SVPWM simple implementation guide

Maciej Radosław Brzycki

There are many available publications describing the SVPWM algorithm. There are also many example implementations on GitHub. However, you may notice that there is a lack of supporting material showing how to combine these two worlds - theory and implementation, and this document is the answer to that. It shows the way starting from the theoretical side of SVPWM, successively implementations in C, and ending with the configuration of PWM on specific pins along with deadtimes.

This manual provides step-by-step advice on how to implement a simple SVPWM algorithm. The operation of the algorithm is explained, and the behavior of the time and voltage vectors with example parameters is shown - this is intended to make debugging easier for the person implementing the algorithm. The implementation (along with the physical PWM signal and deadtimes) on two popular microcontrollers - the F28379D from the **C2000** series from **Texas Instruments** and the H745ZI-Q with **cortex M7/M4** core from **STM** - is presented.

Content

[SVPWM control – basics 2](#_Toc183439791)

[SVPWM C implementation 3](#_Toc183439792)

[Alfa Beta and DQ transformation 3](#_Toc183439793)

[Rotating vector position 4](#_Toc183439794)

[Calculating time vectors 5](#_Toc183439795)

[simulation tests 7](#_Toc183439796)

[Output voltage and current 7](#_Toc183439797)

[Time Vectors 8](#_Toc183439798)

[Transistor gate signal 8](#_Toc183439799)

[Microcontroller implementation 9](#_Toc183439800)

[Texas Instruments C2000 9](#_Toc183439801)

[Timer configuration 9](#_Toc183439802)

[PWM configuration 10](#_Toc183439803)

[Deadtime configuration 12](#_Toc183439804)

[STM32 13](#_Toc183439805)

[Timer configuration 13](#_Toc183439806)

[PWM configuration 13](#_Toc183439807)

[Deadtime configuration 14](#_Toc183439808)

[Result verification 15](#_Toc183439809)

[REFERENCES 16](#_Toc183439810)

# SVPWM control – basics

SVPWM works by representing the three-phase voltage of an AC motor as a single rotating vector in a two-dimensional plane, called the space vector. Instead of directly generating sinusoidal waveforms for the motor phases, SVPWM calculates which combinations of the inverter's switching states can best approximate the desired space vector at any given time. The method divides one cycle into six sectors and selects two active vectors (producing voltage) and one zero vector (no voltage) within each sector to create the desired output. By adjusting the timing (duty cycle) of these vectors, SVPWM achieves precise control over the motor's voltage and current, resulting in smoother operation, reduced harmonics and less power losses. SVPWM can produce up to 15% more voltage compared to traditional sinusoidal methods from the same DC link voltage [3].

A hexagon with numbers and lines

Description automatically generated

Figure 1 Position sectors of the output voltage vector of the inverter [1]

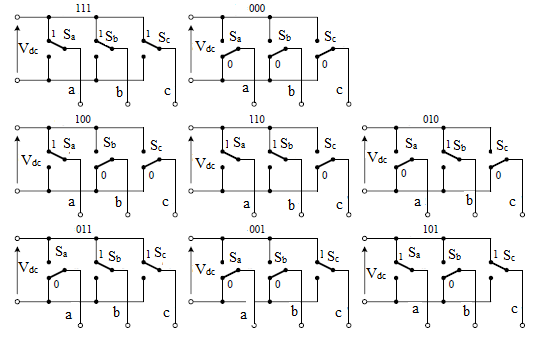


Figure 2 Eight Possible Switching states of Voltage Source Inverter [2]

Table 1 Sectors and the corresponding values of the components of the output voltage vector of the inverter

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Upper Transistor states (T1, T2, T3) | 100 | 110 | 010 | 011 | 001 | 101 | 000 | 111 |
| Down Transistor states  (T4, T5, T6) | 011 | 001 | 101 | 100 | 110 | 010 | 111 | 000 |
| Vα vector voltage |  |  |  |  |  |  | 0 | 0 |
| Vβ vector voltage |  |  |  |  |  |  | 0 | 0 |

# SVPWM C implementation

## Alfa Beta and DQ transformation

To prepare the SVPWM to work with the most common type of drive control (FOC), this algorithm takes input data in the form of d, q components and phase angle θ.

**Note** - if you want to independently control the amplitude of the generated voltage and frequency, it is possible by giving the q component a value of 0 and giving the d component the desired amplitude value. The frequency is set by controlling theta angle.

Since the target SVPWM algorithm works in alpha and beta axes, it is necessary to make transition from the d, q to alpha beta system.

The transformation dq to alfa beta in many papers is expressed in matrix form [4, 5, 6]:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

and it corresponds to the two equations:

|  |  |  |
| --- | --- | --- |
|  |  | (2) |
|  |  | (3) |

C implementation:

A black background with green and white text

Description automatically generated

## Rotating vector position

The algorithm calculates the current position of the voltage vector in the alpha beta axis. The floor function assigns it to one of six sectors. Depending on the current sector, the algorithm determines the two voltage vectors forming this sector (V1 and V2) and assigns the corresponding transistor states (Figure 1). V1 is the vector associated with the values corresponding to the current sector, and V2 with the values corresponding to sector + 1.

C implementation – determination of current sector:

A screenshot of a computer program

Description automatically generated

C implementation - determination of transistor states associated with the vector:



A screen shot of a computer screen

Description automatically generated

A computer screen with white and blue text

Description automatically generated

## Calculating time vectors

The vector U0 continuously spins at a given frequency and the algorithm in real time calculates the changing lengths of time vectors. The amplitude of the U0 vector is equal to the geometric sum of the given amplitudes of the d and q components. If the q component is equal to 0, the amplitude of the U0 vector is equal to the d component.

The vectors Uα1​ and Uβ1​ define the components of the first resultant voltage vector αβ (V1​). The vectors Uα2​ and Uβ2​ contain the component values of voltages for the next vector (V2​). The SVPWM modulation operates in such a way that, using two adjacent active vectors switched on for specific durations (t1, t2,) it is possible to generate a voltage vector located between these vectors (U0​).

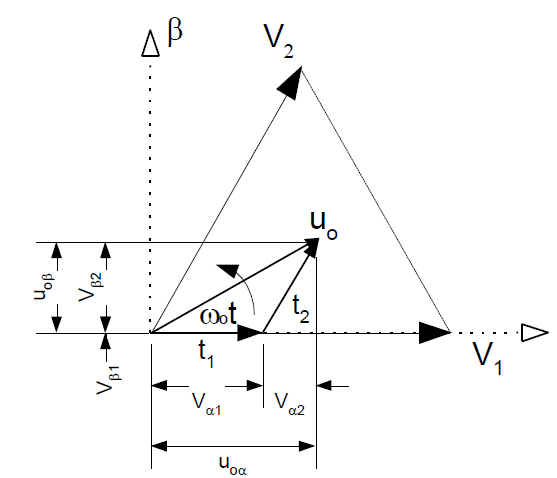
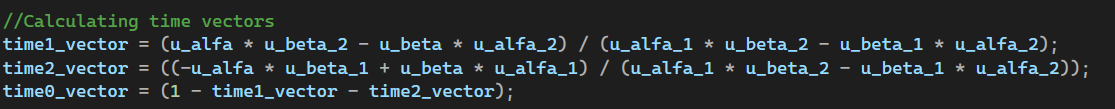


Figure 3 Determination of the component vectors of time and voltage

Formulas for the relative values of time vectors - calculated relative to one switching period. To get the actual values of these vectors, multiply them by the inverse of the switching frequency (Timp ) of the algorithm (for example, for 10kHz it would be 0.0001).

|  |  |  |
| --- | --- | --- |
|  |  | (4) |
|  |  | (5) |
|  |  | (6) |

C implementation – time vector calculation:



Adding the component vectors (*PWM\_Vector*) to form the resultant vector (*duty\_cycles*):

A number and equation on a black background

Description automatically generated

A black background with white text

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Whole c-code can be found in *SVPWM.c* and *SVPWM.h* files in GitHub repository.**

<https://github.com/MRadekTCZ/SVPWM-simple>

A screenshot of a computer

Description automatically generated

# simulation tests

Code written in C can be directly verified by simulation, for example, in plecs or matlab. The c-script SVPWMf is a main function called in microcontroller with 10kHz frequency. Symmetrical PWM block simulates work of PWM peripheral in µC.

A diagram of a circuit

Description automatically generated

Figure 4 Example SVPWM simulation schematic in Plecs

## Output voltage and current

Phase voltage (blue) has two levels (2/3 Udc and 1/3 Udc). It changes with PWM switching frequency. Moving average of this voltage should create sinusoidal signal with amplitude equal to 0.866 of set Ud voltage. If load contains some inductance, the current will be filtered by it, so the load current will also be similar to sinusoid without additional averaging.

A diagram of a waveform

Description automatically generated

Figure 5 Current and Voltage measured on Load (L = 0.01H, R = 10Ω)

## Time Vectors

Calculated time vectors are relative regarding one switching period. Their value can be interpreted as duty cycle for particular PWM period. Sum of the three time vectors t0, t1, t2 at any time should be equal to 1. The value of every time vector must be greater than zero and less than one. These requirements may not be met if the modulation index is too high (for a too high set amplitude).

A diagram of a graph

Description automatically generated

Figure 6 Time vectors

## Transistor gate signal

During a single cycle of the sine wave, a single transistor takes part in the switching in 2/3 part of the cycle. Averaged gate signal switching should look as one of the SVPWM voltage vectors (red).

A green graph with red lines

Description automatically generated

Figure 7 Gate signal (green) with averaged gate signal (red)

A diagram of a graph

Description automatically generated

Figure 8 Three averaged gate signals

# Microcontroller implementation

For any development environment, the first implementation step will be to import the SVPWM library (located in the GitHub repository). The following code should be placed in an interrupt of a certain frequency. The whole algorithm is fit inside one function with 4 input arguments: Ud and Uq component voltages, actual angle of rotating vector and dc link voltage.

A black background with colorful text

Description automatically generated

## Texas Instruments C2000

The program was developed in Code Composer studio. Before testing the code remember to change floating point mode optimization to *relaxed* for floating point operations support – this will significantly boost execution speed of this algorithm.

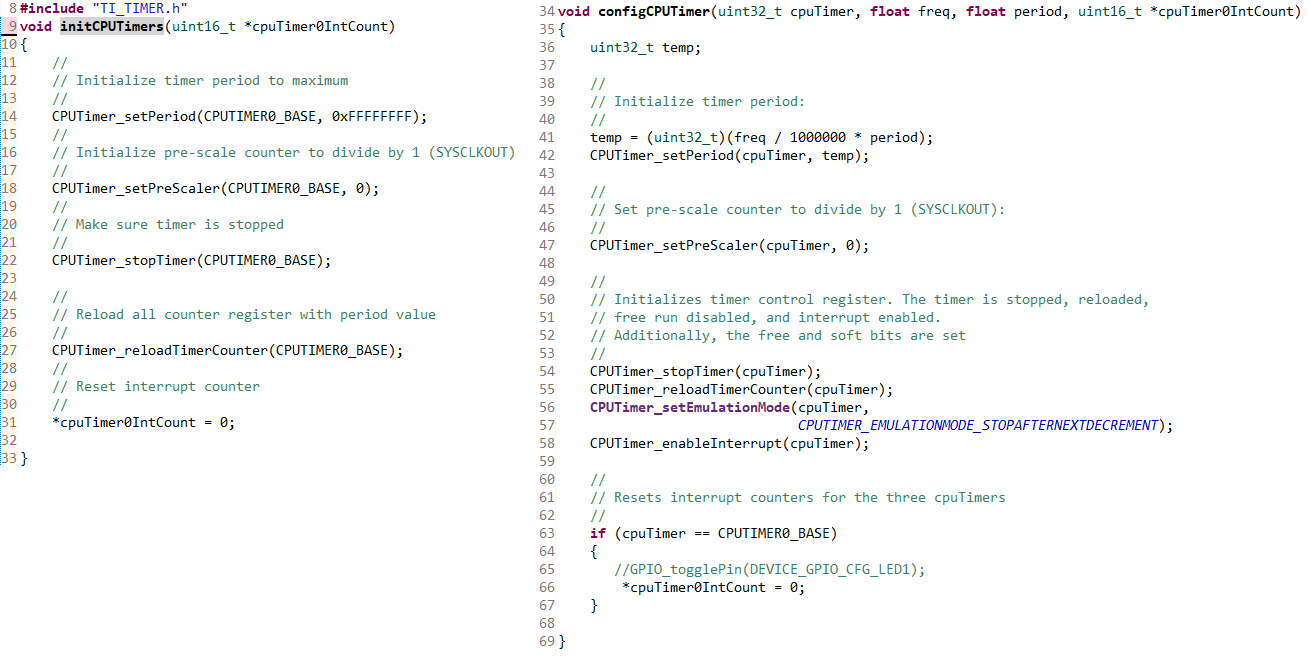
A screenshot of a computer

Description automatically generated

Figure 9 Changing floating point mode to relaxed

### Timer configuration

To configure the 10kHz timer there is a need to call the Init and Config functions. In the example given in GitHub, both these functions are in *TI\_TIMER.c* file.



In main.c file, before the infite loop, but after the Board\_init() function, a number of functions must be called:

Interrupt\_register(INT\_TIMER0, &cpuTimer0ISR);

initCPUTimers(&cpuTimer0IntCount);

configCPUTimer(CPUTIMER0\_BASE, DEVICE\_SYSCLK\_FREQ, 100, &cpuTimer0IntCount); // 10000Hz

CPUTimer\_enableInterrupt(CPUTIMER0\_BASE);

**Interrupt\_enable**(INT\_TIMER0);

CPUTimer\_startTimer(CPUTIMER0\_BASE);

100 in timer argument means, that main timer frequency (1Mhz) is divided by 100, so the timer frequency is 10kHz.

Timer interrupt call in Texas instruments microcontroller:

A computer code with black text

Description automatically generated

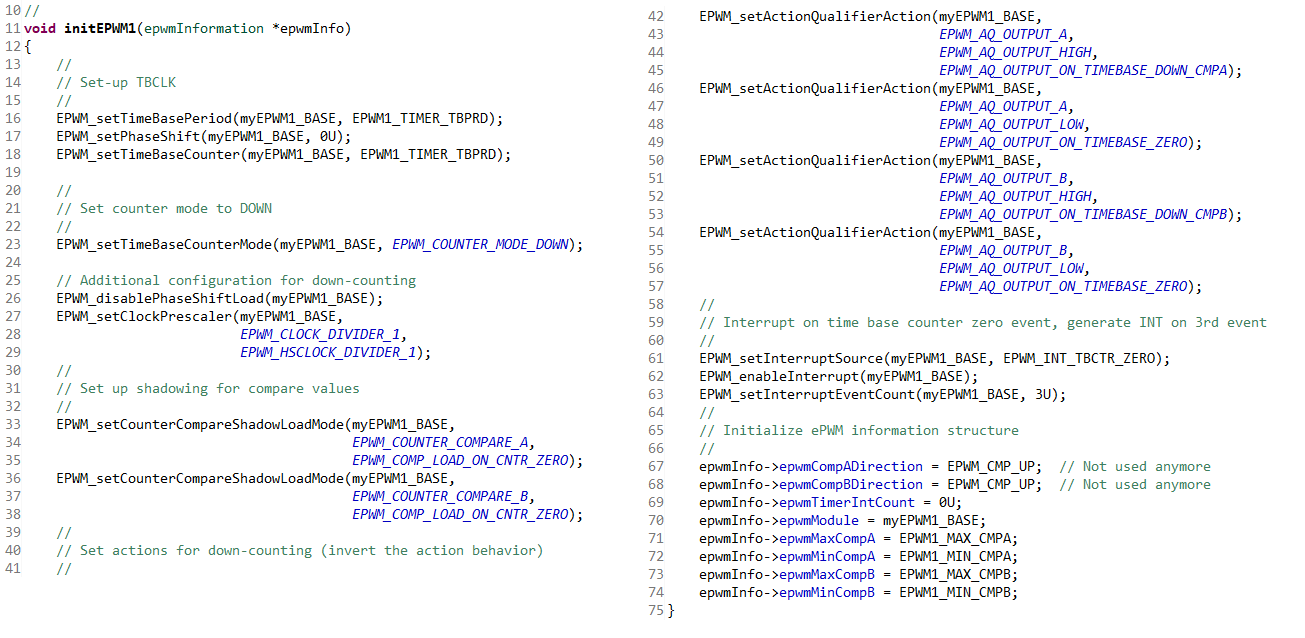
### PWM configuration

At the very beginning it is necessary to activate the pins associated with ePWM in the syscfg file.

A screenshot of a computer

Description automatically generated

Functions to configure the PWM signals are in TI\_PWM.c file. Three ePWM signals are initialized. Every of these signals has two channels A and B which are negated. The declaration of every ePWM is the same with only difference in ePWM base address.



A screenshot of a computer program

Description automatically generatedPwm signal initialization in main.c:

Interrupt\_register(INT\_EPWM1, &epwm1ISR);

Interrupt\_register(INT\_EPWM2, &epwm2ISR);

Interrupt\_register(INT\_EPWM3, &epwm3ISR);

initEPWM1(&epwm1Info);

initEPWM2(&epwm2Info);

initEPWM3(&epwm3Info);

Interrupt\_enable(INT\_EPWM1);

Interrupt\_enable(INT\_EPWM2);

Interrupt\_enable(INT\_EPWM3);

With every PWM interrupt *updateCompare* function is called. One of this function arguments is duty cycle of the vector calculated by SVPWM algorithm. This is how PWM counting time is changed every cycle.

A screen shot of a computer program

Description automatically generated

Before using PWM it is necessary to define proper counting periods. In *TI\_PWM.h* file there are defined values:

**#define** EPWM1\_TIMER\_TBPRD 10000U

**#define** EPWM1\_MAX\_CMPA 9999U

**#define** EPWM1\_MIN\_CMPA 0U

**#define** EPWM1\_MAX\_CMPB 9999U

**#define** EPWM1\_MIN\_CMPB 0U

*EPWM1\_TIMER\_TBPRD* is a value to which PWM counter counts in one period. Clock frequency of TMS-F28379D microcontroller is 200MHz. The highest possible PWM counting frequency is equal to main clock frequency divided by 2 (100Mhz). If *EPWM1\_TIMER\_TBPRD* is equal to 10000, then PWM timer will count to 10000 clock cycles in one period, so PWM switching frequency will be 10000Hz.

### Deadtime configuration

In *TI\_PWM.c* file there are multiple deadtime activation functions (eg. *setupEPWMOutputSwap, setupEPWMActiveHighComplementary, setupEPWMActiveLow*). For SVPWM the most suitable deadtime function is *setupEPWMActiveHighComplementary*.

A screenshot of a computer program

Description automatically generated

The value of deadtime prescaler for both falling and rising edge is equal to ***10***. It means that the chosen deadtime period is equal to 10 PWM clock cycles (100ns).

Activating deadtimes in *main.c* file:

setupEPWMActiveHighComplementary(myEPWM1\_BASE);

setupEPWMActiveHighComplementary(myEPWM2\_BASE);

setupEPWMActiveHighComplementary(myEPWM3\_BASE);

## STM32

Program and configuration were developed in STM cube IDE.

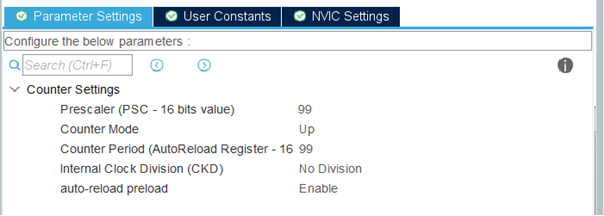
### Timer configuration

In this program, two timers are used - one for the interrupts associated with the main algorithm with a frequency of 10 kHz - TIM13 and the other for PWM generation (also with a switching frequency of 10kHz - TIM1). Before configuring the timers, it is necessary to find the clock frequency associated with the particular timer base. There are two main timer frequencies - APB1 and APB2 timer clocks. Check the documentation or configuration files with which frequency the currently configured timer is associated. For the STM32H745ZI microcontroller, Timer 13 is associated with the APB1 frequency (figure 1. in STN32H745 datasheet [7]).

A screenshot of a computer

Description automatically generated

Since the frequency of APB1 is 100 Mhz, after setting the prescaler to 99 (counting starts from 0 rather than 1) and the counting period to 99, the resulting frequency is 100000000/(100\*100) is 10kHz.



Be sure to set auto-reload preload to enable and allow interrupts in the NVIC panel

A blue and white flag

Description automatically generated

In the code, timer is activated with the command:



### PWM configuration

To initialize SVPWM control, 3 PWM channels will be necessary - each with two complementary outputs. The PWM counts pulses at APB1 frequency, and the counting period is 10000 pulses - the PWM switching frequency is 10 kHz. The count mode should be set as PWM mode 2 (it will matter later whether the deadtime is implemented as two enabled outputs or two disabled outputs). The polarity of CH and CHN should be set for both as HIGH (CHN initially is negated, so negating it here would give a double negation effect). For the same reason, the Idle state of both channels should be set as reset.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Be sure to initialize the complementary channels in the code in addition to the basic initialization of the PWM channels:

A screen shot of a computer program

Description automatically generated

### Deadtime configuration

Deadtime configured to last 20 clock cycles of APB1 (200ns)

A white background with black text

Description automatically generated

## Result verification

A screenshot of a computer

Description automatically generated

Figure 10 transistor gate signals overview - generated by microcontroller

A screen shot of a computer

Description automatically generated

Figure 11 Gate signal overview - zoom in

A screenshot of a computer

Description automatically generated

Figure 12 Deadtime on rising edge and falling edge

# REFERENCES

[1] A. Lewicki *Space Vector PWM. Effect of dead time on drive operation* - lab manual.

[2] Pradeep, Jayarama & Devanathan, Rajagopalan. (2015). *Adoption of Park’s Transformation for Inverter Fed Drive. International Journal of Power Electronics and Drive Systems*. 5. 366-373. 10.11591/ijpeds.v5i3.6849.

[3] N. Prakash, J. Jacob and V. Reshmi, "Comparison of DVR performance with Sinusoidal and Space Vector PWM techniques," *2014 Annual International Conference on Emerging Research Areas: Magnetics, Machines and Drives (AICERA/iCMMD)*, Kottayam, India, 2014, pp. 1-6, doi: 10.1109/AICERA.2014.6908196.

[4] Z. Wen and S. He, "Analysis of three-phase magnitude-phase detection method based on double dq transformation," *Proceedings of The 7th International Power Electronics and Motion Control Conference*, Harbin, China, 2012, pp. 340-344, doi: 10.1109/IPEMC.2012.6258869.

[5] M. Gonzalez, V. Cardenas and F. Pazos, "DQ transformation development for single-phase systems to compensate harmonic distortion and reactive power," *9th IEEE International Power Electronics Congress, 2004. CIEP 2004*, Celaya, Mexico, 2004, pp. 177-182, doi: 10.1109/CIEP.2004.1437575.

[6] V. M. Goswami and K. Vakharia, "High Performance Induction Machine Drive Using Rotor Field Oriented Control," *2019 International Conference on Intelligent Sustainable Systems (ICISS)*, Palladam, India, 2019, pp. 559-564, doi: 10.1109/ISS1.2019.8907959.

[7] STM32H745ZxI datasheet - https://www.st.com/resource/en/datasheet/stm32h745zg.pdf