UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'informazione e Matematica

Renewable Energy and Storage Systems Project

On

Sinusoidal Pulse Width Modulation (SPWM) for Three-Phase Inverter.

By

Aromose Qudus Ayinde

Submitted to: Prof. Carlo Cecati

PROGRAM: ELECTRIC VEHICLE PROPULSION AND CONTROL (E-PICO).

June, 2024.

EPiCo
Electric Vehicle Propulsion and Control

Table of Contents

List of Figures

Abstract

This project describes developing and implementing a three-phase inverter using the SPWM technique to enhance the performance and efficiency of renewable energy systems. The use of the SPWM technique is done to generate consistent amplitude pulses having different duty cycles. The output voltage closely resembles the target sinusoidal waveform, which results in a significant reduction in THD. The inverter model is developed and simulated using MATLAB Simulink, with FFT tools applied for harmonic analysis. Results show that the SPWM method reduces harmonic content, further ensuring stable and efficient inverter operation. This will be another advancement in inverter technology to integrate renewable energy sources more effectively into the power grid[1].

Introduction

Several renewable energy technologies, including solar power, have entered the realm of power generation. This is because the technology remains sustainable and abundant. However, integrating existing electrical grids poses an excellent need for tackling various technical challenges, primarily in power conversion and management. One of the most vital components for such integration is the inverter, which converts the DC power generated by solar panels into AC power that is fit for grid use[2].

The three-phase inverters have high utilization in high-power applications because of their ability to handle oversized loads and give stable AC output. These inverters find wide applications in many fields, including industrial applications, microgrid systems, and bidirectional energy storage systems. The performance and efficiency in using these inverters rely quite a lot on pulse width modulation techniques. SPWM is the most preferred method because it is simple and effective in controlling inverter output voltage and frequency[3].

In essence, SPWM provides pulses by comparing the sinusoidal reference signal with a high-frequency triangular carrier wave. These techniques ensure that the output voltage follows the reference sinusoidal waveform closely, hence reducing the generation of harmonic distortions. Inverter output harmonics can lead to conductor and transformer heating, possible system failures, and eventually result in efficiency reduction. Consequently, the minimization of THD remains very critical for the reliable operation of a three-phase inverter, for which reason this project is targeted at designing and implementing a robust SPWM-based three-phase inverter model toward the minimization of THD and overall performance improvement in renewable energy systems.

A simulation of the model is performed on MATLAB Simulink; the harmonic content in the output is analysed with FFT tools. The project will aim to show these crucial enhancements for the efficiency and stability of inverters through applying proper filtering techniques with SPWM, which is very helpful for the broad penetration of renewable energy technologies into high-power applications[3].

Objectives

The project mainly has the following objectives:

 SPWM-Based Inverter Model Design and Development

- Design a three-phase inverter model using the Sinusoidal Pulse Width Modulation (SPWM) technique.
- Develop the inverter model in MATLAB Simulink with due diligence for simulating its operational behaviour with fidelity.

 Lower THD

- Lower the Total Harmonic Distortion of inverter output voltage and current.

 Simulation and Analysis

- The model in the inverter is simulated with MATLAB Simulink and FFT analysis tools to carry out the harmonic analysis.
- The simulation results are documented, showing the improvements in harmonic performances.

 Filter Circuits Design

- Design appropriate LC filter circuits to further reduce the generation of higher-order harmonics in the inverter output.
- Add the filter circuits to the inverter model and determine their effectiveness for reducing harmonics.

 Lab Implementation

- SPWM signals designed through Texas Instrument TMS320F28379D DSP to drive the gate of the three-phase Inverter.

Methodology

Design of three-phase voltage source inverter

This design is done in a MATLAB/Simulink environment. The power converter is mainly used to produce an AC output waveform from a DC input supply. Fig. 1 shows an ideal three-phase inverter with six IGBTs acting as controlled switches. In this model, the desired output voltage can be regulated, and line currents can be sinusoidal at unity power factor. The inverter can be integrated into a grid by connecting filter circuits in between[4].

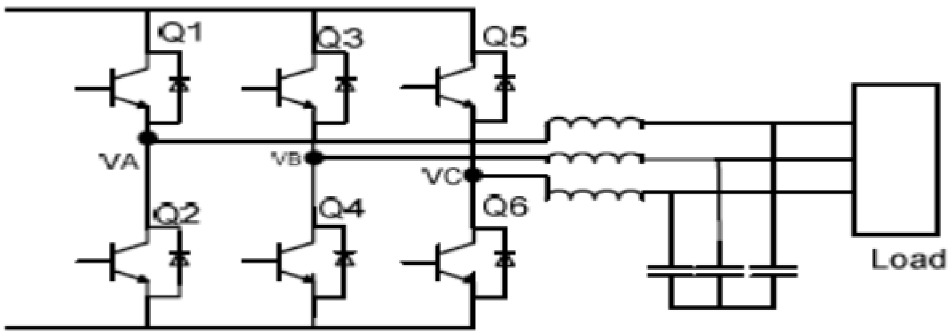


Figure 1:Three phase inverter model

The three-phase converter needs six switching topologies used in microgrid systems for power conversion. It is required to generate six PWM signals for the operation of three-phase inverters.

Depending on the type of output waveform, three-phase inverters that have a voltage waveform as the independently controlled AC output are called voltage source inverters (VSIs). These inverters are widely used as a voltage source with controllable voltage, phase, and frequency in industrial applications. In a three-phase VSI, simultaneous switching of any leg cannot be done as it will cause a short circuit. Similarly, inverter legs cannot be switched off simultaneously to avoid undefined line voltages and states [5]. In a VSI, voltage always has the same polarity, whereas the DC current polarity reverses, causing power reversal. The VSI has bidirectional flow of DC current, so converter valves need to be bidirectional, and as voltage does not reverse, reverse voltage capability is not needed in turn-off devices. This can be achieved by using IGBTs or MOSFETs with a parallel reverse diode [6].

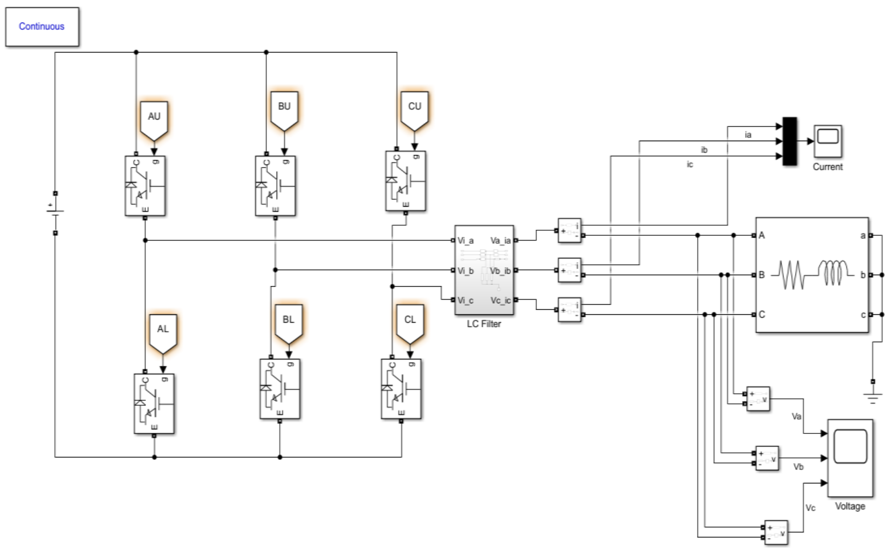


Figure 2: MATLAB Simulink model of voltage source inverter with Gate pulse triggering

In Figure 2, a MATLAB Simulink model of a three-phase voltage source inverter using IGBTs as the switching devices with gate pulse triggering is depicted. This is a 180º conduction mode VSI, and the switches are triggered based on the calculated duty cycles for each of them. The VSI is powered by a DC voltage source of 440 volts and is supplying power to a nonlinear RL load. The output voltage and current waveforms of the inverter are being analysed. Additionally, THD results are calculated using the FFT (fast Fourier transform) toolbox. At the DC input side, a capacitor is provided for the unipolar voltage. The value of the capacitor is sufficient to handle charging or discharging currents according to the converter switching sequence.

Total Harmonic Distortion

The presence of nonlinear devices in a power system network causes harmonic distortion. Nonlinear components are those whose current is not proportional to the voltage applied. The need to control harmonics is given in the IEEE 519-1992 standard [7]. Calculating the effective value of the harmonic component in a distorted waveform is known as total harmonic distortion or THD. THD analysis is important for measuring the quality of the output current and voltage of the inverter. Due to the presence of harmonics, the output waves can become non-sinusoidal.

$$THD_V = \frac{\sqrt{\sum_{h>1}^{hmax} V_h^2}}{V_1} \tag{1}$$

Where,

h is the order of harmonics,

$V_h$is the harmonic voltage,

$V_1$is the fundamental voltage.

Eq. (1) is used for calculation of THD in output voltage of the inverter model.

$$THD_I = \frac{\sqrt{\sum_{h>1}^{hmax} I_h^2}}{I_1} \tag{2}$$

Where,

I_h is the harmonic current
I_1 is the fundamental current
Eq. (2) is the equation used for calculation of THD in output current of the inverter [8].

In MATLAB simulation, FFT toolbox will be used for the calculation of THD in output voltage and current of the three phase VSI Simulink model developed.

SPWM Techniques and Its Implementation in Three-Phase VSI

The output of inverters can be controlled by using pulse width modulation (PWM) techniques. The PWM method adjusts the ON and OFF periods of inverter IGBTs. Among various PWM techniques, the SPWM technique is preferred because it allows direct control of the inverter output voltage and frequency based on the sine functions used. In this method, the pulse amplitude remains constant, while the duty cycles vary for each period. By modulating the width of pulses, the inverter output voltage can be controlled, and total harmonic distortion (THD) can be reduced [9].

In the SPWM technique, the pulse signal is generated by comparing the sinusoidal reference signal with a triangular carrier signal of cut-off frequency fc. The switching devices are turned on whenever the reference sinusoidal signal exceeds the carrier triangular wave [10]. The fundamental component frequencies and magnitudes on the line side can be varied by adjusting the modulation signal frequency and magnitude [11].

In a three-phase voltage source inverter, three phase-shifted sine waves are used as reference signals, with a 120° phase difference, and the desired output voltage frequency is selected.

These signals are compared to a carrier signal with a very high frequency. The waveforms of the sine and triangular wave comparison, as well as pulse generation, are shown in Fig. 3 [12].

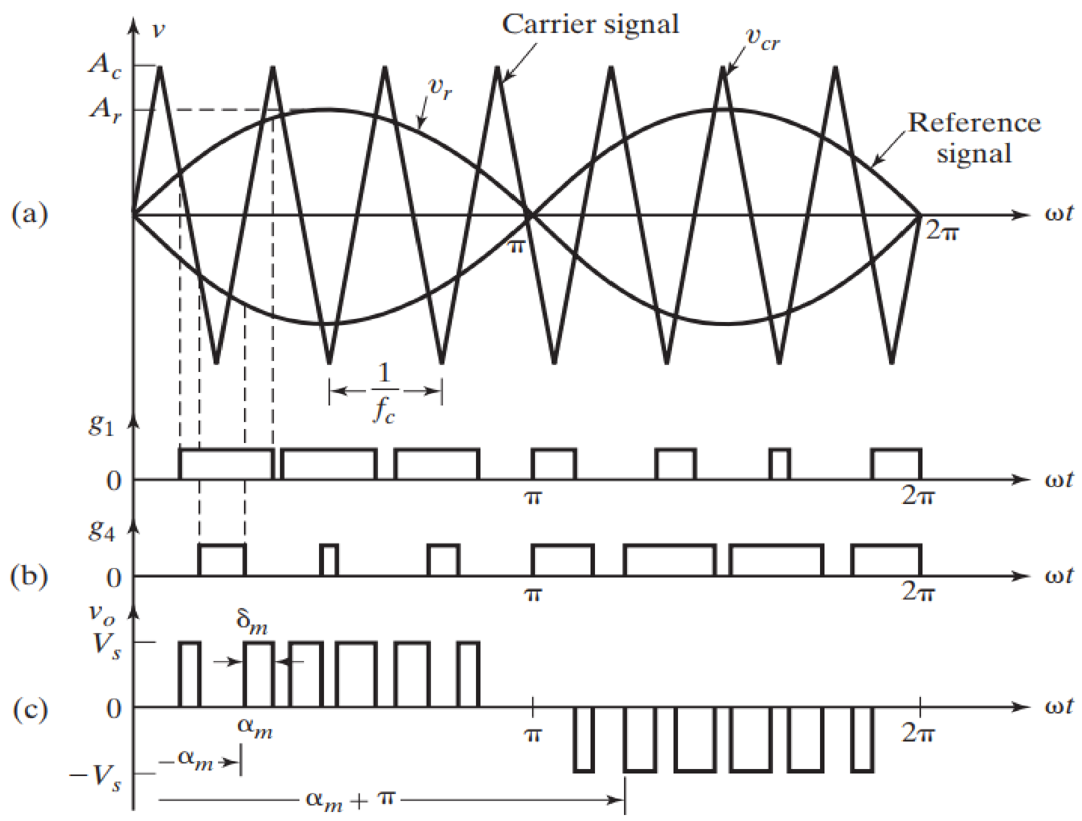


Figure 3:SPWM method: (a) Sinusoidal and triangular comparison; (b) Generation of switching pulses.

[13]

The modulation index or MI of SPWM generator is the ratio of reference signal amplitude and carrier signal amplitude as given in eq. (3) [14].

$$M = \frac{A_r}{A_c} \qquad (3)$$

Where,
M is the modulation index of the SPWM generator,
A_r is reference signal amplitude,
A_c is carrier signal amplitude.

It's important to remember that the modulation index (MI) of the SPWM generator should be maintained between 0 and 1. A higher modulation index value results in lower total harmonic distortion (THD).

In the proposed three-phase voltage source inverter, the SPWM technique is being utilized. As illustrated in Fig. 4, a three-phase, six-pulse bridge-type SPWM generator with a carrier frequency of 10 kHz is employed. A modulation index (MI) of 1 is chosen to achieve the best output results from the VSI model when operating with a nonlinear load.

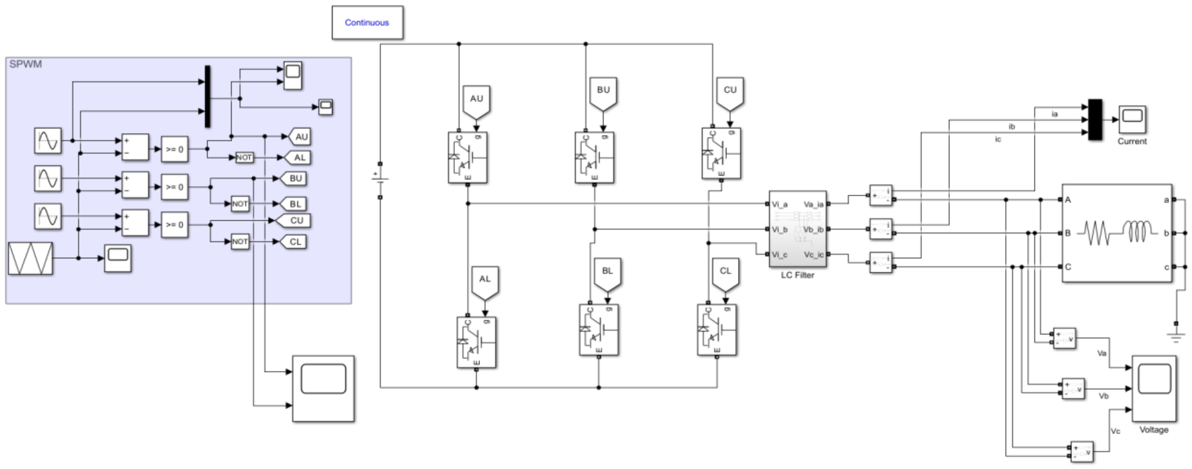


Figure 4: Three-phase Voltage Source Inverter with Sine Pulse Width Modulation generator

Advantages of using SPWM techniques in VSI [15]:

- The implementation and control of the method are easy.
- Linear operation controls the outputs, leading to less power dissipation.
- There is no degradation of the device due to aging or temperature variation effects.
- The method is compatible with digital microprocessors.
- Switching losses are lower, and DC power can be utilized more effectively, resulting in higher output voltage with the same DC input.
- There is lower reduction of lower-order harmonics in the output voltage and current of the VSI.

Design Of LC Filter Circuit

In a three-phase VSI model, to reduce higher-order harmonics, it is necessary to design an LC filter circuit. When using the SPWM technique, lower-order harmonics can be reduced, so the LC filter should be designed as a low-pass filter to reduce higher-order harmonics economically. The inductor used in the filter circuit should have a current rating greater than or equal to the maximum value of the inverter output current. A smaller inductor can be used to reduce ripples in the output. If a large inductor is chosen, a similar capacitor will be needed to counterbalance the substantial voltage drops shown in equation 4 due to load transients [16].

$$\frac{V_{ripple\,max}}{V_{ac}} = \frac{X_c}{X_c+X_L} \qquad (4)$$

$$X_L = 2\pi f L \qquad (5)$$

$$X_c = \frac{1}{2\pi f C} \qquad (6)$$

Where,

$V_{ripple\,max}$ is maximum ripple output of the filter

V_{ac} is the maximum ripple output of the inverter,

X_L is the inductor reactance,

X_c is the capacitor reactance,

L is inductance,

C is capacitance,

f is frequency.

The cut-off frequency of the low-pass LC filter is : $f_c = \frac{1}{2\pi\sqrt{LC}}$

The Characteristic impedance of the filter is: $Z = \sqrt{\frac{L}{C}}$

The low pass LC filter circuit is suitable for the network where load impedance across C is high and above switching frequency [17].

The LC low pass filter circuit connected in T configuration and designed in MATLAB/simulink for harmonic reduction of the three phase VSI model is shown in Fig. 5.

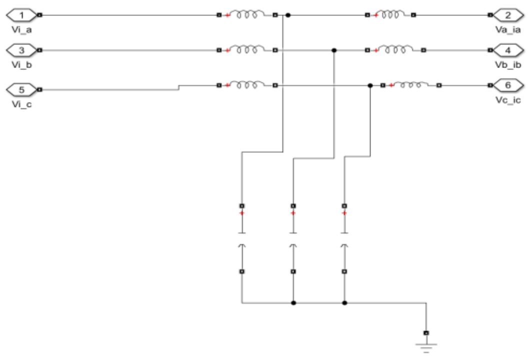


Figure 5: Low pass LC filter circuit in T configartion

Lab Implementation of SPWM

The lab implementation of the SPWM-based three-phase inverter encompasses designing the SPWM signals, selecting suitable hardware components, and integrating and testing the system to validate the MATLAB Simulink simulation results.

Design of SPWM Signals

SPWM signals are generated using the Texas Instrument TMS320F28379D DSP. This DSP creates high-frequency triangular carrier signals and sinusoidal reference signals, which are compared to produce PWM pulses for the IGBTs in the inverter.

SPWM Signal Generation:

- The DSP generates three sinusoidal reference signals with a 120° phase shift.
- A high-frequency triangular carrier signal is produced.
- The reference signals are compared with the carrier signal to generate PWM pulses.

Programming the DSP:

- Code Composer Studio (CCS) is used to program the DSP.
- The SPWM algorithm, including signal generation and comparison, is implemented in the DSP code.
- The code is tested and debugged in a controlled environment to ensure accurate SPWM signal generation.

Hardware Setup

The hardware setup includes several key components:

Inverter Module:

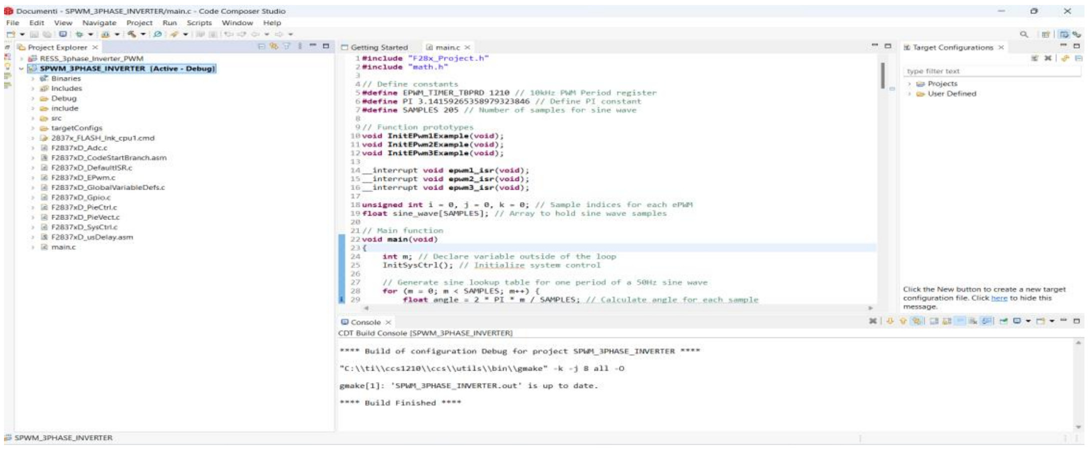
A three-phase inverter module with IGBTs is connected to a DC power supply, such as a solar panel or battery system.

Gate Driver Circuit:

The DSP-generated PWM signals are fed into a gate driver circuit, which amplifies these signals to the required voltage levels for the IGBTs.

Filtering Circuit:

An LC filter circuit, designed according to simulation specifications, is connected to the inverter output to reduce higher-order harmonics and smooth the output waveform.



```c
#include "F28x_Project.h"
#include "math.h"

// Define constants
#define EPWM_TIMER_TBPRD 1210 // 10kHz PWM Period register
#define PI 3.14159265358979323846 // Define PI constant
#define SAMPLES 205 // Number of samples for sine wave

// Function prototypes
void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);

__interrupt void epwm1_isr(void);
__interrupt void epwm2_isr(void);
__interrupt void epwm3_isr(void);

unsigned int i = 0, j = 0, k = 0; // Sample indices for each ePWM
float sine_wave[SAMPLES]; // Array to hold sine wave samples

// Main function
void main(void)
{
    int m; // Declare variable outside of the loop
    InitSysCtrl(); // Initialize system control

    // Generate sine lookup table for one period of a 50Hz sine wave
    for (m = 0; m < SAMPLES; m++) {
        float angle = 2 * PI * m / SAMPLES; // Calculate angle for each sample
```

```
**** Build of configuration Debug for project SPWM_3PHASE_INVERTER ****

"C:\\ti\\ccs1210\\ccs\\utils\\bin\\gmake" -k -j 8 all -O

gmake[1]: 'SPWM_3PHASE_INVERTER.out' is up to date.

**** Build Finished ****
```

Figure 6: Code Composer Studio Environment for generating the SPWM signals

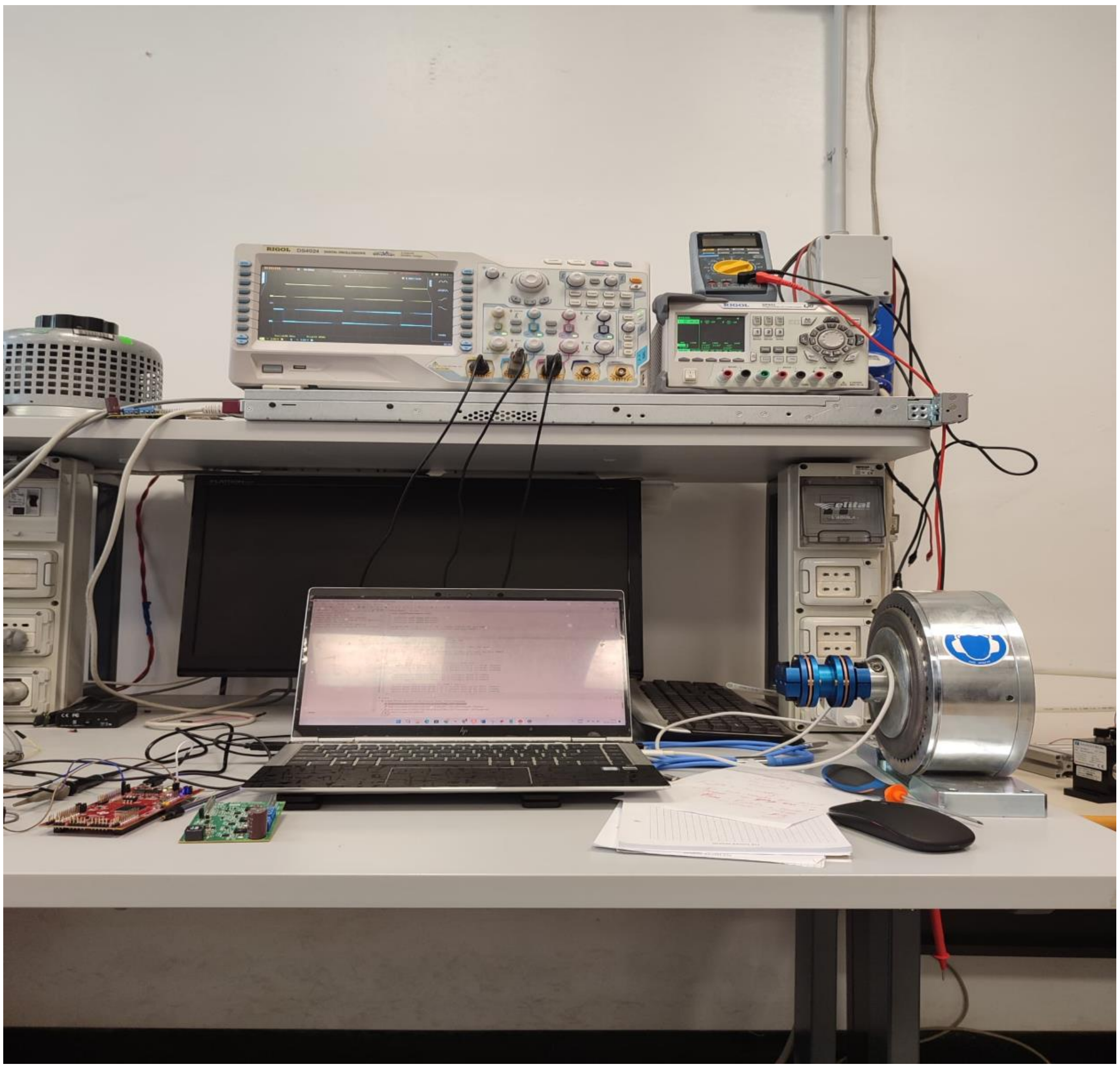


Figure 7: Lab Set up

Results and Discussion

The output voltage and current waveforms of three phase voltage source inverter with harmonic analysis using FFT toolbox with Gate pulse triggering and SPWM triggering are shown below using MATLAB simulink. The data considered in this model is as follows[3]:

$Fundamental\ frequency\ =\ 50\ Hz$

$Modulation\ index\ =\ 1$

$DC\ input\ voltage\ =\ 440\ V$

$Carrier\ Frequency\ =\ 10\ kHz$

$LC\ filter\ Cutoff\ Frequency\ =\ 5\ kHz$

The three phase VSI model triggered using gate pulses and phase delay calculated for 180º conduction mode, the output current and voltage waveforms are shown in Fig. 8 and 9. The waveforms are distorted because of the presence of harmonics and amount of total harmonic distortion has been calculated and shown in Fig. 10 and 11 by performing FFT analysis.

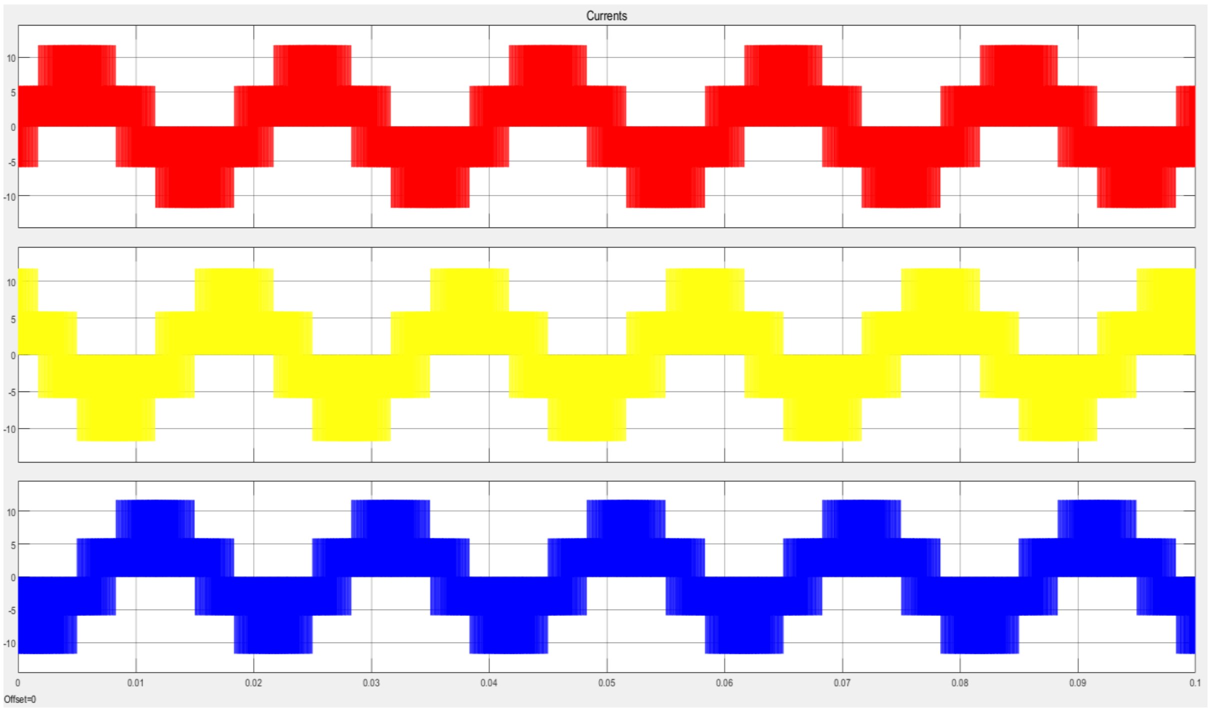


Figure 8: output current waveform with harmonics

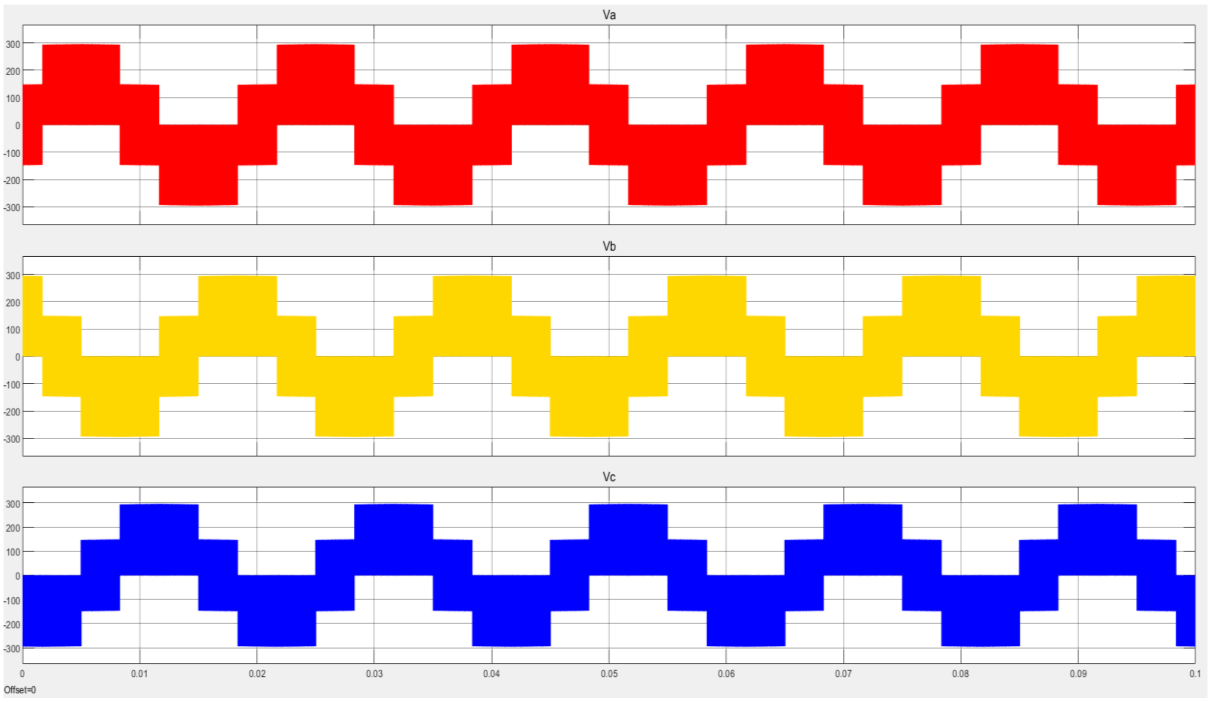


Figure 9: output Voltage waveform with harmonics

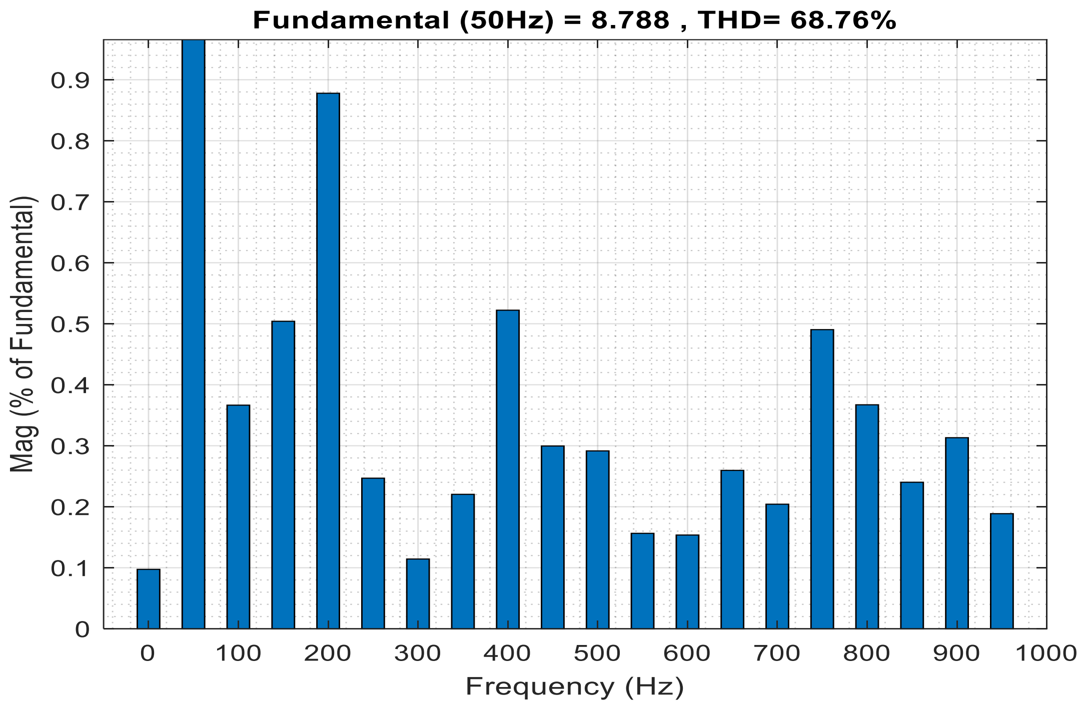


Figure 10: output current THD analysis

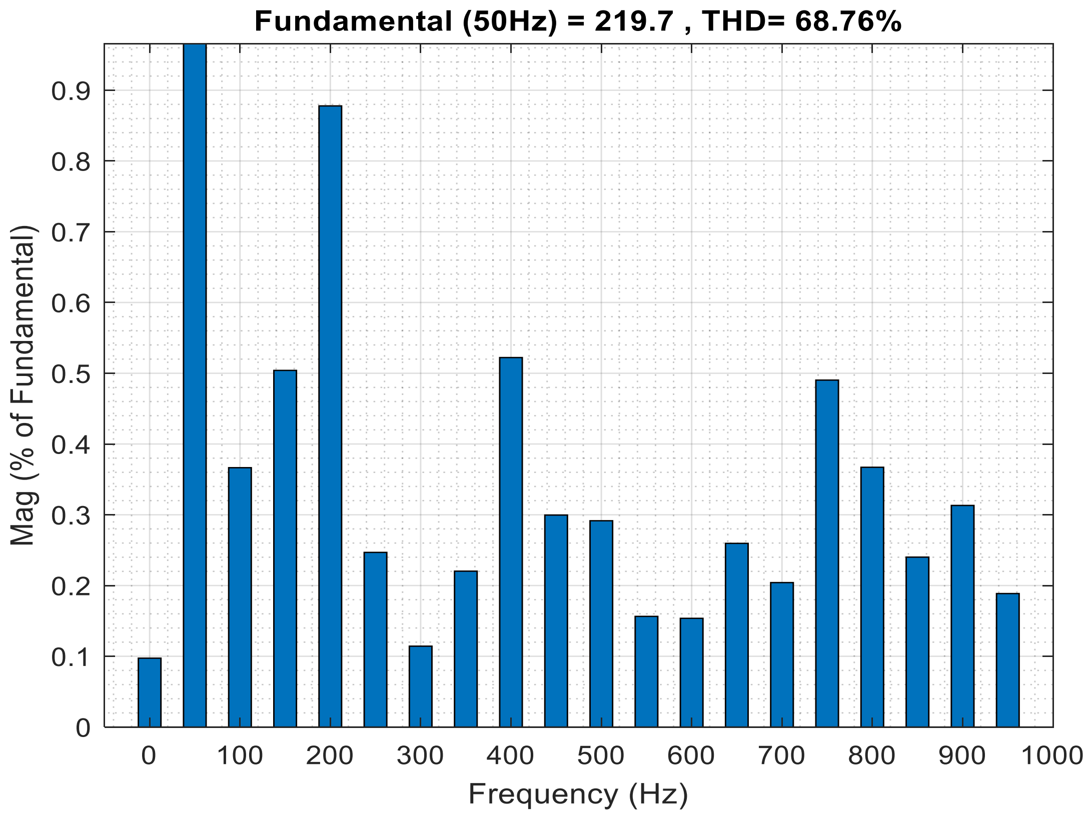


Figure 11: output voltage THD analysis

From Fig. 10 and 11 the THD present in output current of the three phase VSI is 68.76% and in output voltage of VSI is 68.76% respectively, when it is only gate pulse triggered.

For reduction of harmonics and THD in output current and voltage of three phase VSI SPWM generator circuit and LC filter circuit has been connected with the VSI model as shown in Fig. 4 and now the waveforms of output current and voltage waveform of VSI with SPWM generator and low pass LC filter is being analysed and shown in Fig. 12 and 13. THD of current and voltage outputs of VSI, using FFT toolbox in MATLAB are shown in Fig. 14 and 15.

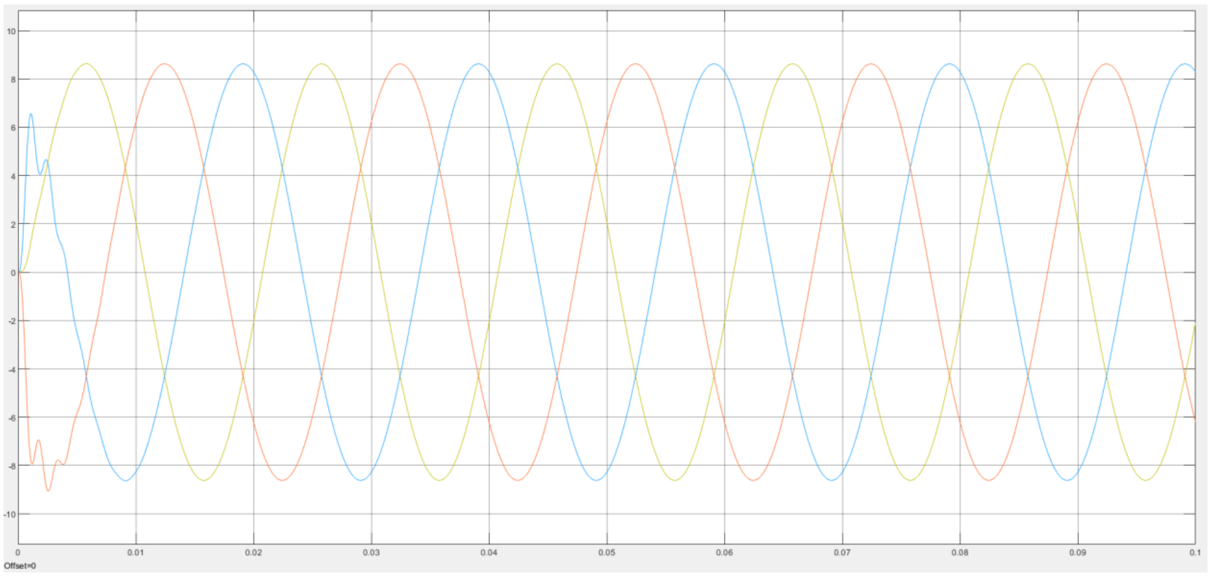


Figure 12: output current waveform with reduced-harmonics after filter application

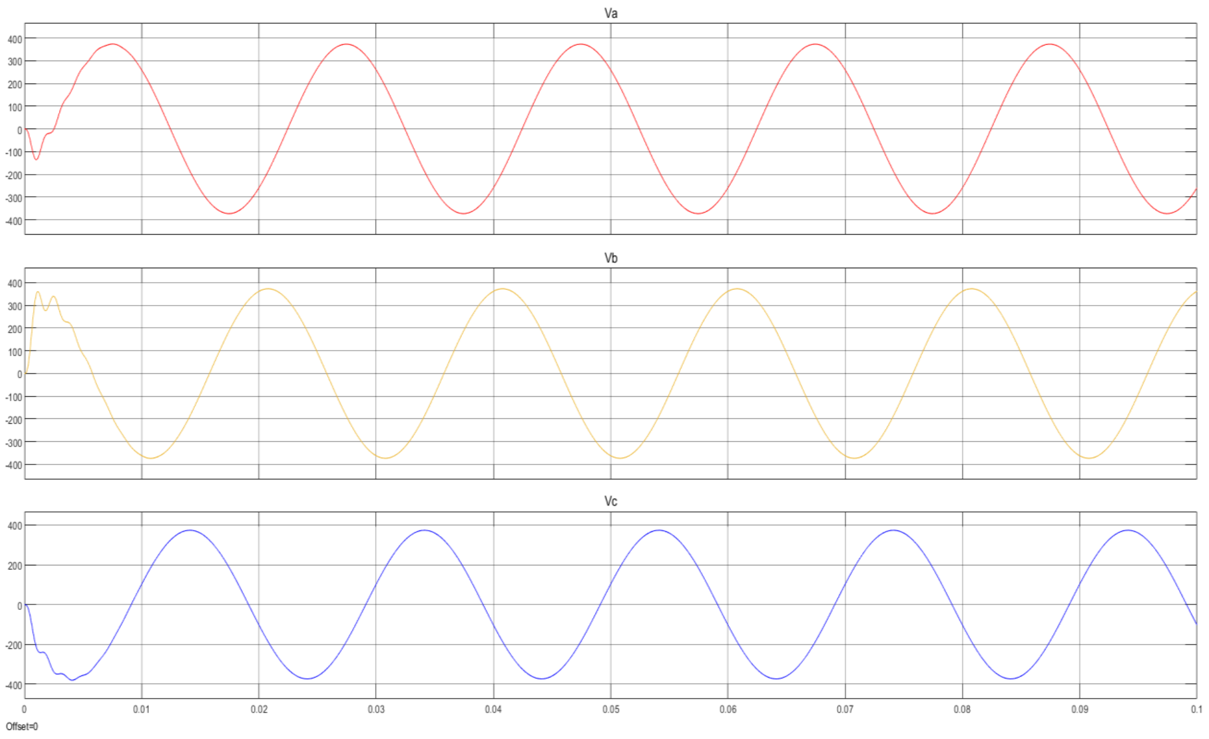


Figure 13: output voltage waveform with reduced-harmonics after filter application

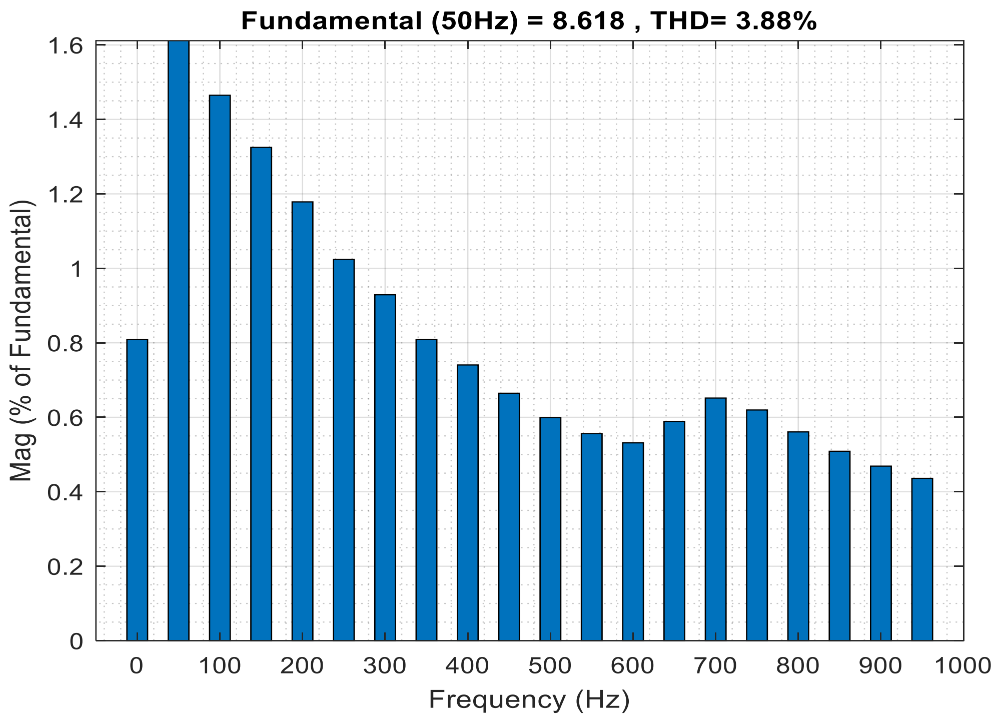


Figure 14: output current THD analysis after filter application

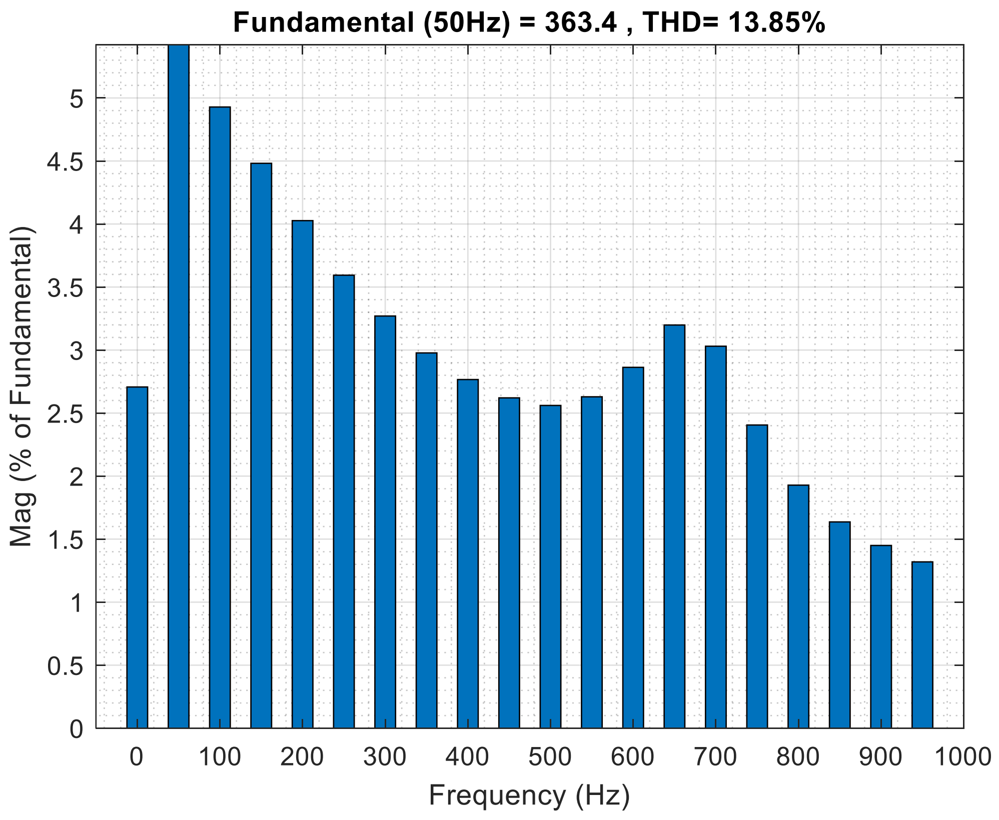


Figure 15: output voltage THD analysis after filter application

As shown in Fig. 14 and 15 the total harmonic distortion of the three phase VSI with SPWM generator and LC filter connection is 3.88% in output current waveform and 13.85% in output voltage waveform. So, with the use of SPWM generator and LC filter the total harmonic distortion has been largely reduced.

The SPWM signal was generated using DSP TMS320f28379D and the code to generate these signals was written on Code Composer Studio and the generated signal was verified using the oscilloscope. Fig 16 shows the generated PWM signal that was used to drive the gate of the IGBTs of the three-phase inverter.

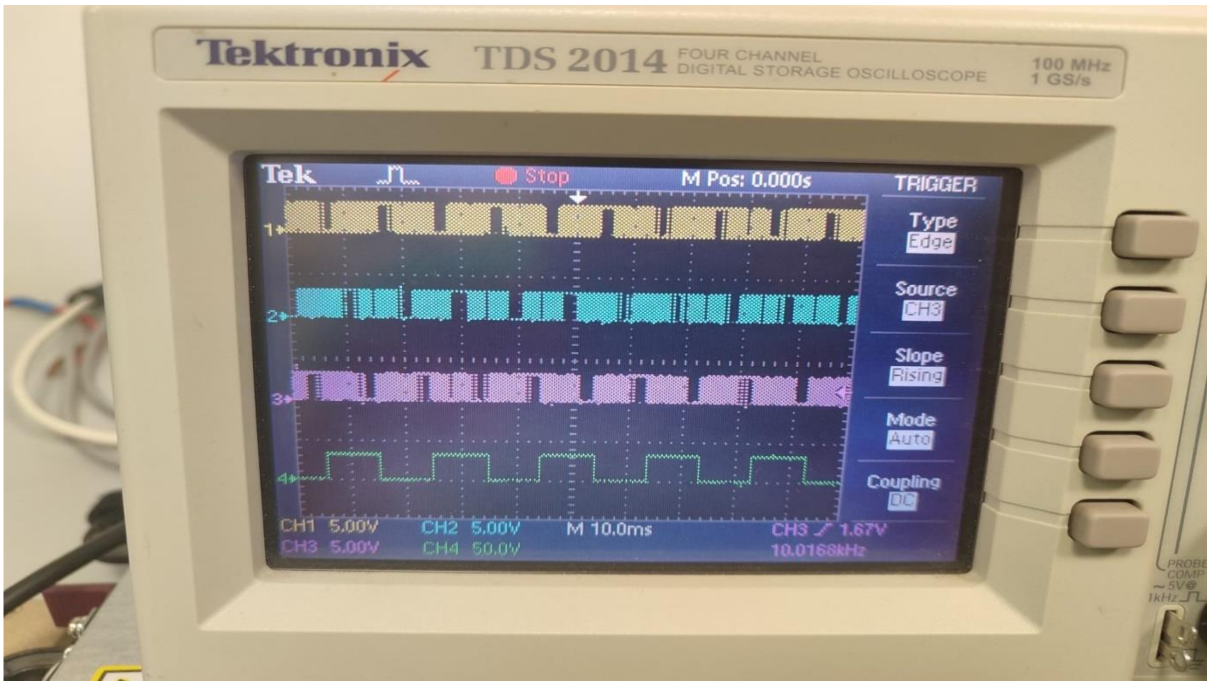


Figure 16: PWM signal used in driving the IGBTs gate

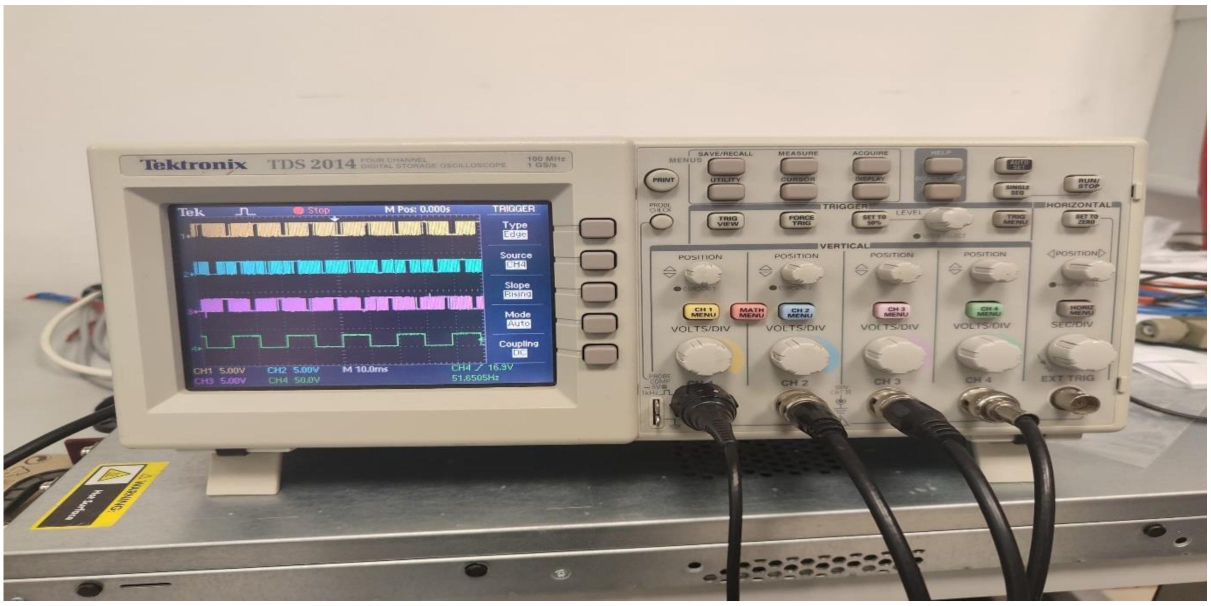


Figure 17: Signal of the trigger sine wave signal

Conclusion

The design and implementation of an SPWM-based Three Phase Inverter have proved to a significant improvement in the performance and effectiveness of renewable energy systems. In this project, the three-phase inverter model was designed and analysed with MATLAB Simulink software to attain lower Total Harmonic Distortion for better stability and improved efficiency of the inverter. The SPWM technique, with careful design and testing, has been proven to be effective in reducing harmonic content in the inverter output, thereby making the generated AC power smooth and more reliable for grid integration. Further addition of LC filter circuits weakened higher-order harmonics, ensuring a more precise output waveform. Results from both the simulation and the practical lab implementation stress the viability of SPWM in generating low levels of THD, which can be observed as a reduction in THD in the output current from 68.76% to 3.88% and in the output voltage from 68.76% to 13.85%.

In brief, the project met its prime objective by proving that SPWM-based inverters will aid in the addition of renewable sources of energy to the existing grids. This shall also go a long way toward enhancing the efficiency and reliability of renewable energy systems towards achieving the broader goal of sustainable energy development. Further works could explore further the optimization of the SPWM technique and incorporation of advanced filtering methods to enhance the performance of power conversion systems in renewable energy applications.

References

[1] NORCE, ‘Energy management for a novel hybrid energy storage system for the integration of renewable energy sources into the power grid’. Accessed: Jun. 25, 2024. [Online]. Available: https://www.norceresearch.no/prosjekter/energy-management-for-a-novel-hybrid-energy-storage-system-for-the-integration-of-renewable-energy-sources-into-the-power-grid

[2] ‘Solar Inverter A solar inverter is an electron’. Accessed: Jun. 27, 2024. [Online]. Available: https://techsmake.com/solar-inverter-a-solar-inverter-is-an-electron/

[3] T. Bhattacharjee, M. Jamil, and A. Jana, ‘Design of SPWM based three phase inverter model’, in *2018 Technologies for Smart-City Energy Security and Power (ICSESP)*, Bhubaneswar: IEEE, Mar. 2018, pp. 1–6. doi: 10.1109/ICSESP.2018.8376696.

[4] M. A. A. Younis, N. A. Rahim, and S. Mekhilef, ‘Simulation of grid connected THIPWM-three-phase inverter using SIMULINK’, in *2011 IEEE Symposium on Industrial Electronics and Applications*, Langkawi, Malaysia: IEEE, Sep. 2011, pp. 133–137. doi: 10.1109/ISIEA.2011.6108683.

[5] B. C. Babu, ‘Department of Electrical Engineering National Institute Of Technology Rourkela, Rourkela - 769008 May 2009’.

[6] B. Singh, K. Al-Haddad, and A. Chandra, ‘A review of active filters for power quality improvement’, *IEEE Trans. Ind. Electron.*, vol. 46, no. 5, pp. 960–971, Oct. 1999, doi: 10.1109/41.793345.

[7] ‘IEEE Recommended Practice and Requirements for Harmonic Control in Electric Power Systems’. IEEE. doi: 10.1109/IEEESTD.2014.6826459.

[8] M. Esa and Mohd Abdul Muqeem Nawaz, ‘THD analysis of SPWM &amp; THPWM Controlled Three phase Voltage Source Inverter’, p. 1559059 Bytes, 2019, doi: 10.6084/M9.FIGSHARE.7641887.

[9] V. K. Pal and S. Singh, ‘Design of Sinusoidal Pulse Width Modulation 3 Phase Bridge Inverter’, vol. 07, no. 07, 2020.

[10] M. S. Alkhazragi and N. K. AL-Shamaa, ‘Cascaded H-Bridge Multilevel Inverter Using SPWM and MSPWM Strategies’, *IJERA*, vol. 07, no. 06, pp. 14–20, Jun. 2017, doi: 10.9790/9622-0706021420.

[11] S. Umashankar, V. K. Arun Shankar, G. Jain, and M. L. Kolhe, ‘Comparative evaluation of pulse width modulation techniques on effective DC link voltage utilization of grid connected inverter’, in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Chennai, India: IEEE, Mar. 2016, pp. 2376–2383. doi: 10.1109/ICEEOT.2016.7755120.

[12] D. B. S. Kumar and G. Avinash, ‘MITIGATION OF LOWER ORDER HARMONICS IN A GRID CONNECTED THREE PHASE PV INVERTER’, vol. 02, no. 05, 2015.

[13] M. H. Rashid, *Power electronics: devices, circuits, and applications*, Fourth edition. Upper Saddle River, NJ: Pearson, 2014.

[14] P. Udakhe, D. Atkar, S. Chiriki, and V. B. Borghate, ‘Comparison of different types of SPWM techniques for three phase seven level cascaded H-Bridge inverter’, in *2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES)*, Delhi, India: IEEE, Jul. 2016, pp. 1–5. doi: 10.1109/ICPEICES.2016.7853286.

[15] ‘[Front matter]’, in *2014 IEEE 2nd International Conference on Electrical Energy Systems (ICEES)*, Chennai, India: IEEE, Jan. 2014, pp. i–xxiv. doi: 10.1109/ICEES.2014.6924131.

[16] T. Rahman, S. M. A. Motakabber, and M. I. Ibrahimy, ‘Design of a Switching Mode Three Phase Inverter’, in *2016 International Conference on Computer and Communication Engineering (ICCCE)*, Kuala Lumpur, Malaysia: IEEE, Jul. 2016, pp. 155–160. doi: 10.1109/ICCCE.2016.43.

[17] K. H. Ahmed, S. J. Finney, and B. W. Williams, ‘Passive Filter Design for Three-Phase Inverter Interfacing in Distributed Generation’, in *2007 Compatibility in Power Electronics*, Gdansk, Poland: IEEE, May 2007, pp. 1–9. doi: 10.1109/CPE.2007.4296511.

Appendix

Source code for generating SPWM for driving the VSI gate:

```
#include "F28x_Project.h"
#include "math.h"

// Define constants
#define EPWM_TIMER_TBPRD 1210 // 10kHz PWM Period register
#define PI 3.14159265358979323846 // Define PI constant
#define SAMPLES 205 // Number of samples for sine wave

// Function prototypes
void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);

__interrupt void epwm1_isr(void);
__interrupt void epwm2_isr(void);
__interrupt void epwm3_isr(void);

unsigned int i = 0, j = 0, k = 0; // Sample indices for each ePWM
float sine_wave[SAMPLES]; // Array to hold sine wave samples

// Main function
void main(void)
{
    int m; // Declare variable outside of the loop
    InitSysCtrl(); // Initialize system control

    // Generate sine lookup table for one period of a 50Hz sine wave
    for (m = 0; m < SAMPLES; m++) {
        float angle = 2 * PI * m / SAMPLES; // Calculate angle for each sample
        sine_wave[m] = sin(angle); // Store sine value in array
    }

    EALLOW;
    // Disable pull-up on GPIOs and configure as ePWM outputs
    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1; // Disable pull-up for GPIO0 (ePWM1A)
    GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1; // Disable pull-up for GPIO2 (ePWM2A)
    GpioCtrlRegs.GPAPUD.bit.GPIO4 = 1; // Disable pull-up for GPIO4 (ePWM3A)

    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1; // Configure GPIO0 as ePWM1A
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1; // Configure GPIO2 as ePWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1; // Configure GPIO4 as ePWM3A

    // Configure additional GPIOs for ePWM B channels
    GpioCtrlRegs.GPAPUD.bit.GPIO1 = 1; // Disable pull-up for GPIO1 (ePWM1B)
    GpioCtrlRegs.GPAPUD.bit.GPIO3 = 1; // Disable pull-up for GPIO3 (ePWM2B)
    GpioCtrlRegs.GPAPUD.bit.GPIO5 = 1; // Disable pull-up for GPIO5 (ePWM3B)

    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1; // Configure GPIO1 as ePWM1B
    GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1; // Configure GPIO3 as ePWM2B
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1; // Configure GPIO5 as ePWM3B

    // Configure additional GPIOs for phase-shifted sine wave output
    GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1; // Disable pull-up for GPIO6
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 0; // Configure GPIO6 as GPIO
    GpioCtrlRegs.GPADIR.bit.GPIO6 = 1; // Set GPIO6 as output

    GpioCtrlRegs.GPAPUD.bit.GPIO7 = 1; // Disable pull-up for GPIO7
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0; // Configure GPIO7 as GPIO
    GpioCtrlRegs.GPADIR.bit.GPIO7 = 1; // Set GPIO7 as output

    GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1; // Disable pull-up for GPIO8
    GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 0; // Configure GPIO8 as GPIO
    GpioCtrlRegs.GPADIR.bit.GPIO8 = 1; // Set GPIO8 as output

    EDIS;

    DINT; // Disable CPU interrupts
    InitPieCtrl(); // Initialize PIE control registers
    IER = 0x0000; // Clear CPU interrupt register
    IFR = 0x0000; // Clear CPU interrupt flag register
    InitPieVectTable(); // Initialize PIE vector table

    EALLOW;
    // Map ISR functions to PIE vector table
    PieVectTable.EPWM1_INT = &epwm1_isr;
    PieVectTable.EPWM2_INT = &epwm2_isr;
    PieVectTable.EPWM3_INT = &epwm3_isr;
    EDIS;

    EALLOW;
    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 0; // Stop all TB clocks
    EDIS;

    // Initialize ePWM modules
    InitEPwm1Example();
    InitEPwm2Example();
    InitEPwm3Example();

    EALLOW;
    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1; // Start all TB clocks
    EDIS;

    // Enable CPU interrupts for ePWM modules
    IER |= M_INT3;
    PieCtrlRegs.PIEIER3.bit.INTx1 = 1; // Enable PIE interrupt for ePWM1
    PieCtrlRegs.PIEIER3.bit.INTx2 = 1; // Enable PIE interrupt for ePWM2
    PieCtrlRegs.PIEIER3.bit.INTx3 = 1; // Enable PIE interrupt for ePWM3

    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global realtime interrupt DBGM

    for (;;)
    {
        asm (" NOP"); // Infinite loop
    }
}

// ISR for ePWM1
__interrupt void epwm1_isr(void)
{
    float value = sine_wave[i]; // Get current sine wave sample
    EPwm1Regs.CMPA.bit.CMPA = (value >= 0 ? value : -value) * EPWM_TIMER_TBPRD; //
Set CMPA value based on sine sample
    if (value >= 0) {
        EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Set action qualifier to clear on
up-count
        EPwm1Regs.AQCTLA.bit.CAD = AQ_SET; // Set action qualifier to set on down-
count
        EPwm1Regs.AQCTLB.bit.CAU = AQ_SET; // Set action qualifier for 1B
        EPwm1Regs.AQCTLB.bit.CAD = AQ_CLEAR; // Set action qualifier for 1B
    } else {
        EPwm1Regs.AQCTLA.bit.CAU = AQ_SET; // Set action qualifier to set on up-
count
        EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR; // Set action qualifier to clear on
down-count
        EPwm1Regs.AQCTLB.bit.CAU = AQ_CLEAR; // Set action qualifier for 1B
        EPwm1Regs.AQCTLB.bit.CAD = AQ_SET; // Set action qualifier for 1B
    }

    // Output phase-shifted sine wave sample on GPIO6
    if (value >= 0) {
        GpioDataRegs.GPASET.bit.GPIO6 = 1;
    } else {
        GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
    }

    i = (i + 1) % SAMPLES; // Increment and wrap sample index
    EPwm1Regs.ETCLR.bit.INT = 1; // Clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Acknowledge PIE interrupt
}

// ISR for ePWM2
__interrupt void epwm2_isr(void)
{
    float value = sine_wave[(j + SAMPLES / 3) % SAMPLES]; // Get 120-degree phase-
shifted sine wave sample
    EPwm2Regs.CMPA.bit.CMPA = (value >= 0 ? value : -value) * EPWM_TIMER_TBPRD; //
Set CMPA value based on sine sample
    if (value >= 0) {
        EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Set action qualifier to clear on
up-count
        EPwm2Regs.AQCTLA.bit.CAD = AQ_SET; // Set action qualifier to set on down-
count
        EPwm2Regs.AQCTLB.bit.CAU = AQ_SET; // Set action qualifier for 2B
        EPwm2Regs.AQCTLB.bit.CAD = AQ_CLEAR; // Set action qualifier for 2B
    } else {
        EPwm2Regs.AQCTLA.bit.CAU = AQ_SET; // Set action qualifier to set on up-
count
        EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR; // Set action qualifier to clear on
down-count
        EPwm2Regs.AQCTLB.bit.CAU = AQ_CLEAR; // Set action qualifier for 2B
        EPwm2Regs.AQCTLB.bit.CAD = AQ_SET; // Set action qualifier for 2B
    }

    // Output phase-shifted sine wave sample on GPIO7
    if (value >= 0) {
        GpioDataRegs.GPASET.bit.GPIO7 = 1;
    } else {
        GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;
    }
    j = (j + 1) % SAMPLES; // Increment and wrap sample index
    EPwm2Regs.ETCLR.bit.INT = 1; // Clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Acknowledge PIE interrupt
}

// ISR for ePWM3
__interrupt void epwm3_isr(void)
{
    float value = sine_wave[(k + 2 * SAMPLES / 3) % SAMPLES]; // Get 240-degree
phase-shifted sine wave sample
    EPwm3Regs.CMPA.bit.CMPA = (value >= 0 ? value : -value) * EPWM_TIMER_TBPRD; //
Set CMPA value based on sine sample
    if (value >= 0) {
        EPwm3Regs.AQCTLA.bit.CAU = AQ_CLEAR; // Set action qualifier to clear on
up-count
        EPwm3Regs.AQCTLA.bit.CAD = AQ_SET; // Set action qualifier to set on down-
count
        EPwm3Regs.AQCTLB.bit.CAU = AQ_SET; // Set action qualifier for 3B
        EPwm3Regs.AQCTLB.bit.CAD = AQ_CLEAR; // Set action qualifier for 3B
    } else {
        EPwm3Regs.AQCTLA.bit.CAU = AQ_SET; // Set action qualifier to set on up-
count
        EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR; // Set action qualifier to clear on
down-count
        EPwm3Regs.AQCTLB.bit.CAU = AQ_CLEAR; // Set action qualifier for 3B
        EPwm3Regs.AQCTLB.bit.CAD = AQ_SET; // Set action qualifier for 3B
    }

    // Output phase-shifted sine wave sample on GPIO8
    if (value >= 0) {
        GpioDataRegs.GPASET.bit.GPIO8 = 1;
    } else {
        GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
    }

    k = (k + 1) % SAMPLES; // Increment and wrap sample index
    EPwm3Regs.ETCLR.bit.INT = 1; // Clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3; // Acknowledge PIE interrupt
}

// Initialize ePWM1
void InitEPwm1Example()
{
    ClkCfgRegs.PERCLKDIVSEL.bit.EPWMCLKDIV = 0;
    EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO; // Select interrupt event on counter
zero
    EPwm1Regs.ETSEL.bit.INTEN = 1; // Enable interrupt
    EPwm1Regs.ETPS.bit.INTPRD = ET_1ST; // Generate interrupt on first event
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Set up-down count mode
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Set high-speed time-base clock
prescaler
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV2; // Set time-base clock prescaler
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_IMMEDIATE; // Set shadow mode for CMPA
    EPwm1Regs.CMPA.bit.CMPA = 0; // Initialize CMPA register
    EPwm1Regs.TBPRD = EPWM_TIMER_TBPRD; // Set timer period
    EPwm1Regs.TBPHS.bit.TBPHS = 0x0000; // Initialize phase register
    EPwm1Regs.TBCTR = 0x0000; // Clear counter
    EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // Enable dead-band
    EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active high complementary
    EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL; // Dead-band applied to both rising and
falling edges
    EPwm1Regs.DBRED.bit.DBRED = 50; // Rising edge dead-band
    EPwm1Regs.DBFED.bit.DBFED = 50; // Falling edge dead-band
}

// Initialize ePWM2
void InitEPwm2Example()
{
    EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO; // Select interrupt event on counter
zero
    EPwm2Regs.ETSEL.bit.INTEN = 1; // Enable interrupt
    EPwm2Regs.ETPS.bit.INTPRD = ET_1ST; // Generate interrupt on first event
    EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Set up-down count mode
    EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
    EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Set high-speed time-base clock
prescaler
    EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV2; // Set time-base clock prescaler
    EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_IMMEDIATE; // Set shadow mode for CMPA
    EPwm2Regs.CMPA.bit.CMPA = 0; // Initialize CMPA register
    EPwm2Regs.TBPRD = EPWM_TIMER_TBPRD; // Set timer period
    EPwm2Regs.TBPHS.bit.TBPHS = 0x0000; // Initialize phase register
    EPwm2Regs.TBCTR = 0x0000; // Clear counter
    EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // Enable dead-band
    EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active high complementary
    EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL; // Dead-band applied to both rising and
falling edges
    EPwm2Regs.DBRED.bit.DBRED = 50; // Rising edge dead-band
    EPwm2Regs.DBFED.bit.DBFED = 50; // Falling edge dead-band
}

// Initialize ePWM3
void InitEPwm3Example()
{
    EPwm3Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO; // Select interrupt event on counter
zero
    EPwm3Regs.ETSEL.bit.INTEN = 1; // Enable interrupt
    EPwm3Regs.ETPS.bit.INTPRD = ET_1ST; // Generate interrupt on first event
    EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Set up-down count mode
    EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Disable phase loading
    EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; // Set high-speed time-base clock
prescaler
    EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV2; // Set time-base clock prescaler
    EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_IMMEDIATE; // Set shadow mode for CMPA
    EPwm3Regs.CMPA.bit.CMPA = 0; // Initialize CMPA register
    EPwm3Regs.TBPRD = EPWM_TIMER_TBPRD; // Set timer period
    EPwm3Regs.TBPHS.bit.TBPHS = 0x0000; // Initialize phase register
    EPwm3Regs.TBCTR = 0x0000; // Clear counter
    EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // Enable dead-band
    EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active high complementary
    EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL; // Dead-band applied to both rising and
falling edges
    EPwm3Regs.DBRED.bit.DBRED = 50; // Rising edge dead-band
    EPwm3Regs.DBFED.bit.DBFED = 50; // Falling edge dead-band
}
```