TFPWM Source Code, and Appendix C: Example Main shows a three-phase sine generator with a PWM output frequency of 15 kHz. Within the entire sine period, 300 samples are counted, and *P* = 199. This results in a three-phase output frequency of 50 Hz. All PWM8 User Modules use a VC2 clock, which has divisor=8. To see the three-phase output, simply apply RC networks to pins P1[2], P1[3], and P1[4] 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 the necessary API functions to correctly drive the three-phase generator:

- 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 the 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 twice when a change in pulse width occurs. If the stability criteria are 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 startup 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.

20 Summary

This application note described a general algorithm for generating an arbitrary function, with some limitations, through a three-phase generator. It also provided a detailed analysis of 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, that is, bandwidth limited, to achieve all the specified criteria.

A 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 immediatelly. */
extern unsigned char pwm_load_flag;


/* function protoypes */

/* 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
```

B Appendix B: TFPWM Source Code

```
/* Part specific constants and macros */
#include <m8c.h>

/* PSoC API definitions for all User Modules */
#include "PSoCAPI.h"

/* This file contains APIs that use to generate Three-Phase Sine Wave. */
#include "tfpwmapi.h"

/* This file contains functions that use to generate Three-Phase Sine Wave.
*/
#include "tfpwm.h"

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

/* Table of Half-Wave Symetric 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_calcwidth(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 analyzation:

      IRQ_COUNT   PWM_UPLOAD  CYCLES
      ---------------------------------------
      0                 -               27+35+43 = 105
      1                 -               27+43    = 70
      2                 YES             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
*****************************************************************************
/

/*****************************************************************************
Interrupt Handler:
Interrupt vectors from all PWM blocks should points single ISR tfpwm_isr()
*****************************************************************************
/
#pragma interrupt_handler tfpwm_isr
void tfpwm_isr(void)
{
      static unsigned char irq_count=0;

      if (irq_count==0)
      {
```

```
            PWM8_1_COMPARE_REG = bpwm1;
            PWM8_2_COMPARE_REG = bpwm2;
            PWM8_3_COMPARE_REG = bpwm3;
      }
      else if (irq_count==1)
      {
            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 unsuccessfull, 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==2)
            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_calcwidth(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_calcwidth(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_calcwidth(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_calcwidth(sine_phase, amplitude);
```

```
        period1 += (pwm1 >> 1);

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

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

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

C Appendix C: Example Main

```
/* Part specific constants and macros */
#include <m8c.h>

/* PSoC API definitions for all User Modules */
#include "PSoCAPI.h"

/* This file contains functions that use to generate Three-Phase Sine Wave.
*/
#include "tfpwm.h"

void main(void)
{
	/*Uncomment this line to enable Global Interrupts*/
	// M8C_EnableGInt ;

	/* Phase variable to select phase diffrance between three Sine waves */
	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();
}
```

D Appendix D: PSoC Configuration

The Global Resources settings are shown in Figure 17. Global Resources Setting.

Figure 17. Global Resources Setting

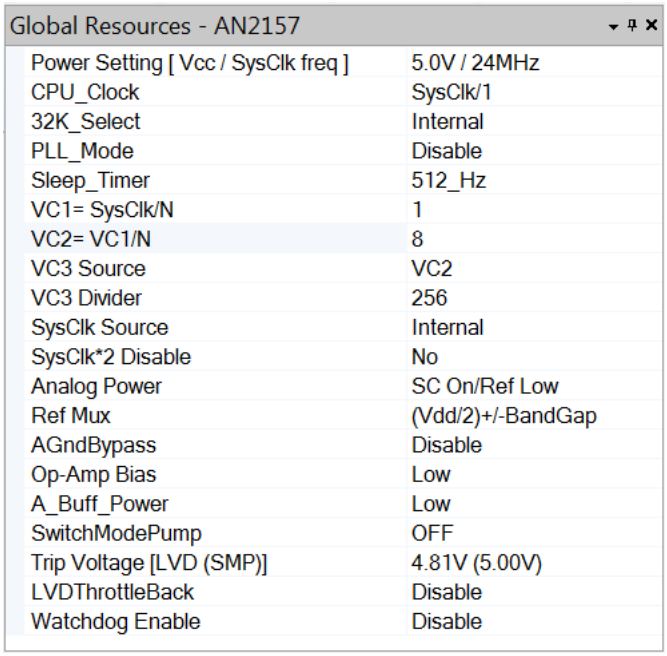


Figure 18 Shows the settings for parameters of PWM8_1 user module. PWM8_2 and PWM8_3 user modules should have same parameter settings except CompareOut is routed to Row_0_Output_3 and Row_0_Output_0, respectively.

Figure 18. PWM8_1 User Module Settings

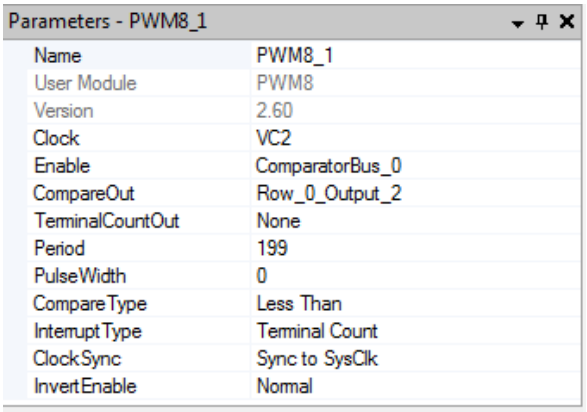
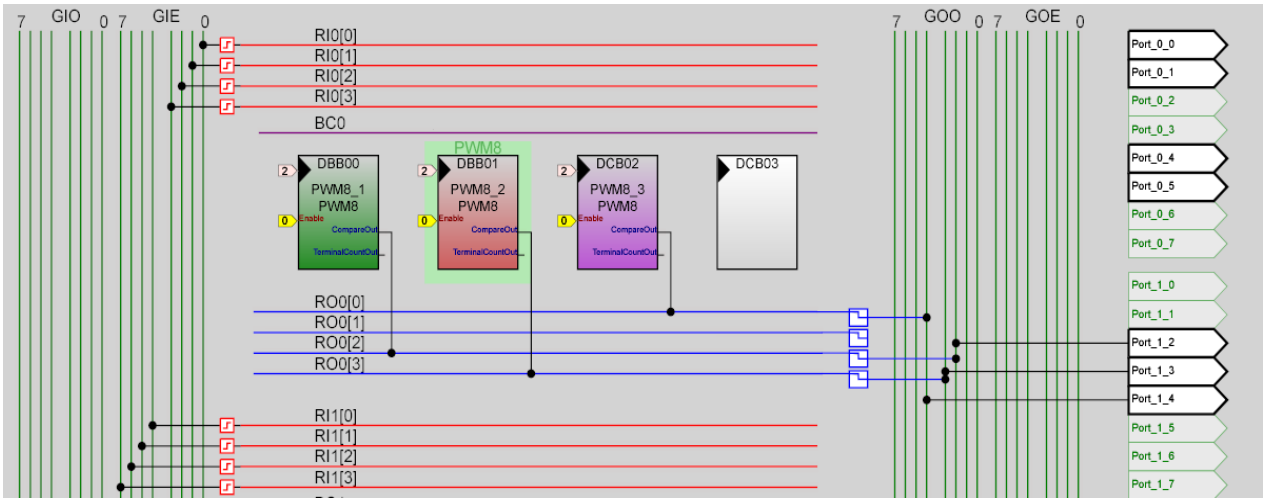


Figure 19 Shows three PWM8 User Modules are placed in highest interrupt priorities digital blocks.

Figure 19. PWM8 User Modules Placement and Pinout Connection



Document History

Document Title: AN2157 - Analog Three-Phase Sine Wave Generator

Document Number: 001-35339

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1536344	SSFTMP4	10/03/2007	Recataloged application note.
*A	3250924	UDAY	05/06/2011	Updated Project and Software Version to PSoC® Designer™ 5.1. Updated template.
*B	4392066	RICA	05/28/2014	Updated in new template. Completing Sunset Review.
*C	4778510	ASRI	06/24/2015	Added example code section Updated project for PSoC Designer 5.4 Sunset update Updated template

Worldwide Sales and Design Support

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

Products

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

PSoC® Solutions

psoc.cypress.com/solutions

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

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

PSoC is a registered trademark and PSoC Designer is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

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

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

© Cypress Semiconductor Corporation, 2007-2015. 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.