LLC RESONANT CONVERTER MODELLING

by

Vasil Panov

B.Eng., University of Victoria, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical & Computer Engineering)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 2014

Abstract

Many of today's power converters use pulse-width-modulation(PWM) techniques to regulate the circulating currents and voltages. A significant problem with most dc-dc converters is the increased power loss during switching. These devices typically operate in hard-switching mode which results in switching losses. Resonant converters have been used to minimize or even eliminate this problem.

Although LLC resonant converters have shown significant gains in terms of efficiency, their modeling is still a challenge. LLC converters are designed to function in a specific mode and region of operation. It has been difficult to design a stable and robust controller with consistent bandwidth and disturbance rejection for every application. The complexity of the control design is magnified when the LLC converters are controlled using embedded digital control techniques. Recent developments in micro-controllers, including processing speed, power, and memory management, make possible the use of innovative non-linear or adaptive control algorithms, in order to produce high performance LLC circuits. Accurate modeling of the hardware is the key to an effective solution.

This thesis presents several modeling techniques of the LLC resonant converter. Previous research is discussed and relevant techniques are used as reference for deriving the models presented here. A new approach will be used to describe the characteristics of the LLC within the operating region. This approach is derived using the method of Least Squares of errors. The method estimates the coefficients of the plant transfer function, which then help to calculate control coefficients in the instantaneous operating condition of the LLC resonant power converter.

Preface

The work presented in this thesis is based on the original ideas of Vasil Panov and Dr. Rahul Khandekar, an employee at Alpha Technologies. I have derived and shown several models of the LLC resonant converter which were evaluated for accuracy by my project supervisor, Dr. Rahul Khandekar, and my university supervisor, Dr. William Dunford. Chapters 4,5 and 6 show the original contributions to the subject of LLC resonant converter modelling and digitally controlled circuits.

Table of Contents

List of Figures

List of Symbols

L_r	Series Resonant Inductance
L_m	Magnetizing Inductance
L_n	Inductance Ratio
C_r	Resonant Capacitor
C_o	Output Capacitor
n	Transformer turns ratio $\frac{N_s}{N_p}$
N_s	Number of turns on primary side of the transformer
N_p	Number of turns on secondary side of the transformer
Q	Quality factor
R_{ac}	Equivalent output resistance
R_o	Output Resistance
r	Series resistance
T_{on}	Transistor switch on time
T_{off}	Transistor switch off time
d	Duty cycle
V_{sq}	Equivalent input voltage into the resonant tank
V_c	Voltage across resonating capacitor
V_o	Output Voltage
I_r	Resonant inductor current
I_m	Magnetizing Current
I_p	Peak Current
f_{sw}	Switching frequency
f_o	Resonant frequency
f_p	Pole frequency
ω_s	Angular frequency
Z_C, Z_L	Circuit impedance

List of Abbreviations

AC	Alternating Current
ADC	Analog-to-Digital converter
BLPF	Butterworth Low Pass Filter
DC	Direct Current
DLL	Dynamic Link Library
DSP	Digital Signal Processor
EDF	Extended Describing Function
EMI	Electromagnetic Interference
HRPWM	High Resolution Pulse Width Modulation
LCC	Inductor-capacitor-capacitor
LLC	Inductor-inductor-capacitor
LPF	Low Pass Filter
LTI	Linear and time invariant
LSM	Least Squares Method
MEP	Micro-edge positioning
MOSFET	Metal-oxide-semiconductor-field-effect-transistor
PRC	Parallel Resonant Circuit
PWM	Pulse-width-modulation
SRC	Series Resonant Circuit
ZOH	Zero order hold
ZVS	Zero Voltage Switching

List of Units

A	Amperes
dB	Decibels
Hz	Hertz
S	Seconds
V	Volt
P	Pico (10^{-12})
N	Nano (10^{-9})
µ	Micro (10^{-6})
M	Mili (10^{-3})
K	Kilo (10^{3})
M	Mega (10^{6})
G	Giga (10^{9})

Acknowledgements

First, I would like to thank my university supervisor, Dr. William Dunford, for his guidance throughout my studies at UBC. His support in my two years as a master's student at UBC has been essential to the completion of my degree. His advice towards both my academic life and career has been greatly appreciated.

Secondly, I would like to express my gratitude towards my project supervisor at Alpha Technologies, Dr. Rahul Khandekar. His help through the course of this project has been invaluable. I would like to thank him for his guidance and taking the time to meet with me and discuss the project on a weekly basis. Without his support much of the work presented here would not be possible.

Next, I would like to thank the Natural Sciences and Engineering Research Council of Canada(NSERC) and Alpha Technologies Ltd. for their generous financial support. I would specifically like to thank Mr. Victor Goncalves for granting me the opportunity to use a research project at Alpha Technologies as my graduate work. I would also like to thank everyone else at Alpha Technologies that was involved in this process. Through them I have gained valuable and indispensable experience. In addition, I would also like to thank Mr. Brian Bella of the Faculty of Graduate Studies at UBC for his cooperation and support throughout the NSERC IPS scholarship application process.

Lastly, but most importantly, I would like to thank my family and friends for their support over the years. My parents have always believed in my ability to accomplish every task I have pursued, and for that I am deeply grateful.

Dedicated to my parents

CHAPTER 1 Introduction

1.1 Resonant Converters

A power supply with high efficiency , high power density, and low number of components, is highly desired in power electronics. This explains the popularity of resonant converters, such as the LLC. In addition, these circuits also contain low switching losses at high switching frequencies. Switching losses can be minimized by operating these circuits under soft-switching conditions. However, high leakage inductance, occurring across the resonating inductor or the transformer, often causes electromagnetic interference (EMI). Interferences in the circuit can be minimized by proper component selection. Further, controlling resonant type power converters is complicated, since it requires *frequency modulation,* instead of the simpler *duty cycle modulation* (also known as *pulse-width-modulation,* or PWM*)*. The methods of operation are almost identical in many resonant converter topologies.

Some of the most common converters are the series-resonant converter (SRC) and the parallel-resonant converter (PRC). Varying the switching frequency leads to a change in the circuit impedance of the inverter, which regulates the output voltage. The SRC circuit usually functions as a voltage divider, where the input voltage is divided between the impedance and the load of the circuit. Because of this function, high impedance is reached under light load conditions. As a result, regulating the output voltage is significantly more difficult[**1**]. In comparison, the PRC type requires a larger circulating current, since the load is in parallel with the resonating tank. This drawback makes it harder to apply the topology to high power density designs and large loads[**1**].

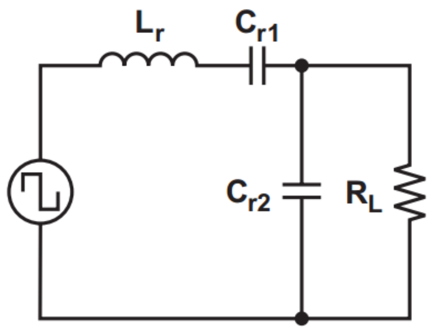


Figure 1- 1a : LCC Resonant Converter

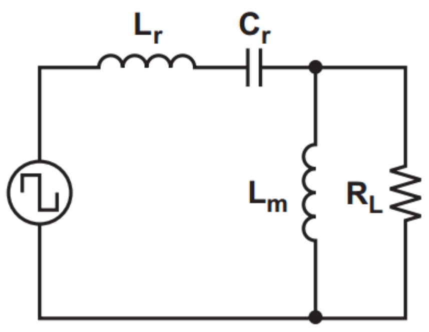


Figure 1- 1b : LLC Resonant Converter

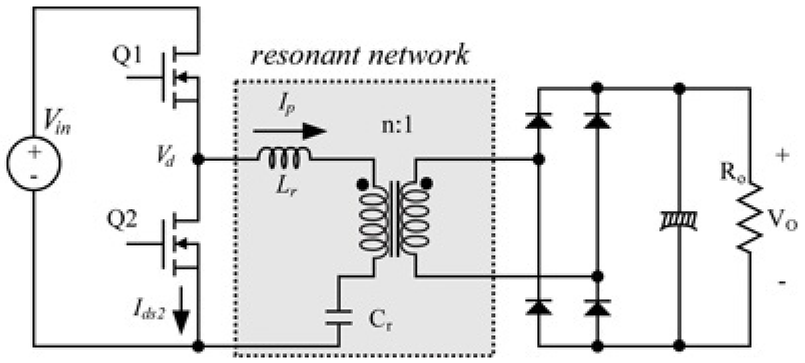


Figure 1- 1 c : LLC Resonant Converter

The limitations of both resonant converter types presented above can be eliminated with a series-parallel converter (SPRC) topology such as an LCC or an LLC. The LCC converter uses two capacitors and one inductor. The equivalent circuit configuration is shown in figure 1-1a. A drawback of this topology is the use of two large and expensive capacitors, used to handle large circulating currents[**1**].

An LLC, a type of SPRC, can be designed in order to avoid large components and minimize circuit size. This circuit requires two inductors and one capacitor. Although similar in characteristics to the LCC, the LLC circuit can be further minimized by combining the two inductors, L_r and L_m, into one physical component [**1**]. Other benefits of the LLC configuration include high efficiency during output voltage regulation, over a wide variety of loading conditions, but with little change in the switching frequency. During this operation, the circuit can also maintain zero-voltage switching (ZVS). This topology can be used with either half- or full-bridge inverters.

Zero-voltage switching is achieved when the MOSFET is turned on, but only after the drain-source voltage (*Vds)* reaches zero. This can be achieved by reversing the current through the bode diode of the MOSFET, provided the switching wave at the gate of the transistor passes a turn-on signal[**1**].

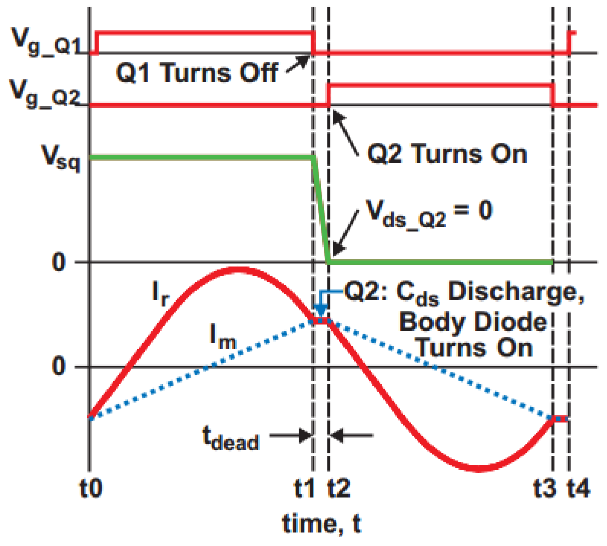


Figure 1- 2 : ZVS characteristics [1]

As seen in Figure 1- 2, switch Q1 is turned off at time t1, and switch Q2 is turned on at t2, but only after its drain-source voltage has reached 0V. This presents a dead time between t1 and t2. During this time, the resonating current is transferred from Q1 to Q2, thus discharging Q2's drain-to-source capacitance and forward biasing the Q2 body diode. At time t2, switch Q2 conducts and maintains zero *Vds*.

1.2 LLC Resonant Converter

Figure 1-1c shows a typical setup of a half-bridge LLC resonant converter, using a full-wave rectifier and an output capacitor on the secondary side of the transformer. The switches Q1 and Q2 represent MOSFETs designed to generate a square wave voltage input into the resonating tank.

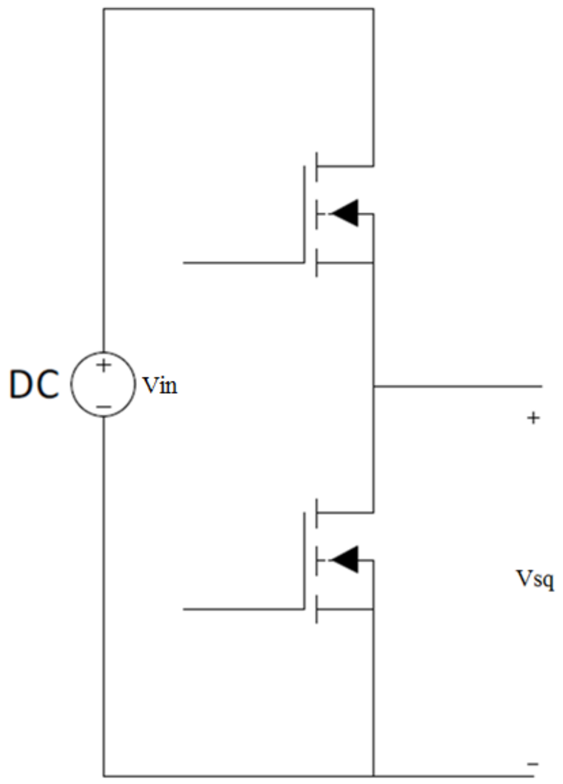


Figure 1- 3 : Half Bridge Inverter

The half bridge inverter is the first stage of the LLC resonant converter. It converts a DC input voltage into a square wave, whose frequency matches the switching frequency of the MOSFETs. Typically, the MOSFET's switching duty cycle (*d)* is set to alternate at 50% for a symmetric square waveform. In hardware, a small dead time is allowed between switching, in order to allow complete switch-off of the MOSFETs. This also achieves a *zero-voltage-switching* (ZVS) condition[**1**]. The output of the inverter is then fed into the resonant tank of the converter. The amplitude value of the square wave voltage, V_{sq}, is represented by:

$$V_{sq} \approx \frac{4}{\pi} \sin\left(\pi \frac{d}{2}\right) Vg \sin(\omega t) \tag{1.2a}$$

where V_g has the value of half of the amplitude of the DC voltage source, V_{in}.

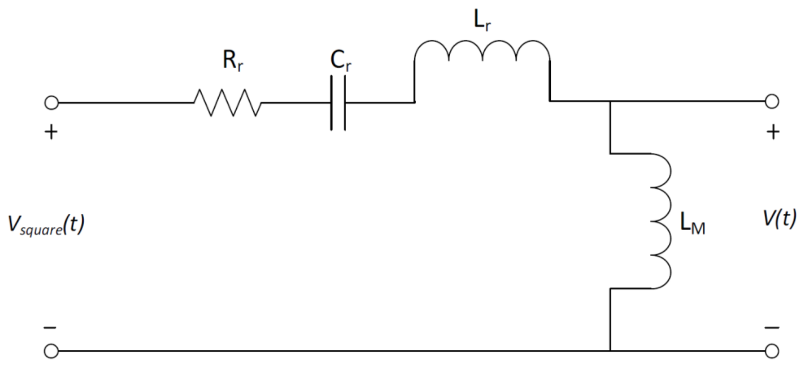


Figure 1- 4 : LLC Resonant Tank

The next component of the circuit is the resonating tank. It consists of a series inductance (*Lr*), series capacitance (*Cr*), series resistance (*Rr*), and the transformer's magnetizing inductance (*Lm*). The transformer turn's ratio (*n)*, shown in Figure 1-1c, sets the amplitude of the voltage and electrical current on the secondary side of the transformer.

$$\frac{Vp}{Vs} = \frac{Np}{Ns} = n \tag{1.2b}$$

$$\frac{Ip}{Is} = \frac{Ns}{Np} = \frac{1}{n} \tag{1.2c}$$

In the above formula, *N* represents the number of winding coil turns on the primary and secondary side of the transformer, *V* is the voltage on the primary and secondary side of the transformer, and *I* refers to the current on the primary and secondary side of the transformer. The current circulates inside the resonating network and is then delivered to the transformer. As stated previously, the input voltage (*Vsq)* is a square wave being transferred to the secondary side of the transformer. In this case, the transformer serves as isolator and regulator through the turn's ratio. Figure 1- 5 shows a simplified version of the LLC resonant converter.

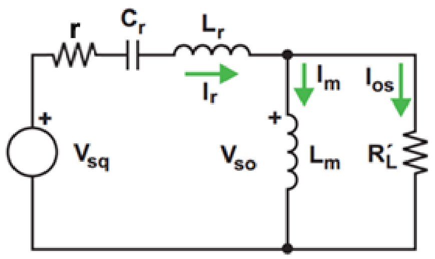


Figure 1- 5 : Simplified LLC circuit

The value of the load resistor *RL'* contains the load at the output side of the converter, as well as the losses from the transformer and the rectifier diodes. Since the rectifier on the secondary side servers as a voltage regulator, the equivalent load at the output does not equal the value of the load resistor. The value of the output resistor as seen on the primary side of the transformer, assuming transformer and diode losses are small and are neglected, is given by[**1**]:

$$RL' = 8n^2 \frac{Ro}{\pi^2} \tag{1.2d}$$

1.3 Operation

Minimum impedance is achieved when the circuit operates around the resonating frequency of the LLC network. The impedance of the circuit becomes higher, as the operating conditions deviate away from resonance[**2**]. The circulating current is greatly affected by a change of the impedance. The resonant frequency of the circuit is dependent on the series inductor and series capacitor as shown in 1.3a:

$$f_o = \frac{1}{2\pi\sqrt{L_r C_r}} \tag{1.3a}$$

In most resonating converters, f_o is the only frequency affecting the performance of the circuit. Due to the magnetizing inductance *(L_m)*, the operation of the LLC is also dependent on the output load conditions. The magnetizing inductor introduces a second frequency into the circuit, referred to as the pole frequency, given by the following equation:

$$f_p = \frac{1}{2\pi\sqrt{(L_r + L_m)C_r}} \tag{1.3b}$$

The frequency, where the highest gain is achieved, varies between the resonating frequency in (1.3a) and the pole frequency shown in (1.3b). Therefore, the converter's switching frequency (f_s) must be set in the range $f_p \leq f_s \leq f_o$. At zero load, i.e. an open circuit, $f_s = f_p$. At a shorted load $f_s = f_o$. This behaviour complicates the analysis of the LLC converter, but it also reduces the operating frequency range[**1**]. Typically, the LLC is designed to operate around *fo*, making this the dominant frequency and a critical factor in the operation and behaviour of the converter.

The LLC circuit exhibits a different behaviour when operating below, above, and at the resonant frequency. Figure 1- 6 shows the waveforms of voltage and current at these operating conditions.

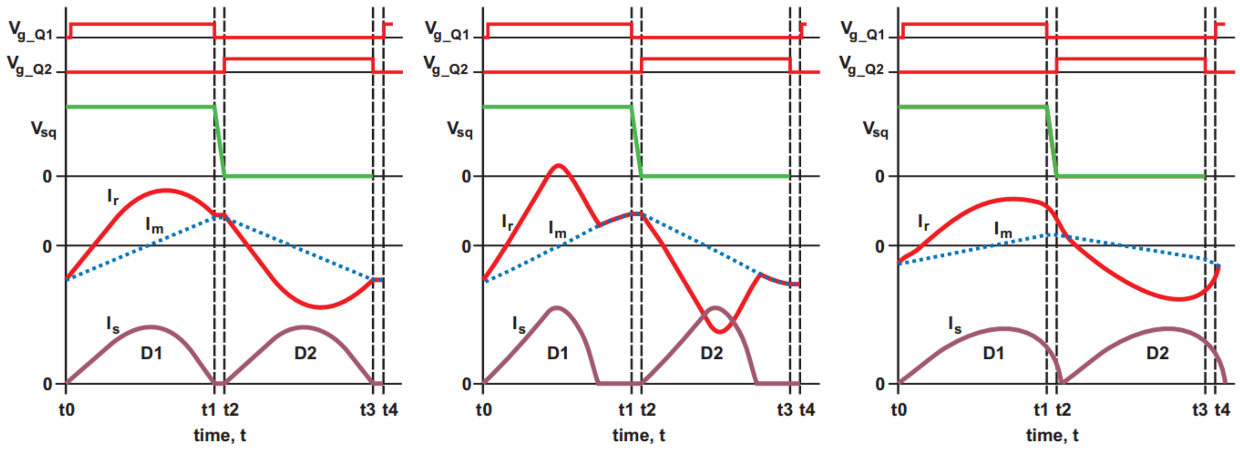


Figure 1- 6 : From left to right: At fo, below fo and above fo [1]

The plots show the input square wave voltage (*Vsq*), the magnetizing current (*Im*), the resonating current *(Ir),* and the secondary side current *(Is).* The current on the primary side (*Ir*) is the sum of the magnetizing current and secondary side current, as referred to the primary side. It should be noted, that since the magnetizing current is only present on the primary side, it does not contribute to the power transfer from the primary to the secondary side of the transformer[**1**]. When the converter's switching frequency operates at the resonant frequency, and the MOSFET (Q1) turns off, the resonant current equals the magnetizing current. Therefore, there is no transfer of power to the secondary side. At the same time the circuit achieves ZVS and soft commutation.

During operation below resonant frequency, the resonant current *(Ir)* goes below the magnetizing current before the end of the switching cycle, therefore stopping power transfer earlier than it would at the resonant frequency. In addition, the magnetizing current is still flowing on the primary side. Operating in this mode still achieves ZVS and soft switching on the secondary side. Because the current through the diodes on the secondary side is in discontinuous mode, more current is required to deliver the same power to the load[**1**]. In this case, the additional current causes more losses in the circuit. ZVS can be lost if the frequency goes below a certain low point.

In operation above the resonant frequency the circulating current is usually the smallest. This reduces the losses in the circuit and puts the device into continuous-current mode. This mode however, can cause drastic frequency increase under light loads.

In addition to the resonant frequency, another key characteristic of the LLC circuit is the *quality factor* (Q) given by the following equation, where the quality factor represents the ratio between the characteristic impedance and the resistive load:

$$Q = \frac{\sqrt{Lr/Cr}}{n^2 R_o} \tag{1.3c}$$

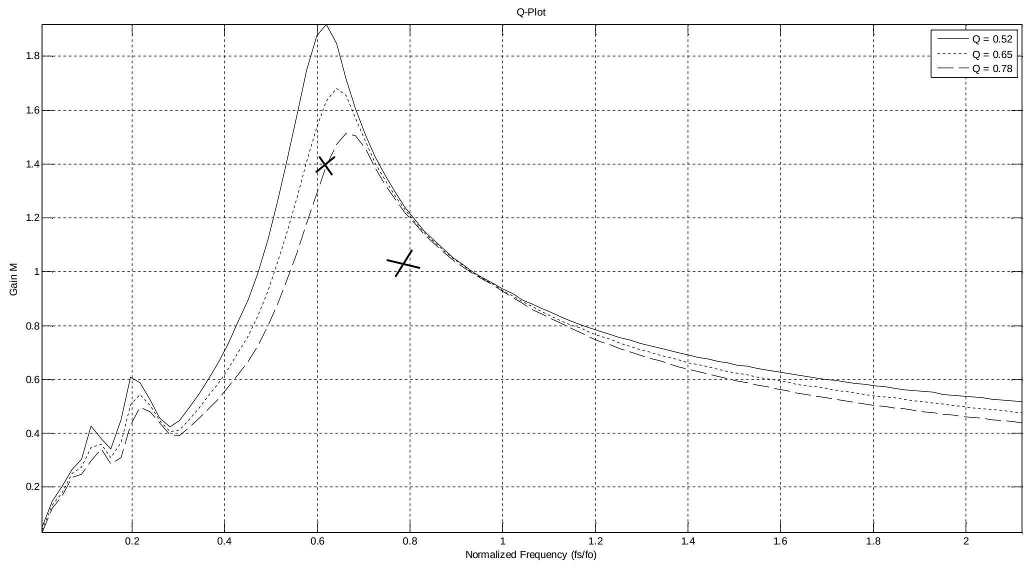


Figure 1- 7 : LLC Gain vs. frequency plot, compared at different values of the quality factor, Q

The plot above represents the typical characteristic of an LLC converter with changing quality factor. Typically the LLC operates near or along the negative slope region around the peak. The slope in this region is not as steep, therefore a variation in the switching frequency will cause a smaller, more controllable disturbance in the voltage gain. For example, in the case where the Quality factor is 0.78, the optimum normalized frequency (*fs*/*fo*) is in the range of 0.7 to 0.8.

The last factor that influences the operation of the LLC resonant converter is the inductance ratio *Ln*, shown below:

$$L_n = \frac{L_m}{L_r} \tag{1.3d}$$

The following plot shows the gain of the LLC converter with several *Ln* values:

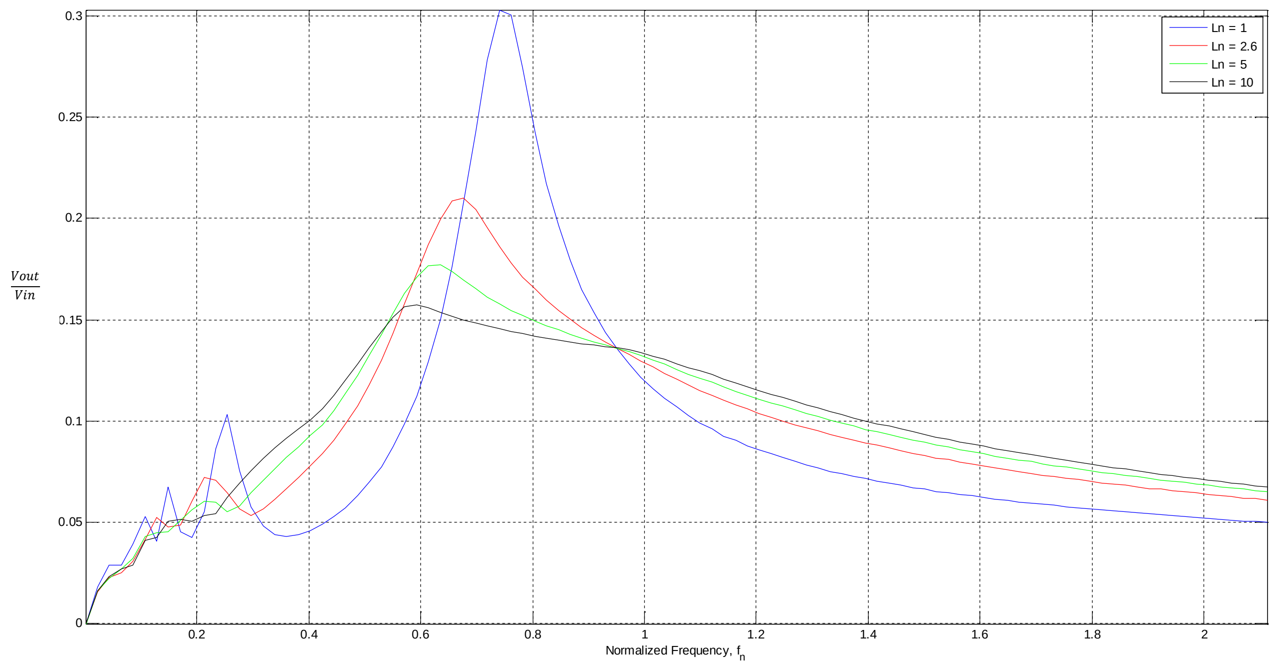


Figure 1- 8 : LLC Gain vs. frequency plot under varying Ln (Q = 0.65)

When the Q is fixed, a decrease in the inductance ratio (*Ln*) reduces the region of the curve horizontally, and as a result *fs* moves closer to *fo*. This leads to easier frequency control with smaller range and higher voltage gain. Furthermore, a decrease in *Ln* will cause an increase in *Lr*, which will consequently increase the quality factor, Q. The increase in Q also causes the curve to compress, as evident in Figure 1- 7.

A further description of the circuit can be given by its transfer function. The transfer function of the LLC circuit can be derived using the setup in Figure 1- 4 with the circuit impedances expressed as:

$$Z_c = \frac{1}{j\omega C_r} \tag{1.3e}$$

$$Z_L = j\omega L \tag{1.3f}$$

then the no load transfer function becomes:

$$H(s) = \frac{Vout}{Vin} = \frac{s^2 L_m C_r}{s^2(L_r + L_m) + sR_r C_r + 1} \tag{1.3g}$$

where

$s = j\omega$ and

$\omega = 2\pi f_s$

R_r is a resistor in series with *Cr* and *Lr*

With load resistor, R_L as show in Figure 1- 5 the transfer function becomes

$$H(s) = \frac{s^2 R_L C_r L_m}{s^3 C_r L_r L_m + s^2 C_r(L_r R_L + L_m R_L + L_m R_r) + s(L_m + C_r R_L R_r) + R_L} \tag{1.3h}$$

The equations above show that the impedance of the circuit is dependent on operating frequency. Therefore, varying the frequency will change the voltage at the output of the converter.

1.4 Objective of the Thesis

The objective of this thesis is to obtain a mathematical model that shows the LLC circuit's time and frequency domain response at any operating condition. The circuit's performance is evaluated during load and operational frequency changes. To show the time and frequency domain behavior of the LLC power converter, the circuit was first simulated in Powersim(PSIM)[35], where the voltage and current waveforms were observed. The results were used as a reference to assess the accuracy of the proposed LLC models. The proposed models are tested for stability and are proven accurate when, and only if, both the model and the reference yield the same time and frequency domain plots. In addition, the successful model shown in this thesis adapts to different operational regions, but perhaps most significantly, it generates an approximation of the circuit's transfer function. Because of this feature, the results obtained by the new model are superior to those generated in PSIM. Further, given that the LLC behaves differently according to operational region, the transfer function can be used in the design of an adaptive feedback controller.

In addition, the thesis evaluates and simulates the effects of a digital controller, used in the feedback control loop of the LLC circuit. The thesis shows existing PSIM digital control simulation models and discusses some original PSIM modules programmed by the author.

1.5 Structure of the Thesis

The thesis includes seven chapters. Chapter 1 is an introduction to several types of resonant converters. It specifically explains the operation of the LLC converter circuit. The chapter covers different operating regions of the LLC power converter and features of the circuit. In addition, the chapter also provides key circuit equations. Chapter 2 shows a simulation of the LLC circuit conducted in PSIM. The results are evaluated and used as a reference when qualifying the proposed models. The chapter shows the time and frequency response of the circuit under frequency and load variation. Further, Chapter 2 also provides an equivalent circuit of the LLC, used to obtain the mathematical models in Chapters 4-5. Chapter 3 introduces some of the mathematical techniques used in the derivation of the models in Chapters 4-5. It demonstrates important procedures used to create the models but also asses their stability and accuracy. The first modelling approach is discussed in Chapter 4. It uses the Extended Describing Function to derive the equations of the circuit. Existing models are evaluated before a new model is proposed. The chapter covers a step by step derivation of each circuit equation and explains important approximations before providing a complete model of the LLC circuit. Similar to the PSIM simulation in Chapter 2, the model evaluates the circuit's behavior under load and frequency variation. The results are assessed and their quality discussed. The results obtained from this model prove unreliable and a new modelling approach is discussed in Chapter 5. In Chapter 5, the circuit is modelled by the *Least Squares Approximation Method* (LSM). This model produced the transfer function of the LLC circuit. The frequency response of the circuit was evaluated via bode plot comparisons with the reference data obtained in PSIM. Chapter 7 discussed the digitizing effects of the analog-to-digital converter (ADC) and the

High Resolution Pulse-Width Modulation (HRPWM) module, used in the control loop of the physical LLC converter. Existing PSIM modelling techniques of digital control are evaluated. New ADC and HRPWM simulation models are proposed. Their merit and success is discussed. The final chapter summarizes the results and contributions of the thesis and gives suggestions to future work on the subject.

CHAPTER 2 LLC Simulation & Equivalent Circuit

2.1 Introduction

The first step in mathematical modelling is to observe the circuit's behavior when simulated in a circuit simulator such as Powersim(PSIM). The results obtained in PSIM can then be compared to those of the proposed mathematical model to ensure the accuracy of the new model. In the simulation of the LLC, the circuit is slightly adjusted in comparison to the design in Figure 1-1c. Two additional components are includes: LC output filter and an Open Loop Controller.

Two LC filters are added on the secondary side of the transformer to filter any high frequency components generated by the LLC converter. In addition, the filters prevent electromagnetic interference at the output of the circuit. The cut-off frequency of each filter is designed to be 58kHz. As will be evident later on, this filter has an effect on the frequency response of the circuit.

In hardware, the switching of the MOSFETs is controlled by a digital signal processor (DSP). To control the switching frequency generated by the DSP the user sets the period of the time-base counter (TBPRD) register[7]. The clock frequency of the DSP is divided by this number to set the switching frequency. In this case, the DSP sampling frequency was 60MHz (i.e. 120kHz switching frequency can be achieved with f_{DSP} = 60MHz and a register value $TBPRD$ = 500). The sampling frequency of the DSP also puts a constraint on the range of the available switching frequencies. This setup is reproduced in the open loop controller build in PSIM.

A copy of the open loop controller is given in the figure below. Once the frequency is set by the technique described above, the rest of the computational blocks generate a symmetric square wave signal with the specified frequency; the signal is then fed into the gate pin of the MOSFETs.

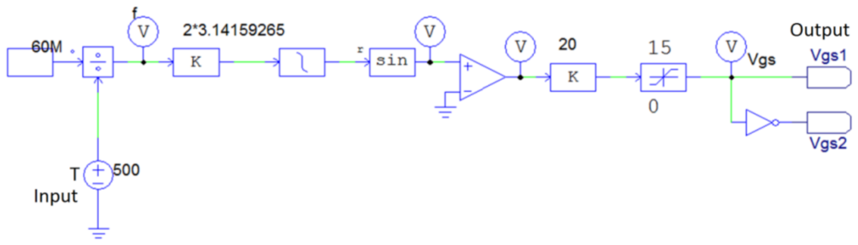


Figure 2- 1 : Open Loop controller

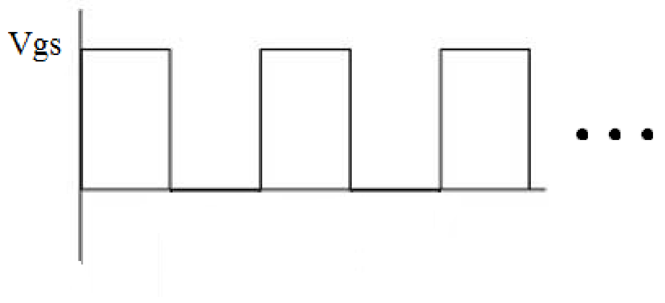


Figure 2- 2 : MOSFET Gate Waveform

The complete circuit diagram is shown in Figure 2- 3. The circuit includes a DC voltage source, half-wave inverter, LLC resonating tank, a transformer, full wave rectifier diodes, an output capacitor, additional output filters and a load resistor.

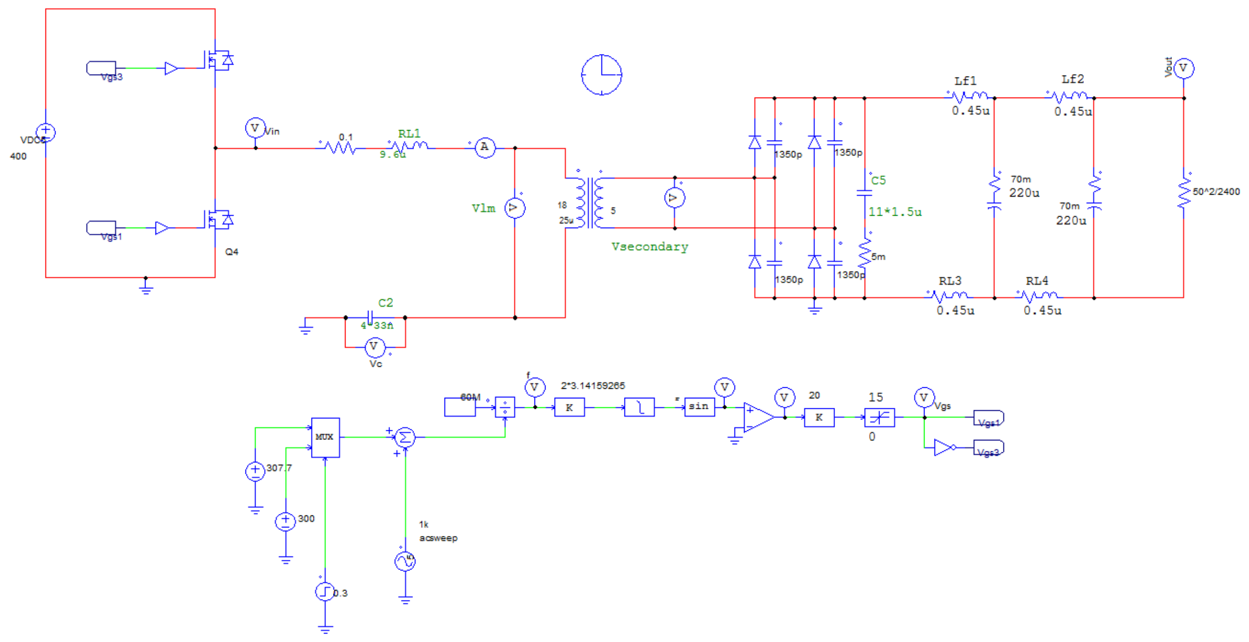


Figure 2- 3 : PSIM circuit layout

Before forming a model of the LLC resonant converter it is necessary to observe the behavior and dynamics of the typical circuit. The circuit in Figure 2- 3 resembles the design of a device in the line of products at Alpha Technologies Ltd. The typical circuit parameters are given in the table below.

Simulation parameters

V_{in}	400 V
n	3.6
L_R	9.5µH
C_R	132nF
L_M	25µH
R_L	1.04Ω
f_{sw}	142kHz

The results from the simulation of the LLC circuit in PSIM are shown in the figures to follow. The operating frequency is set to 142kHz which is exactly equal to the resonant frequency of the circuit, f_o. Figure 2-4 shows the output voltage and current behavior under constant switching frequency.

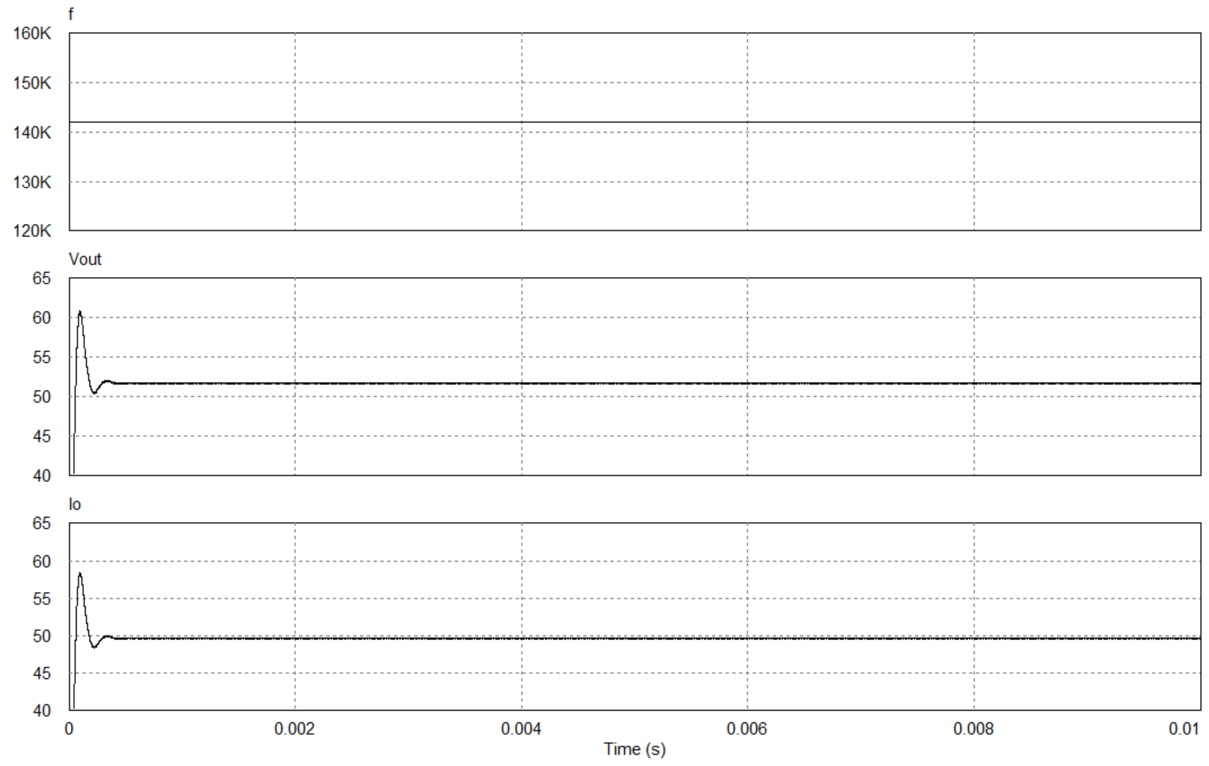


Figure 2- 4 : LLC's output voltage and output current levels at constant frequency (142kHz)

In the next few figures, the frequency in the circuit was increased during the simulation, and the response was observed. The initial frequency was 142kHz and at 30ms it was changed to 147kHz.

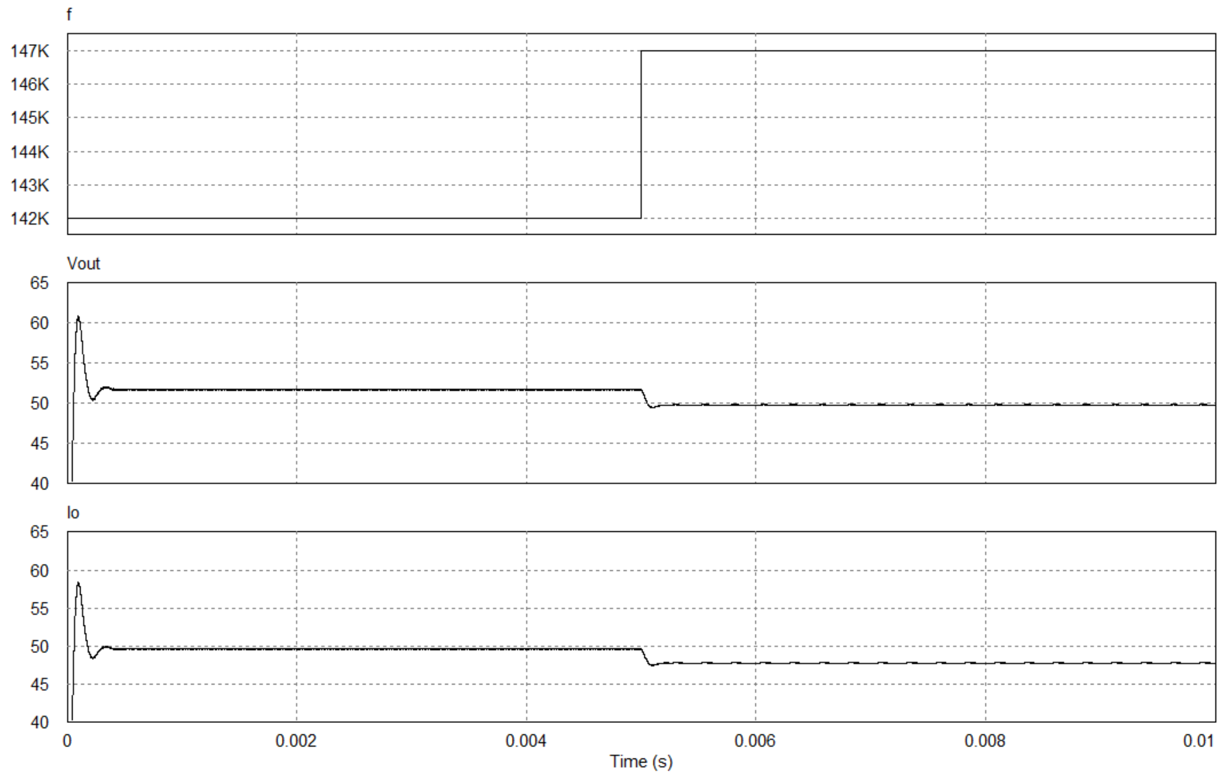


Figure 2- 5 : LLC's output voltage and output current during frequency change

To analyze the behavior of the power converter, changes in both frequency variation and load variation had to be tested and observed. Hence, the following figures show the circuit response when subjected to load variation with the switching frequency kept constant at 142kHz. For this purpose the load resistor value was doubled and the performance was observed.

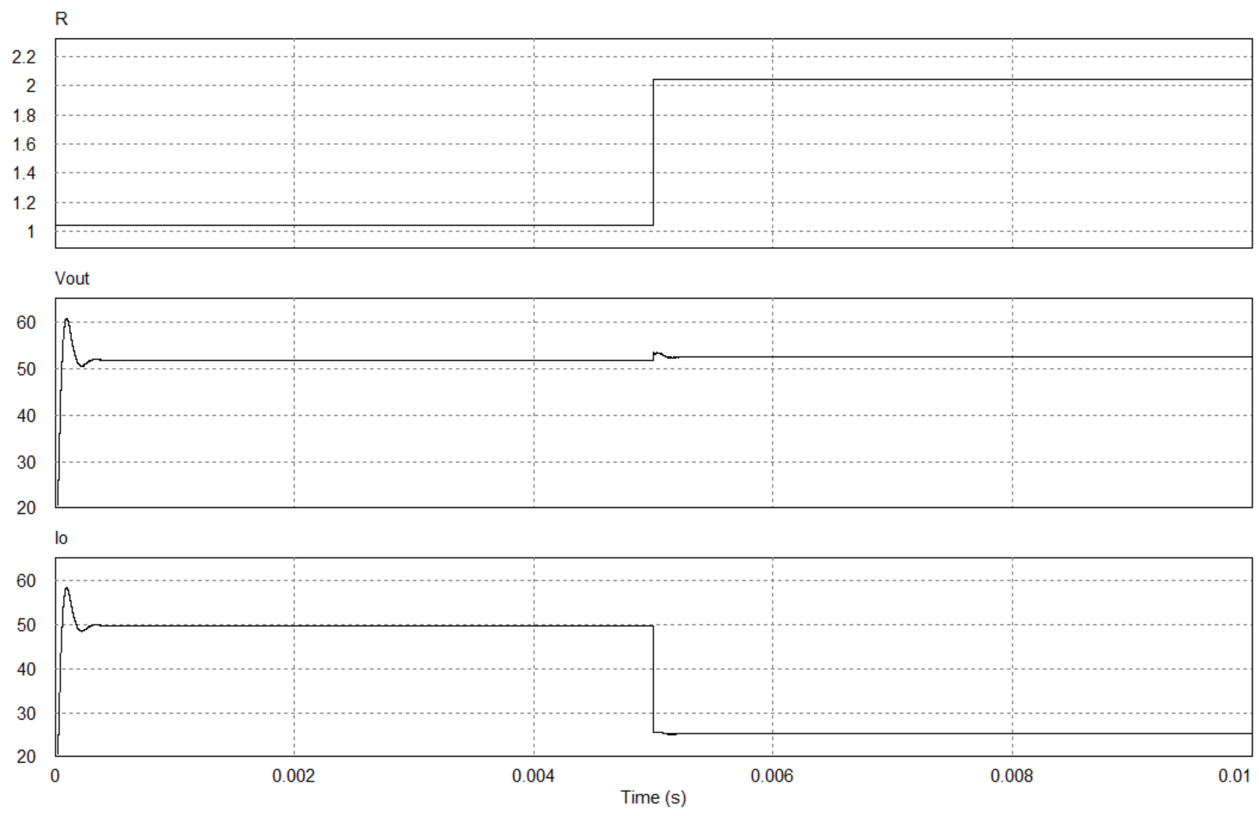


Figure 2- 6 : LLC's output voltage and output current during load change

Even though the typical LLC type resonant converter relies on frequency tuning to adjust the output parameters of the circuit, the final test in this section includes a scenario where the duty cycle of the inverter was varied. This is shown in Figure 2- 7.

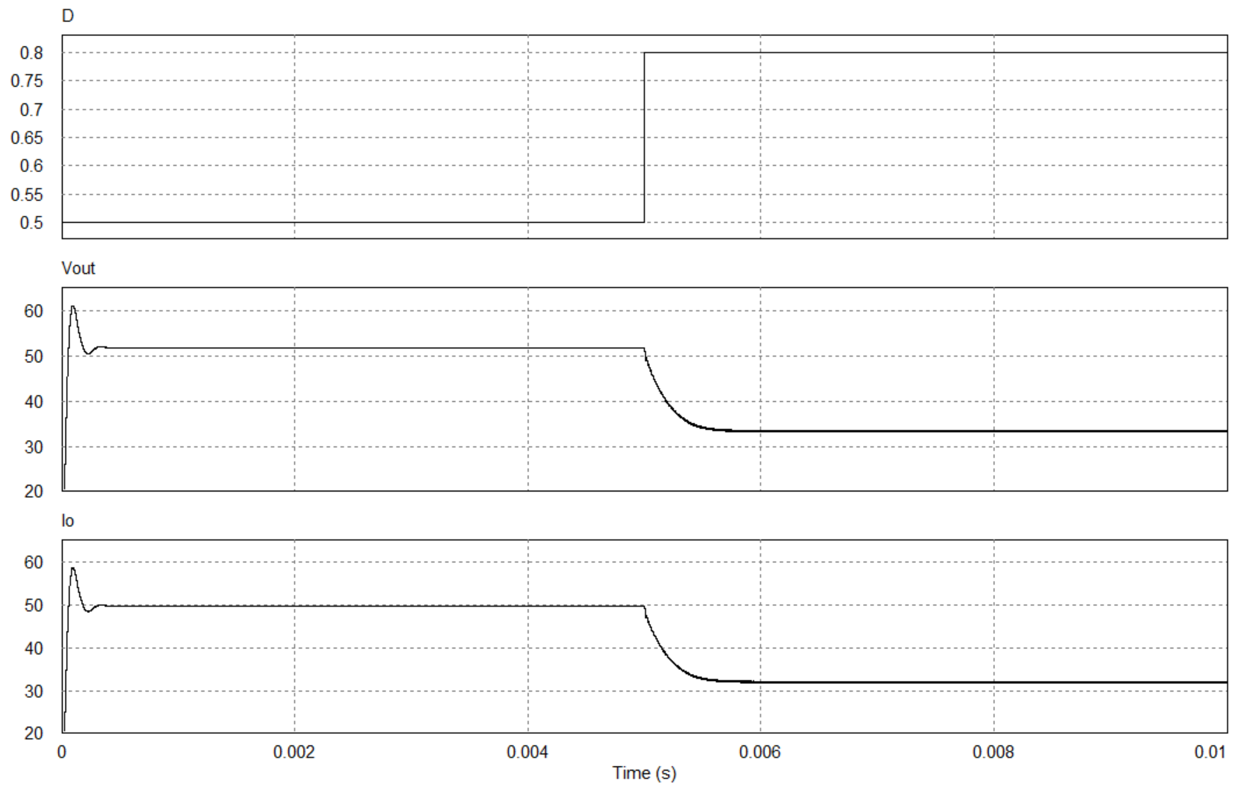


Figure 2- 7 : LLC's output voltage and output current during duty cycle change

Section 2.1 showed the behavior of the LLC circuit in time domain and how variations in switching frequency, load and duty cycle affect its performance. The next section will explore the frequency response of the circuit at several load conditions.

2.2 Bode Magnitude and Phase Plot

A bode plot of the circuit is an essential part of fully capturing the characteristics of the LLC resonant converter. The figures below compare the circuit's response at several loading conditions: 1.04Ω, 1.25Ω, 1.6Ω which result in power load of 1.6kW, 2kW and 2.4kW respectively.

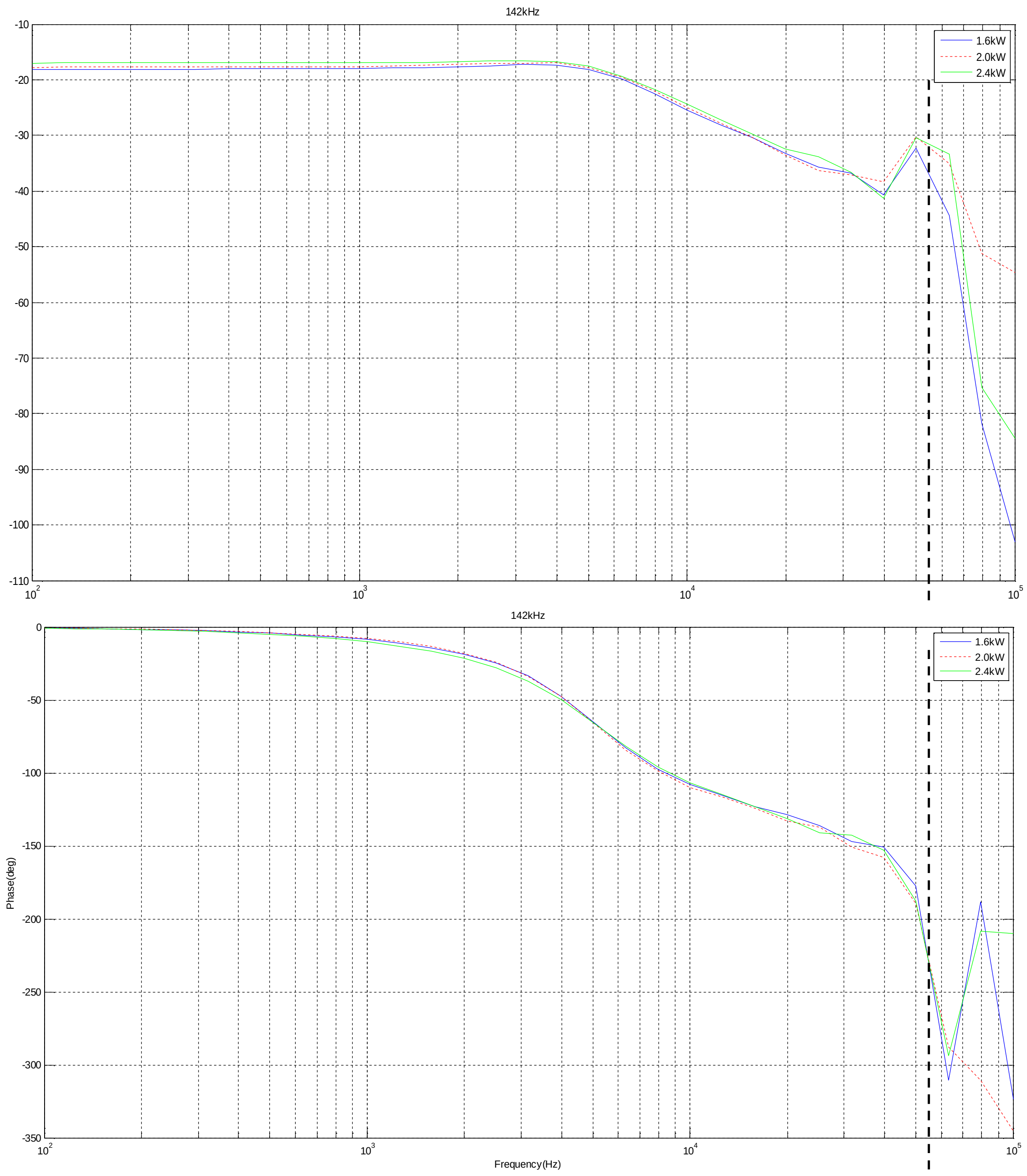


Figure 2- 8 : LLC PSIM magnitude and phase plot at several loading conditions

As mentioned earlier, the output filter plays a role in the frequency response of the circuit. The poles causing a second peak at 58kHz are produced by the filter's frequency. Without the filter, the second peak value would disappear from the magnitude and phase plots.

$$fo = \frac{1}{2\pi\sqrt{L_f C_f}} = 58.4\ kHz \tag{2.2a}$$

L_f = 450nH

C_f = 16.5uF

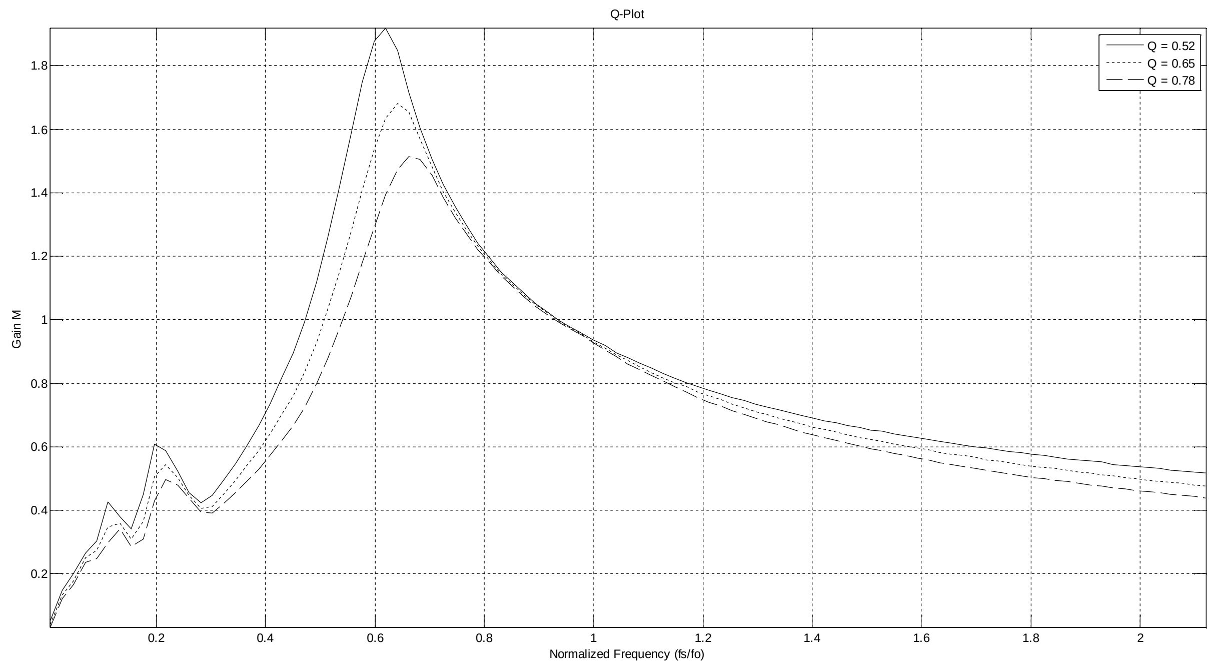


Figure 2- 9 : LLC Gain vs. frequency plot, compared at different values of the quality factor, Q

The Q-plot in Figure 2- 9 represents the voltage gain, M, of the LLC resonant converter over a range of switching frequencies. Three load conditions were included to analyze the behavior of the circuit: 1.6Ω, 1.25Ω and 1.04Ω which result in quality factors of 0.52, 0.65 and 0.78 respectively. The graph shows the peak gain decreasing and shifting closer to the resonant frequency as the load is decreases, which consequently increased Q. In the case where the load is the smallest, i.e. Q is at 0.78, best performance can achieved at switching frequencies between 70% and 100% of the resonating tank frequency, f_o. It should be noted that when the circuit is operating along the negative slope, past the peak, the performance is not affected as much by variations in the switching frequency as it would be when operating along the positive slope, toward the peak.

Before creating a mathematical model it is beneficial to simplify the LLC resonant circuit in Figure 2- 3. This will greatly reduce the circuit analysis of the mathematical model. This is done by constructing an equivalent circuit of the LLC resonant converter, containing fewer components and exhibiting similar performance. Such a design is shown in Figure 2- 10. Here, the circuit only contains the main parameters, resonating inductor, resonating capacitor, magnetizing inductor, load resistor and series resistance to account for loss in the inductors and capacitor. In this configuration, components such as the half bridge inverter and the full wave rectifier can be eliminated.

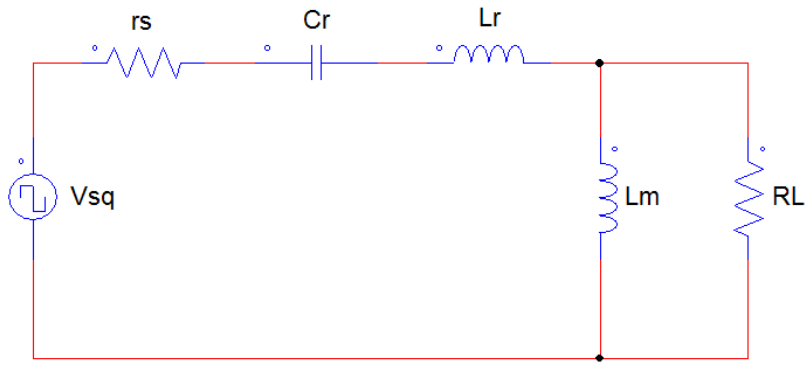


Figure 2- 10 : LLC converter equivalent circuit

A comparison between the LLC circuit and its equivalent shows high accuracy between the two models. The model was tested under switching frequency and duty cycle variation. The circuit was initially excited with 142KHz and the frequency was later increased to 180KHz. In duty cycle control, the circuit duty cycle was increased from 50% to 80%. The current across the inductors and the voltage across the resonating capacitor were chosen as the main variables to be monitored since these are responsible for the circuit behavior. It was discovered that the voltage across the equivalent load resistor represented the voltage on the primary side of the transformer, hence to find the value of the output voltage it had to be divided by the turns ratio and the losses across the full wave rectifier had to be subtracted. This gave a satisfactory approximation of LLC output voltage value. The current across the resonating inductor and the voltage across the resonating capacitor display similar response in both models with small error in amplitude.

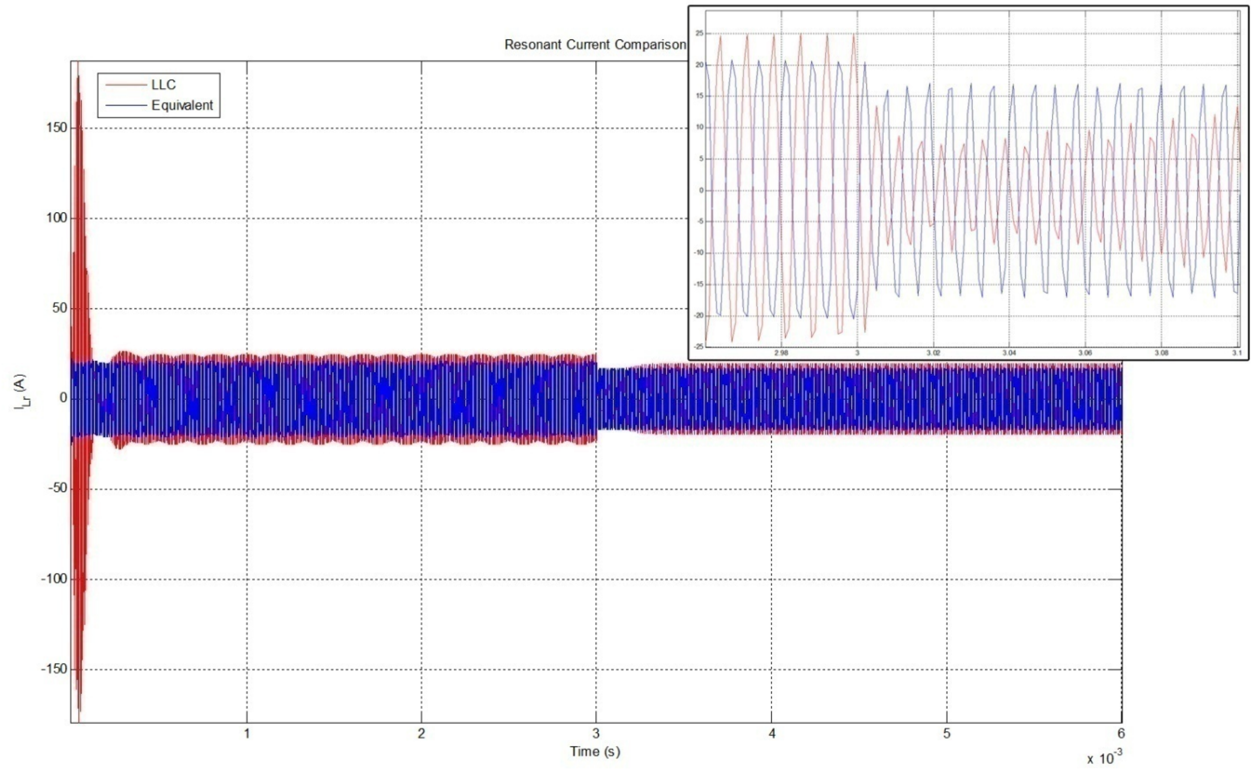


Figure 2- 11 : Resonant Current Comparison under frequency variation

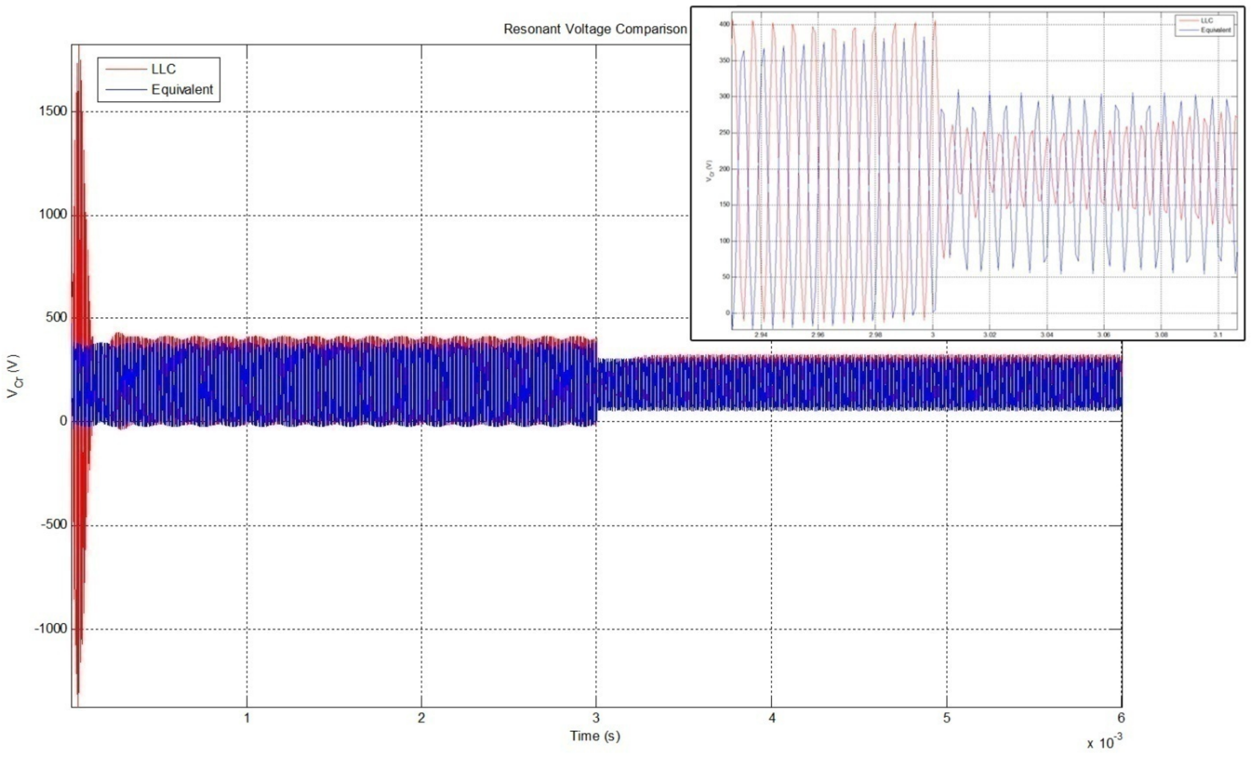


Figure 2- 12 : Resonant Voltage Comparison under frequency variation

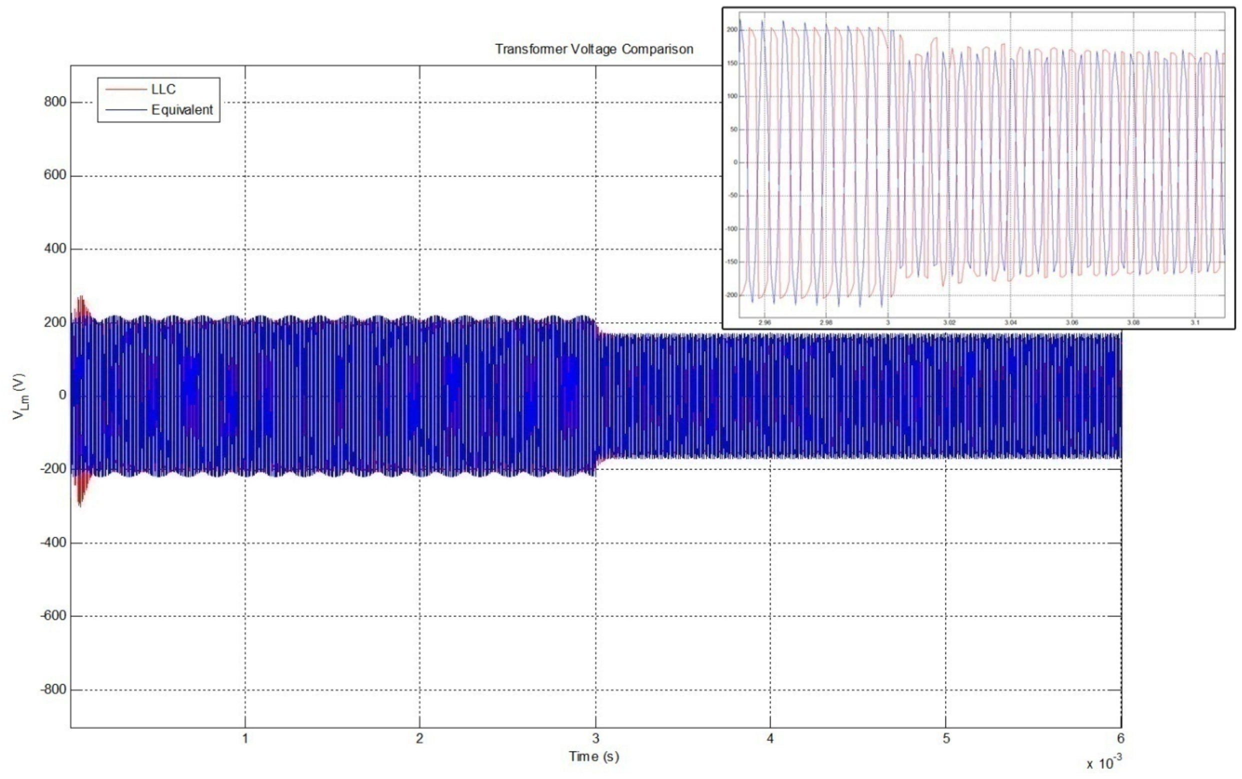


Figure 2- 13 : Transformer Voltage Comparison under frequency variation

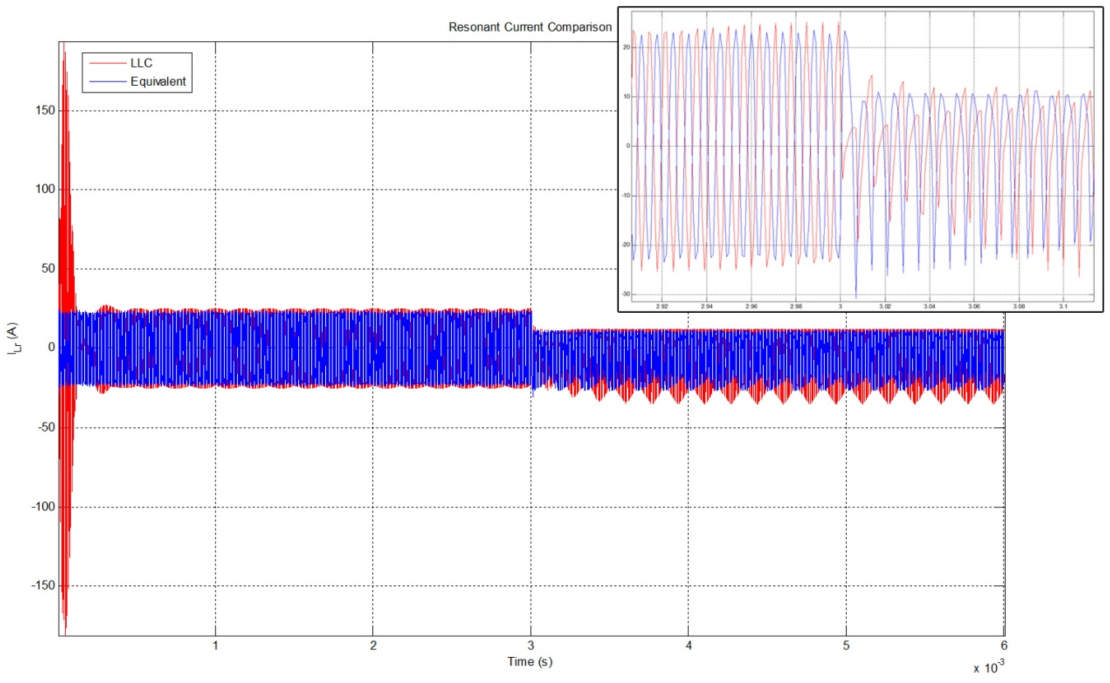


Figure 2- 14 : Resonant Current during duty cycle change

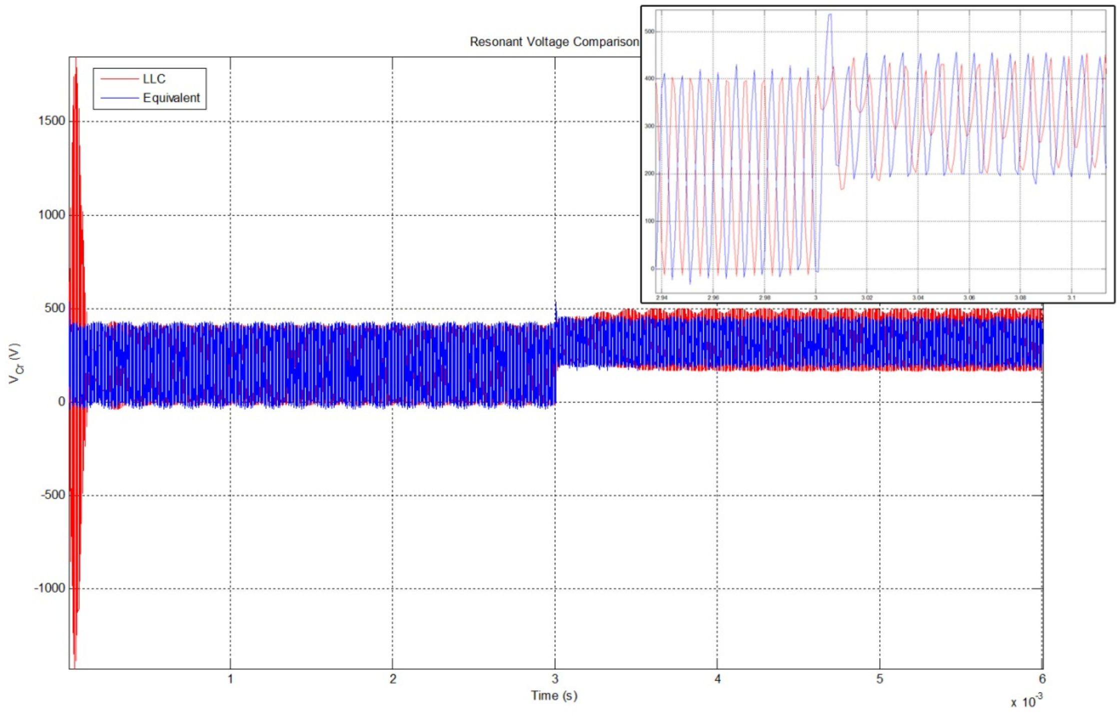


Figure 2- 15 : Resonant Voltage during duty cycle change

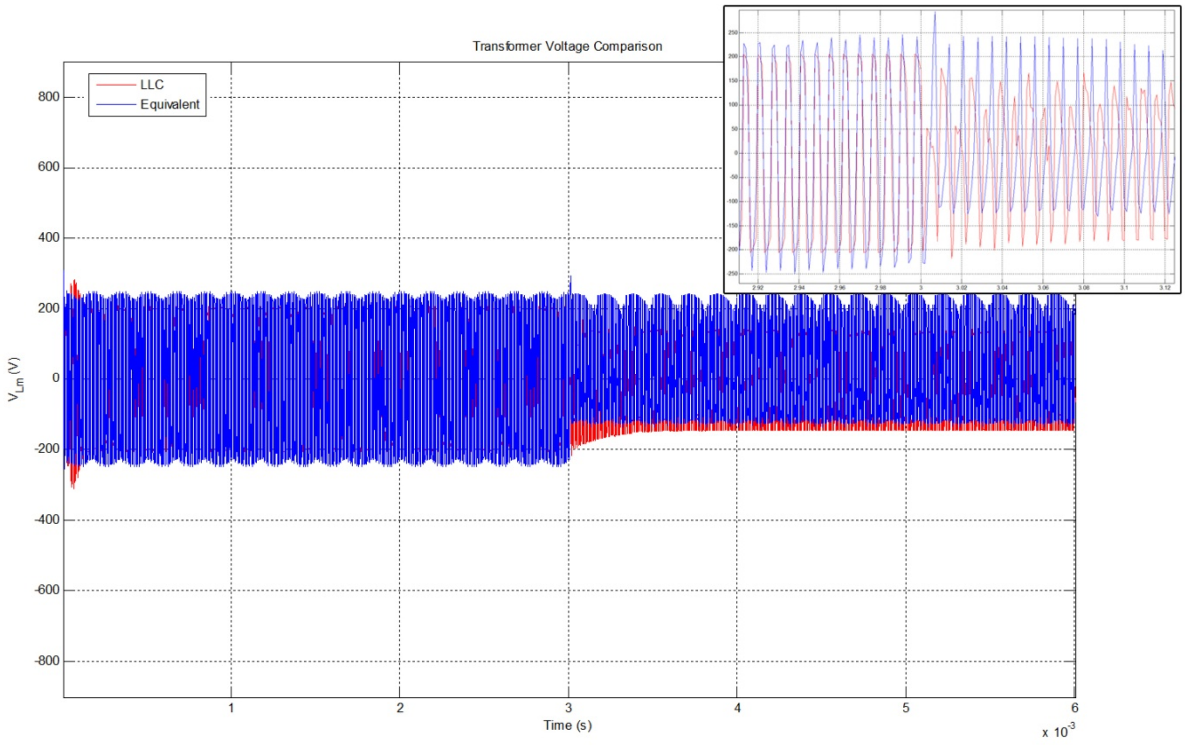


Figure 2- 16 : Transformer Voltage during duty cycle change

It is evident from the comparison in each scenario above, both the LLC circuit and its equivalent circuit exhibit similar behavior. Thus, a mathematical model using state space representation with voltages and currents as state variables can be derived using the equivalent circuit. The amplitude and time response of the resonating current, resonating voltage and the voltage across the transformer in the equivalent circuit closely match the values of the original LLC circuit. The slight difference in amplitude is due to voltage drops across components such as the transformer, rectifier diodes and the equivalent series resistance of the output capacitor. The behavior of the two circuits is closely matched during both frequency and duty cycle disturbances with the exception of a 90° phase shift between the two plots as seen in the zoomed-in images of figures 2-11 to 2-16.

CHAPTER 3 State Space Modeling

3.1 Introduction

The modelling techniques used in this thesis utilized the state space representation of the LLC's equivalent circuit. This section will explain the basics behind this modeling method.

The state space model consists of a series of equations with a set of inputs, outputs and state variables related by a first order differential equation. For a linear and time invariant system (LTI) the equations can be placed into matrix form and then solved. The state space model is an efficient way to demonstrate the behaviour of a system with multiple inputs and outputs and obtain the transfer function of the system. Figure 3- 1 shows the typical block diagram representation of a state space model.

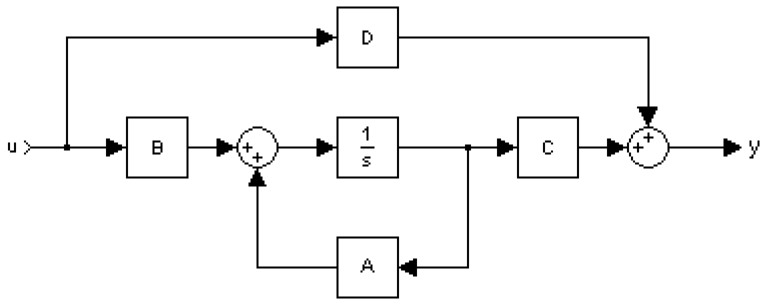


Figure 3- 1: Block diagram of state space

The continuous time state space model is in the form of

$$\frac{dx}{dt} = Ax(t) + Bu(t) \tag{3.1a}$$

$$y(t) = Cx(t) + Du(t) \tag{3.1b}$$

where the vector x(t) represents the state variables, u(t) contains the inputs to the system and y(t) has the output variables. Typically, in an electrical circuit, the number of state variable equals the number of energy storage elements such as capacitors and inductors.

Also:

$x(t) \in \ R^n$

$y(t) \in \ R^q$

$u(t) \in \ R^p$

A is the state matrix with dimensions n x n

B is the input matrix with dimensions n x p

C is the output matrix with dimensions q x n

D is the output feed-forward matrix with dimensions q x p

*An example of a state space model is given in the Appendix section.

3.2 State Space Evaluation

The A matrix can be used to assess the stability of the system. In the LTI case the eigenvalues (λ) of the state matrix, A, correspond to the poles of the system's transfer function.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{(s-z1)(s-z2)(s-z3)}{(s-p1)(s-p2)(s-p3)(s-p4)} \tag{3.2a}$$

$$\lambda = det\ (sI - A) \tag{3.2b}$$

z1,z2,z3 - zeros

p1,p2,p3,p4 - poles

The transfer function can be derived using

$$G(s) = C(sI - A)^{-1}B + D \tag{3.2c}$$

$$G(s) = \frac{Y(s)}{U(s)} \tag{3.2d}$$

The system can also be tested for controllability and observability.

Controllability allows the states of the system to be controlled by an external input and move the system from its initial conditions to the final value in finite time. For the system to be controllable, the rank of the controllability matrix must be equal to n, the number of rows in the state matrix A.

$$C = rank[B\ AB\ AB^2 \dots\ A^{n-1}B] = n$$

Observability allows to determine the current system states at any time, t, by using its outputs. This also allows to determine the behaviour of the system using just the outputs. For a system to be observable the rank of the observability matrix must be the same as n.

$$O = rank[C\ CA \dots CA^{n-1}]^T = n$$

The following chapters will derive models of the LLC resonant converters in state space representation.

CHAPTER 4 Frequency Control by Describing Function Method

4.1 Introduction

State space averaging is the most popular method of modeling PWM power converters. It shows a simple solution with satisfactory accuracy. However, this method cannot be used to fully describe the LLC resonant converter due to its natural frequency being close the switching frequency. Typically a small-signal method based on the extended describing function (EDF)[3] has been used to model resonant converters. The model shows the circuit behavior under small signal changes of the input voltage, switching frequency and duty cycle.

The model based on the EDF has been simplified by the following assumptions:

I. The perturbation signal's frequency is much lower than that of the switching frequencies and its amplitude is very small compared to the amplitude of the variable being disturbed.
II. The resonant component of the waveforms is assumed sinusoidal
III. The switches and components are ideal

This approach approximates the current and voltage parameters using sine and cosine components of the circuit waveforms. Along with Fourier series expansion, this approximation is used to turn the non-linear system into linear. The final stage of the process includes small-signal analysis.

4.2 Existing Models

Few attempts of modelling LLC converters have been performed in the past with little success. This section will cover some of the most recent research in the area. Most academic papers use the small signal approach based on the extended describing function (EDF) to model the circuit.

Chang[4] at I-Shou University in Kaohsiung, Taiwan has used the EDF to model an equivalent circuit of the LLC resonant converter. The research follows the EDF method to obtain the circuit equations, however, very small portion of the results has been shown in the paper; it only includes a magnitude and phase plots which can be obtained from a simulation of the LLC circuit or a simpler mathematical model. None of the results reference any state space modelling results for voltage and currents flowing inside the circuit. When this model was duplicated and tested to ensure its accuracy, the results were inconclusive, showing very different performance than the one shown in the results section of Chang's paper.

In addition, when the equivalent circuit proposed by the author was simulated, the values shown did not match the results from the detailed LLC circuit.

The second paper attempting to produce a model of the LLC converter comes from Tianjin University in Tianjin, China. Wang[8] uses the Generalized State Space Averaging method (GSSA) to obtain a circuit model with state variables $x = [i_s \ i_c \ V_s \ V_c \ V_f]^T$.

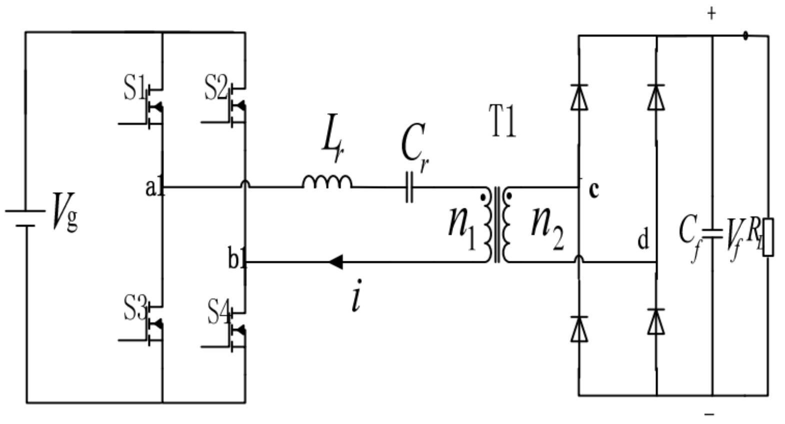


Figure 4- 1 : Wang's LLC Equivalent circuit[8]

The flaw of this approach is that the author does not account for the effects of the magnetizing inductor and the current flowing through it. The circuit current is simply represented by just the current through the resonating inductor, Lr, completely ignoring the effects of the magnetizing inductance of the transformer. This turns the circuit into an LC type, which makes it simpler to analyze but does not provide an accurate solution in time or in frequency domain. The current equations given by Wang are as follows:

$$i_r(t) = i_{Lrs}(t)\sin(\omega t) + i_{Lrc}(t)\cos(\omega t) \tag{4.2a}$$

$$i_p = \sqrt{i_{Lrs}^2 + i_{Lrc}^2} \tag{4.2b}$$

The more complete representation that includes the magnetizing current is given by

$$i_{Lr}(t) = i_{Lrs}(t)\sin(\omega t) + i_{Lrc}(t)\cos(\omega t)$$

$$i_{Lm}(t) = i_{Lms}(t)\sin(\omega t) + i_{Lmc}(t)\cos(\omega t)$$

$$i_p = \sqrt{(i_{Lrs} - i_{Lms})^2 + (i_{Lrc} - i_{Lmc})^2} \tag{4.2c}$$

where i_p represents the current on the secondary side of the transformer, a combination of the resonant and magnetizing currents. The proposed equations by Wang[8] would only be acceptable if the value of the magnetizing current was much smaller than that of the resonating current

if $i_{Lr}(t) \gg i_{Lm}(t)$,

$$i_p = \sqrt{(i_{Lrs} - i_{Lms})^2 + (i_{Lrc} - i_{Lmc})^2} \rightarrow i_p = \sqrt{i_{Lrs}^2 + i_{Lrc}^2}$$

During simulation of the circuit using the method proposed by Wang[8], the value and dynamics of the resonant current waveform did not match that of the PSIM simulation results. The only similarity between the model proposed by Wang and the PSIM simulation of the LLC circuit appeared in the waveform dynamics of the resonating capacitor voltage. However, this is not enough to completely describe the model.

It might be possible for the EDF model proposed by Wang[8] to present a successful solution when dealing with a LCC type resonant tank, since this model will include two voltages and one current as state variables[6]; this will simplify the current equation to (4.2b).

4.3 New Model

Since the existing models did not provide accurate representation of the LLC converter dynamics, there was a need for a different model based on the EDF approach that would yield the necessary results. Figure 4- 2 shows the equivalent circuit.

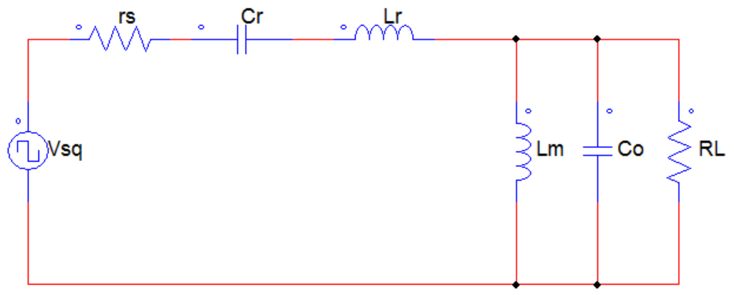


Figure 4- 2 : LLC Equivalent circuit

The input square wave is assumed to be symmetric with its magnitude depending on half the magnitude of the DC input voltage. The halved DC voltage value shall be labelled as V_g. The equivalent circuit provides the following nonlinear state space equations:

$$V_{sq} = i_{Lr}r + L_r\frac{di_{Lr}}{dt} + L_m\frac{di_{Lm}}{dt} + V_{Cr} \tag{4.3a}$$

$$L_m\frac{di_{Lm}}{dt} = (i_{Lr} - i_{Lm})R_L = sgn(i_{Lr} - i_{Lm})V_o \tag{4.3b}$$

$$\frac{dv_{Cr}}{dt} = \frac{i_L}{C_r} \tag{4.3c}$$

where iLr, iLm and VCr are the state variables and Vo is the output variable.

4.4 Harmonic Approximation

At steady state the waveforms of the LLC resonant converter are assumed sinusoidal therefore the fundamental harmonics of the state variables can be approximated by a combination of sine and cosine components as:

$$i_{Lr} = i_{Lrs}(t)\sin(\omega_s t) + i_{Lrc}(t)\cos(\omega_s t) \tag{4.4a}$$

$$i_{Lm} = i_{Lms}(t)\sin(\omega_s t) + i_{Lmc}(t)\cos(\omega_s t) \tag{4.4b}$$

$$v_{Cr} = v_{Crs}(t)\sin(\omega_s t) + v_{Crc}(t)\cos(\omega_s t) \tag{4.4c}$$

The derivatives of iLr, iLm and VCr are given by:

$$\frac{di_{Lr}}{dt} = \left(\frac{di_{Lrs}}{dt} - \omega_s i_{Lrc}\right) sin(\omega_s t) + \left(\frac{di_{Lrc}}{dt} + \omega_s i_{Lrs}\right) \cos(\omega_s t) \tag{4.4d}$$

$$\frac{di_{Lm}}{dt} = \left(\frac{di_{Lms}}{dt} - \omega_s i_{Lmc}\right) sin(\omega_s t) + \left(\frac{di_{Lmc}}{dt} + \omega_s i_{Lms}\right) cos(\omega_s t) \tag{4.4e}$$

$$\frac{dv_{Cr}}{dt} = \left(\frac{dv_{Crs}}{dt} - \omega_s v_{Crc}\right) \sin(\omega_s t) + \left(\frac{dv_{Crc}}{dt} + \omega_s v_{Crs}\right) \cos(\omega_s t) \tag{4.4f}$$

4.5 Extended Describing Function

The extended describing function approximates the behavior of a continuously operating power converter by expressing the circuit variables as a sum of harmonics of the switching frequency. Typically the fundamental frequency component is used when modelling resonant type converters[3] . The following equations are derived by EDF methods, using the fundamental frequency component of the voltage and current waveforms after Fourier expansion:

$$V_{sq} \approx \frac{4}{\pi} \sin(\pi \cdot d) * V_g sin(\omega_s t) \tag{4.5a}$$

$$sgn(i_{Lr} - i_{Lm}) V_o = \frac{4}{\pi} \frac{(i_{Lrs} - i_{Lms})}{i_p} V_o \sin(\omega_s t) + \frac{4}{\pi} \frac{(i_{Lrc} - i_{Lmc})}{i_p} V_o \cos(\omega_s t) \tag{4.5b}$$

$$|i_{Lr} - i_{Lm}| = \frac{2}{\pi} i_p \tag{4.5c}$$

$$i_p = \sqrt{(i_{Lrs} - i_{Lms})^2 + (i_{Lrc} - i_{Lmc})^2} \tag{4.5d}$$

By substituting (4.4a)-(4.5d) into (4.3a)-(4.3c) and separating the sine and cosine terms, the following equations are obtained:

$$\frac{4}{\pi} \sin(\pi \cdot d) V_g = i_{Lrs} r + L_r \left(\frac{di_{Lrs}}{dt} - \omega_s i_{Lrc}\right) + v_{Crs} + L_m \left(\frac{di_{Lms}}{dt} - \omega_s i_{Lmc}\right) \tag{4.5e}$$

$$0 = i_{Lrc} r + L_r \left(\frac{di_{Lrc}}{dt} + \omega_s i_{Lrs}\right) + v_{Crc} + L_m \left(\frac{di_{Lmc}}{dt} + \omega_s i_{Lms}\right) \tag{4.5f}$$

$$L_m \left(\frac{di_{Lms}}{dt} - \omega_s i_{Lmc}\right) = \frac{4}{\pi} \frac{i_{Lrs} - i_{Lms}}{i_p} V_o \tag{4.5g}$$

$$L_m\left(\frac{di_{Lmc}}{dt} + \omega_s i_{Lms}\right) = \frac{4}{\pi}\frac{i_{Lrc} - i_{Lmc}}{i_p} V_o \tag{4.5h}$$

$$C_r\left(\frac{dv_{Crs}}{dt} - \omega v_{Crc}\right) = i_{Lrs} \tag{4.5i}$$

$$C_r\left(\frac{dv_{Crc}}{dt} + \omega v_{Crs}\right) = i_{Lrc} \tag{4.5j}$$

In addition

$$V_o = \frac{2}{\pi} i_p R_L \tag{4.5k}$$

$$\frac{dV_{co}}{dt} = -\frac{V_{co}}{C_o R_L} + \frac{2i_p}{\pi C_o} \tag{4.5l}$$

4.6 Small Signal Analysis

By substituting (4.5g) into (4.5e), the equation becomes

$$\frac{di_{Lrs}}{dt} = \omega i_{Lrc} - \frac{4}{\pi}\frac{V_o i_{Lrs}}{L_r i_p} - \frac{v_{Crc}}{L_r} + \frac{4}{\pi}\frac{\sin(\pi \cdot d)\, V_g}{L_r} - \frac{i_{Lrs} r}{L_r} \tag{4.6a}$$

For the small signal model to be put into state space form, each state variable needs to be isolated in the equation and represented as the sum of the remaining variables.

Since a ratio between state variables exists in (4.6a) , $\frac{i_{Lrs}}{i_p}$, where

$i_p = \sqrt{(i_{Lrs} - i_{Lms})^2 + (i_{Lrc} - i_{Lmc})^2}$,

it is clear that this relationship of the state variables cannot be put into the states vector, therefore the ratio $\frac{i_{Lrs}}{i_p}$ is put into small signal form with the use of *Taylor Series* expansion. Expanding brackets gives:

$$i_p = \sqrt{i_{Lrs}^2 + i_{Lms}^2 - 2i_{Lrs}i_{Lms} + i_{Lrc}^2 + i_{Lmc}^2 - 2i_{Lrc}i_{Lmc}}$$

After small signal substitution, $i_{Lrs} = I_{Lrs} + \tilde{\imath}_{Lrs}$, $i_{Lrc} = I_{Lrc} + \tilde{\imath}_{Lrc}$, $i_{Lms} = I_{Lms} + \tilde{\imath}_{Lms}$, $i_{Lmc} = I_{Lmc} + \tilde{\imath}_{Lmc}$, and eliminating 2nd order small signal terms, i_p becomes

$$\tilde{\iota}_p = \sqrt{I_{Lrs}^2 + I_{Lms}^2 - 2(\tilde{\iota}_{Lrs}I_{Lrs} + \tilde{\iota}_{Lms}I_{Lms}) + I_{Lrc}^2 + I_{Lmc}^2 - 2(\tilde{\iota}_{Lrc}I_{Lrc} + \tilde{\iota}_{Lmc}I_{Lmc})}$$

Knowing that by Taylor Series expansion

$$\sqrt{1+k} \approx 1 + \frac{k}{2}$$

and grouping DC terms

$$\tilde{\iota}_p = \sqrt{I_{Lrs}^2 + I_{Lms}^2 + I_{Lrc}^2 + I_{Lmc}^2} - \frac{\tilde{\iota}_{Lrs}I_{Lrs} + \tilde{\iota}_{Lms}I_{Lms} + \tilde{\iota}_{Lrc}I_{Lrc} + \tilde{\iota}_{Lmc}I_{Lmc}}{\sqrt{I_{Lrs}^2 + I_{Lms}^2 + I_{Lrc}^2 + I_{Lmc}^2}}$$

$$\therefore \frac{i_{Lrs}}{i_p} = \frac{i_{Lrs}i_{p^*}}{I_p^2} = (\tilde{\iota}_{Lrs} + I_{Lrs})\frac{\tilde{\iota}_p^*}{i_p^2}$$

where $\tilde{\iota}_p^*$ is the conjugate of $\tilde{\iota}_p$ therefore the denominator $I_p^2 = \tilde{\iota}_p \cdot \tilde{\iota}_p^*$ becomes a constant term.

$$\frac{i_{Lrs}}{i_p} = \frac{\tilde{\iota}_{Lrs} + I_{Lrs}}{I_p^2} \cdot \left[\sqrt{I_{Lrs}^2 + I_{Lms}^2 + I_{Lrc}^2 + I_{Lmc}^2} + \frac{\tilde{\iota}_{Lrs}I_{Lrs} + \tilde{\iota}_{Lms}I_{Lms} + \tilde{\iota}_{Lrc}I_{Lrc} + \tilde{\iota}_{Lmc}I_{Lmc}}{\sqrt{I_{Lrs}^2 + I_{Lms}^2 + I_{Lrc}^2 + I_{Lmc}^2}}\right]$$

Note: $I_p^2 = \left(\sqrt{I_{Lrs}^2 + I_{Lms}^2 + I_{Lrc}^2 + I_{Lmc}^2}\right)^2$

Second order terms and DC constant terms are eliminated to give

$$\frac{\tilde{\iota}_{Lrs}}{\tilde{\iota}_p} = \frac{\tilde{\iota}_{Lrs}}{I_p} + \frac{\tilde{\iota}_{Lrs}I_{Lrs}^2}{I_p^3} + \frac{\tilde{\iota}_{Lms}I_{Lms}I_{Lrs}}{I_p^3} + \frac{\tilde{\iota}_{Lrc}I_{Lrc}I_{Lrs}}{I_p^3} + \frac{\tilde{\iota}_{Lmc}I_{Lmc}I_{Lrs}}{I_p^3}$$

Equations for $\frac{\tilde{\iota}_{Lrc}}{\tilde{\iota}_p}$, $\frac{\tilde{\iota}_{Lms}}{\tilde{\iota}_p}$ and $\frac{\tilde{\iota}_{Lmc}}{\tilde{\iota}_p}$ can be derived using the same algorithm.

By substituting $i_{Lrs} = I_{Lrs} + \tilde{\iota}_{Lrs}$, $i_{Lrc} = I_{Lrc} + \tilde{\iota}_{Lrc}$, $i_{Lms} = I_{Lms} + \tilde{\iota}_{Lms}$, $i_{Lmc} = I_{Lmc} + \tilde{\iota}_{Lmc}$, $v_{Crs} = V_{Crs} + \tilde{v}_{Crs}$, $v_{Crc} = V_{Crc} + \tilde{v}_{Crc}$, $v_{Co} = V_{Co} + \tilde{v}_{Co}$, $d = D + \tilde{d}$, $\omega = W + \tilde{\omega}$ where *I, V, D* and *W* represent the steady stage values, the small-signal model becomes:

$$\frac{d\tilde{\iota}_{Lrs}}{dt} = \tilde{\omega}I_{Lrc} + W\tilde{\iota}_{Lrc} - \frac{4}{\pi}\frac{\tilde{\iota}_{Lrs} - \tilde{\iota}_{Lms}}{\tilde{\iota}_p}V_o - \frac{4}{\pi}\frac{I_{Lrs} - I_{Lms}}{I_p}\tilde{v}_o - \frac{\tilde{v}_{Crc}}{L_r} + 2\tilde{d}\frac{\cos(\pi \cdot D)\,V_g}{L_r} - \frac{\tilde{\iota}_{Lrs}r}{L_r} \qquad (4.6b)$$

$$\frac{d\tilde{\iota}_{Lrc}}{dt} = -\tilde{\omega}_s I_{Lrs} - \tilde{\iota}_{lrs}W - \tilde{v}_{Crc} + \frac{4}{\pi}\frac{\tilde{\iota}_{Lrc} - \tilde{\iota}_{Lmc}}{\tilde{\iota}_p}V_o - \frac{4}{\pi}\frac{I_{Lrc} - I_{Lmc}}{I_p}\tilde{v}_o - \frac{\tilde{\iota}_{Lrc}r}{L_r} \qquad (4.6c)$$

$$\frac{d\tilde{\imath}_{Lms}}{dt} = \tilde{\omega}_s I_{Lmc} + \tilde{\imath}_{Lmc}W + \frac{4}{\pi}\frac{\tilde{\imath}_{Lrs} - \tilde{\imath}_{Lms}}{\tilde{\imath}_p L_m}V_o + \frac{4}{\pi}\frac{I_{Lrs} - I_{Lms}}{I_p L_m}\tilde{v}_o \tag{4.6d}$$

$$\frac{d\tilde{\imath}_{Lmc}}{dt} = -\tilde{\omega}_s I_{Lms} - \tilde{\imath}_{Lms}W + \frac{4}{\pi}\frac{\tilde{\imath}_{Lrc} - \tilde{\imath}_{Lmc}}{\tilde{\imath}_p L_m}V_o + \frac{4}{\pi}\frac{I_{Lrc} - I_{Lmc}}{I_p L_m}\tilde{v}_o \tag{4.6e}$$

$$\frac{d\tilde{v}_{Crs}}{dt} = \frac{\tilde{\imath}_{Lrs}}{C_r} + \frac{\tilde{\omega}V_{Crc}}{C_r} + \frac{W\tilde{v}_{Crc}}{C_r} \tag{4.6f}$$

$$\frac{d\tilde{v}_{Crc}}{dt} = \frac{\tilde{\imath}_{Lrc}}{C_r} - \frac{\tilde{\omega}V_{Crs}}{C_r} - \frac{W\tilde{v}_{Crs}}{C_r} \tag{4.6g}$$

$$\tilde{v}_o = \frac{2}{\pi}\tilde{\imath}_p R_L \tag{4.6h}$$

$$d\tilde{v}_{co} = -\frac{\tilde{v}_{co}}{C_o R_L} + \frac{2\tilde{\imath}_p}{\pi C_o} \tag{4.6i}$$

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} & A_{17} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} & A_{27} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} & A_{37} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} & A_{47} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} & A_{57} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} & A_{67} \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} \end{bmatrix}$$

A complete representation of the matrix equations is shown in the appendix.

$$x = \begin{bmatrix} \tilde{\imath}_{Lrs} \\ \tilde{\imath}_{Lrc} \\ \tilde{\imath}_{Lms} \\ \tilde{\imath}_{Lmc} \\ \tilde{v}_{Crs} \\ \tilde{v}_{Crc} \\ \tilde{v}_{co} \end{bmatrix}$$

$$B = \begin{bmatrix} 2\cos\left(\frac{\pi D}{2}\right) & I_{Lrc} \\ 0 & -I_{Lrs} \\ 0 & I_{Lmc} \\ 0 & -I_{Lms} \\ 0 & V_{Crc} \\ 0 & -V_{Crs} \\ 0 & 0 \end{bmatrix}$$

$$u = \begin{bmatrix} \tilde{d} \\ \tilde{\omega} \end{bmatrix}$$

$C = [\,0\,0\,0\,0\,0\,0\,1]$

To compute the steady state values of the currents and voltages the derivatives inside circuit equations (4.5e) to (4.5*l*) were set to zero. These produced 7 equations with 7 unknowns with the state variables placed inside the x-vector and the constants inside matrix B. Using matrix multiplication and inversion the state variables were computed.

$$Ax = B \rightarrow x = A^{-1}B$$

4.7 Results and Discussion

The results of the EDF MATLAB model are shown in figures 4-3 to 4-8. The duty cycle is kept constant at 50% and the frequency is varied from 142kHz to 147kHz. Since, the EDF model does not account for the filter on the secondary side of the transformer, the circuit in Figure 2- 3 was simulated in PSIM without the output filter to ensure proper comparison.

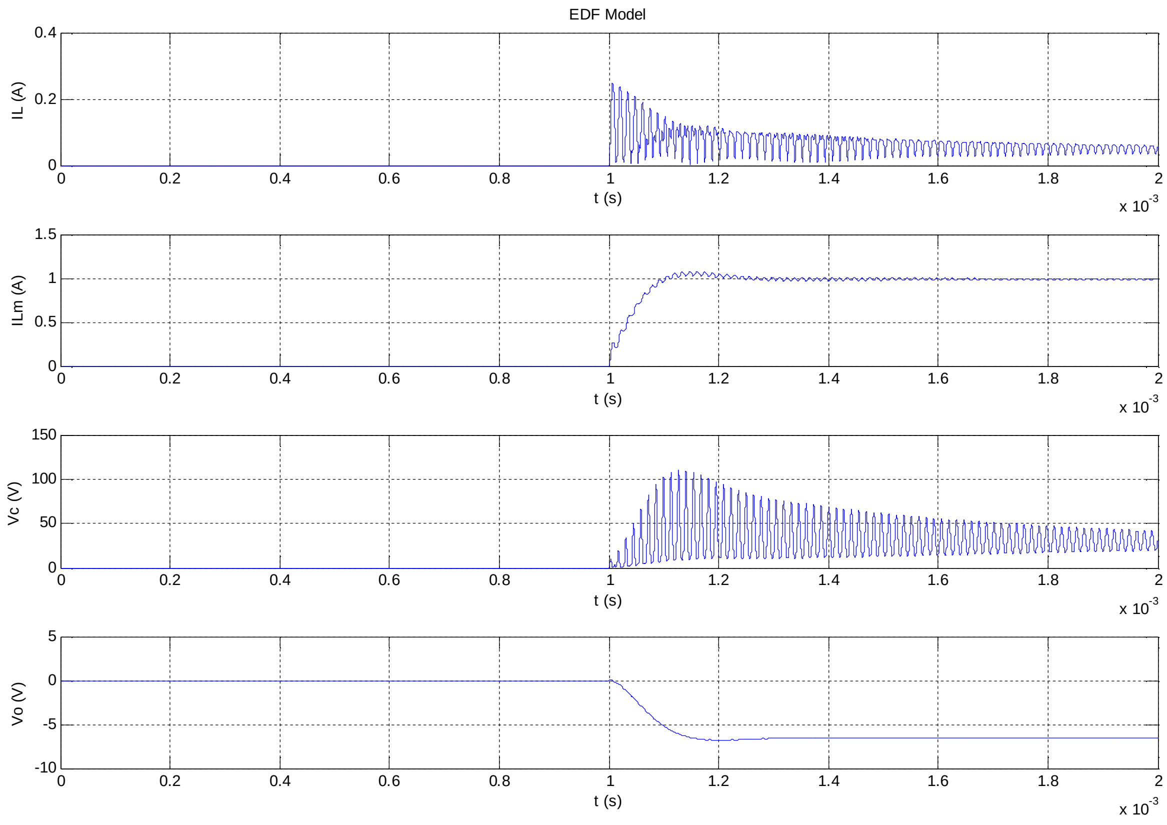


Figure 4- 3 : EDF model currents and voltages during frequency change (142-147KHz)

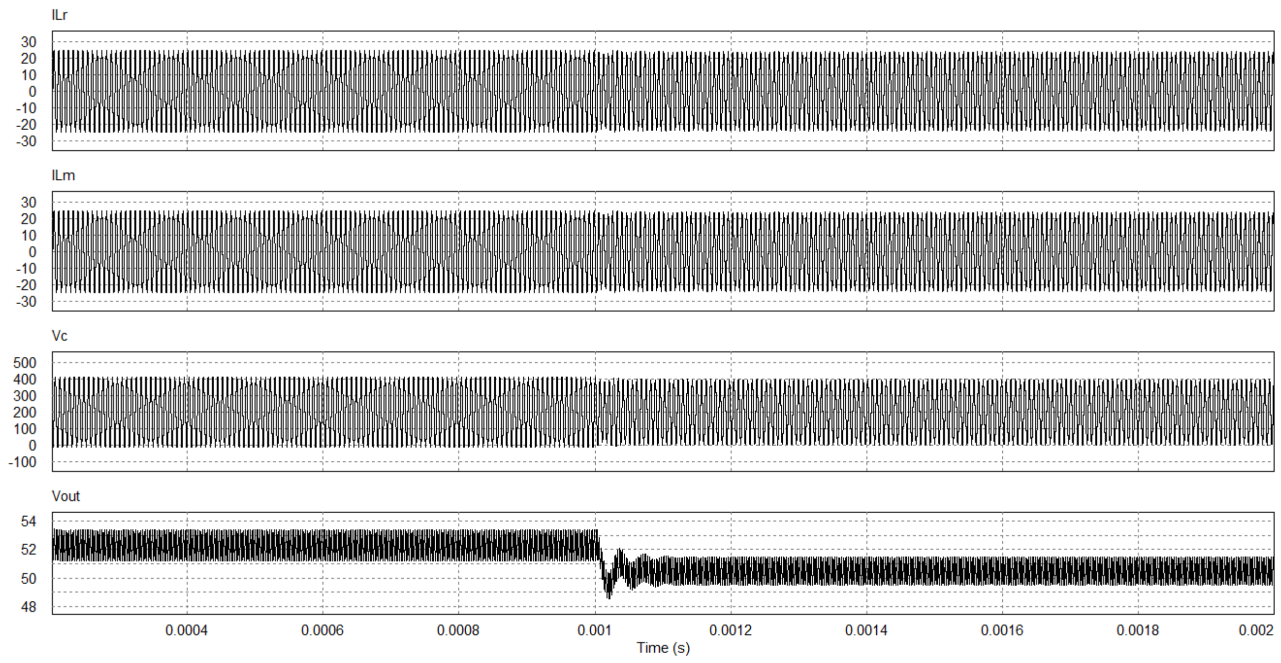


Figure 4- 4 : PSIM model currents and voltages during frequency change (142-147KHz)

As with other EDF models the results from this approach do not match the circuit dynamics obtained in PSIM. The behaviour of the output voltage with respect to frequency change is similar in PSIM and the MATLAB model but with a different magnitude. When the switching frequency is increased by 5kHz, the output voltage drops in both simulations; the rest of the variables however, have very different response. Since the EDF model ignores any transients and focuses on the average values, the current waveforms are expected to oscillated after a disturbance and then settle around zero amperes. As seen in the figures above, it is noticeable that this is not the case in the EDF model.

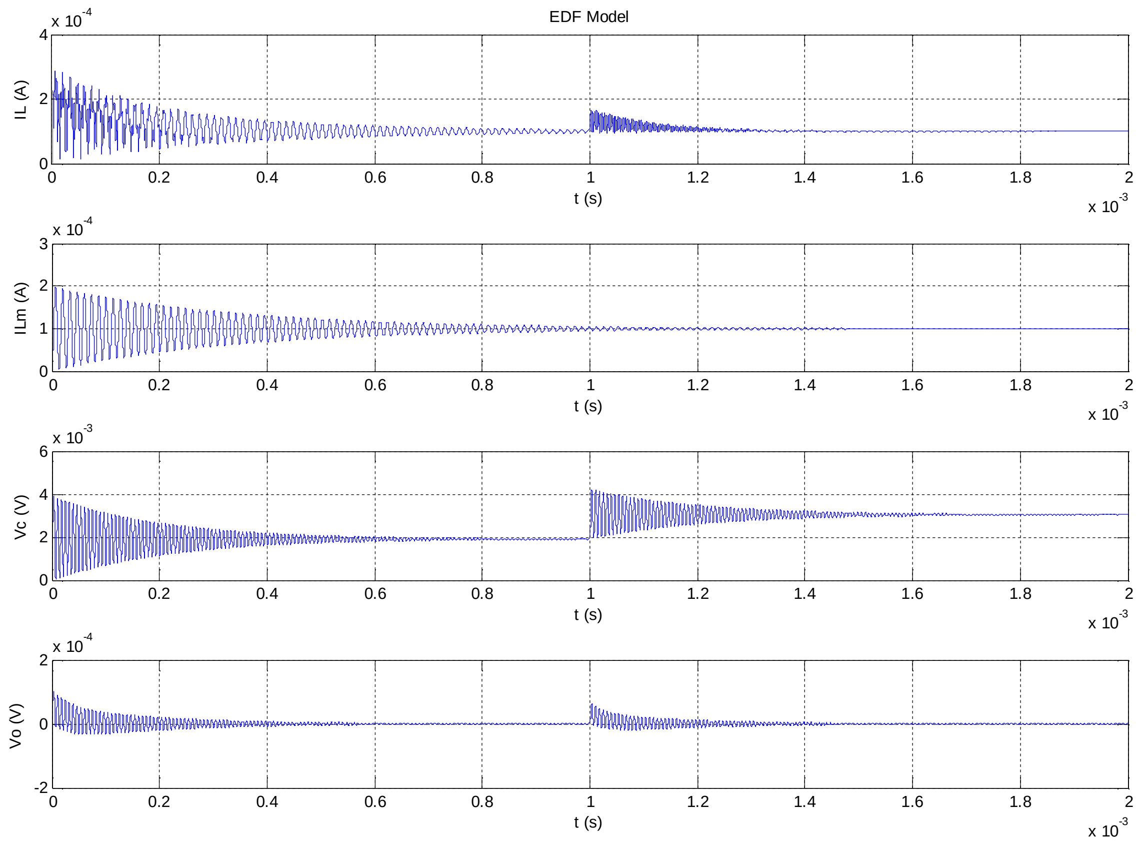


Figure 4- 5 : EDF model during duty cycle change (50%-80%)

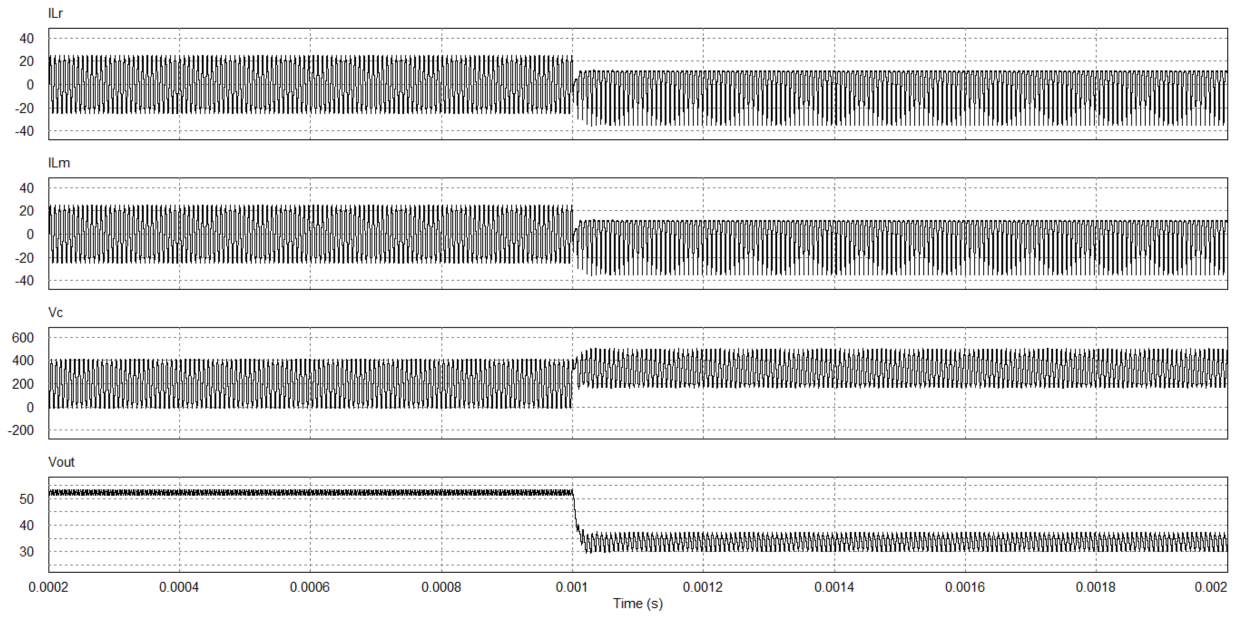


Figure 4- 6 : PSIM model during duty cycle change

As in the frequency disturbance case, during duty cycle changes only some of the parameters behave correctly. In this case, the amplitude of the resonant capacitor voltage shows the correct behavior as a result of a duty cycle disturbance where the rest of the parameters are not exact.

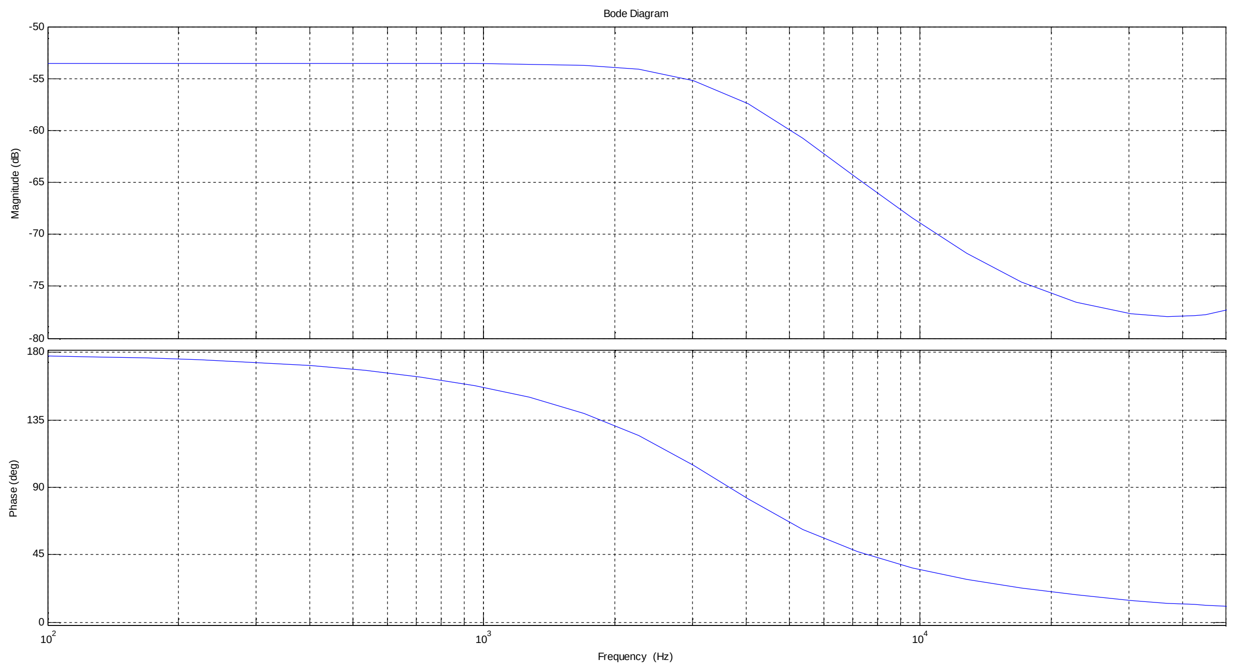


Figure 4- 7 : EDF model Bode plot

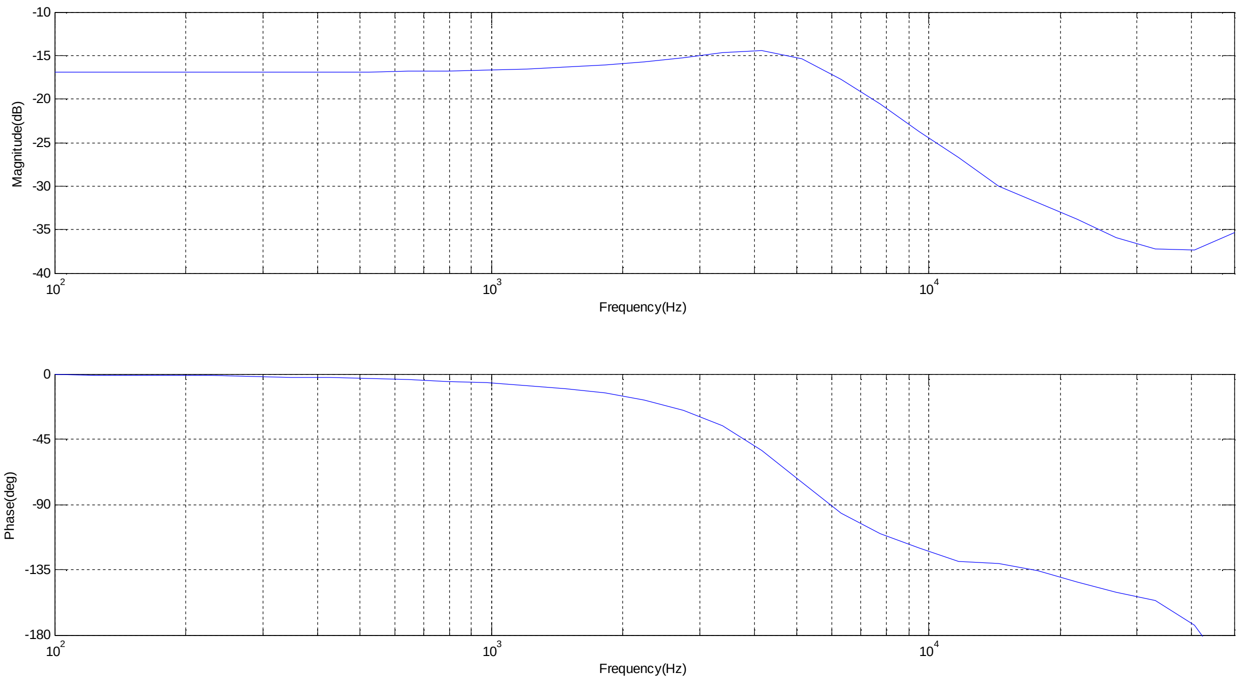


Figure 4- 8 : PSIM Bode plot

A closer look at the bode plots acquired by PSIM and the model at hand show some similarities, especially in the phase comparison. Previous attempts by others and the results in this section show that the EDF approach has the potential to succeed at modelling resonant power converters; but it does need some additional configuration before it can be used to represent the characteristics of the LLC resonant converter discussed in this work.

In addition to showing the time domain characteristics of the circuit, it is important to have its frequency response. A closed loop controller for the circuit can be easily designed if the transfer function of the LLC at varying operating conditions was available.

Since it appears that the EDF model with the configuration described in this section cannot be used to describe the behavior of the LLC circuit a new approach is needed; possibly a method that focuses on the frequency response instead of the behavior of the circuit in time domain. The upcoming chapter describes a technique called the Least Squares Parametric Estimation which can provide such a solution.

CHAPTER 5 Least Squares Parametric Estimation

5.1 Introduction

Modeling the LLC converter via methods such as Average-State-Space modeling or an approach based on the Extended Describing Function has proven inadequate. Both of the modeling techniques used in the previous chapters do not provide adequate representation of the LLC circuit response when exposed to disturbances such as load variation, frequency control or duty control. The main goal of this research is to obtain the circuit dynamics in time and frequency domain along with the transfer function which in turn will help in the design of an adaptive or robust controller for the LLC. Hence, due to the lack of information given by the time domain models explored thus far, a method based on capturing the transfer function and frequency dynamics was explored.

The transfer function of a mathematical model serves as an effective tool for analysis of control systems. With the transfer function available, the system can maintain desired operational conditions regardless of disturbances. Furthermore, a mathematical model of a system has to be updatable in response to a change in the system's variables. The Least Squares Method (LSM) is one of several approaches that exhibit this behavior.

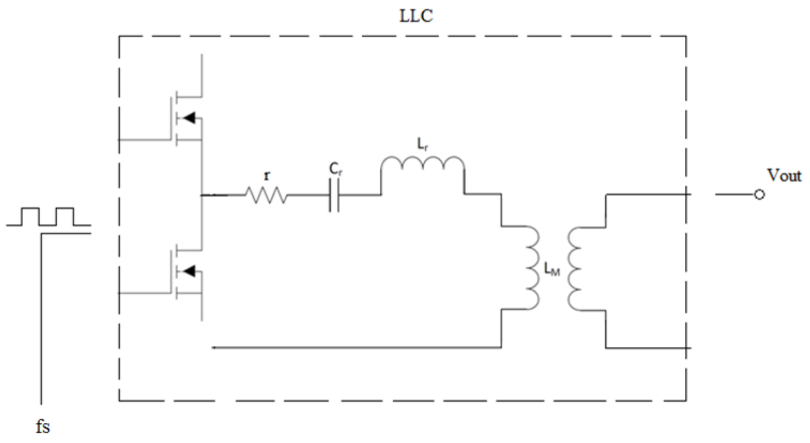


Figure 5- 1 : LSM Model diagram

The LSM is based on data points captured at the input of the system and data points captured at the output of the system. These data are then placed into an array as such

$$u = [\ x_1(t), x_2(t), x_3(t), \dots x_n(t)\ ]$$

$y = [\ y_1(t), y_2(t), y_3(t), \dots\ y_n(t)\ ]$

A typical transfer function of a given system can be expressed as

$$\frac{Y(z)}{U(z)} = \frac{b_2 z^2 + b_1 z + b_0}{z^3 + a_2 z^2 + a_1 z + a_0} \tag{5.1a}$$

$$Y(k) = a_2 Y(k-1) + a_1 Y(k-2) + a_0 Y(k-3) + b_2 U(k-1) + b_1 U(k-2) + b_0 U(k-3) \tag{5.1b}$$

This can be expressed in matrix form as:

$$y = Ax \tag{5.1c}$$

where x contains the coefficients of the transfer function.

$$A^{-1}y = A^{-1}Ax$$

$$\therefore x = A^{-1}y \tag{5.1d}$$

In a case with many input and output samples the matrices become:

$$y_n = \begin{bmatrix} y(1) \\ y(2) \\ \dots \\ y(n) \end{bmatrix}$$

$$A_n = \begin{bmatrix} x_1(1) & x_2(1) \dots & x_m(1) \\ x_1(2) & x_2(2) \dots & x_m(2) \\ \dots & \dots & \dots \\ x_1(n) & x_2(n) \dots & x_m(n) \end{bmatrix}$$

$$x = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

In this case the input vector contains

$x_1(k) = U(k-3)$, $x_2(k) = U(k-2)$, $x_3(k) = U(k-1)$, $x_4(k) = Y(k-3)$, $x_5(k) = Y(k-2)$ and $x_6(k) = Y(k-1)$

where k is a variable with range 1 to n.

Since the inverse of a matrix requires the matrix to be square ($n_x n$), the pseudo-inverse is used to convert A into a square matrix.

$$A^T y = A^T A x \tag{5.1e}$$

$$(A^T A)^{-1} A^T y = (A^T A)^{-1} (A^T A) x$$

A matrix multiplied by its inverse becomes the identity matrix, therefore the matrix containing the transfer function coefficients is given by

$$x = (A^T A)^{-1} A^T y \tag{5.1f}$$

To ensure invertability of the matrix the rank of the square matrix $(A^T A)$ must equal n. The rank evaluates the linear independence of the rows, a condition for invertability. Using equation (5.1f), the LSM can be used to calculate the values of parameters b_0, b_1, b_2, a_0, a_1, a_2 and produce an approximation of the system's open loop transfer function. This transfer function can then be used to design a controller for the given circuit.

5.2 Second Order Filter

In frequency control mode of the LLC, the change in switching and hence sampling frequency poses a big challenge in calculating the model parameters. Additionally, high frequency noise in the signal might produce aliasing effects. This makes the LSM algorithm prone to errors. When capturing the data the sampling frequency was kept constant and fixed at 1MHz.

For the LSM algorithm to produce a correct solution of the given system the following criteria must be satisfied:

I. Input values must be noise free
II. The sampling frequency must be fixed

Therefore, in addition to the LSM algorithm, a 2nd order Butterworth low pass filter was incorporated into the model to eliminate any high frequency components introduced by the difference in sampling. The general form of a Butterworth filter is given by

$$H(s) = \frac{G_o}{\sqrt{1 + \left(\frac{\omega}{\omega_c}\right)^{2n}}} \tag{5.2a}$$

where Go is the DC gain, n is the order of the filter, ω_c is the cut-off frequency and $\omega = 2\pi f$.

In MATLAB

[b, a] = butter(n,Wc) ;

produces an n order low-pass Butterworth filter with normalized cut-off frequency W_c. The filter coefficients are stored in the vectors b and a. Figure 5- 3 shows a comparison between the Bode magnitude and phase plot of the PSIM software, LSM model without filtered parameters and the LSM model with its values filtered through a 2nd order Butterworth low pass filter (BLPF).

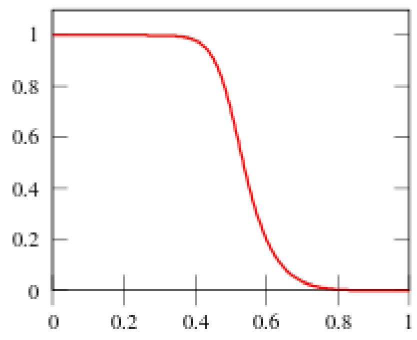


Figure 5- 2 : Butterworth filter frequency response

The low pass filter passes frequencies lower than the cut-off and reduces the amplitude of signals with frequencies higher than the cut-off frequency. The Butterworth filter was chosen due to its flat magnitude response in the pass-band region.

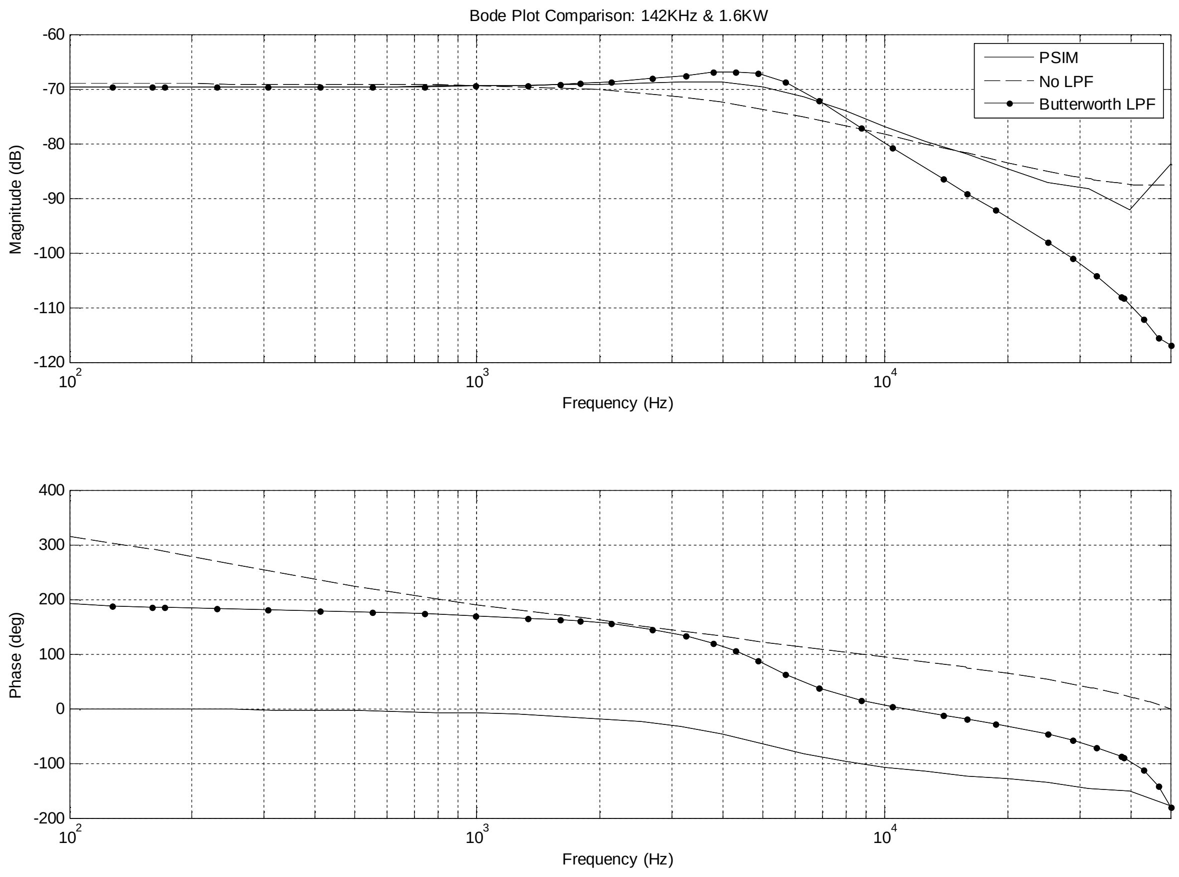


Figure 5- 3 : Bode plot comparison between PSIM and LSM models with and without filtered data

Clearly, with the help of the BLPF the model is able to capture the complex poles, causing a maxima value at 5kHz, similar to the PSIM plot, but with a slightly higher magnitude. Improved performance can be obtained by matching the sampling frequency between PSIM and the LSM model as well as by including more data points (i.e. sampling at a higher frequencies). A similar behavior can be observed in the phase plot. The response of the filtered LSM model is a much closer approximation of the response of the PSIM simulation, although the starting point is offset by 180 degrees.

5.3 Simulation Results

For the LSM model to be valid it has to closely resemble the results obtained by the PSIM software. If successful, the LSM model would produces frequency and time domain plots that match those obtained in PSIM. The advantage of the LSM model is that it will also produce the transfer function of the circuit, where PSIM only gives the user the frequency and time response plots. Since the goal is to obtain the circuit's transfer function which is to be used in the design of a feedback controller, the transfer function is essential.

The diagrams to follow will show magnitude and phase plots as well as voltage vs. time, compiled using PSIM and the LSM model computed in MATLAB. The similarities and differences will be explained.

Since there are three active components in the circuit, *Lr*, *Lm*, *Cr*, the LSM algorithm was set to produce a 3rd order transfer function in the form:

$$\frac{Y(z)}{U(z)} = \frac{b_2 z^2 + b_1 z + b_0}{z^3 + a_2 z^2 + a_1 z + a_0} \tag{5.3a}$$

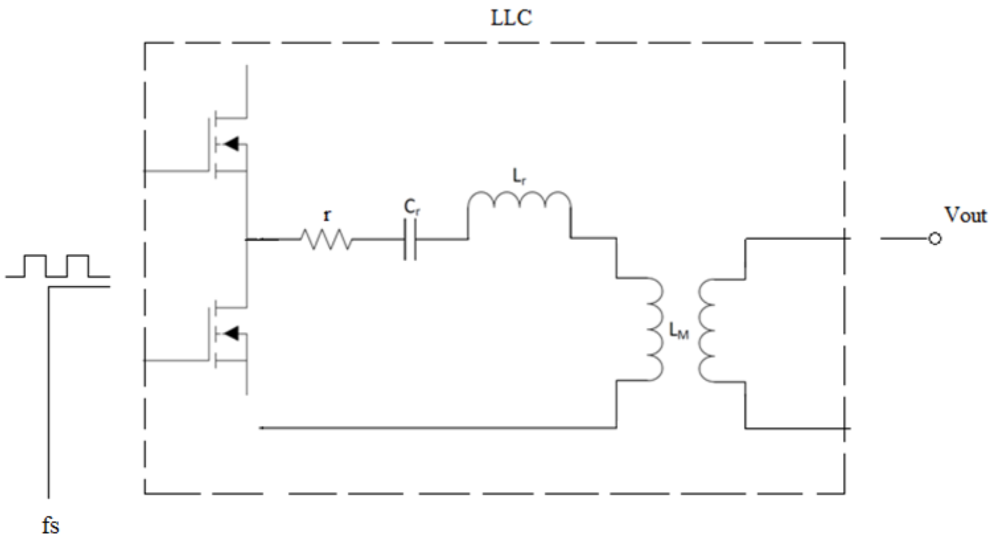


Figure 5- 4 : LSM diagram

The diagram above shows a simpler representation of the LSM model. In this case, the frequency was used as the input, with varying values stored in a vector matrix, and the output voltage represented the output values. As previously mentioned, the LSM model is a simple solution to the circuit dynamics in frequency domain by only dealing with the interaction between inputs and outputs of the circuit and not the individual components. In theory, one can replace the LLC circuit with another circuit and obtain a successful frequency domain model of the circuit using the same algorithm.

5.3.1 PSIM vs. LSM Modelling of the LLC

To perform LSM on the LLC power converter, the PSIM input and output data were used as the input and output vectors. The input values were represented by the switching frequency and the output by the output voltage, V_{out}. Two tests were done to observe the behavior of the power converter. First, the switching frequency was increased in a step fashion by 5 kHz during the simulation. This test was performed for switching frequencies of 120 kHz, 142 kHz and 195 kHz. Since the resonant frequency of the circuit is at 142 kHz, it is desired to observe circuit behavior at resonance, above resonance and below resonance. Second, the output resistance was increased, keeping the switching frequency the same throughout the simulation at 120 kHz, 142 kHz and 195 kHz. A portion of the results are shown in this section, with additional findings in the Appendix pages.

Figure 5- 5 shows the output voltage response in both the PSIM simulation software and the LSM approximation model. The simulation was performed in PSIM, using 142 kHz as the switching frequency, which increased by 5 kHz to 147 kHz at time t = 30ms. The disturbance in the frequency caused a decrease in the output voltage from 52V to 50V. The data points collected from the simulation in PSIM were used to generate the LSM model and the transfer function of the system.

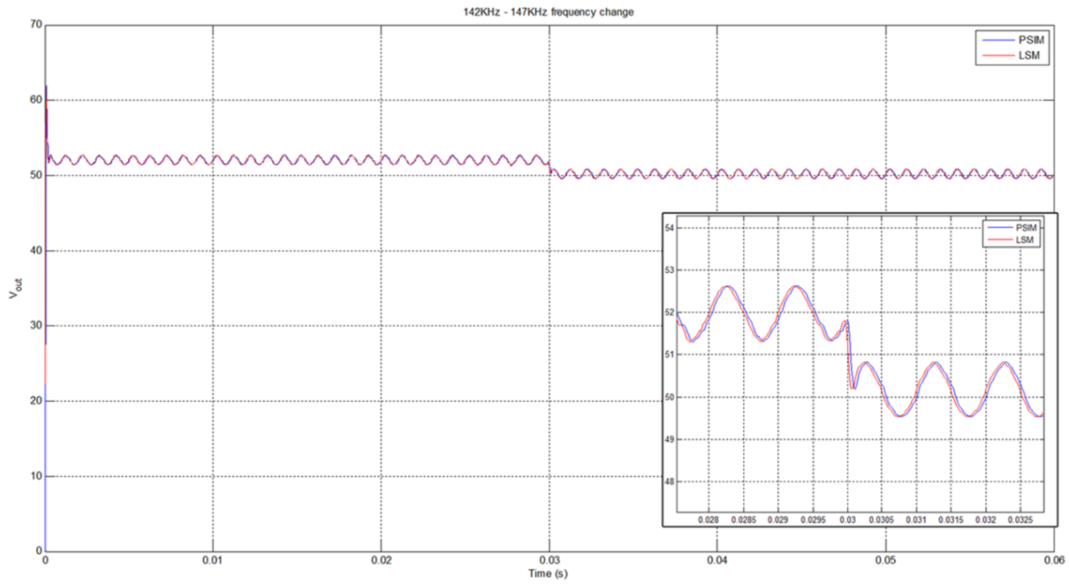


Figure 5- 5 : Output voltage at frequency change, PSIM vs. LSM approximation

The frequency response of the LLC circuit was compared between PSIM and LSM to analyze the accuracy of the proposed model. Figure 5- 6 shows the results of the PSIM simulation. Three load conditions were tested: 1.6kW, 2.0kW and 2.4kW.

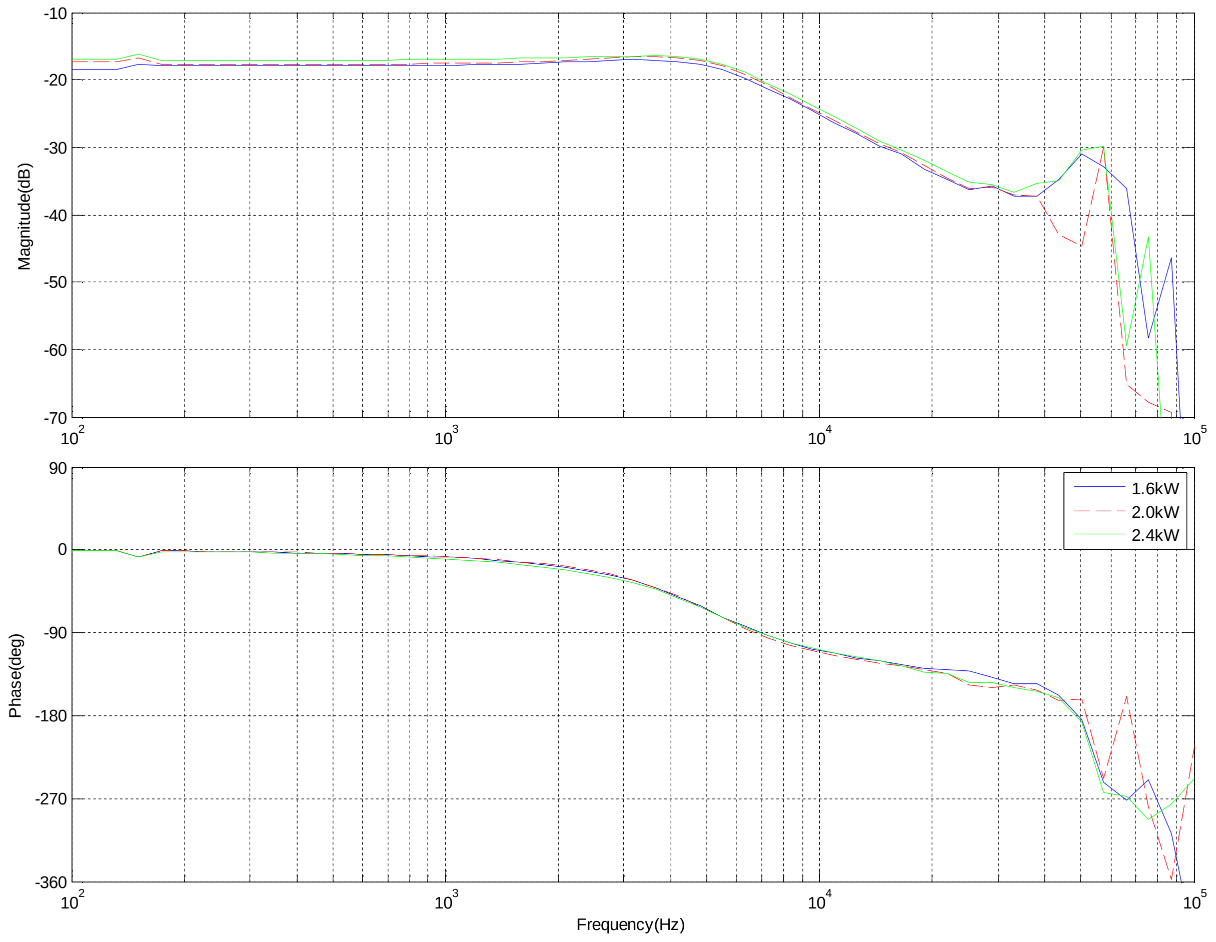


Figure 5- 6 : PSIM magnitude and phase Bode plot at 142KHz switching frequency

The results from PSIM were compared against the LSM approximation model in Figure 5- 7.

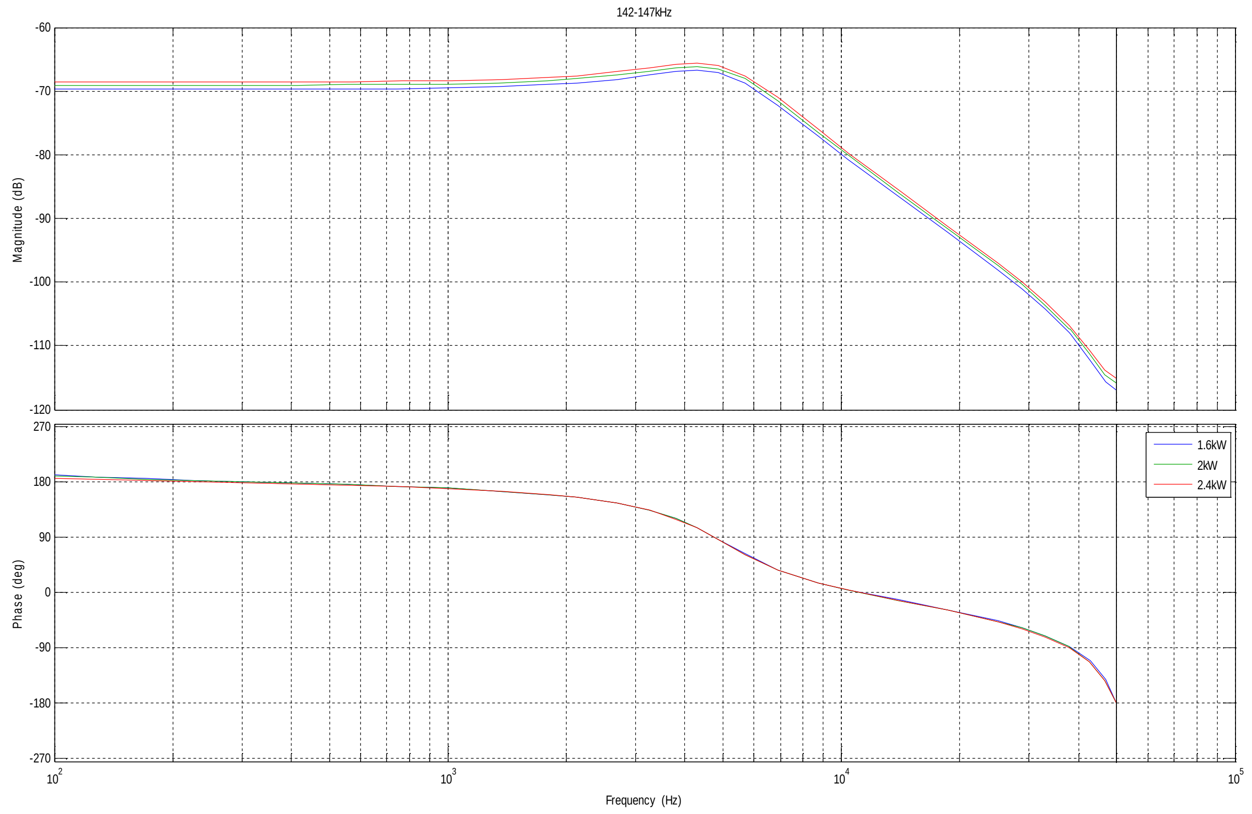


Figure 5- 7 : LSM magnitude and phase Bode plot at 142KHz switching frequency

Transfer Function of 1.6kW load:

$$-\frac{1.2361\cdot 10^{-5}(z+1.394)(z-1.001)}{(z-0.9994)(z^2-\ 1.689z+0.7799)} \tag{6.3b}$$

a0	0.7794	b0	$1.724\text{x}10^{-5}$
a1	-2.4578	b1	$-4.9643\text{x}10^{-6}$
a2	2.6884	b2	$-1.2360\text{x}10^{-5}$

The general shape of the bode plots in PSIM and those computed by the LSM exhibit similar behaviour. Both appear to have complex poles at around the same value of 4-5kHz. The major difference is that of a dominant pole at 40-50kHz. This is caused by the LC filter at the secondary side of the transformer, prior to the load resistor. The LSM model misses to capture this pole because of a sampling difference between PSIM and the values used in MATLAB. But it is safe to ignore this since it plays no role in system stability. The negative gain in the transfer function introduces a phase shift of -180°. As already explained, the LSM uses the time domain values of PSIM to perform its approximation algorithm. PSIM is forced to capture values at 100KHz by simulating at 1^{-7}s time step and capturing every 100th sample. This sampling difference causes not only slightly different pole/zero values between PSIM and LSM but also a difference in the magnitude. Yet, sampling at higher frequencies might demonstrate better performance. It should be noted that higher sampling frequency will also increase the number of samples and simulation time in both PSIM and LSM.

Next we observed resistive load change at the output of the LLC. The plot below represents the change in output voltage when the output resistance was increased from 1.04Ω to 2.04Ω. It appears that load change did not affect the value of the output voltage as much as switching frequency disturbance in the previous section. It is possible that the initial value of the load has an effect on the voltage variation. If the resistor is given an initial value of 100Ω or 1000Ω, much higher than the 1Ω, the circuit's behavior will be affected since the quality factor , Q, will be of a much different value.

Once again, the LSM model provides a satisfactory approximation of the PSIM plot. In fact, the shape of the LSM generated waveform resembles a delayed version of the PSIM results.

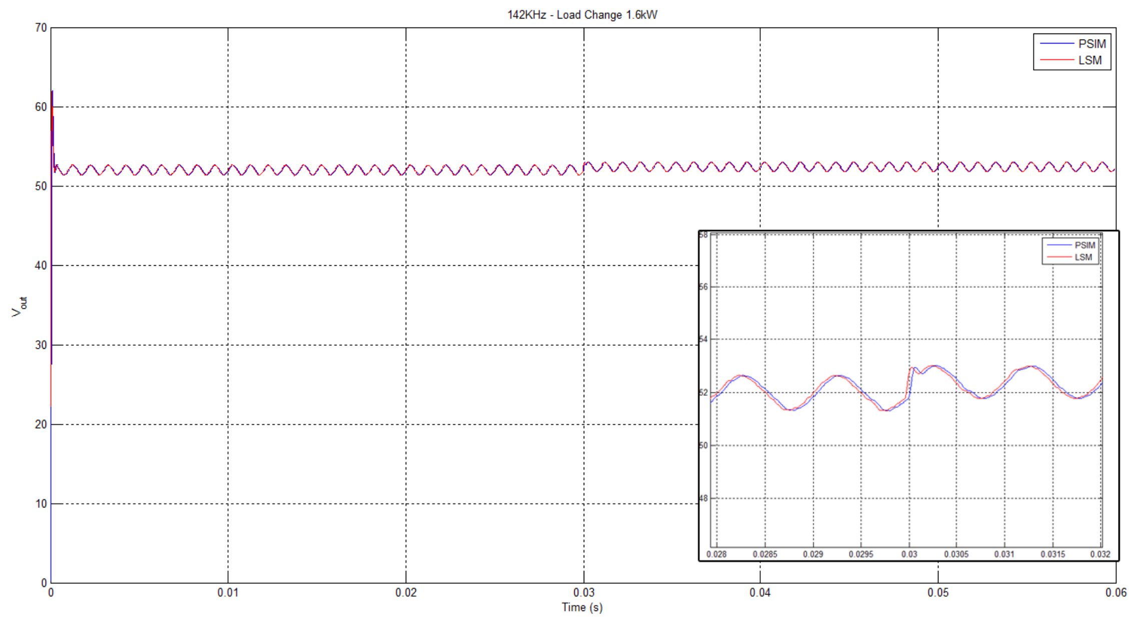


Figure 5- 8 : Output voltage with increased load PSIM vs. LSM approximation

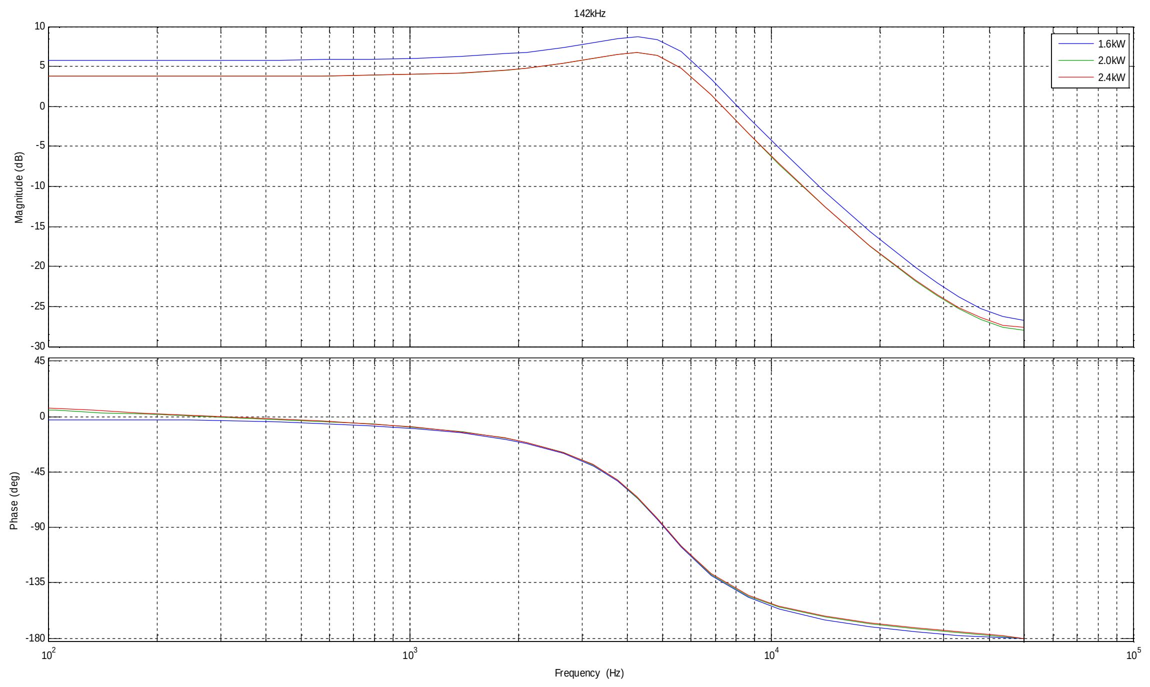


Figure 5- 9 : LSM model frequency response during load change

During load change, once again the high frequency pole was not captured by the LSM model and a difference in the magnitude is evident. In addition, results at different switching frequencies were compared in both the LSM model and PSIM.

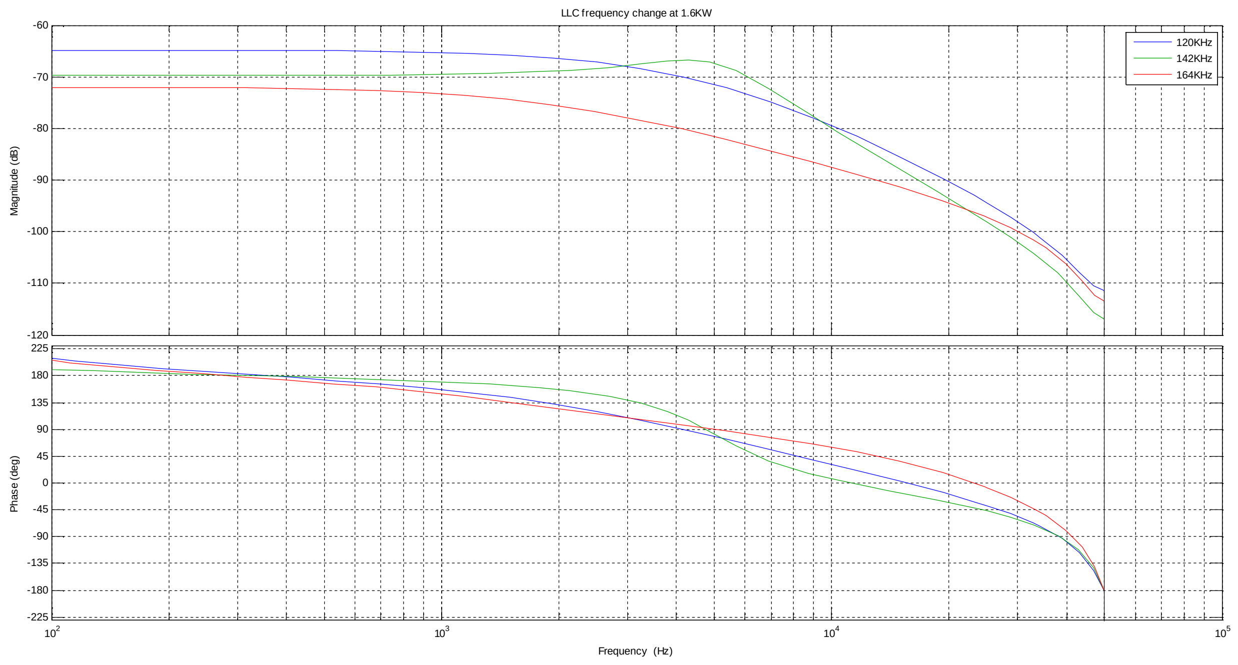


Figure 5- 10 : LSM Bode plot comparison: Below fo, above fo, at fo

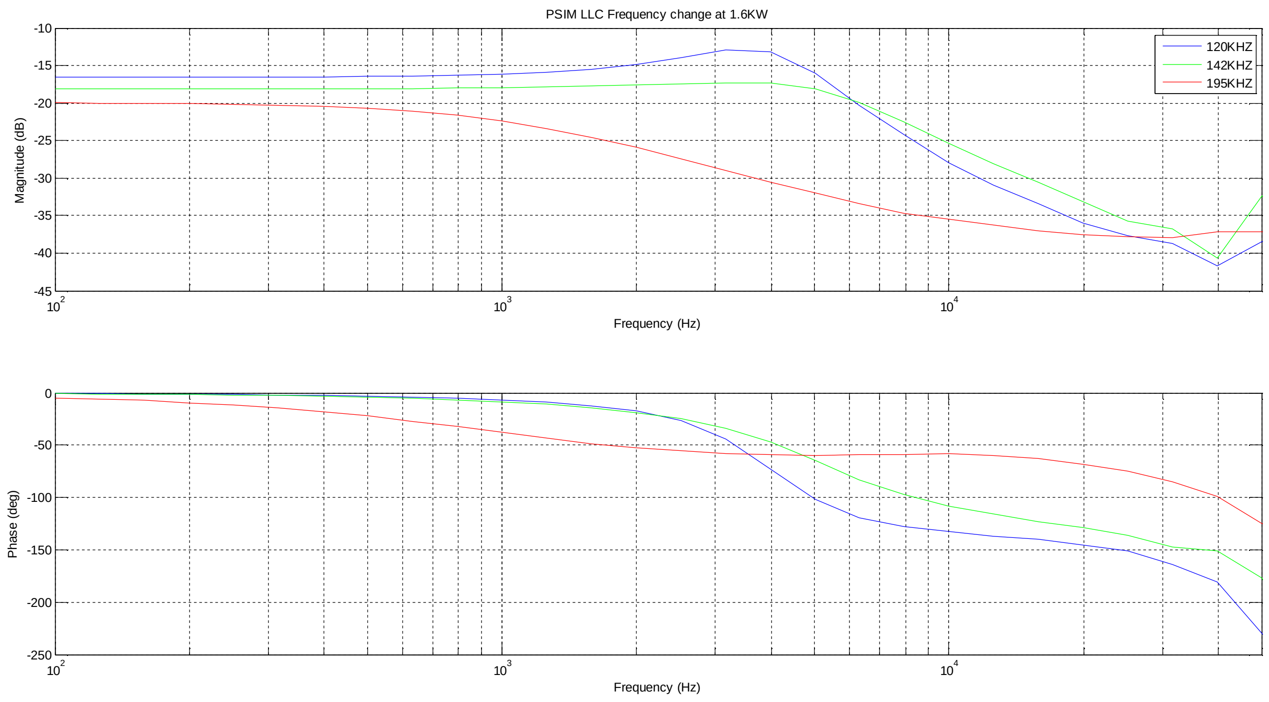


Figure 5- 11 : PSIM Bode plot comparison: Below fo, above fo, at fo

A major difference between PSIM and the LSM approximation appears in the 120kHz plot. As observed from the figures above, the LSM was unable to capture the complex poles, and represent the maxima at 4kHz, the same way it was represented in the 142kHz solution. The main reason behind this behavior is the mismatched sampling frequency between PSIM and the LSM. If both were sampled at a higher frequency and both used the exact same data points, a closer match is possible. Nonetheless, the model provides a satisfactory approximation for the 142kHz and 195kHz simulations. The phase plot comparison between PSIM and LSM shows a more accurate approximation. The phase plots of both simulations show similar behavior and approximately the same pole/zero placements. The only difference appears in the starting point. The PSIM software shows a start at 0°, where the LSM model has a 180° start and it goes through a change of 360°, compared to 180° in PSIM. This is caused by the -ve sign captured in the LSM model which is not present in the PSIM model mainly due to noise injection and measurement location.

Additionally, as stated in the earlier sections of the paper, the LSM model fails to capture the poles caused by the LC filter on the secondary side of the transformer. The values of the LSM model continue to decrease while the PSIM model shows a rise in magnitude past 40kHz. A higher sampling frequency can improve the response of the LSM approximation.

The results in this section show that the LSM model provides an accurate approximation of the frequency response of the LLC resonant converter. With the exception of some minor differences, the model and the approximation of the system's transfer function can be successfully used to obtain a closed-loop controller for the LLC circuit.

Before designing a control algorithm it is desirable to evaluate the performance of the LLC resonant converter when controlled digitally. Digital control via a digital signal processor (DSP) introduces delays, quantization errors and other digitizing effects into the system. The next chapter will focus on accurately portraying these effects in simulation. The discussion and results in the following sections can be used to further improve the control algorithm for digitally controller resonant converters such as the LLC.

CHAPTER 6 Digitizing Effects of an Analog-to-Digital Converter

6.1 Introduction

To design a closed loop controller, the output of the system is typically fed back into the control box where a control algorithm is executed and the output signal of the control box adjusts the circuit's performance accordingly. In the case of power converters, the output voltage or current is typically measured to set the behaviour of the controller. If the output voltage or current is to be kept at a certain value, if that value changes, the controller is designed to adjust its settings and bring the output parameter back to its original position.

In the typical LLC converter the controller of the circuit sends a square waveform with 50% duty cycle and particular frequency to the gate of the MOSFETs. The output voltage is continuously monitored and if for any reason it is not at the desired value the controller uses frequency modulation to adjust the voltage amplitude.

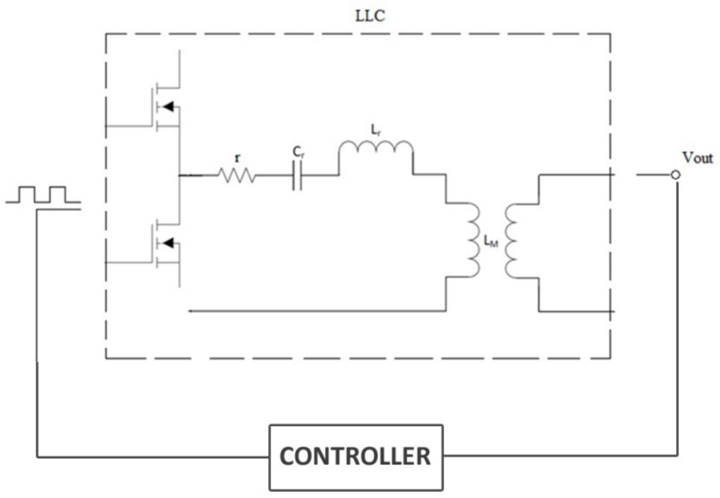


Figure 6- 1 : LLC control diagram

In digital control, the control box is represented by a digital signal processor (DSP). These controllers are implemented using stored computer code rather than analog components, reducing circuit size, complexity and cost.

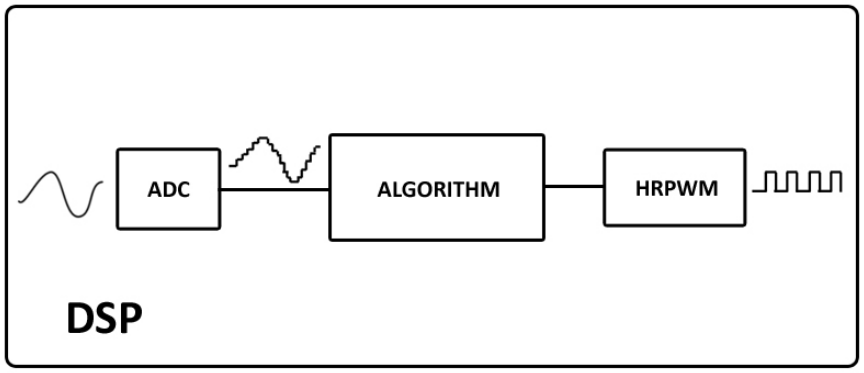


Figure 6- 2 : DSP diagram

Unlike analog controllers where the output of the power converter is fed directly into the components and control is established, in digital control the signal fed into the controller needs to be modified before it can be used. Since the signal is in analog form it must be converted to digital. This is done with the help of an Analog-to-Digital converter (ADC), located inside the DSP.

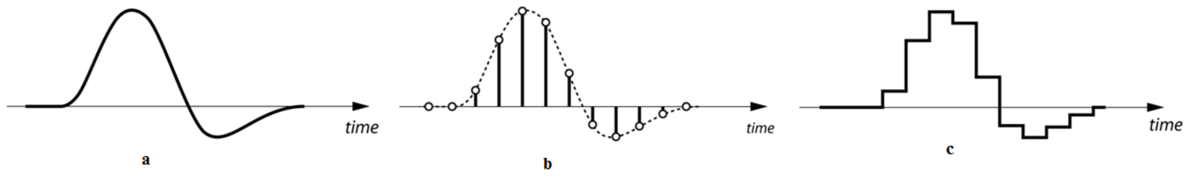


Figure 6- 3 : Analog-to-Digital Conversion

To convert the signal from analog to digital, the ADC samples the incoming waveform at some sampling frequency, typically in the kHz or MHz range for power converters. The higher the frequency the more accurate the sampled waveform. After the sample is captured, the ADC holds its value until the next sample. This is called the Sample and Hold technique. Diagrams a), b) and c) in Figure 6- 3 represent each stage of the process. The Zero-order-hold (ZOH) is the simplest method in A/D conversion[9].

In addition, each ADC has a set resolution. These can be 4bit, 8bit, 12bit etc. The resolution defines the number of levels to which the ADC can quantize the input signal. In an 8bit ADC this is

2^8 or 256 levels that can be used to map an analog signal to one of a digital nature. The resolution, R, can be calculated by

$$R = \frac{V_{max} - V_{min}}{2^n - 1} \tag{7.1b}$$

where n represents the ADC's resolution in bits and (V_{max} - V_{min}) is the analog voltage range. In the 8bit example with maximum analog voltage of five volts and minimum of zero volts

$$R = \frac{5}{255} = 0.0196 = 19.6mV$$

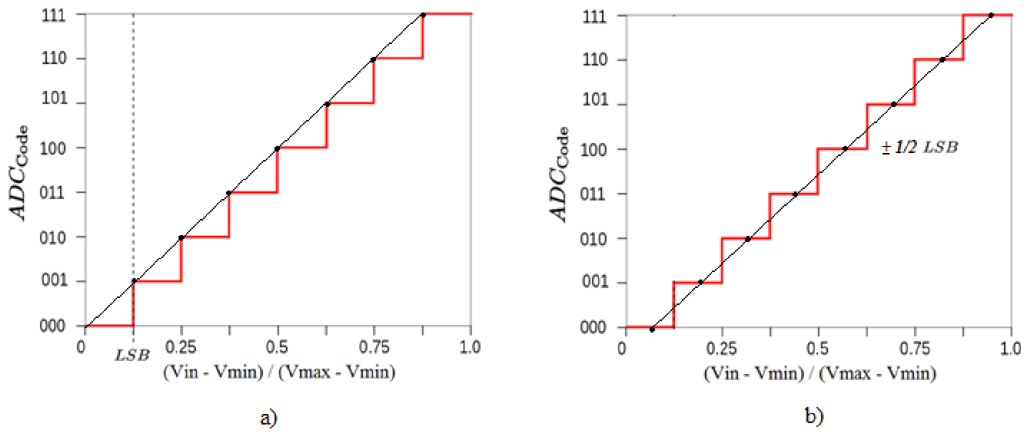


Figure 6- 4 : 8 levels of quantization

Figure 6- 4a) shows how the ADC encodes an analog signal. It shows the scheme for 8 levels of quantization or 3bits. The horizontal scale represents the normalized voltage and the vertical scale is the ADC code in binary digits. The diagram shows another variable, the least significant bit (LSB). The LSB represents the voltage value of the resolution R. It is also said that the quantization error is equal to a value between one and zero LSB. Some improvement is achievable if the "staircase" values are shifted by 1/2 LSB, making the error in the range of ± 1/2 LSB. This quantization method is shown in Figure 6- 4b).

Although digital control provides a more flexible control solution, decreases the number of components and in some cases uses less power, during A\D conversion quantization errors are introduced into the circuit. Since the incoming analog wave is sampled, it is possible to miss important points on the curve.

6.2 Implementation

It is clear that a more accurate representation of the LLC resonant converter would include the quantization errors caused by the analog-to-digital converter. This section will focus on implementing a programmable block in PSIM to simulate the behavior of an ADC.

Although PSIM includes an ADC module, it can only perform at constant sampling frequency. In some applications it is desirable to vary the ADC sampling frequency in the hardware to save time and improve performance. If the dynamics of the circuit are known, the ADC can be programmed to use higher sampling rate around transients and lower sampling rate when the input analog waveform is in a plateau region.

Fortunately, PSIM allows the user to generate a programmable block. Inside this module the user can insert an algorithm written in C/C++ to perform a given task. In this case the programmable block also called .DLL block is used to simulate the performance of an analog-to-digital converter with user specified parameters such as resolution bits, sampling frequency and input voltage range.

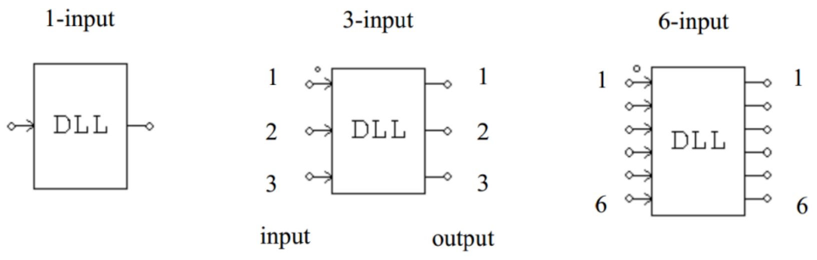


Figure 6- 5 : PSIM .DLL Blocks

To test the accuracy of the ADC algorithm programmed inside the DLL block, the output waveforms of the PSIM ADC block and the .DLL block were compared, given the same analog input. As mentioned before, the biggest advantage of the .DLL block is the ability to perform at variable sampling frequencies during the simulation. This evaluation however, was done with constant sampling frequency to compare the accuracy between the two implementations.

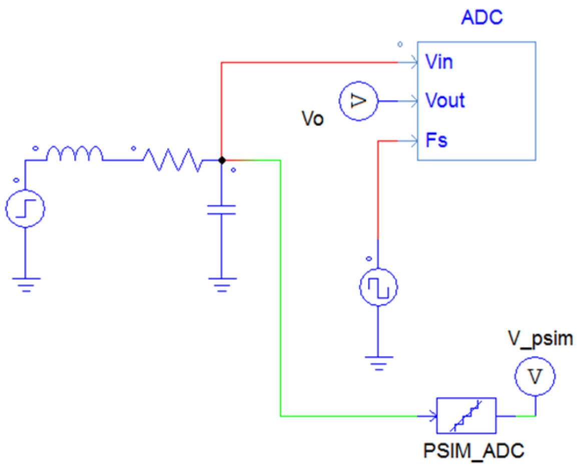


Figure 6- 6 : PSIM simulation setup

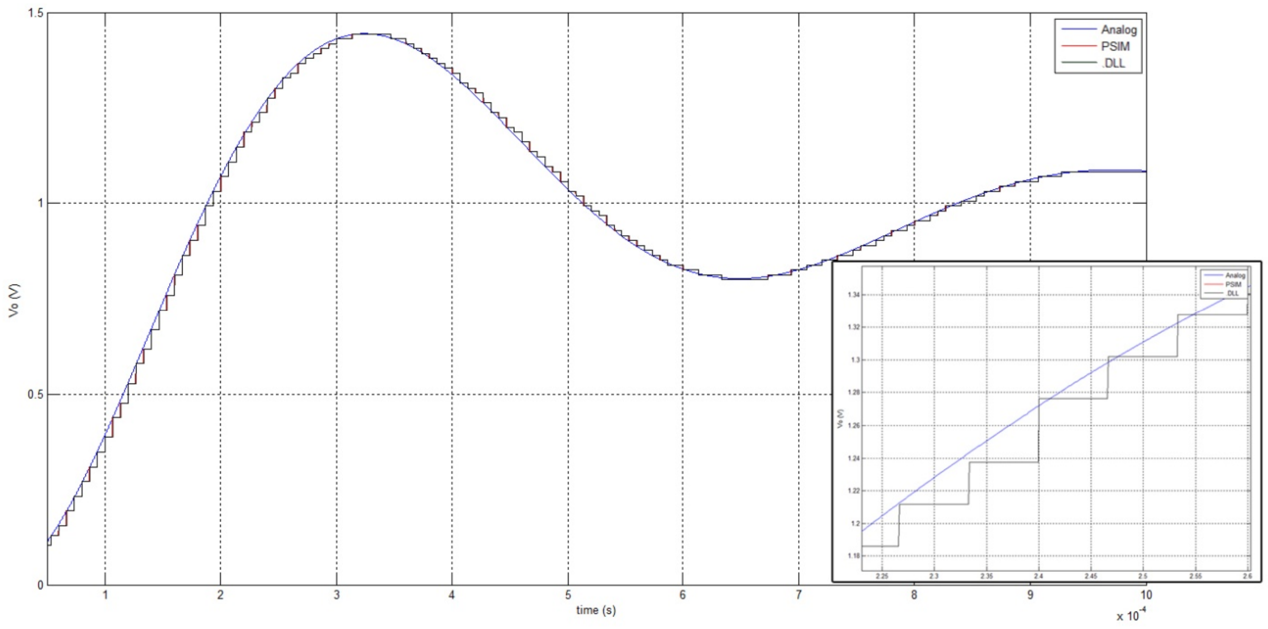


Figure 6- 7 : PSIM ADC block vs. .DLL block comparison

The results show an identical performance between the pre-programmed ADC block and the .DLL block at constant sampling frequency. Given this accuracy, it can be assumed that the .DLL block performs correctly and its performance with variable switching frequency will be acceptable. The sampling frequency can be easily adjusted by varying the frequency of the square wave source in Figure 6- 6, connected to the *Fs* pin of the DLL block. The algorithm performs measurements on the rising edge of the square wave source to compute the frequency of the incoming pulses. In addition, the programmed block has the ability to include a one sample delay if specified by the user.

Figure 6- 8 shows another comparison between the ADC block designed by PSIM and the DLL block produced in this work. At time t=0.5ms the sampling frequency was changed from 150kHz to 100kHz. The figure below shows the difference in the sampled voltage produced by the two blocks.

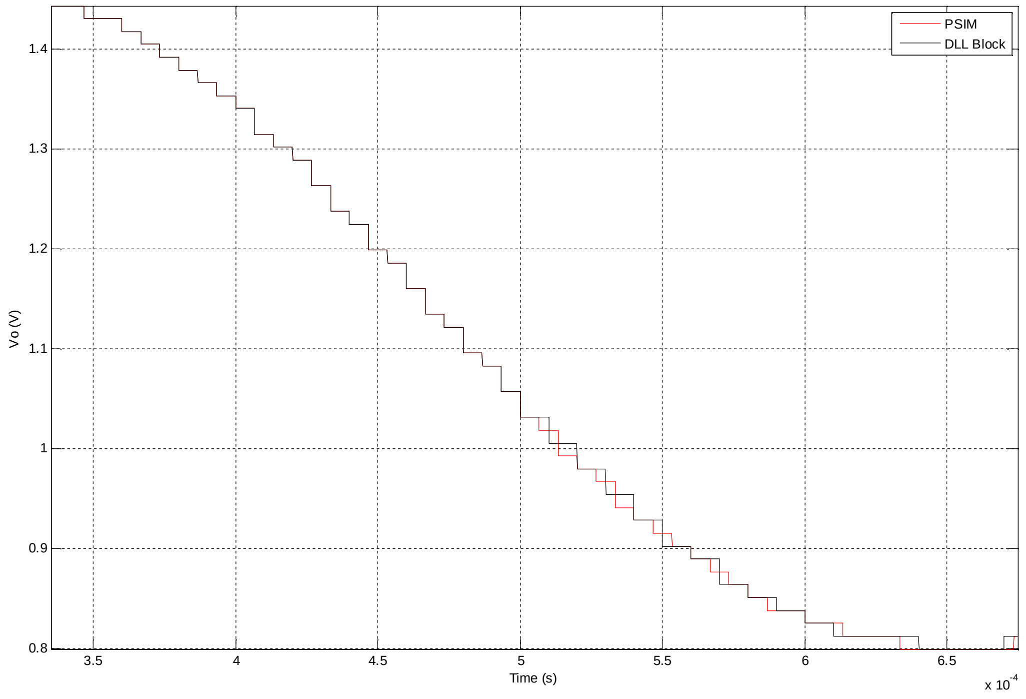
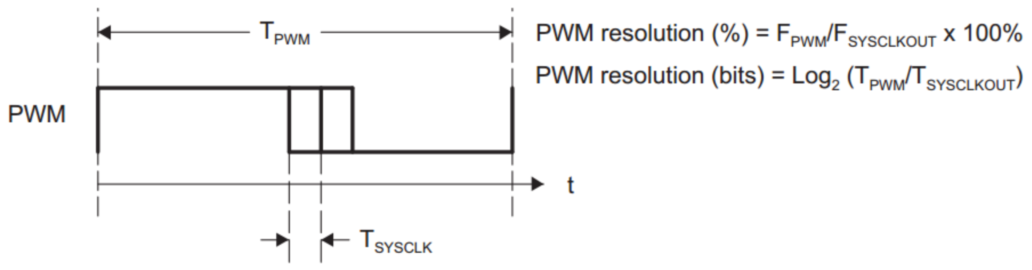


Figure 6- 8 : ADC sampling frequency comparison

6.3 High Resolution Pulse Width Modulation (HRPWM)

Once the analog signal is converted to digital, the output is run through an algorithm, converting it back to analog square wave signal that feeds into the gate of the MOSFETs thus, controlling the behavior of the power converter. Since this is digital control, the precision of the generated PWM waveform is limited by the system clock. For a DSP that runs at 60MHz, the duty cycle of the PWM waveform is constructed with samples with a period of 16.67ns. This might seem as very precise waveform but it some cases might have damaging effects in circuit performance.



T_{PWM}
PWM resolution (%) = $F_{PWM}/F_{SYSCLKOUT}$ x 100%
PWM resolution (bits) = Log_2 ($T_{PWM}/T_{SYSCLKOUT}$)
PWM
t
T_{SYSCLK}

Figure 6- 9 : Conventional PWM[7]

For example, given the parameters below

System Clock	60MHz (16.67ns)
Duty Cycle	0.405 (40.5%)
PWM Frequency	1.25MHz (800ns)
Number of steps	800ns/16.67 = 48
Number of steps for 40.5% duty	800ns*0.405 = 324ns
	324/16.67 = 19 steps
	19*16.67 = 316.73ns
	Error = 7.27ns (2.24%)

In the example above PWM alone provides accuracy with error of slightly more than 2%. In some applications this might be acceptable but it is desirable to minimize this value and consequently improve the performance of the circuit. A method called High-Resolution PWM is used to enhance the precision of the generated waveform. The Texas Instruments (TI) TMS320x2000 series Piccolo DSP is an example where HRPWM can be generated using micro-edge positioner technology (MEP). With MEP the DSP is able to position the edge of the PWM waveform with high accuracy of up to 150ps [7]. The algorithm performs its usual PWM steps of 16.67ns and when it reaches the falling or rising edge it adjusts its step to a magnitude of 150ps. This way, the duty cycle of the PWM waveform is much closer to the user specified value.

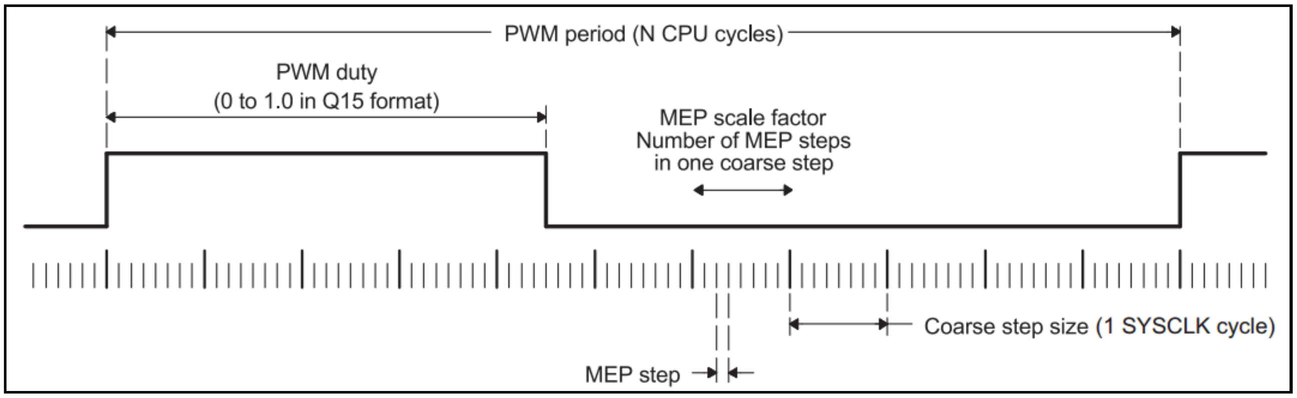


Figure 6- 10 : HRPWM using MEP [7]

For the previous example, if the DSP is equipped with HRPWM:

System Clock	60MHz (16.67ns)
Duty Cycle	0.405 (40.5%)
PWM Frequency	1.25MHz (800ns)
MEP	180ps
Number of PWM steps	800ns/16.67ns = 48 steps
Number of MEP per PWM step	16.67ns/150ps = 111 steps
Number of steps for 40.5% duty	800ns*0.405 = 324ns
	324/16.67 = 19 steps
	19*16.67 = 316.73ns
	Error = 7.27ns (2.24%)
Error with MEP	7.27ns/150ps = 48 steps
	316.73ns + 48*150ps = 323.93ns
	Error = 0.07ns (0.02%)

As expected, the MEP technique has a significant effect on the performance, minimizing the error by a magnitude of more than 100.

As in the ADC simulation, the .DLL programmable block was used to implement HRPWM in PSIM. The functionality described above was used to program the block and match the hardware performance of HRPWM. First, the waveform was generated using larger steps of 16.67ns, and once it reached the next falling or rising edge, smaller steps specified by the user (default at 180ps) were employed to generate the waveform up to the edge. This algorithm continued until the complete PWM waveform was generated.

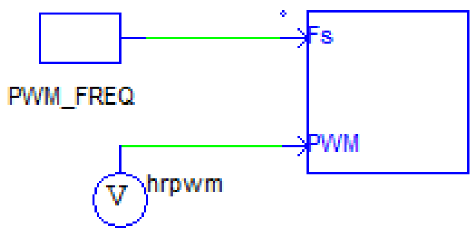


Figure 6- 11 : HRPWM DLL block

Example

System Clock	60MHz (16.67ns)
MEP	180ps
Duty	40.5%
PWM Frequency	151KHz
T_{on}	1/151kHz * 0.405 = 2.682119µs
Number of PWM steps (T_{on})	160
PWM T_{on}	160*16.67ns = 2.6667µs
Number of MEP steps to complete T_{on}	85
HRPWM T_{on}	MEP*85 + (1/System Clock)*160 = 2.6820µs

The example above shows a comparison of the PWM waveform with and without HRPWM. If the signal is generated using conventional PWM methods, the period of the positive duty cycle (T_{on}) showed a value of 2.6667µs, where the true value was at 2.682119µs. Calculations show that when HRPWM is utilized, the result was a much closer value of 2.6820µs. In PSIM simulation, the HRPWM block presented a value of 2.68209µs. It was agreed that the error of 90ps was due to rounding errors in both the calculation of the parameters and the simulation execution in PSIM. Nonetheless, the DLL block offered an acceptable performance. A copy of the C-code used to generate the HRPWM DLL block is given in the Appendix section.

Although accurate, the major drawback of this implementation is the influence of the simulation step size. For the MEP algorithm to position the PWM edges precisely it requires the simulation time step to be a few magnitudes smaller than the MEP step size. In this case a simulation step of 18ps was set to work with a MEP step size of 180ps. Since the PSIM software does not offer variable simulation steps the simulation run time is greatly extended. If variable simulation step sizes were possible however, the simulation run time can be much improved by running the model at a larger step when it is between the rising and falling edges of the PWM signal, and running at a smaller step when the simulation reaches the edges.

Give proper selection of the simulation step, the ADC and HRPWM modules can be successfully used to observe the behavior and effects of quantization and micro-edge positioning in the LLC circuit discussed in this work. The behavior of each block can be improved once PSIM allows for variable simulation time steps. At this time, the results displayed in this section provide a sufficient model of a DSP and its effects when used along with power converters.

The information shown here and the results in the previous chapters provide a good starting point in the design of digital control of LLC resonant power converters. As evident it is essential to accurately model the hardware and observe the disturbances introduces with the addition of DSPs when designing a digital control algorithm for resonant converters. The next step is to use the contributions in this thesis and attempt to design a superior digital controller that would function over a wide range of operational conditions.

CHAPTER 7 Conclusion and Future Work

7.1 Conclusions

The recent rise in popularity of resonant type converters and the switch from analog to digital control methods requires the need for more research in these fields. In the case of the LLC type resonant converter a robust adaptive control is required since the dynamics of the circuit change with switching frequency, as discussed in the introductory section of this work. To understand the circuit dynamics and design a controller a modeling approach is required that would provide the designer with information about the circuit's time and frequency response as well as the circuit's transfer function(s).

Several methods are available for modeling resonant power converters. A technique based on the Extended-Describing Function was employed in Chapter 4. The EDF approach appears to correctly represent the dynamics of LC and LCC type resonant tanks[6] but it has inconclusive results in trials where two or more inductors are present in the circuit, such as is the case of the LLC resonant converter. As is shown in chapter 4, the model fails to correctly represent the dynamics of the LLC circuit. Using approaches similar to those of Kazimierczuk[6] and Yang[5], the model displayed inconclusive results and it was clear that additional adjustment was needed to perfect the model. The problem appears to be in the mathematical representation of the circulating currents since in the presence of a single inductor current both Kazmierczuk [6] and Wang[8] show successful representation of a resonant type converters. Higher order harmonic approximation might lead to more accurate results. Due to the inconclusive results of this model a new approach based on the Least Squares Approximation method was considered in Chapter 5.

The LSM approach is a measurement based modeling capable of describing the model dynamics. These dynamics can be tested in both frequency and time domain. Since one of the main reasons for this research is to aid in the design of a better closed-loop controller for the LLC, a transfer function that closely matches the system is vital. With the transfer function readily available one can create a constantly updating adaptive controller that would account for the circuit's dynamics over a large range of switching frequencies.

The results obtained by the LSM model show that a third order transfer function in the form of

$$\frac{Y(z)}{U(z)} = \frac{b_2z^2 + b_1z + b_0}{z^3 + a_2z^2 + a_1z + a_0}$$

is successful in approximating the frequency response of the LLC circuit. The transfer function obtained from this simple approach, based on the collection of data at the input and output of the resonant power converter circuit, is in an acceptable form that can be easily incorporated into the design of a digital controller.

It is evident from all the modeling attempts in this work that the LSM model provides the best and most useful solution to the LLC resonant converter circuit. The model provides an adequate representation of the real circuit dynamics in frequency domain, where the other models fail to do so. The small-signal model based on the EDF has potential for modelling other types of resonant circuits but in the case of the LLC some adjustments to the model are needed before it can compete with the accuracy of the LSM.

7.2 Future Work

In addition to the results shown here, any future continuation of this work should focus on perfecting the state space model of the LLC converter based on the EDF. This model shows promise if some of the characteristic equations are modified to give a more accurate solution to the circuit's circulating currents.

A direct continuation of this work can focus on the design of a closed-loop adaptive controller using the LSM model. This can be easily implemented in PSIM via the DLL programmable block module. If accomplished, the block can provide an adaptive control that can continuously evolve with the circuit dynamics. This module combined with the blocks of the ADC and HRPWM can give an improved representation of the hardware based LLC resonant converter circuit.

Furthermore, it is also possible to improve the performance of the HRPWM block. Once the PSIM software is equipped with a variable simulation time step feature it will be achievable to reprogram the HRPWM to improve its accuracy and speed performance.

Lastly, research and modeling of resonant type power converters has greatly increased with the recent popularity of this type of power converter therefore, it is likely that a new modeling approach will be discovered soon. Any future work based on the information given here needs to explore and review the most updated methods of modeling before any further study is pursued.

References

[1] Huang, Hong, "Designing LLC Resonant Half-Bridge Power Converter" SEM1900 Topic 3, TI Lecture

[2] Y. G. Kang, A. K. Upadhyay, D. L. Stephens, “Analysis and design of a half-bridge parallel resonant converter operating above resonance,” *IEEE Transactions on Industry Applications* Vol. 27, March-April 1991, pp. 386 – 395.

[3] E.X. Yang, "Extended describing function method for small-signal modeling of resonant and multi-resonant converters", Ph.D. Dissertation, 1994.

[4] Chang, Chien-Hsuan et al, "Small Signal Modeling of LLC Resonant Converters Based on Extended Describing Function", *International Symposium on Computers*, 2012, pp. 365-368.

[5] Yang, E.X., Lee, F.C., Jovanovic, M.M., "Small-signal modeling of series and parallel resonant converters" , Proc. *IEEE Applied Power Electronics Conference and Exposition*, (APEC 92), Feb. 1992, pp.785-792.

[6] Kazimierczuk, Marian K., Czarkowski, Dariusz., "Resonant Power Converters" 2nd ed. New York, NY. Wiley, 2011. Print

[7] Texas Instruments Incorporated. *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*, Texas Instruments, 2011.

[8] Wang, Ping, Liu, Can, Guo, Lin, "Modelling and Simulation of Full-bridge Series Resonant Converter Based on Generalized State Space Averaging", *ICCSEE 2013*, pp. 2263-2266, 2013

[9] Smith, Roy., "Modelling Sampled Systems", ECE147b Notes, University of California ,2010. Lecture.

[10] Zhang, Weiping, "The model and control of switch converter", Beijing: Electric Power Publishing House, 2005, pp.285-300.

[11] Zhang, Weiping, Mao, Peng, Liu, Yuanchao, "Small-Signal Modeling of Series and Parallel Resonant Converters", PEDS 2007.

[12] Fairchild. *Half-Bridge LLC Resonant Converter Design Using FSFR - Series Fairchild Power Switch*, Fairchild Semiconductor Corporation, 2012.

[13] Steigerwald, Robert L., “A Comparison of Half-bridge resonant converter topologies,” *IEEE Transactions on Power Electronics*, Vol. 3, No. 2, April 1988.

[14] Lee, Bo Yang, F.C, A.J Zhang, Guisong Huang, "LLC resonant converter for front end DC/DC conversion" APEC 2002. pp.1108 – 1112.

[15] Duerbaum, Thomas, “First harmonic approximation including design constraints,” *Twentieth International Telecommunications Energy Conference (INTELEC) 1998*, pp. 321–328.

[16] De Simone, S., et al., “Design-oriented steady state analysis of LLC resonant converters based on first-harmonic approximation,” *3-27 International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM) 2006*, pp. S41-16 to S41-23.

[17] Lu, Bing, et al., "Optimal Design Method for LLC Resonant Converter", *IEEE Applied Power Electronics Conference and Exposition*, March 2006. pp. 533-538.

[18] Ivensky, G., Bronstein, S., Ben-Yaakov, S., “Approximate analysis of the resonant LCL DC DC converter,” *IEEE Electrical and Electronics Engineers in Israel*, 2004. Proceedings, pp. 44-47.

[19] Batarseh, I., Liu, R., Ortiz-Conde, A., Yacoub, A., Siri, K., “Steady state analysis and performance characteristics of the LLC-type parallel resonant converter,” PESC’94, pp. 597 - 606

[20] Aboushady, A. A., Ahmed, K. H., Finney, S.J., Williams, B. W.,"Control Design of Phase-Controlled Series-Parellel Resonant Converters Using State Feedback", *IEEE Transactions on Power Electronics*, Vol. 28, No. 8, August 2013, pp. 3896-3911.

[21] Lin, B.R., Wu, S. F., “Implementation of a series resonant converter with series-parallel transformers,” *IET Power Electron.*, vol. 4, no. 8, pp. 919–926, 2011.

[22] Chuang, Y.-C., Ke, Y.-L., Chuang, H-S., Chen, Y.-M., “Analysis and implementation of half-bridge series-parallel resonant converter for battery chargers,” *IEEE Trans. Ind. Appl.*, vol. 47, no. 1, pp. 258–270, Feb. 2011.

[23] Wong, S. C., Brown, A. D., “Analysis, modeling and simulation of series-parallel resonant converter circuits,” *IEEE Transactions on Power Electron*ics, vol. 10, no. 5, pp. 605–614, Sep. 1995.

[24] Sano, K., Fujita, H., “Performance of high-efficiency switched capacitor- based resonant converter with phase-shift control,” *IEEE Transactions on Power Electronics*, vol. 26, no. 2, pp. 344–354, Feb. 2011.

[25] Bhat, A. K. S., “Fixed-frequency PWM series-parallel resonant converter,” *IEEE Trans. Ind. Appl.*, vol. 28, no. 5, pp. 1002–1009, Sep. 1992.

[26] Czarkowski, D., Kazimierczuk, M. K., “Phase-controlled series-parallel resonant converter,” *IEEE Transactions on Power Electronics*, vol. 8, no. 3, pp. 309– 319, Jul. 1993.

[27] Agarwal, V., Bhat, A. K. S., “Small signal analysis of the LCC-type parallel resonant converter using discrete time domain modeling,” *IEEE Trans. Ind. Electron.*, vol. 42, no. 6, pp. 604–614, Dec. 1995.

[28] Castilla, M., Garcia de Vicuna, L., Guerrero, J. M., Matas, J., Miret, J., “Sliding-mode control of quantum series-parallel resonant converters via input-output linearization,” *IEEE Trans. Ind. Electron.*, vol. 52, no. 2, pp. 566–575, Apr. 2005.

[29] Liu, T., Zhou, Z., Xiong, A., Zeng, J., Ying, J., “A novel Precise Design Method for LLC Series Resonant Converter”, *Telecommunications Energy Conference*, 2006. INTELEC '06. 28th Annual International Sept. 2006, pp. 1-6.

[30] Yang, B., “Topology Investigation for Front End DC/DC Power Conversion for Distributed Power System”, PhD dissertation, Virginia Polytechnic Institute and State University, 2003

[31] Adragna, C., De Simone, S., Spini, C., “A design methodology for LLC resonant converters based on inspection of resonant tank currents”, *Applied Power Electronics Conference*, APEC 2008 – Twenty-third Annual IEEE Feb. 2008 Pages: 1361-1367

[32] Duerbaum, T., "First harmonica approximation including design constraints", *Telecommunications Energy Conference*, 1998. INTELEC. pp. 321-328.

[33] Fang, Y., Xu, D., Zhang, Y., Gao. F., Zhu, L., "Design of High Power Density LLC Resonant Converter with Extra Wide Input Range", *Applied Power Electronics Conference*, APEC 2007 - Twenty Second Annual IEEE, Feb. 2007, pp.976 - 981

[34] Jha, V., Rai, P. "State Space Averaged Modeling of Basic Converter Topologies", *VSRD International Journal of Electrical, Electronics & Communication Engineering*, vol. 2, no. 8, pp. 566-575, 2012.

[35] Powersim 9.3, Powersim Inc., Rockville, Maryland, United States.

APPENDIX

A.1 State Space Example

$\dot{x_1} = 2x_1 + 6x_3 + 3u$

$\dot{x_2} = 3x_1 + x_2$

$\dot{x_3} = 4x_2 - 3x_3$

$y = x_1$

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = \begin{bmatrix} 2 & 0 & 6 \\ 3 & 1 & 0 \\ 0 & 4 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

A.2 Duty Cycle Model: MATLAB

```
r = 0.1;
n = 18/5;
Ro = 50^2/2400;
RL = 8*(n^2)*Ro/(pi^2);
Lr = 9.6e-6;
Cr = 4*33e-9;
Lm = 25e-6;
d = 0.001;
Vin = 400;
D = 0; %D matrix

%Steady State Values

a = [-(r+RL)/Lr, RL/Lr, -1/Lr;
    RL/Lm, -RL/Lm, 0;
    1/Cr, 0, 0];

b = [-d*Vin/Lr;0;0];

k = a\b;
IL=k(1);
ILm=k(2);
VC=k(3);
```

```
% DUTY MODEL


A = [-(r+RL)/Lr, RL/Lr, -1/Lr;
    RL/Lm, -RL/Lm, 0;
    1/Cr, 0, 0];

 C = [RL -RL 0];
 B = [Vin/Lr;0;0];

%-------------------------------------------------------------------------

t1= 0:1e-7:0.006;

%change in input voltage, u-vector

s = size(t1);
s = s(2)/2;

%change in duty cycle, 0.5 for half the simulation, 0.8 for the second half
ds = [0.5*ones(1,s), 0.8*ones(1,s+1)];
[y,t,x] = lsim(sys,ds,t1);

plot(t,x(:,3));
ylabel('Vc');
xlabel('t');

h = bodeoptions;
h.frequnits = 'Hz';
h.xlim = [100 1e5];
figure
bode(sys,h)
```

A.3 Frequency Control Model Based on the Extended Describing Function:MATLAB

```
r = 0.1;
Lr = 9.5e-6;
Cr = 4*33e-9;
Lm = 25e-6;
Vg = 200;
W = 2*pi*0.0001;
Co = 3e-6;
RL = 10.92;
d = 0.001;

ILs = 0.000001;
ILc = 0;
ILms = 0;
ILmc = 0;
Vcs = 0;
```

```
Vcc = 0;
Voc = 0;
Ip = sqrt((ILs - ILms)^2 + (ILc - ILmc)^2);



A = [(-r - 4*Voc/(pi*Ip) + 4*ILs*(ILs-ILms)*Voc/(pi*Ip^3) - 4*ILms*(ILs-
ILms)*Voc/(pi*Ip^3))/Lr, W + 4*ILs*(ILc-ILmc)*Voc/(Lr*pi*Ip^3) - 4*ILms*(ILc-
ILmc)*Voc/(Lr*pi*Ip^3), 4*Voc/(pi*Ip*Lr) + 4*Voc*ILs*(ILms-ILs)/(Lr*pi*Ip^3)
- 4*Voc*ILms*(ILms-ILs)/(Lr*pi*Ip^3), 4*Voc*ILs*(ILmc-ILc)/(Lr*pi*Ip^3) -
4*Voc*ILms*(ILmc-ILc)/(Lr*pi*Ip^3), -1/Lr, 0, 4*(ILs-ILms)/(Ip*pi*Lr);
 -W + 4*ILc*(ILs-ILms)*Voc/(Lr*pi*Ip^3) - 4*ILmc*(ILs-ILms)*Voc/(Lr*pi*Ip^3),
(-r - 4*Voc/(pi*Ip) + 4*ILc*(ILc-ILmc)*Voc/(pi*Ip^3) - 4*ILmc*(ILc-
ILmc)*Voc/(pi*Ip^3))/Lr, 4*Voc*ILc*(ILms-ILs)/(Lr*pi*Ip^3) -
4*Voc*ILmc*(ILms-ILs)/(Lr*pi*Ip^3), 4*Voc/(pi*Ip*Lr) + 4*Voc*ILc*(ILmc-
ILc)/(Lr*pi*Ip^3) - 4*Voc*ILmc*(ILmc-ILc)/(Lr*pi*Ip^3), 0, -1/Lr, 4*(ILc-
ILmc)/(Ip*pi*Lr);
 4*Voc/(Ip*pi*Lm) - 4*ILs*(ILs - ILms)*Voc/(pi*Lm*Ip^3) + 4*ILms*(ILs -
ILms)*Voc/(pi*Lm*Ip^3), -4*ILs*(ILc - ILmc)*Voc/(pi*Lm*Ip^3) + 4*ILms*(ILc -
ILmc)*Voc/(pi*Lm*Ip^3), -4*ILs*(ILms - ILs)*Voc/(pi*Lm*Ip^3) + 4*ILms*(ILms -
ILs)*Voc/(pi*Lm*Ip^3) - 4*Voc/(pi*Ip*Lm), W - 4*ILs*(ILmc -
ILc)*Voc/(pi*Lm*Ip^3) + 4*ILms*(ILmc - ILc)*Voc/(pi*Lm*Ip^3), 0, 0, 4*(ILs-
ILms)/(pi*Ip);
 -4*ILc*(ILs - ILms)*Voc/(pi*Lm*Ip^3) + 4*ILmc*(ILs - ILms)*Voc/(pi*Lm*Ip^3),
4*Voc/(Ip*pi*Lm) - 4*ILc*(ILc - ILmc)*Voc/(pi*Lm*Ip^3) + 4*ILmc*(ILc -
ILmc)*Voc/(pi*Lm*Ip^3), -W - 4*ILc*(ILms - ILs)*Voc/(pi*Lm*Ip^3) +
4*ILmc*(ILms - ILs)*Voc/(pi*Lm*Ip^3), -4*ILc*(ILmc - ILc)*Voc/(pi*Lm*Ip^3) +
4*ILmc*(ILmc - ILc)*Voc/(pi*Lm*Ip^3) - 4*Voc/(pi*Ip*Lm), 0, 0, 4*(ILc-
ILmc)/(pi*Ip);
 1/Cr, 0, 0, 0, 0, W, 0;
 0, 1/Cr, 0, 0, -W, 0, 0;
 2*(ILs-ILms)/(Co*pi*Ip), 2*(ILc-ILmc)/(Co*pi*Ip), 2*(ILms-ILs)/(Co*pi*Ip),
2*(ILmc-ILc)/(Co*pi*Ip),0,0,-1/(Co*RL)];



 B = [2*cos(pi*d/2)*Vg ILc;
     0 -ILs;
     0 ILmc;
     0 -ILms;
     0 Vcc;
     0 -Vcs;
     0 0];

 C = [0 0 0 0 0 0 1];

 D = 0;

sys = ss(A,B,C,D);
t1= 0:0.0000001:2;
s = size(t1);
s = s(2)/2;
ds = [0.5*ones(1,s),0.8*ones(1,s+1)];
z = [2*pi*120e3*ones(1,s),2*pi*120e3*ones(1,s+1)];
[y,t,x] = lsim(sys,[ds;z],t1);

subplot(4,1,1)
```

```
plot(t,sqrt(x(:,1).^2 + x(:,2).^2));
title('EDF Model')
ylabel('IL (A)');
xlabel('t (s)');
grid on
subplot(4,1,2)
plot(t,sqrt(x(:,3).^2 + x(:,4).^2));
ylabel('ILm (A)');
xlabel('t (s)');
grid on
subplot(4,1,3)
plot(t,sqrt(x(:,5).^2 + x(:,6).^6));
ylabel('Vc (V)');
xlabel('t (s)');
grid on
subplot(4,1,4)
plot(t,y);
ylabel('Vo (V)');
xlabel('t (s)');
grid on
```

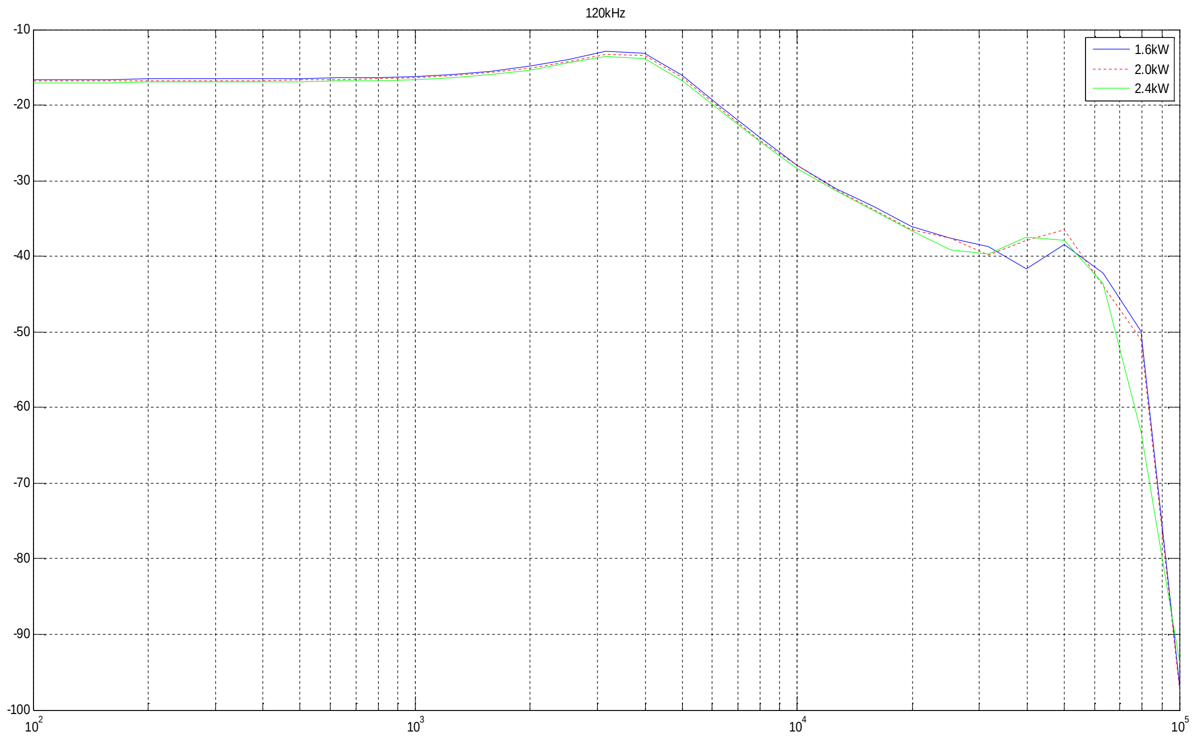
A.4 Least Squares Method : MATLAB

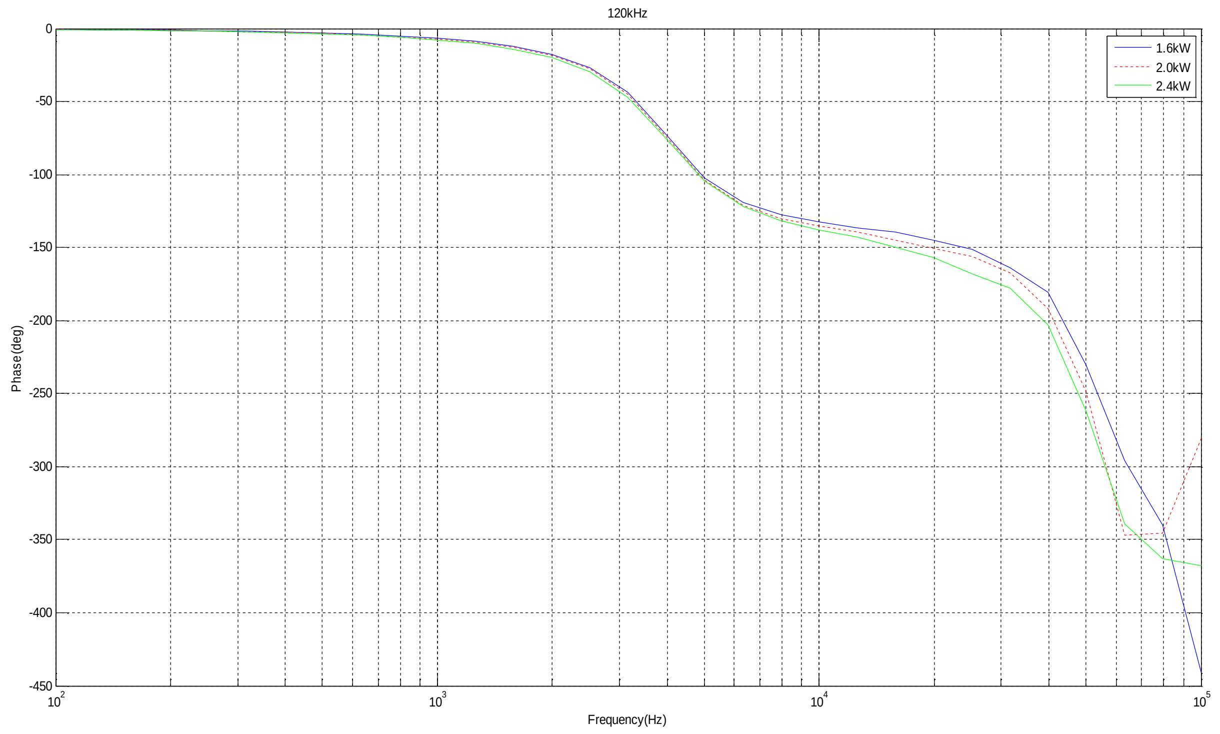
```
% clear all
% close all
clc
V = v2;
f = f2;
format long

% load('StepChange_120_125kHz.mat');

%-------------------------------------------------------------------------
% LSM without Low Pass Filter
%-------------------------------------------------------------------------

N = length(f);

% USING 26 Samples from above vectors u & y
x1 = f(4:N-1); % u[n-1]
x2 = f(3:N-2); % u[n-2]
x3 = f(2:N-3); % u[n-3]
x4 = V(4:N-1); % y[n-1]
x5 = V(3:N-2); % y[n-2]
x6 = V(2:N-3); % y[n-3]

y = V(5:N);

x = [x1 x2 x3 x4 x5 x6];

A = (inv(x'*x)*(x'*y));
```

```
b2 = A(1,1);
b1 = A(2,1);
b0 = A(3,1);
a2 = A(4,1); %a1
a1 = A(5,1); %a2
a0 = A(6,1); %a3

Y = b2*x1 + b1*x2 + b0*x3 + a2*x4 + a1*x5 + a0*x6;

Fs = 100e3;
Ts = 1/Fs;
t = (Ts*(0:(N-5)))';

%figure
%plot(t,V(1:N-4),'b',t,Y,'r')

% Transfer Function

Gpz = tf([b2 b1 b0],[1 -a2 -a1 -a0],Ts);
zpk(Gpz)

h = bodeoptions;
h.frequnits = 'Hz';
h.xlim = [100 1e5];

figure
[m,p,w1] = bode(Gpz,h);
grid on
% hold on


%--------------------------------------------------------------------------
%       Low Pass Filter and LSM
%--------------------------------------------------------------------------

%% Low Pass Filter data
[b,a] = butter(2,0.2,'low');
%
v = filter(b,a,V);

% v = v(2000:length(v));
% fr = f(2000:length(f));

fr = f;
L = length(fr);

% USING 26 Samples from above vectors u & y
x1 = fr(4:L-1); % u[n-1]
x2 = fr(3:L-2); % u[n-2]
x3 = fr(2:L-3); % u[n-3]
x4 = v(4:L-1); % y[n-1]
x5 = v(3:L-2); % y[n-2]
x6 = v(2:L-3); % y[n-3]
```

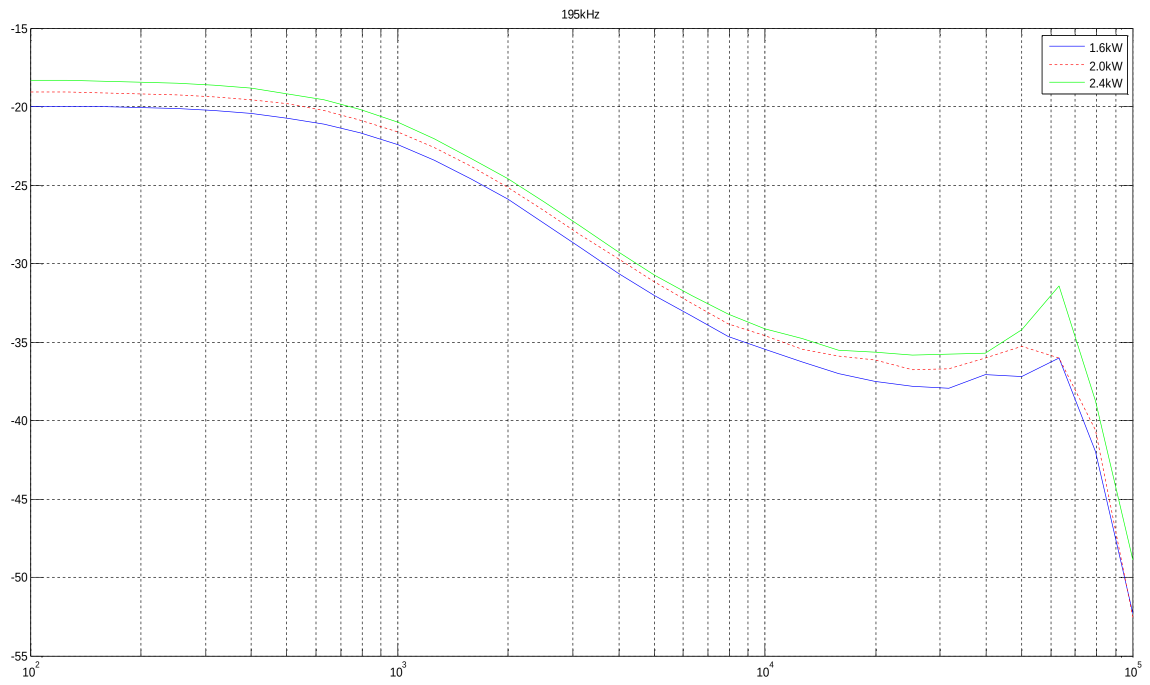
```
y = v(5:L);

x = [x1 x2 x3 x4 x5 x6];

A = (inv(x'*x)*(x'*y));

b2 = A(1,1);
b1 = A(2,1);
b0 = A(3,1);
a2 = A(4,1);
a1 = A(5,1);
a0 = A(6,1);

Y1 = b2*x1 + b1*x2 + b0*x3 + a2*x4 + a1*x5 + a0*x6;

Fs = 100e3;
Ts = 1/Fs;
t1 = (Ts*(0:(L-5)))';

% figure
% plot(t1,v(1:L-4),'b',t1,Y1,'r')
% grid on

%% Transfer Function

Gpz1 = tf([b2 b1 b0],[1 -a2 -a1 -a0],Ts);
zpk(Gpz1)

h = bodeoptions;
h.frequnits = 'Hz';
h.xlim = [100 1e5];

figure
[m2,p2,w2] = bode(Gpz1,h);
grid on
% hold on
```

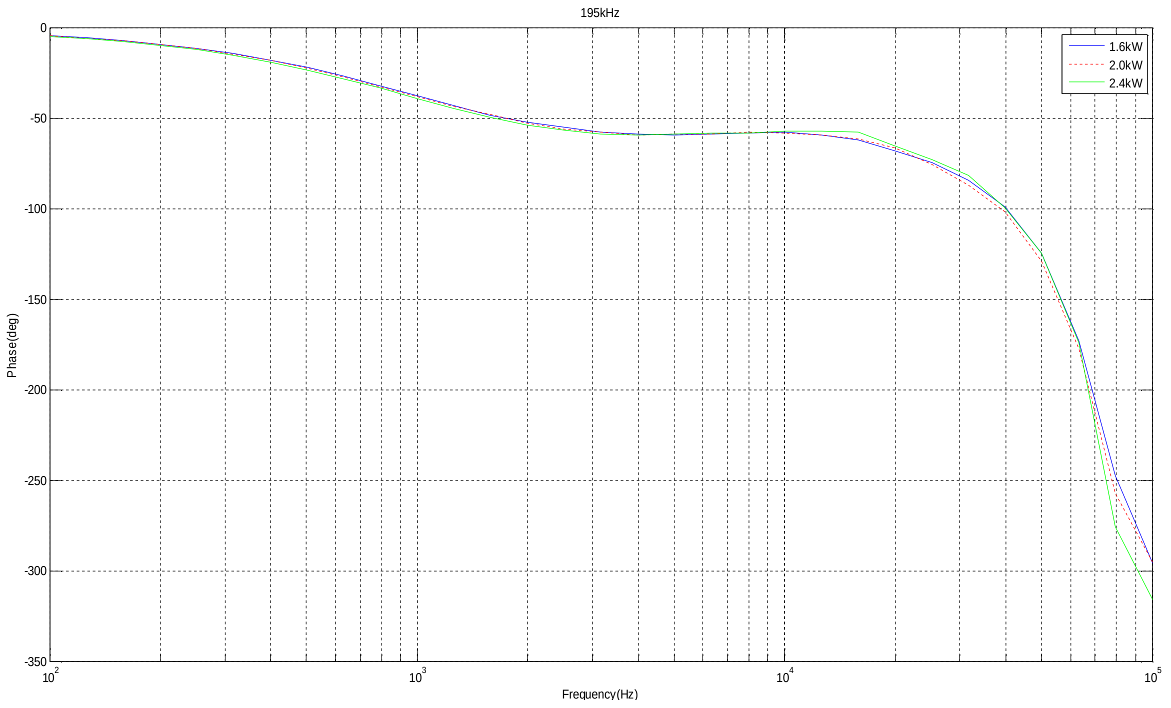
A.5 Additional PSIM and LSM Results



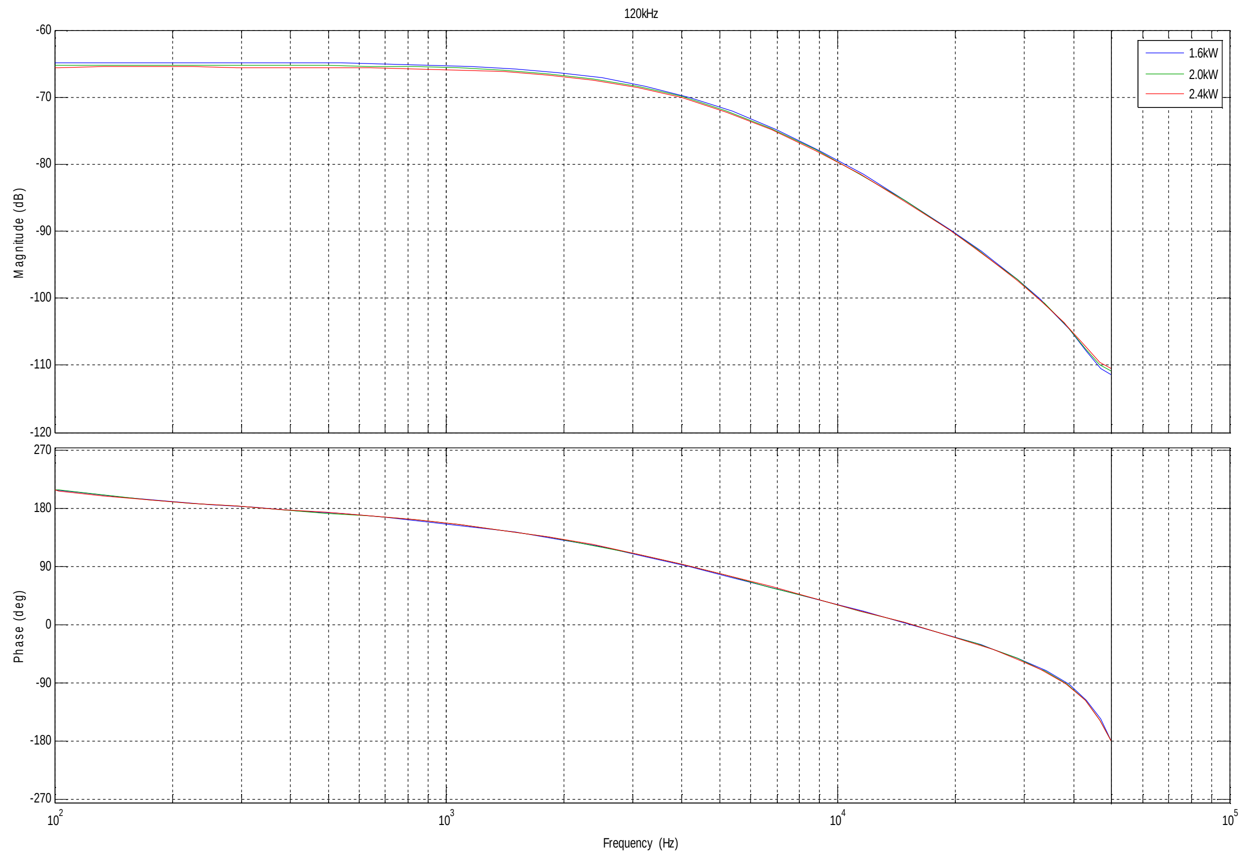


A- 1 : PSIM Bode plot - 120kHz - 125kHz

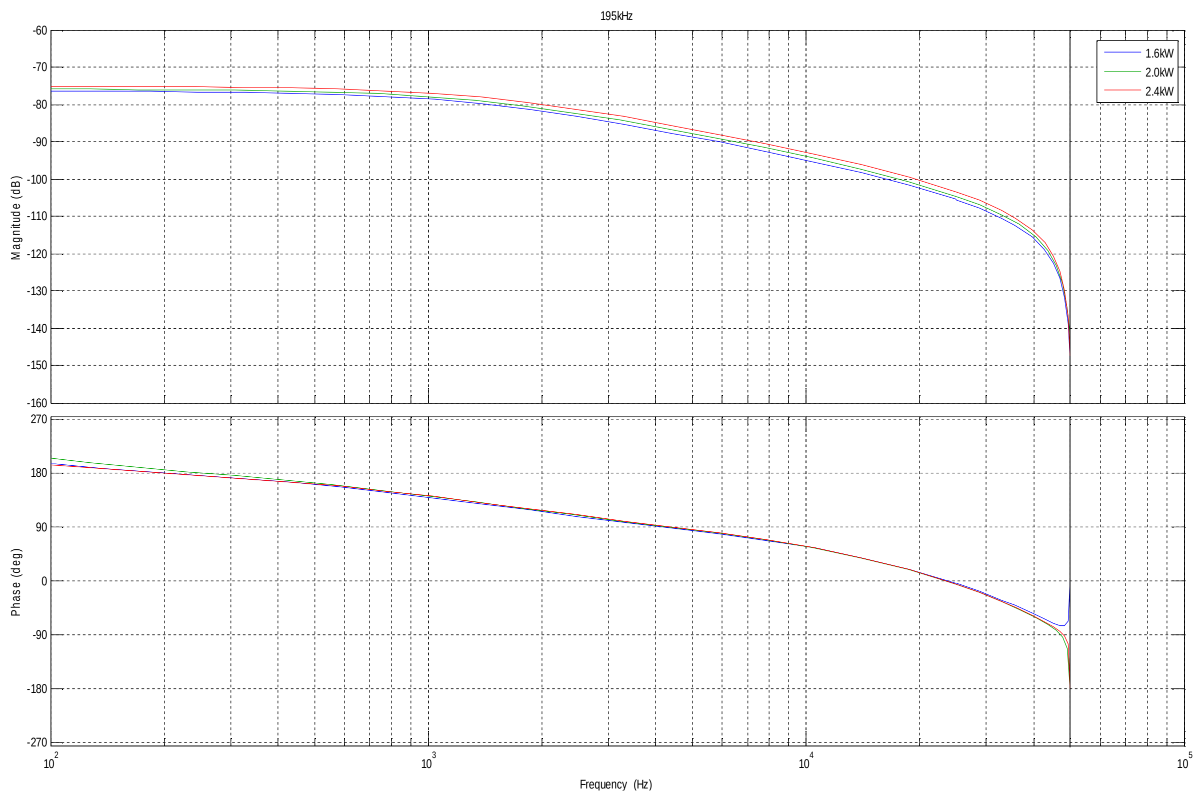




A- 2 : PSIM Bode Plot - 195kHz - 200kHz



A- 3 : LSM Model - 120kHz-125kHz



A- 4 : LSM Model - 195kHz-200kHz

A.6 Extended Describing Function : Matrix Equations

$$A_{11} = \frac{1}{L_r}\left(-r - 4\frac{V_o}{\pi I_p} + 4I_{Lrs}(I_{Lrs} - I_{Lms})\frac{V_o}{\pi I_p^3} - 4I_{Lms}(I_{Lrs} - I_{Lms})\frac{V_o}{\pi I_p^3}\right)$$

$$A_{12} = W + 4I_{Lrs}(I_{Lrc} - I_{Lmc}) * \frac{V_o}{\pi L_r I_p^3} - 4I_{Lms}(I_{Lrc} - I_{Lmc}) * \frac{V_o}{\pi L_r I_p^3}$$

$$A_{13} = 4\frac{V_o}{\pi I_p L_r} + 4V_o I_{Lrs}\frac{I_{Lms} - I_{Lrs}}{\pi L_r I_p^3} - 4V_o I_{Lms}\frac{I_{Lms} - I_{Lrs}}{\pi L_r I_p^3}$$

$$A_{14} = 4V_o I_{Lrs}\frac{I_{Lmc} - I_{Lrc}}{\pi L_r I_p^3} - 4V_o I_{Lms}\frac{I_{Lmc} - I_{Lrc}}{\pi L_r I_p^3}$$

$$A_{15} = -\frac{1}{L_r}$$

$$A_{16} = 0$$

$$A_{17} = 4\frac{I_{Lrs} - I_{Lms}}{\pi I_p L_r}$$

$$A_{21} = -W + 4I_{Lrc}(I_{Lrs} - I_{Lms})\frac{Voc}{\pi L_r I_p^3} - 4I_{Lmc}(I_{Lrs} - I_{Lms})\frac{V_o}{\pi L_r I_p^3}$$

$$A_{22} = \frac{1}{L_r}\left(-r - 4\frac{V_o}{\pi I_p} + 4I_{Lrc}(I_{Lrc} - I_{Lmc})\frac{V_o}{\pi I_p^3} - \frac{4I_{Lmc}(I_{Lrc} - I_{Lmc})V_o}{\pi I_p^3}\right)$$

$$A_{23} = \frac{4V_o I_{Lrc}(I_{Lms} - I_{Lrs})}{\pi L_r I_p^3} - \frac{4V_o I_{Lmc}(I_{Lms} - I_{Lrs})}{\pi L_r I_p^3}$$

$$A_{24} = 4\frac{V_o}{\pi I_p L_r} + 4V_o I_{Lrc}\frac{I_{Lmc} - I_{Lrc}}{\pi L_r I_p^3} - \frac{4V_o I_{Lmc}(I_{Lmc} - I_{Lrc})}{\pi L_r I_p^3}$$

$$A_{25} = 0$$

$$A_{26} = -\frac{1}{L_r}$$

$$A_{27} = 4\frac{I_{Lrc} - I_{Lmc}}{\pi I_p L_r}$$

$$A_{31} = 4\frac{V_o}{\pi I_p L_m} - \frac{4I_{Lrs}(I_{Lrs} - I_{Lms})V_o}{\pi L_m I_p^3} + \frac{4I_{Lms}(I_{Lrs} - I_{Lms})V_o}{\pi L_m I_p^3}$$

$$A_{32} = -\frac{4I_{Lrs}(I_{Lrc} - I_{Lmc})V_o}{\pi L_m I_p^3} + \frac{4I_{Lms}(I_{Lrc} - I_{Lmc})V_o}{\pi L_m I_p^3}$$

$$A_{33} = -\frac{4I_{Lrs}(I_{Lms} - I_{Lrs})V_o}{\pi L_m I_p^3} + \frac{4I_{Lms}(I_{Lms} - I_{Lrs})V_o}{\pi L_m I_p^3} - \frac{4V_o}{\pi I_p L_m}$$

$$A_{34} = W - \frac{4I_{Lrs}(I_{Lmc} - I_{Lrc})V_o}{\pi L_m I_p^3} + \frac{4I_{Lms}(I_{Lmc} - I_{Lrc})V_o}{\pi L_m I_p^3}$$

$$A_{35} = 0$$

$$A_{36} = 0$$

$$A_{37} = \frac{4(I_{Lrs} - I_{Lms})}{\pi I_p}$$

$$A_{41} = -\frac{4I_{Lrc}(I_{Lrs} - I_{Lms})V_o}{\pi L_m I_p^3} + \frac{4I_{Lmc}(I_{Lrs} - I_{Lms})V_o}{\pi L_m I_p^3}$$

$$A_{42} = \frac{4V_o}{\pi I_p L_m} - \frac{4I_{Lrc}(I_{Lrc} - I_{Lmc})V_o}{\pi L_m I_p^3} + \frac{4I_{Lmc}(I_{Lrc} - I_{Lmc})V_o}{\pi L_m I_p^3}$$

$$-W - \frac{4I_{Lrc}(I_{Lms} - I_{Lrs})V_o}{\pi L_m I_p^3} + \frac{4I_{Lmc}(I_{Lms} - I_{Lrs})V_o}{\pi L_m I_p^3}$$

$$A_{43} = -\frac{4I_{Lrc}(I_{Lmc} - I_{Lrc})V_o}{\pi L_m I_p^3} + \frac{4I_{Lmc}(I_{Lmc} - I_{Lrc})V_o}{\pi L_m I_p^3} - \frac{4V_o}{\pi I_p L_m}$$

$$A_{45} = 0$$

$A_{46} = 0$

$$A_{47} = \frac{4(I_{Lrc} - I_{Lmc})}{\pi I_p}$$

$$A_{51} = -\frac{1}{C_r}$$

$A_{52} = 0$

$A_{53} = 0$

$A_{54} = 0$

$A_{55} = 0$

$A_{56} = W$

$A_{57} = 0$

$A_{61} = 0$

$$A_{62} = \frac{1}{C_r}$$

$A_{63} = 0$

$A_{64} = 0$

$A_{65} = -W$

$A_{66} = 0$

$A_{67} = 0$

$$A_{71} = \frac{2(I_{Lrs} - I_{Lms})}{\pi C_o I_p}$$

$$A_{72} = \frac{2(I_{Lrc} - I_{Lmc})}{\pi C_o I_p}$$

$$A_{73} = \frac{2(I_{Lms} - I_{Lrs})}{\pi C_o I_p}$$

$$A_{74} = \frac{2(I_{Lmc} - I_{Lrc})}{\pi C_o I_p}$$

$A_{75} = 0$

$A_{76} = 0$

$$A_{77} = -\frac{1}{C_o R_L}$$

A.7 State Space Averaging Model

Average Model

The first modeling approach to be evaluated was based on the State-Space Averaging technique. This method provides accurate results when dealing with DC-DC power converters with constant switching frequency. Typically it is applied to duty cycle controlled converters such as boost, buck or buck-boost. In this case duty cycle control refers to varying the *ON* and *OFF* time of a transistor switch embedded in the circuit. The performance of these circuits is evaluated by separating the circuit into two or more equivalent circuits, based on the characteristics of the duty cycle, and then producing an average equation.

Unfortunately, the average model can be used to partially evaluate the LLC power converter since the performance of this type of converter is dependent on both duty cycle and frequency of the switched waveform but this approach is only applicable to duty cycle variation.

The objective of the model was to correctly represent the circuit dynamics during duty cycle change and provide a transfer function that approximates the frequency response of the system.

A variation or disturbance in a parameter can be included into the state space model using the small signal modeling. This method alters the model by representing the state variables stored in the x vector matrix as well as the input variables stored in the u vector matrix as a sum of the steady state value of the variable and its disturbance parameter. The representation is as such:

$$x = X + \tilde{x}$$

Where X is the steady state value and $\tilde{x}$ represents the disturbance. Using this approach, the duty cycle was set as the input to the circuit.

Average modeling requires two or sometimes three (if the circuit operates in discontinuous current mode) equivalent circuits of the power converter. One circuit presents the design when the switch is in an *ON* state and the second shows the switch *OFF* configuration. The two equivalent circuits of the LLC are shown in Figure A7- 1.

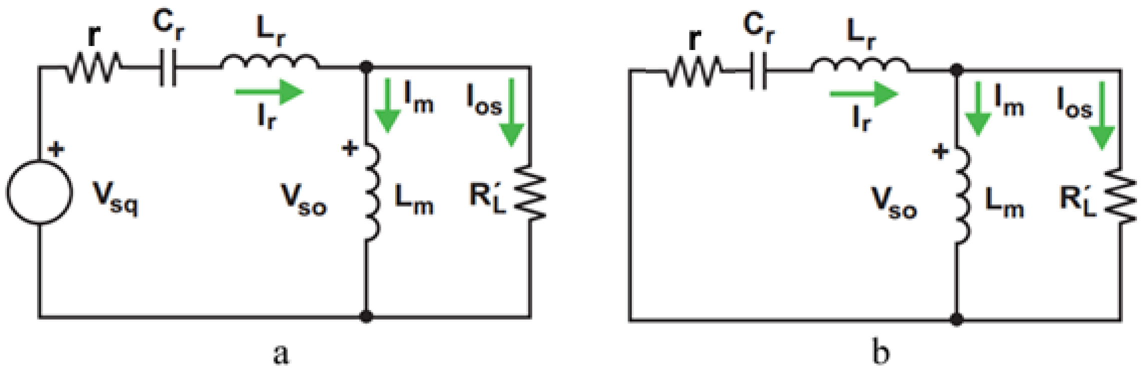


Figure A7- 1 : LLC Equivalent Circuits (a) ON and (b) OFF

Each configuration above is described by its unique set of equations using mesh analysis. The two configurations can be combined into one using the linear weighted averaging technique, producing one equation for the complete switching cycle of the converter. The state variables of the circuit in this case are based on the three active components, L_r, C_r and L_m. Hence the variables are labelled as i_r, the current through the resonating inductor, i_m, the current through the magnetizing inductor and V_{Cr}, the voltage of the resonating capacitor.

Configuration: a)

$$V_{in} = i_r r + L_r \frac{di_r}{dt} + L_m \frac{di_m}{dt} + V_{Cr} \tag{4.1a}$$

$$Lm \frac{di_m}{dt} = (i_r - i_m) R_L \tag{4.1b}$$

$$C \frac{dV_{Cr}}{dt} = i_r \tag{4.1c}$$

$$V_{out} = (i_r - i_m) \tag{4.1d}$$

Hence, into state space configuration the model becomes

a. Switch ON

The state variables are selected as: $x = \begin{bmatrix} i_r \\ i_m \\ V_{Cr} \end{bmatrix}$

$$\begin{bmatrix} \frac{di_r}{dt} \\ \frac{di_m}{dt} \\ \frac{dV_C}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{(r-R_L)}{L_r} & \frac{R_L}{L_r} & -\frac{1}{L_r} \\ \frac{R_L}{L_m} & -\frac{R_L}{L_m} & 0 \\ \frac{1}{Cr} & 0 & 0 \end{bmatrix} \begin{bmatrix} i_r \\ i_m \\ V_{Cr} \end{bmatrix} + \begin{bmatrix} \frac{1}{L_r} \\ 0 \\ 0 \end{bmatrix} V_{in}$$

$$y = \begin{bmatrix} R_L & -R_L & 0 \end{bmatrix} \begin{bmatrix} i_r \\ i_m \\ V_{Cr} \end{bmatrix}$$

b. Switch OFF

$$\begin{bmatrix} \frac{di_r}{dt} \\ \frac{di_m}{dt} \\ \frac{dV_C}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{(r+R_L)}{L_r} & \frac{R_L}{L_r} & -\frac{1}{L_r} \\ \frac{R_L}{L_m} & -\frac{R_L}{L_m} & 0 \\ \frac{1}{Cr} & 0 & 0 \end{bmatrix} \begin{bmatrix} i_r \\ i_m \\ V_{Cr} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} V_{in}$$

$$y = \begin{bmatrix} R_L & -R_L & 0 \end{bmatrix} \begin{bmatrix} i_r \\ i_m \\ V_{Cr} \end{bmatrix}$$

The new matrices of the Average model are expressed as

$$A = dA_{on} + (1-d)A_{off} \tag{4.1e}$$

$$B = dB_{on} + (1-d)B_{off} \tag{4.1f}$$

$$C = dC_{on} + (1-d)C_{off} \tag{4.1g}$$

where $\boldsymbol{d}$ represents the duty cycle of the half bridge inverter.

Next, small signal approximation is applied

$x = x + \check{x}$, $u = u + \check{u}$, $d = d + \check{d}$ where $\check{x}$ denotes a small signal perturbation of the variable. By substituting these variable into the circuit equations the small signal matrices become

$$\begin{bmatrix} \frac{d\hat{\imath}_r}{dt} \\ \frac{d\hat{\imath}_m}{dt} \\ \frac{d\hat{V}_C}{dt} \end{bmatrix} = \begin{bmatrix} -\frac{(r+R_L)}{L_r} & \frac{R_L}{L_r} & -\frac{1}{L_r} \\ \frac{R_L}{L_m} & -\frac{R_L}{L_m} & 0 \\ \frac{1}{Cr} & 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\imath}_r \\ \hat{\imath}_m \\ \hat{V}_{Cr} \end{bmatrix} + \begin{bmatrix} \frac{d}{L_r} & \frac{V_{in}}{L_r} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{V}_{in} \\ \hat{d} \end{bmatrix}$$

$$\hat{y} = \begin{bmatrix} R_L & -R_L & 0 \end{bmatrix} \begin{bmatrix} \hat{\imath}_r \\ \hat{\imath}_m \\ \hat{V}_{Cr} \end{bmatrix}$$

Next, by substituting circuit parameters into the small signal matrices, giving appropriate initial conditions and varying the input parameters the model was able to display behavior similar to that of the LLC circuit simulated in PSIM.

Simulation Results

To evaluate the performance of the LLC model in PSIM and the Average model in MATLAB, the duty cycle of the circuit was varied from 50% to 80% with constant switching frequency. For the purpose of comparison, the model used the same circuit parameters as section 2.1 Introduction.

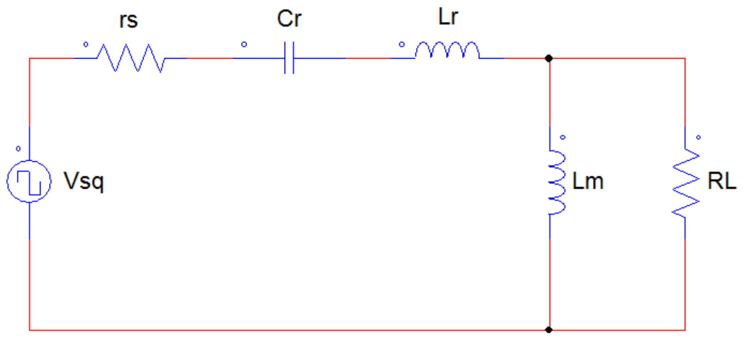


Figure A7- 2 : LLC Equivalent Circuit

Simulation parameters

V_{in}	400 V
n	3.6
L_R	9.6μH
C_R	132nF
L_M	25μH
R_L	1.04Ω
f_{sw}	142kHz
d (duty)	50-80%

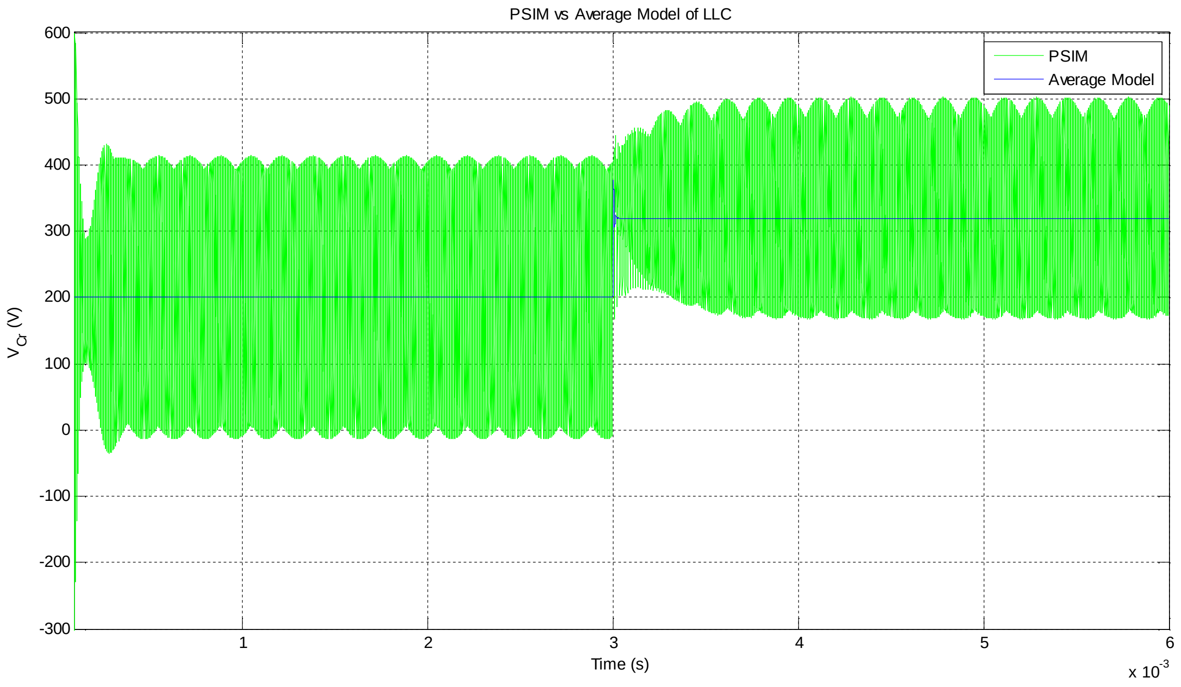


Figure A7- 3 : PSIM vs. Average Model: capacitor voltage comparison

Figure A7- 3 shows the waveform of the voltage across the resonating capacitor obtained by simulating the equivalent circuit in PSIM and by Average modeling. As can be seen, the downfall of the average model is that it does not include the transient data points. This is due to the evaluation by small signal analysis where higher frequencies are eliminated from the model. Nonetheless, the average model presents an accurate mean value of the resonating capacitor voltage during duty cycle change. Several duty cycle disturbances were tested with a similar outcome. In both cases, the mean voltage value starts at 200V and is elevated to 320V as a result of duty cycle increase to 80%.

The relationship between the input and output voltage with respect to the duty cycle of the switched waveform appears to be similar to that of a buck converter's

$$d = \frac{V_{out}}{V_{in}} \tag{4.2a}$$

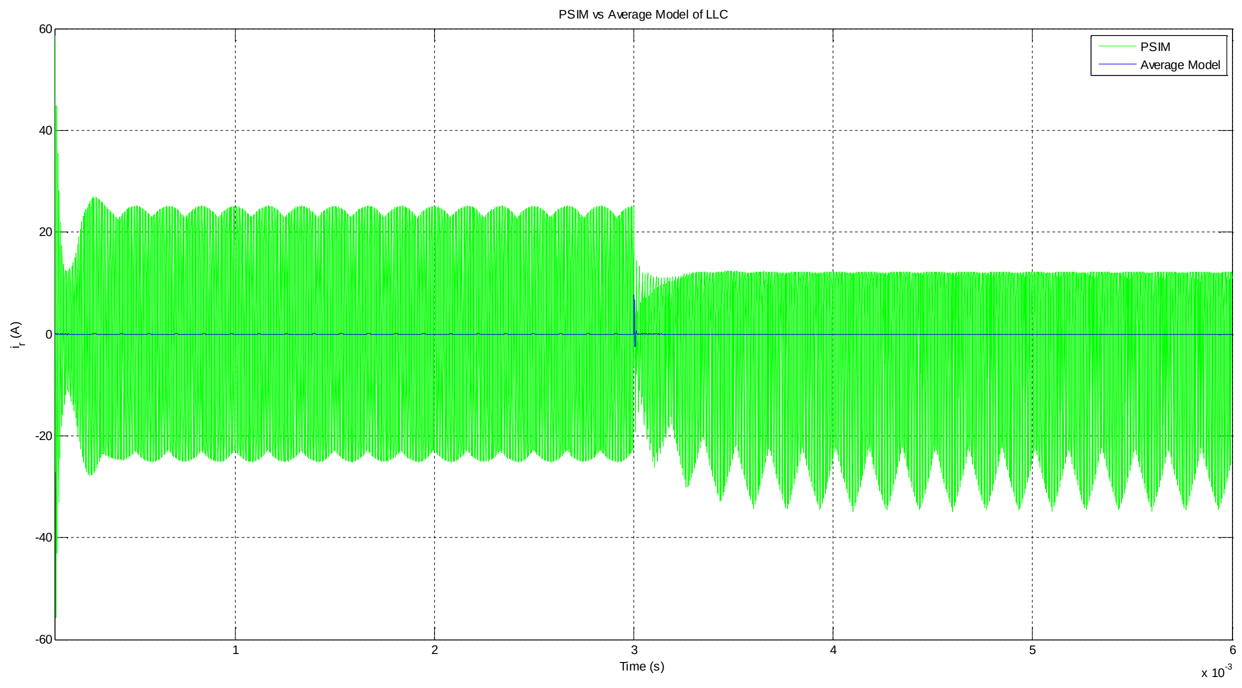


Figure A7- 4 : PSIM vs. Average Model: resonant inductor current comparison

It is hard to evaluate the inductor current in the model since its value fluctuates around zero amps. The average model shows the mean value and this value does not shift with duty cycle the same way capacitor voltage does, therefore the results are inconclusive. A duty cycle controlled model of the LLC converter must include the time response of the resonating current and resonating voltage. In some cases the value of the transients of the current computed by the average model appeared to show peak amperage equal to the peak value of the current in PSIM, but not in all cases.

Still, the current decreased with rising duty cycle, and the average model shows a decreased in the transient peak, even though it is of different amplitude than in PSIM.

The average model appears to be a good approximation of the behaviour of the capacitor voltage but not of the currents circulating in the LLC converter. Since the primary objective is to obtain a model that includes both duty cycle and frequency control the average model did not provide an adequate analysis of the circuit. Other methods were investigated for a more accurate representation of the LLC converter operation. One such method is based on the extended describing function (EDF)[3][4][5]. This method considers the sinusoidal components present in the state variables. Typically, the EDF approach is used to approximate the behavior of non-linear systems and convert them to linear and time invariant (LTI).

A.8 DLL Block C/C++ Code

A.8.1 ADC

```
#include "stdafx.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include "psimblock.h"
#include "psimutil.h"
#include "blockdata.h"

#define TYPE_PORT_INPUT   0
#define TYPE_PORT_OUTPUT  1
double d = 0;
int temp_1 = 0;
double vsamp=0;
double jj=0;
double jjj = 0;
double tmp = 0;
BOOL APIENTRY DllMain( HANDLE hModule,
DWORD  ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
```

```
break;
}
return TRUE;
}

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
#define MyApp_VERSION  "1.2"


class Internal_DLL_Block_RuntimeData
{
public:
Internal_DLL_Block_RuntimeData()
{
memset(m_szInputFile, 0, 260);
m_nInputNodes = 0;
m_nOutputNodes = 0;

m_arrayInputNodes = NULL;
m_arrayOutputNodes = NULL;
}

virtual ~Internal_DLL_Block_RuntimeData()
{
Clear();
}

void Clear()
{
if( m_arrayInputNodes != NULL )
{
for(int nCtr=0; nCtr<m_nInputNodes; nCtr++)
{
if( m_arrayInputNodes[nCtr] != NULL )
{
delete [] m_arrayInputNodes[nCtr];
}
}
delete [] m_arrayInputNodes;
}
if( m_arrayOutputNodes != NULL )
{
for(int nCtr=0; nCtr<m_nOutputNodes; nCtr++)
{
if( m_arrayOutputNodes[nCtr] != NULL )
{
delete [] m_arrayOutputNodes[nCtr];
}
}
delete [] m_arrayOutputNodes;
}
}


BOOL LoadFile(char * szFilePath)
{
```

```
char szTemp[300];

if( GetFileAttributes(szFilePath) == 0XFFFFFFFF )
{ //File does not exist
sprintf(szTemp, "File does not Exist.\r\nFilename: %s", szFilePath);
::MessageBox(NULL, szTemp, "My Program", MB_OK);
return FALSE;
}


//Open selected file.
FILE * inputStream = fopen( szFilePath, "r" );
if( inputStream == NULL )
{ //Reject file if can not open
sprintf(szTemp, "Failed to open file.\r\nFilename: %s", szFilePath);
::MessageBox(NULL, szTemp, "My Program", MB_OK);
return FALSE;
}


//Delete previously allocated memory for m_arrayInputNodes and m_arrayOutputNodes
Clear();

//Read input and output nodes from file.
int nCtr = 0;
int i = 0;
while( fgets( szTemp, 100, inputStream ) != NULL )
{
i = 0;
// Trim input and use  ;   for comment
while( (szTemp[i] != '\0') && (szTemp[i] != ';') )
{
i++;
}
i--;
while( (i >= 0) &&
((szTemp[i] == ' ') || (szTemp[i] == '\t') || (szTemp[i] == '\r') || (szTemp[i] == '\n')
)
)
{
i--;
}
szTemp[i+1] = '\0';



nCtr++;
if( nCtr == 1 )
{      //Get number of input nodes from file
m_nInputNodes = atoi(szTemp);
if(m_nInputNodes > 0)
{
m_arrayInputNodes = new LPSTR[m_nInputNodes];
memset(m_arrayInputNodes, 0, sizeof(LPSTR) * m_nInputNodes);
}
}
else if( nCtr == 2 )
{
```

```
//Get number of output nodes from file
m_nOutputNodes = atoi(szTemp);
if(m_nOutputNodes > 0)
{
m_arrayOutputNodes = new LPSTR[m_nOutputNodes];
memset(m_arrayOutputNodes, 0, sizeof(LPSTR) * m_nOutputNodes);
}
}
else if( (nCtr >= 3) && (nCtr < (3 + m_nInputNodes) ) )
{
//Get input node labels from file
m_arrayInputNodes[nCtr-3] = new char[strlen(szTemp)+2];
strcpy(m_arrayInputNodes[nCtr-3], szTemp);
}
else if( (nCtr >= (3 + m_nInputNodes)) && (nCtr < (3 + m_nInputNodes + m_nOutputNodes) )
)
{
//Get output node labels from file
m_arrayOutputNodes[nCtr-(3+m_nInputNodes)] = new char[strlen(szTemp)+2];
strcpy(m_arrayOutputNodes[nCtr-(3+m_nInputNodes)], szTemp);
}
else
{
//...
}
}
//end-of-file
fclose(inputStream);
inputStream = NULL;

if( ( (m_nInputNodes == 0) && (m_nInputNodes == 0) )  ||
(nCtr < (2 + m_nInputNodes + m_nOutputNodes) )
)
{
// file was not good
return FALSE;
}

//keep a copy of input file path
strcpy(m_szInputFile, szFilePath);

return TRUE;
}

char * GetInputNodeLabel(int nIndex) // nIndex: Zero based index
{
if( (m_arrayInputNodes != NULL) &&
( (nIndex >= 0) && (nIndex < m_nInputNodes) ) &&
(m_arrayInputNodes[nIndex] != NULL)
)
{
return m_arrayInputNodes[nIndex];
}
else
{
return m_emptyNode;
}
}
```

```
char * GetOutputNodeLabel(int nIndex) // nIndex: Zero based index
{
if( (m_arrayOutputNodes != NULL) &&
( (nIndex >= 0) && (nIndex < m_nOutputNodes) ) &&
(m_arrayOutputNodes[nIndex] != NULL)
)
{
return m_arrayOutputNodes[nIndex];
}
else
{
return m_emptyNode;
}
}




public:
char m_szInputFile[260];
int m_nInputNodes;
int m_nOutputNodes;

LPSTR * m_arrayInputNodes;
LPSTR * m_arrayOutputNodes;

static char m_emptyNode[2];
};

char Internal_DLL_Block_RuntimeData::m_emptyNode[2] = {'\0','\0'};


void REQUESTUSERDATA(int nRequestReason,
int nRequestCode,
int nRequestParam,
void ** ptrUserData,
int * pnParam1,
int * pnParam2,
char * szParam1,
char * szParam2
)
{

char szTemp[100];
int nNode;

Internal_DLL_Block_RuntimeData * pData = (Internal_DLL_Block_RuntimeData
*)(*ptrUserData);
Internal_DLL_Block_RuntimeData * pData2 = NULL;

switch(       nRequestReason )
{
case ACTION_DLL_SELECTED: //New Element was placed on the schematic window and this DLL
was selected.
{
switch(nRequestCode)
{
```

```
case REQUEST_BEGIN:
//Allocate User data
assert(*ptrUserData == NULL);
*ptrUserData = new Internal_DLL_Block_RuntimeData();
pData = (Internal_DLL_Block_RuntimeData *)(*ptrUserData);
return;

case REQUEST_IN_OUT_NODES:  //Define the number of nodes
// int * pnParam1(Read, Write):  returns the number of nodes.
// int * pnParam2(Read, Write):  set to 0. not used for Embedded Software Block.
*pnParam1 = 3;
*pnParam2 = 0;
return;


case REQUEST_INPUT_NODE_INFO:      //Define node names
//      this is called several times with  "nRequestParam" set to  0 to 'number of input
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
nNode = nRequestParam;
switch(nNode)
{
case 0:
strcpy(szParam1, "Vin");
break;

case 1:
strcpy(szParam1, "Vout");
break;

case 2:
strcpy(szParam1, "Fs");
break;
default:
//assert(0);
break;

}
return;

case REQUEST_PARAM_COUNT:   //Define number of input parameters
*pnParam1 = 4; // 5 parameters
*pnParam2 = 0; // Input Data File not required
strcpy(szParam1, "All Files|*.*||"); //File Open Dialog Filter for InputFile.
return;

case REQUEST_DATAFILE_INFO:
// Get Data File parameter information
// char * szParam1: Label  20 characters maximum.
// char * szParam2: Full file path   260 characters maximum.
// int * pnParam1:  1: Show Display check box in property Dialog box    0: Do not show
Display check box
strcpy(szParam1, "Input Data File");
*pnParam1 = 1; // Show Display check box
return;
```

```
case REQUEST_PARAM_INFO:    //Define input parameter names
{
// char * szParam1: parameter Label  20 characters maximum.
// char * szParam2: parameter default value  50 characters maximum.
// int * pnParam1:  1: Show Display check box     0: Do not show Display check box
switch(nRequestParam)
{
//One Parameter

case 0:
strcpy(szParam1, "Bits");
strcpy(szParam2, "8");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;

case 1:
strcpy(szParam1, "Vmin");
strcpy(szParam2, "0");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;

case 2:
strcpy(szParam1, "Vmax");
strcpy(szParam2, "3.3");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;

case 3:
strcpy(szParam1, "Sample Delay");
strcpy(szParam2, "0");//Default Value
*pnParam1 = 0; //Do not Show Display check box
break;

}
}
return;


default:
return;
}
}
return;

case ACTION_ELEMENT_LOAD:
{
switch(nRequestCode)
{
case REQUEST_BEGIN:
//Allocate User data
assert(*ptrUserData == NULL);
*ptrUserData = new Internal_DLL_Block_RuntimeData();
pData = (Internal_DLL_Block_RuntimeData *)(*ptrUserData);

// Copy saved data in schematic file to szTemp. in this case only the DLL version was
saved
if( *pnParam1 == 0 )
{
```

```
szTemp[0] = '\0';
}
else
{
memcpy(szTemp, szParam1, *pnParam1);
}

//Compare versions
if( atof(MyApp_VERSION) < atof(szTemp) )
{
::MessageBox(NULL, "Data was saved by Newer version of \"My Program\". Please upgrade.",
"My Program", MB_OK);
//Continue loading anyway
}

if( strlen(szParam2) > 0 )
{      //Reload input file.
pData->LoadFile(szParam2);
}
return;


case REQUEST_IN_OUT_NODES:
return;


case REQUEST_INPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of input
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Input node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
nNode = nRequestParam;
switch(nNode)
{
case 0:
//     strcpy(szParam1, "Vm");
break;

case 1:
//     strcpy(szParam1, "Vcarr");
break;

case 2:
//     strcpy(szParam1, "Vgat");
break;

default:
//assert(0);
break;

}
return;

case REQUEST_OUTPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of output
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Output node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
```

```
//strcpy(szParam1, pData->GetOutputNodeLabel(nRequestParam));
return;


//  Schematic file saves and restores parameter information from last session. unless DLL
version was changed
//     and parameter number or labels are different, there is no need to modify parameter
information
case REQUEST_PARAM_COUNT:
return;

case REQUEST_PARAM_INFO:
return;


default:
return;
}
}
return;


case ACTION_ELEMENT_SAVE:  //Saving element to schematic file
//           char * szParam1(Write only):  copy binary data to be saved in .SCH file(DLL
version, File Version, ...) (maximum 100 bytes)
//           int * pnParam1(Write only):   number of valid bytes in szParam1
//           char * szParam2(Read only): Selected Input file path
memcpy(szParam1, MyApp_VERSION, strlen(MyApp_VERSION)+1);
*pnParam1 = strlen(MyApp_VERSION)+1; //size of data
return;



case ACTION_INPUTFILE_CHANGED:
{
switch(nRequestCode)
{
case REQUEST_BEGIN:
//            char * szParam1(Read, Write): Selected Input file path
//            int * pnParam1(Write only):  0: Reject the file    1: set to 1 or Leave
unchanged to accept the file
pData2 = new Internal_DLL_Block_RuntimeData();
if( !(pData2->LoadFile(szParam1)) )
{
//Reject File.
*pnParam1 = 0;
delete pData2;
}

// file was good
if( pData != NULL )
{
//Delete old User data and assign new one
delete pData;
*ptrUserData = pData = pData2;
}
return;
```

```
case REQUEST_IN_OUT_NODES:
// Get the number of Input and Output nodes
// int * pnParam1(Read, Write):  returns the number of input nodes.
// int * pnParam2(Read, Write):  returns the number of output nodes.
*pnParam1 = pData->m_nInputNodes;
*pnParam2 = pData->m_nOutputNodes;
return;


case REQUEST_INPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of input
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Input node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
strcpy(szParam1, pData->GetInputNodeLabel(nRequestParam));
return;

case REQUEST_OUTPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of output
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Output node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
strcpy(szParam1, pData->GetOutputNodeLabel(nRequestParam));
return;

default:
return;
}
}
return;



case ACTION_ELEMENT_DELETE:
{
//Delete User data
pData = (Internal_DLL_Block_RuntimeData *)(*ptrUserData);

if( pData == NULL )
{
return;
}

delete pData;
*ptrUserData = NULL;
}
return;


case ACTION_PARAMETERS_CHANGED:  //parameters were changed in the property dialog box.
{
if( nRequestCode == REQUEST_PARAM_INFO )
{
//      this is called several times with  "nRequestParam" set to  0 to 'number of
parameters - 1 (set in REQUEST_PARAM_COUNT)'
// Get parameter information
// char * szParam1: parameter Label  20 characters maximum.
// char * szParam2: parameter default value  50 characters maximum.
// int * pnParam1:  1: Show Display check box     0: Do not show Display check box
switch(nRequestParam)
{
//Ten Parameters
case 0:
//Verify Parameter value
itoa(atoi(szParam2), szParam2, 10);//must be an integer
break;

case 1:
//itoa(atoi(szParam2), szParam2, 10);//must be an integer
break;

case 2:
break;

case 3:
break;

}
}
}
return;
}
}

// Simulation Functions

typedef struct Coeff2P2Z
{
double Bits;
double Vmin;
double Vmax;
double Fs;
double sampd;
} Coeff2P2Z;
/*
typedef struct Values2P2Z
{
double x0;
double x1;
double x2;
double y0;
double y1;
double y2;
} Values2P2Z;
*/
struct Internal_DLL_Block_SimulationData
{
int m_nNodes, m_nTmp;

char m_szInputFile[260];

// Add DLL Specific variables
int flag_exact_switching;
int previousSampleInput;
struct Coeff2P2Z loopCoeff2P2Z;
```

```
//      struct Values2P2Z loopValues2P2Z;

};


void OPENSIMUSER(const char *szId, const char * szNetlist, void ** ptrUserData, int
*pnError, LPSTR szErrorMsg, void * pPsimParams)
{

EXT_FUNC_PSIM_INFO * pPsimInfo = (EXT_FUNC_PSIM_INFO *)pPsimParams;


assert(*ptrUserData == NULL);
*ptrUserData = new Internal_DLL_Block_SimulationData;

Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
memset(pData, 0, sizeof(Internal_DLL_Block_SimulationData) );


CNetListParams netlist;

netlist.parse_netlist(szNetlist);


//assert( strcmp(netlist[0],"DLL_EXT") == 0 );
assert( strcmp(netlist[1], szId) == 0 );


pData->m_nNodes  = atoi(netlist[2]); //number of nodes

pData->m_nTmp = atoi(netlist[3]); // netlist[3] should be 0 for Embedded Software Block
// netlist[4] : DLL FilePath


int nParamStartIndex = 5 + pData->m_nNodes + pData->m_nTmp;

// value of parameter 0
pData->loopCoeff2P2Z.Bits = atof( netlist[nParamStartIndex] );

// value of parameter 1
pData->loopCoeff2P2Z.Vmin = atof( netlist[nParamStartIndex+1] );

// value of parameter 2
pData->loopCoeff2P2Z.Vmax = atof( netlist[nParamStartIndex+2] );

// value of parameter 3
pData->loopCoeff2P2Z.sampd = atof( netlist[nParamStartIndex+3] );

//Initialization
pData->previousSampleInput = 0;
*pnError = 0; //Success
}


void STARTSIMUSER(int *portTypes, void ** ptrUserData, int *pnError, LPSTR szErrorMsg)
{
```

```
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
if( pData == NULL) { return; }


//============================================================
// Place your code here............begin
portTypes[0] = TYPE_PORT_INPUT;
portTypes[1] = TYPE_PORT_OUTPUT;
portTypes[2] = TYPE_PORT_INPUT;

// Place your code here............end
//==============================================================


*pnError = 0; //Success
}

void RUNSIMUSER2(double t, double delt, double *ports, double *ports2, int *portTypes,
void ** ptrUserData, int *pnError, LPSTR szErrorMsg)
{
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
if( pData == NULL) { return; }

//============================================================
// Place your code here............begin
int iflag;
double sample_delay = pData->loopCoeff2P2Z.sampd;
double bits = pData->loopCoeff2P2Z.Bits;        //Store the #Bits set by the user
double tsample = ports[2];  //Store clock wave
bits = pow(2,bits);             //2^bits
double LSB = (pData->loopCoeff2P2Z.Vmax - pData->loopCoeff2P2Z.Vmin)/bits; //Calculates
LSB, (Vmax - Vmin)/(2^bits)
if(tsample == 1 && jj == 0) //Sample on Rising Edge of clock
{
vsamp = ports[0] + 0.5*LSB; //Sample the input value (port 0), + offset by 1/2LSB
jj = 1;                                         //dummy variable to prevent sampling
until next rising edge
jjj = 1;
}
if(tsample == 0)
{
jj = 0;                                  //when clock transistions from 1 to 0, reset
dummy variable and wait for next rising edge to sample
}
temp_1 = int(vsamp/LSB);    //compute the # of LSB's & convert to int -> ex. Sample Vin =
2.012V, LSB = 0.1, therefore it takes 2.012/0.1 = 20.12 -> int -> 20 LSB steps
d = temp_1*LSB;                          //multiply the # of LSB "steps" by the LSB to
get the ACTUAL number set by the ADC resolution
if(sample_delay == 0)
{
ports[1] = d;              //Output the value to port 1
}
if(sample_delay != 0 && jjj == 1)
{
ports[1] = tmp;                    //Output the delayed value to port 1
}
```

```
if(jj == 1 && jjj == 1)
{
tmp = d;
jjj = 0;
}
*pnError = 0;                     // Success
}

void CLOSESIMUSER(const char *szId, void ** ptrUserData)
{
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
assert(*ptrUserData != NULL);

if( pData == NULL )
{
return;
}

delete pData;
*ptrUserData = NULL;
}
```

A.8.2 HRPWM

```
#include "stdafx.h"
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include "psimblock.h"
#include "psimutil.h"
#include "blockdata.h"

#define TYPE_PORT_INPUT   0
#define TYPE_PORT_OUTPUT  1
double step = 0;
double time_on = 0;
double i = 0;
double k = 0;
BOOL APIENTRY DllMain( HANDLE hModule,
DWORD  ul_reason_for_call,
LPVOID lpReserved
)
{
switch (ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
case DLL_PROCESS_DETACH:
break;
}
return TRUE;
```

```
}

#define MyApp_VERSION  "1.2"


class Internal_DLL_Block_RuntimeData
{
public:
Internal_DLL_Block_RuntimeData()
{
memset(m_szInputFile, 0, 260);
m_nInputNodes = 0;
m_nOutputNodes = 0;

m_arrayInputNodes = NULL;
m_arrayOutputNodes = NULL;
}

virtual ~Internal_DLL_Block_RuntimeData()
{
Clear();
}

void Clear()
{
if( m_arrayInputNodes != NULL )
{
for(int nCtr=0; nCtr<m_nInputNodes; nCtr++)
{
if( m_arrayInputNodes[nCtr] != NULL )
{
delete [] m_arrayInputNodes[nCtr];
}
}
delete [] m_arrayInputNodes;
}
if( m_arrayOutputNodes != NULL )
{
for(int nCtr=0; nCtr<m_nOutputNodes; nCtr++)
{
if( m_arrayOutputNodes[nCtr] != NULL )
{
delete [] m_arrayOutputNodes[nCtr];
}
}
delete [] m_arrayOutputNodes;
}
}


BOOL LoadFile(char * szFilePath)
{
char szTemp[300];

if( GetFileAttributes(szFilePath) == 0XFFFFFFFF )
{ //File does not exist
sprintf(szTemp, "File does not Exist.\r\nFilename: %s", szFilePath);
::MessageBox(NULL, szTemp, "My Program", MB_OK);
```

```
return FALSE;
}


//Open selected file.
FILE * inputStream = fopen( szFilePath, "r" );
if( inputStream == NULL )
{ //Reject file if can not open
sprintf(szTemp, "Failed to open file.\r\nFilename: %s", szFilePath);
::MessageBox(NULL, szTemp, "My Program", MB_OK);
return FALSE;
}


//Delete previously allocated memory for m_arrayInputNodes and m_arrayOutputNodes
Clear();

//Read input and output nodes from file.
int nCtr = 0;
int i = 0;
while( fgets( szTemp, 100, inputStream ) != NULL )
{
i = 0;
// Trim input and use  ;   for comment
while( (szTemp[i] != '\0') && (szTemp[i] != ';') )
{
i++;
}
i--;
while( (i >= 0) &&
((szTemp[i] == ' ') || (szTemp[i] == '\t') || (szTemp[i] == '\r') || (szTemp[i] == '\n')
)
)
{
i--;
}
szTemp[i+1] = '\0';


nCtr++;
if( nCtr == 1 )
{      //Get number of input nodes from file
m_nInputNodes = atoi(szTemp);
if(m_nInputNodes > 0)
{
m_arrayInputNodes = new LPSTR[m_nInputNodes];
memset(m_arrayInputNodes, 0, sizeof(LPSTR) * m_nInputNodes);
}
}
else if( nCtr == 2 )
{
//Get number of output nodes from file
m_nOutputNodes = atoi(szTemp);
if(m_nOutputNodes > 0)
{
m_arrayOutputNodes = new LPSTR[m_nOutputNodes];
memset(m_arrayOutputNodes, 0, sizeof(LPSTR) * m_nOutputNodes);
}
}
else if( (nCtr >= 3) && (nCtr < (3 + m_nInputNodes) ) )
{
//Get input node labels from file
m_arrayInputNodes[nCtr-3] = new char[strlen(szTemp)+2];
strcpy(m_arrayInputNodes[nCtr-3], szTemp);
}
else if( (nCtr >= (3 + m_nInputNodes)) && (nCtr < (3 + m_nInputNodes + m_nOutputNodes) )
)
{
//Get output node labels from file
m_arrayOutputNodes[nCtr-(3+m_nInputNodes)] = new char[strlen(szTemp)+2];
strcpy(m_arrayOutputNodes[nCtr-(3+m_nInputNodes)], szTemp);
}
else
{
//...
}
}
//end-of-file
fclose(inputStream);
inputStream = NULL;

if( ( (m_nInputNodes == 0) && (m_nInputNodes == 0) )  ||
(nCtr < (2 + m_nInputNodes + m_nOutputNodes) )
)
{
// file was not good
return FALSE;
}

//keep a copy of input file path
strcpy(m_szInputFile, szFilePath);

return TRUE;
}

char * GetInputNodeLabel(int nIndex) // nIndex: Zero based index
{
if( (m_arrayInputNodes != NULL) &&
( (nIndex >= 0) && (nIndex < m_nInputNodes) ) &&
(m_arrayInputNodes[nIndex] != NULL)
)
{
return m_arrayInputNodes[nIndex];
}
else
{
return m_emptyNode;
}
}

char * GetOutputNodeLabel(int nIndex) // nIndex: Zero based index
{
if( (m_arrayOutputNodes != NULL) &&
( (nIndex >= 0) && (nIndex < m_nOutputNodes) ) &&
(m_arrayOutputNodes[nIndex] != NULL)
)
```

```
{
return m_arrayOutputNodes[nIndex];
}
else
{
return m_emptyNode;
}
}


public:
char m_szInputFile[260];
int m_nInputNodes;
int m_nOutputNodes;

LPSTR * m_arrayInputNodes;
LPSTR * m_arrayOutputNodes;

static char m_emptyNode[2];
};

char Internal_DLL_Block_RuntimeData::m_emptyNode[2] = {'\0','\0'};


void REQUESTUSERDATA(int nRequestReason,
int nRequestCode,
int nRequestParam,
void ** ptrUserData,
int * pnParam1,
int * pnParam2,
char * szParam1,
char * szParam2
)
{

char szTemp[100];
int nNode;

Internal_DLL_Block_RuntimeData * pData = (Internal_DLL_Block_RuntimeData
*)(*ptrUserData);
Internal_DLL_Block_RuntimeData * pData2 = NULL;

switch(       nRequestReason )
{
case ACTION_DLL_SELECTED: //New Element was placed on the schematic window and this DLL
was selected.
{
switch(nRequestCode)
{
case REQUEST_BEGIN:
//Allocate User data
assert(*ptrUserData == NULL);
*ptrUserData = new Internal_DLL_Block_RuntimeData();
pData = (Internal_DLL_Block_RuntimeData *)(*ptrUserData);
return;

case REQUEST_IN_OUT_NODES:  //Define the number of nodes
// int * pnParam1(Read, Write):  returns the number of nodes.
// int * pnParam2(Read, Write):  set to 0. not used for Embedded Software Block.
*pnParam1 = 2;
*pnParam2 = 0;
return;


case REQUEST_INPUT_NODE_INFO:      //Define node names
//      this is called several times with  "nRequestParam" set to  0 to 'number of input
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
nNode = nRequestParam;
switch(nNode)
{
case 0:
strcpy(szParam1, "Fs");
break;

case 1:
strcpy(szParam1, "PWM");
break;

default:
//assert(0);
break;

}
return;

case REQUEST_PARAM_COUNT:   //Define number of input parameters
*pnParam1 = 3; // 3 parameters
*pnParam2 = 0; // Input Data File not required
strcpy(szParam1, "All Files|*.*||"); //File Open Dialog Filter for InputFile.
return;

case REQUEST_DATAFILE_INFO:
// Get Data File parameter information
// char * szParam1: Label  20 characters maximum.
// char * szParam2: Full file path   260 characters maximum.
// int * pnParam1:  1: Show Display check box in property Dialog box    0: Do not show
Display check box
strcpy(szParam1, "Input Data File");
*pnParam1 = 1; // Show Display check box
return;


case REQUEST_PARAM_INFO:    //Define input parameter names
{
// char * szParam1: parameter Label  20 characters maximum.
// char * szParam2: parameter default value  50 characters maximum.
// int * pnParam1:  1: Show Display check box     0: Do not show Display check box
switch(nRequestParam)
{
//One Parameter

case 0:
strcpy(szParam1, "System Frequency (MHz)");
strcpy(szParam2, "60");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;

case 1:
strcpy(szParam1, "Duty Cycle (%)");
strcpy(szParam2, "40.5");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;

case 2:
strcpy(szParam1, "MEP Step (ps)");
strcpy(szParam2, "180");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;

/*
case 5:
strcpy(szParam1, "Sampling Frequency (Hz)");
strcpy(szParam2, "150000");//Default Value
*pnParam1 = 0; // Do not Show Display check box
break;
*/
}
}
return;

default:
return;
}
}
return;



case ACTION_ELEMENT_LOAD:
{
switch(nRequestCode)
{
case REQUEST_BEGIN:
//Allocate User data
assert(*ptrUserData == NULL);
*ptrUserData = new Internal_DLL_Block_RuntimeData();
pData = (Internal_DLL_Block_RuntimeData *)(*ptrUserData);

// Copy saved data in schematic file to szTemp. in this case only the DLL version was
saved
if( *pnParam1 == 0 )
{
szTemp[0] = '\0';
}
else
{
memcpy(szTemp, szParam1, *pnParam1);
}

//Compare versions
if( atof(MyApp_VERSION) < atof(szTemp) )
{
```

```
::MessageBox(NULL, "Data was saved by Newer version of \"My Program\". Please upgrade.",
"My Program", MB_OK);
//Continue loading anyway
}

if( strlen(szParam2) > 0 )
{      //Reload input file.
pData->LoadFile(szParam2);
}
return;


case REQUEST_IN_OUT_NODES:
// Get the number of nodes
// int * pnParam1(Read, Write):  returns the number of nodes.
// int * pnParam2(Read, Write):  not used for Embedded Software Block
//     *pnParam1 = 0;
//     *pnParam2 = 0;
return;


case REQUEST_INPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of input
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Input node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
nNode = nRequestParam;
switch(nNode)
{
case 0:
//     strcpy(szParam1, "Vm");
break;

case 1:
//     strcpy(szParam1, "Vcarr");
break;

case 2:
//     strcpy(szParam1, "Vgat");
break;

default:
//assert(0);
break;

}
return;

case REQUEST_OUTPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of output
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Output node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
//strcpy(szParam1, pData->GetOutputNodeLabel(nRequestParam));
return;
```

```
//  Schematic file saves and restores parameter information from last session. unless DLL
version was changed
//     and parameter number or labels are different, there is no need to modify parameter
information
case REQUEST_PARAM_COUNT:
return;

case REQUEST_PARAM_INFO:
return;


default:
return;
}
}
return;


case ACTION_ELEMENT_SAVE:  //Saving element to schematic file
//            char * szParam1(Write only):  copy binary data to be saved in .SCH file(DLL
version, File Version, ...) (maximum 100 bytes)
//            int * pnParam1(Write only):   number of valid bytes in szParam1
//            char * szParam2(Read only): Selected Input file path
memcpy(szParam1, MyApp_VERSION, strlen(MyApp_VERSION)+1);
*pnParam1 = strlen(MyApp_VERSION)+1; //size of data
return;



case ACTION_INPUTFILE_CHANGED:
{
switch(nRequestCode)
{
case REQUEST_BEGIN:
//             char * szParam1(Read, Write): Selected Input file path
//             int * pnParam1(Write only):  0: Reject the file    1: set to 1 or Leave
unchanged to accept the file
pData2 = new Internal_DLL_Block_RuntimeData();
if( !(pData2->LoadFile(szParam1)) )
{
//Reject File.
*pnParam1 = 0;
delete pData2;
}

// file was good
if( pData != NULL )
{
//Delete old User data and assign new one
delete pData;
*ptrUserData = pData = pData2;
}
return;


case REQUEST_IN_OUT_NODES:
// Get the number of Input and Output nodes
// int * pnParam1(Read, Write):  returns the number of input nodes.
// int * pnParam2(Read, Write):  returns the number of output nodes.
*pnParam1 = pData->m_nInputNodes;
*pnParam2 = pData->m_nOutputNodes;
return;


case REQUEST_INPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of input
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Input node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
strcpy(szParam1, pData->GetInputNodeLabel(nRequestParam));
return;

case REQUEST_OUTPUT_NODE_INFO:
//      this is called several times with  "nRequestParam" set to  0 to 'number of output
nodes - 1 (set in REQUEST_IN_OUT_NODES)'
//             Get Output node information
//             char * szParam1(Read, Write): Node Label  20 characters maximum.
strcpy(szParam1, pData->GetOutputNodeLabel(nRequestParam));
return;

default:
return;
}
}
return;



case ACTION_ELEMENT_DELETE:
{
//Delete User data
pData = (Internal_DLL_Block_RuntimeData *)(*ptrUserData);

if( pData == NULL )
{
return;
}

delete pData;
*ptrUserData = NULL;
}
return;


case ACTION_PARAMETERS_CHANGED:  //parameters were changed in the property dialog box.
{
if( nRequestCode == REQUEST_PARAM_INFO )
{
//      this is called several times with  "nRequestParam" set to  0 to 'number of
parameters - 1 (set in REQUEST_PARAM_COUNT)'
// Get parameter information
// char * szParam1: parameter Label  20 characters maximum.
// char * szParam2: parameter default value  50 characters maximum.
// int * pnParam1:  1: Show Display check box     0: Do not show Display check box
switch(nRequestParam)
{
```

```
//Ten Parameters
case 0:
//Verify Parameter value
itoa(atoi(szParam2), szParam2, 10);//must be an integer
break;

case 1:
//itoa(atoi(szParam2), szParam2, 10);//must be an integer
break;

case 2:
break;

case 3:
break;

}
}
}
return;
}
}


/////////////////////////////////////////////////////////////////////////////////////////
// Simulation Functions

typedef struct Coeff2P2Z
{
double sysfreq;
double duty;
double MEP;
} Coeff2P2Z;

struct Internal_DLL_Block_SimulationData
{
int m_nNodes, m_nTmp;

char m_szInputFile[260];

// Add DLL Specific variables
int flag_exact_switching;
int previousSampleInput;
struct Coeff2P2Z loopCoeff2P2Z;
//     struct Values2P2Z loopValues2P2Z;

};


void OPENSIMUSER(const char *szId, const char * szNetlist, void ** ptrUserData, int
*pnError, LPSTR szErrorMsg, void * pPsimParams)
{

EXT_FUNC_PSIM_INFO * pPsimInfo = (EXT_FUNC_PSIM_INFO *)pPsimParams;


assert(*ptrUserData == NULL);
*ptrUserData = new Internal_DLL_Block_SimulationData;
```

```
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
memset(pData, 0, sizeof(Internal_DLL_Block_SimulationData) );


CNetListParams netlist;

netlist.parse_netlist(szNetlist);


//assert( strcmp(netlist[0],"DLL_EXT") == 0 );
assert( strcmp(netlist[1], szId) == 0 );


pData->m_nNodes  = atoi(netlist[2]); //number of nodes

pData->m_nTmp = atoi(netlist[3]); // netlist[3] should be 0 for Embedded Software Block
// netlist[4] : DLL FilePath


int nParamStartIndex = 5 + pData->m_nNodes + pData->m_nTmp;

// value of parameter 0
pData->loopCoeff2P2Z.sysfreq = atof( netlist[nParamStartIndex] );

// value of parameter 1
pData->loopCoeff2P2Z.duty = atof( netlist[nParamStartIndex+1] );

// value of parameter 2
pData->loopCoeff2P2Z.MEP = atof( netlist[nParamStartIndex+2] );

//Initialization
pData->previousSampleInput = 0;
*pnError = 0; //Success
}


void STARTSIMUSER(int *portTypes, void ** ptrUserData, int *pnError, LPSTR szErrorMsg)
{
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
if( pData == NULL) { return; }


//==========================================================
// Place your code here............begin
portTypes[0] = TYPE_PORT_INPUT;
portTypes[1] = TYPE_PORT_OUTPUT;

// Place your code here............end


*pnError = 0; //Success
}

void RUNSIMUSER2(double t, double delt, double *ports, double *ports2, int *portTypes,
void ** ptrUserData, int *pnError, LPSTR szErrorMsg)
```

```
{
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
if( pData == NULL) { return; }

//============================================================
// Place your code here...........begin
int iflag;
double sysfreq = (pData->loopCoeff2P2Z.sysfreq); //
sysfreq = sysfreq*pow(10.0,6);                                // Convert to MHz
double duty = pData->loopCoeff2P2Z.duty/100;     //
int MEP = pData->loopCoeff2P2Z.MEP * pow(10.0,-12);
double pwmf = ports[0];                                              //Store pwm
frequency
time_on = (1/pwmf)*duty;
double ti = (1/pwmf);
if(i <= (time_on - (1/sysfreq)))
{
i+= 1/sysfreq;
ports[1] = 1;
}
if(i > (time_on-(1/sysfreq)) && i < time_on)
{
i += MEP;
ports[1] = 1;
}
if(i > time_on && i < (ti-(1/sysfreq)))
{
ports[1] = 0;
i+=1/sysfreq;
}
if(i > (ti-(1/sysfreq)) && i < ti)
{
ports[1] = 0;
i += MEP;
}
if(i >= ti)
{
i = 0;
}

*pnError = 0;                     //Success
}

void CLOSESIMUSER(const char *szId, void ** ptrUserData)
{
Internal_DLL_Block_SimulationData * pData = (Internal_DLL_Block_SimulationData
*)(*ptrUserData);
assert(*ptrUserData != NULL);

if( pData == NULL )
{
return;
}

delete pData;
*ptrUserData = NULL;
}
```