Fall 2013

Inverter design and analysis using multiple reference frame theory

Luke Dale Watson

Department: Electrical and Computer Engineering

INVERTER DESIGN AND ANALYSIS USING MULTIPLE REFERENCE FRAME THEORY

by

LUKE DALE WATSON

A DISSERTATION

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

ELECTRICAL ENGINEERING

2013

Approved
Jonathan Kimball, Advisor
Mehdi Ferdowsi
Mariesa Crow
Jagannathan Sarangapani
Bruce McMillin

ABSTRACT

Multiple reference frame theory allows for periodically time varying signals to be represented as a set of dc signals. In other words, every periodic signal can be expanded into a Fourier series representation. By modeling an inverter connected to a boost maximum power point tracker (MPPT) in this manner, frequency transfer properties can be preserved and harmonics throughout the system can be predicted. A state space model taking into account the dc and fundamental grid frequency is presented and used to optimize the controller gains of the system. Using information from the dq-axis values of the measured grid current and voltage, the double frequency dc-link voltage component is predicted. The double frequency component is removed from the controller input using feedforward. As a result, there is a reduction in output harmonics in the grid current. The same method is applied to the MPPT, where the double frequency component is predicted and removed from the controller input. This allows for a MPPT with reduced oscillations in the input power waveform. Next, a method is presented to generate a large-signal model of a H-bridge inverter. A set of algorithms are presented, which take a standard set of large-signal (user generated) dynamic equations and performs a Fourier series expansion on the inputs and states of the equations. These algorithms work for an arbitrary finite set of harmonics and preserve the frequency transfer properties between harmonics. The solution to the generated equations is the steady state output of the inverter. Lastly, a set of algorithms are presented which take a user generated netlist in and automatically outputs a truncated harmonic transfer function (THTF).

ACKNOWLEDGMENTS

To complete this dissertation, I spent many sleepless nights hunched over my desk surrounded by lab notebooks and wadded up scrap paper. Despite all of the technical challenges that I dealt with, one of the sections I struggled with writing most was this section. To my professors, my friends, and my family, I cannot find the words to describe how truly grateful I am for all of your support.

First, I would like to thank my advisor, Dr. Jonathan Kimball for providing me the opportunity to pursue a PhD. He has been a great role model and will be someone I strive to be like as I start my career. His experience in both academia and industry has proven invaluable. I would not be where I am today if it was not for Dr. Kimball, and it has been a pleasure working with him the past five years.

I would also like to thank my committee members, Dr. Mehdi Ferdowsi, Dr. Mariesa Crow, Dr. Jagannathan Sarangapani, and Dr. Bruce McMillin. All of you have provided me with your valuable time and guidance while on my road to completing this dissertation.

I have had the opportunity of working with Dr. Stan Atcitty for two summers while pursuing my PhD. Dr. Atcitty has provided me with many exciting opportunities ranging from networking with leading experts in the field of energy storage to traveling to the heart of Navajo Nation. He has taken many students under his wing during his career and I consider myself lucky to have been one of them.

Lastly, I would like to thank my parents, Dale and Deana Watson, for their love and support both growing up and during my time as a student in Rolla. You guys rock. A portion of my studies were supported on a fellowship from the Department of Education. This work was supported in part by the National Science Foundation under award ECCS-0900940.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOMENCLATURE

$\hat{\mathbf{A}}$	Maps dependent currents to independent currents (dc case)
$\mathbf{A_a}$	Node incidence matrix
$\hat{\mathbf{A}}_{dq}$	Maps dependent currents to independent currents (dq case)
$(a)_{d\ell\omega}$	Dc component of a variable *a*
$(a)_{d\ell\omega}$	D-axis component of a variable *a* at the ℓth harmonic
$(a)_{q\ell\omega}$	Q-axis component of a variable *a* at the ℓth harmonic
$\mathbf{A_{sys}}$	A matrix for generating system transfer functions
$\mathbf{A_{tf}}$	A matrix for generating low level transfer functions
b	Number of branches in a netlist
$\mathbf{b_{dqi}}$	Maps input current to dependent current (dq case)
$\mathbf{b_i}$	Maps input current to dependent current (dc case)
$\mathbf{B_{sys}}$	B matrix for generating system level transfer functions
$\mathbf{B_{tf}}$	B matrix for generating low level transfer functions
$\mathbf{b_{dqv}}$	Maps input voltage to dependent voltage (dq case)
$\mathbf{b_v}$	Maps input voltage to dependent voltage (dc case)
C	Dc-link capacitance
d_{dc}	output duty ratio of the MPPT converter
d_{ac}	120 Hz perturbation added to d_{dc}
d_{2dq}	d_{ac} in the 2dq axis
$\mathbf{E}$	Maps independent voltages to dependent currents (dc case)
$\mathbf{E_{dq}}$	Maps independent voltages to dependent currents (dq case)

$\mathbf{F}$	Maps independent currents to dependent voltages (dc case)
$\mathbf{F_{dq}}$	Maps independent currents to dependent voltages (dq case)
$\mathbf{G_{dq}}$	Maps dependent currents to dependent currents (dq case)
$\mathbf{H_{dq}}$	Maps dependent voltages to dependent voltages (dq case)
$\mathbf{I}$	Identity matrix
$\mathbf{i_{dqx}}$	Vector of dependent currents (dq case)
$\mathbf{i_{dqy}}$	Vector of independent currents (dq case)
i_{grid}*	commanded grid current
i_{grid}	grid current
i_{griddq}	i_{grid} in the dq-axis
i_{in}	dc-link input current
i_{inv}	current entering the inverter from the dc-link
i_{inv2dq}	120 Hz dq component of i_{inv}
i_{invdc}	dc component of i_{inv}
i_{L1}	L_1 current
i_{Ld}	d-axis current through L at 1ω (60 Hz grid current)
i_{L1dq}	i_{L1} in the dq-axis
i_{MPPT}	current into the dc-link from the MPPT converter
i_{pv}	PV panel current
$\mathbf{i_x}$	Vector of dependent currents (dc case)
$\mathbf{i_y}$	Vector of independent currents (dc case)
K_{idc}	Dc-link voltage loop integral gain
K_{idq}	Grid current loop integral gain

K_{ii} inverter current loop integral gain

K_{ipv} P&O voltage loop integral gain

K_{pdc} Dc-link voltage loop proportional gain

K_{pdq} Grid current loop proportional gain

K_{pi} inverter current loop proportional gain

K_{ppv} P&O voltage loop proportional gain

L Filter inductance

M Number of states in the system

n Number of nodes at the circuit level

$\mathbf{N}$ Netlist

N_i Number of system inputs

N_{max} Number of harmonics to be analyzed

$N_{sysNode}$ Number of nodes at the system level

N_ω Number of harmonics to analyze

$\mathbf{u}$ Input vector (dc case)

u System input

$\mathbf{u_{dq}}$ Input vector (dq case)

v_{av} average voltage on the inverter's grid side over a switching period

v_{avdq} v_{av} in the dq-axis

v_{dc} dc-link voltage

v_{dc}^{*} Commanded dc-link voltage

v_{dc2dq} 120 Hz dq component of v_{dc}

v_{dcac} ac component of v_{dc}

v_{dcdc} dc component of v_{dc}

$\mathbf{v_{dqx}}$ Vector of dependent voltages (dq case)

$\mathbf{v_{dqy}}$ Vector of independent voltages (dq case)

v_{grid} grid voltage

$v_{griddelay}$ grid voltage delayed 90°

v_{L1} MPPT converter inductor voltage

v_{pv} PV panel voltage

$\mathbf{v_x}$ Vector of dependent voltages (dc case)

$\mathbf{v_y}$ Vector of independent voltages (dc case)

x System state

x_{dc} dc component of a signal

x_{dk} d-axis component at the kth harmonic of a signal

x_{qk} q-axis component at the kth harmonic of a signal

1. INTRODUCTION

1.1. OVERVIEW OF POPULAR TOPOLOGIES

There has recently been an increase in the amount of research in single phase inverters. This is partially due to an increase in the study of microinverters. Microinverters are single phase inverters typically connected to a maximum power point tracker (MPPT). They are connected to one or a few photovoltaic (PV) panels, instead of many series and parallel connected PV panels (such as in a standard inverter case). Since more inverters are required using this scheme, it is vital that components be low cost while keeping the reliability high and the amount of harmonics injected into the grid low [1-9].

Microinverters are advantageous in that each inverter covers a relatively small area. This is beneficial in the case of partial shading. If even just one cell in a series connected string is partially shaded, the total output power of the entire string is disproportionately affected. Microinverters alleviate this problem by giving each PV panel its own individual MPPT [10-15].

Another advantage of microinverters is that they result in less resistance losses in cables and connections. They are safer during installation and maintenance, since the output of each module is a standard 60 Hz 120 V (in the US) output. As a result of the output voltage of each PV module being a standard output, installation is also faster. A potential downside of microinverters is that since they are installed on the back of the PV panel, they are exposed to higher temperatures [16, 17]. Some of the more popular microinverter topologies are discussed below.

One topology introduced by Huang-Jen in [18] is to use an "inverse-buck current-fed isolated dual-boost" converter as the MPPT. This requires more switches than a standard buck or boost MPPT (two mosfets and three diodes required in the MPPT). However, two film capacitors can be used in lieu of a large electrolytic capacitor, which will increase the system's reliability.

Another method which eliminates the need for a large dc-link capacitor is to use a modified flyback converter [2, 4, 19, 20]. An extra power decoupling circuit is added to the flyback MPPT, which reduces the size of capacitor required on the dc-link. This topology offers an efficient low cost isolated solution to microinverters.

Yeong-Chan took another approach in [21]. In this topology, a simple diode is used on the PV panel connected to a grid-tied H-bridge inverter. The commanded dc-link voltage is changed to modify the MPP of the converter. This is a simple solution with minimal components, but requires a large electrolytic dc-link capacitor as a tradeoff.

Work has been done on single phase boost dc-ac inverters. These converters can generate an output voltage that is higher than the dc-link voltage. Only four mosfets are required in this topology. This topology has potential as a single stage MPPT microinverter. A big issue in using these converters as an MPPT is that the input power waveform has a large sinusoidal component if a large electrolytic capacitor is not used on the input [22-32].

A novel topology introduced by Shimizu [6, 33, 34] is a half bridge topology with a sting of PV panels in series. Because of the switch arrangement, each individual PV panel can operate at its MPP, due to the generation control circuit (GCC). One downside to this converter is that there is a high voltage dc-link connection between each module in

a string. Also, there must be enough PV panels in series to have a greater voltage on the dc-link than the grid voltage. This is due to the inverter only being capable of bucking the voltage from the dc-link to the ac side.

A few MPPT topologies that are good candidates for microinverters were listed above. All of the topologies discussed above can benefit from the analysis in the following sections, even though only one topology was used an example. The topology analyzed in this dissertation is a standard dc-dc boost converter connected to a standard H-bridge inverter, similar to [35-43]. This topology has the advantage of being simple to understand and is very common.

1.2. OVERVIEW OF DQ CONVERSION METHODS

Each of the methods described in the following sections is highly dependent on arithmetic in the dq-axis synchronous reference frame. The synchronous reference frame is a method to analyze ac signals using dc techniques. In generating a dq signal from a single phase source, a false orthogonal signal must be generated. There are multiple ways to generate this signal.

Ryan [44] uses an LC output filter on the H-bridge inverter. Since the current on the output capacitor is 90° out of phase from the grid voltage, it can be scaled and used as the orthogonal signal for the conversion to the dq-axis. This is similar to using differentiation to generate the orthogonal signal [45], without the adverse effect of magnifying measurement noise.

Another method is to multiply the input signal by $2\sin(\omega t)$ and $2\cos(\omega t)$. The effect of this can be seen by,

$$\begin{aligned} 2\cos(\omega t)\times v_{grid}\sin(\omega t+\varphi) &= \sin(2\omega t+\varphi)-\sin(-\varphi) \\ 2\sin(\omega t)\times v_{grid}\sin(\omega t+\varphi) &= \cos(-\varphi)-\cos(2\omega t+\varphi) \end{aligned}, \quad (1)$$

where $v_{grid}sin(\omega t+\varphi)$ is the signal to be converted to a dq quantity. After multiplication, a dc term plus a double frequency term remains. By placing this signal through a notch filter tuned at the double frequency, only the desired dc d and q axis terms remain [46, 47].

An all pass filter can also be used to generate the orthogonal signal [48, 49]. This method works by creating a filter which has unity gain across its frequency spectrum and a 90° phase delay at ω. The all pass filter method is illustrated as a block diagram in Figure 1.1.

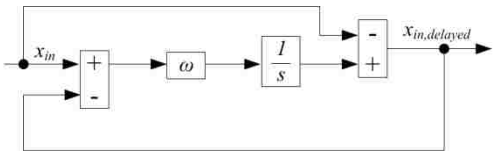

Figure 1.1. All pass filter block diagram.

The simplest method utilized in [46, 50-52] is to simply place the measured signal into a buffer and delay the output $1/(4f_s)$ s. This method requires little computational power compared to the all pass filter method and the notch filter method. It does have a slower response than these methods, however, and is not recommended for systems requiring a fast response [46].

The delay method is used in this study, due to its popularity and ease of implementation. Even though all of the examples in this dissertation utilize the delay

method for single phase dq conversion, all of the mentioned conversion methods can be analyzed using the proposed methods.

1.3. MULTIPLE REFERENCE FRAME THEORY

An extension of the simple dq arithmetic is multiple reference frame theory (MRF). In MRF, in addition to a reference frame rotating at the fundamental grid frequency ω, there are reference frames rotating at the harmonics of interest. MRF was first introduced by Krause [53] to allow for dc circuit theory to be used to analyze a symmetrical induction machine. It was later expanded to include a multitude of other electric machines [54-56].

MRF can estimate the amplitude of each harmonic on the output grid current in three phase converters. One method to reduce the unwanted harmonics is to feed the harmonic amplitudes into a proportional integral (PI) loop, which is commanded to reduce each harmonic to zero. MRF proves to be an effective method of reducing output harmonics in the grid current [57-59].

The previous MRF publications were all three phase examples. Padmavathi [60] uses MRF to reduce harmonics in a single phase system. The MRF used in Section 2 of this dissertation varies from the previously mentioned studies, in that the predicted harmonic amplitudes are not fed into a PI loop. Instead, they are used as feedforward terms.

MRF is one method to decompose a periodic signal into its harmonic amplitudes. Another way of saying this is that the signal can be decomposed into its Fourier series

coefficients. Since MRF is where Fourier series concepts presented in this dissertation originated from, the Fourier series subscripts remain *d* and *q*.

1.4. HARMONIC BALANCE

Another concept utilized in each of the studies presented in this dissertation is harmonic balance [61-63]. Harmonic balance takes a periodically time varying signal and converts it to a time invariant signal. This is done by separating the signal into its various harmonics,

$$y = x_{dc} + \sum_{k=1}^{\infty}\left[x_{dqk} e^{j\omega kt} \right], \tag{2}$$

then equating like harmonics,

$$\begin{aligned} y_{dc} &= x_{dc} \\ y_{dq1} e^{j\omega 1t} &= x_{dq1} e^{j\omega 1t} \\ y_{dq2} e^{j\omega 2t} &= x_{dq2} e^{j\omega 2t} \\ &\vdots \end{aligned} \tag{3}$$

After this, the periodically time varying terms can be cancelled,

$$\begin{aligned} y_{dc} &= x_{dc} \\ y_{dq1} &= x_{dq1} \\ y_{dq2} &= x_{dq2} \\ &\vdots \end{aligned} . \tag{4}$$

1.5. HARMONIC TRANSFER FUNCTION

Harmonic balance allows for linear time periodic systems (LTP) to be modeled as linear time invariant (LTI). This concept can be applied to transfer functions. By generating transfer functions which use the input's amplitude at a specific harmonic and

output the response as an amplitude at a specific harmonic, a harmonic transfer function (HTF) is created. A HTF is an infinitely dimensional matrix of s-domain transfer functions, which describes the input to output relationship of every harmonic of every system input to every harmonic of every system output. The infinite matrix is a trade off for the capability of modeling an LTP system as an LTI system [62-64].

To simplify the generation of the HTF, a truncated harmonic transfer function (THTF) is used in this dissertation. The THTF only generates transfer functions for dc and an arbitrary number of harmonics. This makes model generation much simpler and allows for algorithms to aid in their generation (presented in subsequent sections).

1.6. WORK SUMMARY

In Section 2, introduces a feedforward method to eliminate the effects caused by the double frequency ripple on the dc-link in a single-phase inverter. One issue is the double frequency ripple passes through the dc-link voltage loop, and appears as a third harmonic on the grid current. The third harmonic then reflects back to the dc-link as a second and fourth harmonic term. This effect is a major source of output harmonics in single phase inverters. The feedforward method allows the dc-link voltage loop to ignore the double frequency component of the dc-link voltage.

The double frequency dc-link ripple also travels through the MPPT and appears on the PV power waveform. This causes the MPPT to track slightly below the potential MPP of the PV panel. A feedforward term is added to the output of the MPPT algorithm which offsets the ripple due to the double frequency dc-link voltage term.

Section 3 proposes a set of algorithms which are capable of predicting the steady state operating point of a system. First, the user must generate a set of large-signal dynamic equations to describe the system. The algorithms then convert the equations to a set of harmonic functions, which use exponentially modulated periodic (EMP) signals as inputs and outputs.

After the harmonic equations are generated, a solver is used to solve for the steady state operating point of the system. The advantage of the proposed method is that if the input harmonics to the system are known, the harmonic amplitudes at every point in the system can be solved for directly. Normally, a simulation needs to be run until well after the system reaches steady state, then a fast Fourier transform is performed on the data. Using the proposed method, the steady state harmonics can be directly calculated, allowing the user to change component values and controller gains and receive feedback in a timely manner. Matlab code for Section 3 is found in Appendix A.

Section 4 proposes a set of algorithms to generate a THTF for a system given a user generated netlist. Generating a THTF by hand for even a small system is impractical, due to the sheer number of equations required. The proposed method allows for the THTF to be generated automatically, eliminating the time consuming and error prone process of generating the THTF by hand.

There are a few advantages of the THTF over standard TF's. First, frequency transfer properties are maintained between the dc-link and the ac side of the inverter, which allows for a highly accurate linear model. Next, because of how each point in the system is represented as a *d* and *q* axis value, blocks such as "*ToDQ*" and ”*FromDQ*" can

be directly modeled. In the hand generated model in Section 2, the entire inverter had to be modeled using *dq*, which neglected the effects of converting to and from *dq*. Matlab code for Section 4 is found in Appendix B.

2. FEEDFORWARD DOUBLE FREQUENCY ELIMINATION IN TWO-STAGE SINGLE-PHASE INVERTER

2.1. INTRODUCTION

As residential PV installations increase, the need to improve both reliability and power quality of grid tied MPPT converters increases as well. A popular topology for grid tied residential PV applications is a two-stage system. This system utilizes a boost type converter with an H-bridge inverter. With this topology, a double frequency component is present in the dc-link voltage, due to the rectification from the grid to the dc-link. This double frequency propagates through the inverter's controls and appears on the injected grid current as a third harmonic [57]. If this issue is to be mitigated, the controller must be properly designed to attenuate the double frequency ripple in the dc-link voltage control loop.

In [65], Brekken presented a system where the DC-link voltage feedback loop is responsible for rejecting the double frequency ripple. This requires a bandwidth of less than 12 Hz, which slows the overall system. The low bandwidth can lead to a large overshoot/undershoot in the dc-link capacitor voltage during perturbations to the system [66].

Another method involves subtracting a predicted dc-link double frequency ripple from the measured dc-link voltage, inside the controller. In [67], Mamarelis estimates the double frequency ripple using filtering. Steady state equations are then used to derive a perturbation signal. This signal can be added to the duty ratio of the MPPT converter, removing the double frequency component from the PV power waveforms, and thus improving the PV's power output. Because steady-state equations are used, this method

may be ineffective during system transients. The previous method also fails to address the output power quality of the inverter with regard to the double frequency. Finally, if this method were used in a weak grid setting (with a varying frequency), the filter may not accurately extract the ripple in v_{dc}.

The method proposed in this paper utilizes a single-phase dq-axis controller. This controller is similar to the controller described in [68]. The ac double frequency component of v_{dc} can be mathematically predicted using both the measured dq-axis inverter current and the commanded dq-axis inverter output voltage. This process is completed with small signal equations, which allows for the proposed method to work even while not operating at steady state. Additionally, no phase delays are added to the system due to filtering. By rejecting the ripple mathematically, as proposed in this study, only the dc component from the dc-link voltage is seen by the controller. As a result, a higher bandwidth voltage feedback loop can be utilized. This same dq-axis analysis is then applied to the MPPT converter. Here, a perturbation duty ratio is added to the output of the MPPT controller to compensate for the double frequency ripple on v_{dc}. The result is a PV output power waveform free of the double frequency ripple.

A state-space inverter model is presented in Section 2.2. In this configuration, a boost converter with MPPT capabilities is connected to a PV panel. The MPPT converter is connected to the H-bridge inverter with a dc-link capacitor between the two. The inverter is connected to the grid via an inductive filter. Experimental results were taken to demonstrate the standard operation of an inverter.

Section 2.3 introduces the feedforward method. This method eliminates the effects of the double frequency ripple on the dc-link. Experimental results are presented

to illustrate the effectiveness of the proposed feedforward control. Section 2.4 introduces a state-space model for the MPPT converter. The MPPT converter used is a boost converter utilizing perturb and observe (P&O) [69, 70]. Experimental results are presented using the given controller gains. Section 2.5 takes the feedforward concept introduced in Section 2.3 and applies it to the MPPT converter. Experimental results are presented to show the effectiveness of the proposed feedforward control. Results indicate that the proposed feedforward method eliminates the effects of the double frequency component of v_{dc} in a two-stage grid tied MPPT converter.

2.2. INVERTER ANALYSIS

The inverter used was a single phase H-bridge converter (see Figure 2.1). The control objective of the inverter was to maintain the dc-link voltage at 60 V. Both phase and frequency of the grid were tracked with a phase locked loop (PLL). The PLL generates two sets of sine and cosine signals: one set operates at the grid's frequency, and the other operates at the grid's double frequency.

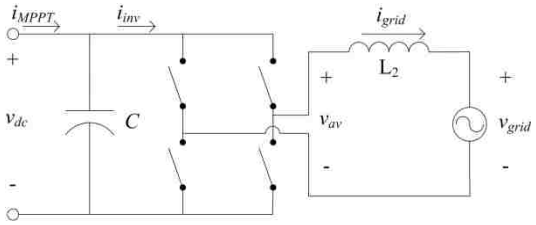

Figure 2.1. Inverter side of the two-stage converter.

2.2.1. DQ-Axis Control. The inverter was controlled with a single phase dq axis controller. The control objective of the inverter was to maintain a constant dc-link voltage while outputting a 60 Hz sinusoidal grid current. The output current was converted into a dq-axis synchronous reference frame using the signal delay method [46, 71, 72]. In this method, the signal to be converted is delayed one-quarter of a cycle to create a second signal orthogonal to the original signal. The conversion to the dq axis frame,

$$\begin{bmatrix} x_d \\ x_q \end{bmatrix} = \begin{bmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{bmatrix} \begin{bmatrix} x \\ x_{delay} \end{bmatrix}, \tag{5}$$

is used to convert a signal x to the rotating reference frame. The notation used in this paper is illustrated in Figure 2.2.

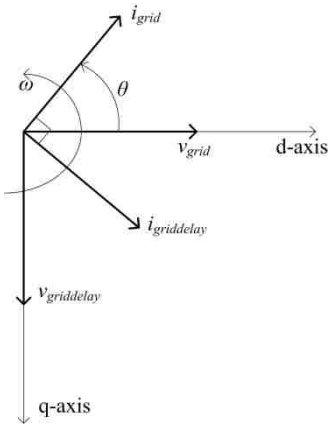

Figure 2.2. Phase diagram.

A block diagram for the inverter's control is presented in Figure 2.3. An outer voltage feedback loop utilizes a proportional-integral (PI) controller to regulate the dc-

link voltage. The inner current feedback loop regulates the inverter's output current with a PI controller. Cross-coupling elimination between the d and q axis currents was used on the inner current loop.

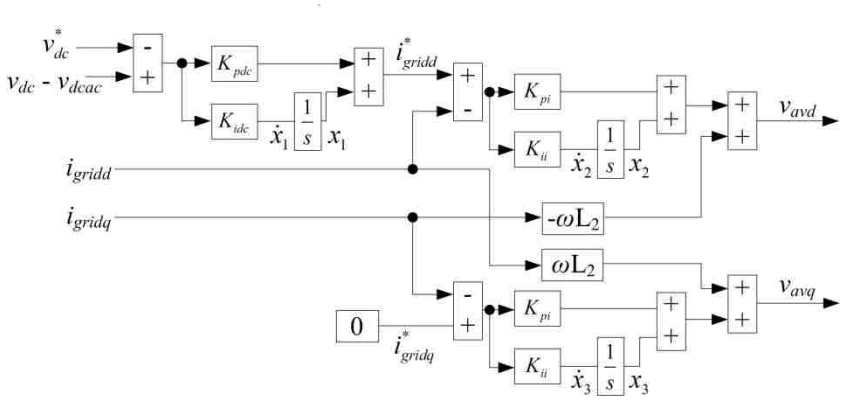

Figure 2.3. Inverter control diagram.

In the inverter controller, the commanded q-axis grid current, i^*_{gridq}, is set to zero to null the reactive power. i^*_{gridq} can later be modified with an additional control loop, to provide voltage support in a weak grid setting. In the dq reference frame, when either an inductor or a capacitor is present, a cross coupling exists between the d and q axes. The ωL_2 blocks provide decoupling between the d and q axis circuits, due to the inverter's output inductor. Decoupling these circuits simplifies the controller's design.

As a result of the rectifying action of the inverter, a double frequency ripple is present on v_{dc}, which is seen by the controller. This double frequency ripple increases in amplitude as the dc-link capacitance decreases. Higher gains on the v_{dc} control loop allow the inverter to respond quickly to changes on both v_{grid} and v_{dc}. These higher gains

also result in less attenuation on the double frequency ripple in v_{dc}. As a result, additional harmonics are injected into i_{grid}.

2.2.2. Inverter Model. A linear model was derived using equations in the synchronous reference frame to determine the best controller gains for use in this inverter. Assuming ideal switches in the H-bridge inverter, i_{inv} can be determined by using instantaneous power:

$$i_{inv} = \frac{v_{av} i_{grid}}{v_{dc}} \tag{6}$$

The 120 Hz component in (6) must be predicted. One method to extract the 120 Hz component in i_{inv} is to convert (6) to the synchronous reference frame. By rewriting v_{av} and i_{grid} into their dq components,

$$x = x_d \cos(\omega t) - x_q \sin(\omega t), \tag{7}$$

(6) becomes

$$i_{inv} = \frac{i_{L1d} v_{avd} + i_{L1q} v_{avq} + \cos(2\omega t)\left(i_{L1d} v_{avd} - i_{L1q} v_{avq}\right) - \sin(2\omega t)\left(i_{L1d} v_{avq} + i_{L1q} v_{avd}\right)}{2 v_{dc}}. \tag{8}$$

Both the sine and the cosine components of (8) are at twice the frequency of v_{grid}. Multiple reference frames are used in the controller: a reference frame operating at the frequency of v_{grid} and a reference frame operating at twice the frequency of v_{grid} [53, 54, 56-58].

The inverter current (8) can be separated into three terms:

- a dc current:

$$i_{invdc} = \frac{i_{gridd} v_{avd} + i_{gridq} v_{avq}}{2v_{dc}} \tag{9}$$

- a d-axis current at twice grid frequency:

$$i_{inv2d} = \frac{i_{gridd} v_{avd} - i_{gridq} v_{avq}}{2v_{dc}} \tag{10}$$

- a q-axis current at twice grid frequency:

$$i_{inv2q} = \frac{i_{gridd} v_{avq} + i_{gridq} v_{avd}}{2v_{dc}} \tag{11}$$

The effect of i_{inv2dq} is negligible on the system's stability. Only the dc component of i_{inv} will be considered for the plant model used to optimize controller gains. An equation for v_{dcdc} can be derived after (9) is linearized,

$$\dot{v}_{dcdc} = \frac{-1}{2V_{dc}} \left(I_{Ld} v_{avd} + I_{Lq} v_{avq} + V_{avd} i_{Ld} + V_{avq} i_{Lq} - \left(\frac{I_{Ld} V_{avd} + I_{Lq} V_{avq}}{V_{dc}} \right) v_{dcdc} \right). \tag{12}$$

The filter inductor dq currents can be derived from Figure 2.1,

$$\dot{i}_{L2d} = \frac{1}{L_2} v_{avd} - \frac{1}{L_2} v_{gridd} \cancel{+ \omega L_2 i_{L2q}} \tag{13}$$

$$\dot{i}_{L2q} = \frac{1}{L_2} v_{avq} - \frac{1}{L_2} v_{gridq} \cancel{- \omega L_2 i_{L2d}}, \tag{14}$$

and both v_{avd} and v_{avq} can be derived from Figure 2.3,

$$v_{avd} = K_{pi}\left[K_{pdc}\left(v_{dcdc} - v_{dc}^{*}\right) + x_1 - i_{Ld}\right] + x_2 - \omega L_2 i_{gridq} \tag{15}$$

$$v_{avq} = -K_{pi} i_{Lq} + x_3 + \omega L_2 i_{gridd} \,. \tag{16}$$

The x-ed out ωL_2 terms in (13-16) are removed by the cross coupling elimination in the controller. These terms are not included in the remainder of the controller design.

The integral control states can also be derived from Figure 2.3,

$$\dot{x}_1 = K_{idc}\left(v_{dcdc} - v_{dc}^{*}\right) \tag{17}$$

$$\dot{x}_2 = K_{ii}\left(K_{pdc}\left(v_{dcdc} - v_{dc}^{*}\right) + x_1 - i_{L2d}\right) \tag{18}$$

$$\dot{x}_3 = -K_{ii} i_{L2q} \,. \tag{19}$$

Rewriting (12-19) in state-space form, $\dot{x} = Ax + Bu$ yields

$$A = \begin{bmatrix} \frac{-K_{pdc}K_{pi}I_{Ld}}{2CV_{dc}} + \frac{I_{Ld}V_{avd} + I_{Lq}V_{avq}}{2CV_{dc}^2} & \frac{-V_{avd} + K_{pi}I_{L2d}}{2CV_{dc}} \\ \frac{K_{pdc}K_{pi}}{L_2} & \frac{-K_{pi}}{L_2} \\ 0 & 0 \\ K_{idc} & 0 \\ K_{pdc}K_{ii} & -K_{ii} \\ 0 & 0 \end{bmatrix} \cdots$$

$$\begin{bmatrix} \frac{K_{pi}I_{L2q} - V_{avq}}{2CV_{dc}} & \frac{-K_{pi}I_{L2d}}{2CV_{dc}} & \frac{-I_{L2d}}{2CV_{dc}} & \frac{-K_{pi}I_{L2q}}{2CV_{dc}} \\ 0 & \frac{K_{pi}}{L_2} & \frac{1}{L_2} & 0 \\ \frac{-K_{pi}}{L_2} & 0 & 0 & \frac{1}{L_2} \\ 0 & 0 & 0 & 0 \\ 0 & K_{ii} & 0 & 0 \\ -K_{ii} & 0 & 0 & 0 \end{bmatrix}, \tag{20}$$

$$x = \begin{bmatrix} v_{dcdc} \\ i_{L2d} \\ i_{L2q} \\ x_1 \\ x_2 \\ x_3 \end{bmatrix},\ B = \begin{bmatrix} \frac{K_{pi}K_{pdc}I_{L2d}}{2CV_{dc}} & 0 & 0 \\ \frac{-K_{pi}K_{pdc}}{L_2} & \frac{-1}{L_2} & 0 \\ 0 & 0 & \frac{-1}{L_2} \\ -K_{idc} & 0 & 0 \\ -K_{ii}K_{pdc} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},\ u = \begin{bmatrix} v_{dc}^{*} \\ v_{gridd} \\ v_{gridq} \end{bmatrix}. \quad (21)$$

The plant model in (20) and (21) describes a single phase H-bridge dq controlled inverter. PI control and an inductor output filter are utilized in the inverter. The values used in this study are listed in Table 2.1.

Table 2.1. Inverter Linearization Points.

Variable	Description	Value
I_{L2d}	L_2 d-axis steady state current	4 A
I_{L2q}	L_2 q-axis steady state current	0 A
V_{dc}	Dc-link steady state voltage	60 V
V_{avd}	Steady state d-axis inverter output	30 V
V_{avq}	Steady state q-axis inverter output	-0.1 V

This model can be used to optimize the controller gains K_{pdc}, K_{idc}, K_{pi}, *and* K_{ii}. Three sets of controller gains were chosen to demonstrate the effectiveness of the proposed method at various operating points and are listen in Table 2.2. These gains were chosen by setting a maximum integral gain, then finding the corresponding proportional gains which yield robust negative real eigenvalues.

Table 2.2. Inverter Gains.

	Case	Case	Case
K_{pdc}	0.3488	0.7189	1.001
K_{idc}	10	50	100
K_{pi}	2.509	4.407	5.398
K_{ii}	10	50	100

2.2.3. Inverter Experimental Results without Feedforward. To verify the proposed method, the single phase inverter was built and tested. The parameters in Table 2.1 were used in the experimental setup and the cases in Table 2.2 were used in the tests. Experimental results without the proposed feedforward are illustrated in Figure 2.4, for each test case. Increasing controller gains increases distortion. The THD of the three cases is 5.23%, 12.8%, and 22.2%.

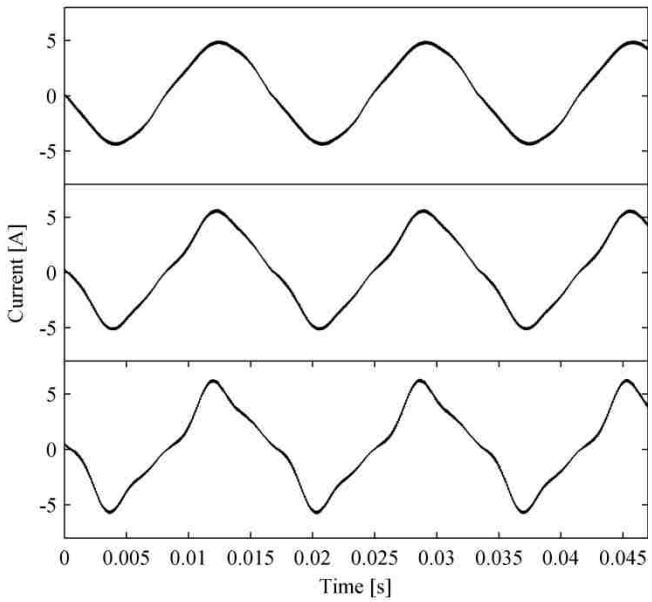

Figure 2.4. Experimental results for i_{grid}. Top: i_{grid} case #1. Middle: i_{grid} case #2. Bottom: i_{grid} case #3.

2.3. INVERTER DOUBLE FREQUENCY ELIMINATION

Few options are available to mitigate the effect of the double frequency ripple on v_{dc}. One option involves choosing smaller controller gains so that the double frequency term has enough attenuation. This option, however, will lead to a slower system response and a less robust controller. Another method is to include a bandstop filter into the v_{dc} control loop. This method may work for a strong grid case, when minimal fluctuations are present in the grid frequency. In a weak grid setting, however, there may be significant changes in the grid frequency. The proposed method uses information from the dq-axis controller to add a feedforward term to the sensed dc-link voltage. The 120 Hz ripple normally seen by the controller is thus removed.

The 120 Hz ripple current from the inverter (which causes the 120 Hz ripple in the dc-link voltage) was solved for in (10) and (11). Using these equations, the voltage ripple can be estimated. The dc-link voltage ripple, due to i_{inv}, can be found in the synchronous reference frame using

$$v_{dc2d} = -\frac{i_{inv2q}}{2\omega C} - \frac{d}{dt}\frac{v_{dc2q}}{2\omega} \tag{22}$$

and

$$v_{dc2q} = \frac{i_{inv2d}}{2\omega C} + \frac{d}{dt}\frac{v_{dc2d}}{2\omega}\,. \tag{23}$$

By substituting (10) and (11) into (22) and (23), the 120 Hz voltage ripple in v_{dc} (due to the inverter) is found with

$$v_{dcac} = \frac{-i_{gridd}v_{avq} - i_{gridq}v_{avd}}{8\omega C v_{dc}}\cos(2\omega t) - \frac{i_{gridd}v_{avd} - i_{gridq}v_{avq}}{8\omega C v_{dc}}\sin(2\omega t)\,. \tag{24}$$

This equation assumes small changes in both v_{dc2q} and v_{dc2d} and neglects the derivative term in (22) and (23). The feedforward term is v_{dcac}. This term can be subtracted from the measured v_{dc} inside the controller. This feedforward method eliminates the 120 Hz ripple normally seen by the controller, without the need of small controller gains or a filter.

The proposed feedforward method was carried out experimentally. Experimental results using feedforward elimination are illustrated in Figure 2.5. The results using feedforward in Figure 2.5 suggest a significant improvement over the results without feedforward in Figure 2.4. The feedforward elimination reduced the impact of increased controller gains on the harmonic content of i_{grid}. The THD for the three cases is now 4.83%, 5.97%, and 7.69%.

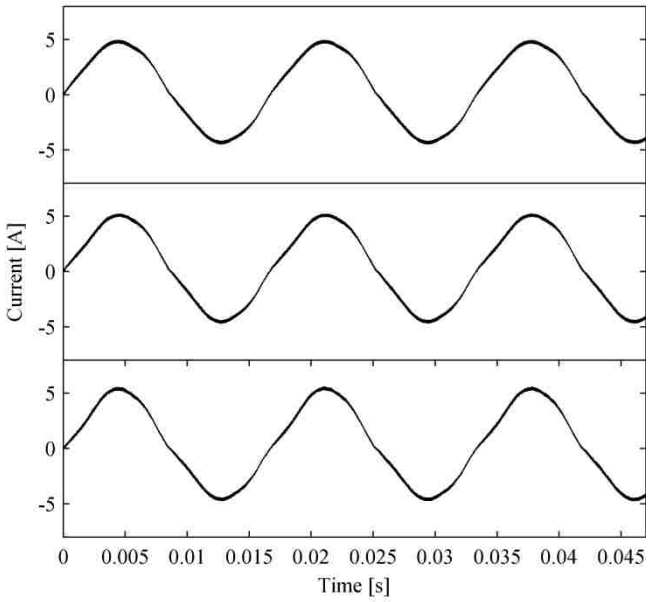

Figure 2.5. Experimental results for i_{grid}. Top: i_{grid} Case #1. Middle: i_{grid} Case #2. Bottom: i_{grid} Case #3.

2.4. MPPT ANALYSIS

A boost converter connects the solar panel to the dc-link capacitor. The control objective of the boost converter is to perform MPPT on a solar panel. The boost converter diagram is illustrated in Figure 2.6. A resistor (R_{in}) was placed in series with v_{in} to approximate the behavior of a solar panel near its MPP. R_{in} was used for modeling purposes only. It was not present in the experimental circuit.

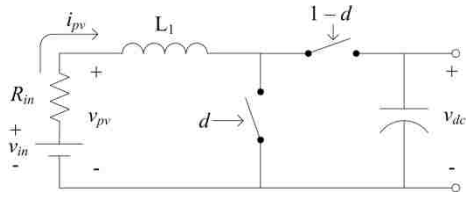

Figure 2.6. MPPT converter.

The control diagram for the MPPT converter is illustrated in Figure 2.7. P&O was the MPPT algorithm used. This algorithm was chosen because of its ease of implementation and its common use [69, 70].

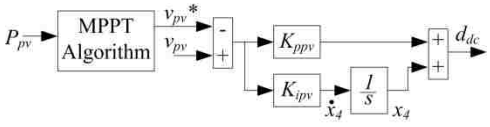

Figure 2.7. MPPT converter control.

System equations can be derived from Figure 2.6 and Figure 2.7:

$$\dot{i}_{L1} = \frac{d\left(-i_{L1}R_{in}\right) - \left(1-d\right)v_{dc} + v_{pv}}{L_1} \tag{25}$$

$$\dot{v}_{dc} = \frac{\left(1-d\right)i_{L1}}{C} \tag{26}$$

$$\dot{x}_4 = K_{ipv}\left(v_{in} - R_{in}i_{L1} - v_{pv}^{*}\right) \tag{27}$$

$$d = K_{ppv}\left(v_{in} - R_{in}i_{L1} - v_{pv}^{*}\right) + x_4 \tag{28}$$

These system equations can be linearized and put in the form $\dot{x} = Ax + Bu$,

$$A = \begin{bmatrix} \frac{-2K_{ppv}I_{L1}^2R_{in}^3 - K_{ppv}R_{in}V_{dc} - K_{ppv}R_{in}^3I_{L1}^2}{L_1} & \frac{-1 - I_{L1}K_{ppv}R_{in}}{L_1} & \frac{1+V_{dc}}{L_1} \\ \frac{1+2I_{L1}K_{ppv}R_{in}}{C} & 0 & \frac{-I_{L1}}{C} \\ K_{ipv}R_{in} & 0 & 0 \end{bmatrix} \tag{29}$$

$$x = \begin{bmatrix} i_{L1} \\ v_{dc} \\ x_4 \end{bmatrix}, B = \begin{bmatrix} \frac{1 - K_{ppv}I_{L1}R_{in} + K_{ppv}V_{dc}}{L_1} & \frac{K_{ppv}I_{L1}R_{in} - K_{ppv}V_{dc}}{L_1} & 0 \\ \frac{-I_{L1}K_{ppv}}{C} & \frac{I_{L1}K_{ppv}}{C} & \frac{-1}{C} \\ K_{ipv} & -K_{ipv} & 0 \end{bmatrix}, u = \begin{bmatrix} v_{in} \\ v_{pv}^{*} \\ i_{inv} \end{bmatrix}. \tag{30}$$

The plant model in (29) and (30) describes a boost converter that controls its input voltage (v_{pv}) via a PI loop. This model can be used to optimize the controller gains K_{ppv} and K_{ipv}. The values used in this study are I_{L1}=5 A and V_{dc}=60 V.

Using this model, the controller gains can be chosen. Just as in Section 2.2, three sets of controller gains are chosen. These are chosen by setting a maximum integral gain, then setting the proportional gain to a value which results in robust negative real

eigenvalues. The gains used are listed in Table 2.3. Smaller integral gains were used in this section, due to the eigenvalues' high natural frequencies, compared to Section 2.2.

Table 2.3. MPPT Gains.

	Case #1	Case #2	Case #3
K_{ppv}	0.003782	0.004736	0.006117
K_{ipv}	10	25	40.91

The cases listed in Table 2.3 were used to test. Figure 2.8 illustrates the experimental results. In Case 1, a large 120 Hz ripple is present. In Case 2, the ripple is reduced somewhat, and in Case 3 the ripple is further reduced. This demonstrates that an increase in K_{ipvmax} will reduce the 120 Hz ripple effect. Since results were taken using a PV panel in the field, the MPPT operated at slightly different points for each measurement. This explains the small offset in Figure 2.8 i_{L1}, Case 1.

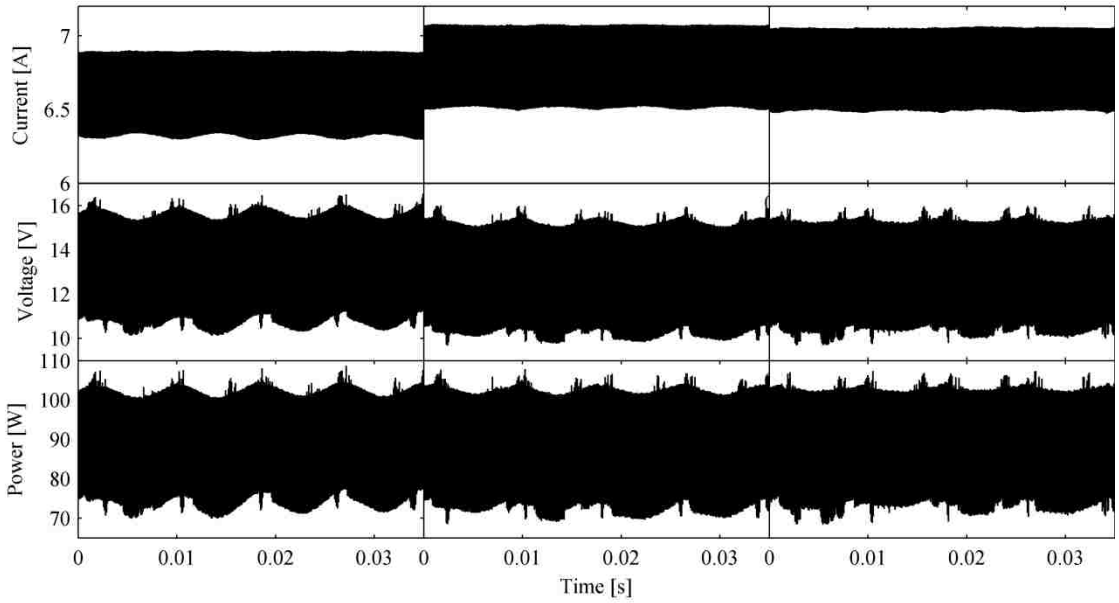

Figure 2.8. Exp results for i_{L1}, v_{pv}, P_{pv}. Left: Case #1. Middle: Case #2. Right: Case #3.

2.5. MPPT DOUBLE FREQUENCY ELIMINATION

There are a few options to mitigate the effect of the double frequency ripple on the PV panel. One is to choose larger controller gains, so that the double frequency term has enough attenuation. For most cases, this is an acceptable answer. If a very small dc-link capacitor is used, there may be a large enough 120 Hz ripple to still affect the PV panel performance. This is one scenario where the feedforward method introduced in Section 2.3 can be applied to the MPPT. The proposed method from Section 2.3's goal was to remove the 120 Hz ripple as seen by the controller, which results in a smaller 3rd harmonic term on i_{grid}. The proposed method in this section's goal is to remove the 120 Hz voltage and current ripple seen at the PV panel's terminals. The proposed method uses information from the dq-axis controller to add a feedforward term to the output duty ratio of the MPPT controller. This will eliminate the 120 Hz ripple at the PV panel.

2.5.1. MPPT 120 Hz Ripple Elimination. One solution to eliminate this power ripple is to add a perturbation signal (d_{ac}) to the MPPT generated duty ratio (d_{dc}, see Figure 2.7). Ideally, this perturbation signal will offset the 120 Hz dc-link ripple and result in a dc input power, with no 120 Hz component. The MPPT algorithm can then track the desired power level, without also compensating for the 120 Hz component of v_{dc} that propagates through the boost converter and into the PV panel. From Figure 2.6, average model equations can be derived (assuming CCM operation):

$$v_{L1} = v_{pv} - (1-d)v_{dc} \tag{31}$$

$$i_{pv} = \frac{1}{L_1}\int v_{L1}dt \tag{32}$$

It has already been established in (22) and (23) that v_{dc} can be partitioned into a dc component and an ac component. Assuming d_{dc} can also be split into a dc component and an ac component at twice the grid frequency:

$$d = D + d_{2dq}e^{j2\omega t} \tag{33}$$

i_{pv} can also be partitioned into a dc component:

$$i_{pvdc} = \frac{1}{L_1}\int\left[v_{pv} - (1-d_{dc})v_{dcdc}\right]dt \tag{34}$$

and an ac component at twice the grid frequency:

$$i_{pv2dq}e^{j2\omega t} = \frac{1}{L_1}\int\left(v_{dcdc}d_{2dq}e^{j2\omega t} - v_{dc2dq}e^{j2\omega t} + d_{dc}v_{dc2dq}e^{j2\omega t} + d_{2dq}v_{dc2dq}e^{j4\omega t}\right)dt \tag{35}$$

Differentiating both sides, and setting i_{pv2dq} and $\dot{i}_{pv2dq}$ to 0, (35) becomes

$$0 = v_{dcdc}d_{2dq} - v_{dc2dq} + d_{dc}v_{dc2dq} + d_{2dq}v_{dc2dq}e^{j2\omega t}. \tag{36}$$

Solving for d_{2dq} yields

$$d_{2dq} = \frac{v_{dc2dq}}{v_{dcdc} + v_{dc2dq} e^{j2\omega t}} (1 - d_{dc}) = \frac{v_{dc2dq}}{v_{dc}} (1 - d_{dc}) \tag{37}$$

$$d_{2d} = \frac{v_{dc2d}}{v_{dc}} (1 - d_{dc}) \tag{38}$$

$$d_{2q} = \frac{v_{dc2q}}{v_{dc}} (1 - d_{dc}), \tag{39}$$

where d_{dc} is the output of the MPPT algorithm.

From (38) and (39), the duty ratio, d, passed to the boost converter is,

$$d = d_{dc} + \frac{v_{dc2d}}{v_{dc}} (1 - d_{dc}) \cos(2\omega t) - \frac{v_{dc2q}}{v_{dc}} (1 - d_{dc}) \sin(2\omega t). \tag{40}$$

Equation (40) shows the feedforward equation proposed for a boost MPPT converter.

2.5.2. MPPT 120 Hz Ripple Elimination Experimental Results. Figure 2.9 illustrates the experimental results utilizing the proposed feedforward method. These results illustrate the effectiveness of the proposed feedforward method. The proposed feedforward elimination greatly reduced the 120 Hz voltage and current ripple seen at the PV panel.

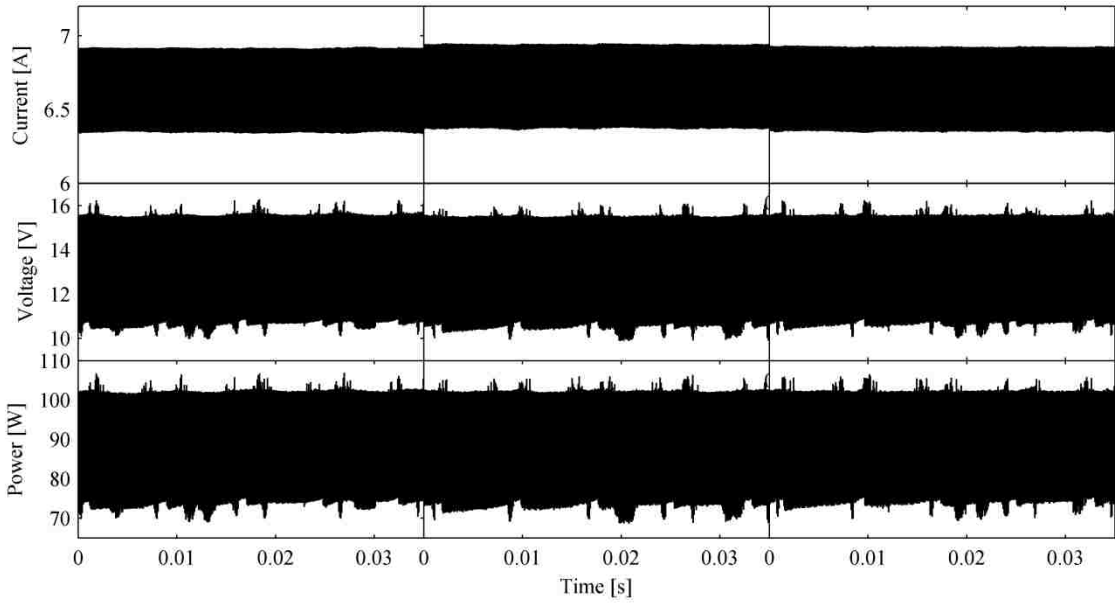

Figure 2.9. Experimental results for i_{L1}, v_{pv}, P_{pv}. Left: Case #1. Middle: Case #2. Right: Case #3.

2.6. CONCLUSIONS

The proposed feedforward method eliminates the second harmonic seen by the inverter controller while measuring v_{dc}. If this second harmonic is not accounted for, a significant 3rd harmonic on i_{grid}, is produced (see Figure 2.4). The proposed feedforward method is effective in removing the 3rd harmonic on i_{grid} (see Figure 2.5). Case 1 THD was reduced from 5.23% to 4.83%. Case 2 THD was reduced from 12.8% to 5.97%. Case 3 THD was reduced from 22.2% to 7.69%.

This same feedforward concept was applied to the MPPT converter. Oscillations will occur on the PV panel terminals if the 120 Hz dc-link voltage ripple remains unchecked (see Figure 2.8). The proposed feedforward method removes this double frequency ripple (see Figure 2.9). Thus, a designer does not need to worry about double frequency oscillations in a two-stage single phase grid tied inverter setup.

The double frequency elimination from (40) can be used with other MPPT algorithms; it is not limited to the P&O MPPT algorithm used in this paper. For example, hill-climbing is similar to P&O. The difference is that hill-climbing perturbs the duty ratio, as opposed to perturbing the commanded voltage. The effects of the feedforward control proposed would be greater with the hill climbing algorithm. This is due to the lack of an extra voltage control loop to help attenuate the 120 Hz ripple. The method proposed in this study is a feedforward technique that can simply be added to any MPPT algorithm's output.

The feedforward method presented is not limited to PV applications. It can also be applied to single phase electrochemical energy storage applications (e.g., uninterruptible power supplies) to remove the low frequency component seen at the battery terminals. This would allow for an increase in lifetime and efficiency of electrochemical energy storage applications [66]. Similarly, the method could be used for fuel cell based systems.

3. FOURIER SERIES ANALYSIS OF A SINGLE PHASE INVERTER

3.1. INTRODUCTION

The interactions between the dc-link and ac side of a single phase H-bridge inverter are nonlinear. This is due to the fundamental of the grid voltage generating a dc voltage plus a second harmonic voltage on the dc-link. This second harmonic in turn results in a third harmonic on the grid current, which in turn generates more harmonics. An excitation at a specific frequency on the ac side will result in minimal output on the dc-link at the excitation frequency, but significant output at the excitation frequency harmonics (this also applies to applying an excitation to the dc-link and observing the grid side). Due to this nonlinearity, predicting these harmonics is challenging. In this study, a method of generating a large-signal time-invariant model of a single phase H-bridge inverter is presented. The model is capable of accurately predicting the harmonics of the steady state output.

Modeling of harmonics in an inverter is challenging, due to the frequency transfer properties described above. By taking the Fourier series expansion of each input and state, the model can be converted to a periodically time varying system. Using the concept of harmonic balance [62-64], the periodically time varying terms are eliminated by equating like harmonics. This leaves behind a time invariant system. The challenge is in deriving the time invariant system.

Many techniques exist for analyzing circuits in the frequency domain. Harmonic transfer functions (HTF) were first presented by Wereley [62, 63]. HTF's take a linear periodically time varying system (LPTV) and convert it to a linear time invariant (LTI) system of Fourier coefficients. The periodically varying portion of each signal is

eliminated by utilizing harmonic balance. Computing the HTF is nontrivial, due to the HTF being infinite dimensional.

Another method which utilizes Fourier coefficients for system modeling is generalized state space averaging (GSSA) [73]. This method was designed to allow for the modeling of power electronic converters, while taking into account a finite number of harmonics generated by switching. For fast switching PWM circuits with low ripple, the GSSA would result in a standard state-space averaged model.

State space averaging requires that the switching frequency is much higher than the system's resonant frequencies. GSSA allows for the switching frequency to be near a resonant frequency [74]. GSSA is also commonly used in low frequency converters, where the fundamental frequency of a state is the converter's switching frequency [75, 76].

Modeling the effects of pulsewidth modulation (PWM) is challenging, due to the nonlinear properties of PWM [77]. Many papers have been recently published on finding the Fourier coefficients of a power converter's PWM waveform [78-80]. Work has also been done on finding the Fourier coefficients of an N-phase rectifier by Sandberg [64] and of a hysteresis controlled inverter by Albanna [81]. For this study, a high switching frequency is assumed, and the effects of PWM are neglected.

In all of the methods described above, there is a challenge in generating the harmonic equations. A popular method is to use describing functions (DF) and extended describing functions (EDF) [82]. The limitation with DF's and EDF's is that only the fundamental harmonic of each state is used, which only allows for minimal modeling of the frequency transfer properties. In this study, a set of algorithms are presented, which

generate a set of harmonic equations, given a standard large-signal model. The frequency transfer properties for the first N_{max} harmonics (an arbitrary finite set determined by the designer) are preserved using the presented algorithms.

These algorithms allow for multiple harmonics of the fundamental grid frequency for each input and state. This allows for exponentially modulated periodic (EMP) [62] inputs to the system. EMP signals are common in grid tied systems, where the grid input voltage is a fundamental frequency (60 Hz in the US) plus some odd harmonics. Frequency transfer properties between each harmonic are also preserved in the presented algorithms.

The goal of this study is to present a large-signal model of a single-phase H-bridge inverter. This model will be used to generate the steady state harmonic output of the converter. Comparison of the predicted steady state grid current and dc-link voltage between the mathematical model, simulation, and experimental results are presented. Since the interactions between harmonics are preserved, the predicted periodic steady state output of the presented mathematical model is highly accurate.

Section 3.2 presents the single phase inverter used in this study and also presents a standard set of state equations. Algorithms are presented to computationally generate the Fourier series representation of each operation required for the inverter used in this study. Section 3.3 demonstrates how the algorithms and state equations from Section 3.2 can be used to generate a Fourier series expanded large-signal model. In Section 3.4, an iterative method is proposed to solve for the steady-state operating point of the expanded large-signal model. Section 3.5 contains the numeric results of the expanded large-signal model and compares them with a Simulink simulation.

3.2. FOURIER SERIES EXPANSION OF VARIABLES

Every periodic signal can be represented by

$$u = u_{dc} + \sum_{k=1}^{\infty}\left[u_{dk\omega}\cos(\omega kt) - u_{qk\omega}\sin(\omega kt)\right]. \tag{41}$$

This the real form of a Fourier series expansion. Each state, input, and output can be separated into a dc portion (u_{dc}), a sum of d-axis portions ($u_d cos(\omega kt)$), and a sum of q-axis portions ($-u_q sin(\omega kt)$). This requires $2(N_{max}+1)M$ equations, where N_{max} is the number of harmonics to be analyzed and M is the number of states in the system. The truncated real form Fourier series is used in this study for each variable.

Using non-harmonic analysis, interactions between the different input and output frequencies are lost. In other words, an input with a specific frequency can only generate an output at that frequency with possibly a different phase and magnitude [62]. In a single phase inverter, however, it is well known that a 60 Hz grid voltage will yield a grid current that contains 60 Hz and odd harmonics. One challenge in generating harmonic functions is in determining the various frequency interactions.

The process to generate a time invariant large-signal model is illustrated in Figure 3.1. First, the user must derive the system's state equations. This is demonstrated in Section 3.2.1. Next, these equations are programmed into a computer. This is demonstrated in Section 3.3.1. Using the algorithms presented in Section 3.2.2, the computer generates the harmonic state equations. This step of generating the harmonic state equations only needs to be performed once.

To predict the steady state harmonic output of the system, the user must give the computer the magnitude and phase of each harmonic for each input. For each input, u_{dc}, $\mathbf{u_{dk\omega}}$, and $\mathbf{u_{qk\omega}}$ from (41) must be programmed into the computer. After receiving the

input and loading the saved harmonic state equations, the computer generates the predicted harmonic steady-state operating point of the system.

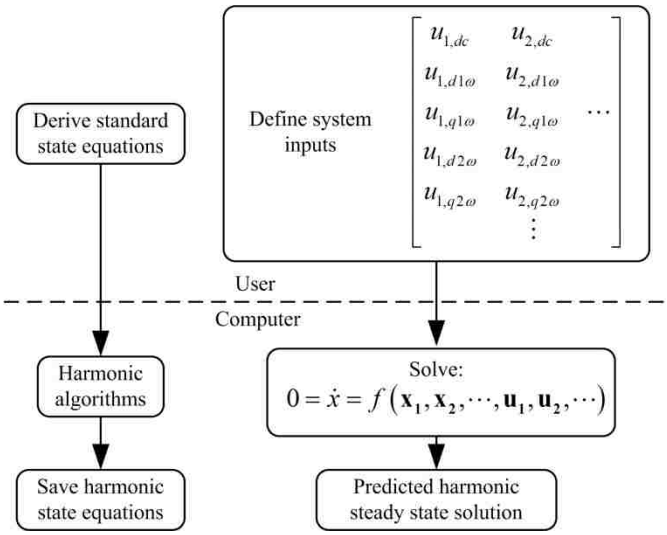

Figure 3.1. Process to predict a system's harmonic steady state solution.

3.2.1. Generation of Standard State Equations. One advantage of the harmonic equation generation proposed in this study is that once the standard state equations are generated by the user, the harmonic state equations can be generated computationally. The single phase H-bridge inverter to be modeled is illustrated in Figure 3.2. A similar circuit is analyzed by Gaviria [83], where a generalized state space model is generated for a full-bridge single-phase inverter. Gaviria models only the fundamental harmonic of each state, where this study models multiple harmonics of each state.

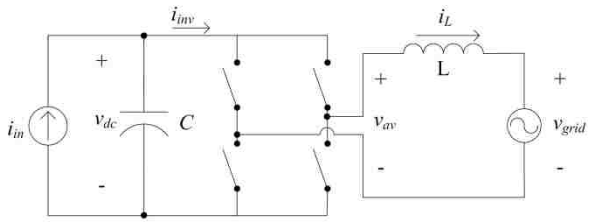

Figure 3.2. Inverter diagram.

The inverter is controlled using the single phase dq axis controller illustrated in Figure 3.3. The control objective of the inverter is to maintain a constant dc-link voltage while outputting a 60 Hz sinusoidal grid current. A PI loop is used to control the dc-link voltage, where K_{pdc} and K_{idc} are the proportional and integral controller gains. D-axis and q-axis currents are controlled using PI loops, where K_{pdq} and K_{idq} are the proportional and integral controller gains. Cross coupling elimination between the d and q-axis currents is utilized (ωL blocks).

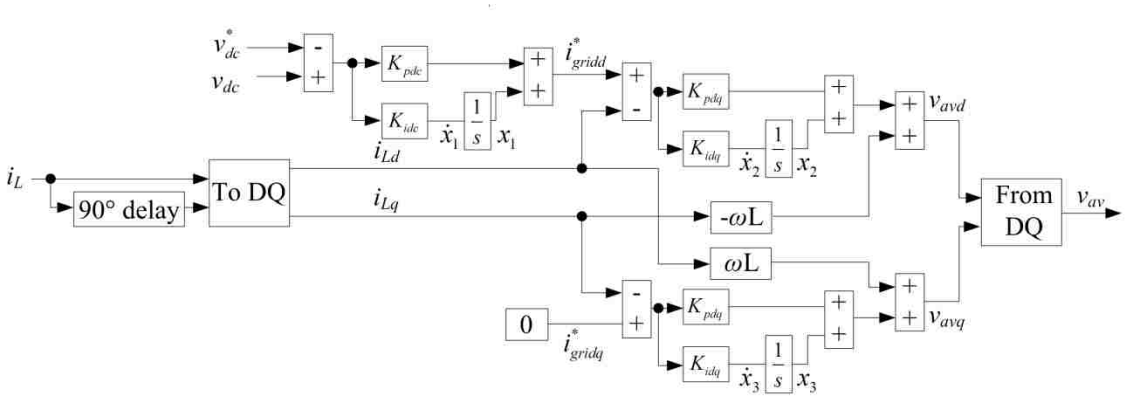

Figure 3.3. Inverter controller.

Equations can be generated using Figure 3.2 and Figure 3.3. An average model is assumed, where v_{av} is equal to the commanded v_{av} from the controller. The current going into the H-bridge from the dc-link is

$$i_{inv} = \frac{v_{av} i_L}{v_{dc}}. \tag{42}$$

Here are the system equations:

$$v_{avd} = x_1 + K_{pdq}\left(x_3 + K_{pdc}\left[v_{dc} - v_{dc}^*\right] - i_{Ld}\right) - \omega L i_{Lq} \tag{43}$$

$$v_{avq} = x_2 - K_{pdq} i_{Lq} + \omega L i_{Ld} \tag{44}$$

Utilizing Figure 3.2, Figure 3.3, (42), (43), and (44), the system's state equations can be determined:

$$\dot{v}_{dc} = \frac{1}{C}\left(i_{in} - i_{inv}\right) \tag{45}$$

$$\dot{i}_L = \frac{1}{L}\left(v_{av} - v_{grid}\right) \tag{46}$$

$$\dot{x}_1 = K_{idq}\left(x_3 + K_{pdc}\left[v_{dc} - v_{dc}^*\right] - i_{Ld}\right) \tag{47}$$

$$\dot{x}_2 = -K_{idq} i_{Lq} \tag{48}$$

$$\dot{x}_3 = K_{idc}\left(v_{dc} - v_{dc}^*\right) \tag{49}$$

3.2.2. Algorithm Generation. Notice the "90° delay", "To DQ" and "From DQ" blocks from Figure 3.3 do not appear in the state equations. Equations representing these blocks require the use of harmonic functions, since these blocks result in phase and frequency transfer. The effects of these blocks on the state equations are computationally determined and are transparent to the user. Algorithms for these computational

calculations are presented in the following sections; multiplication, inversion, delay 90°, "To DQ", "From DQ", and differentiation are shown.

3.2.2.1 Multiplication. Finding the Fourier series terms at the ℓth harmonic has been previously found [73, 74]. It is shown here because of the different notation used in this study. When multiplying two signals (*a* and *b*) represented as a Fourier series,

$$\begin{aligned}(ab) = &\left(a_{dc} + \sum_{k=1}^{k\le N_{\max}} \left[a_{dk\omega}\cos(\omega kt) - a_{qk\omega}\sin(\omega kt)\right]\right)\times \\ &\left(b_{dc} + \sum_{k=1}^{k\le N_{\max}} \left[b_{dk\omega}\cos(\omega kt) - b_{qk\omega}\sin(\omega kt)\right]\right)\end{aligned}, \tag{50}$$

the d-axis amplitude and q-axis amplitude need to be found, at each frequency. The dc amplitude is

$$(ab)_{dc} = a_{dc}b_{dc} + \sum_{k=1}^{k\le N_{\max}} \left[\frac{a_{dk\omega}b_{dk\omega}}{2} + \frac{a_{qk\omega}b_{qk\omega}}{2}\right]. \tag{51}$$

The d-axis ℓth harmonic amplitude is

$$\begin{aligned}(ab)_{d\ell\omega} = &a_{d\ell\omega}b_{dc} + a_{dc}b_{d\ell\omega} + \frac{1}{2}\sum_{k=1}^{k<\ell}\left[a_{d(\ell-k)\omega}b_{dk\omega} - a_{q(\ell-k)\omega}b_{qk\omega}\right] + \\ &\frac{1}{2}\sum_{k=\ell+1}^{k\le N_{\max}}\left[a_{dk\omega}b_{d(k-\ell)\omega} + a_{qk\omega}b_{q(k-\ell)\omega} + a_{d(k-\ell)\omega}b_{dk\omega} + a_{q(k-\ell)\omega}b_{qk\omega}\right]\end{aligned} \tag{52}$$

and the q-axis ℓth harmonic amplitude is

$$\begin{aligned}(ab)_{q\ell\omega} = &a_{q\ell\omega}b_{dc} + a_{dc}b_{q\ell\omega} + \frac{1}{2}\sum_{k=1}^{k<\ell}\left[a_{qk\omega}b_{d(\ell-k)\omega} + a_{dk\omega}b_{q(\ell-k)\omega}\right] + \\ &\frac{1}{2}\sum_{k=\ell+1}^{k\le N_{\max}}\left[-a_{dk\omega}b_{q(k-\ell)\omega} + a_{qk\omega}b_{d(k-\ell)\omega} + a_{d(k-\ell)\omega}b_{qk\omega} - a_{q(k-\ell)\omega}b_{dk\omega}\right]\end{aligned}. \tag{53}$$

3.2.2.2 Inversion. Inversion is not as straightforward as multiplication. Assume the signal *a* is to be inverted and the solution is represented by *b*. If *a* is a periodic signal, then it can be represented as a sum of sinusoids. Assuming $a_{dc} >> |a_{ac}|$, then *b* can be approximated by a finite sum of sinusoids:

$$\frac{1}{a_{dc}+\sum_{k=1}^{N_{max}}\left[a_{dk\omega}\cos\left(\omega kt\right)-a_{qk\omega}\sin\left(\omega kt\right)\right]}\approx \frac{}{b_{dc}+\sum_{k=1}^{N_{max}}\left[b_{dk\omega}\cos\left(\omega kt\right)-b_{qk\omega}\sin\left(\omega kt\right)\right]} \quad . \tag{54}$$

Multiplying both sides by *a* yields

$$1\approx\left(a_{dc}+\sum_{k=1}^{N_{max}}\left[a_{dk\omega}\cos\left(\omega kt\right)-a_{qk\omega}\sin\left(\omega kt\right)\right]\right)\times \left(b_{dc}+\sum_{k=1}^{N_{max}}\left[b_{dk\omega}\cos\left(\omega kt\right)-b_{qk\omega}\sin\left(\omega kt\right)\right]\right) \quad , \tag{55}$$

which was examined in the previous section. The dc component of the product in (55) must be equal to 1. All d-axis and q-axis harmonic components must be equal to 0. This gives a set of ($2N_{max}$+1) equations with ($2N_{max}$+1) unknowns.

To solve for *b* programmatically, a coefficient matrix ($\mathbf{A_{coeff}}$) is be generated. In the coefficient matrix, each component of *b* is an unknown. Using (51), (52), and (53), the coefficients of *b* can be determined in terms of *a* for each harmonic. Placing the augmented $\mathbf{A_{aug}}$ matrix,

$$\begin{array}{c} \quad\; b_{dc}\;\; b_{d1\omega}\quad\; b_{q1\omega}\quad\cdots \\ \quad\;\downarrow\;\;\downarrow\quad\;\;\downarrow \\ \begin{array}{r} dc\rightarrow \\ 1\omega_d\rightarrow \\ 1\omega_q\rightarrow \\ \vdots \end{array} \end{array} \left[\begin{array}{cccc|c} \square & \square & \square & & 1 \\ \square & \square & \square & \cdots & 0 \\ \square & \square & \square & & \vdots \\ & \vdots & & \ddots & 0 \end{array}\right], \tag{56}$$

$$\mathbf{A_{aug}}=$$

in reduced row echelon form will result in the last column containing the desired terms for *b*. The Matlab® code for generating the harmonic inverse is in the Appendix.

3.2.2.3 Delay 90°. The delay block does not dealy magnitude of the signal, rather it causes a 90° phase shift by swapping and negating various d-axis and q-axis terms. If the harmonic being analyzed is odd, then the d-axis and q-axis values are swapped. If the harmonic being analyzed is even, then the d-axis and q-axis values are not swapped. To determine the sign of each element, each d-axis and q-axis value is multiplied by its corresponding value in Table 3.1. For values greater than the 4th harmonic, the values loop back around (i.e. the 5th harmonic uses 1st harmonic values, the 6th harmonic uses 2nd harmonic values, etc.).

Table 3.1. DQ multiplier for delay 90°.

	Harmonic			
Harmonic	1st	2nd	3rd	4th
d-axis:	-1	-1	1	1
q-axis:	1	-1	-1	1

3.2.2.4 To DQ. The To DQ block is a single phase DQ conversion, utilizing a 90° delay. The conversion used is

$$\begin{bmatrix} y_d \\ y_q \end{bmatrix} = \begin{bmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{bmatrix} \begin{bmatrix} x \\ x_{delay} \end{bmatrix}. \tag{57}$$

The dc amplitude at the d-axis and q-axis output is

$$\left(y_d\right)_{dc} = \frac{x_{d1\omega}}{2} - \frac{x_{delayq1\omega}}{2} \tag{58}$$

and

$$\left(y_q\right)_{dc} = \frac{x_{delayd1\omega}}{2} + \frac{x_{q1\omega}}{2}. \tag{59}$$

The d-axis and q-axis amplitude of y_d at the ℓth harmonic is

$$\left(y_d\right)_{d\ell\omega} = \begin{cases} x_{dc} + \frac{x_{d(\ell+1)\omega}}{2} - \frac{x_{delayq(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \frac{x_{d(\ell-1)\omega}}{2} + \frac{x_{delayq(\ell-1)\omega}}{2} + \frac{x_{d(\ell+1)\omega}}{2} - \frac{x_{delayq(\ell+1)\omega}}{2} & \text{otherwise} \end{cases} \tag{60}$$

and

$$\left(y_d\right)_{q\ell\omega} = \begin{cases} -x_{delaydc} + \frac{x_{delayd(\ell+1)\omega}}{2} + \frac{x_{q(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ -\frac{x_{delayd(\ell-1)\omega}}{2} + \frac{x_{q(\ell-1)\omega}}{2} + \frac{x_{delayd(\ell+1)\omega}}{2} + \frac{x_{q(\ell+1)\omega}}{2} & \text{otherwise} \end{cases}. \tag{61}$$

The d-axis and q-axis amplitude of y_q at the ℓth harmonic is

$$\left(y_q\right)_{d\ell\omega} = \begin{cases} x_{delaydc} + \frac{x_{delayd(\ell+1)\omega}}{2} + \frac{x_{q(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \frac{x_{delayd(\ell-1)\omega}}{2} - \frac{x_{q(\ell-1)\omega}}{2} + \frac{x_{delayd(\ell+1)\omega}}{2} + \frac{x_{q(\ell+1)\omega}}{2} & \text{otherwise} \end{cases} \tag{62}$$

and

$$\left(y_q\right)_{q\ell\omega} = \begin{cases} x_{dc} - \frac{x_{d(\ell+1)\omega}}{2} + \frac{x_{delayq(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \frac{x_{d(\ell-1)\omega}}{2} + \frac{x_{q(\ell-1)\omega}}{2} - \frac{x_{d(\ell+1)\omega}}{2} + \frac{x_{delayq(\ell+1)\omega}}{2} & \text{otherwise} \end{cases}. \tag{63}$$

The DQ sections can be confusing, due to equations for d-axis and q-axis values being broken up into a Fourier series representation. Recall that every variable in this study is represented as a Fourier series, even the d-axis and q-axis variables in Figure 3.3. This enables modeling controller concepts (like delay, to DQ, and from DQ) that cannot be conventionally modeled.

3.2.2.5 From DQ. Conversion from DQ is simple:

$$y = x_d \cos(\omega t) - x_q \sin(\omega t). \tag{64}$$

The dc component of *y* is

$$y_{dc} = \frac{(x_d)_{d1\omega}}{2} + \frac{(x_q)_{q1\omega}}{2} \tag{65}$$

The d-axis and q-axis amplitude of *y* at the ℓth harmonic is

$$y_d = \begin{cases} (x_d)_{dc} + \dfrac{(x_d)_{d(\ell+1)\omega}}{2} + \dfrac{(x_q)_{q(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \dfrac{(x_d)_{d(\ell-1)\omega}}{2} - \dfrac{(x_q)_{q(\ell-1)\omega}}{2} + \dfrac{(x_d)_{d(\ell+1)\omega}}{2} + \dfrac{(x_q)_{q(\ell+1)\omega}}{2} & \text{otherwise} \end{cases} \tag{66}$$

and

$$y_q = \begin{cases} -(x_q)_{dc} + \dfrac{(x_d)_{q(\ell+1)\omega}}{2} - \dfrac{(x_q)_{d(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \dfrac{(x_d)_{q(\ell-1)\omega}}{2} + \dfrac{(x_q)_{d(\ell-1)\omega}}{2} + \dfrac{(x_d)_{q(\ell+1)\omega}}{2} - \dfrac{(x_q)_{d(\ell+1)\omega}}{2} & \text{otherwise} \end{cases}. \tag{67}$$

3.2.2.6 Differentiation. Differentiation results in a cross coupling between the d and q axis terms. This has been solved in previous work [73, 74], however, it is re-derived here for completeness. The chain rule is used to differentiate a periodic time variant signal,

$$\frac{d}{dt}(x_{dq} e^{j\omega t}) = \dot{x}_{dq} e^{j\omega t} + j\omega x_{dq} e^{j\omega t}. \tag{68}$$

In generating state equations, by utilizing harmonic balance, the periodically time varying portion of each term is eliminated. Splitting x_{dq} into its d and q axis terms yields the real form of the solution,

$$\begin{aligned} \dot{x}_{d\ell\omega} &= f\left(x_{dc}, x_{d1\omega}, x_{q1\omega}, \ldots\right) + \ell\omega x_{q\ell\omega} \\ \dot{x}_{q\ell\omega} &= f\left(x_{dc}, x_{d1\omega}, x_{q1\omega}, \ldots\right) - \ell\omega x_{d\ell\omega} \end{aligned}, \tag{69}$$

where the $f(x_{dc},x_{d1\omega},x_{q1\omega},...)$ term represents the user generated state equation and the $\ell\omega x_{q\ell\omega}$ and $\ell\omega x_{d\ell\omega}$ terms represent the cross coupling terms resulting from differentiation. In Section 3.3.1, these cross coupling terms are represented by the *XCoupling()* function.

3.3. STATE EQUATION GENERATION

Utilizing the derived state equations and algorithms from Section 3.2, a set of harmonic functions can be derived. The presented algorithms can be implemented in a symbolic solver tool such as found in Matlab®, Mathematica®, Maple®, and Mathcad®. This study utilizes the Mupad® solver found in Matlab® R2008b.

3.3.1. System Equations. Implementing (2-9) in a symbolic math solver is shown below. The system equations are:

$$\mathbf{i}_{\mathbf{LDelay}} = Delay\left(\mathbf{i}_{\mathbf{L}}\right) \tag{70}$$

$$\left[\mathbf{i}_{\mathbf{Ld}},\mathbf{i}_{\mathbf{Lq}}\right] = ToDQ\left(\mathbf{i}_{\mathbf{L}},\mathbf{i}_{\mathbf{LDelay}}\right) \tag{71}$$

$$\mathbf{v}_{\mathbf{avd}} = \mathbf{x}_{\mathbf{1}} + K_{pdq}\left(\mathbf{x}_{\mathbf{3}} + K_{pdc}\left[\mathbf{v}_{\mathbf{dc}} - \mathbf{v}_{\mathbf{dc}}^{*}\right] - \mathbf{i}_{\mathbf{Ld}}\right) - \omega L\mathbf{i}_{\mathbf{Lq}} \tag{72}$$

$$\mathbf{v}_{\mathbf{avq}} = \mathbf{x}_{\mathbf{2}} - K_{pdq}\mathbf{i}_{\mathbf{Lq}} + \omega L\mathbf{i}_{\mathbf{Ld}} \tag{73}$$

$$\mathbf{v}_{\mathbf{av}} = FromDQ\left(\mathbf{v}_{\mathbf{avd}},\mathbf{v}_{\mathbf{avq}}\right) \tag{74}$$

$$\mathbf{i}_{\mathbf{inv}} = Mult\left(Mult\left(\mathbf{v}_{\mathbf{av}},\mathbf{i}_{\mathbf{L}}\right),Inverse\left(\mathbf{v}_{\mathbf{dc}}\right)\right) \tag{75}$$

These system equations are used in the state equations:

$$\dot{\mathbf{v}}_{\mathbf{dc}} = \frac{1}{C}\left(\mathbf{i}_{\mathbf{in}} - \mathbf{i}_{\mathbf{inv}}\right) - XCoupling\left(\mathbf{v}_{\mathbf{dc}}\right) \tag{76}$$

$$\dot{\mathbf{i}}_{\mathbf{L}} = \frac{1}{L}\left(\mathbf{v}_{\mathbf{av}} - \mathbf{v}_{\mathbf{grid}}\right) - XCoupling\left(\mathbf{i}_{\mathbf{L}}\right) \tag{77}$$

$$\dot{\mathbf{x}}_1 = K_{idq}\left(\mathbf{x}_3 + K_{pdc}\left[\mathbf{v}_{dc} - \mathbf{v}_{dc}^*\right] - \mathbf{i}_{Ld}\right) - XCoupling\left(\mathbf{x}_1\right) \tag{78}$$

$$\dot{\mathbf{x}}_2 = -K_{idq}\mathbf{i}_{Lq} - XCoupling\left(\mathbf{x}_2\right) \tag{79}$$

$$\dot{\mathbf{x}}_3 = K_{idc}\left(\mathbf{v}_{dc} - \mathbf{v}_{dc}^*\right) - XCoupling\left(\mathbf{x}_3\right) \tag{80}$$

All bold typeface variables in (70-80) represent vectors of length ($2N_{max}$+1) of Fourier series expanded terms. In this study N_{max}=6 harmonics are used with M=5 states. A total of 70 equations are generated from the original 5 state equations. Once the symbolic math solver completes generating (76-80), the symbolic state equations can be stored. Next, numeric values can be plugged into the stored equations to solve for the steady state operating point of the system.

3.3.2. Note on $\mathbf{i_{inv}}$ Calculation. The computational complexity of the harmonic inverse operation increases dramatically with N_{max}. This is due to symbolically placing $\mathbf{A_{aug}}$ in reduced row echelon form. A numeric implementation, such as *rref()* in Matlab®, is $O(n^3)$, and here a symbolic implementation is needed. Any shortcuts to lessen the load on the inverse calculation greatly reduce the total computation time.

One shortcut is to add numeric zeros into $\mathbf{v_{dc}}$ in (75). It can be seen by observation that the odd harmonic terms in $\mathbf{v_{dc}}$ are near zero. By replacing the 1st, 3rd, and 5th harmonic d and q axis values of $\mathbf{v_{dc}}$ with zeros, the computational complexity of the inverse is reduced. This does not significantly affect the steady state accuracy of the system, since these terms are near zero.

Another shortcut is to truncate the output of the inverse calculation. It can be seen that the even harmonics beyond the 4th harmonic are also very close to zero. By truncating the results and only finding the inverse results for the dc term and the first 4

harmonics, the calculation time can be drastically reduced. Since the 5th and 6th harmonic terms are small, the steady state accuracy of the system is minimally impacted.

3.4. SOLVING STATE EQUATIONS

The resulting equations from Section 3.3 are nonlinear time invariant equations. To solve for the steady state operating point, $\dot{\mathbf{v}}_{\mathbf{dc}}$, $\dot{\mathbf{i}}_{\mathbf{L}}$, $\dot{\mathbf{x}}_{\mathbf{1}}$, $\dot{\mathbf{x}}_{\mathbf{2}}$, and $\dot{\mathbf{x}}_{\mathbf{3}}$ are set to $\mathbf{0}$. In this study, the Matlab® function *solve()* is utilized to solve the set of nonlinear equations. Solving for all 70 unknowns at the same time is computationally expensive. An iterative process is presented in this section to reduce the computation time.

In the first step, the solver will neglect all harmonics past the 1st harmonic. All states corresponding to the 2nd through 6th harmonic are set to a numeric value of zero. The solver will only be solving for the dc and 1st harmonic in this step. This gives the solver a much more manageable 15 unknowns to solve for.

After solving for the dc and first harmonic terms, the first and second harmonic terms are solved for. In this iteration, the dc solution from the first step is used for the dc terms instead of zero. The third harmonic and higher terms remain at zero for this step. This process of incrementing the harmonics being solved for and using the solution from the previous iteration repeats until the fifth and sixth harmonic is solved for.

After the fifth and sixth harmonic is solved for, the process starts over; the dc term and first harmonic are solved for, while substituting the solutions from the previous iterations for the second through sixth harmonics, then the first harmonic and second harmonic are solved for, and so on. This entire process of iterating up to the sixth harmonic then returning to the dc terms can be continuously repeated, to obtain a more

accurate solution. The results in this study show that only two iterations are necessary to yield an accurate solution.

3.5. SIMULATION AND EXPERIMENTAL RESULTS

An average value simulation of Figure 3.2 and Figure 3.3 was generated using Matlab®/Simulink®. Parameters used in the system are shown in Table 3.2. Inputs to the system are shown in Table 3.3.

Table 3.2. System parameters.

Variable	Description	Value
L	Filter Inductance	8.2 mH
C	dc-link Capacitance	110 μF
ω	Inverter Fundamental Frequency	2π60
Kidq	Integral Current Loop Gain	50
Kpdq	Proportional Current Loop Gain	2.174
Kidc	Integral dc-link Voltage Loop Gain	1
Kpdc	Proportional dc-link Voltage Loop Gain	0.1

The system inputs were derived by taking the FFT of the no-load grid voltage. The real and imaginary parts of the FFT were used to find the grid voltage magnitude and phase. The resulting magnitude and phase was converted to the form in (41). The phase was then shifted so that $\left(v_{grid}\right)_{q1\omega} = 0$.

Table 3.3. System inputs.

Harmonic	v_{grid}	i_{in}	v_{dc}^{*}
dc	0.0000	1	60
d1ω	29.5000	0	0
q1ω	0.0000	0	0
d2ω	0.0000	0	0
q2ω	0.0000	0	0
d3ω	0.0169	0	0
q3ω	-0.1885	0	0
d4ω	-0.0025	0	0
q4ω	-0.0153	0	0
d5ω	0.1077	0	0
q5ω	0.3133	0	0
d6ω	-0.0013	0	0
q6ω	-0.0400	0	0

The experimental setup is illustrated in Figure 3.4. In the experimental setup, the 1 A constant input was generated using a 100 V DC source in series with a BK Precision 8500 programmable dc electronic load. The electronic load was set to regulate a constant 1 A. The controls from Figure 3.3 were implemented using a TI TMS320F28335 DSP and programmed using Code Composer Studio.

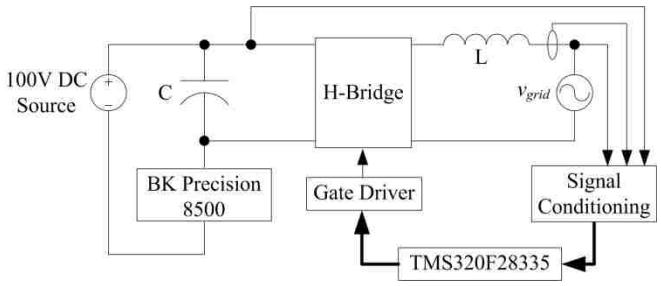

Figure 3.4. Experimental setup.

The simulated dc-link voltage is compared to the predicted steady-state dc-link voltage in Figure 3.5. The simulated grid current is compared to the predicted steady-state grid current in Figure 3.6. An FFT was taken of the simulated dc-link voltage (at steady-state) and experimental dc-link voltage. These results are compared with the predicted steady-state dc-link voltage harmonic amplitudes in Figure 3.7. Lastly, an FFT was taken of the simulated grid current (at steady-state) and experimental grid current. These results were compared with the predicted steady-state grid current harmonic amplitudes in Figure 3.8.

In Figure 3.7 and Figure 3.8, values were predicted for dc, 60 hz, 120 Hz, 180 Hz, 240 Hz, 300 Hz, and 360 Hz. Some of these predicted values are missing from these figures. This is due to the predicted values being far below the noise floor for the simulation and experimental results.

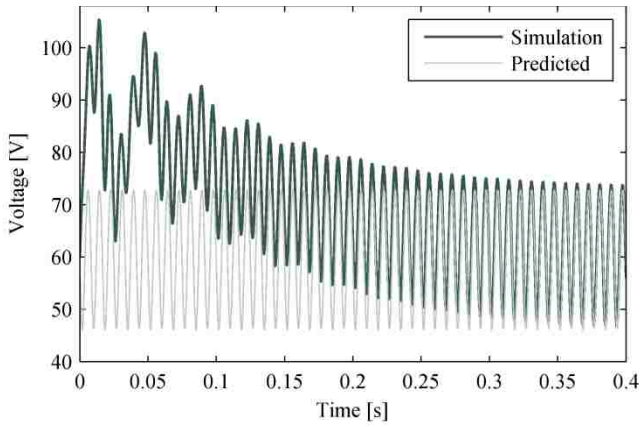

Figure 3.5. v_{dc} simulation and predicted steady state v_{dc}.

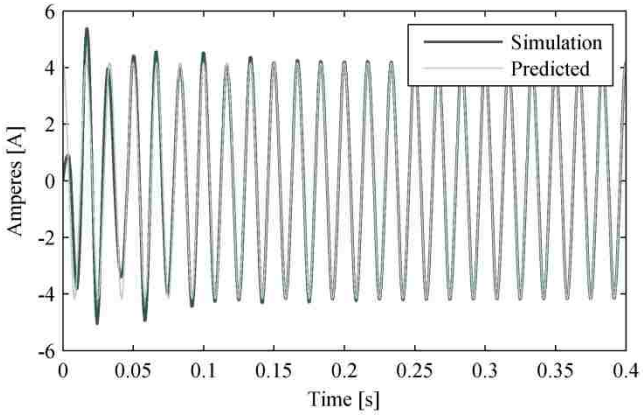

Figure 3.6. i_{grid} simulation and predicted steady state i_{grid}.

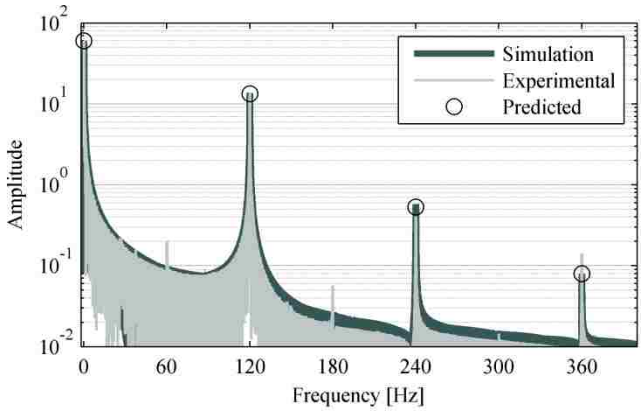

Figure 3.7. FFT of simlated and experimental v_{dc} and predicted v_{dc} harmonic amplitudes.

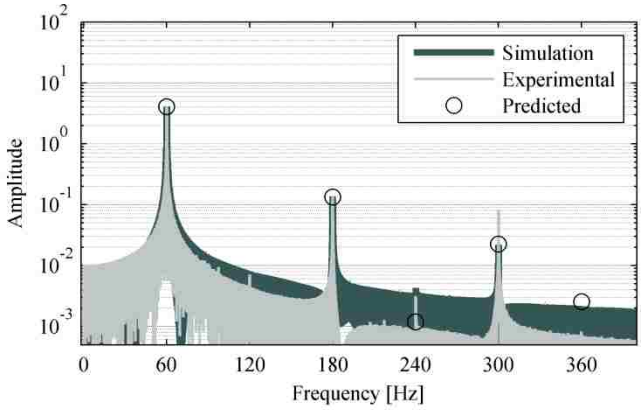

Figure 3.8. FFT of simlated and experimental i_{grid} and predicted i_{grid} harmonic amplitudes.

3.6. CONCLUSIONS

This study proposed a new method to generate an accurate time invariant large-signal model of a single phase inverter. Algorithms were presented, which allowed the user to take their standard large-signal model, and have its variables computationally expanded to their Fourier series representation. This allowed for the preservation of frequency transfer properties in the inverter. Because of presented algorithms like *To DQ*

and *From DQ*, inverters with dq-axis controllers can be easily modeled using the proposed method.

With the proposed algorithms, an accurate large-signal model was generated. A Simulink simulation was utilized to verify the predicted results. The simulation results from Figure 3.5 and Figure 3.6 shows v_{dc} and i_L approaching the predicted values, as they reach steady state. These simulation results verify that the algorithms in this paper are correct.

Figure 3.7 and Figure 3.8 take the FFT of the simulated and experimental results and compare them with the predicted harmonic steady state output. Those results are once again very accurate. At frequencies near the noise floor of the FFT, there are minor discrepancies, however this is to be expected. At the frequencies of interest, the results are accurate.

Future work for this study is to use the generated results as a linearization point for the system. A harmonic state space representation of the system can then be generated. Since all frequency transfer properties between the first N_{max} harmonics are preserved, this will allow for more accurate linear models. Advanced stability methods could then be applied to the linear model.

The same type of analysis presented here could be utilized in three phase inverters. Since inputs to the proposed model are EMP's, there could be an advantage in environments with a high THD in the grid voltage. Another area which could gain some insight from the proposed methods is in unbalanced three phase inverters. In a balanced three phase system, there is a constant power draw from the dc-link. In an unbalanced

three phase system, harmonics appear in the power draw from the dc-link. Using the methods proposed in this study, the effects of those harmonics on the grid current could be studied.

4. AUTOMATIC GENERATION OF TRUNCATED HARMONIC TRANSFER FUNCTIONS

4.1. INTRODUCTION

Generating a harmonic transfer function (HTF) to represent a system is a nontrivial process. This is due to the HTF being infinite-dimensional. Even if the infinite dimensional HTF is reduced to a truncated harmonic transfer function (THTF), a time consuming and error prone process is still required to derive the THTF. In generating the THTF, a transfer function going from every combination of inputs and outputs and each of those input's and output's harmonics must be generated. The solution presented in this study is a computational algorithm which automatically generates the THTF of a system.

HTF's were first introduced by Wereley for the application of modeling helicopter rotor blades [62, 63]. HTF's are an infinitely dimensional set of transfer functions which map a set of exponentially modulated periodic (EMP) inputs to a set of EMP outputs. What this means is that the inputs and outputs of the system are a set of Fourier series coefficients. A system which contains periodically time varying elements is converted to a system which is time invariant. The conversion to a time invariant system is done by equating like harmonics, then cancelling out the periodically time varying terms. This is known as harmonic balance. In this study, to reduce the complexity of the system the higher order harmonics of the HTF's are truncated. This concept was explored thoroughly by Sandberg in [64].

One type of system which benefits from the use of HTFs is inverters. On the dc side of the inverter, the fundamental grid frequency is rectified to a dc plus double grid

frequency component. The double frequency component then becomes a third harmonic component on the ac side of the inverter. The third harmonic reflects back to the dc side of the inverter, and so on and so forth [57, 84]. HTFs allow for these interfrequency interactions to be modeled.

There are many ways that the THTF can be calculated, with each method providing its own challenges. One method is to use system identification methods to calculate the THTF. Louarroudi proposed a nonparametric estimation procedure to identify a system's THTF [85]. Generalized averaging (GA) is used by Qin in [86] to generate a model of a DAB converter. GA is similar to the THTF in that periodically time varying signals are converted to time invariant signals. A popular option to model time varying signals as time invariant signals is to use describing functions (DF). In [87], DFs are used to generate transfer functions which preserves the switching frequency harmonic. From [87], it is apparent that adding even one frequency to be analyzed can greatly complicate transfer function generation.

The easiest way to generate the THTF is to have a computational algorithm automatically generate the THTF. Wasynczuk formulated the building blocks for this study with the automatic state model generator (ASMG) [88]. The AMSG is capable of generating a state space model of a system consisting of resistors, inductors, capacitors, voltage sources, and current sources. Johnson later utilized the ASMG for the modeling of microgrids [89]. Henry expanded upon the AMSG to allow for the generation of state models for switch capacitor converters [90]. This study builds upon the AMSG to generate a THTF for a single phase dq-axis current controlled inverter.

In this study, the generation of the THTF is divided into two sections: the generation of low level circuit transfer functions and the generation of the high level THTF. Section 4.2 describes the low level circuit transfer function generation. Components modeled at the low level are: resistors, inductors, capacitors, voltage sources, and current sources. The notation of the user generated netlist which describes the circuits within the system is explained. The user generated netlist is converted into a set of dc transfer functions, as well as a set of transfer functions in the dq rotating reference frame.

Section 4.3 describes the high level THTF generation. Components modeled at the high level are: inputs, adds, gains, integrations, 90° delays, conversions to dq, conversions from dq, circuits, multiplications, and inversions. Next, the notation of the user generated system level netlist is explained. Finally, the user generated netlist is converted into a THTF.

Section 4.4 provides an example user generated netlist which describes a single-phase grid-tied H-bridge inverter. Section 4.5 highlights a generated transfer function which shows how the 120 Hz dc-link ripple on the inverter will affect the output grid current. This is verified using experimental results.

4.2. LOW LEVEL CIRCUIT TF GENERATION

Every periodic signal can be represented by

$$x = x_{dc} + \sum_{k=1}^{\infty}\left[x_{dk}\cos(\omega kt) - x_{qk}\sin(\omega kt)\right]. \tag{81}$$

This is essentially a Fourier series representation of a signal. Each state can be separated into a dc portion (x_{dc}), a sum of d-axis portions ($x_{dk}\cos(\omega kt)$), and a sum q-axis portions

($-x_{qk}\sin(\omega kt)$). Doing this requires $4(N_\omega+1)M$ equations, where N_ω is the number of harmonics to be analyzed and M is the number of branches in the circuit.

In generating low level circuit transfer functions is a two step process: First, the dc transfer functions must be generated. Second, the direct-quadrature (DQ) transfer functions must be generated.

For generating the dc circuit TFs, equations must be derived in the form of

$$\mathbf{x} = \mathbf{A}_{tf}\mathbf{x} + \mathbf{B}_{tf}\mathbf{u}\,. \tag{82}$$

Solving for **x** yields

$$\mathbf{x} = (\mathbf{I} - \mathbf{A}_{tf})^{-1}\mathbf{B}_{tf}\mathbf{u}\,. \tag{83}$$

The software must generate $\mathbf{A}_{tf}$ and $\mathbf{B}_{tf}$ from a user generated netlist. Equation (82) can be further broken down into

$$\begin{bmatrix} \mathbf{i}_{y(n-1,1)} \\ \mathbf{i}_{x(b-n+1,1)} \\ \mathbf{v}_{x(n-1,1)} \\ \mathbf{v}_{y(b-n+1,1)} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{(n-1,n-1)} & -\hat{\mathbf{A}}_{(n-1,b-n+1)} & \mathbf{0}_{(n-1,n-1)} & \mathbf{0}_{(n-1,b-n+1)} \\ \mathbf{0}_{(b-n+1,n-1)} & \mathbf{0}_{(b-n+1,b-n+1)} & \mathbf{0}_{(b-n+1,n-1)} & \mathbf{E}_{(b-n+1,b-n+1)} \\ \mathbf{F}_{(n-1,n-1)} & \mathbf{0}_{(n-1,b-n+1)} & \mathbf{0}_{(n-1,n-1)} & \mathbf{0}_{(n-1,b-n+1)} \\ \mathbf{0}_{(b-n+1,n-1)} & \mathbf{0}_{(b-n+1,b-n+1)} & \hat{\mathbf{A}}^{\mathbf{T}}_{(b-n+1,n-1)} & \mathbf{0}_{(b-n+1,b-n+1)} \end{bmatrix} \begin{bmatrix} \mathbf{i}_{y(n-1,1)} \\ \mathbf{i}_{x(b-n+1,1)} \\ \mathbf{v}_{x(n-1,1)} \\ \mathbf{v}_{y(b-n+1,1)} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{(n-1,ni)} & \mathbf{0}_{(n-1,nv)} \\ \mathbf{b}_{i(b-n+1,ni)} & \mathbf{0}_{(b-n+1,nv)} \\ \mathbf{0}_{(n-1,ni)} & \mathbf{b}_{v(n-1,nv)} \\ \mathbf{0}_{(b-n+1,ni)} & \mathbf{0}_{(b-n+1,nv)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{(ni,1)} \\ \mathbf{u}_{(nv,1)} \end{bmatrix}, \tag{84}$$

where the values in ()'s give the dimension of each matrix, *n* is the number of nodes, *b* is the number of branches, *nv* is the number of voltage sources, and *ni* is the number of current sources. Further explanation will be given for each component of (84) in the following sections.

4.2.1. Netlist Generation. To generate $\hat{\mathbf{A}}$, first the user must input a netlist describing a circuit. Each row of the netlist describes a branch. A branch for this system is illustrated in Figure 4.1.

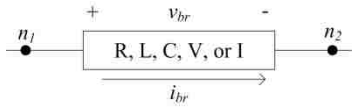

Figure 4.1. General branch description.

In this paper, at the circuit level, each branch consists of only one component, a resistor, an inductor, a capacitor, a voltage source, or a current source. Each row of the netlist contains the branch type (R, L, C, V, or I), the node that the positive terminal of the branch is connected to (n_1 in Figure 4.1), the node that the negative terminal of the branch is connected to (n_2 in Figure 4.1), the branch value, and the branch's name. A simple circuit is shown in Figure 4.2. This circuit contains 3 nodes and 4 branches. The netlist for this circuit is shown in Figure 4.3.

The value for the voltage sources and current sources in the netlist is used to specify the order that the inputs appear in the transfer functions matrix in (83). Before processing the netlist, voltage sources must be placed at the top of the netlist and current sources must be placed at the bottom of the netlist. This will be discussed further in Section 4.2. The netlist in Figure 4.3 happens to have the voltage source at the top and current source at the bottom. If the user does not input the voltage source and current source branches in the correct position, the software must rearrange the branches to a correct order, before processing.

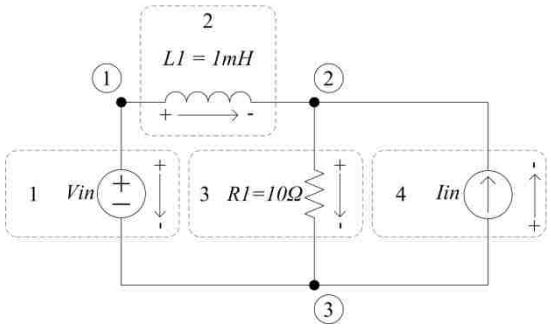

Figure 4.2. Sample circuit.

$$\mathbf{N} = \begin{array}{c} \begin{matrix} \text{type} & n_1 & n_2 & \text{value} & \text{name} \end{matrix} \\ \begin{bmatrix} V & 1 & 3 & 1 & \text{"}Vin\text{"} \\ L & 1 & 2 & 1e-3 & \text{"}L1\text{"} \\ R & 2 & 3 & 10 & \text{"}R1\text{"} \\ I & 3 & 2 & 2 & \text{"}Iin\text{"} \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} \text{ Branch \#} \end{array}$$

Figure 4.3. Sample circuit netlist.

4.2.2. Circuit TF Generation. From Figure 4.3, the node incidence matrix, A_a can be generated. Each column in $\mathbf{A}_a$ corresponds with a branch in **N**. Each row in $\mathbf{A}_a$ corresponds with a node in **N**. If branch *i*'s positive (negative) terminal is on node *j*, then $\mathbf{A}_{a(i,j)} = 1$ ($\mathbf{A}_{a(i,j)} = -1$). All other entries in $\mathbf{A}_a$ are 0. From Figure 4.3, $\mathbf{A}_a$ is generated and shown in Figure 4.4.

$$\mathbf{A}_a = \begin{array}{c} \text{branch \#} \rightarrow \\ \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & -1 & 1 & -1 \\ -1 & 0 & -1 & 1 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \text{ node \#} \end{array}$$

Figure 4.4. Sample node incidence matrix.

Placing $\mathbf{A}_a$ in reduced row echelon form yields $\tilde{\mathbf{A}}_a$, in the form of:

$$\tilde{\mathbf{A}}_a = \begin{bmatrix} \mathbf{I}_{n-1,n-1} & \hat{\mathbf{A}}_{n-1,b-n+1} \\ \mathbf{0}_{1,n-1} & \mathbf{0}_{1,b-n+1} \end{bmatrix} = \left[\begin{array}{cc:cc} 1 & 0 & 1 & -1 \\ 0 & 1 & -1 & 1 \\ \hdashline 0 & 0 & 0 & 0 \end{array}\right]. \tag{85}$$

Columns in (85) may need to be swapped to get the correct form. If this is the case, the corresponding branches (rows) in **N** must also be swapped. The $\tilde{\mathbf{A}}_a$ matrix in (85) gives the $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}^{\mathbf{T}}$ components of $\mathbf{A}_{tf}$.

$\hat{\mathbf{A}}$ gives *n-1* linearly independent KCL equations. Let $\mathbf{i_y}$ be currents corresponding to the first *n-1* branches in the netlist, **N** and let $\mathbf{i_x}$ be the currents corresponding to the last *b-n+1* branches in the netlist **N**. From this and (85),

$$\mathbf{i}_y = -\hat{\mathbf{A}}\mathbf{i}_x \tag{86}$$

is derived, which relates the dependent currents ($\mathbf{i_y}$) to the independent currents ($\mathbf{i_x}$). By taking the transpose of $\hat{\mathbf{A}}$, a set of *b-n+1* linearly independent KVL equations is given in the form of

$$\mathbf{v}_y = \hat{\mathbf{A}}^T \mathbf{v}_x, \tag{87}$$

which relates the dependent voltages ($\mathbf{v_y}$) to the independent voltages ($\mathbf{v_x}$). From (86) and (87), *b* linearly independent equations are given. There must be *2b* linearly independent equations, since the current and voltage must be derived for each branch. The remaining *b* equations are derived using branch constitutive equations, which are defined in **E**, **F**, $\mathbf{b_i}$, and $\mathbf{b_v}$ from (84).

Equations for the independent variables ($\mathbf{i_x}$ and $\mathbf{v_x}$) can be derived programmatically. The algorithm used to define **F** and $\mathbf{b_v}$, which maps the dependent

currents to the independent voltages, is shown in Figure 4.5. First, **F**, $\mathbf{b}_v$, and *i* are initialized. The first *n-1* branches (rows) of netlist **N** are parsed through by algorithm's loop. **F** is populated by setting its diagonal entries to the corresponding branch's impedance. If the *i*th branch is a resistor, the *i*th diagonal entry of **F** is set to the branch's resistive value. If the *i*th branch is an inductor, the *i*th diagonal entry of **F** is set to the branch's impedance (*sL*). If the *i*th branch is a capacitor, the *i*th diagonal entry of **F** is set to the branch's impedance ($1/sC$). If the *i*th branch is a voltage source, then the *i*th row and the column corresponding with the voltage source of $\mathbf{b}_v$ is set to 1. If the netlist **N** follows the convention of ordering all voltage source branches first and all current source branches last, then there will be no current sources in the first *n-1* branches and the error case will not be encountered. The exception to this rule is if the user has violated KCL, by placing multiple current sources in series.

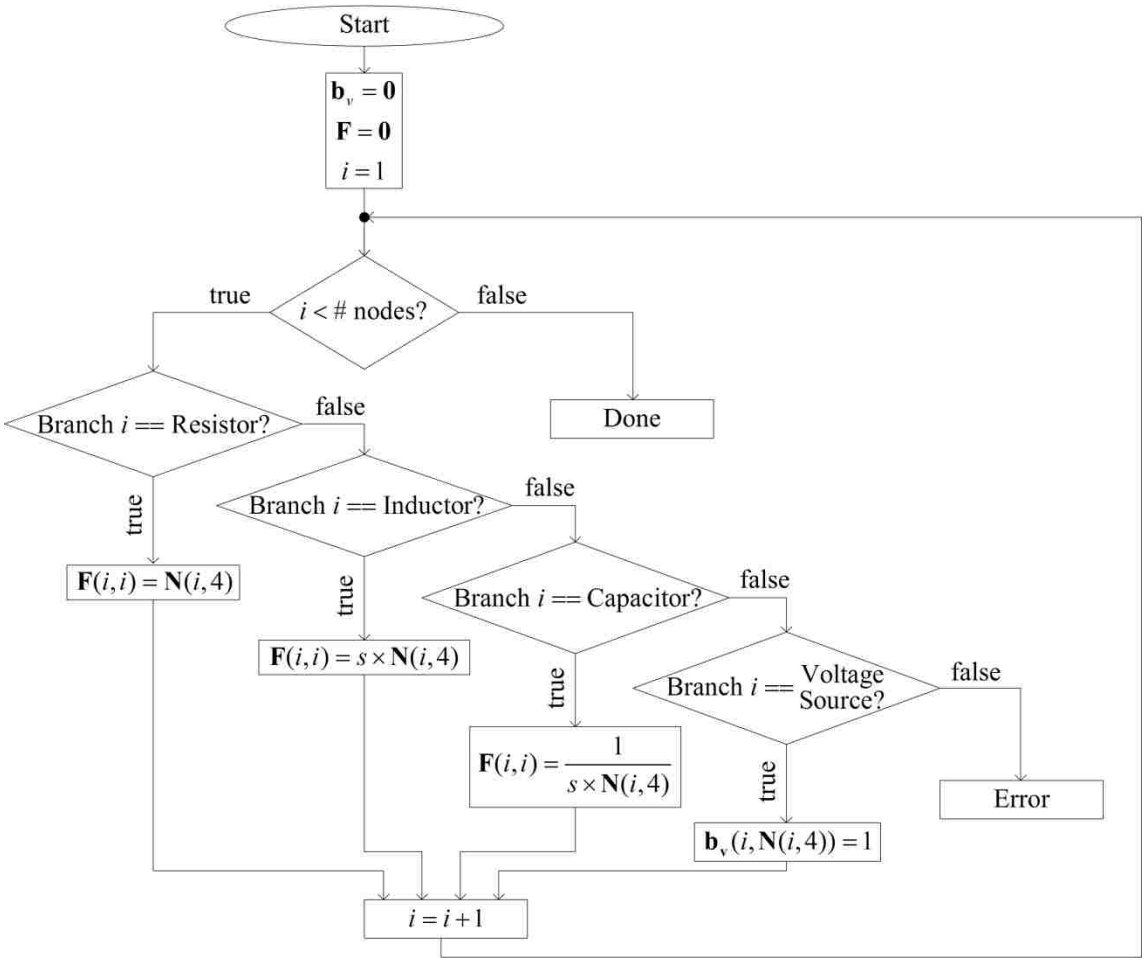

Figure 4.5. **F** and $\mathbf{b_v}$ generation algorithm.

The algorithm used to define **E** and $\mathbf{b_i}$, which maps the dependent voltages to the independent currents, is shown in Figure 4.6. First, **E**, $\mathbf{b_i}$, and *i* are initialized. The last *b-n+1* branches (rows) of netlist **N** are parsed through by algorithm's loop. **E** is populated by setting its diagonal entries to the corresponding branch's admittance. If the *i*th branch is a resistor, the *i*th diagonal entry of **E** is set to the inverse of the branch's resistive value. If the *i*th branch is an inductor, the *i*th diagonal entry of **E** is set to the branch's admittance ($1/sL$). If the *i*th branch is a capacitor, the *i*th diagonal entry of **E** is set to the branch's admittance (sC). If the *i*th branch is a current source, then the *i*th row and the

column corresponding with the voltage source of $\mathbf{b}_i$ is set to -1. If the netlist **N** follows the convention of ordering all voltage source branches first and all current source branches last, then there will be no current sources in the last *b-n+1* branches and the error case will not be encountered. The exception to this rule is if the user has violated KVL, by placing multiple voltage sources in parallel.

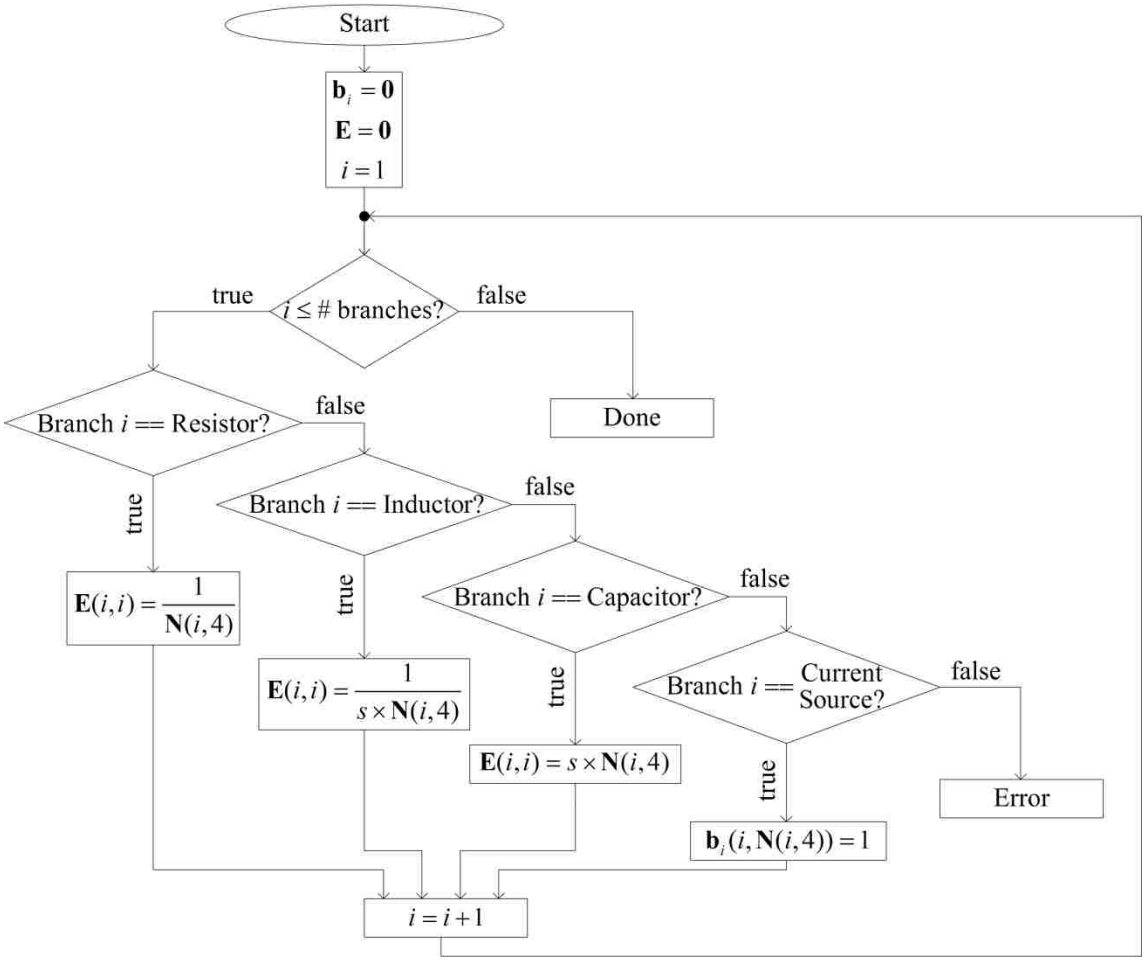

Figure 4.6. **E** and $\mathbf{b_i}$ generation algorithm.

All variables in (84) are now defined. Transfer functions for the basic case can be solved for using (83). These transfer functions are in the form

$$\frac{\mathbf{x}}{\mathbf{u}} = \begin{bmatrix} \frac{i_1}{i_{in(1)}} & \frac{i_1}{i_{in(2)}} & \cdots & \frac{i_1}{i_{in(ni)}} & \frac{i_1}{v_{in(1)}} & \frac{i_1}{v_{in(2)}} & \cdots & \frac{i_1}{v_{in(nv)}} \\ \frac{i_2}{i_{in(1)}} & \frac{i_2}{i_{in(2)}} & & \frac{i_2}{i_{in(ni)}} & \frac{i_2}{v_{in(1)}} & \frac{i_2}{v_{in(2)}} & & \frac{i_2}{v_{in(nv)}} \\ \vdots & & \ddots & & \vdots & & \ddots & \vdots \\ \frac{i_b}{i_{in(1)}} & \frac{i_b}{i_{in(2)}} & & \frac{i_b}{i_{in(ni)}} & \frac{i_b}{v_{in(1)}} & \frac{i_b}{v_{in(2)}} & & \frac{i_b}{v_{in(nv)}} \\ \frac{v_1}{i_{in(1)}} & \frac{v_1}{i_{in(2)}} & \cdots & \frac{v_1}{i_{in(ni)}} & \frac{v_1}{v_{in(1)}} & \frac{v_1}{v_{in(2)}} & \cdots & \frac{v_1}{v_{in(nv)}} \\ \frac{v_2}{i_{in(1)}} & \frac{v_2}{i_{in(2)}} & & \frac{v_2}{i_{in(ni)}} & \frac{v_2}{v_{in(1)}} & \frac{v_2}{v_{in(2)}} & & \frac{v_2}{v_{in(nv)}} \\ \vdots & & \ddots & & \vdots & & \ddots & \vdots \\ \frac{v_b}{i_{in(1)}} & \frac{v_b}{i_{in(2)}} & \cdots & \frac{v_b}{i_{in(ni)}} & \frac{v_b}{v_{in(1)}} & \frac{v_b}{v_{in(2)}} & \cdots & \frac{v_2}{v_{in(nv)}} \end{bmatrix}. \tag{88}$$

The numerator in each entry represents either the current though a branch or the voltage across the branch. The branch number is specified by the numerator's subscript (with *b* representing the number of branches in the circuit). The denominator in each entry represents an input current or voltage. The input source number is specified by the subscript in parenthesis (with *ni* representing the number of current sources in the circuit and *nv* representing the number of voltage sources in the circuit).

In this section, standard dc transfer functions were derived for the voltage and current of each branch. In the next section, transfer functions are derived at the circuit level, for the periodically time varying terms of the Fourier series expanded terms. This allows for a periodically time varying system to be accurately analyzed as a time invariant system.

4.2.3. DQ TF Generation. Section 4.2.2 discussed generating a set of standard dc transfer functions for a system. This section discusses generating circuit level transfer functions using a DQ rotating reference frame. This is equivalent to finding the Fourier series coefficients of each variable. Each DQ equation describes the input to output relationship of a signal at a specific frequency. The concept of is the same as generating equations which use Fourier series coefficients as their inputs and outputs.

For each harmonic to be analyzed, a new set of equations in the form of (82) is generated. In generating DQ equations, (82) is represented as (89). Equation (89) is very similar to (84). The main difference lies in the size of the DQ equations. There are twice as many DQ equations as there are dc equations (for a specific frequency). This is due to each DQ equation being represented by both the amplitude of a cosine signal (d-axis) at a specific frequency and the amplitude of a negative sine signal (q-axis) at a specific frequency.

$$\begin{bmatrix} \mathbf{i}_{dqy(2(n-1),1)} \\ \mathbf{i}_{dqx(2(b-n+1),1)} \\ \mathbf{v}_{dqx(2(n-1),1)} \\ \mathbf{v}_{dqy(2(b-n+1),1)} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{(2(n-1),2(n-1))} & -\hat{\mathbf{A}}_{dq(2(n-1),2(b-n+1))} & \mathbf{0}_{(2(n-1),2(n-1))} & \mathbf{0}_{(2(n-1),2(b-n+1))} \\ \mathbf{0}_{(2(b-n+1),2(n-1))} & \mathbf{G}_{dq(2(b-n+1),2(b-n+1))} & \mathbf{0}_{(2(b-n+1),2(n-1))} & \mathbf{E}_{dq(2(b-n+1),2(b-n+1))} \\ \mathbf{F}_{dq(2(n-1),2(n-1))} & \mathbf{0}_{(2(n-1),2(b-n+1))} & \mathbf{H}_{dq(2(n-1),2(n-1))} & \mathbf{0}_{(2(n-1),2(b-n+1))} \\ \mathbf{0}_{(2(b-n+1),2(n-1))} & \mathbf{0}_{(2(b-n+1),2(b-n+1))} & \hat{\mathbf{A}}^{\mathbf{T}}_{dq(2(b-n+1),2(n-1))} & \mathbf{0}_{(2(b-n+1),2(b-n+1))} \end{bmatrix} \cdots (89)$$
$$\begin{bmatrix} \mathbf{i}_{dqy(2(n-1),1)} \\ \mathbf{i}_{dqx(2(b-n+1),1)} \\ \mathbf{v}_{dqx(2n-1),1)} \\ \mathbf{v}_{dqy(2(b-n+1),1)} \end{bmatrix} + \begin{bmatrix} \mathbf{0}_{(2(n-1),2ni)} & \mathbf{0}_{(2(n-1),2nv)} \\ \mathbf{b}_{dqi(2(b-n+1),2ni)} & \mathbf{0}_{(2(b-n+1),2nv)} \\ \mathbf{0}_{(2(n-1),2ni)} & \mathbf{b}_{dqv(2(n-1),2nv)} \\ \mathbf{0}_{(2(b-n+1),2ni)} & \mathbf{0}_{(2(b-n+1),2nv)} \end{bmatrix} \begin{bmatrix} \mathbf{u}_{dq(2ni,1)} \\ \mathbf{u}_{dq(2nv,1)} \end{bmatrix}$$

To generate dq equations, each submatrix from (89) must be defined. The $\hat{\mathbf{A}}_{\mathbf{dq}}$, $\mathbf{E}_{\mathbf{dq}}$, and $\mathbf{F}_{\mathbf{dq}}$ matrices are easily generated using $\hat{\mathbf{A}}$, **E**, and **F** respectively from (84). Multiplying each individual entry in $\hat{\mathbf{A}}$, **E**, and **F** by a 2x2 identity matrix will generate the $\hat{\mathbf{A}}_{\mathbf{dq}}$, $\mathbf{E}_{\mathbf{dq}}$, and $\mathbf{F}_{\mathbf{dq}}$ matrices. For example, if

$$\mathbf{E} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \text{ then } \mathbf{E}_{\mathbf{dq}} = \begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 4 & 0 \\ 0 & 3 & 0 & 4 \end{bmatrix}. \tag{90}$$

This can be done because the variable and input vectors are in the form:

$$\mathbf{x} = \begin{bmatrix} x_{1d} \\ x_{1q} \\ x_{2d} \\ x_{2d} \\ \vdots \end{bmatrix}, \mathbf{u} = \begin{bmatrix} u_{1d} \\ u_{1q} \\ u_{2d} \\ u_{2d} \\ \vdots \end{bmatrix}. \tag{91}$$

Another difference is that (89) adds the $\mathbf{G}_{\mathbf{dq}}$ and $\mathbf{H}_{\mathbf{dq}}$ matrices. These matrices represent a cross coupling between the d-axis and q-axis equations, which occurs at energy storage elements in the circuits (this cross coupling is why many DQ controlled inverters have a $\pm\omega L$ decoupling term, just before generating the commanded output voltage). Cross coupling terms must also be added to the existing $\mathbf{E}_{\mathbf{dq}}$ and $\mathbf{F}_{\mathbf{dq}}$ matrices. $\mathbf{G}_{\mathbf{dq}}$ contains exclusively cross coupling terms for the independent branch currents. $\mathbf{H}_{\mathbf{dq}}$ contains exclusively cross coupling terms for the independent branch voltages. Addition of the cross coupling terms to (89) are shown in Figure 4.7 and Figure 4.8.

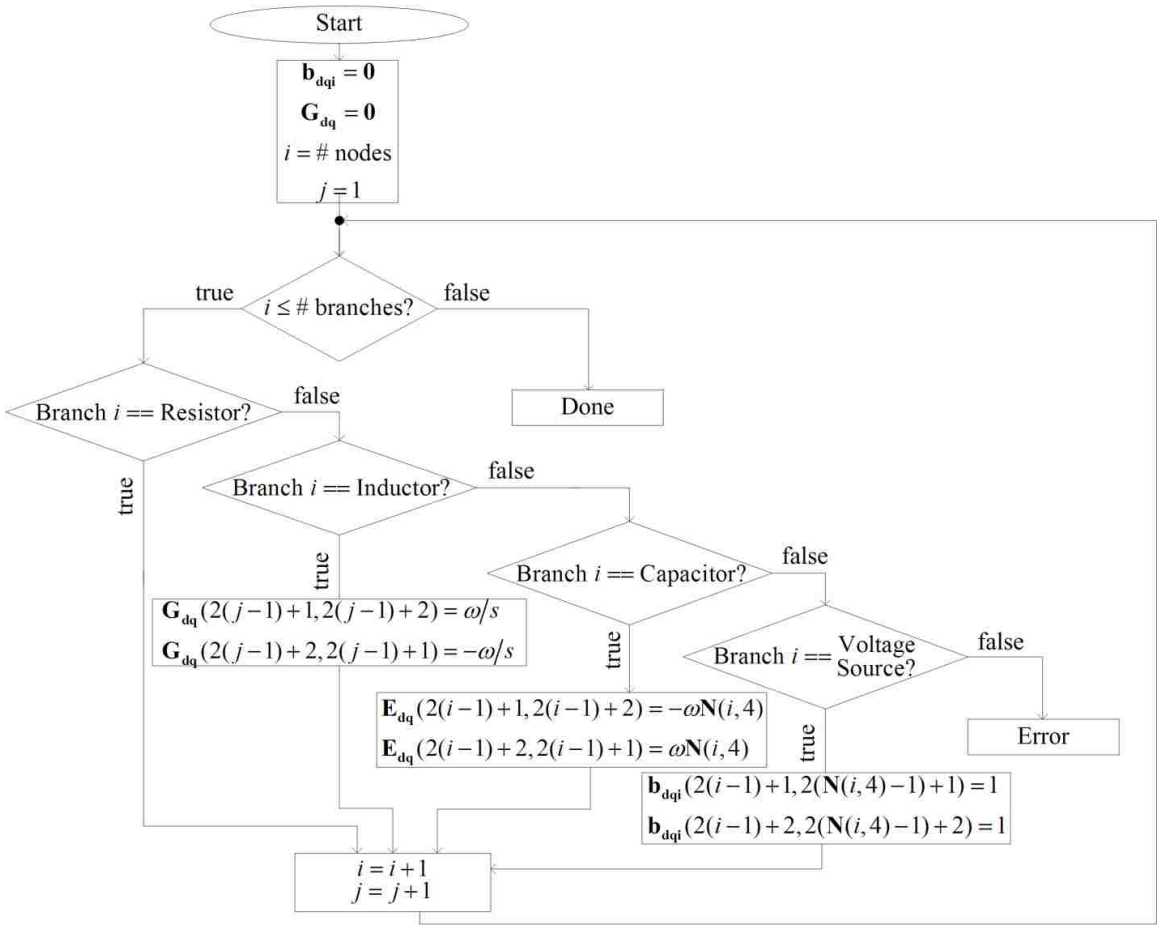

Figure 4.7. Generation of $\mathbf{G}_{dq}$ and $\mathbf{b}_{dqi}$. Addition of cross coupling terms to $\mathbf{E}_{dq}$.

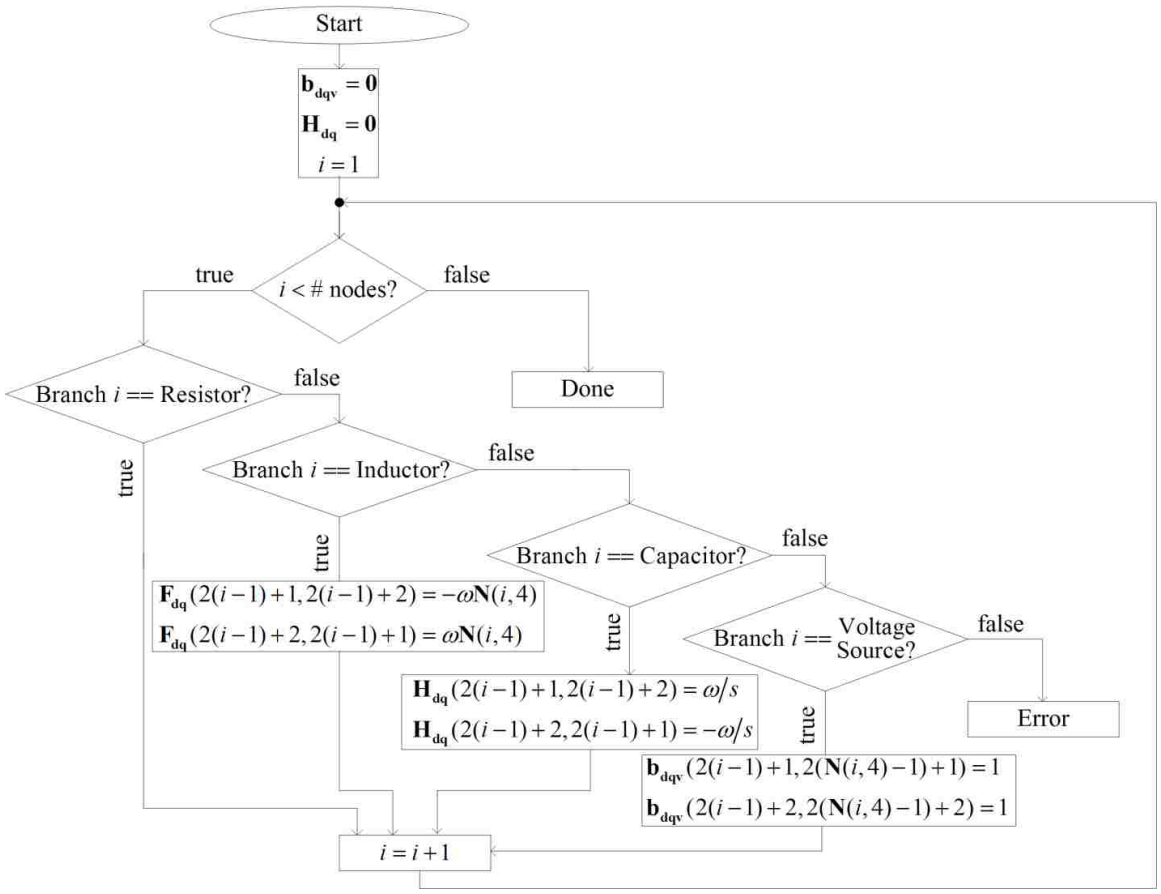

Figure 4.8. Generation of $\mathbf{H}_{dq}$ and $\mathbf{b}_{dqv}$. Addition of cross coupling terms to $\mathbf{F}_{dq}$.

The ω term in Figure 4.7 and Figure 4.8 is the frequency being analyzed. For example, if a user wants to analyze their system at dc, 60 Hz, 120 Hz, and 180 Hz. There will be one equation in the form of (84). There will also be three equations in the form of (89); One of these equations will have ω=2π60, one will have ω=2π120, and one will have ω=2π180. Since the individual circuits are linear, there is no transfer between the dc, 60 Hz, 120 Hz, and 180 Hz circuits. The transfer properties will come into play when processing the system level netlist.

4.3. HIGH LEVEL SYSTEM TF GENERATION

Section 4.2 described the generation of circuits containing simple blocks consisting of resistors, inductors, capacitors, voltage sources, and current sources. How control systems and these circuits interact with one another is described by the high level system netlist. The high level netlist defines the inputs to each circuit voltage and current source. The control system is also defined in the high level system netlist.

At the circuit level, multiple sets of equations are generated, one for the dc case, and one for each harmonic to be analyzed. At the system level however, only one set of equations in the form of (82) are generated. This set of equations will contain the dc equations, as well as all of the dq equations. $\mathbf{A_{sys}}$ and $\mathbf{B_{sys}}$ are the variables used to define the system of equations at the system level. $\mathbf{A_{sys}}$ is a $N_{sysNode}\left(2N_{\omega}+1\right)\times N_{sysNode}\left(2N_{\omega}+1\right)$ matrix, where $N_{sysNode}$ is the number of nodes at the system level. $\mathbf{B_{sys}}$ is a $N_{sysNode}\left(2N_{\omega}+1\right)\times N_{i}\left(2N_{\omega}+1\right)$ matrix, where N_i is the number of inputs to the system.

After solving for $\mathbf{A_{sys}}$ and $\mathbf{B_{sys}}$, system level transfer functions are solved for using (83). $\mathbf{x_{sys}}$ is a vector of length $N_{sysNode}\left(2N_{\omega}+1\right)$. Each $\left(k-1\right)\left(2N_{\omega}+1\right)+1$ entry is the dc value of the *k*th system node. Each $\left(k-1\right)\left(2N_{\omega}+1\right)+2h$ and $\left(k-1\right)\left(2N_{\omega}+1\right)+2h+1$ entry is the d-axis and q-axis value respectively of the *k*th system node at the *h*th harmonic.

The high level netlist differs from the low level system netlist in that a "through variable", such as current is not utilized. Each column in the system level netlist differs from the circuit netlist introduced in Figure 4.3. The columns for the system level netlist are: type, input node(s), output node, and value. Linear element types used in the high level system netlist are: input, add, constant, gain, integrate, 90° delay, to dq, from dq,

and circuit. Nonlinear element types used in the high level system netlist are: multiply and divide. Each of these elements is described in the following sections.

4.3.1. Input. An input row in the system level netlist is in the form

$$\begin{bmatrix}\text{IN} & 0 & N_{out} & U_{num}\end{bmatrix}, \tag{92}$$

where N_{out} is the node that the input connects to and U_{num} specifies the order that the input will appear in $\mathbf{u}_{\mathbf{sys}}$. The "0" column is a "don't care" column. An input row only affects the $\mathbf{B}_{\mathbf{sys}}$ matrix. The dc portion of $\mathbf{B}_{\mathbf{sys}}$ for an input is:

$$\mathbf{B}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\left(U_{num}-1\right)\left(2N_{\omega}+1\right)+1\right)=1. \tag{93}$$

The d-axis portion of $\mathbf{B}_{\mathbf{sys}}$ for an input is:

$$\mathbf{B}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\left(U_{num}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\right)=1. \tag{94}$$

The q-axis portion of $\mathbf{B}_{\mathbf{sys}}$ for an input is:

$$\mathbf{B}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\left(U_{num}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\right)=1, \tag{95}$$

where ℓ specifies the harmonic number of each input.

4.3.2. Add. An ADD row in the system level netlist is in the form

$$\begin{bmatrix}\text{ADD} & \begin{bmatrix}N_{in,1} & N_{in,2}\end{bmatrix} & N_{out} & \begin{bmatrix}Sgn_1 & Sgn_2\end{bmatrix}\end{bmatrix}. \tag{96}$$

$N_{in,1}$ and $N_{in,2}$ specifies the two input nodes to be added. N_{out} specifies the output node of the ADD block. Sgn_1 and Sgn_2 are either +1 or -1. These entries specify if its respective input node is added or subtracted.

ADD affects the $\mathbf{A}_{\mathbf{sys}}$ matrix. The dc portion of $\mathbf{A}_{\mathbf{sys}}$ for an ADD block is:

$$\begin{aligned}\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1\right)&=Sgn_1\\ \mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+1\right)&=Sgn_2\end{aligned} \tag{97}$$

The d-axis portion of $\mathbf{A}_{\mathbf{sys}}$ for an ADD block is:

$$\begin{aligned}\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\right)&=Sgn_1\\ \mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\right)&=Sgn_2\end{aligned}. \quad (98)$$

The q-axis portion of $\mathbf{A}_{\mathbf{sys}}$ for an ADD block is:

$$\begin{aligned}\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\right)&=Sgn_1\\ \mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\right)&=Sgn_2\end{aligned}, \quad (99)$$

where ℓ specifies the harmonic number.

4.3.3. Gain. A GAIN row in the system level netlist is in the form

$$\begin{bmatrix}\text{GAIN} & N_{in} & N_{out} & G\end{bmatrix}, \quad (100)$$

where N_{in} specifies the input node, N_{out} specifies the output node, and G specifies the gain value. The dc portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1\right)=G. \quad (101)$$

The d-axis portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\right)=G. \quad (102)$$

The q-axis portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\right)=G, \quad (103)$$

where ℓ specifies the harmonic number.

4.3.4. Integrate. An INT row is in the form

$$\begin{bmatrix}\text{INT} & N_{in} & N_{out} & 0\end{bmatrix}, \quad (104)$$

where N_{in} specifies the input node and N_{out} specifies the output node. Once again, the "0" column specifies a "don't care" column. The dc portion of $\mathbf{A}_{\mathbf{sys}}$ for an INT block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1\right)=\frac{1}{s}. \quad (105)$$

The d-axis input to d-axis output portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\right)=\frac{1}{s}. \quad (106)$$

The d-axis input to q-axis output portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\right)= -\frac{\ell\omega}{s} \quad (107)$$

The q-axis input to d-axis output portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\right)= \frac{\ell\omega}{s} \quad (108)$$

The q-axis input to q-axis output portion of $\mathbf{A}_{\mathbf{sys}}$ for a GAIN block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\right)=\frac{1}{s}, \quad (109)$$

where ℓ specifies the harmonic number.

4.3.5. 90° Delay. The delay block does not dealy magnitude of the signal, rather it causes a 90° phase shift by swapping and negating various d-axis and q-axis terms. If the harmonic being analyzed is odd, then the d-axis and q-axis values are swapped. If the harmonic being analyzed is even, then the d-axis and q-axis values are not swapped. To determine the sign of each element, each d-axis and q-axis value is multiplied by its corresponding value in Table 4.1. For values greater than the 4th harmonic, the values loop back around (i.e. the 5th harmonic uses 1st harmonic values, the 6th harmonic uses 2nd harmonic values, etc.). DC values are not affected by this modeling technique.

A DELAY row is in the form:

$$\left[\text{DELAY} \quad N_{in} \quad N_{out} \quad 0\right], \quad (110)$$

where N_{in} is the input node to the DELAY block and N_{out} is the output node from the DELAY block. The "0" column represents a "don't care" case.

The dc portion of $\mathbf{A}_{\mathbf{sys}}$ for a DELAY block is:

$$\mathbf{A}_{\mathbf{sys}}\left(\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1\right)=1 \tag{111}$$

The d-axis portion of $\mathbf{A}_{\mathbf{sys}}$ for a DELAY block is:

$$\begin{cases} \mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\end{pmatrix}=\mathrm{LUT}(1,\ell) & \text{if } \ell \text{ is even} \\ \mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\end{pmatrix}=\mathrm{LUT}(1,\ell) & \text{if } \ell \text{ is odd} \end{cases} \tag{112}$$

The q-axis portion of $\mathbf{A}_{\mathbf{sys}}$ for a DELAY block is:

$$\begin{cases} \mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3\end{pmatrix}=\mathrm{LUT}(2,\ell) & \text{if } \ell \text{ is even} \\ \mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2\end{pmatrix}=\mathrm{LUT}(2,\ell) & \text{if } \ell \text{ is odd} \end{cases} \tag{113}$$

where LUT is defined by Table 4.1 and ℓ specifies the harmonic number.

Table 4.1. DQ multiplier for delay 90°.

	Harmonic			
Harmonic	1st	2nd	3rd	4th
d-axis:	-1	-1	1	1
q-axis:	1	-1	-1	1

4.3.6. To DQ. The TODQ block is a single phase DQ conversion, utilizing a 90° delay. The conversion used is

$$\begin{bmatrix} y_d \\ y_q \end{bmatrix} = \begin{bmatrix} \cos(\omega t) & \sin(\omega t) \\ -\sin(\omega t) & \cos(\omega t) \end{bmatrix} \begin{bmatrix} x \\ x_{delay} \end{bmatrix}. \tag{114}$$

The dc amplitude at the d-axis and q-axis output is

$$\left(y_d\right)_{dc} = \frac{x_{d1\omega}}{2} - \frac{x_{delayq1\omega}}{2} \tag{115}$$

and

$$\left(y_q\right)_{dc} = \frac{x_{delayd1\omega}}{2} + \frac{x_{q1\omega}}{2}. \tag{116}$$

The d-axis and q-axis amplitude of y_d at the ℓth harmonic is

$$\left(y_d\right)_{d\ell\omega} = \begin{cases} x_{dc} + \dfrac{x_{d(\ell+1)\omega}}{2} - \dfrac{x_{delayq(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \dfrac{x_{d(\ell-1)\omega}}{2} + \dfrac{x_{delayq(\ell-1)\omega}}{2} + \dfrac{x_{d(\ell+1)\omega}}{2} - \dfrac{x_{delayq(\ell+1)\omega}}{2} & \text{otherwise} \end{cases} \tag{117}$$

and

$$\left(y_d\right)_{q\ell\omega} = \begin{cases} -x_{delaydc} + \dfrac{x_{delayd(\ell+1)\omega}}{2} + \dfrac{x_{q(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ -\dfrac{x_{delayd(\ell-1)\omega}}{2} + \dfrac{x_{q(\ell-1)\omega}}{2} + \dfrac{x_{delayd(\ell+1)\omega}}{2} + \dfrac{x_{q(\ell+1)\omega}}{2} & \text{otherwise} \end{cases}. \tag{118}$$

The d-axis and q-axis amplitude of y_q at the ℓth harmonic is

$$\left(y_q\right)_{d\ell\omega} = \begin{cases} x_{delaydc} + \dfrac{x_{delayd(\ell+1)\omega}}{2} + \dfrac{x_{q(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \dfrac{x_{delayd(\ell-1)\omega}}{2} - \dfrac{x_{q(\ell-1)\omega}}{2} + \dfrac{x_{delayd(\ell+1)\omega}}{2} + \dfrac{x_{q(\ell+1)\omega}}{2} & \text{otherwise} \end{cases} \tag{119}$$

and

$$\left(y_q\right)_{q\ell\omega}=\begin{cases} x_{dc}-\dfrac{x_{d(\ell+1)\omega}}{2}+\dfrac{x_{delayq(\ell+1)\omega}}{2} & \text{if } \ell=1 \\ \dfrac{x_{d(\ell-1)\omega}}{2}+\dfrac{x_{q(\ell-1)\omega}}{2}-\dfrac{x_{d(\ell+1)\omega}}{2}+\dfrac{x_{delayq(\ell+1)\omega}}{2} & \text{otherwise} \end{cases}. \tag{120}$$

The DQ sections can be confusing, due to equations for d-axis and q-axis values being broken up into a Fourier series representation. Recall that every variable in this study is represented as a Fourier series, even the d-axis and q-axis current values inside the controller. This enables modeling controller concepts (like delay, to DQ, and from DQ) that cannot be conventionally modeled.

After generating a set of equations from the algorithms in (115)-(120), the Jacobian of these set of equations is found. Since TODQ is a linear operation, it is not necessary to use the Jacobian, but it is still a valid method to populate $\mathbf{A_{sys}}$. For this study, **TodqJacob** is the Jacobian matrix of the TODQ operation described above.

A TODQ row is in the form:

$$\left[\text{TODQ} \quad \left[N_{in} \quad N_{in,delay}\right] \quad \left[N_{out,d} \quad N_{out,q}\right] \quad 0\right]. \tag{121}$$

N_{in} is the input signal to be converted to the dq reference frame. $N_{in,delay}$ is the N_{in} signal delayed 90°. $N_{out,d}$ and $N_{out,q}$ are the d-axis output and q-axis output respectively. The "0" column is a "don't care" column.

The dc term of the d-axis output of $\mathbf{A_{sys}}$, due to N_{in} is:

$$\mathbf{A_{sys}}\begin{pmatrix}\left(N_{out,d}-1\right)\left(2N_\omega+1\right)+1,\cdots \\ \left(N_{in}-1\right)\left(2N_\omega+1\right)+1:\left(N_{in}-1\right)\left(2N_\omega+1\right)+1+2N_\omega\end{pmatrix}= \\ \mathbf{TodqJacob}\left(1,1:2N_\omega+1\right) \tag{122}$$

The dc term of the d-axis output of $\mathbf{A_{sys}}$, due to $N_{in,delay}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out,d}-1\right)\left(2N_{\omega}+1\right)+1,\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{TodqJacob}\left(1,2N_{\omega}+2:4N_{\omega}+2\right)\quad(123)$$

The dc term of the q-axis output of $\mathbf{A}_{\mathbf{sys}}$, due to N_{in} is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out,q}-1\right)\left(2N_{\omega}+1\right)+1,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{TodqJacob}\left(2N_{\omega}+2,1:2N_{\omega}+1\right)\quad(124)$$

The dc term of the q-axis output of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,delay}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out,q}-1\right)\left(2N_{\omega}+1\right)+1,\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{TodqJacob}\left(2N_{\omega}+2,2N_{\omega}+2:4N_{\omega}+2\right)\quad(125)$$

The d-axis term of the d-axis output of $\mathbf{A}_{\mathbf{sys}}$, due to N_{in} is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out,d}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{TodqJacob}\left(2\ell,1:2N_{\omega}+1\right)\quad(126)$$

The d-axis term of the d-axis output of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,delay}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out,d}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{TodqJacob}\left(2\ell,2N_{\omega}+2:4N_{\omega}+2\right)\quad(127)$$

The q-axis term of the d-axis output of $\mathbf{A}_{\mathbf{sys}}$, due to N_{in} is:

$$\mathbf{A_{sys}}\begin{pmatrix}\left(N_{out,d}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{TodqJacob}\left(2\ell+1,1:2N_{\omega}+1\right) \quad (128)$$

The q-axis term of the d-axis output of $\mathbf{A_{sys}}$, due to $N_{in,delay}$ is:

$$\mathbf{A_{sys}}\begin{pmatrix}\left(N_{out,d}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}= \quad (129)$$
$$\mathbf{TodqJacob}\left(2\ell+1,2N_{\omega}+2:4N_{\omega}+2\right)$$

The d-axis term of the q-axis output of $\mathbf{A_{sys}}$, due to N_{in} is:

$$\mathbf{A_{sys}}\begin{pmatrix}\left(N_{out,q}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}= \quad (130)$$
$$\mathbf{TodqJacob}\left(2N_{\omega}+1+2\ell,1:2N_{\omega}+1\right)$$

The d-axis term of the q-axis output of $\mathbf{A_{sys}}$, due to $N_{in,delay}$ is:

$$\mathbf{A_{sys}}\begin{pmatrix}\left(N_{out,q}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}= \quad (131)$$
$$\mathbf{TodqJacob}\left(2N_{\omega}+1+2\ell,2N_{\omega}+2:4N_{\omega}+2\right)$$

The q-axis term of the q-axis output of $\mathbf{A_{sys}}$, due to N_{in} is:

$$\mathbf{A_{sys}}\begin{pmatrix}\left(N_{out,q}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}= \quad (132)$$
$$\mathbf{TodqJacob}\left(2N_{\omega}+2\ell+2,1:2N_{\omega}+1\right)$$

The q-axis term of the q-axis output of $\mathbf{A_{sys}}$, due to $N_{in,delay}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out,q}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,delay}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}= \\ \mathbf{TodqJacob}\left(2N_{\omega}+2\ell+2,2N_{\omega}+2:4N_{\omega}+2\right) \tag{133}$$

where ℓ specifies the harmonic number. The ":" operator is used to specify a range of values in a matrix. In the TODQ case, the first half of columns in **TodqJacob** correspond to the N_{in} node input and the second half of columns in **TodqJacob** correspond to the $N_{in,delay}$ node input.

4.3.7. From DQ. Conversion from DQ is simple:

$$y = x_d \cos\left(\omega t\right) - x_q \sin\left(\omega t\right). \tag{134}$$

The dc component of y is

$$y_{dc} = \frac{\left(x_d\right)_{d1\omega}}{2} + \frac{\left(x_q\right)_{q1\omega}}{2} \tag{135}$$

The d-axis and q-axis amplitude of y at the ℓth harmonic is

$$y_d = \begin{cases} \left(x_d\right)_{dc} + \dfrac{\left(x_d\right)_{d(\ell+1)\omega}}{2} + \dfrac{\left(x_q\right)_{q(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \dfrac{\left(x_d\right)_{d(\ell-1)\omega}}{2} - \dfrac{\left(x_q\right)_{q(\ell-1)\omega}}{2} + \dfrac{\left(x_d\right)_{d(\ell+1)\omega}}{2} + \dfrac{\left(x_q\right)_{q(\ell+1)\omega}}{2} & \text{otherwise} \end{cases} \tag{136}$$

and

$$y_q = \begin{cases} -\left(x_q\right)_{dc} + \dfrac{\left(x_d\right)_{q(\ell+1)\omega}}{2} - \dfrac{\left(x_q\right)_{d(\ell+1)\omega}}{2} & \text{if } \ell = 1 \\ \dfrac{\left(x_d\right)_{q(\ell-1)\omega}}{2} + \dfrac{\left(x_q\right)_{d(\ell-1)\omega}}{2} + \dfrac{\left(x_d\right)_{q(\ell+1)\omega}}{2} - \dfrac{\left(x_q\right)_{d(\ell+1)\omega}}{2} & \text{otherwise} \end{cases}. \tag{137}$$

After generating a set of equations from the algorithms in (135)-(137), the Jacobian of these set of equations is found. Since FROMDQ is a linear operation, it is

not necessary to use the Jacobian, but it is still a valid method to populate $\mathbf{A}_{\mathbf{sys}}$. For this study, **FromdqJacob** is the Jacobian matrix of the FROMDQ operation described above.

A FROMDQ row is in the form:

$$\begin{bmatrix} \text{FROMDQ} & \begin{bmatrix} N_{in,d} & N_{in,q} \end{bmatrix} & N_{out} & 0 \end{bmatrix}, \tag{138}$$

where $N_{in,d}$ is the d-axis input, $N_{in,q}$ is the q-axis input, and N_{out} is the output.

The dc term of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,d}$ is:

$$\mathbf{A}_{\mathbf{sys}} \begin{pmatrix} \left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\cdots \\ \left(N_{in,d}-1\right)\left(2N_{\omega}+1\right)+1:\cdots \\ \left(N_{in,d}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega} \end{pmatrix} = \mathbf{FromdqJacob}\left(1,1:2N_{\omega}+1\right) \tag{139}$$

The dc term of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,q}$ is:

$$\mathbf{A}_{\mathbf{sys}} \begin{pmatrix} \left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\cdots \\ \left(N_{in,q}-1\right)\left(2N_{\omega}+1\right)+1:\cdots \\ \left(N_{in,q}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega} \end{pmatrix} = \mathbf{FromdqJacob}\left(1,2N_{\omega}+2:4N_{\omega}+2\right) \tag{140}$$

The d-axis term of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,d}$ is:

$$\mathbf{A}_{\mathbf{sys}} \begin{pmatrix} \left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots \\ \left(N_{in,d}-1\right)\left(2N_{\omega}+1\right)+1:\cdots \\ \left(N_{in,d}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega} \end{pmatrix} = \mathbf{FromdqJacob}\left(2\ell,1:2N_{\omega}+1\right) \tag{141}$$

The d-axis term of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,q}$ is:

$$\mathbf{A}_{\mathbf{sys}} \begin{pmatrix} \left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots \\ \left(N_{in,q}-1\right)\left(2N_{\omega}+1\right)+1:\cdots \\ \left(N_{in,q}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega} \end{pmatrix} = \mathbf{FromdqJacob}\left(2\ell,2N_{\omega}+2:4N_{\omega}+2\right) \tag{142}$$

The q-axis term of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,d}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+3,\cdots\\(N_{in,d}-1)(2N_{\omega}+1)+1:\cdots\\(N_{in,d}-1)(2N_{\omega}+1)+1+2N_{\omega}\end{pmatrix}= \\ \mathbf{FromdqJacob}(2\ell+1,1:2N_{\omega}+1) \tag{143}$$

The q-axis term of $\mathbf{A}_{\mathbf{sys}}$, due to $N_{in,q}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+3,\cdots\\(N_{in,q}-1)(2N_{\omega}+1)+1:\cdots\\(N_{in,q}-1)(2N_{\omega}+1)+1+2N_{\omega}\end{pmatrix}= \\ \mathbf{FromdqJacob}(2\ell+1,2N_{\omega}+2:4N_{\omega}+2) \tag{144}$$

where ℓ specifies the harmonic number. In the FROMDQ case, the first half of columns in **FromdqJacob** correspond to the $N_{in,d}$ node input and the second half of columns in **FromdqJacob** correspond to the $N_{in,q}$ node input.

4.3.8. Circuit. A CKT row is in the form

$$\left[\text{CKT}\quad\left[N_{in,1}\quad N_{in,2}\quad\ldots\right]\quad\left[N_{out,1}\quad N_{out,2}\quad\ldots\right]\quad CktNum\right]. \tag{145}$$

$N_{in,k}$ specifies the system level node connected to the *k*th input of the circuit. $N_{out,k}$ specifies the system level node connected to the *k*th branch's current in the circuit specified by *CktNum* (if $k \le b$). If $k > b$, then $N_{out,k}$ specifies the system level node connected to the *(k-b)*th branch's voltage in the circuit specified by *CktNum.* The dc and dq circuit level TF's generated from Section 4.2 are placed in variables T_{dc} and T_{dq} respectively.

The dc portion of $\mathbf{A}_{\mathbf{sys}}$ for a CKT block is:

$$\mathbf{A}_{\mathbf{sys}}\left((N_{out}-1)(2N_{\omega}+1)+1,(N_{in}-1)(2N_{\omega}+1)+1\right)=T_{dc}\left(N_{out},N_{in}\right), \tag{146}$$

for each N_{in} and N_{out} specified in the CKT row of the netlist. Like the INT block, the CKT block also contains cross coupling terms between the d and q-axis components. The d-axis input to d-axis output component of the CKT block is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+2,\cdots\\(N_{in}-1)(2N_{\omega}+1)+2(\ell-1)+2\end{pmatrix}= T_{dq}\begin{pmatrix}2N_{\omega}(N_{out}-1)+2(\ell-1)+1,\cdots\\2N_{\omega}(N_{in}-1)+2(\ell-1)+1\end{pmatrix}. \tag{147}$$

The d-axis input to q-axis output component of the CKT block is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+3,\cdots\\(N_{in}-1)(2N_{\omega}+1)+2(\ell-1)+2\end{pmatrix}= T_{dq}\begin{pmatrix}2N_{\omega}(N_{out}-1)+2(\ell-1)+2,\cdots\\2N_{\omega}(N_{in}-1)+2(\ell-1)+1\end{pmatrix}. \tag{148}$$

The q-axis input to d-axis output component of the CKT block is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+2,\cdots\\(N_{in}-1)(2N_{\omega}+1)+2(\ell-1)+3\end{pmatrix}= T_{dq}\begin{pmatrix}2N_{\omega}(N_{out}-1)+2(\ell-1)+1,\cdots\\2N_{\omega}(N_{in}-1)+2(\ell-1)+2\end{pmatrix}. \tag{149}$$

The q-axis input to q-axis output component of the CKT block is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+3,\cdots\\(N_{in}-1)(2N_{\omega}+1)+2(\ell-1)+3\end{pmatrix}= T_{dq}\begin{pmatrix}2N_{\omega}(N_{out}-1)+2(\ell-1)+2,\cdots\\2N_{\omega}(N_{in}-1)+2(\ell-1)+2\end{pmatrix}. \tag{150}$$

The dq-axis components occur for each N_{in} and N_{out} listed in that CKT block's netlist, as well as every harmonic, ℓ.

4.3.9. Multiplication. Finding the Fourier series terms at the ℓth harmonic has been previously found [73, 74]. It is shown here because of the different notation used in this study. When multiplying two signals (a and b) represented as a Fourier series,

$$(ab) = \left(a_{dc} + \sum_{k=1}^{k \le N_{\max}} \left[a_{dk\omega} \cos(\omega k t) - a_{qk\omega} \sin(\omega k t) \right] \right) \times \left(b_{dc} + \sum_{k=1}^{k \le N_{\max}} \left[b_{dk\omega} \cos(\omega k t) - b_{qk\omega} \sin(\omega k t) \right] \right), \tag{151}$$

the d-axis amplitude and q-axis amplitude need to be found, at each frequency. The dc amplitude is

$$(ab)_{dc} = a_{dc} b_{dc} + \sum_{k=1}^{k \le N_{\max}} \left[\frac{a_{dk\omega} b_{dk\omega}}{2} + \frac{a_{qk\omega} b_{qk\omega}}{2} \right]. \tag{152}$$

The d-axis ℓth harmonic amplitude is

$$(ab)_{d\ell\omega} = a_{d\ell\omega} b_{dc} + a_{dc} b_{d\ell\omega} + \frac{1}{2} \sum_{k=1}^{k<\ell} \left[a_{d(\ell-k)\omega} b_{dk\omega} - a_{q(\ell-k)\omega} b_{qk\omega} \right] + \frac{1}{2} \sum_{k=\ell+1}^{k \le N_{\max}} \left[a_{dk\omega} b_{d(k-\ell)\omega} + a_{qk\omega} b_{q(k-\ell)\omega} + a_{d(k-\ell)\omega} b_{dk\omega} + a_{q(k-\ell)\omega} b_{qk\omega} \right] \tag{153}$$

and the q-axis ℓth harmonic amplitude is

$$(ab)_{q\ell\omega} = a_{q\ell\omega} b_{dc} + a_{dc} b_{q\ell\omega} + \frac{1}{2} \sum_{k=1}^{k<\ell} \left[a_{qk\omega} b_{d(\ell-k)\omega} + a_{dk\omega} b_{q(\ell-k)\omega} \right] + \frac{1}{2} \sum_{k=\ell+1}^{k \le N_{\max}} \left[-a_{dk\omega} b_{q(k-\ell)\omega} + a_{qk\omega} b_{d(k-\ell)\omega} + a_{d(k-\ell)\omega} b_{qk\omega} - a_{q(k-\ell)\omega} b_{dk\omega} \right]. \tag{154}$$

After generating a set of equations from the algorithms in (50)-(53), the Jacobian of these set of equations is found. Values used in the resulting Jacobian matrix are the linearization points of the system. Linearization points must be known for each input node connected to a nonlinear block (MULT and DIV in this study). For this study, **MultJacob** is the Jacobian matrix of the multiplication operation described above.

A MULT row is in the form:

$$\begin{bmatrix}\text{MULT} & \begin{bmatrix}N_{in,1} & N_{in,2}\end{bmatrix} & N_{out} & 0\end{bmatrix}, \tag{155}$$

where $N_{in,k}$ specifies the input nodes to the MULT block. N_{out} specifies the output node of the MULT block. The dc component of $\mathbf{A}_{\mathbf{sys}}$ due to node $N_{in,1}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\cdots\\ \left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{MultJacob}\left(1,1:2N_{\omega}+1\right) \tag{156}$$

The dc component of $\mathbf{A}_{\mathbf{sys}}$ due to node $N_{in,2}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+1,\cdots\\ \left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{MultJacob}\left(1,2N_{\omega}+2:4N_{\omega}+2\right) \tag{157}$$

The d-axis component of $\mathbf{A}_{\mathbf{sys}}$ due to node $N_{in,1}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{MultJacob}\left(2\ell,1:2N_{\omega}+1\right) \tag{158}$$

The d-axis component of $\mathbf{A}_{\mathbf{sys}}$ due to node $N_{in,2}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+2,\cdots\\ \left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,2}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}= \mathbf{MultJacob}\left(2\ell,2N_{\omega}+2:4N_{\omega}+2\right) \tag{159}$$

The q-axis component of $\mathbf{A}_{\mathbf{sys}}$ due to node $N_{in,1}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}\left(N_{out}-1\right)\left(2N_{\omega}+1\right)+2\left(\ell-1\right)+3,\cdots\\ \left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1:\cdots\\ \left(N_{in,1}-1\right)\left(2N_{\omega}+1\right)+1+2N_{\omega}\end{pmatrix}=\mathbf{MultJacob}\left(2\ell+1,1:2N_{\omega}+1\right) \tag{160}$$

The q-axis component of $\mathbf{A}_{\mathbf{sys}}$ due to node $N_{in,2}$ is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+3,\cdots\\(N_{in,2}-1)(2N_{\omega}+1)+1:\cdots\\(N_{in,2}-1)(2N_{\omega}+1)+1+2N_{\omega}\end{pmatrix}= \mathbf{MultJacob}\left(2\ell+1,2N_{\omega}+2:4N_{\omega}+2\right) \tag{161}$$

where ℓ specifies the harmonic number of each input. In the MULT case, the first half of columns in **MultJacob** correspond to the $N_{in,1}$ node input and the second half of columns in **MultJacob** correspond to the $N_{in,2}$ node input.

4.3.10. Inversion. Inversion is not as straightforward as multiplication. Assume the signal *a* is to be inverted and the solution is represented by *b*. If *a* is a periodic signal, then it can be represented as a sum of sinusoids. Assuming $a_{dc}>>|a_{ac}|$, then *b* can be approximated by a finite sum of sinusoids:

$$\frac{1}{a_{dc}+\sum_{k=1}^{N_{\max}}\left[a_{dk\omega}\cos(\omega kt)-a_{qk\omega}\sin(\omega kt)\right]}\approx \frac{}{b_{dc}+\sum_{k=1}^{N_{\max}}\left[b_{dk\omega}\cos(\omega kt)-b_{qk\omega}\sin(\omega kt)\right]}. \tag{162}$$

Multiplying both sides by *a* yields

$$1\approx\left(a_{dc}+\sum_{k=1}^{N_{\max}}\left[a_{dk\omega}\cos(\omega kt)-a_{qk\omega}\sin(\omega kt)\right]\right)\times \left(b_{dc}+\sum_{k=1}^{N_{\max}}\left[b_{dk\omega}\cos(\omega kt)-b_{qk\omega}\sin(\omega kt)\right]\right), \tag{163}$$

which was examined in the previous section. The dc component of the product in (55) must be equal to 1. All d-axis and q-axis harmonic components must be equal to 0. This gives a set of $(2N_{max}+1)$ equations with $(2N_{max}+1)$ unknowns.

To solve for *b* programmatically, a coefficient matrix ($\mathbf{A_{coeff}}$) is be generated. In the coefficient matrix, each component of *b* is an unknown. Using (51), (52), and (53), the coefficients of *b* can be determined in terms of *a* for each harmonic. Placing the augmented $\mathbf{A_{aug}}$ matrix,

$$\mathbf{A_{aug}} = \begin{array}{c} \\ dc \rightarrow \\ 1\omega_d \rightarrow \\ 1\omega_q \rightarrow \\ \vdots \end{array} \begin{array}{c} \begin{array}{cccc} b_{dc} & b_{d1\omega} & b_{q1\omega} & \cdots \\ \downarrow & \downarrow & \downarrow & \end{array} \\ \left[\begin{array}{cccc|c} \square & \square & \square & & 1 \\ \square & \square & \square & \cdots & 0 \\ \square & \square & \square & & \vdots \\ & \vdots & & \ddots & 0 \end{array}\right] \end{array}, \tag{164}$$

in reduced row echelon form will result in the last column containing the desired terms for *b*.

After generating a set of equations from the algorithms in (162)-(164), the Jacobian of these set of equations is found. Values used in the resulting Jacobian matrix are the linearization points of the system. For this study, **InvJacob** is the Jacobian matrix of the inverse operation described above.

An INV row is in the form:

$$\begin{bmatrix} \text{INV} & N_{in} & N_{out} & 0 \end{bmatrix}, \tag{165}$$

where N_{in} specifies the input node to the INV block and N_{out} specifies the output node from the INV block. The "0" column is a "don't case" case. The dc component of $\mathbf{A_{sys}}$ for an INV block is:

$$\mathbf{A_{sys}} \begin{pmatrix} (N_{out}-1)(2N_{\omega}+1)+1, \cdots \\ (N_{in}-1)(2N_{\omega}+1)+1 : \cdots \\ (N_{in}-1)(2N_{\omega}+1)+1+2N_{\omega} \end{pmatrix} = \mathbf{InvJacob}\left(1, 1 : 2N_{\omega}+1\right) \tag{166}$$

The d-axis component of $\mathbf{A_{sys}}$ for an INV block is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+2,\cdots\\(N_{in,1}-1)(2N_{\omega}+1)+1:\cdots\\(N_{in,1}-1)(2N_{\omega}+1)+1+2N_{\omega}\end{pmatrix}=\mathbf{InvJacob}(2\ell,1:2N_{\omega}+1) \quad (167)$$

The q-axis component of $\mathbf{A}_{\mathbf{sys}}$ for an INV block is:

$$\mathbf{A}_{\mathbf{sys}}\begin{pmatrix}(N_{out}-1)(2N_{\omega}+1)+2(\ell-1)+3,\cdots\\(N_{in,1}-1)(2N_{\omega}+1)+1:\cdots\\(N_{in,1}-1)(2N_{\omega}+1)+1+2N_{\omega}\end{pmatrix}=\mathbf{InvJacob}(2\ell+1,1:2N_{\omega}+1) \quad (168)$$

where ℓ specifies the harmonic number of each input.

4.4. GENERATING EQUATIONS FOR THE INVERTER

In this section, truncated harmonic transfer functions are generated for the single phase grid tied H-bridge inverter illustrated in Figure 4.9. The control objective of the inverter is to maintain a constant dc-link voltage and output a sinusoidal grid current. Figure 4.10 illustrates the dq current controller used in the inverter. Node numbers are indicated by the circled numbers. Proportional integral (PI) controllers are used to regulate the dc-link voltage, the d-axis current and the q-axis current.

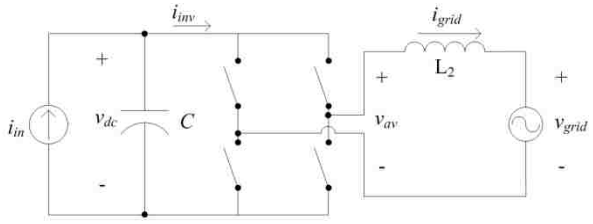

Figure 4.9. Inverter example circuit.

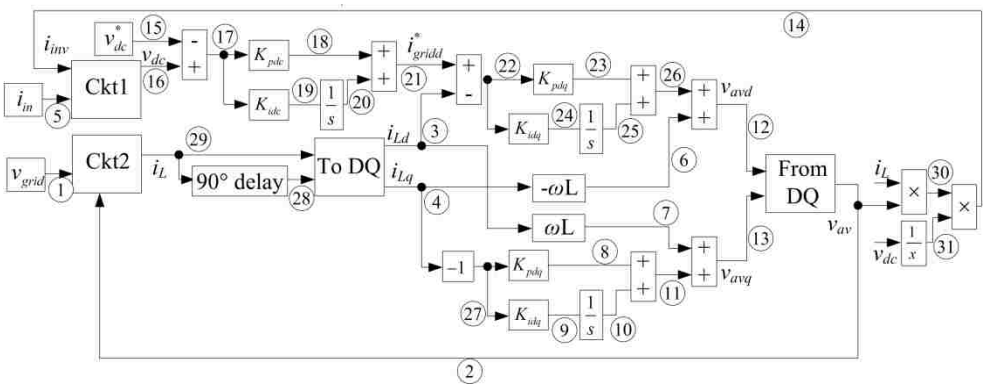

Figure 4.10. Inverter controller.

To model this circuit using the proposed method, the circuit in Figure 4.9 must first be split into two (average model) circuits, illustrated in Figure 4.11. This is required because the circuit level v_{av} is generated from controls at the system level.

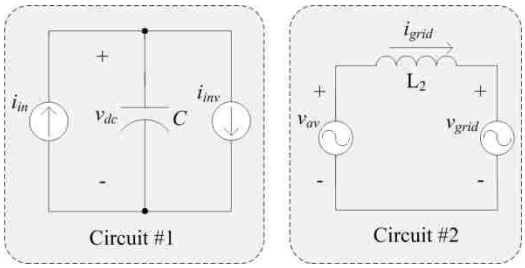

Figure 4.11. Averaged model inverter circuit.

The transfer functions describing the circuits in Figure 4.11 are generated by the low level circuit netlist from Section 4.2. The overall system equations are generated using the system level netlist, the controls illustrated in Figure 4.10, and the transfer functions generated from the low level circuit netlist.

The circuit level netlists are

$$CktNet1=\begin{bmatrix} \text{I} & 1 & 2 & 1 \\ \text{CAP} & 1 & 2 & 680e-6 \\ \text{I} & 2 & 1 & 2 \end{bmatrix} \tag{169}$$

and

$$CktNet2=\begin{bmatrix} \text{V} & 1 & 3 & 1 \\ \text{L} & 1 & 2 & 8.4e-3 \\ \text{V} & 2 & 3 & 2 \end{bmatrix}. \tag{170}$$

The system level netlist is generated from Figure 4.10:

$$SysNet = \begin{bmatrix} \text{IN} & 0 & 5 & 1 \\ \text{CKT} & [2\ \ 1] & [0\ \ 29\ \ 0\ \ 0\ \ 0\ \ 0] & 2 \\ \text{IN} & 0 & 1 & 3 \\ \text{DELAY} & 29 & 28 & 0 \\ \text{TODQ} & [29\ \ 28] & [3\ \ 4] & 0 \\ \text{GAIN} & 4 & 6 & -3.1667 \\ \text{GAIN} & 3 & 7 & 3.1667 \\ \text{GAIN} & 4 & 27 & -1 \\ \text{GAIN} & 27 & 8 & 2.509 \\ \text{GAIN} & 27 & 9 & 10 \\ \text{INT} & 9 & 10 & 0 \\ \text{ADD} & [8\ \ 10] & 11 & [1\ \ 1] \\ \text{ADD} & [11\ \ 7] & 13 & [1\ \ 1] \\ \text{CKT} & [5\ \ 14] & [0\ \ 0\ \ 0\ \ 0\ \ 16\ \ 0] & 1 \\ \text{IN} & 0 & 15 & 2 \\ \text{ADD} & [15\ \ 16] & 17 & [-1\ \ 1] \\ \text{GAIN} & 17 & 18 & 0.3488 \\ \text{GAIN} & 17 & 19 & 10 \\ \text{INT} & 19 & 20 & 0 \\ \text{ADD} & [18\ \ 20] & 21 & [1\ \ 1] \\ \text{ADD} & [21\ \ 3] & 22 & [1\ \ -1] \\ \text{GAIN} & 22 & 23 & 2.509 \\ \text{GAIN} & 22 & 24 & 10 \\ \text{INT} & 24 & 25 & 0 \\ \text{ADD} & [23\ \ 25] & 26 & [1\ \ 1] \\ \text{ADD} & [26\ \ 6] & 12 & [1\ \ 1] \\ \text{FROMDQ} & [12\ \ 13] & 2 & 0 \\ \text{MULT} & [2\ \ 29] & 30 & 0 \\ \text{MULT} & [30\ \ 31] & 14 & 0 \\ \text{INV} & 16 & 31 & 0 \end{bmatrix}. \quad (171)$$

Using the algorithms presented in Sections 4.2 and 4.3, the netlists from (169), (170), and (171) can be used to populate $\mathbf{A}_{\mathbf{sys}}$ and $\mathbf{B}_{\mathbf{sys}}$. The TF's from each input to each system node can then be solved for using (83).

4.5. RESULTS

The generated transfer functions provide multiple uses for analysis. This section will demonstrate the TF's ability to predict the effect of the 120 Hz dc-link ripple on the grid current. To do this, the generated i_{Ld}/v_{dcdc} TF is analyzed for three cases, shown in Table 4.2. These results are compared against and experimental setup.

The experimental setup is illustrated in Figure 4.12. In this study, C=680 uF and L=8.4 mH. A 1.33 A constant input was generated using a 100 V DC source in series with a BK Precision 8500 programmable dc electronic load. The electronic load was set to regulate a constant 1.33 A. The control system from Figure 4.10 was implemented using a TI TMS320F28335 DSP and programmed using Code Composer Studio.

Table 4.2. Inverter Gains.

	Case	Case	Case
K_{pdc}	0.3488	0.7189	1.001
K_{idc}	10	50	100
K_{pi}	2.509	4.407	5.398
K_{ii}	10	50	100

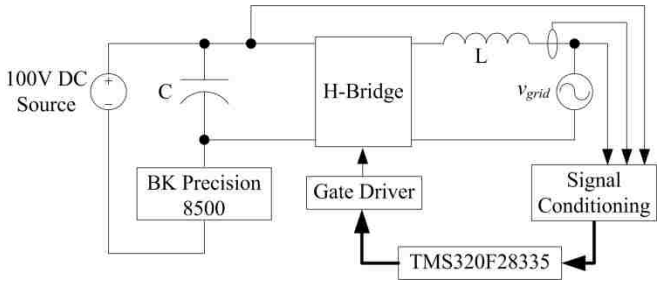

Figure 4.12. Experimental setup.

The automatically generated i_{Ld}/v_{dcdc} TFs for the three cases are illustrated in Figure 4.13. The measured i_{grid} current from the experimental setup is illustrated in Figure 4.14 for each case.

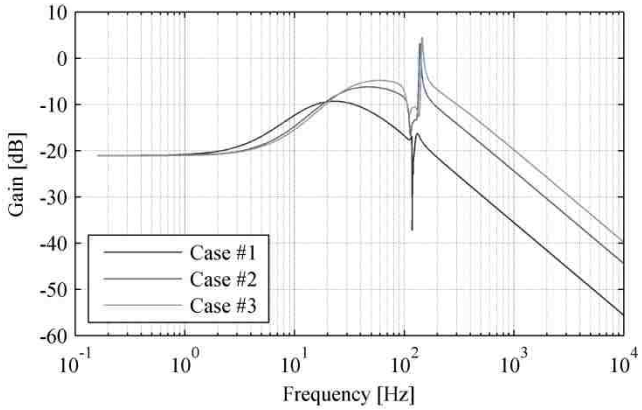

Figure 4.13. Commanded dc-link voltage to d-axis grid current TF.

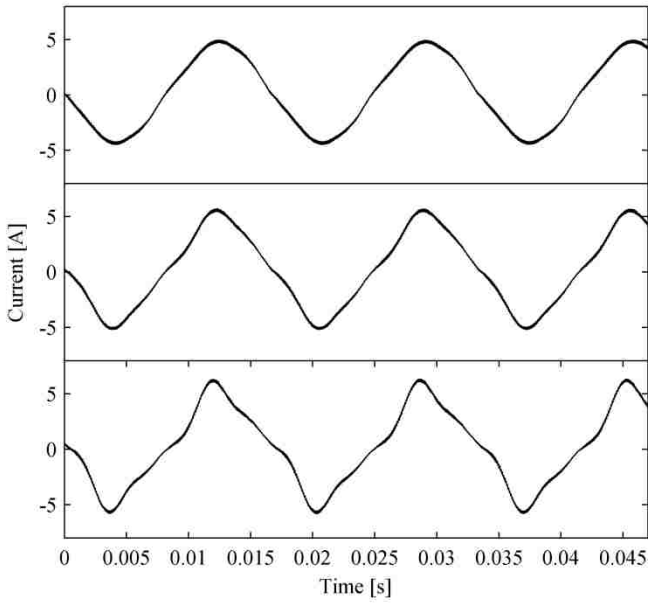

Figure 4.14. Experimental results for i_{grid}. Top: i_{grid} Case #1 Middle: i_{grid} Case #2 Bottom: i_{grid} Case #3.

4.6. CONCLUSIONS

This study proposed a new method to automatically generate transfer functions for a single phase inverter. Algorithms were presented, which converted user generated netlists representing low level circuits into TFs. These low level circuits consisted of inductors, capacitors, resistors, voltage sources. Next, more algorithms were presented, which converted a user generated netlist representing the overall system into TFs. The overall system netlist consisted of the following blocks: input, add, constant, gain, integrate, 90° delay, to dq, from dq, circuit, multiply, and divide.

Using these algorithms, a set of TFs were derived. Section 4.5 illustrated the i_{Ld}/v_{dcdc} transfer function for the three cases in Table 4.2. Case 1 uses low controller gains, case 2 uses medium controller gains, and case 3 uses high controller gains. The results illustrated in Figure 4.13 predict that for increasing controller gains, i_{Ld} will have

less attenuation from the dc-link voltage at 120 Hz. This means that the dc-link ripple will have a stronger effect on case 3 than it does on case 1.

The results illustrated in Figure 4.14 clearly demonstrate what the generated TFs predicted. Increasing controller gains increases distortion on i_{Ld}, due to the dc-link voltage ripple. The THD of the three cases is 5.23%, 12.8%, and 22.2%, respectively.

While the proposed method was able to successfully predict the effects of the 120 Hz dc-link ripple on the system, it does suffer from some inaccuracies. This is due to inaccuracies introduced from attempting to directly generate TFs for a large system. This is apparent in the TFs Bode plot in Figure 4.13. At 120 Hz, there are clearly accuracy issues with the generated transfer functions.

A more accurate method would be to automatically generate the state space representation of the system. This method would not suffer from the accuracy issues described and would not require the use of a symbolic math solver. It would also open the door to even more LTI analysis and design techniques.

5. SUMMARY AND CONCLUSIONS

5.1. OVERVIEW

Section 2 presented a new feedforward method to mitigate the adverse effects of the dc-link voltage double frequency ripple on the converter. A reference frame operating at the double frequency was used to predict the double frequency voltage ripple on the dc-link. By using this predicted value as a feedforward term, the effects of the double frequency ripple were eliminated from the dc-link voltage loop. The same methodology was applied to the MPPT, where a feedforward term was added to the duty ratio generated by the MPPT, to eliminate the double frequency ripple seen at the input power waveform. The simulated results were verified experimentally.

Many methods exist to allow for small film capacitors to be used on the dc-link on a microinverter. Most of these require extra components to existing designs. Since the presented feedforward method allows the converter to withstand a large ripple on the dc-link, reliable film capacitors can be used on the dc-link while requiring only software changes to existing designs. This allows for old designs to be used in new inverters with minimal changes.

The system used in Section 2 used a simple boost converter as the MPPT and a H-bridge as the inverter. The presented feedforward method is not restricted to a boost converter; it could be easily modified and applied to other MPPT topologies. By following the process laid out in Section 2.4, one could derive the feedforward term required to remove the double frequency component from their MPPT.

Section 3 presented a set of algorithms that can be used to predict the periodic steady state operating point of a system. The user must generate a set of large-signal

dynamic equations for the system being analyzed. Using the presented functions, these standard equations are converted to harmonic equations, which represent the amplitude of a signal at the specified frequencies. With the presented solver, the nonlinear set of equations are solved for. The resulting solution is the steady-state operating point of the converter.

Typically, finding the Fourier series coefficients of a systems output requires a simulation to be made and run until well after the steady state operating point reached. After running the simulation, the fast Fourier transform of a large amount of data must be computed to find the Fourier coefficients of interest. The presented work gives the user the ability to directly calculate the Fourier series coefficients of the steady state output.

Section 4 presented another algorithm which takes a user generated netlist and generates the system's THTF. Typically the frequency transfer properties are lost between the dc-link and ac side of an inverter in linearized models. The automatically generated THTF is a large set of linear transfer functions which preserves the frequency transfer properties. This gives a designer a tool which can be used to design detailed linear microinverter models.

The preservation of the frequency transfer properties is especially important in microinverters, where the dc-link capacitor is small. The small dc-link capacitor may result in a large dc-link voltage ripple, such as discussed in Section 2. Using the proposed method of generating a THTF, the transient effects of having a small dc-link capacitor can be accurately predicted.

Another advantage of the proposed THTF generator is that even systems with a complex feedforward control (such as presented in Section 2) can easily be modeled.

The user simply needs to add the extra feedforward control to a netlist and the extra work is done by the computer. The tedious work of generating transfer functions of a detailed model by hand are no longer an issue with the proposed THTF generator.

5.2. FUTURE WORK

The methods presented in this dissertation have proven themselves experimentally for basic systems consisting of only a dc-link capacitor on the dc bus. In their current state, each of the proposed methods is useful in analyzing and designing microinverters. Another application in which these methods would prove useful is in grid-tied energy storage. One of the original motivations behind much of this work was to provide better analysis tools for grid-tied electrochemical energy storage applications. Modifying the example netlist in Section 4 to analyze an energy storage application will take little effort. The dc-link capacitor can simply be replaced with the circuit in Figure 5.1 [52].

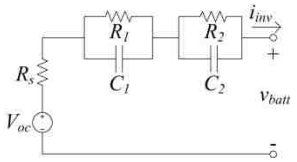

Figure 5.1. Battery circuit equivalent.

Modifying the equations in Section 3 to predict the periodic steady-state operating point is a nontrivial process. This is due to the i_{inv} calculation in (75). For (75) to be valid for the proposed algorithm, all of its variables must be inputs or states. By

replacing the dc-link capacitor with the circuit in Figure 5.1, v_{dc} is no longer a state. Instead, v_{dc} becomes,

$$v_{dc} = -i_{inv}R_s - v_{C1} - v_{C2} + V_{oc}\,. \tag{172}$$

Placing v_{dc} into (42) yields,

$$i_{inv} = \frac{v_{av}i_L}{-i_{inv}R_s - v_{C1} - v_{C2} + V_{oc}}\,. \tag{173}$$

The quadratic equation is then required to solve for i_{inv},

$$i_{inv} = \frac{V_{oc} - v_{C1} - v_{C2} \pm \sqrt{\left(v_{C1} + v_{C2} - V_{oc}\right)^2 - 4R_s i_L}}{2R_s}\,. \tag{174}$$

While all the variables in (174) are states, there is a square root operation involved. The algorithms in Section 3 did not present an algorithm to perform a square root. Ideally, an algorithm will be generated to perform the square root, which preserves the frequency transfer properties. For a quick solution, the jacobian of the square root operation can be found (a best guess will have to be made for the linearization point). This will provide a linearized approximation of the square root. A downside of the linearized approximation is that all of the frequency transfer properties of the square root will be lost.

Once the method presented in Section 3 has been modified to include a square root operation, it will be very useful in predicting harmonics that will appear in an energy storage application. The effect of various harmonics on the performance of batteries is not well studied. This dissertation presents tools which can be used to gain a better idea of how the harmonic content seen at the battery terminals effects battery performance.

Lastly, the algorithms in Section 4 produce a THTF for a given netlist of a system. This proved useful, however can be inaccurate for large systems or for when a

large number of harmonics are analyzed. A truncated harmonic state space (THSS) model would not only be more accurate, it would take much less time to generate and would open the door to even more LTI design and analysis techniques.

APPENDIX A.

MATLAB CODE FOR FOURIER SERIES ANALYSIS OF A SINGLE PHASE INVERTER

```
% HTFMultiply.m
% Luke Watson
% 4/22/13
% This code is used to find the symbolic harmonic result of multiplication

% Due to only computing to Nmax harmonics, there is error on the edge
% frequencies, due to componets of the Nmax frequency going past the
% Nmax frequency during certain operations and being deleted as a result.
% This can be fixed by setting Nmax to be higher than the desired result,
% at the cost of more computational resources.

% VERIFIED WORKING 4/23/13

function [HTFVarResult] = HTFMultiply(Nmax, iVarA, iVarB)


HTFVarResult = sym(zeros(2*Nmax+1, 1));


%-------------------------------------------------------------------------
%----------------------DC PORTION-----------------------------------------
%-------------------------------------------------------------------------
HTFVarResult(1,1) = iVarA(1,1)*iVarB(1,1);

i = 1;
while (i <= Nmax)

    HTFVarResult(1,1) = HTFVarResult(1,1) + ...
        0.5*(iVarA(2*i,1)*iVarB(2*i,1) + ...
        iVarA(2*i+1,1)*iVarB(2*i+1,1));

    i = i + 1;
end

%-------------------------------------------------------------------------
%----------------DC INFLUENCE-----------------------------------------
%-------------------------------------------------------------------------
l = 1;
while (l <= Nmax)
    % d-axis
    HTFVarResult(2*l) = iVarB(1,1)*iVarA(2*l) + ...
        iVarA(1,1)*iVarB(2*l);
    % q-axis
    HTFVarResult(2*l+1) = iVarB(1,1)*iVarA(2*l+1) + ...
        iVarA(1,1)*iVarB(2*l+1);

    l = l + 1;
end


%-------------------------------------------------------------------------
%----------------ADDITIVE PORTION-----------------------------------------
%-------------------------------------------------------------------------

l = 1;
while (l <= Nmax)

    i = 1;
    while (i < l)
        % D-axis additive portion
        HTFVarResult(2*l,1) = HTFVarResult(2*l,1) + ...
            0.5*(iVarA(2*(l-i),1)*iVarB(2*i,1) - ...
            iVarA(2*(l-i)+1,1)*iVarB(2*i+1,1));
```

```
        % Q-axis additive portion
        HTFVarResult(2*l+1,1) = HTFVarResult(2*l+1,1) + ...
            0.5*(iVarA(2*i+1,1)*iVarB(2*(l-i),1) + ...
            iVarA(2*i,1)*iVarB(2*(l-i)+1,1));

        i = i + 1;
    end
    l = l + 1;
end


%-------------------------------------------------------------------------
%--------------------SUBTRACTIVE PORTION----------------------------------
%-------------------------------------------------------------------------

l = 1;
while (l <= Nmax)

    i = l + 1;
    while (i <= Nmax)
        % D-axis subtractive portion
        HTFVarResult(2*l,1) = HTFVarResult(2*l,1) + ...
            0.5*(iVarA(2*i,1)*iVarB(2*(i-l),1) + ...
            iVarA(2*i+1,1)*iVarB(2*(i-l)+1,1) + ...
            iVarA(2*(i-l),1)*iVarB(2*i,1) + ...
            iVarA(2*(i-l)+1,1)*iVarB(2*i+1,1));
        % Q-axis subtractive portion
        HTFVarResult(2*l+1,1) = HTFVarResult(2*l+1,1) + ...
            0.5*(-iVarA(2*i,1)*iVarB(2*(i-l)+1,1) + ...
            iVarA(2*i+1,1)*iVarB(2*(i-l),1) + ...
            iVarA(2*(i-l),1)*iVarB(2*i+1,1) - ...
            iVarA(2*(i-l)+1,1)*iVarB(2*i,1));
        i = i + 1;
    end
    l = l + 1;
end
```

```
% HTFInverse.m
% Luke Watson
% 4/22/13
% This code is used to find the symbolic harmonic result of inversion

% Due to only computing to Nmax harmonics, there is error on the edge
% frequencies, due to componets of the Nmax frequency going past the
% Nmax frequency during certain operations and being deleted as a result.
% This can be fixed by setting Nmax to be higher than the desired result,
% at the cost of more computational resources.

% VERIFIED WORKING 4/23/13...GENERATES VERY LARGE ANSWERS...NEED TO
% CREATE A FILE WHICH CAN APPROXIMATE THE SOLUTIONS...

function [HTFVarResult] = HTFInverse(Nmax, iVar)


% This matrix is an augmented matrix for the equations to do an inverse
% Rref is performed to solve for each value of HTFVarResult
B = sym(zeros(2*Nmax+1, 2*Nmax+2));

% Initialize coefficient matrix
B(1, 2*Nmax+2) = 1;

% Calculate dc componet
B(1, 1) = iVar(1,1);

i = 1;
while(i <= Nmax)

    % Find Bwd componet
    B(1, 2*i) = iVar(2*i,1)/2;

    % Find Bwq componet
    B(1, 2*i+1) = iVar(2*i+1,1)/2;

    i = i + 1;
end

% Calculate ac componet
l = 1;
while (l <= Nmax)


    % dc influence on Bwd componet
    B(2*l, 1) = B(2*l, 1) + iVar(2*l,1);
    B(2*l, 2*l) = B(2*l, 2*l) + iVar(1,1);


    % dc influence on Bwq componet
    B(2*l+1, 1) = B(2*l+1, 1) + iVar(2*l+1,1);
    B(2*l+1, 2*l+1) = B(2*l+1, 2*l+1) + iVar(1,1);


    % Addition influence on Bwd componet
    i = 1;
    while (i < l)
        B(2*l, 2*i) = B(2*l, 2*i) + iVar(2*(l-i),1)/2;
        B(2*l, 2*i+1) = B(2*l, 2*i+1) - iVar(2*(l-i)+1,1)/2;
        i = i + 1;
    end
```

```
    % Addition influence on Bwq componet
    i = 1;
    while (i < l)
        B(2*l+1, 2*i) = B(2*l+1, 2*i) + iVar(2*(l-i)+1,1)/2;
        B(2*l+1, 2*i+1) = B(2*l+1, 2*i+1) + iVar(2*(l-i),1)/2;
        i = i + 1;
    end


    % Subtraction influence on Bwd componet
    i = l + 1;
    while (i <= Nmax)
        B(2*l, 2*(i-l)) = B(2*l, 2*(i-l)) + iVar(2*i,1)/2;
        B(2*l, 2*(i-l)+1) = B(2*l, 2*(i-l)+1) + iVar(2*i+1,1)/2;
        B(2*l, 2*i) = B(2*l, 2*i) + iVar(2*(i-l),1)/2;
        B(2*l, 2*i+1) = B(2*l, 2*i+1) + iVar(2*(i-l)+1,1)/2;
        i = i + 1;
    end


    % Subtraction influence on Bwq componet
    i = l + 1;
    while (i <= Nmax)
        B(2*l+1, 2*(i-l)+1) = B(2*l+1, 2*(i-l)+1) - iVar(2*i,1)/2;
        B(2*l+1, 2*(i-l)) = B(2*l+1, 2*(i-l)) + iVar(2*i+1,1)/2;
        B(2*l+1, 2*i+1) = B(2*l+1, 2*i+1) + iVar(2*(i-l),1)/2;
        B(2*l+1, 2*i) = B(2*l+1, 2*i) - iVar(2*(i-l)+1,1)/2;
        i = i + 1;
    end


    l = l + 1;
end

% this solution simply uses rref to find the solution...slower for larger
% matrices
% tic
% RrefResult = rref(B);
% toc
%
% HTFVarResult = RrefResult(:, 2*Nmax+2);

% This code seems to be the best at finding the solution...this works by
% finding the cofactor matrix of the top row of B (not including the
% augmented last column of B).  Since the transpose of the cofactor
% matrix of the top row results in the first column, this divided by
% the det(B(:, 1:2*Nmax+1)) effectively finds the inverse of B for the
% first column only.

HTFVarResult = sym(zeros(2*Nmax+1, 1));
DetB = det(B(:, 1:2*Nmax+1));

% 1st entry
HTFVarResult(1,1) = det(B(2:2*Nmax+1, 2:2*Nmax+1))/DetB;
% 2nd entry to 2nd to last entry
i = 2;
while (i < 2*Nmax+1)
    HTFVarResult(i, 1) = ((-1)^(i+1))*det([B(2:2*Nmax+1, 1:i-1), B(2:2*Nmax+1,
i+1:2*Nmax+1)])/DetB;
    i = i + 1;
end
% last entry
HTFVarResult(2*Nmax+1,1) = ((-1)^(i+1))*det(B(2:2*Nmax+1, 1:i-1))/DetB;
```

```
% HTFDelaySignal.m
% Luke Watson
% 4/17/13
% This code is used to find the symbolic harmonic result of a single phase
% dq conversion.

% Verified by checking coefficients in simulink model on 4/18/13.

function HTFDelay = HTFDelaySignal(Nmax, iVars)

SignLookup = [-1, 1; -1, -1; 1, -1; 1, 1];

HTFDelay = sym(zeros(2*Nmax+1, 1));

HTFDelay(1,1) = iVars(1,1);

i = 1;
while (i <= Nmax)

    % used to see which set of signs to use
    SignIdx = mod(i,4);
    if (SignIdx == 0)
        SignIdx = 4;
    end

    % check to see if i is even
    if (mod(i,2) == 0)
        HTFDelay(2*i,1) = SignLookup(SignIdx,1)*iVars(2*i,1);
        HTFDelay(2*i+1,1) = SignLookup(SignIdx,2)*iVars(2*i+1,1);
    % i is odd
    else
        HTFDelay(2*i+1,1) = SignLookup(SignIdx,1)*iVars(2*i,1);
        HTFDelay(2*i,1) = SignLookup(SignIdx,2)*iVars(2*i+1,1);
    end

    i = i + 1;
end
```

```
% HTFToSinglePhaseDQ.m
% Luke Watson
% 4/17/13
% This code is used to find the symbolic harmonic result of a single phase
% dq conversion.


% CODE VERIFIED WORKING 4/19/13
% Due to only computing to Nmax harmonics, there is error on the edge
% frequencies, due to componets of the Nmax frequency going past the
% Nmax frequency during certain operations and being deleted as a result.
% This can be fixed by setting Nmax to be higher than the desired result,
% at the cost of more computational resources.


function [HTFVarD, HTFVarQ] = HTFToSinglePhaseDQ(Nmax, iVars, iVarsDelay)


HTFVarD = sym(zeros(2*Nmax+1, 1));
HTFVarQ = sym(zeros(2*Nmax+1, 1));



%-------------------------------------------------------------------------
%---------------ADDITIVE PORTION------------------------------------------
%-------------------------------------------------------------------------

% HTFVarD d-axis componet (additive dc influence)
HTFVarD(2, 1) = iVars(1, 1);
% HTFVarD q-axis componet (additive dc influence)
HTFVarD(3, 1) = -iVarsDelay(1, 1);

% HTFVarQ d-axis componet (additive dc influence)
HTFVarQ(2, 1) = iVarsDelay(1, 1);
% HTFVarQ q-axis componet (additive dc influence)
HTFVarQ(3, 1) = iVars(1, 1);

l = 2;
while (l <= Nmax)
    % HTFVarD d-axis componet
    HTFVarD(2*l, 1) = iVars(2*(l-1), 1)/2 + iVarsDelay(2*(l-1)+1,1)/2;
    % HTFVarD q-axis componet
    HTFVarD(2*l+1, 1) = -iVarsDelay(2*(l-1), 1)/2 + iVars(2*(l-1)+1,1)/2;

    % HTFVarQ d-axis componet
    HTFVarQ(2*l, 1) = iVarsDelay(2*(l-1), 1)/2 - iVars(2*(l-1)+1,1)/2;
    % HTFVarQ q-axis componet
    HTFVarQ(2*l+1, 1) = iVars(2*(l-1), 1)/2 + iVarsDelay(2*(l-1)+1,1)/2;

    l = l + 1;
end

%-------------------------------------------------------------------------
%--------------------SUBTRACTIVE PORTION----------------------------------
%-------------------------------------------------------------------------

% d-axis dc componet (subtractive)
HTFVarD(1,1) = iVars(2,1)/2 - iVarsDelay(3,1)/2;

% q-axis dc componet (subtractive)
HTFVarQ(1,1) = iVarsDelay(2,1)/2 + iVars(3,1)/2;

l = 1;
```

```
while (l < Nmax)
    % HTFVarD d-axis componet
    HTFVarD(2*l, 1) = HTFVarD(2*l, 1) + iVars(2*(l+1),1)/2 - ...
        iVarsDelay(2*(l+1)+1, 1)/2;
    % HTFVarD q-axis componet
    HTFVarD(2*l+1, 1) = HTFVarD(2*l+1, 1) + iVarsDelay(2*(l+1),1)/2 + ...
        iVars(2*(l+1)+1, 1)/2;

    % HTFVarQ d-axis componet
    HTFVarQ(2*l, 1) = HTFVarQ(2*l, 1) + iVarsDelay(2*(l+1),1)/2 + ...
        iVars(2*(l+1)+1, 1)/2;
    % HTFVarQ q-axis componet
    HTFVarQ(2*l+1, 1) = HTFVarQ(2*l+1, 1) - iVars(2*(l+1),1)/2 + ...
        iVarsDelay(2*(l+1)+1, 1)/2;

    l = l + 1;
end
```

```
% HTFFromSinglePhaseDQ.m
% Luke Watson
% 4/17/13
% This code is used to find the symbolic harmonic result of a single phase
% dq conversion.

% CODE VERIFIED WORKING 4/19/13
% Due to only computing to Nmax harmonics, there is error on the edge
% frequencies, due to componets of the Nmax frequency going past the
% Nmax frequency during certain operations and being deleted as a result.
% This can be fixed by setting Nmax to be higher than the desired result,
% at the cost of more computational resources.


function [HTFVarsOut] = HTFFromSinglePhaseDQ(Nmax, iVarsD, iVarsQ)


HTFVarsOutCOS = sym(zeros(2*Nmax+1, 1));
HTFVarsOutSIN = sym(zeros(2*Nmax+1, 1));


%-------------------------------------------------------------------------
%----------------ADDITIVE PORTION-----------------------------------------
%-------------------------------------------------------------------------

% COS term d-axis componet (additive dc influence)
HTFVarsOutCOS(2,1) = iVarsD(1,1);
% COS term q-axis componet (additive dc influence)
HTFVarsOutCOS(3,1) = 0;

% SIN term d-axis componet (additive dc influence)
HTFVarsOutSIN(2,1) = 0;
% SIN term q-axis componet (additive dc influence)!!!
HTFVarsOutSIN(3,1) = -iVarsQ(1,1);

l = 2;
while (l <= Nmax)

    % COS term d-axis componet
    HTFVarsOutCOS(2*l,1) = iVarsD(2*(l-1),1)/2;
    % COS term q-axis componet
    HTFVarsOutCOS(2*l+1,1) = iVarsD(2*(l-1)+1,1)/2;

    % SIN term d-axis componet
    HTFVarsOutSIN(2*l,1) = iVarsQ(2*(l-1)+1,1)/2;
    % SIN term q-axis componet
    HTFVarsOutSIN(2*l+1,1) = -iVarsQ(2*(l-1),1)/2;

    l = l + 1;
end

%-------------------------------------------------------------------------
%--------------------SUBTRACTIVE PORTION----------------------------------
%-------------------------------------------------------------------------

% COS term dc componet (subtractive)
HTFVarsOutCOS(1,1) = iVarsD(2, 1)/2;

% SIN term dc componet (subtractive)!
HTFVarsOutSIN(1,1) = -iVarsQ(3, 1)/2;

l = 1;
while (l < Nmax)
```

```
    % COS term d-axis componet
    HTFVarsOutCOS(2*l,1) = HTFVarsOutCOS(2*l,1) + iVarsD(2*(l+1),1)/2;
    % COS term q-axis componet
    HTFVarsOutCOS(2*l+1,1) = HTFVarsOutCOS(2*l+1,1) +...
        iVarsD(2*(l+1)+1,1)/2;

    % SIN term d-axis componet
    HTFVarsOutSIN(2*l,1) = HTFVarsOutSIN(2*l,1) - iVarsQ(2*(l+1)+1,1)/2;
    % SIN term q-axis componet
    HTFVarsOutSIN(2*l+1,1) = HTFVarsOutSIN(2*l+1,1) + iVarsQ(2*(l+1),1)/2;

    l = l + 1;
end

HTFVarsOut = HTFVarsOutCOS - HTFVarsOutSIN;
```

```
% HTFXCoupling.m
% Luke Watson
% 4/22/13
% This code is used to find the cross coupling terms in the dot equations.
function [HTFResult] = HTFXCoupling(Nmax, w, iVar)

HTFResult = sym(zeros(2*Nmax+1, 1));

i = 1;
while(i <= Nmax)

    HTFResult(2*i,1) = -i*w*iVar(2*i+1,1);
    HTFResult(2*i+1,1) = i*w*iVar(2*i);

    i = i + 1;
end
```

```
% HTFGenerateFunctions.m
% Luke Watson
% 4/17/13
%
% This file must be run first if implementing "Fourier Series Analysis of
% a Single Phase Inverter"
%
% This code is used to generate the dot equations for the single phase
% inverter in "Fourier Series Analysis of a Single Phase Inverter"
% Equations 70-80 in Luke's dissertation are implemented by running this
% file.
%
%
% Symbolic large-signal harmonic (harmonic meaning the inputs and output
% to each equation are fourier series coefficients) equations are generated
% and stored in VDC_DOT.m, IL_DOT.m, X1_DOT.m, X2_DOT.m, and X3_DOT.m.


clear all;

display('Starting mupad calculations');

% Number of harmonics to consider
% Need to make slight modifications to code if less than 4 harmonics
% are to be analyzed.  It takes this code around 4 minutes to run
% on my laptop using 4 harmonics.
Nmax = 6;

% system scalars
syms w L C Kpdc Kidc Kpdq Kidq Rl Rc RL

% Initialize system state variables
VDC = HTFInitVariable(Nmax, 'VDCx');
IL = HTFInitVariable(Nmax, 'ILx');
X1 = HTFInitVariable(Nmax, 'X1x');
X2 = HTFInitVariable(Nmax, 'X2x');
X3 = HTFInitVariable(Nmax, 'X3x');

% Initialize system inputs
VGRID = HTFInitVariable(Nmax, 'VGRIDx');
VDCCOM = HTFInitVariable(Nmax, 'VDCCOMx');
IIN = HTFInitVariable(Nmax, 'IINx');


% Equations 70-74 in Luke Watson's dissertation
IL_DELAY = HTFDelaySignal(Nmax, IL);
[ILD ILQ] = HTFToSinglePhaseDQ(Nmax, IL, IL_DELAY);
VAVD = X1 + Kpdq*(X3 + (Kpdc)*(VDC - VDCCOM) - ILD) - w*ILQ*0.0084;
VAVQ = X2 + Kpdq*(-ILQ) + w*ILD*0.0084;
VAV = HTFFromSinglePhaseDQ(Nmax, VAVD, VAVQ);


% Odd harmonics of vdc are set to zero, since these
% terms will be near zero in the final solution
display('starting subs on vdc');
syms VDCx1d VDCx1q VDCx3d VDCx3q
VDC_SUBS = [VDCx1d, VDCx1q, VDCx3d, VDCx3q];
VDC_SUBS_NUMS = [0, 0, 0, 0];
% Equation 75
VDC_INV = subs(VDC, VDC_SUBS, VDC_SUBS_NUMS);
display('done with subs on vdc');
```

```
display('starting iinv inversion')
tic
% Still Equation 75
IINV = HTFMultiply(Nmax, HTFMultiply(Nmax, VAV, IL),...
    HTFInverseLimit(Nmax, 4, VDC_INV));
toc
display('done with iinv calculation')


display('starting dot equations')
tic;
% Equations 76-80
VDC_DOT = (1/C)*(-IINV+IIN) - HTFXCoupling(Nmax, w, VDC);
IL_DOT = (1/L)*(VAV-VGRID-RL*IL) - HTFXCoupling(Nmax, w, IL);
X1_DOT = Kidq*(X3 + (Kpdc)*(VDC - VDCCOM) - ILD) - ...
    HTFXCoupling(Nmax, w, X1);
X2_DOT = Kidq*(-ILQ) - HTFXCoupling(Nmax, w, X2);
X3_DOT = (Kidc)*(VDC - VDCCOM) - HTFXCoupling(Nmax, w, X3);
toc;
display('done with dot equations')

tic;
display('GENERATING DOT EQUATIONS');
matlabFunction(VDC_DOT, 'file', 'VDC_DOT.m');
matlabFunction(IL_DOT, 'file', 'IL_DOT.m');
matlabFunction(X1_DOT, 'file', 'X1_DOT.m');
matlabFunction(X2_DOT, 'file', 'X2_DOT.m');
matlabFunction(X3_DOT, 'file', 'X3_DOT.m');
display('done generating dot equations');
toc;
```

```
% SinglePhaseInverterFSOLVE.m
% Luke Watson
% 5/14/13
%
% This code generates the predicted steady state fourier coefficients for
% a system, using the dot equations generated from HTFGenerateFunctions.m
%
% The predicted steady state dc-link voltage is stored in VDC_SS_Result.
% The predicted steady state grid current is stored in IL_SS_Result
%

% VERIFIED WORKING 5/15/13


function [VDC_SS_Result, IL_SS_Result, X1_SS_Result, ...
    X2_SS_Result, X3_SS_Result] = ...
    SinglePhaseInverterFSOLVE(VGRID_INPUT, IIN_INPUT)

Nmax = 6;
NumStates = 5;
Result = zeros((2*Nmax+1)*NumStates, 1);
VDCENUM = 1;
ILENUM = 2;
X1ENUM = 3;
X2ENUM = 4;
X3ENUM = 5;


% Determine knowns and unknowns for each case.  Each Unknowns{idx} and
% Knowns{idx} represents the indexing of Unknown and Known states during
% iterating.
Unknowns = cell(6,1);
Unknowns{1} = 1:3;
Unknowns{2} = 2:5;
Unknowns{3} = 4:7;
Unknowns{4} = 6:9;
Unknowns{5} = 8:11;
Unknowns{6} = 10:13;

Knowns = cell(6,1);
Knowns{1} = 4:13;
Knowns{2} = [1,6:13];
Knowns{3} = [1:3,8:13];
Knowns{4} = [1:5,10:13];
Knowns{5} = [1:7,12:13];
Knowns{6} = 1:9;

%-----------------------------------------------
% -------------Define Constants--------------
%-----------------------------------------------
L = 8.4e-3;
RL = 0;
C = 680e-6;
w = 2*pi*60;
% Case 1
Kidq = 10;
Kpdq = 2.509;
Kidc = 10;
Kpdc = 0.3488;

% Case 2
% Kidq = 50;
% Kpdq = 4.407;
```

```
% Kidc = 50;
% Kpdc = 0.7189;

% % Case 3
% Kidq = 100;
% Kpdq = 5.398;
% Kidc = 100;
% Kpdc = 1.001;

ConstCell = cell(1,8);
ConstCell{1,1} = L;
ConstCell{1,2} = C;
ConstCell{1,3} = RL;
ConstCell{1,4} = w;
ConstCell{1,5} = Kidq;
ConstCell{1,6} = Kpdq;
ConstCell{1,7} = Kidc;
ConstCell{1,8} = Kpdc;

%-------------------------------------------
% ---------Define System Inputs--------------
%-------------------------------------------
VGRID = zeros(2*Nmax+1,1);
VDCCOM = zeros(2*Nmax+1,1);
IIN = IIN_INPUT;
VGRID = VGRID_INPUT;
VDCCOM(1,1) = 60;

VGRIDCell = num2cell(VGRID');
VDCCOMCell = num2cell(VDCCOM');
IINCell = num2cell(IIN');

%-------------------------------------------
% ---------Init State Variables--------------
%-------------------------------------------
% a, b, c, d, e prefixes are used to keep
% the output of solution() in the correct order
VDC = HTFInitVariable(Nmax, 'aVDCx');
IL = HTFInitVariable(Nmax, 'bILx');
X1 = HTFInitVariable(Nmax, 'cX1x');
X2 = HTFInitVariable(Nmax, 'dX2x');
X3 = HTFInitVariable(Nmax, 'eX3x');



VDCIter = cell(2*Nmax+1,1);
ILIter = cell(2*Nmax+1,1);
X1Iter = cell(2*Nmax+1,1);
X2Iter = cell(2*Nmax+1,1);
X3Iter = cell(2*Nmax+1,1);


tic;

FullIterationNum = 2;
FullIterationCount = 0;
Case = 1;
GoUpFlag = 1;
exitFlag = 0;
while(~exitFlag)

%-------------------------------------------
% ---------Start Iterations-----------------
```

```
%-------------------------------------------


for i = Knowns{Case}
    VDCIter{i} = Result((2*Nmax+1)*(VDCENUM-1)+i, 1);
    ILIter{i} = Result((2*Nmax+1)*(ILENUM-1)+i, 1);
    X1Iter{i} = Result((2*Nmax+1)*(X1ENUM-1)+i, 1);
    X2Iter{i} = Result((2*Nmax+1)*(X2ENUM-1)+i, 1);
    X3Iter{i} = Result((2*Nmax+1)*(X3ENUM-1)+i, 1);
end
for i = Unknowns{Case}
    VDCIter{i} = VDC(i, 1);
    ILIter{i} = IL(i,1);
    X1Iter{i} = X1(i,1);
    X2Iter{i} = X2(i,1);
    X3Iter{i} = X3(i,1);
end

% Generate dot equations
VDC_DOT_EQN = VDC_DOT(ConstCell{:},VGRIDCell{:},VDCCOMCell{:}, ...
    IINCell{:},VDCIter{:},ILIter{:},X1Iter{:},X2Iter{:},X3Iter{:});
IL_DOT_EQN = IL_DOT(ConstCell{:},VGRIDCell{:},VDCCOMCell{:}, ...
    IINCell{:},VDCIter{:},ILIter{:},X1Iter{:},X2Iter{:},X3Iter{:});
X1_DOT_EQN = X1_DOT(ConstCell{:},VGRIDCell{:},VDCCOMCell{:}, ...
    IINCell{:},VDCIter{:},ILIter{:},X1Iter{:},X2Iter{:},X3Iter{:});
X2_DOT_EQN = X2_DOT(ConstCell{:},VGRIDCell{:},VDCCOMCell{:}, ...
    IINCell{:},VDCIter{:},ILIter{:},X1Iter{:},X2Iter{:},X3Iter{:});
X3_DOT_EQN = X3_DOT(ConstCell{:},VGRIDCell{:},VDCCOMCell{:}, ...
    IINCell{:},VDCIter{:},ILIter{:},X1Iter{:},X2Iter{:},X3Iter{:});

% Reset equations being sent to solver
vdc_dot_cell = cell(1,length(Unknowns{Case}));
il_dot_cell = cell(1,length(Unknowns{Case}));
x1_dot_cell = cell(1,length(Unknowns{Case}));
x2_dot_cell = cell(1,length(Unknowns{Case}));
x3_dot_cell = cell(1,length(Unknowns{Case}));

% Reset unknown variables sent to solver
vdc_cell = cell(1,length(Unknowns{Case}));
il_cell = cell(1,length(Unknowns{Case}));
x1_cell = cell(1,length(Unknowns{Case}));
x2_cell = cell(1,length(Unknowns{Case}));
x3_cell = cell(1,length(Unknowns{Case}));


% Populate equations and variables sent to the solver
j = 1;
for i = Unknowns{Case}
    % populate equations
    vdc_dot_cell{j} = VDC_DOT_EQN(i);
    il_dot_cell{j} = IL_DOT_EQN(i);
    x1_dot_cell{j} = X1_DOT_EQN(i);
    x2_dot_cell{j} = X2_DOT_EQN(i);
    x3_dot_cell{j} = X3_DOT_EQN(i);

    % populate variables
    vdc_cell{j} = VDC(i);
    il_cell{j} = IL(i);
    x1_cell{j} = X1(i);
    x2_cell{j} = X2(i);
    x3_cell{j} = X3(i);

    j = j + 1;
```

```
end

% Solve dot equations
solution = solve(vdc_dot_cell{:}, il_dot_cell{:}, x1_dot_cell{:},...
    x2_dot_cell{:}, x3_dot_cell{:}, vdc_cell{:}, il_cell{:},...
    x1_cell{:}, x2_cell{:}, x3_cell{:});

solution_cell = struct2cell(solution);

Result = HTFConvertSoln(Nmax, NumStates, solution_cell, ...
    Unknowns{Case}, Result);

display('Done with case #:');
disp(Case);


% Incrementing the case numbers
% Use to increment Case until it reaches the maximum case value,
% then reset case to 1.
if ((Case < 6) && GoUpFlag)
   Case = Case + 1;
% Switch to decrementing case numbers
elseif (Case >= 6)
    Case = 1;
    FullIterationCount = FullIterationCount + 1;
    if (FullIterationCount >= FullIterationNum)
       exitFlag = 1;
    end
else
    assert(false, 'Error while updating Case #');
end


end

toc;

VDC_SS_Result = Result(1:13);
IL_SS_Result = Result(14:26);
X1_SS_Result = Result(27:39);
X2_SS_Result = Result(40:52);
X3_SS_Result = Result(53:65);
```

APPENDIX B.

MATLAB CODE FOR AUTOMATIC GENERATION OF TRUNCATED HARMONIC TRANSFER FUNCTIONS

```
% THTFGenerator.m
%  Luke Watson
%  8/16/13
% This file is used to generate the TF's for a single phase dq controlled
% inverter

clear all;

tic;

% Each column represents a node.  The first column is the 1st node, 2nd
% column is the 2nd node...and so on.  Each row corresponds to a harmonic.
% The first row is the dc term, 2nd row is the 1d term, 3rd row is the 1q
% term, fourth row is the 2d term, fifth row is the 2q term and so on...
%
%NodeLinearization = [];
newData1 = load('-mat', 'NodeLinearizationCase1.mat');
% Create new variables in the base workspace from those fields.
vars = fieldnames(newData1);
for i = 1:length(vars)
    assignin('base', vars{i}, newData1.(vars{i}));
end

% For cirucits with unrealizable transfer functions (ie. has current
% sources in series with inductors, voltage sources in parallel with
% capacitors, using the symbolic solver will still generate the TF.
% syms s
s = tf([1 0], [1]);

BaseFreq = 2*pi*60;

%  harmonics to be analyzed
w = [BaseFreq 2*BaseFreq];

numHarmonics = length(w);

% Define types of branches at the circuit level
R = 1;
L = 2;
CAP = 3;
V = 4;
I = 5;

% Define types of branches at the system level
IN = 1;
TF = 2;  % The TF portion of TF will contain the dc tf, as well as the
         % harmonic TF's
ADD = 3;
CNTRL = 4;% Control block should be different than the TF block
          % in that it comes with the negative feedback terminal
          % attached to the TF block, so that it will not return an
          % error.  The pos terminal SHOULD NOT be required to be
          % connected to an input, for special cases, such as current
          % control in a boost converter.
MULT = 5;  % MULT and DIV should automatically do linearization...
DIV = 6;   % They should also automatically do all of the dq stuff...
           % but we'll see how long that takes
CKT = 7;    % CKT - Circuit level block
            % EX:  CKT [1 3 4; 10 23 1] [2 5; 20 30] 1
            %  For the input and output CKT columns:
            %       The first row corresponds to the port number of the
            %       circuit, given in the identifier matrix.  The
            %       second row corresponds to the system node that the
```

```
%       above port is connected to.
%
%       For the input (first entry after CKT), the first row
%       gives the input number that the system is connected
%       to.  The second row gives the system node that
%       connects to the input.
%
%       For the output, (second entry after CKT), the first
%       row gives the branch that the output is connected to.
%       There are twice as many columns as branches.  The
%       first half of columns refer to currents across the
%       specified branch.  The second half of columns refer
%       to voltage across the specified branch.
%       In other words, if something is referenced to the
%       branch number, it is referring to that branch's
%       current.  If something is referenced to numBranches
%       plus the branch number, it is referencing that branch's
%       voltage.  This is just a quirk of how the solver
%       matrices are set up.
%
%  Where the last '1' indicates the number of the circuit
%  This is the number referenced when connecting to the
%  circuit.
CONST = 8;
INV = 9;
DELAY = 10;
GAIN = 11;
INT = 12;
TODQ = 13;
FROMDQ = 14;


% Circuit level netlist
CktNet1 = {I 1 2 1
           CAP 1 2 [500e-6 0]
           I 2 1 2};

CktNet2 = {V 1 3 1
           L 1 2 [8.4e-3 0]
           V 2 3 2};

CktNet = {CktNet1 CktNet2};

% CktId is used for labelling the circuit branches.  The 2nd entry contains
% the branch that each label corresponds to.  This ensures that system
% level nodes connect to the correct input and output port.
%
% When generating the system level equations, the column number of
% CktId# gives the potentially rearranged order of variables, while
% the number in the second row gives the desired branch to connect to.
% Between the column number and second number, the desired port
% can be connected to.
%
% Remember, the number in CktId1 is the original branch number of the
% label
CktId1 = {{'Iin', 1}, {'C', 2}, {'Iinv', 3}};
CktId2 = {{'Vav', 1}, {'L', 2}, {'Vgrid', 3}};

CktId = {CktId1 CktId2};

% The number in the 1st row in the output portion of CKT referrs to the
% number in CktId#
N = {IN 0 5 1
```

```
     CKT [1 2; 5 14] [1 2 3 4 5 6; 0 0 0 0 16 0] 1
     IN 0 1 3
     DELAY 29 28 0
     TODQ [29 28] [3 4] 0
     GAIN 4 6 -3.1667
     GAIN 3 7 3.1667
     GAIN 4 27 -1
     GAIN 27 8 2.509
     GAIN 27 9 10
     INT 9 10 0
     ADD [8 10] 11 [1 1]
     ADD [11 7] 13 [1 1]
     CKT [1 2; 2 1] [1 2 3 4 5 6; 0 29 0 0 0 0] 2
     IN 0 15 2
     ADD [15 16] 17 [-1 1]
     GAIN 17 18 0.3488
     GAIN 17 19 10
     INT 19 20 0
     ADD [18 20] 21 [1 1]
     ADD [21 3] 22 [1 -1]
     GAIN 22 23 2.509
     GAIN 22 24 10
     INT 24 25 0
     ADD [23 25] 26 [1 1]
     ADD [26 6] 12 [1 1]
     FROMDQ [12 13] 2 0
     MULT [2 29] 30 0
     MULT [30 31] 14 0
     INV 16 31 0};

 % Identifier vector, which the user uses to identify each
 % system node...WILL BE 31 ENTRIES FOR THE DQ INVERTER
Ni = {'Vgrid', 'Vav', 'ILD', 'ILQ', 'Iin', 'node6', 'node7', 'node8',...
    'node9', 'node10', 'node11', 'VavD', 'VavQ', 'IInv', 'VdcComm',...
    'Vdc', 'node17', 'node18', 'node19', 'node20', 'IgriddComm',...
    'node22', 'node23', 'node24', 'node25', 'node26', 'node27',...
    'iLDelayed', 'iL', 'Pav', 'VdcInverse'};

% Names of each input.  Must coincide with the order specified
% in the netlist, N
InputNames = {'Iin', 'VdcComm', 'Vgrid'};

[b, c] = size(N);

% Find the number of nodes in the circuit, n
i = 1;
j = 0;
n = 0;
while (i <= b)
    if (N{i, 1} == CKT)
        j = max([N{i, 2}(2, :) N{i, 3}(2, :)]);
    else
        j = max([N{i, 2} N{i, 3}]);
    end
    if (j > n)
        n = j;
    end
    i = i + 1;
end

%  Generate transfer functions for each circuit
i = 1;
while (i <= b)
```

```
    if (N{i, 1} == CKT)
        [TFs{N{i,4}} InitTFs{N{i,4}} CktIds{N{i,4}}] = ...
            InitialTFDQAutoGenerator(CktNet{N{i,4}}, CktId{N{i,4}}, w);
    end
    i = i + 1;
end

%  Replace CKT blocks with TF and ADD blocks.
[N Ni] = StampCKT(N, CktIds, TFs, Ni, w);

[b, c] = size(N);

% Find the new number of nodes in the circuit, n, after the
% CKT blocks were replaced with ADD and TF blocks
i = 1;
j = 0;
n = 0;
while (i <= b)
    if (N{i, 1} == CKT)
        j = max([N{i, 2}(2, :) N{i, 3}(2, :)]);
    else
        j = max([N{i, 2} N{i, 3}]);
    end
    if (j > n)
        n = j;
    end
    i = i + 1;
end

 % Find the number of inputs to the system
 i = 1;
 NumInputs = 0;
 while(i <= b)
     if (N{i, 1} == IN)
         NumInputs = NumInputs + 1;
     end
     i = i + 1;
 end


% Initialize the A, B, and C matrices.
A = tf(zeros(n*(numHarmonics*2+1), n*(numHarmonics*2+1)));
B = zeros(n*(numHarmonics*2+1), NumInputs*(numHarmonics*2+1));
C = zeros(n*(numHarmonics*2+1), 1);

display('populating A and B matrices');
% Populate the A, B, and C matrices.
i = 1;
while (i <= b)

    i
    % The input node 2nd column is don't care entry.  The 3rd column
    % is the system node that the input connects to.  The 4th column
    % is the input number in the B matrix.
   if (N{i, 1} == IN)
       % dc input
       B((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
           (N{i, 4}-1)*(2*numHarmonics+1)+1) = 1;
       m = 1;
      while (m <= numHarmonics)
          % d-axis input
       B((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
           (N{i, 4}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) = 1;
```

```
        % q-axis input
    B((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
        (N{i, 4}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) = 1;

        m = m + 1;
    end


elseif (N{i, 1} == TF)
    % Ignore TF naming warnings for this section, since TF's will
    % be renamed after this section of code executes
    warning('off','Control:ltiobject:InputNameClash');
    warning('off','Control:ltiobject:OutputNameClash');
    % Remember, i specifies the current node being analyed (assuming
    % only dc terms)
    % If only dc terms are considered (w=[]), then the dc case would
    % be: A(N{i, 3}, N{i, 2}) = N{i, 4};
    % dc-case
    A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
        (N{i, 2}-1)*(2*numHarmonics+1)+1) = N{i, 4}{1};
    % d and q-axis terms
    m = 1;
    while (m <= numHarmonics)
        % d-axis input to d-axis output
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            (N{i, 2}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) =...
            N{i, 4}{4*(m-1)+2};
        % d-axis input to q-axis output
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            (N{i, 2}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) = ...
            N{i, 4}{4*(m-1)+3};
        % q-axis input to d-axis output
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            (N{i, 2}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) =...
            N{i, 4}{4*(m-1)+4};
        % q-axis input to q-axis output
         A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            (N{i, 2}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) = ...
            N{i, 4}{4*(m-1)+5};
        m = m + 1;
    end
    % Re-enable TF naming warnings
    warning('on','Control:ltiobject:InputNameClash');
    warning('on','Control:ltiobject:OutputNameClash');


% Remember for add, 1st entry is the input nodes, 2nd entry is the
% output node, and the 3rd entry is 1 for an add and -1 for a
% subtract.
elseif (N{i, 1} == ADD)
    % There is no cross coupling between frequencies or dq axis terms
    % for an ADD operation.
    j = 1;
    while (j <= length(N{i, 4}))
        % dc add
        A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
            (N{i, 2}(j)-1)*(2*numHarmonics+1)+1) = N{i, 4}(j);
        m = 1;
        while (m <= numHarmonics)
            % d-axis add
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                (N{i, 2}(j)-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) =...
                N{i, 4}(j);
```

```
                % q-axis add
                A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
                    (N{i, 2}(j)-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) =...
                    N{i, 4}(j);
                m = m + 1;
            end
            j = j + 1;
        end


    elseif (N{i, 1} == CONST)
        % CONST NOT NECESSARY FOR TF GENERATION
%       C(N{i, 3}, 1) = N{i, 4};


    elseif (N{i, 1} == MULT)
        TempCell1 = Conv2Cell(NodeLinearization(:,N{i,2}(1)));
        TempCell2 = Conv2Cell(NodeLinearization(:,N{i,2}(2)));
        MultResult = mult(TempCell1{:}, TempCell2{:});
        BaseNum1 = (N{i, 2}(1)-1)*(2*numHarmonics+1)+1;
        BaseNum2 = (N{i, 2}(2)-1)*(2*numHarmonics+1)+1;
        % DC term
        A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            MultResult(1, 1:2*numHarmonics+1);
        A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            MultResult(1, 2*numHarmonics+2:end);
        % AC term
        m = 1;
        while (m <= numHarmonics)
            % d-axis
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                BaseNum1:BaseNum1+2*numHarmonics) = ...
                MultResult(2*m, 1:2*numHarmonics+1);
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                BaseNum2:BaseNum2+2*numHarmonics) = ...
                MultResult(2*m, 2*numHarmonics+2:end);
            % q-axis
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
                BaseNum1:BaseNum1+2*numHarmonics) = ...
                MultResult(2*m+1, 1:2*numHarmonics+1);
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
                BaseNum2:BaseNum2+2*numHarmonics) = ...
                MultResult(2*m+1, 2*numHarmonics+2:end);
            m = m + 1;
        end


    elseif (N{i, 1} == INV)
        TempCell1 = Conv2Cell(NodeLinearization(:,N{i,2}));
        DivResult = inverse(TempCell1{:});
        BaseNum1 = (N{i, 2}(1)-1)*(2*numHarmonics+1)+1;
        % DC term
        A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            DivResult(1, 1:2*numHarmonics+1);
        % AC term
        m = 1;
        while (m <= numHarmonics)
            % d-axis
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                BaseNum1:BaseNum1+2*numHarmonics) = ...
```

```
            DivResult(2*m, 1:2*numHarmonics+1);
        % q-axis
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            DivResult(2*m+1, 1:2*numHarmonics+1);
        m = m + 1;
    end


elseif (N{i, 1} == DELAY)
    DelayResult = delay(NodeLinearization(:,N{i,2}));
    BaseNum1 = (N{i, 2}(1)-1)*(2*numHarmonics+1)+1;
    % DC term
    A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
        BaseNum1:BaseNum1+2*numHarmonics) = ...
        DelayResult(1, 1:2*numHarmonics+1);
    % AC term
    m = 1;
    while (m <= numHarmonics)
        % d-axis
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            DelayResult(2*m, 1:2*numHarmonics+1);

        % q-axis
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            DelayResult(2*m+1, 1:2*numHarmonics+1);
        m = m + 1;
    end

% The todq function outputs the D-axis output in the first
% (2*numHarmonics+1) rows and outputs the Q-axis output in the last
% (2*numharmonics+1) rows.
elseif (N{i, 1} == TODQ)
    ToDQResult = todq(NodeLinearization(:,N{i,2}(1)), ...
        NodeLinearization(:,N{i,2}(2)));
    BaseNum1 = (N{i, 2}(1)-1)*(2*numHarmonics+1)+1;
    BaseNum2 = (N{i, 2}(2)-1)*(2*numHarmonics+1)+1;
    % DC term of D-Axis output
    A((N{i, 3}(1)-1)*(2*numHarmonics+1)+1, ...
        BaseNum1:BaseNum1+2*numHarmonics) = ...
        ToDQResult(1, 1:2*numHarmonics+1);
    A((N{i, 3}(1)-1)*(2*numHarmonics+1)+1, ...
        BaseNum2:BaseNum2+2*numHarmonics) = ...
        ToDQResult(1, 2*numHarmonics+2:end);
    % DC term of Q-Axis output
    A((N{i, 3}(2)-1)*(2*numHarmonics+1)+1, ...
        BaseNum1:BaseNum1+2*numHarmonics) = ...
        ToDQResult((2*numHarmonics+2), 1:2*numHarmonics+1);
    A((N{i, 3}(2)-1)*(2*numHarmonics+1)+1, ...
        BaseNum2:BaseNum2+2*numHarmonics) = ...
        ToDQResult((2*numHarmonics+2), 2*numHarmonics+2:end);
    % AC term
    m = 1;
    while (m <= numHarmonics)
        % D-axis term of D-Axis output
        A((N{i, 3}(1)-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            ToDQResult(2*m, 1:2*numHarmonics+1);
        A((N{i, 3}(1)-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            ToDQResult(2*m, 2*numHarmonics+2:end);
```

```
        % Q-axis term of D-Axis output
        A((N{i, 3}(1)-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            ToDQResult(2*m+1, 1:2*numHarmonics+1);
        A((N{i, 3}(1)-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            ToDQResult(2*m+1, 2*numHarmonics+2:end);
        % D-axis term of Q-Axis output
        A((N{i, 3}(2)-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            ToDQResult(2*numHarmonics+1+2*m, 1:2*numHarmonics+1);
        A((N{i, 3}(2)-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            ToDQResult(2*numHarmonics+1+2*m, 2*numHarmonics+2:end);
        % Q-axis term of Q-Axis output
        A((N{i, 3}(2)-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            ToDQResult(2*numHarmonics+1+2*m+1, 1:2*numHarmonics+1);
        A((N{i, 3}(2)-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            ToDQResult(2*numHarmonics+1+2*m+1, 2*numHarmonics+2:end);
        m = m + 1;
    end


elseif (N{i, 1} == FROMDQ)
    FromDQResult = fromdq(NodeLinearization(:,N{i,2}(1)), ...
        NodeLinearization(:,N{i,2}(2)));
    BaseNum1 = (N{i, 2}(1)-1)*(2*numHarmonics+1)+1;
    BaseNum2 = (N{i, 2}(2)-1)*(2*numHarmonics+1)+1;
    % DC term
    A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
        BaseNum1:BaseNum1+2*numHarmonics) = ...
        FromDQResult(1, 1:2*numHarmonics+1);
    A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
        BaseNum2:BaseNum2+2*numHarmonics) = ...
        FromDQResult(1, 2*numHarmonics+2:end);
    % AC term
    m = 1;
    while (m <= numHarmonics)
        % d-axis
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            FromDQResult(2*m, 1:2*numHarmonics+1);
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            FromDQResult(2*m, 2*numHarmonics+2:end);
        % q-axis
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum1:BaseNum1+2*numHarmonics) = ...
            FromDQResult(2*m+1, 1:2*numHarmonics+1);
        A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
            BaseNum2:BaseNum2+2*numHarmonics) = ...
            FromDQResult(2*m+1, 2*numHarmonics+2:end);
        m = m + 1;
    end


elseif (N{i, 1} == GAIN)
    BaseNum1 = (N{i, 2}-1)*(2*numHarmonics+1)+1;
    % DC term
    A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
        BaseNum1) = N{i,4};
```

```
        % AC term
        m = 1;
        while (m <= numHarmonics)
            % d-axis
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                BaseNum1+(2*(m-1))+1) = N{i,4};
            % q-axis
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
                BaseNum1+(2*(m-1))+2) = N{i,4};
            m = m + 1;
        end


    elseif (N{i, 1} == INT)
        A((N{i, 3}-1)*(2*numHarmonics+1)+1, ...
            (N{i, 2}-1)*(2*numHarmonics+1)+1) = 1/s;
        % d and q-axis terms
        m = 1;
        while (m <= numHarmonics)
            % d-axis input to d-axis output
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                (N{i, 2}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) = 1/s;
            % cross coupling to q-axis output
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
                (N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) = -w(m)/s;
            % cross coupling to d-axis output
            A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+1, ...
                (N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) = w(m)/s;
            % q-axis input to q-axis output
             A((N{i, 3}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2, ...
                (N{i, 2}-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) = 1/s;
            m = m + 1;
        end


    else
        % error
    end
    i = i + 1;
end


% Generate the final input to node transfer functions
display('calculating inverse');
tic;
Inverse = (eye(length(A)) - A)^-1;
toc;
display('finished calculating inverse');

TFs = Inverse*B;

% ConstTFs = Inverse*C;


 % !!!Modify to take into account the dq terms
 % Name the inputs of the transfer functions
 i = 1;
 while(i <= NumInputs)
     % dc-input
     TFs.InputName((i-1)*(2*numHarmonics+1)+1) = ...
         cellstr(strcat(InputNames(i), '-dc'));
     m = 1;
     while (m <= numHarmonics)
```

```
        % d-axis input
        TFs.InputName((i-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) = ...
            cellstr(sprintf('%s%s%u', InputNames{i}, '-d', m));
        % q-axis input
        TFs.InputName((i-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) = ...
            cellstr(sprintf('%s%s%u', InputNames{i}, '-q', m));
        m = m + 1;
    end
    i = i + 1;
end


% Name the outputs of the transfer functions
% Also name the inputs/outputs of the constant transfer function.
i = 1;
while(i <= n)
    % dc-input
    TFs.OutputName((i-1)*(2*numHarmonics+1)+1) = ...
        cellstr(strcat(Ni(i), '-dc'));
    m = 1;
    while (m <= numHarmonics)
        % d-axis input
        TFs.OutputName((i-1)*(2*numHarmonics+1)+1+(2*(m-1))+1) = ...
            cellstr(sprintf('%s%s%u', Ni{i}, '-d', m));
        % q-axis input
        TFs.OutputName((i-1)*(2*numHarmonics+1)+1+(2*(m-1))+2) = ...
            cellstr(sprintf('%s%s%u', Ni{i}, '-q', m));
        m = m + 1;
    end


   i = i + 1;
end

toc;


beep ;
pause(0.1);
beep ;
pause(0.1);
beep ;
pause(0.1);
beep ;
pause(0.1);
beep ;
pause(0.1);
beep ;
pause(0.1);
beep ;
pause(0.1);


% Each row corresponds to an output.
% Each column corresponds to an input.

GenerateSimulinkFromTFs(TFs(76:80,:), 'Vdc');
GenerateSimulinkFromTFs(TFs(141:145,:), 'iL');
% GenerateSimulinkFromTFs(TFs(53:65,:), 'FromDQ');

%bode(TFs(142,6)); %shows bode plot of VdcComm-dc to iL-d1 bode plot
%bode(TFs(143,6)); %shows bode plot of VdcComm-dc to iL-q1 bode plot
```

```
% DelayJacobian.m
% Luke Watson
% 8/6/13
% This code is used to find the jacobian of a delay operation
Nmax = 2;

% function HTFVars = HTFInitVariable(Nmax, VarName)
inA = HTFInitVariable(Nmax, 'inA');

display('Finding product');
tic;
Result = HTFDelaySignal(Nmax, inA);
toc;

display('Calculating jacobian');
tic;
Jacobian = jacobian(Result, inA);
toc;

display('Saving fromdq.m');
tic;
matlabFunction(Jacobian, 'file', 'delay.m', 'vars', inA);
toc;
```

```
% ToDQJacobian.m
% Luke Watson
% 8/6/13
% This code is used to generate the Jacobian of a ToDQ operation.

Nmax = 2;

% function HTFVars = HTFInitVariable(Nmax, VarName)
inA = HTFInitVariable(Nmax, 'inA');
inB = HTFInitVariable(Nmax, 'inB');

display('Finding product');
tic;
[ResultD ResultQ] = HTFToSinglePhaseDQ(Nmax, inA, inB);
toc;

display('Calculating jacobian');
tic;
Jacobian = jacobian([ResultD; ResultQ], [inA; inB]);
toc;

display('Saving todq.m');
tic;
matlabFunction(Jacobian, 'file', 'todq.m', 'vars', [inA; inB]);
toc;
```

```
% FromDQJacobian.m
% Luke Watson
% 8/6/13
% This code is used to find the Jacobian of a FromDQ operation

Nmax = 2;

% function HTFVars = HTFInitVariable(Nmax, VarName)
inA = HTFInitVariable(Nmax, 'inA');
inB = HTFInitVariable(Nmax, 'inB');

display('Finding product');
tic;
Result = HTFFromSinglePhaseDQ(Nmax, inA, inB);
toc;

display('Calculating jacobian');
tic;
Jacobian = jacobian(Result, [inA; inB]);
toc;

display('Saving fromdq.m');
tic;
matlabFunction(Jacobian, 'file', 'fromdq.m', 'vars', [inA; inB]);
toc;
```

```
% MultiplyJacobian.m
% Luke Watson
% 8/6/13
% This code is used to manipulate the A matrix of a system for a
% multiplication operation.

Nmax = 2;

% function HTFVars = HTFInitVariable(Nmax, VarName)
inA = HTFInitVariable(Nmax, 'inA');
inB = HTFInitVariable(Nmax, 'inB');

display('Finding product');
tic;
multResult = HTFMultiply(Nmax, inA, inB);
toc;

display('Calculating jacobian');
tic;
multJacobian = jacobian(multResult, [inA; inB]);
toc;

display('Saving mult.m');
tic;
matlabFunction(multJacobian, 'file', 'mult.m', 'vars', [inA; inB]);
toc;
```

```
% InverseJacobian.m
% Luke Watson
% 8/6/13
% This code is used to find the Jacobian of an inverse operation.

Nmax = 2;

% function HTFVars = HTFInitVariable(Nmax, VarName)
in = HTFInitVariable(Nmax, 'in');
inTemp = in;

% Set even harmonics to zero, to speed up calculations.

display('Finding inverse');
tic;
%invResult = HTFInverseLimit(Nmax, 4, in);
invResult = HTFInverse(Nmax, in);
toc;

display('Calculating jacobian');
tic;
invJacobian = jacobian(invResult, inTemp);
toc;

display('Saving inverse.m');
tic;
matlabFunction(invJacobian, 'file', 'inverse.m', 'vars', inTemp);
toc;
```

```
% StampCKT.m
%  Luke Watson
%  6/20/12
%  StampCKT replaces a circuit block with a 'stamp'.  This 'stamp'
%  consists of each circuit's transfer functions and add blocks,
%  to sum up transfer function outputs, for the multiple input case.
%
%
function [oNet oSysIds] = StampCKT(iNet, iCktIds, iTFs, iSysIds, w)


% Define types of branches at the system level
IN = 1;
TF = 2;
ADD = 3;
CNTRL = 4;% Control block should be different than the TF block
          % in that it comes with the negative feedback terminal
          % attached to the TF block, so that it will not return an
          % error.  The pos terminal SHOULD NOT be required to be
          % connected to an input, for special cases, such as current
          % control in a boost converter.
MULT = 5;  % MULT and DIV should automatically do linearization...
DIV = 6;   % They should also automatically do all of the dq stuff...
           % but we'll see how long that takes
CKT = 7;    % CKT - Circuit level block
            % EX:  CKT [1 3 4; 10 23 1] [2 5; 20 30] 1
            %  For the input and output CKT columns:
            %       The first row corresponds to the port number of the
            %       circuit, given in the identifier matrix.  The
            %       second row corresponds to the node that the above
            %       port is connected to.
            %  Where the last '1' indicates the number of the circuit
CONST = 8;
INV = 9;

% Find the number of branches in the circuit, b
[b, l] = size(iNet);

numHarmonics = length(w);

oNet = iNet;
oSysIds = iSysIds;

% Find the number of nodes in the circuit, n
n = 0;
i = 1;
while (i <= b)
    j = max(max(oNet{i, 2}));
    if (j > n)
        n = j;
    end
    % If the current Net is a CKT, then only consider output nodes
    % being used.
    if (oNet{i, 1} == CKT)
        j = max(oNet{i,3}(2,:));
    else
        j = max(max(oNet{i, 3}));
    end
    if (j > n)
        n = j;
    end
    i = i + 1;
end
```

```
i = 1;
NewBranches = 0;
NewNodes = 0;
DummyInputIdx = 0;

%  Check each branch in the netlist for a CKT branch
while (i <= b)

    %  If the current branch is a Circuit branch
    if (oNet{i, 1} == CKT)

        % Go through TF outputs
        j = 1;
        while (j <= length(oNet{i, 3}))

            % Check to see if output is actually connected
            if (oNet{i, 3}(2, j) ~= 0)

                % Go through TF inputs
                k = 1;
                InputCount = 0;
                [DC, numInputs] = size(oNet{i, 2});
                while (k <= numInputs)

                    %  Ensure input is actually connected.  If it
                    %  is not, ignore for now.  Possibly throw error
                    %  or warning later.  Currently, if input is not
                    %  connected, it will treat the input as a 0 input.
                    if (oNet{i, 2}(2, k) ~= 0)
                        InputCount = InputCount + 1;
                        NewBranches = NewBranches + 1;
                        NewNodes = NewNodes + 1;
                        oNet{b+NewBranches, 1} = TF;
                        oNet{b+NewBranches, 2} = oNet{i, 2}(2, k);
                        oNet{b+NewBranches, 3} = n + NewNodes;
                        %  The input number specified should match up with
                        %  the input number specified in the netlist.
                        %
                        %  SearchIDTFs() must be used to determine which
                        %  output index for iTFs[] should be used, since
                        %  the output number is specified by the index
                        %  number of the matching port, if the first port
                        %  is a match (current output), otherwise it is b
                        %  plus the matching index (voltage output).!!!
                        inIdx = oNet{i, 2}(1, k);
                        outIdx = SearchIDTFs(iCktIds{oNet{i, 4}}, ...
                            oNet{i, 3}(1, j));

                        % DC Transfer function
                        oNet{b+NewBranches, 4}(1) = ...
                            {iTFs{oNet{i, 4}}{1}( outIdx, inIdx )};
                        % D-axis and q-axis tranfer functions.
                        m = 1;
                        while (m <= numHarmonics)
                           % in-d to out-d
                           oNet{b+NewBranches, 4}((m-1)*4+2) = ...
                               {iTFs{oNet{i, 4}}{m+1}((outIdx-1)*2+1,...
                               (inIdx-1)*2+1)};
                           % in-d to out-q
                           oNet{b+NewBranches, 4}((m-1)*4+3) = ...
                               {iTFs{oNet{i, 4}}{m+1}((outIdx-1)*2+2,...
```

```
                        (inIdx-1)*2+1)};
                    % in-q to out-d
                    oNet{b+NewBranches, 4}((m-1)*4+4) = ...
                        {iTFs{oNet{i, 4}}{m+1}((outIdx-1)*2+1,...
                        (inIdx-1)*2+2)};
                    % in-d to out-q
                    oNet{b+NewBranches, 4}((m-1)*4+5) = ...
                        {iTFs{oNet{i, 4}}{m+1}((outIdx-1)*2+2,...
                        (inIdx-1)*2+2)};

                    m = m + 1;
                end


                %  Also add a dummy entry to the system node
                %  identifier matrix.  This allows for all
                %  of the outputs of the system level TFs
                %  to be named.
                oSysIds(length(oSysIds)+1) =...
                    cellstr(sprintf('Internal-%d',DummyInputIdx));
                DummyInputIdx = DummyInputIdx + 1;

                %  Used to store the input node, for the
                %  single input case.
                InputNode = oNet{i, 2}(2, k);
            end

            k = k + 1;
        end

        %  Generate add block to sum up the transfer functions
        if (InputCount > 1)
            NewBranches = NewBranches + 1;
            oNet{b+NewBranches, 1} = ADD;
            oNet{b+NewBranches, 2} = (n+NewNodes):(-1): ...
                (n+NewNodes-InputCount+1);
            oNet{b+NewBranches, 3} = oNet{i, 3}(2, j);
            oNet{b+NewBranches, 4} = ones(1, InputCount);
        else
            %  If there is only one input, there is no need for
            %  an addition block, connect the output of the
            %  TF directly to the output.
            oNet{b+NewBranches, 3} = oNet{i, 3}(2, j);
            oNet{b+NewBranches, 2} = InputNode;
            NewNodes = NewNodes - 1;
            DummyInputIdx = DummyInputIdx - 1;
            oSysIds(length(oSysIds)) = [];
        end
    end
    j = j + 1;
end

% Need to move all entries below the CKT up one, to erase the
% circuit branch
j = i+1;
while (j <= b+NewBranches)
    oNet(j-1, :) = oNet(j, :);
    j = j + 1;
end
```

```
        % Take into account erased branch in NewBranches
        NewBranches = NewBranches-1;
        oNet(j-1, :) = [];
    end

    i = i + 1;
end
```

```
% SearchIDTFs.m
%  Luke Watson
%  6/20/12
%  SearchIDTFs searches a circuit's transfer function identifier matrix for
%  a matching port number (the port number is what is input by the user,
%  to determine where a node connects to a circuit).  This is used
%  for finding where to connect
%
%
function oIdx = SearchIDTFs(iId, iPortNum)

CktBranches = length(iId);

oIdx = 0;

i = 1;
while (i <= CktBranches)

    % Check to see if port number corresponds to an output current
    if (iId{i}{2} == iPortNum)

        oIdx = i;

    % Check to see if port number corresponds to an output voltage
    elseif (iId{i}{2} + CktBranches == iPortNum)
        oIdx = i + CktBranches;
    end

    i = i + 1;
end
```

```
% Conv2Cell.m
% Converts a matrix to a cell matrix
function oCell = Conv2Cell(iMat)

[N L] = size(iMat);

oCell = cell(N, 1);

i = 1;
while(i<=N)

    oCell{i,1} = iMat(i,1);
    i = i + 1;
end
```

BIBLIOGRAPHY

[1] C. Trujillo Rodriguez, D. Velasco de la Fuente, G. Garcera, E. Figueres, and J. A. Guacaneme Moreno, "Reconfigurable Control Scheme for a PV Microinverter Working in Both Grid-Connected and Island Modes," *Industrial Electronics, IEEE Transactions on*, vol. 60, pp. 1582-1595, 2013.

[2] H. Haibing, S. Harb, N. H. Kutkut, Z. J. Shen, and I. Batarseh, "A Single-Stage Microinverter Without Using Eletrolytic Capacitors," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 2677-2687, 2013.

[3] D. Hamza, Q. Mei, and P. K. Jain, "Application and Stability Analysis of a Novel Digital Active EMI Filter Used in a Grid-Tied PV Microinverter Module," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 2867-2874, 2013.

[4] T. Shimizu, K. Wada, and N. Nakamura, "Flyback-Type Single-Phase Utility Interactive Inverter With Power Pulsation Decoupling on the DC Input for an AC Photovoltaic Module System," *Power Electronics, IEEE Transactions on*, vol. 21, pp. 1264-1272, 2006.

[5] S. B. Kjaer, J. K. Pedersen, and F. Blaabjerg, "Power inverter topologies for photovoltaic modules-a review," in *Proc. Industry Applications Conference, 2002. 37th IAS Annual Meeting. Conference Record of the*, 2002, vol. 2, pp. 782-788 vol.2, 10.1109/ias.2002.1042648.

[6] T. Shimizu, M. Hirakata, T. Kamezawa, and H. Watanabe, "Generation control circuit for photovoltaic modules," *Power Electronics, IEEE Transactions on*, vol. 16, pp. 293-300, 2001.

[7] S. Yatsuki, K. Wada, T. Shimizu, H. Takagi, and M. Ito, "A novel AC photovoltaic module system based on the impedance-admittance conversion theory," in *Proc. Power Electronics Specialists Conference, 2001. PESC. 2001 IEEE 32nd Annual*, 2001, vol. 4, pp. 2191-2196 vol. 4, 10.1109/pesc.2001.954445.

[8] S. B. Kjaer, J. K. Pedersen, and F. Blaabjerg, "A review of single-phase grid-connected inverters for photovoltaic modules," *Industry Applications, IEEE Transactions on*, vol. 41, pp. 1292-1306, 2005.

[9] C. Yaow-Ming, C. Chia-Hsi, and W. Hsu-Chin, "DC-link capacitor selections for the single-phase grid-connected PV system," in *Proc. Power Electronics and Drive Systems, 2009. PEDS 2009. International Conference on*, 2009, pp. 72-77, 10.1109/peds.2009.5385801.

[10] B. A. Yount, F. F. Xiao, Z. Jie, and J. W. Kimball, "Quantifying Insolation in Multiple Shading Scenarios," in *Proc. Green Technologies Conference (IEEE-Green), 2011 IEEE*, 2011, pp. 1-6, 10.1109/green.2011.5754882.

[11] S. M. MacAlpine, R. W. Erickson, and M. J. Brandemuehl, "Characterization of Power Optimizer Potential to Increase Energy Capture in Photovoltaic Systems Operating Under Nonuniform Conditions," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 2936-2945, 2013.

[12] C. R. Sullivan, J. J. Awerbuch, and A. M. Latham, "Decrease in Photovoltaic Power Output from Ripple: Simple General Calculation and the Effect of Partial Shading," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 740-747, 2013.

[13] I. Abdalla, J. Corda, and L. Zhang, "Multilevel DC-Link Inverter and Control Algorithm to Overcome the PV Partial Shading," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 14-18, 2013.

[14] J. Young-Hyok, J. Doo-Yong, K. Jun-Gu, K. Jae-Hyung, L. Tae-Won, and W. Chung-Yuen, "A Real Maximum Power Point Tracking Method for Mismatching Compensation in PV Array Under Partially Shaded Conditions," *Power Electronics, IEEE Transactions on*, vol. 26, pp. 1001-1009, 2011.

[15] P. S. Shenoy, K. A. Kim, B. B. Johnson, and P. T. Krein, "Differential Power Processing for Increased Energy Production and Reliability of Photovoltaic Systems," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 2968-2979, 2013.

[16] H. Oldenkamp, I. J. De Jong, C. W. A. Baltus, S. A. M. Verhoeven, and S. Elstgeest, "Reliability and accelerated life tests of the AC module mounted OKE4 inverter," in *Proc. Photovoltaic Specialists Conference, 1996., Conference Record of the Twenty Fifth IEEE*, 1996, pp. 1339-1342, 10.1109/pvsc.1996.564381.

[17] L. E. De Graaf and T. C. J. Van der Weiden, "Characteristics and performance of a PV-system consisting of 20 AC-modules," in *Proc. Photovoltaic Energy Conversion, 1994., Conference Record of the Twenty Fourth. IEEE Photovoltaic Specialists Conference - 1994, 1994 IEEE First World Conference on*, 1994, vol. 1, pp. 921-924 vol.1, 10.1109/wcpec.1994.520112.

[18] C. Huang-Jen, L. Yu-Kang, Y. Chun-Yu, *et al.*, "A Module-Integrated Isolated Solar Microinverter," *Industrial Electronics, IEEE Transactions on*, vol. 60, pp. 781-788, 2013.

[19] S. B. Kjaer and F. Blaabjerg, "Design optimization of a single phase inverter for photovoltaic applications," in *Proc. Power Electronics Specialist Conference, 2003. PESC '03. 2003 IEEE 34th Annual*, 2003, vol. 3, pp. 1183-1190 vol.3, 10.1109/pesc.2003.1216616.

[20] T. Shimizu, K. Wada, and N. Nakamura, "A flyback-type single phase utility interactive inverter with low-frequency ripple current reduction on the DC input for an AC photovoltaic module system," in *Proc. Power Electronics Specialists Conference, 2002. pesc 02. 2002 IEEE 33rd Annual*, 2002, vol. 3, pp. 1483-1488 vol.3, 10.1109/psec.2002.1022385.

[21] K. Yeong-Chan, L. Tsorng-Juu, and C. Jiann-Fuh, "Novel maximum-power-point-tracking controller for photovoltaic energy conversion system," *Industrial Electronics, IEEE Transactions on*, vol. 48, pp. 594-601, 2001.

[22] R. O. Caceres and I. Barbi, "A boost DC-AC converter: analysis, design, and experimentation," *Power Electronics, IEEE Transactions on*, vol. 14, pp. 134-141, 1999.

[23] C. Albea, C. Canudas-de-Wit, and F. Gordillo, "Adaptive Control of the Boost DC-AC Converter," in *Proc. Control Applications, 2007. CCA 2007. IEEE International Conference on*, 2007, pp. 611-616, 10.1109/cca.2007.4389299.

[24] P. Sanchis, A. Ursaea, E. Gubia, and L. Marroyo, "Boost DC-AC inverter: a new control strategy," *Power Electronics, IEEE Transactions on*, vol. 20, pp. 343-353, 2005.

[25] R. O. Caceres, W. M. Garcia, and O. E. Camacho, "A buck-boost DC-AC converter: operation, analysis, and control," in *Proc. Power Electronics Congress, 1998. CIEP 98. VI IEEE International*, 1998, pp. 126-131, 10.1109/ciep.1998.750672.

[26] C. Albea and F. Gordillo, "Control of the boost DC-AC converter with RL load by energy shaping," in *Proc. Decision and Control, 2007 46th IEEE Conference on*, 2007, pp. 2417-2422, 10.1109/cdc.2007.4434621.

[27] S. Menaka and S. Muralidharan, "Design and performance analysis of novel boost DC-AC converter," in *Proc. Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, 2011, vol. 2, pp. 168-172, 10.1109/icectech.2011.5941678.

[28] M. Coppola, S. Daliento, P. Guerriero, D. Lauria, and E. Napoli, "On the design and the control of a coupled-inductors boost dc-ac converter for an individual PV panel," in *Proc. Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), 2012 International Symposium on*, 2012, pp. 1154-1159, 10.1109/speedam.2012.6264548.

[29] E. Achille, T. Martire, C. Glaize, and C. Joubert, "Optimized DC-AC boost converters for modular photovoltaic grid-connected generators," in *Proc. Industrial Electronics, 2004 IEEE International Symposium on*, 2004, vol. 2, pp. 1005-1010 vol. 2, 10.1109/isie.2004.1571951.

[30] R. Caceres, R. Rojas, and O. Camacho, "Robust PID control of a buck-boost DC-AC converter," in *Proc. Telecommunications Energy Conference, 2000. INTELEC. Twenty-second International*, 2000, pp. 180-185, 10.1109/intlec.2000.884248.

[31] S. Yuvarajan and X. Shanguang, "Single-stage resonant boost AC-DC-AC converter," in *Proc. Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE*, 2002, vol. 1, pp. 537-541 vol.1, 10.1109/apec.2002.989296.

[32] J. H. Ramirez and P. B. Sanchez, "A Soft Switching Boost DC-AC Converter: Analysis, Design and Experimentation," in *Proc. Telecommunications Conference, 2005. INTELEC '05. Twenty-Seventh International*, 2005, pp. 383-383, 10.1109/intlec.2005.335125.

[33] O. Hashimoto, T. Shimizu, and G. Kimura, "A novel high performance utility interactive photovoltaic inverter system," in *Proc. Industry Applications Conference, 2000. Conference Record of the 2000 IEEE*, 2000, vol. 4, pp. 2255-2260 vol.4, 10.1109/ias.2000.883139.

[34] T. Shimizu, O. Hashimoto, and G. Kimura, "A novel high-performance utility-interactive photovoltaic inverter system," *Power Electronics, IEEE Transactions on*, vol. 18, pp. 704-711, 2003.

[35] R. Teodorescu, F. Blaabjerg, U. Borup, and M. Liserre, "A new control structure for grid-connected LCL PV inverters with zero steady-state error and selective harmonic compensation," in *Proc. Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE*, 2004, vol. 1, pp. 580-586 Vol.1, 10.1109/apec.2004.1295865.

[36] L. Asiminoaei, R. Teodorescu, F. Blaabjerg, and U. Borup, "A new method of on-line grid impedance estimation for PV inverter," in *Proc. Applied Power Electronics Conference and Exposition, 2004. APEC '04. Nineteenth Annual IEEE*, 2004, vol. 3, pp. 1527-1533 Vol.3, 10.1109/apec.2004.1296067.

[37] A. V. Timbus, R. Teodorescu, F. Blaabjerg, and U. Borup, "Online grid measurement and ENS detection for PV inverter running on highly inductive grid," *Power Electronics Letters, IEEE*, vol. 2, pp. 77-82, 2004.

[38] F. Renken, "The DC-link capacitor current in pulsed single-phase H-bridge inverters," in *Proc. Power Electronics and Applications, 2005 European Conference on*, 2005, pp. 10 pp.-P.10, 10.1109/epe.2005.219197.

[39] K. Jung-Min, N. Kwang-Hee, and K. Bong-Hwan, "Photovoltaic Power Conditioning System With Line Connection," *Industrial Electronics, IEEE Transactions on*, vol. 53, pp. 1048-1054, 2006.

[40] M. Armstrong, D. J. Atkinson, C. M. Johnson, and T. D. Abeyasekera, "Auto-Calibrating DC Link Current Sensing Technique for Transformerless, Grid Connected, H-Bridge Inverter Systems," *Power Electronics, IEEE Transactions on*, vol. 21, pp. 1385-1393, 2006.

[41] L. Asiminoaei, R. Teodorescu, F. Blaabjerg, and U. Borup, "Implementation and Test of an Online Embedded Grid Impedance Estimation Technique for PV Inverters," *Industrial Electronics, IEEE Transactions on*, vol. 52, pp. 1136-1144, 2005.

[42] Y. Bo, L. Wuhua, Z. Yi, and H. Xiangning, "Design and Analysis of a Grid-Connected Photovoltaic Power System," *Power Electronics, IEEE Transactions on*, vol. 25, pp. 992-1000, 2010.

[43] G. Feng, L. Ding, L. Poh Chiang, T. Yi, and W. Peng, "Indirect dc-link voltage control of two-stage single-phase PV inverter," in *Proc. Energy Conversion Congress and Exposition, 2009. ECCE 2009. IEEE*, 2009, pp. 1166-1172, 10.1109/ecce.2009.5316399.

[44] M. J. Ryan and R. D. Lorenz, "A synchronous-frame controller for a single-phase sine wave inverter," in *Proc. Applied Power Electronics Conference and Exposition, 1997. APEC '97 Conference Proceedings 1997., Twelfth Annual*, 1997, vol. 2, pp. 813-819 vol.2, 10.1109/apec.1997.575739.

[45] A. Roshan, R. Burgos, A. C. Baisden, F. Wang, and D. Boroyevich, "A D-Q Frame Controller for a Full-Bridge Single Phase Inverter Used in Small Distributed Power Generation Systems," in *Proc. Applied Power Electronics Conference, APEC 2007 - Twenty Second Annual IEEE*, 2007, pp. 641-647, 10.1109/apex.2007.357582.

[46] J. Salaet, S. Alepuz, A. Gilabert, and J. Bordonau, "Comparison between two methods of DQ transformation for single phase converters control. Application to a 3-level boost rectifier," in *Proc. Power Electronics Specialists Conference, 2004. PESC 04. 2004 IEEE 35th Annual*, 2004, vol. 1, pp. 214-220 Vol.1, 10.1109/pesc.2004.1355744.

[47] J. Salaet, S. Alepuz, A. Gilabert, J. Bordonau, and J. Peracaula, "D-Q modeling and control of a single-phase three-level boost rectifier with power factor correction and neutral-point voltage balancing," in *Proc. Power Electronics Specialists Conference, 2002. pesc 02. 2002 IEEE 33rd Annual*, 2002, vol. 2, pp. 514-519 vol.2, 10.1109/psec.2002.1022505.

[48] K. Rae-Young, C. See-Young, and S. In-Young, "Instantaneous control of average power for grid tie inverter using single phase D-Q rotating frame with all pass filter," in *Proc. Industrial Electronics Society, 2004. IECON 2004. 30th Annual Conference of IEEE*, 2004, vol. 1, pp. 274-279 Vol. 1, 10.1109/iecon.2004.1433322.

[49] M. T. Haque and T. Ise, "Implementation of single-phase pq theory," in *Proc. Power Conversion Conference, 2002. PCC-Osaka 2002. Proceedings of the*, 2002, vol. 2, pp. 761-765 vol.2, 10.1109/pcc.2002.997615.

[50] R. Zhang, M. Cardinal, P. Szczesny, and M. Dame, "A grid simulator with control of single-phase power converters in D-Q rotating frame," in *Proc. Power Electronics Specialists Conference, 2002. pesc 02. 2002 IEEE 33rd Annual*, 2002, vol. 3, pp. 1431-1436 vol.3, 10.1109/psec.2002.1022377.

[51] L. D. Watson and J. W. Kimball, "Frequency regulation of a microgrid using solar power," in *Proc. Applied Power Electronics Conference and Exposition (APEC), 2011 Twenty-Sixth Annual IEEE*, 2011, pp. 321-326, 10.1109/apec.2011.5744615.

[52] L. D. Watson, J. W. Kimball, and S. Atcitty, "Linear single phase inverter model for Battery Energy Storage System evaluation and controller design," in *Proc. Applied Power Electronics Conference and Exposition (APEC), 2012 Twenty-Seventh Annual IEEE*, 2012, pp. 1861-1867, 10.1109/apec.2012.6166075.

[53] P. C. Krause, "Method of Multiple Reference Frames Applied to the Analysis of Symmetrical Induction Machinery," *Power Apparatus and Systems, IEEE Transactions on*, vol. PAS-87, pp. 218-227, 1968.

[54] S. D. Sudhoff, "Multiple reference frame analysis of an unsymmetrical induction machine," *Energy Conversion, IEEE Transactions on*, vol. 8, pp. 425-432, 1993.

[55] S. D. Sudhoff, "Multiple reference frame analysis of a multistack: variable-reluctance stepper motor," *Energy Conversion, IEEE Transactions on*, vol. 8, pp. 418-424, 1993.

[56] P. L. Chapman, S. D. Sudhoff, and C. A. Whitcomb, "Multiple reference frame analysis of non-sinusoidal brushless DC drives," *Energy Conversion, IEEE Transactions on*, vol. 14, pp. 440-446, 1999.

[57] X. Peng, K. A. Corzine, and G. K. Venayagamoorthy, "Multiple Reference Frame-Based Control of Three-Phase PWM Boost Rectifiers under Unbalanced and Distorted Input Conditions," *Power Electronics, IEEE Transactions on*, vol. 23, pp. 2006-2017, 2008.

[58] P. L. Chapman and S. D. Sudhoff, "A multiple reference frame synchronous estimator/regulator," *Energy Conversion, IEEE Transactions on*, vol. 15, pp. 197-202, 2000.

[59] L. Sang-Joon and S. Seung-Ki, "A harmonic reference frame based current controller for active filter," in *Proc. Applied Power Electronics Conference and Exposition, 2000. APEC 2000. Fifteenth Annual IEEE*, 2000, vol. 2, pp. 1073-1078 vol.2, 10.1109/apec.2000.822821.

[60] L. Padmavathi and P. A. Janakiraman, "Self-Tuned Feed-Forward Compensation for Harmonic Reduction in Single-Phase Low-Voltage Inverters," *Industrial Electronics, IEEE Transactions on*, vol. 58, pp. 4753-4762, 2011.

[61] O. J. Nastov and J. K. White, "Time-mapped harmonic balance," in *Proc. Design Automation Conference, 1999. Proceedings. 36th*, 1999, pp. 641-646, 10.1109/dac.1999.782021.

[62] N. M. Wereley and S. R. Hall, "Frequency response of linear time periodic systems," in *Proc. Decision and Control, 1990., Proceedings of the 29th IEEE Conference on*, 1990, pp. 3650-3655 vol.6, 10.1109/cdc.1990.203516.

[63] N. M. Wereley, "Analysis and Control of Linear Periodically Time Varying Systems," Massachusetts Institute of Technology, 1991.

[64] H. Sandberg, E. Mollerstedt, and Bernhardsson, "Frequency-domain analysis of linear time-periodic systems," *Automatic Control, IEEE Transactions on*, vol. 50, pp. 1971-1983, 2005.

[65] T. Brekken, N. Bhiwapurkar, M. Rathi, N. Mohan, C. Henze, and L. R. Moumneh, "Utility-connected power converter for maximizing power transfer from a photovoltaic source while drawing ripple-free current," in *Proc. Power Electronics Specialists Conference, 2002. pesc 02. 2002 IEEE 33rd Annual*, 2002, vol. 3, pp. 1518-1522 vol.3, 10.1109/psec.2002.1022391.

[66] A. Ale Ahmad, A. Abrishamifar, and S. Samadi, "Low-frequency current ripple reduction in front-end boost converter with single-phase inverter load," *Power Electronics, IET*, vol. 5, pp. 1676-1683, 2012.

[67] E. Mamarelis, C. A. Ramos-Paja, G. Petrone, G. Spagnuolo, M. Vitelli, and R. Giral, "FPGA-based controller for mitigation of the 100 Hz oscillation in grid connected PV systems," in *Proc. Industrial Technology (ICIT), 2010 IEEE International Conference on*, 2010, pp. 925-930, 10.1109/icit.2010.5472556.

[68] M. Monfared, M. Sanatkar, and S. Golestan, "Direct active and reactive power control of single-phase grid-tie converters," *Power Electronics, IET*, vol. 5, pp. 1544-1550, 2012.

[69] L. Piegari and R. Rizzo, "Adaptive perturb and observe algorithm for photovoltaic maximum power point tracking," *Renewable Power Generation, IET*, vol. 4, pp. 317-328, 2010.

[70] T. Esram and P. L. Chapman, "Comparison of Photovoltaic Array Maximum Power Point Tracking Techniques," *Energy Conversion, IEEE Transactions on*, vol. 22, pp. 439-449, 2007.

[71] U. A. Miranda, M. Aredes, and L. G. B. Rolim, "A DQ Synchronous Reference Frame Current Control for Single-Phase Converters," in *Proc. Power Electronics Specialists Conference, 2005. PESC '05. IEEE 36th*, 2005, pp. 1377-1381.

[72] B. Crowhurst, E. F. El-Saadany, L. El Chaar, and L. A. Lamont, "Single-phase grid-tie inverter control using DQ transform for active and reactive load power compensation," in *Proc. Power and Energy (PECon), 2010 IEEE International Conference on*, 2010, pp. 489-494, 10.1109/pecon.2010.5697632.

[73] S. R. Sanders, J. M. Noworolski, X. Z. Liu, and G. C. Verghese, "Generalized averaging method for power conversion circuits," *Power Electronics, IEEE Transactions on*, vol. 6, pp. 251-259, 1991.

[74] V. A. Caliskan, G. C. Verghese, and A. M. Stankovic, "Multifrequency averaging of DC/DC converters," *Power Electronics, IEEE Transactions on*, vol. 14, pp. 124-133, 1999.

[75] Q. Hengsi and J. W. Kimball, "Generalized Average Modeling of Dual Active Bridge DC–DC Converter," *Power Electronics, IEEE Transactions on*, vol. 27, pp. 2078-2084, 2012.

[76] C. Zhiyu, H. Manli, N. Frohleke, and J. Bocker, "Modeling and Control Design for a Very Low-Frequency High-Voltage Test System," *Power Electronics, IEEE Transactions on*, vol. 25, pp. 1068-1077, 2010.

[77] H. Mouton and B. Putzeys, "Understanding the PWM Nonlinearity: Single-Sided Modulation," *Power Electronics, IEEE Transactions on*, vol. 27, pp. 2116-2128, 2012.

[78] D. J. Kostic, Z. Z. Avramovic, and N. T. Ciric, "A New Approach to Theoretical Analysis of Harmonic Content of PWM Waveforms of Single- and Multiple-Frequency Modulators," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 4557-4567, 2013.

[79] W. Bingsen and E. Sherif, "Spectral Analysis of Matrix Converters Based on 3-D Fourier Integral," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 19-25, 2013.

[80] F. Koeslag, H. D. Mouton, and J. Beukes, "Analytical Modeling of the Effect of Nonlinear Switching Transition Curves on Harmonic Distortion in Class D Audio Amplifiers," *Power Electronics, IEEE Transactions on*, vol. 28, pp. 380-389, 2013.

[81] A. Z. Albanna and C. J. Hatziadoniu, "Harmonic Modeling of Hysteresis Inverters in Frequency Domain," *Power Electronics, IEEE Transactions on*, vol. 25, pp. 1110-1114, 2010.

[82] E. X. Yang, F. C. Lee, and M. M. Jovanovic, "Small-signal modeling of power electronic circuits using extended describing function technique," presented at Proc. Virginia Power Electronics Seminar, 1991.

[83] C. Gaviria, E. Fossas, Gri, x00F, and R. o, "Robust controller for a full-bridge rectifier using the IDA approach and GSSA modeling," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, pp. 609-616, 2005.

[84] D. Basic, "Input Current Interharmonics of Variable-Speed Drives due to Motor Current Imbalance," *Power Delivery, IEEE Transactions on*, vol. 25, pp. 2797-2806, 2010.

[85] E. Louarroudi, R. Pintelon, J. Lataire, and G. Vandersteen, "Estimation of nonparametric harmonic transfer functions for linear periodically time-varying systems using periodic excitations," in *Proc. Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE*, 2011, pp. 1-6, 10.1109/imtc.2011.5944191.

[86] H. Qin and J. W. Kimball, "Closed-Loop Control of DC–DC Dual-Active-Bridge Converters Driving Single-Phase Inverters," *Power Electronics, IEEE Transactions on*, vol. 29, pp. 1006-1017, 2014.

[87] L. Yu-Cheng, C. Ching-Jan, C. Dan, and B. Wang, "A Ripple-Based Constant On-Time Control With Virtual Inductor Current and Offset Cancellation for DC Power Converters," *Power Electronics, IEEE Transactions on*, vol. 27, pp. 4301-4310, 2012.

[88] O. Wasynczuk and S. D. Sudhoff, "Automated state model generation algorithm for power circuits and systems," *Power Systems, IEEE Transactions on*, vol. 11, pp. 1951-1956, 1996.

[89] B. B. Johnson, A. Davoudi, P. L. Chapman, and P. Sauer, "Microgrid dynamics characterization using the automated state model generation algorithm," in *Proc. Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 2758-2761, 10.1109/iscas.2010.5537011.

[90] J. M. Henry and J. W. Kimball, "Switched-Capacitor Converter State Model Generator," *Power Electronics, IEEE Transactions on*, vol. 27, pp. 2415-2425, 2012.

VITA

Luke Dale Watson was born in St. Joseph, Missouri and grew up in Savannah, Missouri. He received his Bachelor's of Science degree in computer engineering from Missouri S&T in 2008, and his Ph.D. in electrical engineering in from Missouri S&T in 2013.

Luke has been a student member of IEEE since 2007. He was awarded the Chancellor's Fellowship and GAANN Fellowship in 2009.