Reactive Power Support Capability of Flyback Micro-inverter with Pseudo-dc Link

by

Edwin Fonkwe Fongang
MSc, Masdar Institute of Science and Technology (2013)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2015

Author______________________________

Department of Electrical Engineering and Computer Science
May 20, 2015

Certified by______________________________

James L. Kirtley
Professor of Electrical Engineering
Thesis Supervisor

Accepted by______________________________

Professor Leslie A. Kolodziejski
Chair of the Department Committee on Graduate Students

Reactive Power Support Capability of Flyback Micro-inverter with Pseudo-dc Link

by
Edwin Fonkwe Fongang

Submitted to the Department of Electrical Engineering and Computer Science
On May 20, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

The flyback micro-inverter with a pseudo-dc link has traditionally been used for injecting only active power in to the power distribution network. In this thesis, a new approach will be proposed to control the micro-inverter to supply reactive power to the grid which is important for grid voltage support. Circuit models and mathematical analyses are developed to explain underlying issues such as harmonic distortion, and power losses, which can limit the reactive power support capability. A novel current decoupling circuit is proposed to effectively mitigate zero crossing distortion. Simulations and experimental results are provided to support the theoretical propositions.

Thesis Supervisor: James L. Kirtley

Title: Professor of Electrical Engineering

Acknowledgements

I would like to thank in a special way my research advisor Professor James L Kirtley for accepting me in to his research group and for providing invaluable guidance in the course of this work. I also appreciate his jovial countenance which has certainly helped make my MIT experience a great one.

I am grateful to all the members of my research group for answering my questions and for their friendship; also, the members of the Laboratory for Electromagnetic and Electronic Systems (LEES) deserve appreciation for I have usually found a helping hand when I called out for help. I must also thank the Skolkovo foundation for funding my research assistantship. Gratitude is expressed to my former advisor Dr. Michael Weidong Xiao, whom I worked with two years ago. Some of the ideas which I have explored in this work initially sparked up in my mind while I was his student at Masdar Institute of Science and Technology, Abu Dhabi.

I am forever thankful to my family. Special thanks go to my dad, *Papa Joe*, my mum, *Mama Joe,* and to my siblings Sandrine, Fritz, Horace, and their respective families. I also cannot forget the love and warmth I have found here in the family of Mr. & Mrs. Cornelius Bella. Thanks for the moral support, and for all the delicious food.

Finally, thank you LORD for the strength to accomplish this work.

List of figures

List of tables

Table of Contents

Chapter 1

Introduction

1.1 The growth of solar photovoltaic energy

Globally, new solar photovoltaic (PV) energy installations grew by 38.4 GW by the end of 2013 [1] to a cumulative 138.9GW of installed PV capacity. Other estimates put the additional installed capacity in 2014 at 40.8GW [2] and predict 57GW of global solar PV demand in 2015 [3]. In fact, solar PV is the fastest growing renewable energy source by installed capacity after hydro and wind power [1]. The main reason for increased PV deployment, and indeed all forms of renewable energy sources, lies in the established fact that the use of conventional carbon-based fuels by our societies is driving up the atmospheric carbon dioxide levels and causing global climate change. The IEA suggests that 27% of total global electricity generation capacity should come from solar in the long term (by 2050) [4]. To put this ambition in to perspective, the Energy Information Administration estimates that the global electricity installed capacity is greater than 5.54TW [5]. Thus global solar PV installed capacity stands at about 2.5%. A huge growth of this resource is therefore expected in the future.

1.2 Micro-inverters and reactive power

AC modules, also known as micro-inverters (MIC), are grid-interactive converters with power ratings generally less than 500W [6]. Compared to traditional centralized inverters, the module-converter integration provides a parallel configuration and independent operation of each photovoltaic (PV) panel. The individual maximum power point tracking (MPPT) algorithm allows local optimization and reduces power losses that result from PV module mismatch and partial shading [7, 8]. Furthermore, the parallel structure of MIC helps prevent the single point failure mechanism, and increases the generation stability [9]. Modeling and analytical studies have predicted the energy yield improvement when the MIC topologies are widely applied to PV power systems [10].

The IEEE Standard 1547 [11] prohibits the active regulation of the voltage at the point of common coupling by any distributed resource (DR) rated at or less than 60MVA. However, as evidenced by the Smart Grid Initiative [12], there is growing anticipation that this will change in the near future and consumers will be allowed to provide ancillary utility grid services such as grid voltage support. To understand the importance of reactive power for grid voltage support, consider the diagram in Fig. 1.

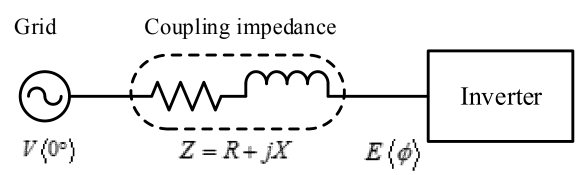

Fig. 1. Inverter system coupled to the grid

To a first order approximation [13], the difference between the grid and inverter voltages depends essentially on the reactive power when the power angle is small and the coupling impedance is mostly inductive, as shown in equation (1.1).

$$\frac{Q}{V} \approx \frac{E-V}{X} \qquad (1.1)$$

While it is well-known that traditional centralized inverters can be controlled for harmonics and voltage support functions [14], the research on micro-inverters has generally not addressed these possibilities. The number of micro-inverters is on the increase, with an estimated 3GW of installed capacity in 2014 [15]. Therefore there is a significant and as yet untapped potential for these devices to contribute to the grid ancillary services such as reactive power. The situation is even more interesting if one considers microgrids which generally have limited actuators for maintaining power quality that is acceptable by traditional grid standards.

1.3 Thesis scope and organization

The flyback with a pseudo-dc link (FMICpseudo-dc) is a single phase MIC topology that has received a lot of attention in the research community [6, 9, 16-20]. It is attractive because of its relatively low component count which offers the possibilities of high power density, lower losses, and reliability [6, 21]. This topology can operate in discontinuous conduction mode (DCM) [17, 18, 20] as well as in continuous conduction mode (CCM) [9, 16, 19]. It is suitable for use with (but not limited to) low input voltages in the range 20V to 55V.

As it will be shown in the first half of chapter 2, there is a need for improved modeling of the converter because of the observations of current distortion around the grid zero-volt crossing [19]. This can limit the amount of reactive power that can be injected in to the grid while maintaining acceptable harmonic distortion levels. Addressing current distortion is important because the IEEE Standard 1547 [11] sets an upper limit on the total demand distortion to 5% for distributed resources supplying linear loads. Therefore circuit models and mathematical analysis will be developed to explain the grid zero-volt crossing phenomenon and mitigation techniques will be examined. Experimental results will also be shown.

In the second half of chapter 2, another contribution of this work will be to show how the device can be controlled to inject reactive power which is important for future grid voltage support. Simulation and experimental results are provided. A current decoupling circuit is proposed to help mitigate the zero-crossing distortion problem when the converter is operated as a controlled reactive power source.

Furthermore, it has been shown in previous work that the converter operation in CCM can have efficiency improvements over the DCM operating region [19]. However, to the author's knowledge, a detailed and compact power loss model does not yet exist for this converter in CCM. In chapter 3 of this thesis a power loss model for the FMICpseudo-dc operating in CCM is developed. The theoretical model is compared with experimental results.

In addition, when the converter operates in DCM [17, 18, 20], in which the current injection is an open loop system, it has been observed that the converter power factor will vary with the parameters of the output CL filter [22]. This phenomenon will be explained with a circuit model and relevant equations. Experimental verification will be used to support the theoretical analysis.

It should be noted that closed-loop controller design for the FMICpseudo-dc in CCM is necessary to shape the output current appropriately. Since this has been accomplished in [9, 16, 19], it is not addressed in this thesis.

Chapter 2

Current Distortion around Grid Zero-Volt Crossing in Flyback AC Module with a Pseudo-DC Link

2.1 Introduction

In the operation of the flyback micro-inverter with a pseudo-dc link (FMICpseudo-dc), some observations have been made in the literature on the distortion of the grid-injected current around the grid voltage zero-crossings. This phenomenon can be especially observed in [19] for a an FMICpseudo-dc and in [23] for a buck converter. This chapter proposes a theoretical explanation for this phenomenon based on Fourier analysis. It is argued that some distortion is inevitable. An attempt is made to alleviate this distortion by including a synchronous rectifier in to the converter topology.

With the addition of the synchronous rectifier, the converter can operate with bi-directional power flow. Taking advantage of this possibility, it is attempted to control the inverter to inject reactive power in to the grid by injecting a current which is out of phase with the grid-voltage. Notching occurs in the injected current which negatively impacts the total harmonic distortion (THD) as it will be seen from the experimental results which are provided to support the theoretical analysis, and simulations. Unless otherwise stated, all symbols used in this chapter are defined in Table I.

Table I. List of symbols used in chapter 2

Symbol	Definition
C_f	Output filter capacitor
$C_{iss,Q2}$	Q_2 switch input capacitance
C_{pv}	Input capacitor
D	Duty cycle of primary-side flyback switch
d	Instantaneous duty cycle in discontinuous conduction mode
D_1	$1 - D$
d_{pk}	Peak duty cycle in discontinuous conduction mode
f_g	Grid frequency
f_{sw}	Switching frequency
h	Subscript used to represent the harmonic order (harmonics of grid frequency)
$I_{1,rms}$, $I_{2,rms}$	Primary and secondary winding rms current respectively.
i_{Cf}	C_f filter capacitor current
i_d	Diode current
$i_{d,avg}$	Average diode current
i_{Lf}	Grid-injected current
$i_{Lf,ac}$	AC component of 'folded' i_{Lf} waveform
$i_{Lf,dc}$	DC component of 'folded' i_{Lf} waveform
$I_{Lf,pk}$	Peak grid-injected current
i_{Lm}	Magnetizing inductance current
$I_{Lm,avg}(t)$	Average magnetizing inductance current during a switching period.
i_{prim} or i_1	Transformer primary winding current
$i_{prim,pk}$	Transformer primary winding peak current
i_{pv}	PV current
I_s	Equivalent current source
$I_{s,ac}$	AC component of the equivalent current source
$I_{s,dc}$	DC component of current source
$I_{s,rms}$	RMS value of equivalent current source
i_{sec}	Secondary winding current
i_{sync}	Synchronous rectifier current
K_I	Controller integral term
K_P	Controller proportional term

L_f	Output filter inductor
L_{lk}	Leakage inductance
L_m	Magnetizing inductance
N	Number of turns in primary winding
n or n_2/n_1	Secondary-to-primary transformer turns ratio
$P_{ac}(t)$	Instantaneous grid-injected power
P_{avg}	Average grid-injected power
$P_s(t)$	Instantaneous equivalent current source power
R_{Cf}	Filter capacitor ESR
R_{Lf}	Filter inductor ESR
R_{prim}	Primary winding resistance
R_{pv}	Dynamic PV resistance
R_{Q1}, R_{Q2}	On-resistance of Q_1 and Q_2 respectively
R_{sec}	Secondary winding resistance
R_T	Resistance of output diode
$R_{unfolder}$	On-resistance of the unfolder switches
T_{hl}	Grid voltage half-period
t_{HL}	Fall time for switching loss computation
t_{LH}	Rise time for switching loss computation
T_{sw}	Switching period
v_{Cf}	Filter capacitor voltage
v_{Cpv}	Voltage across input capacitor
V_F	Forward voltage drop of output diode
v_g	Instantaneous grid voltage
$V_{g,pk}$	Peak grid voltage
$V_{g,rms}$	RMS value of grid voltage
v_{inv}	Flyback pseudo-dc link voltage
V_{PV}	PV open circuit voltage
X_{Cf}	Magnitude of impedance due to C_f
X_{Lf}	Magnitude of impedance due to L_f
ω_g	Angular frequency of grid voltage

2.2 Current distortion

2.2.1 FMICpseudo-dc without synchronous rectifier

The issue of current distortion is important for distributed resources because the quality of the grid-injected current will affect the quality of the grid voltage. As such, the IEEE standard. 1547 [11] sets an upper limit of 5% on the total demand distortion for distributed resources supplying linear loads.

In the FMICpseudo-dc, distortion in the grid-injected current can arise from two main sources: switching harmonics that are not completely filtered out; and the inability of the injected current to faithfully match a non-distorted sinusoidal reference as a result of component physical limitations and/or controller limitations. From a design perspective, the combination of switching frequency and output filter can be chosen appropriately. Therefore, it is the latter cause of distortion (component/controller limitations) which is of interest in this section. A diagram of the FMICpseudo-dc is shown in Fig. 2.

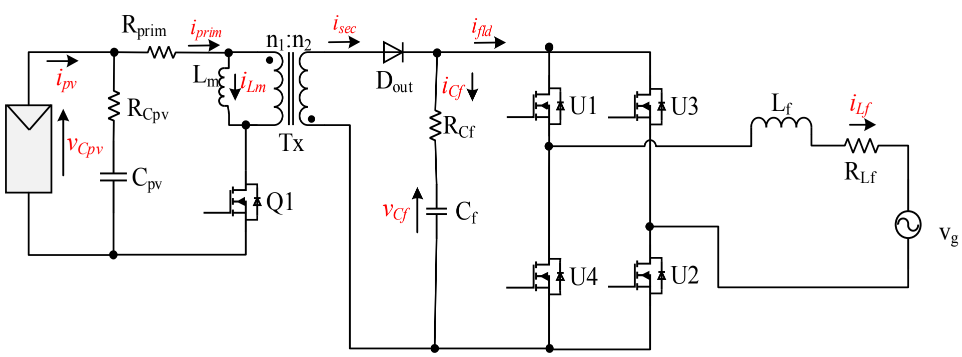

Fig. 2. Schema of FMICpseudo-dc

At this point, it is worth mentioning briefly how the converter operates. The switches U1 – U4 constitute what is referred to as the ‘current-unfolding circuit’. The switch Q1 is the main actively switched component. The flyback output diode is D_{out}. The traditional operation of the converter is briefly explained as follows. Q1 is switched such that a full-wave rectified current with a sinusoidal envelop is produced in the primary and secondary windings of the flyback transformer Tx. These currents i_{prim} and i_{sec}, respectively, are said to be ‘folded’ (meaning they are full-wave rectified waveforms). The current-unfolding unit is synchronized with the grid voltage such that during the positive grid voltage half cycle, switches U1 and U2 are turned on, while U3 and U4 are turned on during the negative half cycle. Thus, the current-unfolder circuit produces a sinusoidal grid-injected current i_{Lf} that is in phase with the grid voltage in the unity power factor case.

For the purpose of the discussion in this section, an equivalent circuit will be developed based on the circuit in Fig. 2. In the equivalent circuit, shown in Fig. 3, the flyback is replaced with a current source I_s. Since, the role of the current-unfolding circuit (U1 – U4) is to always present a positive voltage to the flyback's output, it is now ignored in the equivalent circuit, and the grid is made to appear as a full-wave rectified voltage source. The switching frequency of the FMICpseudo-dc is generally orders of magnitude higher than the grid frequency. Therefore in the time-scale of interest (grid period), the high frequency dynamics are ignored. I_s is the moving average of i_{sec}. The former must be unidirectional because of the presence of the flyback diode in the actual circuit.

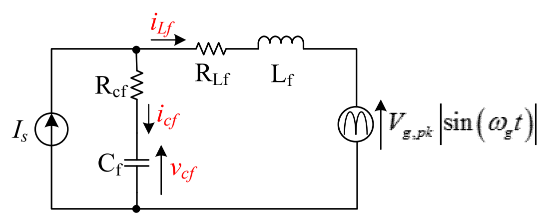

Fig. 3. Equivalent circuit for discussion on current distortion

Since i_{Lf} is the quantity that is being controlled, we start by assuming that it is an ideal full-wave rectified sinusoid. A reduction to absurdity will be used to show that distortion will occur in i_{Lf} around the grid zero volt crossing.

If one assumes a unity power factor operation, then i_{Lf} is in phase with v_g. Both quantities are full-wave rectified sinusoids and can be written in their equivalent Fourier series forms as the system of equations (2.1).

$$\begin{cases} i_{Lf}(t) = \dfrac{2I_{Lf,pk}}{\pi} + \dfrac{4I_{Lf,pk}}{\pi}\displaystyle\sum_{k=1}^{\infty}\left(\dfrac{1}{1-(2k)^2}\right)\cos\left(2k\omega_g t\right) \\ v_g(t) = \dfrac{2V_{g,pk}}{\pi} + \dfrac{4V_{g,pk}}{\pi}\displaystyle\sum_{k=1}^{\infty}\left(\dfrac{1}{1-(2k)^2}\right)\cos\left(2k\omega_g t\right) \end{cases} \tag{2.1}$$

To obtain the required I_s, the principle of superposition is used. The ac response is obtained by evaluating the circuit in Fig. 3 for each harmonic. Solving the circuit in the frequency domain yields:

$$I_{s,ac} = \sum_h i_{Lf,h} + \frac{\sum_h v_{g,h} + \left(R_{Lf} + jX_{Lf,h}\right)\sum_h i_{Lf,h}}{\left(R_{Cf} - jX_{Cf,h}\right)} \tag{2.2}$$

The dc response is:

$$I_{s,dc} = i_{Lf,dc} = \frac{2I_{Lf,pk}}{\pi} \tag{2.3}$$

Combining equations (2.2) and (2.3), the current source I_s can be expressed as:

$$I_s = I_{s,dc} + I_{s,ac} \tag{2.4}$$

Equation (2.4) is computed for the first 20 harmonics and the resulting current source is plotted against angle as shown in Fig. 4.

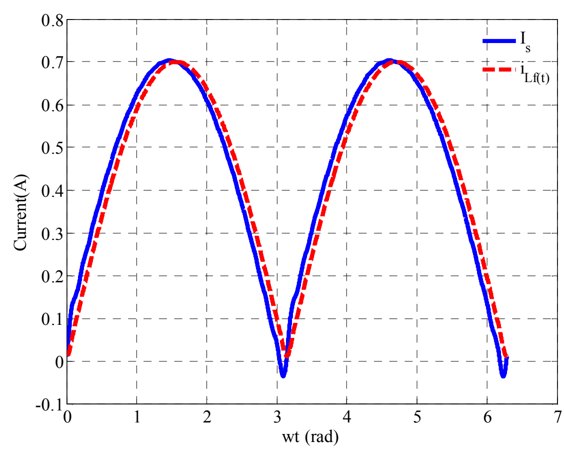

Fig. 4. Resulting current source

It can be seen that for a sinusoidal output current at unity power factor, the equivalent current source I_s becomes negative around the zero-crossings of the grid voltage. Furthermore, it can be seen that I_s is distorted (does not follow a sinusoidal envelope) around the zero-crossing. Indeed, a very high time rate of change of current can be observed at the beginning of the rising edge of I_s. This rate of change tends to infinity as more harmonics are considered. Therefore, two absurdities must be addressed here:

1. i_{sec} cannot be negative because of the diode D_{out}. Therefore, I_s cannot be negative either.

2. i_{sec} is proportional to i_{Lm} during each switching cycle. Therefore the time rate of change of I_s must be finite.

From these observations, it follows that in a closed-loop system, the current controller will not be able to force the output current to follow a sinusoidal reference (precisely around the grid zero volt crossing) which will result in distortion of the grid-injected current. Therefore, for this topology, distortion of the output current is inevitable.

In order to push further this argument, it is assumed that i_{Lf} follows a sinusoidal envelope until the point at which I_s is clipped to zero. The resultant equivalent circuit in this operating region is shown in Fig. 5:

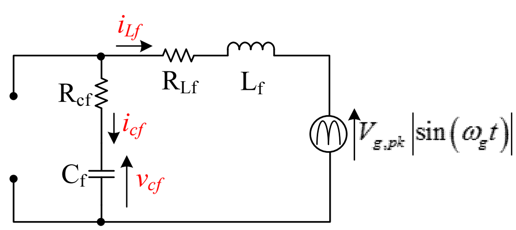

Fig. 5. Resultant equivalent circuit for $I_s = 0$

The second-order non-homogenous linear differential equation in i_{Lf} can be written as:

$$\frac{d^2 i_{Lf}}{dt^2} + \left(\frac{R_{Lf}+R_{Cf}}{L_f}\right)\frac{di_{Lf}}{dt} + \left(\frac{1}{C_f L_f}\right) i_{Lf} = -\frac{\omega_g V_{g,pk}\cos(\omega_g t)}{L_f} \tag{2.5}$$

The characteristic equation of the homogenous equation is:

$$\lambda^2 + \left(\frac{R_{Lf}+R_{Cf}}{L_f}\right)\lambda + \left(\frac{1}{C_f L_f}\right) = 0 \tag{2.6}$$

Solving for the λ parameter gives:

$$\lambda = \frac{-\left(\frac{R_{Lf}+R_{Cf}}{L_f}\right) \pm \sqrt{\left(\frac{R_{Lf}+R_{Cf}}{L_f}\right)^2 - \left(\frac{4}{C_f L_f}\right)}}{2} \tag{2.7}$$

In realistic designs, the discriminant in equation (2.7) will be negative, whereupon the general solution to the homogenous differential equation can be written as:

$$i_{Lf_1} = e^{-\frac{t}{\tau}}\left(A\cos(\alpha t) + B\sin(\alpha t)\right) \tag{2.8}$$

Where:

$$\begin{cases} \tau = \left(\dfrac{2L_f}{R_{Lf} + R_{Cf}}\right) \\ \alpha = \dfrac{1}{2}\sqrt{\left(4\omega_k^2\right) - \left(\dfrac{2}{\tau}\right)^2} \text{ and } (A,B) \in \mathbb{R}^2 \\ \omega_k = \dfrac{1}{C_f L_f} \end{cases} \tag{2.9}$$

The particular solution can be represented by the equations in (2.10).

$$\begin{cases} i_{Lf_2} = E\sin(\omega_g t) + F\cos(\omega_g t) \\ E = -\dfrac{V_{g,pk}\omega_g^2\left(\dfrac{2}{\tau}\right)}{L_f\left(\omega_k^2 - \omega_g^2\right)^2} \\ F = -\dfrac{V_{g,pk}\omega_g}{L_f\left(\omega_k^2 - \omega_g^2\right)} \end{cases} \tag{2.10}$$

The general solution to the non-homogenous differential equation is therefore:

$$i_{Lf} = e^{-\left(\frac{t}{\tau}\right)}\left(A\cos(\alpha t) + B\sin(\alpha t)\right) + E\sin(\omega_g t) + F\cos(\omega_g t) \tag{2.11}$$

The initial conditions at time t_0 are considered to be the values of the grid-injected current, i_{Lf}, and the capacitor voltage, v_{Cf}, at the moment when I_s is zero for the first time. They are denoted I_{Lf0} and V_{Cx0} respectively. The coefficients A, and B are then determined and are shown by the system of equations in (2.12):

$$\begin{cases} B = \dfrac{V_{Cx0} - \left(\left(R_{Lf} + R_{Cf} \right) I_{Lf0} + v_{g0} + L_f \left(\gamma k_1 + k_3 \right) \right)}{L_f \left(-\tan\left(\alpha t_0\right) k_1 + k_2 \right)} \\ A = \gamma - B \tan\left(\alpha t_0\right) \\ \gamma = \dfrac{e^{\left(\frac{t_0}{\tau}\right)} \left(I_{Lf0} - E \sin\left(\omega_g t_0\right) - F \cos\left(\omega_g t_0\right) \right)}{\cos\left(\alpha t_0\right)} \\ k_1 = e^{\left(-\frac{t_0}{\tau}\right)} \left(-\dfrac{1}{\tau} \cos\left(\alpha t_0\right) - \alpha \sin\left(\alpha t_0\right) \right) \\ k_2 = e^{\left(-\frac{t_0}{\tau}\right)} \left(-\dfrac{1}{\tau} \sin\left(\alpha t_0\right) + \alpha \cos\left(\alpha t_0\right) \right) \\ k_3 = E\omega_g \cos\left(\omega_g t_0\right) - F\omega_g \sin\left(\omega_g t_0\right) \end{cases} \tag{2.12}$$

The general solution of equation (2.11) can now be plotted against angle. Fig. 6 shows the resulting output current plotted over a full grid period. Again, it is assumed that i_{Lf} will follow a sinusoidal envelope except where I_s would have been negative. The solution calculated by equation (2.11) only is highlighted in black and shown in more detail in Fig. 7.

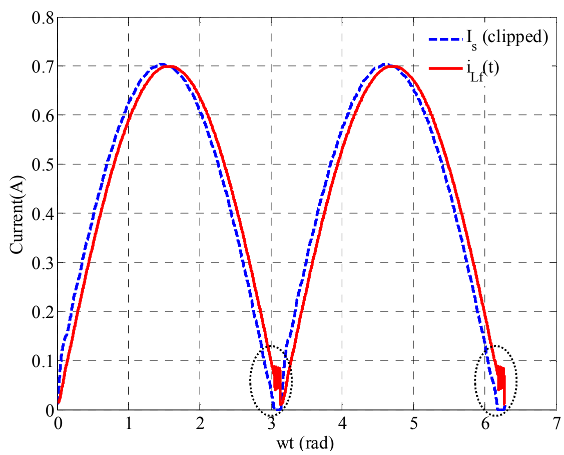

Fig. 6. i

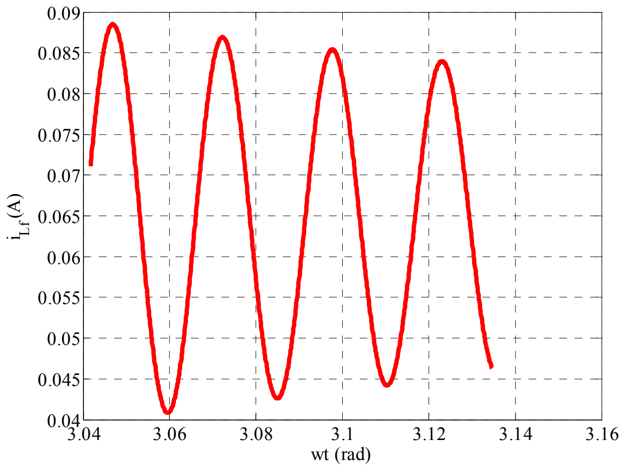

Fig. 7. i_{Lf} *during the period when* $I_s = 0$

In order to better illustrate the zero-crossing distortion issue, experimental waveforms are shown in Fig. 8 and Fig. 9. These waveforms are obtained with a prototype FMICpseudo-dc switching in CCM. A photo of the digitally-controlled prototype is provided in Fig. 10. In both figures, the orange waveform represents v_{Cf}; cyan is v_g; green is i_{Lf} unfolded i.e. the actual grid-injected current; the pink waveform is the turn-on signal for the unfolder switches U_1/U_2.

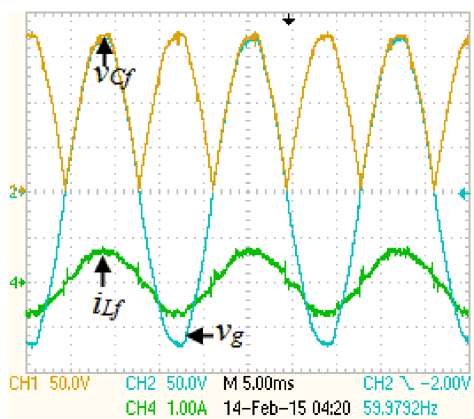

Fig. 8. Experimental waveforms to illustrate distortion around grid zero-crossing

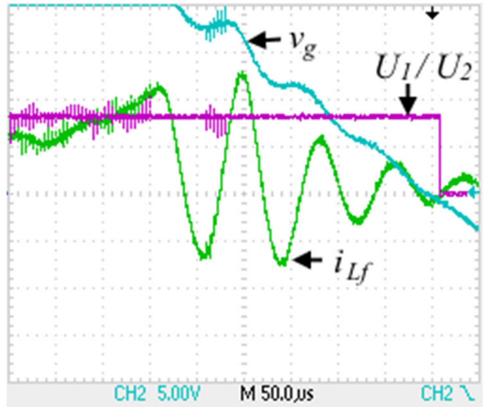

Fig. 9. Zoom-in to illustrate distortion around zero-crossing

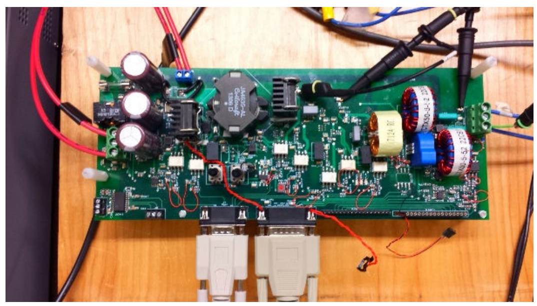

Fig. 10. Photo of FMICpseudo-dc prototype

In Fig. 9, the distortion in i_{Lf} can be seen as v_g approaches zero. However, unlike the theoretical model, the experimental observations show larger amplitude oscillations and a slightly lower pseudo-frequency. A couple of possible reasons might explain the observed disparity:

1. There is a tracking error in the current controller such that i_{Lm} and i_{Lf} deviate from the expected waveforms. Thus the initial conditions used in equation (2.11) are not accurate.
2. It can be observed that v_g is distorted, whereas a perfectly sinusoidal v_g was assumed in the theoretical analysis. Furthermore, a higher impedance behind the ideal grid supply might explain the slightly lower pseudo-frequency.

Fig. 11 shows the theoretical waveform of i_{Lf} around the grid zero-volt crossing when initial conditions of equation (2.11) are modified to match the observed waveforms.

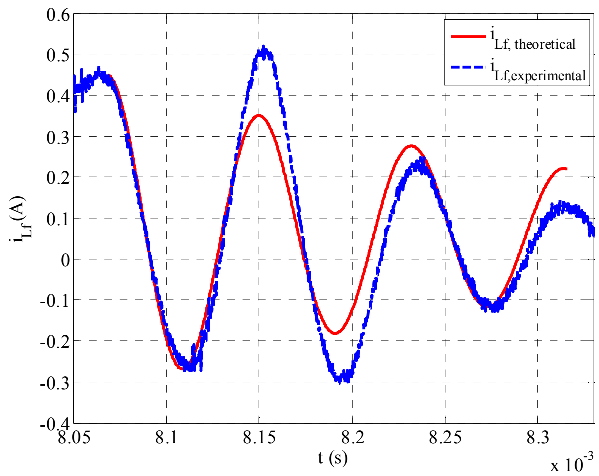

Fig. 11. Theoretical and experimental i_{Lf} *with adjusted initial conditions around grid voltage zero-crossing*

It can be observed in Fig. 9 that, in addition to the distortion on the falling edge of every half-cycle (which has been predicted theoretically), distortion also occurs on the rising edge of every half-cycle. The reason for this is that there is a physical limitation on the time rate of change of I_s. Even though this particular phenomenon has not been numerically computed, the model that has been developed hitherto, together with the experimental waveforms, prove that it is not possible to avoid some distortion around the grid zero-volt crossing in the classical FMICpseudo-dc.

2.2.2 FMICpseudo-dc with synchronous rectifier

It is hypothesized that a synchronous rectifier installed across D_{out} can mitigate the zero-crossing issue by allowing I_s to have a negative value at certain instants. The modified circuit is shown in Fig. 12.

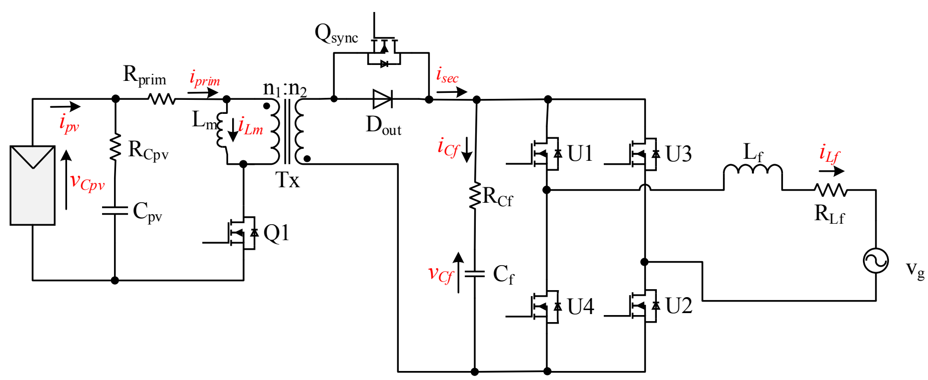

Fig. 12. Modified FMICpseudo-dc with synchronous rectifier

Waveforms are shown in Fig. 13 for an 80W output. Zero-crossing distortion is still seen. Careful observation shows that the distortion occurs mainly on the rising edge of each half cycle. The falling edge of a half cycle appears not to have any significant distortion. This is consistent with the previous explanations because I_s must have a finite time rate of change (which causes distortion on the rising edge of a half-cycle), but is allowed to have negative values (leading to less distortion on the falling edge of the half-cycle). However, the THD is not improved. In fact a THD of 7% is observed in the FMICpseudo-dc with a synchronous rectifier compared to a THD of 5% for the FMICpseudo-dc without the synchronous rectifier. Therefore, in the case of the FMICpseudo-dc with a synchronous rectifier, the distortion around the grid zero volt crossing has a different origin.

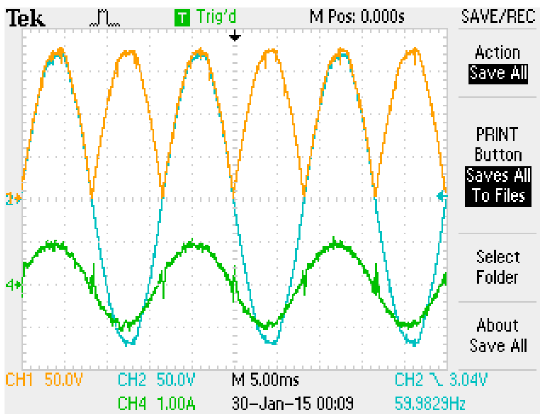

Fig. 13. Waveforms from FMICpseudo-dc with synchronous rectifier

The synchronous rectifier allows for full bi-directionality of power. It is interesting to observe the output current as its phase is modified. Simulation results in Fig. 14 show different waveforms for an output reference power of 80VAR leading while Fig. 15 shows experimental waveforms for the same output power. Severe notching can be observed in i_{Lf} around the grid-zero crossing.

This is to be expected because in Fig. 14(b), it can be seen that the magnetizing inductance current (analogous to I_s) has to reverse to an equal and opposite value instantaneously. The output current becomes distorted during this transition.

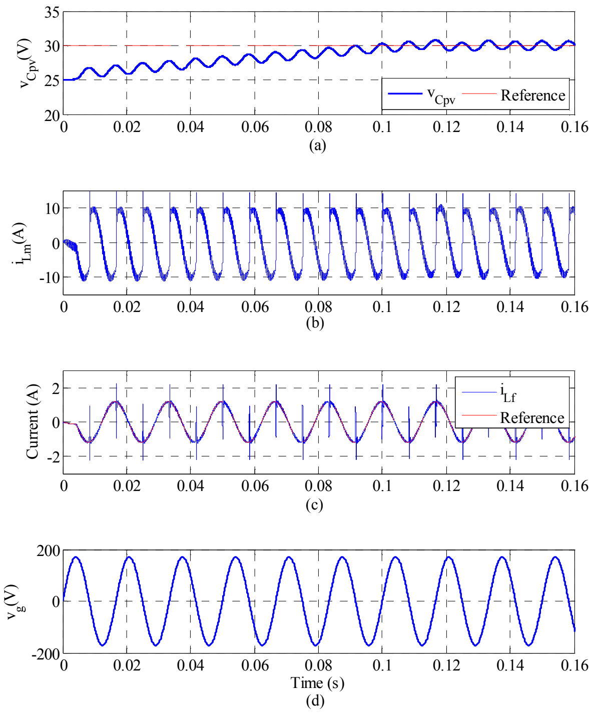

Fig. 14. Simulation waveforms for FMICpseudo-dc with synchronous rectifier at 80VAR leading

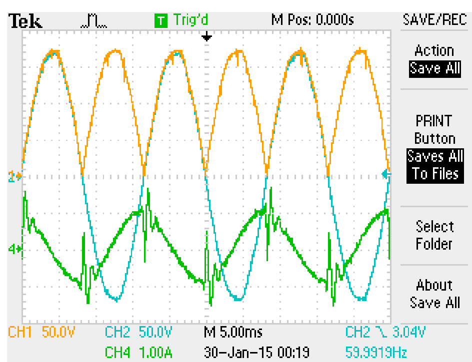

Fig. 15. FMICpseudo-dc with synchronous rectifier injecting 80VAR leading

The THD of the grid-injected current is computed for different values of phase of i_{Lf} while maintaining the apparent power at 80VA. A plot is made of THD vs VAR is shown in Fig. 16.

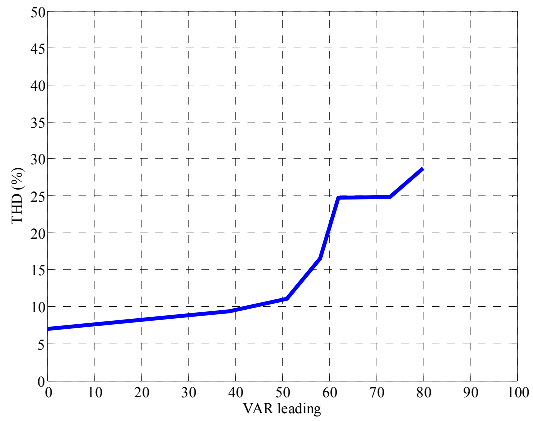

Fig. 16. Plot of THD vs VAR (leading) for FMICpseudo-dc with synchronous rectifier

Circuit parameters for the numerical analysis, simulation, and experimental set-up are provided in Table II.

Table II. FMICPseudo-dc parameters for current-distortion simulation

C_f	1uF
C_{pv}	3*1800 uF
f_g	60 Hz
f_{sw}	250 kHz
K_I	201.5
K_P	0.042
L_f	115 uH
L_m	28 uH
n	6
R_{Cf}	0.2 Ω
R_{Lf}	50 mΩ
R_{prim}	8 mΩ
R_{pv}	0.1 Ω
R_{Q1}	25 mΩ
R_{Q2}	0.35 mΩ
R_{sec}	0.106 Ω
$R_{unfolder}$	0.24 Ω
$V_{g,pk}$	170 V
V_{pv}	30 V

Perhaps it is not clear enough why this notching occurs. Why is it not sufficient to phase shift the primary winding current (or magnetizing inductance current) by the amount of output current phase desired? In order to conceptually explain the phenomenon, consider the hypothetical waveforms in Fig. 17. Here, the usual physical quantities are multiplied by constants (k_x) to show that they are proportional to real world quantities. (U1/U2) represents the gate turn-on signal to the unfolder switches U1/U2 (complementary to U3/U4). It can be seen that i_{Lm} has been shifted such that it leads v_g by 45 degrees in an attempt to obtain a grid-injected current i_{Lf} which, it is expected, will be equally phase-shifted with respect to v_g. However, because the current-unfolder (U1-U4) is inflexible and locked to the grid frequency, the wrong portions of i_{Lm} are unfolded, resulting in an unacceptable output current.

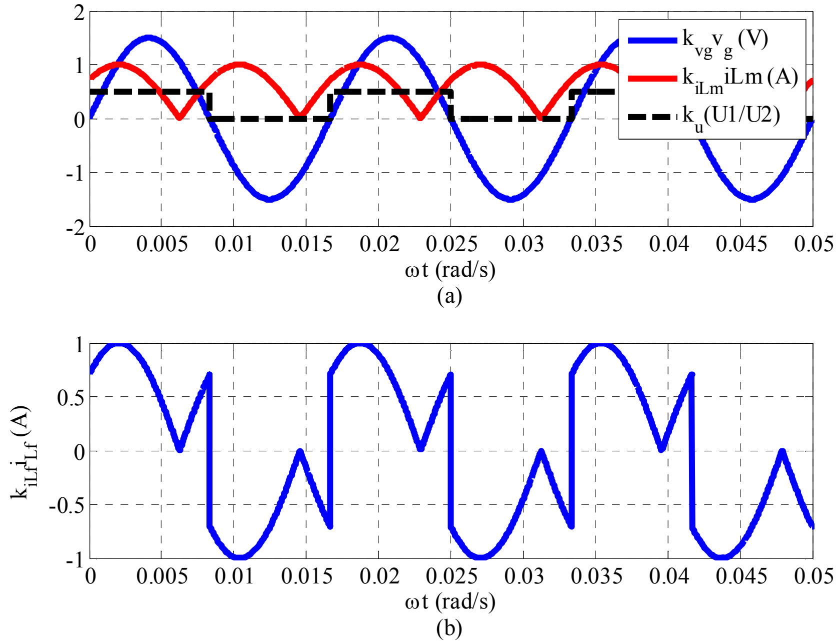

Fig. 17. Hypothetical waveforms with simply shifting i_{Lm}

Now, consider the waveforms of Fig. 18. Here, i_{Lm} is reflected about the time axis at the grid zero-volt crossings, such that when the unfolder switches U3/U4 are turned on during the grid negative half cycle, the correct output current polarity is observed. Since it is not possible to instantaneously reverse i_{Lm}, distortion in i_{Lf} will be inevitable. It should be noted that a similar explanation can be deduced for the case where i_{Lm} lags v_g. It is not attempted to compute this distortion analytically.

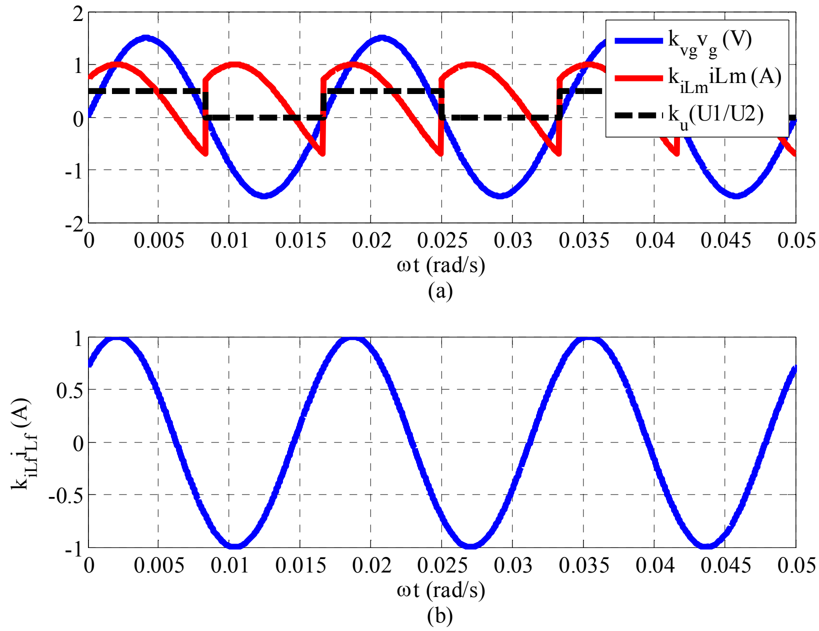

Fig. 18. Hypothetical waveforms with shifting i_{Lm} and reflecting about time axis at grid zero-crossings

2.3 Current decoupling circuit

It has been shown previously that distortion in the grid injected current around the grid zero voltage crossing is severe and unacceptable per the total demand distortion (TDD) standard. One way of eliminating the notching is proposed in this section. The idea is to decouple the magnetizing inductance current i_{Lm} from the grid-injected current i_{Lf} during a short period of time, Δt_c (which we henceforth refer to as the *commutation time, period* or *zone*), around the grid zero voltage crossing. Thus, i_{Lm} can be controlled independently of i_{Lf}. This means that the impossible requirement for i_{Lm} to reverse instantaneously can be relaxed. A more gentle change in i_{Lm} will be allowed. Meanwhile, an auxiliary circuit at the output supplies the required grid current i_{Lf} independently of i_{Lm}. The modified FMICpseudo-dc circuit is shown in Fig. 19. Additional components have been highlighted in blue.

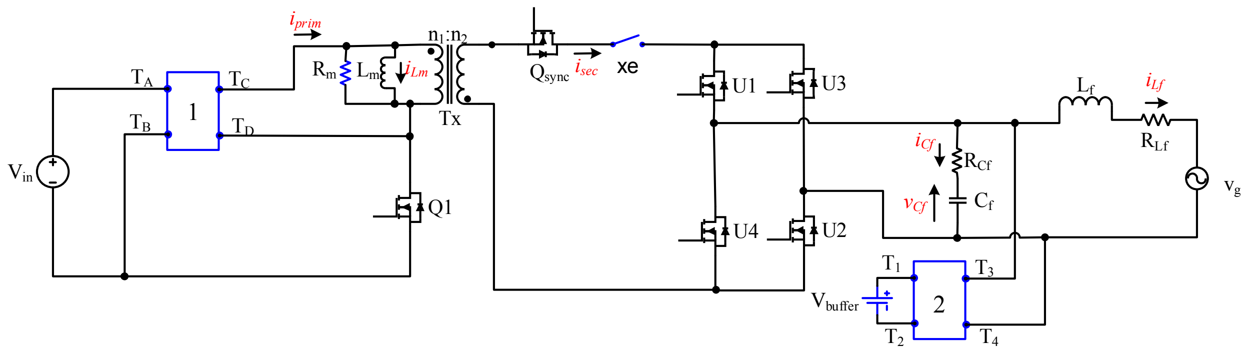

Fig. 19. FMICpseudo-dc with current decoupling

R_m is the magnetizing resistance and has been included in order to obtain a more 'physical' design which can be more easily simulated.

The details of block 1 are shown in Fig. 20, while those of block 2 are shown in Fig. 21. The components *x1* – *x4*, *xa* – *xd*, and *xe* are bi-directional switches and they can be implemented as shown in Fig. 22.

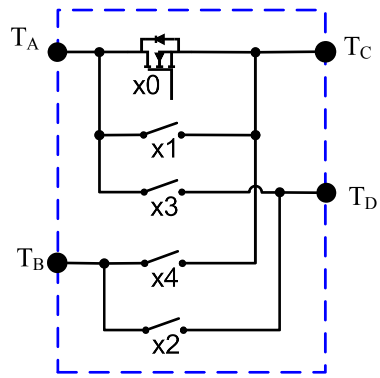

Fig. 20. Details of block 1

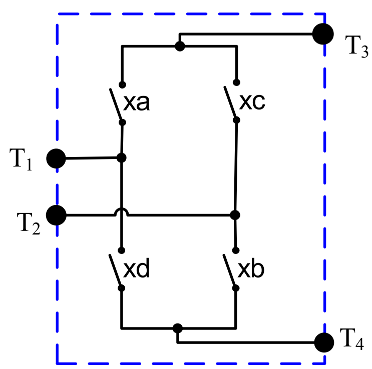

Fig. 21. Details of block 2

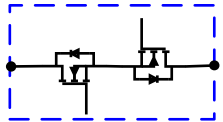

Fig. 22. Implementation of bi-directional switch

The current decoupling circuit's operation can be explained as follows for the case where the reference output current $i_{Lf,ref}$ lags v_g:

- Normal converter operation (with switches *x0*, and *xe* closed) until when the grid voltage $v_g(t)$ is 'close' to the zero crossing. The term 'close' here is used to mean that the grid angle falls inside the commutation zone, i.e.

$$kT_{hl} - \frac{\Delta t_c}{2} \le t \le kT_{hl} + \frac{\Delta t_c}{2} \tag{2.13}$$

Where k has values 1, 2, 3, …

- Open switches *x0*, and *xe*. Using hysteresis control, force i_{Lm} to reverse to its required value by means of switches *x3* and *x4*. Note that *x1* and *x2* are open during this time. Concurrently, using hysteresis control, force i_{Lf} to follow the reference current by operating switches xa, and xb. This process continues for the commutation time, i.e for a duration Δt_c.
- When the grid voltage is out of the commutation zone, return to normal converter operation.

The same description above holds for the case when $i_{Lf,ref}$ leads v_g except that *x1*, and *x2* are operating (instead of *x3* and *x4*), and *xc* & *xd* are operating (instead of *xa* & *xb*). It should be noted that all the switches of the traditional FMICpseudo-dc including the current-unfolder are turned off during the commutation period.

2.3.1 Simulation of FMICpseudo-dc with current decoupling circuit

To demonstrate the current decoupling concept, the circuit in Fig. 19 is simulated in SIMULINK. In order to relax the simulation time constraints, the circuit parameters are modified so that a switching frequency of 75kHz can be used. The updated circuit parameters are shown in

Table III.

Table III. FMICPseudo-dc parameters for current distortion simulation

C_f	1uF
f_g	60 Hz
f_{sw}	75 kHz
L_f	200 uH
L_m	100 uH
n	6
R_{Cf}	2 Ω
R_{Lf}	0.2 Ω
$V_{g,pk}$	170 V
V_{in}	50 V
R_m	1 MΩ
$i_{Lf,ref}$	1.4A peak, 90° lagging
V_{buffer}	10V
Δt_c	155.2 us

The simulation waveforms are shown in Fig. 23 where i_{Lf} has a THD of 4%. This is significantly better than the traditional case. Indeed if no current decoupling circuit is used the resulting distortion as shown in the simulation waveforms in Fig. 24 results in a THD of 49% for i_{Lf}.

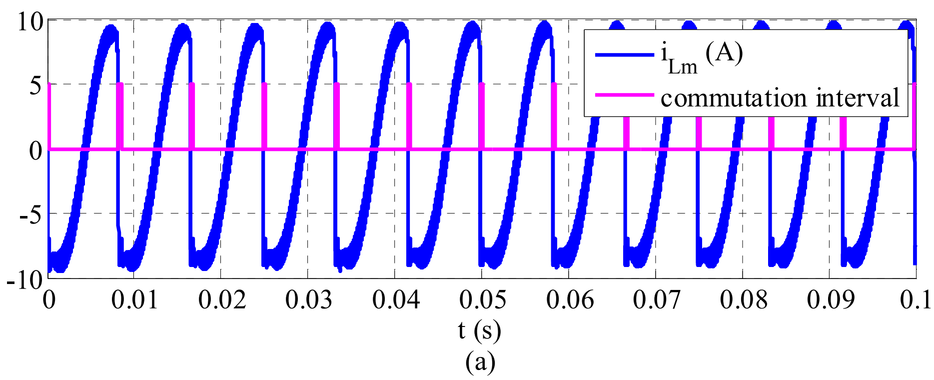

Fig. 23. Simulation waveforms for FMICpseudo-dc with current decoupling circuit

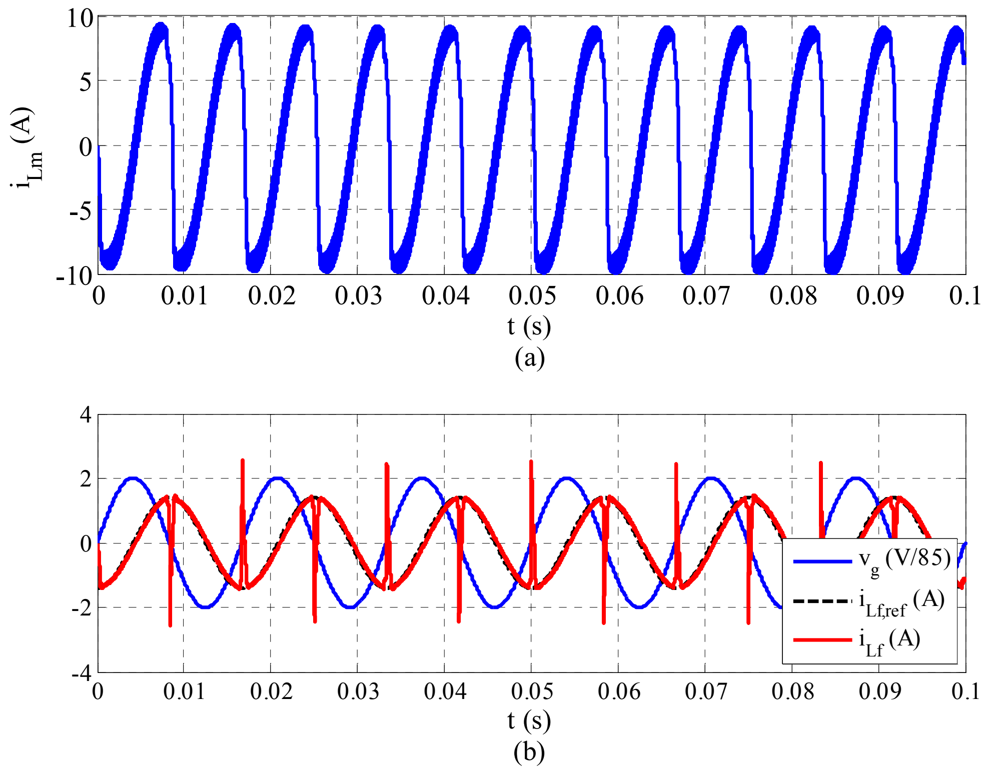

Fig. 24. Simulation waveforms for FMICpseudo-dc without current decoupling circuit

Significant improvements are also achieved in the case where i_{Lf} leads v_g. From these simulation results, it appears that the current decoupling circuit provides a viable solution which allows the FMIC pseudo-dc to deliver desired amounts of reactive power with limited distortion. In the next subsection section, some practical design considerations of the additional units are discussed.

2.3.2 Practical design considerations: V_{buffer}, circuit complexity, and impact on losses

It is obvious that including the current decoupling circuit introduces additional complexity to the system. In addition, it has not yet been discussed how V_{buffer} can be obtained. Also, the additional components will have an impact on converter efficiency. In this section, these issues are addressed, for the most part, in a qualitative manner.

2.3.2.1 V_{buffer}

V_{buffer} provides the energy required to keep the output current close to its reference during the commutation interval. Ignoring losses, as well as the current drawn by C_f, the amount of energy that must be provided by V_{buffer} during Δt_c can be expressed as:

$$\Delta E_c = \left| \int_{T_{hl}-\frac{\Delta t_c}{2}}^{T_{hl}} v_g(t) i_{Lf}(t) \right| + \left| \int_{T_{hl}}^{T_{hl}+\frac{\Delta t_c}{2}} v_g(t) i_{Lf}(t) \right| \tag{2.14}$$

Where it is assumed that the commutation time is equally attributed on either side of the grid voltage zero crossing.

Assuming v_g and i_{Lf} are expressed as:

$$\begin{cases} v_g = V_{g,pk} \sin\left(\omega_g t\right) \\ i_{Lf} = I_{Lf,pk} \sin\left(\omega_g t + \varphi\right) \end{cases} \tag{2.15}$$

Then equation (2.14) can be further written as:

$$\begin{aligned} \Delta E_c = & \frac{V_{g,pk} I_{Lf,pk}}{2} \left| \left(\frac{\Delta t_c}{2}\right) \cos\varphi + \left(\frac{1}{2\omega_g}\right) \left(\sin\left(2\omega_g \left(T_{hl} - \frac{\Delta t_c}{2}\right) + \varphi\right) - \sin\left(2\omega_g T_{hl} + \varphi\right) \right) \right| \\ & + \frac{V_{g,pk} I_{Lf,pk}}{2} \left| \left(\frac{\Delta t_c}{2}\right) \cos\varphi + \left(\frac{1}{2\omega_g}\right) \left(\sin\left(2\omega_g T_{hl} + \varphi\right) - \sin\left(2\omega_g \left(T_{hl} + \frac{\Delta t_c}{2}\right) + \varphi\right) \right) \right| \end{aligned} \tag{2.16}$$

For the parameters in Table III, $\Delta E_c = 540.142\ \mu J$.

V_{buffer} can be implemented with an appropriate capacitor, C_{buffer}. For a desired average-to-peak capacitor ripple Δv_{buffer}, the required capacitance can be computed as:

$$C_{buffer} = \frac{\Delta E_c}{2 \Delta v_{buffer} V_{buffer,avg}} \tag{2.17}$$

Where $V_{buffer,avg}$ is the average (steady state) capacitor voltage (chosen in Table III as 10V).

For a 1% average-to-peak ripple, it can be shown that $C_{buffer} = 270.1\mu F$.

C_{buffer} can be charged by using an auxiliary converter. In order not to lose the galvanic isolation of the FMICpseudo-dc, a 'baby'-forward converter can be used. Its design power rating need not be high because the duty ratio of the buffer capacitor is quite low. During a grid half-period, the buffer capacitor will only be in use for Δt_c. Indeed, neglecting losses, a lower bound on the power rating of the 'baby'-converter, P_{aux} can be computed as follows:

$$P_{aux} = \frac{\Delta E_c}{T_{hl}} = \frac{540.142\mu J}{\left(\frac{1}{60\times 2}s\right)} = 65mW \qquad (2.18)$$

2.3.2.2 Circuit complexity

It is clear from Fig. 19 to Fig. 22 that additional complexities are introduced in to the system in order to reduce the current distortion. Table IV summarizes the additional components and controls and provides some comments.

Table IV. Comments on additional components and controls

Component	Quantity	Comments
Bi-directional switches in block 1 (*x1 – x4*; see Fig. 19)	4	Low duty ratio; operates only during Δt_c. During operation, switches have fast-switching action. Each switch should be able to block V_{in}.
Discrete mosfet in block 1 (*x0*; see Fig. 19)	1	Always on, except during Δt_c. During commutation, it is completely off. This switch should be chosen for slow action and low on-resistance.
Bi-directional switch on pseudo-dc link (*xe*; see Fig. 19)	1	
Bi-directional switches in block 2 (*xa – xd;* see Fig. 19)	4	Low duty ratio; operates only during Δt_c. During operation, switches have fast-switching action. Each switch must be able to block full grid voltage.
Hysteresis controllers	2	Hysteresis controllers must be implemented for controlling i_{Lm}, and i_{Lf} during commutation interval. Since i_{Lm} is not measureable, it must be estimated.
C_{buffer} (V_{buffer})	1	Buffer capacitor can be implemented easily if a low buffer voltage is selected.
'Baby' forward converter	1	Very low required power rating. Must be designed to maintain V_{buffer}.

2.3.2.3 Additional losses

The current decoupling elements will introduce additional losses in the system. However, given that the duty ratio of the fast-acting switches is $\Delta t_c/T_{hl}$ (≈ 1.86% for the parameters in Table III) which is quite low, one would not expect their impact on overall system efficiency to be significant. Besides, apart from switches *x0* and *xe*, all other current decoupling elements can be turned off when the converter operates in unity power factor mode and only turned on when it is desired to provide a certain amount of reactive power.

Since switches *x0* and *xe* are always conducting except during commutation, they must be chosen to have very low conduction losses. Low on-resistance switches tend to be slower acting, but this is not an issue here because the switches require little switching action. However, *x1 – x4* and *xa – xd* must be fast-acting which is required to do accurate hysteresis control of i_{Lm} and i_{Lf}. Fast-acting switches tend to have higher conduction losses, but no significant impact on efficiency is expected because the fast switches are only operational for a fraction of the grid cycle (typically less than 2% of the half grid period).

Chapter 3

A Powertrain Loss Model for the Flyback AC Module with Pseudo-dc Link in Continuous Conduction Mode

3.1 Introduction

It has been shown in previous work that the flyback micro-inverter with pseudo-dc link (FMICpseudo-dc) can operate in discontinuous conduction mode (DCM) [17, 18, 20] and in continuous conduction mode (CCM) [9, 16, 19]. In particular, in [19], it is shown that the converter operation in CCM can have efficiency improvements over the DCM operating region. From an engineering perspective, it is important to be able to accurately predict the converter efficiency during the design stage. A DCM power loss model has been proposed in [20]. However, to the author's knowledge, a CCM power loss model has not been suggested. This chapter attempts to fill that gap by developing a theoretical powertrain loss model for the FMICpseudo-dc in CCM. The theoretical evaluation is compared with experimental results for a 110W digitally-controlled prototype. The loss model shows a good match with the experimental results in the mid-to-nominal power regions.

The rest of this chapter is divided in to three sections. In the second section, the theoretical loss model is developed for the FMICpseudo-dc in CCM. The loss model includes conduction losses, switching losses, transformer core loss, and leakage inductance loss. A note is made on taking in to account the measured heat sink temperatures of the switching elements, from which the internal junction temperatures are estimated. In section 3, the loss model is compared with experimental observations. The conclusion follows and summarizes the main points. It also provides indications for improvements.

A schema of the FMICpseudo-dc used in the analysis is shown in Fig. 25.

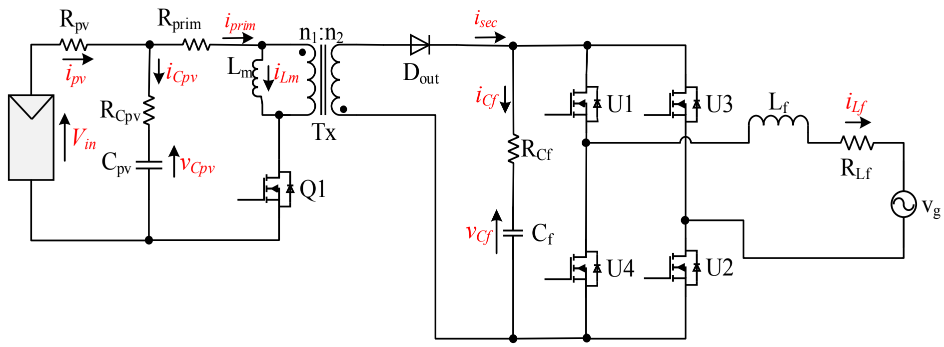

Fig. 25. Schema of FMICpseudo-dc

Unless otherwise stated, all symbols used in this chapter are defined in Table V.

Table V. List of symbols used in chapter 3

Symbol	Definition
A_c	Effective core area
C_f	Output filter capacitor
C_{pv}	Input capacitor
D	Duty cycle of primary-side flyback switch
d	Instantaneous duty cycle in discontinuous conduction mode
D_1	$1 - D$
d_{pk}	Peak duty cycle in discontinuous conduction mode
f_g	Grid frequency
f_{sw}	Switching frequency
$G_{m,Q1}$, $G_{m,Q2}$	Switch Q_1, Q_2 transconductance resp.
h	Subscript used to represent the harmonic order (harmonics of grid frequency)
$I_{1,avg}$	Average transformer primary winding current
$I_{1,avg,pk}$	Peak average (at switching frequency) current in primary winding
$I_{1,pk,fund}$	Amplitude of the fundamental of the primary winding current (at switching frequency)

$I_{1,rms}$, $I_{2,rms}$	Primary and secondary winding rms current respectively.
i_{Cf}	C_f filter capacitor current
i_d	Diode current
$i_{d,avg}$	Average diode current
i_{Lf}	Grid-injected current
$i_{Lf,ac}$	AC component of i_{Lf}
$i_{Lf,dc}$	DC component of i_{Lf}
$I_{Lf,pk}$	Peak grid-injected current
i_{Lm}	Magnetizing inductance current
$I_{Lm,avg}(t)$	Average magnetizing inductance current during a switching period.
i_{prim} or i_1	Transformer primary winding current
$i_{prim,pk}$	Transformer primary winding peak current (instantaneous)
i_{pv}	Input current
i_{sec}	Secondary winding current
L_f	Output filter inductor
L_{lk}	Leakage inductance
L_m	Magnetizing inductance
N	Number of turns in primary winding
n or n_1/n_2	Secondary-to-primary transformer turns ratio
$P_{ac}(t)$	Instantaneous grid-injected power
P_{avg}	Average grid-injected power
$P_s(t)$	Instantaneous equivalent current source power
$Q_{g(sw)}$	Mosfet gate charge
R_{Cf}	Filter capacitor ESR
$R_{gate\_HL,Q1}$	Turn off external gate resistance of Q_1
$R_{gate\_LH,Q1}$	Turn on external gate resistance of Q_1
$R_{int,Q1}$	Internal gate resistance of switch Q_1
R_{Lf}	Filter inductor ESR
R_{prim}	Primary winding resistance
$R_{prim,ac}$	Equivalent primary winding ac resistance
R_{pv}	Input resistance
R_{Q1}	On-resistance of Q_1
R_{sec}	Secondary winding resistance

$R_{sec,ac}$	Equivalent secondary winding ac resistance
R_T	Resistance of output diode
$R_{unfolder}$	On-resistance of the unfolder switches
T_{hl}	Grid voltage half-period
t_{HL}	Fall time for switching loss computation
t_{LH}	Rise time for switching loss computation
T_{sw}	Switching period
v_{Cf}	Filter capacitor voltage
v_{Cpv}	Voltage across input capacitor
$V_{diode,HL}$	Forward voltage of gate turn-off diode
$V_{driver,HL}$	Gate driver turn-off supply voltage
$V_{driver,LH}$	Gate driver turn-on supply voltage
V_F	Forward voltage drop of output diode
v_g	Instantaneous grid voltage
$V_{g,pk}$	Peak grid voltage
$V_{g,rms}$	RMS value of grid voltage
V_{in}	Converter input voltage (constant)
v_{inv}	Flyback pseudo-dc link voltage
$V_{sp,Q1}$	Switching point voltage for Q_1
$V_{th,Q1}$	Q_1, gate threshold voltage
X_{Cf}	Magnitude of impedance due to C_f
X_{Lf}	Magnitude of impedance due to L_f
α, β, k	Transformer core Steinmetz parameters
δ	Skin depth
μ_{Cu}	Magnetic permeability of Copper
$\rho_{Cu,T°}$	Resistivity of Copper at temperature T°
ϕ_{wire}	Diameter of wire
ω_{hl}	Double grid angular frequency

3.2 Power loss modeling in CCM

In this section, a power loss model for the CCM operation is proposed. The breakdown of the losses considered is as follows: conduction losses, switching losses, transformer core loss, and leakage inductance power loss.

3.2.1 Conduction losses

The conduction losses that are considered include: primary winding loss which includes the Tx primary copper loss, the on-state conduction loss of Q_1 and the loss in R_{prim}; secondary winding losses including Tx secondary copper loss, D_{out} forward voltage and conduction losses; power loss in C_f capacitor ESR; power loss in L_f inductor ESR; unfolder conduction loss. In order to compute these losses, the rms currents must be calculated in the input, primary winding, secondary winding, filter capacitor, and current unfolder. Losses due to the skin effect are considered in the primary and secondary windings of the transformer Tx. In order to make the problem more tractable, it is assumed that i_{Lf} is a pure sinusoid, and that the input voltage is constant, V_{in}.

3.2.1.1 Primary winding loss

The rms current in the primary winding of the flyback transformer is determined as follows.

Fig. 26 shows the primary winding current i_{prim} for an arbitrary switching frequency (assumed to be very high relative to grid frequency). The switching ripple in the magnetizing inductance current (red dotted line) is neglected. Some quantities are also shown in the figure which will be useful later in computing the losses.

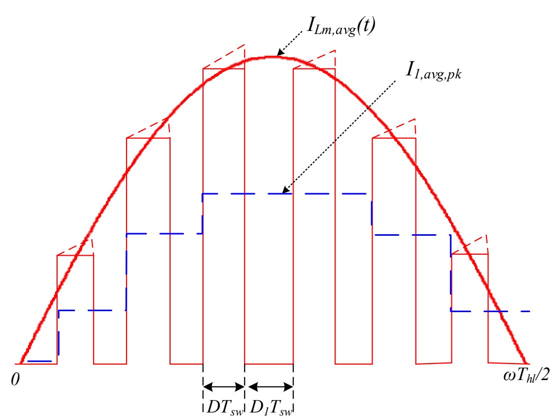

Fig. 26. Primary winding current in CCM

The primary winding rms current can be expressed as:

$$I_{1,rms} = \sqrt{\frac{1}{T_{hl}}\int_{0}^{T_{hl}} i(t)_1^2\,dt} = \sqrt{\frac{1}{T_{hl}}\left[\int_{0}^{T_{sw}} i(t)_1^2\,dt + \int_{T_{sw}}^{2T_{sw}} i(t)_1^2\,dt + \ldots + \int_{T_{hl}-T_{sw}}^{T_{hl}} i(t)_1^2\,dt\right]} \tag{3.1}$$

Equation (3.1) can be further expressed as:

$$\begin{cases} I_{1,rms} = \sqrt{\frac{1}{T_{hl}}\left[\int_{0}^{T_{sw}} i(t)_1^2\,dt + \int_{T_{sw}}^{2T_{sw}} i(t)_1^2\,dt + \ldots + \int_{(p-1)T_{sw}}^{pT_{sw}} i(t)_1^2\,dt + \xi\right]} \\ \left(\forall \xi \in \mathbb{R} : \lim_{p\to\infty} \xi = 0\right) \\ p = \text{Whole part of } \left(\frac{T_{hl}}{T_{sw}}\right) \end{cases} \tag{3.2}$$

Whereupon the rms current will be expressed as:

$$I_{1,rms} = \sqrt{\frac{1}{T_{hl}}\sum_{q=1}^{p}\left(\int_{(q-1)T_{sw}}^{qT_{sw}} i(t)_1^2\,dt\right)} = \sqrt{\frac{T_{sw}}{T_{hl}}\sum_{q=1}^{p}\left(\frac{1}{T_{sw}}\int_{(q-1)T_{sw}}^{qT_{sw}} i(t)_1^2\,dt\right)} \tag{3.3}$$

From the rms values of commonly observed waveforms [24], one can write that:

$$\frac{1}{T_{sw}}\int_{(q-1)T_{sw}}^{qT_{sw}} i(t)_1^2\,dt = i_T^2(t) = D(t)I_{Lm,avg}^2(t) \tag{3.4}$$

Under the assumption that the average input power at switching frequency is equal to the instantaneous output power at line frequency [16], then during any switching period:

$$I_{Lm,avg}(t) = \frac{P_{ac}(t)}{V_{in}D(t)} \tag{3.5}$$

It follows that:

$$i_T^2(t) = \frac{P_{ac}{}^2(t)}{V_{in}^2 D(t)} \tag{3.6}$$

The instantaneous output power is:

$$P_{ac}(t) = v_g(t)i_{Lf}(t) = V_{g,pk}I_{Lf,pk}\sin^2\left(\omega_g t\right) \tag{3.7}$$

Assuming that:

$$D(t) = \frac{v_g(t)}{v_g(t) + nV_{in}} \quad ; 0 \le \omega_g t \le \pi \tag{3.8}$$

Therefore,

$$i_T^2(t) = \frac{V_{g,pk}^2 I_{Lf,pk}^2 \sin^4 \omega_g t}{V_{in}^2} + \frac{nV_{g,pk} I_{Lf,pk}^2 \sin^3 \omega_g t}{V_{in}} \tag{3.9}$$

It is assumed that $\omega_g t = \theta$ can be discretized in to a large number of small angles such that:

$$\theta \cong \frac{\pi q}{p} \; ; q = 1, 2, ..., p \tag{3.10}$$

The assumption (3.10) is only as good as the switching frequency is high. The following equation results:

$$\sum_{q=1}^{p} i_T^2(t) = \left(\frac{V_{g,pk} I_{Lf,pk}}{V_{in}} \right)^2 \sum_{q=1}^{p} \sin^4\left(\frac{\pi q}{p}\right) + \left(\frac{nV_{g,pk} I_{Lf,pk}^2}{V_{in}} \right) \sum_{q=1}^{p} \sin^3\left(\frac{\pi q}{p}\right) \tag{3.11}$$

The power of sines can be expressed as:

$$\begin{cases} \sin^4\left(\dfrac{\pi q}{p}\right) = \dfrac{1}{8}\cos\left(\dfrac{4\pi q}{p}\right) - \dfrac{1}{2}\cos\left(\dfrac{2\pi q}{p}\right) + \dfrac{3}{8} \\ \sin^3\left(\dfrac{\pi q}{p}\right) = \dfrac{1}{4}\left(3\sin\left(\dfrac{\pi q}{p}\right) - \sin\left(\dfrac{3\pi q}{p}\right)\right) \end{cases} \tag{3.12}$$

Then, applying Euler's formula to equation (3.12) and solving the sums of the resultant geometric series, it can be shown that:

$$\sum_{q=1}^{p} i_T^2(t) = \left(\frac{V_{g,pk} I_{Lf,pk}}{V_{in}} \right)^2 \left(\frac{3}{8} p \right) + \left(\frac{nV_{g,pk} I_{Lf,pk}^2}{V_{in}} \right) \frac{1}{4} \left(\frac{3\sin\left(\frac{\pi}{p}\right)}{1-\cos\left(\frac{\pi}{p}\right)} - \frac{\sin\left(\frac{3\pi}{p}\right)}{1-\cos\left(\frac{3\pi}{p}\right)} \right) \tag{3.13}$$

Whereupon it can be shown that:

$$I_{1,rms} = \frac{V_{g,pk} I_{Lf,pk}}{2V_{in}} \sqrt{\frac{T_{sw}}{T_{hl}} \left[\left(\frac{3}{2} p \right) + \left(\frac{nV_{in}}{V_{g,pk}} \right) \left(\frac{3\sin\left(\frac{\pi}{p}\right)}{1-\cos\left(\frac{\pi}{p}\right)} - \frac{\sin\left(\frac{3\pi}{p}\right)}{1-\cos\left(\frac{3\pi}{p}\right)} \right) \right]} \tag{3.14}$$

The skin effect increases the effective resistance of the primary and secondary windings of the transformer at the switching frequency. With this consideration, R_{prim} which is the effective dc-resistance of the primary winding, is related to $R_{prim,ac}$, the ac-resistance, by the resistance multiplier, K_{skin}. In other words:

$$R_{prim,ac} = K_{skin} R_{prim} \tag{3.15}$$

Where K_{skin} has been obtained in [24] and defined as in equation (3.16).

$$K_{skin} = \Delta \frac{\sinh(2\Delta) + \sin(2\Delta)}{\cosh(2\Delta) - \cos(2\Delta)} \tag{3.16}$$

Where the quantity Δ is defined as in equation (3.17).

$$\Delta = \frac{\phi_{wire}}{\delta} \tag{3.17}$$

The skin depth δ is expressed as (3.18):

$$\delta = \sqrt{\frac{\rho_{Cu,T^\circ}}{\mu_{Cu} \pi f_{sw}}} \tag{3.18}$$

The same relationship as in (3.15) applies to R_{sec}, and $R_{sec,ac}$ which are the secondary winding dc and ac resistances respectively.

During a switching period, the amplitude of the fundamental current in the primary winding can be expressed as:

$$I_{1,pk,\text{fund}} = \frac{4}{\pi} \left(\frac{I_{Lm,avg}(t)}{2} \right) \sin\left(n\pi D(t)\right) \tag{3.19}$$

Where $I_{Lm,avg}(t)$ has been defined in (3.5) (refer to Fig. 26 and Table V for symbol definitions).

The average primary winding current is defined as:

$$I_{1,avg} = \frac{P_{avg}}{V_{in}} \tag{3.20}$$

If the ac resistance of the primary winding is defined as:

$$R_{prim,ac} = K_{skin} R_{prim} \tag{3.21}$$

Then the primary-side conduction loss can be expressed as in (3.22):

$$P_{loss,prim} = R_{Q1} I_{1,rms}^2 + R_{prim} I_{1,avg}^2 + R_{prim,ac} \sum_{q=1}^{p} \frac{1}{2p} I_{1,pk,\text{fund}}^2 \tag{3.22}$$

3.2.1.2 R_{pv} and C_{pv} ESR losses

The equivalent circuit in Fig. 27 will be used to help obtain expressions for the source rms current and the rms current through C_{pv}.

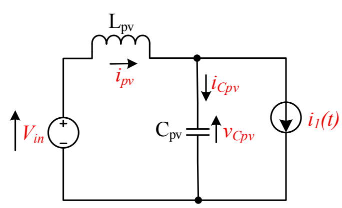

Fig. 27. Equivalent circuit used to compute input rms currents

Assuming that all the switching ripple is absorbed by C_{pv}, the rms input current can be shown to be:

$$i_{pv,rms} = \sqrt{\left(\frac{2I_{1,avg,pk}}{\pi}\right)^2 + \left(\frac{4I_{1,avg,pk}}{3\pi\sqrt{2}\left(1 - L_{pv}C_{pv}\omega_{hl}^2\right)}\right)^2} \tag{3.23}$$

Consequently, the rms of the ripple current through C_{pv} can be written as:

$$i_{Cpv,rms} = \sqrt{i_{1,rms}^2 - i_{pv,rms}^2} \tag{3.24}$$

Finally, the losses associated with these currents can be expressed for R_{pv} and R_{Cpv} as equations (3.25) and (3.26) respectively.

$$P_{Rpv} = R_{pv} i_{pv,rms}^2 \tag{3.25}$$

$$P_{Rcpv} = R_{Cpv} i_{Cpv,rms}^2 \tag{3.26}$$

3.2.1.3 Secondary winding losses

In a similar way used to obtain equation (3.14), the secondary winding rms current can be shown to be:

$$I_{2,rms} = I_{Lf,pk}\sqrt{\frac{T_{sw}}{T_{hl}}\left[\frac{3V_{g,pk}}{4nV_{in}}\left(\frac{\sin\left(\frac{\pi}{p}\right)}{1-\cos\left(\frac{\pi}{p}\right)}\right)-\frac{V_{g,pk}}{4nV_{in}}\left(\frac{\sin\left(\frac{3\pi}{p}\right)}{1-\cos\left(\frac{3\pi}{p}\right)}\right)+\frac{p}{2}\right]} \tag{3.27}$$

During a switching period, the amplitude of the fundamental current in the secondary winding can be expressed as:

$$I_{2,pk,fund} = \frac{4}{\pi}\left(\frac{I_{Lm,avg}(t)}{2n}\right)\sin\left(n\pi\left(1-D(t)\right)\right) \tag{3.28}$$

The average current in the secondary winding can be written as:

$$I_{2,avg} = \frac{2I_{Lf,pk}}{\pi} \tag{3.29}$$

A similar relationship in equation (3.21) holds between the ac and dc resistances of the secondary winding. The secondary-side conduction loss can therefore be written as:

$$P_{loss,\mathrm{sec}} = V_f I_{2,avg} + R_T I_{2,rms}^2 + R_{\mathrm{sec}} I_{2,avg}^2 + R_{\mathrm{sec},ac}\sum_{q=1}^{p}\frac{1}{2p}I_{2,pk,\mathrm{fund}}^2 \tag{3.30}$$

3.2.1.4 C_f ESR loss

The filter capacitor C_f rms current is deduced as:

$$I_{Cf,rms} = \sqrt{\left(I_{\mathrm{sec},rms}^2 - I_{Lf,rms}^2\right)} \tag{3.31}$$

The capacitor ESR loss can therefore be calculated as:

$$P_{Cf,loss} = R_{Cf} I_{Cf,rms}^2 \tag{3.32}$$

3.2.1.5 Current-unfolder loss

The combined current-unfolder conduction loss for all four switches can be easily shown to be:

$$P_{unfolder,loss} = R_{unfolder} I_{Lf,pk}^2 \tag{3.33}$$

3.2.1.6 L_f ESR loss

The filter inductor ESR power loss is directly calculated as:

$$P_{Lf,loss} = R_{Lf} I_{Lf,rms}^2 \tag{3.34}$$

3.2.2 Switching losses

Switching losses are estimated based on the approach in [25] where expressions are developed for the switching loss in a buck converter. As it has been emphasized before, it is assumed that the input voltage is a constant, switching ripple in the magnetizing inductance current is ignored, and the average input power at switching frequency is equal to the instantaneous output power at line frequency. Switching losses are incurred in Q_1. The current-unfolder switching losses are negligible because they switch only at the grid zero-volt crossings. It is assumed that D_{out} is a Schottky diode and therefore has virtually no reverse recovery.

3.2.2.1 Main Switch (Q_1)

The switching loss in Q_1 during each switching period can be expressed as:

$$P_{sw}(q) = \frac{1}{2}\left(V_{in} + \frac{V_{g,pk}}{n}\sin\left(\frac{\pi q}{p}\right)\right) I_{Lm,avg}(q)\left(t_{LH} + t_{HL}\right) f_{sw} \tag{3.35}$$

From equation (3.5), the average magnetizing inductance current can be re-written as:

$$I_{Lm,avg}(q) = \frac{V_{g,pk} I_{Lf,pk}}{V_{in}} \sin^2\left(\frac{\pi q}{p}\right) + n I_{Lf,pk} \sin\left(\frac{\pi q}{p}\right) \tag{3.36}$$

The rise-time can be expressed as:

$$t_{LH} = \frac{Q_{g(sw)}\left(R_{gate\_LH,Q1} + R_{\text{int},Q1}\right)}{V_{driver,LH} - V_{Th,Q1} - \dfrac{I_{Lm,avg}(q)}{G_{m,Q1}}} \tag{3.37}$$

Conversely, the fall-time can be written as:

$$t_{HL}=\frac{Q_{g(sw)}\left(R_{gate\_HL,Q1}+R_{\text{int},Q1}\right)}{V_{Th,Q1}+\frac{I_{Lm,avg}(q)}{G_{m,Q1}}+V_{driver,HL}-v_{diode,HL}} \tag{3.38}$$

Combining equations (3.35) to (3.38), the total Q_1 switching loss averaged during a grid half-period is therefore:

$$P_{sw,Q1}=\frac{1}{p}\sum_{q=1}^{p}P_{sw}(q) \tag{3.39}$$

It is difficult to obtain an analytical expression that further simplifies equation (3.39). Therefore, a numerical solver such as MATLAB must be used in evaluating the expression.

3.2.3 Transformer core loss

In [26], the improved Generalized Steinmetz Equation (iGSE) method is proposed to calculate the transformer core loss for periodic arbitrary waveforms using only the Steinmetz parameters.

According to the Steinmetz Equation (SE) [27], the transformer core loss per unit volume can be estimated as follows:

$$P_{core}=kf_{sw}^{\alpha}\hat{B}^{\beta} \tag{3.40}$$

The parameters *k, α,* and *β* are the Steinmetz parameters and can usually be obtained directly from the transformer datasheet, or through curve-fitting if provided with the transformer core loss plots. The plots which give rise to equation (3.40) are usually developed with sinusoidal flux excitations at different frequencies. Since switching waveforms are generally not sinusoidal, the iGSE, among other techniques, was proposed to overcome this limitation in the computation of core loss.

Following the iGSE, the core loss per unit volume during a switching period can be expressed by the system of equations in (3.41):

$$\begin{cases} P_{core}(t) = \dfrac{k_i\left(\Delta B\right)^{\beta-\alpha}}{T_{sw}} \displaystyle\sum_j \left|\dfrac{V_j}{NA_c}\right|^{\alpha} \left(\Delta t_j\right) \\ k_i = \dfrac{k}{2^{\beta+1}\pi^{\alpha-1}\left(0.2761 + \dfrac{1.7061}{\alpha+1.354}\right)} \\ \Delta B = \dfrac{V_{in}}{NA_c} D(t)T_{sw} \\ V_j = \begin{cases} V_{in} \text{ for } 0 \le t \le D(t)T_{sw} \\ -\dfrac{V_{g,pk}\sin\left(\omega_g t\right)}{n} \text{ for } D(t)T_{sw} \le t \le T_{sw} \end{cases} \\ \Delta t_j = \begin{cases} D(t)T_{sw} \text{ for } 0 \le t \le D(t)T_{sw} \\ (1-D(t))T_{sw} \text{ for } D(t)T_{sw} \le t \le T_{sw} \end{cases} \end{cases} \tag{3.41}$$

Combining equations (3.8) and (3.10), the total core loss during a grid half-cycle can be shown to be:

$$P_{core,avg} = \frac{A}{T_{hl}}\left(B\sum_{q=1}^{p}\left(\frac{\sin\left(\dfrac{\pi q}{p}\right)}{\Gamma}\right)^{1+\beta-\alpha} + C\sum_{q=1}^{p}\frac{\left(\sin\left(\dfrac{\pi q}{p}\right)\right)^{\beta}}{\left(\Gamma\right)^{\beta-\alpha}}\right) \tag{3.42}$$

Where:

$$\begin{cases} A = vol * k_i\left(\dfrac{V_{in}}{NA_c}\right)^{\beta-\alpha} T_{sw}^{1+\beta-\alpha} \; ; B = \left(\dfrac{V_{in}}{NA_c}\right)^{\alpha} - \left(\dfrac{V_{g,pk}}{nNA_c}\right)^{\alpha} \\ C = \left(\dfrac{V_{g,pk}}{nNA_c}\right)^{\alpha} \; ; \Gamma = \sin\left(\dfrac{\pi q}{p}\right) + \dfrac{nV_{in}}{V_{g,pk}} \end{cases} \tag{3.43}$$

Where *vol* is the effective core volume. Equations (3.42) and (3.43) are then evaluated numerically.

3.2.4 Leakage inductance

During each switching period, the transformer leakage inductance will store and release energy. If an energy-recovery process is used, then the power loss due to the leakage inductance energy can be neglected (some power loss will happen in the energy recovery process, but this would typically be small, given a well-designed leakage inductance energy-recovery mechanism). In the case where a dissipative clamping process is used, then the power loss due to the leakage inductance can be expressed as:

$$P_{Llk} = \frac{L_{lk}}{L_m} P_{out} \tag{3.44}$$

3.2.5 Thermal model

In order to improve the theoretical model, an estimate of the junction temperatures of the switching elements can be made using the one-dimensional equivalent thermal model in Fig. 28.

Assuming that the sink temperature and ambient temperature are known, the junction temperature can be estimated as:

$$T_j = T_a + \left(T_s - T_a\right) \frac{R_{j-c} + R_{c-s} + R_{s-a}}{R_{s-a}} \tag{3.45}$$

Where T_j, T_s, T_a are the junction, sink, and ambient temperatures respectively; R_{j-c}, R_{c-s}, R_{s-a} are the junction-to-case, case-to-sink, and sink-to-ambient thermal resistances respectively.

While taking measurements, some amount of time should be given to ensure that a steady state sink temperature is achieved. The device datasheets can then be used to estimate the on-state resistance and forward diode voltage drop.

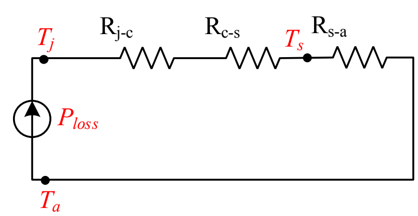

Fig. 28. Thermal model for estimating junction temperature.

3.3 Experimental evaluation

The equations developed in (3.22), (3.25), (3.26), (3.30), (3.32) - (3.34), (3.39), (3.42), and (3.44) are combined to produce an estimate of the total losses of the converter for a 110W prototype which is shown in Fig. 29. The prototype is controlled with a TMS28335 digital signal processor. Table VI shows the circuit parameters and their values. A *IXFK140N30P* Mosfet is used for *Q1* and a *C2D05120* for the flyback diode D_{out}.

Table VI. FMICPseudo-dc parameters for loss modeling

A_c	188 mm^2	L_{lk}	716.8 nH
C_f	1 uF	R_T	0.293
C_{pv}	3*1800 uF	R_{sec}	0.106 Ω
f_g	60 Hz	$R_{unfolder}$	0.24 Ω
f_{sw}	250 kHz	$V_{g,pk}$	170V
$G_{m,Q1}$	90 S	$V_{Th,Q1}$	4 V
L_f	100 uH	V_F	0.8872 V
L_m	28 uH	V_{driver}	12 V
n	6	β	2.3
$Q_{g(sw)}$	96 nC	vol	13.9 cm^3
R_{Cf}	0.2 Ω	f_{hl}	120 Hz
$R_{gate\_LH,Q1}$	0 Ω	V_{driver_HL}	0 V
$R_{int,Q1}$	0.58 Ω	V_{driver_LH}	12 V
R_{Lf}	89.6 mΩ	k	0.015746
R_{prim}	10 mΩ	α	1.44
R_{pv}	18 mΩ	N	6
R_{Q1}	22.8 mΩ	$v_{diode,HL}$	0 V

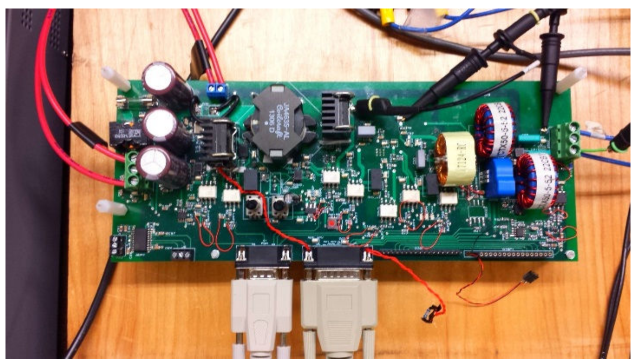

Fig. 29. Experimental setup

For the purposes of comparing the theoretical loss model with the experimental results Fig. 31 shows efficiency plots from the theoretical results (red, dotted) and experimental setup (blue, solid). A close match can be observed in the mid-to-high power range while the match is not so good in the low power region. One likely reason for this is the fact that in lower power regions, the THD of the grid-injected current increases significantly. This defeats the assumption that i_{Lf} is purely sinusoidal – a premise which was used in obtaining the loss equations.

Fig. 31 is a cumulative bar chart showing the loss composition by category. It can be seen that at low power levels the switching and conduction losses are relatively low, but as the output power increases, they constitute the majority of the total losses. On the other hand, the core loss is essentially constant and dominates only in the low power range.

Fig. 32 shows experimental waveforms for V_{in} *(yellow),* i_{pv} *(cyan),* P_{in} *(red),* v_g *(pink),* i_{Lf} *(green).*

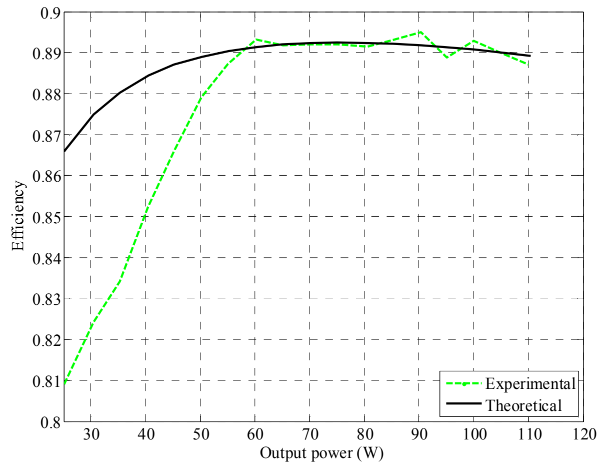

Fig. 30. Plot of efficiency vs output power for theoretical and experimental models

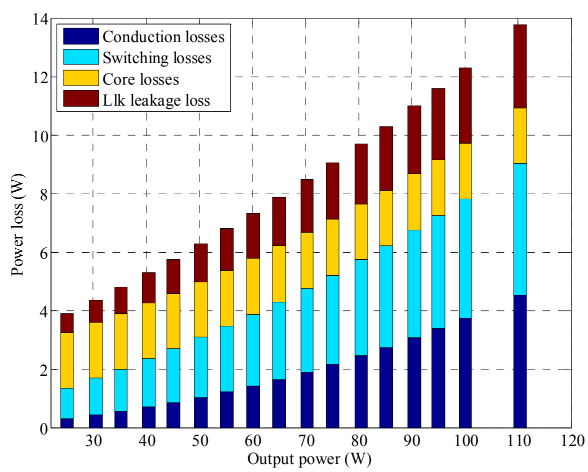

Fig. 31. Bar chart showing distribution of the losses by category.

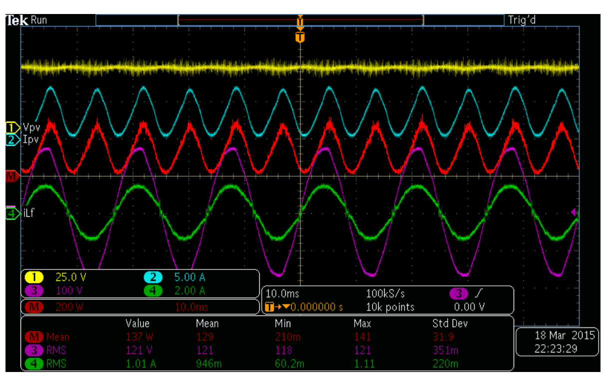

Fig. 32. Experimental waveforms showing V_{in} (yellow), i_{pv} (cyan), P_{in} (red), v_g (pink), i_{Lf} (green).

3.4 Conclusion

This chapter has proposed a theoretical loss model for the powertrain of the FMICpseudo-dc operating in CCM. With a goal of making the model compact, equations are developed that consider conduction losses, switching losses, core losses and leakage inductance losses. The theoretical model is compared with experimental results and a good match is observed in the mid-to-nominal power range. Furthermore, it is seen that the switching and conduction losses dominate at the high output power levels. A loss model taking in to account the harmonics which are dominant at low power levels could improve the efficiency prediction.

Chapter 4

An Analysis of Displacement Power Factor in the Flyback AC Module with Current-Unfolding in DCM

4.1 Introduction

The flyback micro-inverter with a pseudo-dc link (FMICpseudo-dc) can operate in discontinuous conduction mode (DCM) [18]. It has been observed in [22] that while in this mode, the converter's displacement power factor varies as the output capacitance is changed. This chapter is an incremental contribution to that work. It develops a power factor equation based on an equivalent circuit model. The predicted displacement power factor is then compared with experimental results for variations in the CL filter parameters. It is shown that the power factor depends mostly on the filter capacitor C_f.

4.2 Design for open loop operation (DCM)

For the discussion, a simplified schema of the FMICpseudo-dc is provided in Fig. 33.

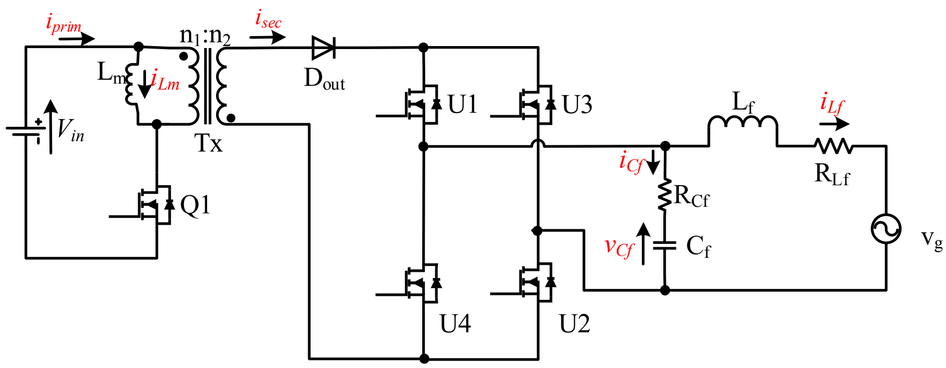

Fig. 33. Schema of FMICpseudo-dc for DCM analysis

In [18], it was shown how the transformer primary winding current can be shaped such that it is bounded in a full wave rectified (or 'folded') sinusoidal envelope. This is done in an open loop manner, and the converter is modeled in DCM. A summary of the conditions that must be satisfied for DCM are provided here below.

Let the duty cycle vary in a sinusoidal manner as expressed in equation (4.1).

$$d(t) = d_p \left|\sin(\omega_g t)\right| \tag{4.1}$$

Where *d(t)* is the instantaneous duty ratio; d_p is the peak duty ratio; and ω_g is the grid angular velocity.

In the primary winding, the peak current during a switching cycle, $i_{prim,pk}$ can be expressed as:

$$i_{prim,pk} = \frac{V_{in}}{L_m f_{sw}} d(t) \tag{4.2}$$

Where f_{sw} is the switching frequency.

It is shown that the inequality condition in (4.3) must be respected in order for the converter to transfer power while remaining in DCM.

$$d_p \leq \frac{1}{1+\lambda n} \tag{4.3}$$

Where λ is defined as:

$$\lambda = \frac{V_{in}}{V_{g,pk}} \tag{4.4}$$

Where $V_{g,pk}$ is the peak grid voltage.

Neglecting converter losses, the average power injected in to the grid in DCM can be expressed as:

$$P = \frac{\lambda^2 d_p^2 V_{g,pk}^2}{2 f_{sw} L_m} \tag{4.5}$$

4.3 Equivalent circuit model and power factor prediction

In order to compute the power factor, the equivalent circuit of Fig. 34 is used. The flyback is considered a controlled current source supplying a CL-filter. The unfolder dynamics are ignored.

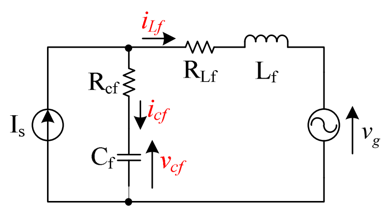

Fig. 34. Equivalent circuit for power factor discussion

Assuming the expression of the current source can be written as $I_s(t) = I_{s,pk} \sin(\omega_g t)$, phasor notation can then be used henceforth. The grid-injected current can be written as:

$$\underline{i}_{Lf} = \underline{I}_s - \underline{i}_{Cf} \tag{4.6}$$

$\underline{i}_{Cf}$ can be obtained as:

$$\underline{i}_{Cf} = \frac{\underline{v}_g + \left(jX_{Lf} + R_{Lf}\right)\underline{i}_{Lf}}{R_{Cf} - jX_{Cf}} \tag{4.7}$$

Whereupon the output current after some computation can be written as the system of equations in (4.8):

$$\begin{cases} \underline{i}_{Lf} = I_{Lf,rms}\angle\varphi \\ I_{Lf,rms} = \dfrac{\sqrt{A^2+B^2}}{\left(R_{Cf}+R_{Lf}\right)^2+\left(X_{Lf}-X_{Cf}\right)^2} \\ A = \left(R_{Cf}+R_{Lf}\right)\left(R_{Cf}I_{s,rms}-V_{g,rms}\right)-X_{Cf}I_{s,rms}\left(X_{Lf}-X_{Cf}\right) \\ B = X_{Cf}R_{Lf}I_{s,rms}+X_{Lf}\left(I_{s,rms}R_{Cf}-V_{g,rms}\right)+X_{Cf}V_{g,rms} \\ \varphi = -\arctan\left(\dfrac{B}{A}\right) \end{cases} \quad (4.8)$$

Equation (4.8) specifies the magnitude and phase shift of the output current when the magnetizing inductance current is in phase with the 'folded' grid voltage. [22] shows the effect of varying the capacitor filter C_f parameters on the power factor, thus showing the importance of careful filter design in open loop operation. It can be shown in a similar process used to obtain equation (4.8) that merely varying only the phase of I_s will not yield unity power factor.

4.4 Comparison with experimental results

4.4.1 Changing filter capacitance, C_f

Table VII shows the circuit parameters used for evaluating the displacement power factor as the converter operates in DCM.

Table VII. FMICPseudo-dc parameters for power factor evaluation in DCM

C_f	1 uF to 5 uF
f_g	60 Hz
f_{sw}	25 kHz
$I_{s,pk}$	1.43 A
L_f	2 mH
L_m	29 uH
n	2
R_{Cf}	0.2 Ω
R_{Lf}	0.274 mΩ
$V_{g,pk}$	171 V

The filter capacitor C_f is varied from 1uF to 5uF while the filter inductor L_f is kept constant at 2mH. Fig. 35 shows the experimental results for a FMICpseudo-dc prototype operating in DCM with C_f =1.2uF. Green is the output current, i_{Lf} while cyan represents the grid voltage v_g.

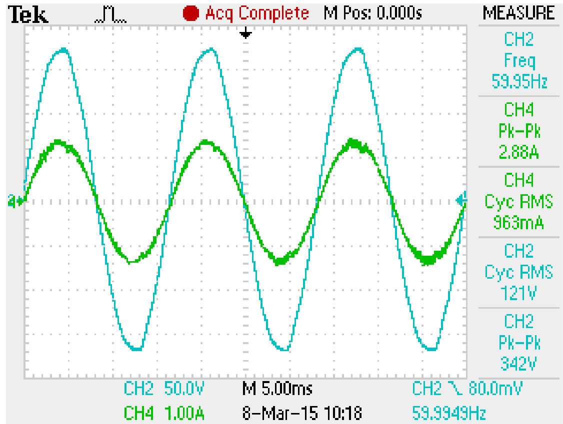

Fig. 35. Experimental waveforms for FMICpseudo-dc operating in DCM

The power factor is then computed for each operating condition of C_f and L_f and compared with the result obtained from equation (4.8). The result of this comparison is shown in Fig. 36 where a close match is seen.

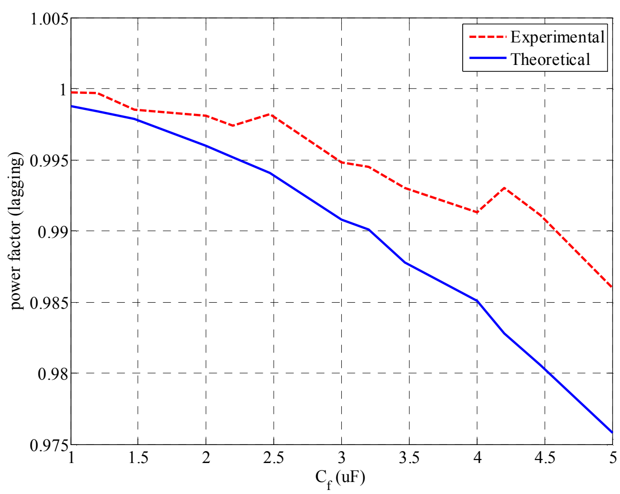

Fig. 36. Comparison of theoretical and measured power factor for varying C_f

The root mean square error (rmse) between the experimental and theoretical results is defined as in equation (4.9), where u is the number of experimental samples. The rmse is computed to be 0.59%. Therefore, the proposed circuit model is a good predictor of the displacement power factor for the case of varying C_f.

$$rmse = \sqrt{\frac{\sum_{i=1}^{u}\left(pf_{i,\text{experimental}} - pf_{i,\text{theoretical}}\right)^2}{u}} \tag{4.9}$$

4.4.2 Changing filter inductance, L_f

In this case, C_f is kept constant at 1uF and L_f changed from 1mH to 3mH. The plot of lagging power factor as a function of L_f is shown in Fig. 37. An rmse of 0.07% is observed between the predicted and experimental values. It is interesting to note that changing L_f does not have as much of an effect on the power factor as changing C_f. However, increasing L_f could lead to more converter losses due to higher inductor resistance.

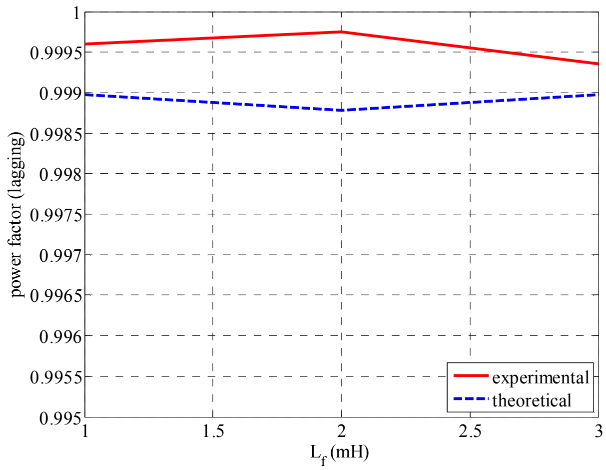

Fig. 37. Measured power factor as a function of changing filter inductor L_f

4.5 Conclusion

This chapter developed a way to predict the power factor of the FMICpseudo-dc operating in discontinuous conduction mode (DCM). It is shown that the power factor is more sensitive to changes in the filter capacitor C_f than changes in the filter inductor L_f . The theoretical model and experimental observations show a close match.

Chapter 5

Conclusion and Future Work

One of the objectives of this thesis was to show how the flyback micro-inverter with a pseudo-dc link (FMICpseudo-dc) can be controlled to inject desired amounts of reactive power in to the distribution network. In order to lay the ground work for accomplishing this, the distortion in the grid-injected current i_{Lf} was examined in chapter 2. It was shown how and why i_{Lf} oscillates for brief periods around the grid zero voltage crossing which causes the distortion. The THD increases slightly when a synchronous rectifier is included across the flyback diode and becomes unacceptable as the phase of the output current with respect to the grid voltage is increased (or decreased) from 0. To overcome the zero-crossing distortion, a current decoupling circuit was proposed and simulated and shows good results. However, it introduces additional components and increased complexity when compared to a traditional FMICpseudo-dc.

Chapter 3 proposed a powertrain loss model for the FMICpseudo-dc operating in continuous conduction mode (CCM). The losses modeled included conduction, switching, and core losses. The experimental efficiency shows a good match with the predicted efficiency in the mid-to-high power regions but not in the low power regions. Future work should address the discrepancy observed at low power.

In chapter 4, equations were developed to predict the open loop power factor of the FMICpseudo-dc. The experimental results of a prototype operating in discontinuous conduction mode (DCM) show a close match with the theoretical predictions. From the results, it appears that the power factor is more sensitive to changes in the filter capacitor C_f than the filter inductor L_f.

In order to further cement the study of the possibility of using the FMICpseudo-dc as a controlled source of reactive power, it will be indispensable to build a prototype that includes the current decoupling circuit. Then, it can be possible to do a full comparison of the system with other micro-inverters or systems that can function as controlled reactive power sources. The comparison should include efficiency, as well as a cost analysis dimension.

Furthermore, it is important to build test photovoltaic power systems that use micro-inverters and centralized inverters and compare the energy outputs of both systems. This will contribute to supporting or not supporting the claim that systems based on micro-inverters can have energy gains compared to those based on centralized inverters.

Appendix A

This appendix shows the MATLAB script used in computing the current waveforms of Fig. 6, Fig. 7, and Fig. 11which show the distortion in the output current around the grid zero volt crossing.

```
%In this script, we develop a numerical solution for explaining the
%zero-crossing problem in the output current of the flyback inverter using
%the results obtained from the theoretical analysis,

clear all;
clc;
close all;

linewidth = 3;
font = 'Times New Roman';
fontsize = 16;

%PART 1
%======
%In this part we develop the equivalent Fourier Series representation for
%the full-wave rectified current.

ILfmax = 0.7; %1.178; %0.7;                    %Peak output current
n = 1:1:20;                        %Harmonic orders (note that these are harmonics of 2*f. In US, this means 120Hz, 240Hz, 360Hz etc.)
f = 60;                           %grid frequency. Note that fundamental frequency is twice this value
w = 2*pi*f;                        %angular grid frequency
wt = [0:0.0001:2*pi];                  %Angle for generating plot (in rad)

ILf_dc = 2*ILfmax/pi;               %Average or dc component of full-wave rectified current
ILf_ac = zeros(length(n),length(wt));   %Matrix for the ac component
ILf_n_pk = (4*ILfmax/pi)*(1./(1 - (2*n).^2));   %Vector of peak value of each harmonic component

for k = 1:length(n)
    ILf_ac(k,:) = ILf_n_pk(k)*cos(2*n(k)*wt);   %Here we generate the AC component at each point of wt
end

iLf = ILf_dc + sum(ILf_ac,1);          %Equivalent Fourier representation

% figure(1);
% plot(wt,iLf);                    %Plot
% grid on;

%PART 2
%======
%In this part, we calculate the equivalent current-source and show that it
%must be distorted and will not be a full-wave rectified sinusoid as
%previously thought.

Lf = 170e-6;%115e-6;    %Filter inductor
RLf = 24.8e-3*2;% + 0.16*2; %10e-3 + 0.16*2;      %Inductor ESR + Unfolder Rdson
Cf = 1e-6;     %Filter capacitor
Rcf = 0.2 + 1;      %Capacitor ESR
Vgmax = 171; %342;    %peak grid voltage

XLf = Lf*(2*n)*w;                          %Impedance of Lf at the different frequencies
Xcf = 1./(Cf*(2*n)*w);                      %Impedance of Cf at the different frequencies
```

```
iLf_n = (ILf_n_pk)/sqrt(2);                          %Complex values of current harmonics
vg_n = (4*Vgmax/(sqrt(2)*pi))*(1./(1 - (2*n).^2));          %Complex values of grid voltage harmonics
Vg_dc = 2*Vgmax/pi;                                  %DC value of grid voltage

%DC response (Inductor Lf appears as a short circuit while capacitor Cf
%appears as an open circuit)

I_source_dc = ILf_dc;
Vcx_dc = RLf*ILf_dc + Vg_dc;                         %Vcx is the voltage across Cf and Rcf

%AC response
for k = 1:length(n)                        %Calculate the phasors for inverter voltage and current source
    vcx_n = vg_n + (RLf + 1i*XLf).*iLf_n;
    i_source_n = iLf_n + vcx_n./(Rcf - 1i*Xcf);
end

vcx_n_mag = abs(vcx_n);                  %Magnitude (rms) values of inverter voltage
vcx_n_angle = angle(vcx_n);               %Angles (rad) of inverter voltage
i_source_n_mag = abs(i_source_n);        %Idem for current source
i_source_n_angle = angle(i_source_n);

%Time-domain plots

%This section generates the equivalent Fourier series for i_source and vcx
%and plots them as functions of time

i_source_n = zeros(length(n),length(wt));
vcx_n = zeros(length(n),length(wt));

for k = 1:length(n)
    i_source_n(k,:) = i_source_n_mag(k)*sqrt(2)*cos(2*n(k)*wt + i_source_n_angle(k));
    vcx_n(k,:) = vcx_n_mag(k)*sqrt(2)*cos(2*n(k)*wt + vcx_n_angle(k));
end

i_source = I_source_dc + sum(i_source_n,1);
vcx = Vcx_dc + sum(vcx_n,1);
vg = Vgmax*abs(sin(wt));
Pgrid = iLf.*vg;
Psource = i_source.*vcx;

figure(2);
plot(wt,i_source,'b',wt,iLf,'--r','linewidth',linewidth);
ylabel('Current(A)','fontname','times','fontsize',fontsize);
xlabel('wt (rad)','fontname','times','fontsize',fontsize);
legend('I_{s}','i_{Lf(t)}','Location','NORTHEAST');
legend boxoff;
grid on;

figure(3);
plot(wt,Psource,'b',wt,Pgrid,'--r','linewidth',linewidth);
ylabel('Power (W)','fontname',font,'fontsize',fontsize);
xlabel('wt (rad)','fontname',font,'fontsize',fontsize);
legend('I_{s}','i_[6]','Location','NORTHEAST');
legend boxoff;
grid on;

figure(4);
plot(wt,vcx);
ylabel('Pseudo-dc link voltage (V)','fontname','times','fontsize',14);
xlabel('wt (rad)','fontname','times','fontsize',14);


%PART 3
%=======
%In this part, we calculate the output current if the source current is
%limited to positive values (such as in the case where only an output diode
%is used in the flyback. The goal is to observe the distortion in iLf when
%Is is less than 0. The shape of iLf is assumed unchanged when Is becomes
%greater than 0 again.
```

```
%Search for and obtain time at which source current is = 0 for the first
%time
counter1 = 1;
while (i_source(counter1) > 0)
    counter1 = counter1+1;
end
zero_position_1 = counter1;

counter2 = counter1 + 1;
while (i_source(counter2) < 0)
    counter2 = counter2+1;
end
end_zero_position_1 = counter2-1; %minus 1 because we do not want to include the moment when i_source becomes positive again.

t0 = wt(zero_position_1)/w;
wt0 = wt(zero_position_1);
Vcx0 = vcx(zero_position_1);
ILf0 = 0.45;% iLf(zero_position_1);
Vg0 = abs(Vgmax*sin(wt(zero_position_1)));

wt_end = wt(end_zero_position_1);
t_end = wt_end/w;
ILf_end = iLf(end_zero_position_1);

wk = 1/sqrt(Lf*Cf);
tau = 2*Lf/(RLf + Rcf);
alpha = (1/2)*sqrt(4*(wk^2) - ((RLf + Rcf)/Lf)^2);
if (alpha < 0)
    error('alpha is less than 0');
end

E = -Vgmax*w^2*(2/tau)/(Lf*(wk^2 - w^2)^2);
if (E > 0)
    error('E is greater than 0');
end

F = -Vgmax*w/(Lf*(wk^2 - w^2));
if (F > 0)
    error('F is greater than 0');
end

gamma = exp(t0/tau)*(ILf0 - E*sin(wt0) - F*cos(wt0))/sin(alpha*t0);

k1 = -exp(-t0/tau)*sin(alpha*t0)/tau + alpha*exp(-t0/tau)*cos(alpha*t0);
k2 = -exp(-t0/tau)*cos(alpha*t0)/tau - alpha*exp(-t0/tau)*sin(alpha*t0);
k3 = E*w*cos(wt0) - F*w*sin(wt0);

B = (1/(k2 - k1*cot(alpha*t0)))*(-k1*gamma - k3 + (Vcx0 - (RLf + Rcf)*ILf0 - Vg0)/Lf);
A = gamma - B*cot(alpha*t0);

wt_distorted = [wt0:0.0001:wt_end];
t_distorted = wt_distorted/w;
iLf_distorted = exp(-t_distorted/tau).*(A*sin(alpha*t_distorted) + B*cos(alpha*t_distorted)) + E*sin(wt_distorted) + F*cos(wt_distorted);

figure(5);
plot(wt_distorted,iLf_distorted,'b',wt_distorted,Vgmax*sin(wt_distorted)/50,'r','linewidth',linewidth);
ylabel('i_[6] (A)','fontname',font,'fontsize',fontsize);
xlabel('wt (rad)','fontname',font,'fontsize',fontsize);
grid on;


%Reconstitute new plot showing the distorted output current waveform
iLf_combined = iLf;
i_source_clipped = i_source;

k = 1;
while (i_source(k) > 0)
    i_source_clipped(k) = i_source(k);
    iLf_combined(k) = iLf(k);
    k = k+1;
```

```
end

m = 1;
while (i_source(k)<=0)
   i_source_clipped(k) = 0;
   iLf_combined(k) = iLf_distorted(m);
   k = k+1;
   m = m+1;
end

while (i_source(k) > 0)
   i_source_clipped(k) = i_source(k);
   iLf_combined(k) = iLf(k);
   k = k+1;
end

m = 1;
while (i_source(k)<=0)
   i_source_clipped(k) = 0;
   iLf_combined(k) = iLf_distorted(m);
   k = k+1;
   m = m+1;
   if (k>length(i_source))
      break; %Break the while loop
   end
end

figure(6);
plot(wt,i_source_clipped,'--b',wt,iLf_combined,'r','linewidth',linewidth);
ylabel('Current(A)','fontname',font,'fontsize',fontsize);
xlabel('wt (rad)','fontname','times','fontsize',fontsize);
legend('I_{s,clipped}','i_{Lf,new}','Location','NORTHEAST');
legend boxoff;
grid on;



%Here we re-calculate the same paramaters as above, but for this is for the
%original equation we used. We are trying to compare both. There is a small
%difference in the
alpha = 0.5*sqrt(4/(Cf*Lf) - ((RLf + Rcf)/Lf)^2);
beta = 1/(Cf*Lf*w) - w;
E2 = -(Vgmax/Lf)/((tau/2)*beta^2 + 2/tau);
F2 = tau*beta*E2/2;

gamma = exp(t0/tau)*(ILf0 - E2*sin(wt0) - F2*cos(wt0))/cos(alpha*t0);
k1 = -exp(-t0/tau)*cos(alpha*t0)/tau - alpha*exp(-t0/tau)*sin(alpha*t0);
k2 = -exp(-t0/tau)*sin(alpha*t0)/tau + alpha*exp(-t0/tau)*cos(alpha*t0);
k3 = E2*w*cos(wt0) - F2*w*sin(wt0);

B2 = (Vcx0 - ((RLf + Rcf)*ILf0 + Vg0 + Lf*(gamma*k1 + k3)))/(Lf*(-tan(alpha*t0)*k1 + k2));
A2 = gamma - B2*tan(alpha*t0);

iLf_distorted2 = exp(-t_distorted/tau).*(A2*cos(alpha*t_distorted) + B2*sin(alpha*t_distorted)) + E2*sin(wt_distorted) + F2*cos(wt_distorted);

load time_iLf;
load iLf_scope;
load time_vg;
load vg_scope;

figure(7);
plot(wt_distorted/w,iLf_distorted2,'r',time+(8.354e-3),iLf_scope,'--b','linewidth',linewidth);
ylabel('i_[6] (A)','fontname',font,'fontsize',fontsize);
xlabel('wt (rad)','fontname',font,'fontsize',fontsize);
grid on;
```

Appendix B

This appendix contains the salient C++ code that implements the digital control of the flyback micro-inverter with a pseudo-dc link on a F28335 Texas Instruments DSP.

- ***Main file (main.c)***

```
//**NB:  Before running this code, check polarity of unfolder and sign of the measured current in the code
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include "IQmathLib.h"
#include "defines.h"
#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"   // DSP2833x Examples Include File
#include "DSP2833x_SWPrioritizedIsrLevels.h"
#include "SFO_V5.h"              // SFO V5 library headerfile - required to use SFO library functions


//Functions used in this program
extern void init_epwm1(void);
extern void init_epwm2(void);
extern void init_epwm5(void);
extern void init_epwm4(void);
extern void init_gpio(void);
extern void init_adc(void);
extern void init_tmr0(void);
extern void init_variables(void);

//Interrupt sub-routines used in this program
extern interrupt void xint1_isr(void);
extern interrupt void cpu_timer0_isr(void);
extern interrupt void epwm4_isr(void);
extern interrupt void adc_isr(void);

//All variables used in this program.
typedef struct ADC_structure{
        Uint16 mean0;
        Uint16 mean1;
        Uint16 mean2;
        Uint16 mean3;
        Uint16 mean4;
        Uint16 mean5;
        Uint16 mean6;
        Uint16 mean7;
        Uint16 mean8;
        Uint16 mean9;
        Uint16 buffer0[2], buffer1[2], buffer2[2], buffer3[2], buffer4[2],
        buffer5[2], buffer6[2], buffer7[2], buffer8[2], buffer9[2]; //Buffers for ADC, will be used to compute mean
        Uint16 timer;
} ADC_structure;

typedef struct current_controller_structure{
        _iq16 error_IQ16[2]; //Controller error and one history term
        _iq16 duty_KI_IQ16[2]; //Duty cycle from integral contribution and one history term
```

```
		_iq16 duty_KP_IQ16; //Duty cycle from proportional contribution
		_iq16 duty_lag_IQ16[2]; //Duty cycle from the lag term and one history term
		_iq16 duty_PI_IQ16; //Duty cycle from the PI controller (sum of Kp and Ki duty cycle terms)
		_iq16 duty_decoupled_IQ16; //Decoupled duty cycle
		_iq16 duty_total_IQ16[2]; //Total duty cycle from the sum of de-coupled and controller terms with one history term
		_iq16 lag_a_IQ16; //Lag coefficients
		_iq16 lag_b_IQ16;
		_iq16 KP_IQ16;		//Controller proportional gain
		_iq16 KI_z_IQ16;
		_iq16 I_grid_reference_folded_IQ16; //Controller reference current in "folded" form
		_iq16 I_grid_folded_IQ16; //Measured grid current in "folded" form
		int duty_DSP; //Duty cycle in DSP units to be fed to PWM Counters.
		long duty_frac; //Fractional duty cycle.
		char V_GRID_POLARITY; //This is used to determine the sign of the "folded" grid reference and measured currents
		char DEAD_TIME;	//The current controller's own dead time shadow variable
		char CURRENT_CONTROLLER_ACTIVATED;
} current_controller_structure;

typedef struct error_structure{
		char I_GRID_OVR_CURRENT;
		char V_INV_OVR_VOLTAGE;

		char V_GRID_OVR_VOLTAGE;
		char V_GRID_OVR_VOLTAGE_SHDW;
		char V_GRID_UNDR_VOLTAGE;
		char V_GRID_UNDR_VOLTAGE_SHDW;

		char V_GRID_WRONG_POLARITY;

		char V_PV_OVR_VOLTAGE;
		char V_PV_OVR_VOLTAGE_SHDW;
		char V_PV_UNDR_VOLTAGE;
		char V_PV_UNDR_VOLTAGE_SHDW;

		char I_PV_OVR_CURRENT;
		char TEMP1_HIGH;
		char TEMP2_HIGH;
		int v_grid_ovr_voltage_timer, v_grid_undr_voltage_timer;
		int v_pv_ovr_voltage_timer, v_pv_undr_voltage_timer;

		char FATAL_ERROR;
		char STOP;
} error_structure;

typedef struct PLL_structure{
		_iq20 Kp; //PLL proportional
		_iq20 Ki; //PLL integral term (Ki*Ts/2)
		_iq20 Ki2; //(VCO) Integrator coefficient for computing the grid angle (Ts/2) in iq20 format
		_iq20 cos_theta; //cos_theta
		_iq20 sin_theta; //sin_theta
		_iq20 w[3]; //This is the angular velocity from the PLL alongside two history term
		_iq20 w_filtered[3]; //This is the filtered angular velocity along with two history terms
		_iq20 w_non_offset_P; //This is the proportional component of the unfiltered output of the PLL's PI controller.
		_iq20 w_non_offset_I[2]; //This is the integral component of the unfiltered output of the PLL's PI controller along with one history
term
		_iq20 w_offset; //This is the offset w that will be added (feedforward) to improve response of the PLL and reduce controller effort
		_iq20 theta[2]; //Angle in radians at output of VCO, with one history term
		_iq20 SOGI_Valpha[3]; //This is the alpha voltage from the second order generalized integrator
		_iq20 SOGI_Vbeta[3]; //This is the beta voltage from the second order generalized integrator
		_iq20 SOGI_k; //SOGI k coefficient
		_iq20 SOGI_wnTs;
		_iq20 SOGI_x;
		_iq20 SOGI_y;
		_iq20 SOGI_denominator;
		_iq20 SOGI_quotient;
		_iq20 SOGI_b0;
		_iq20 SOGI_a1;
		_iq20 SOGI_a2;
		_iq20 SOGI_ky;
		_iq20 SOGI_qb0;
```

```
	_iq20 V_grid_pu[3]; //Normalized (per-unitized) instantaneous grid voltage with two history terms;
	_iq20 V_grid_base;	//Base grid voltage
	_iq20 error[3];//Present error term and two history terms
	_iq20 notch_a; //Notch filter coefficient a(0.979127)
	_iq20 notch_b; //Notch filter coefficient b(-1.957364)
	_iq20 notch_f; //Notch filter coefficient f(0.958254)
	_iq20 notch_out[3]; //Output of notch filter with two history terms.
	_iq20 Vd; //Direct-axis voltage
	_iq20 Vq; //Quadrature-axis voltage
	_iq20 V_grid_pk_pu[2]; //Normalized (per-unitized) unfiltered peak grid voltage
	_iq20 V_grid_pk_filtered_pu[2]; //Normalized filtered peak grid voltage
	_iq20 A,B,C,D,E; //Variables that will be used as coefficients for filtering the angular speed and the peak normalized grid voltage.
Can also be constants in IQ20 format
	char PERIOD_HAS_OCCURED; //Takes note of when one grid period has occurred
	char HAS_STARTED;
	char V_GRID_POLARITY; //Grid polarity as given by PLL module
	char DEAD_TIME; //Dead time as given  by PLL module

/*
	char LOCKED; //This variable is set when it is determined that the PLL is 'locked' (when frequency varies within a narrowband)
	char FAILED; //This variable is set when it is determined that the PLL has failed to lock after several attempts.
	Uint16 lock_timer; //Timer used to keep track of PLL Locked status
*/
}PLL_structure;

typedef struct zcd_structure{
	Uint16 half_period[2]; //Grid half period
	Uint16 full_period; //Grid full period
	Uint16 half_period_counter; //Counter for determining the half_period
	Uint16 full_period_counter; //Counter for determining grid angle from 0 to 360 degrees
	Uint16 angle_pu_IQ16; //Grid angle measurement by using the hardware zcd method.
	char V_GRID_POLARITY; //Grid polarity as given by zcd circuit.
	char PERIOD_TOO_LOW; //High frequency limit as given by zcd module
	char PERIOD_TOO_HIGH; //Low frequency limit as given by zcd module
	char DEAD_TIME; //Dead time event indicator
} zcd_structure;

typedef struct power_control_structure{
	_iq16 P_ref_IQ16;	//Reference active power in IQ16 format in IQ16 format.
	_iq16 Q_ref_IQ16;	//Reference reactive power in IQ16 format in IQ16 format.
	_iq16 error_P_ref[2]; //Active power tracking error
	_iq16 error_Q_ref[2]; //Reactive power tracking error
	_iq16 P_dc_IQ16;	//Active power computed by P-Q transform method in IQ16 format.
	_iq16 Q_dc_IQ16;	//Reactive power computed by P-Q transform method in IQ16 format.
	_iq16 P_output_avg_IQ16; //Average output power. Computed by integration. IQ16 format
	_iq16 P_input_avg_IQ16; //Average input power. Computed by integration. IQ16 format
	_iq16 I_grid_reference_peak_IQ16; //Peak reference grid current from power loop. In IQ16 format.
	_iq16 Kp;  //Proportional gain to be used for power control loop, in IQ16 format
	_iq16 Kiz; //(Integrator gain)*(Ts/2) integrator gain for power control loop in IQ16 format
	_iq16 inst_input_power_integrator; //Sums the instantaneous input power at regular sample intervals.
	_iq16 inst_output_power_integrator; //Sums the instantaneous output power at regular sample intervals.
	Uint16 timer; //Control loop timer (approximates the sampling period).
	Uint16 avg_power_counter; //Counter to be used in computing the average power by integration method
} power_control_structure;


extern _iq16 I_pv_IQ16; //PV current
extern _iq16 V_pv_IQ16; //PV voltage
extern _iq16 V_grid_IQ16; //Grid voltage
extern _iq16 V_inv_IQ16; //Inverter pseudo-dc link voltage
extern _iq16 I_grid_IQ16; //Grid-injected current
extern _iq16 I_grid_reference_IQ16; //amperes
extern _iq16 I_grid_angle_offset_IQ16; //amperes
extern _iq16 dummy1_IQ16, dummy2_IQ16, dummy3_IQ16; //dummy variables for computations
//extern char V_grid_polarity; //Grid voltage polarity
extern struct current_controller_structure cc; //Create an instance of a current controller
extern struct error_structure ERROR_STATUS; //Create an instance of an error structure
extern struct ADC_structure ADC; //Create an instance of an ADC structure
extern struct PLL_structure PLL; //Create an instance of a PLL structure
extern struct zcd_structure zcd; //Create an instance of a zcd structure
```

```
extern struct power_control_structure power_loop; //Create an instance of a power control structure
//extern int debug1[200];
//extern int debug2[200];
extern int dummy_counter;
extern Uint16 start_timer;
extern char CONVERTER_STARTED;
extern char SOURCE_OF_UNFOLDING_SIGNAL;

extern int MEP_ScaleFactor[7]; // Global array used by the SFO library. Only HRPWM1A will be used to compute scale factor. So it's HRPWM
must be disabled
extern volatile struct EPWM_REGS *ePWM[7]; //
extern int SFO_status;

void main(void)
{

  // Step 1. Initialize System Control:
  // PLL, WatchDog, enable Peripheral Clocks
  // This example function is found in the DSP2833x_SysCtrl.c file.
  InitSysCtrl();

  // Step 2. Initialize GPIO:
  // This example function is found in the DSP2833x_Gpio.c file and
  // illustrates how to set the GPIO to it's default state.
  //InitEPwm1Gpio();
  //InitEPwm2Gpio();

  // Specific clock setting for this example:
  EALLOW;
  SysCtrlRegs.HISPCP.all = ADC_MODCLK;          // HSPCLK = SYSCLKOUT/ADC_MODCLK
  EDIS;

  // Step 3. Clear all interrupts and initialize PIE vector table:
  // Disable CPU interrupts
  DINT;


  // Initialize the PIE control registers to their default state.
  // The default state is all PIE interrupts disabled and flags
  // are cleared.
  // This function is found in the DSP2833x_PieCtrl.c file.
  InitPieCtrl();

  // Disable CPU interrupts and clear all CPU interrupt flags:
  IER = 0x0000;
  IFR = 0x0000;

  // Initialize the PIE vector table with pointers to the shell Interrupt
  // Service Routines (ISR).
  // This will populate the entire table, even if the interrupt
  // is not used in this example.  This is useful for debug purposes.
  // The shell ISR routines are found in DSP2833x_DefaultIsr.c.
  // This function is found in DSP2833x_PieVect.c.
  InitPieVectTable();

  // Interrupts that are used in this example are re-mapped to
  // ISR functions found within this file.
  EALLOW;          // This is needed to write to EALLOW protected registers
  PieVectTable.XINT1 = &xint1_isr;
  PieVectTable.TINT0 = &cpu_timer0_isr;
  PieVectTable.EPWM4_INT = &epwm4_isr;
  PieVectTable.SEQ1INT = &adc_isr;
  //PieVectTable.ADCINT = &adc_isr;
  EDIS;   // This is needed to disable write to EALLOW protected registers

  EALLOW;
  SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0; //Turns off EPWM modules
  EDIS;

  init_variables();
```

```
init_epwm1();
init_epwm2();
init_epwm5();
init_epwm4();
init_adc(); //Initialize all ADC modules
init_gpio(); //Initialize certain input/output pins
init_tmr0();


GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Clear output latch. Turns off Q1/Q2
GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Clear output latch. Turns off Q3/Q4
//GpioDataRegs.GPASET.bit.GPIO4 = 1;   // Set outptut latch. Turns on Q1/Q2

EALLOW;
EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
EDIS;

EALLOW;
SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1; //Turns on EPWM
EDIS;
CpuTimer0Regs.TCR.all = 0x4001; // Start TMR0. Use write-only instruction to set TSS bit = 0

//EPwm1Regs.CMPA.half.CMPA = 300;   // Update compare A value (PWM Q0 and PWM Qx)
//EPwm2Regs.CMPA.half.CMPA = 300;   // Update compare A value (PWM Q0 and PWM Qx)


XIntruptRegs.XINT1CR.bit.ENABLE = 1;       // Enable Xint1. Uncomment when ready to turn on hardware zero-crossing detection

PieCtrlRegs.PIECTRL.bit.ENPIE = 1;   // Enable the PIE block
PieCtrlRegs.PIEIER1.bit.INTx4 = 1;        // Enable PIE XINT1
//PieCtrlRegs.PIEIER1.bit.INTx6 = 1;      //Enable ADCINT in PIE
PieCtrlRegs.PIEIER1.bit.INTx1 = 1;        //Enable SEQ1 interrupt in PIE
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;        // Enable TINT0 in the PIE: Group 1 interrupt 7
PieCtrlRegs.PIEIER3.bit.INTx4 = 1;        // Enable EPWM INTn in the PIE: Group 3 interrupt 4
PieCtrlRegs.PIEACK.all = 0xFFFF;   // Acknowledge then enable PIE interrupts
//  PieCtrlRegs.PIEACK.all = (M_INT1|M_INT3);   // Make sure PIEACK for group 1 is clear (default after reset)
IER |= M_INT1; // Enable CPU INT1 which is connected to external interrupt 1 (i.e. XINT1), to  CPU-Timer 0 (TMR0), and to the  ADC
interrupt (ADCINT)
IER |= M_INT3; // Enable CPU INT3 which is connected to EPWM4 INT:
EINT;        // Enable Global interrupt INTM
ERTM;        // Enable Global realtime interrupt DBGM

DELAY_US(10000);       // Delay 30ms to allow ADC to determine the peak grid voltage before we can enable the PLL module
DELAY_US(10000);
DELAY_US(10000);

if (EPwm2Regs.CMPA.half.CMPA != 600) //Apart from the current control loop and PLL loop, all other control loops are designed to work
with
        //a PWM switching frequency of 250kHz. If the code stops you here, you will not be switching at 250kHz. Therefore, verify
        //that all control loops mentioned above will operate at their correct sample times.
{
        asm("   ESTOP0");
}

EPwm4Regs.ETSEL.bit.INTEN = 1;            // Enable INT. This also enables the PLL module (PLL runs at the frequency of PWM4's ISR)

for(;;)
{
        //Start converter (first start unfolding circuit and then start the current control loop)
        if ((CONVERTER_STARTED == 0) && (ERROR_STATUS.FATAL_ERROR == 0) && (ERROR_STATUS.STOP == 0))
        {
                if (start_timer > 50000) //Wait 2 seconds before checking PLL status.
                {
                        if ((PLL.theta[0] <= PI_OVER_2_PLUS_IQ20) && (PLL.theta[0] >= PI_OVER_2_MINUS_IQ20))
                        {
                                //GpioDataRegs.GPASET.bit.GPIO13 = 1;   // Debugging pin
                                GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Always clear unfolder command before setting
                                GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   //
                                GpioDataRegs.GPASET.bit.GPIO4 = 1;   // Turn on U1/U2
```

```
                        CONVERTER_STARTED = 1;
                        start_timer = 0; //Reset start timer
                    }
                    PLL.HAS_STARTED = 1; //By this time, it is assumed that the PLL must have settled to a stable operating
point.
                }
            }

            //Activate current controller
            if ((CONVERTER_STARTED)&&(ERROR_STATUS.FATAL_ERROR == 0)&&(ERROR_STATUS.STOP == 0))
            {
                if (start_timer > 50000) //Wait another 2 seconds before starting the current controller
                {
                    if ((PLL.theta[0] >= 0) && (PLL.theta[0] <= DEAD_TIME_ANGLE_ZERO_PLUS_IQ20))
                    {
                        //GpioDataRegs.GPASET.bit.GPIO11 = 1;   //For debugging
                        cc.CURRENT_CONTROLLER_ACTIVATED = 1;
                    }
                }
            }

            if (CONVERTER_STARTED)
            {
                //Check if PLL and ZCD are not matched. Mismatch is only acknowledged if Vgrid is within +/-5V
                //If there is a mismatch, use hardware zcd, else use software zcd
                if ((PLL.V_GRID_POLARITY != zcd.V_GRID_POLARITY) && ((V_grid_IQ16 > 327680) || (V_grid_IQ16 < -
327680)))
                {
                    SOURCE_OF_UNFOLDING_SIGNAL = HARDWARE_ZCD;//SOFTWARE_ZCD;//
                    cc.V_GRID_POLARITY = zcd.V_GRID_POLARITY;//PLL.V_GRID_POLARITY; //
                }
                else
                {
                    SOURCE_OF_UNFOLDING_SIGNAL = HARDWARE_ZCD; //SOFTWARE_ZCD;//
                    cc.V_GRID_POLARITY = zcd.V_GRID_POLARITY; //PLL.V_GRID_POLARITY;//
                }
/*              if ((zcd.DEAD_TIME == 0)&&(zcd.V_GRID_POLARITY == 1)&&(GpioDataRegs.GPADAT.bit.GPIO4 ==
0))
                {
                    asm("   ESTOP0");
                }*/
                GpioDataRegs.GPBSET.bit.GPIO61 = 1;   // Turn on GRN LED
            }


            if (ERROR_STATUS.FATAL_ERROR)
            {
                cc.CURRENT_CONTROLLER_ACTIVATED = 0;
                GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off unfolder U1/U2
                GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off unfolder U3/U4
                EALLOW;
                EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                EDIS;
                GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;   // Turn off GRN LED
                GpioDataRegs.GPBSET.bit.GPIO59 = 1;   // Turn on RED LED
                CONVERTER_STARTED = 0; //Turn off converter
            }

                //SFO calibration to update MEP
            SFO_status = SFO_MepEn_V5(1);
  }

} //End of "main" function
```

• *Initialization functions*

```
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include "IQmathLib.h"
#include "defines.h"
#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"   // DSP2833x Examples Include File
#include "SFO_V5.h"              // SFO V5 library headerfile - required to use SFO library functions


//All variables used in this program.
typedef struct ADC_structure{
        Uint16 mean0;
        Uint16 mean1;
        Uint16 mean2;
        Uint16 mean3;
        Uint16 mean4;
        Uint16 mean5;
        Uint16 mean6;
        Uint16 mean7;
        Uint16 mean8;
        Uint16 mean9;
        Uint16 buffer0[2], buffer1[2], buffer2[2], buffer3[2], buffer4[2],
        buffer5[2], buffer6[2], buffer7[2], buffer8[2], buffer9[2]; //Buffers for ADC, will be used to compute mean
        Uint16 timer;
}ADC_structure;

typedef struct current_controller_structure{
        _iq16 error_IQ16[2]; //Controller error and one history term
        _iq16 duty_KI_IQ16[2]; //Duty cycle from integral contribution and one history term
        _iq16 duty_KP_IQ16; //Duty cycle from proportional contribution
        _iq16 duty_lag_IQ16[2]; //Duty cycle from the lag term and one history term
        _iq16 duty_PI_IQ16; //Duty cycle from the PI controller (sum of Kp and Ki duty cycle terms)
        _iq16 duty_decoupled_IQ16; //Decoupled duty cycle
        _iq16 duty_total_IQ16[2]; //Total duty cycle from the sum of de-coupled and controller terms with one history term
        _iq16 lag_a_IQ16; //Lag coefficients
        _iq16 lag_b_IQ16;
        _iq16 KP_IQ16;       //Controller proportional gain
        _iq16 KI_z_IQ16;
        _iq16 I_grid_reference_folded_IQ16; //Controller reference current in "folded" form
        _iq16 I_grid_folded_IQ16; //Measured grid current in "folded" form
        int duty_DSP; //Duty cycle in DSP units to be fed to PWM Counters.
        long duty_frac; //Fractional duty cycle.
        char V_GRID_POLARITY; //This is used to determine the sign of the "folded" grid reference and measured currents
        char DEAD_TIME;   //The current controller's own dead time shadow variable
        char CURRENT_CONTROLLER_ACTIVATED;
} current_controller_structure;

typedef struct error_structure{
        char I_GRID_OVR_CURRENT;
        char V_INV_OVR_VOLTAGE;

        char V_GRID_OVR_VOLTAGE;
        char V_GRID_OVR_VOLTAGE_SHDW;
        char V_GRID_UNDR_VOLTAGE;
        char V_GRID_UNDR_VOLTAGE_SHDW;

        char V_GRID_WRONG_POLARITY;

        char V_PV_OVR_VOLTAGE;
        char V_PV_OVR_VOLTAGE_SHDW;
        char V_PV_UNDR_VOLTAGE;
        char V_PV_UNDR_VOLTAGE_SHDW;

        char I_PV_OVR_CURRENT;
        char TEMP1_HIGH;
        char TEMP2_HIGH;
```

```
		int v_grid_ovr_voltage_timer, v_grid_undr_voltage_timer;
		int v_pv_ovr_voltage_timer, v_pv_undr_voltage_timer;

		char FATAL_ERROR;
		char STOP;
//		char SATURATION_HIT
//		char SATURATION_
} error_structure;

typedef struct PLL_structure{
		_iq20 Kp; //PLL proportional
		_iq20 Ki; //PLL integral term (Ki*Ts/2)
		_iq20 Ki2; //(VCO) Integrator coefficient for computing the grid angle (Ts/2) in iq20 format
		_iq20 cos_theta; //cos_theta
		_iq20 sin_theta; //sin_theta
		_iq20 w[3]; //This is the angular velocity from the PLL alongside two history term
		_iq20 w_filtered[3]; //This is the filtered angular velocity along with two history terms
		_iq20 w_non_offset_P; //This is the proportional component of the unfiltered output of the PLL's PI controller.
		_iq20 w_non_offset_I[2]; //This is the integral component of the unfiltered output of the PLL's PI controller along with one history
term
		_iq20 w_offset; //This is the offset w that will be added (feedforward) to improve response of the PLL and reduce controller effort
		_iq20 theta[2]; //Angle in radians at output of VCO, with one history term
		_iq20 SOGI_Valpha[3]; //This is the alpha voltage from the second order generalized integrator
		_iq20 SOGI_Vbeta[3]; //This is the beta voltage from the second order generalized integrator
		_iq20 SOGI_k; //SOGI k coefficient
		_iq20 SOGI_wnTs;
		_iq20 SOGI_x;
		_iq20 SOGI_y;
		_iq20 SOGI_denominator;
		_iq20 SOGI_quotient;
		_iq20 SOGI_b0;
		_iq20 SOGI_a1;
		_iq20 SOGI_a2;
		_iq20 SOGI_ky;
		_iq20 SOGI_qb0;
		_iq20 V_grid_pu[3]; //Normalized (per-unitized) instantaneous grid voltage with two history terms;
		_iq20 V_grid_base;	//Base grid voltage
		_iq20 error[3];//Present error term and two history terms
		_iq20 notch_a; //Notch filter coefficient a(0.979127)
		_iq20 notch_b; //Notch filter coefficient b(-1.957364)
		_iq20 notch_f; //Notch filter coefficient f(0.958254)
		_iq20 notch_out[3]; //Output of notch filter with two history terms.
		_iq20 Vd; //Direct-axis voltage
		_iq20 Vq; //Quadrature-axis voltage
		_iq20 V_grid_pk_pu[2]; //Normalized (per-unitized) unfiltered peak grid voltage
		_iq20 V_grid_pk_filtered_pu[2]; //Normalized filtered peak grid voltage
		_iq20 A,B,C,D,E; //Variables that will be used as coefficients for filtering the angular speed and the peak normalized grid voltage.
Can also be constants in IQ20 format
		char PERIOD_HAS_OCCURED; //Takes note of when one grid period has occurred
		char HAS_STARTED;
		char V_GRID_POLARITY; //Grid polarity as given by PLL module
		char DEAD_TIME; //Dead time as given  by PLL module

/*
		char LOCKED; //This variable is set when it is determined that the PLL is 'locked' (when frequency varies within a narrowband)
		char FAILED; //This variable is set when it is determined that the PLL has failed to lock after several attempts.
		Uint16 lock_timer; //Timer used to keep track of PLL Locked status
*/
}PLL_structure;

typedef struct zcd_structure{
		Uint16 half_period[2]; //Grid half period
		Uint16 full_period; //Grid full period
		Uint16 half_period_counter; //Counter for determining the half_period
		Uint16 full_period_counter; //Counter for determining grid angle from 0 to 360 degrees
		Uint16 angle_pu_IQ16; //Grid angle measurement by using the hardware zcd method.
		char V_GRID_POLARITY; //Grid polarity as given by zcd circuit.
		char PERIOD_TOO_LOW; //High frequency limit as given by zcd module
		char PERIOD_TOO_HIGH; //Low frequency limit as given by zcd module
		char DEAD_TIME; //Dead time event indicator
```

```
} zcd_structure;

typedef struct power_control_structure{
        _iq16 P_ref_IQ16;     //Reference active power in IQ16 format in IQ16 format.
        _iq16 Q_ref_IQ16;     //Reference reactive power in IQ16 format in IQ16 format.
        _iq16 error_P_ref[2]; //Active power tracking error
        _iq16 error_Q_ref[2]; //Reactive power tracking error
        _iq16 P_dc_IQ16;      //Active power computed by P-Q transform method in IQ16 format.
        _iq16 Q_dc_IQ16;      //Reactive power computed by P-Q transform method in IQ16 format.
        long P_output_avg_IQ16; //Average output power. Computed by integration. IQ16 format
        long P_input_avg_IQ16; //Average input power. Computed by integration. IQ16 format
        _iq16 I_grid_reference_peak_IQ16; //Peak reference grid current from power loop. In IQ16 format.
        _iq16 Kp;  //Proportional gain to be used for power control loop, in IQ16 format
        _iq16 Kiz; //(Integrator gain)*(Ts/2) integrator gain for power control loop in IQ16 format
        _iq16 inst_input_power_integrator; //Sums the instantaneous input power at regular sample intervals.
        _iq16 inst_output_power_integrator; //Sums the instantaneous output power at regular sample intervals.
        Uint16 timer; //Control loop timer (approximates the sampling period).
        Uint16 avg_power_counter; //Counter to be used in computing the average power by integration method
} power_control_structure;

//All variables used in this program.
volatile _iq16 V_pv_IQ16; //PV voltage
volatile _iq16 I_pv_IQ16; //PV current
volatile _iq16 V_grid_IQ16; //Grid voltage
volatile _iq16 V_inv_IQ16; //Inverter pseudo-dc link voltage
volatile _iq16 I_grid_IQ16; //Grid-injected current
volatile _iq16 I_grid_reference_IQ16; //amperes
volatile _iq16 I_grid_reference_peak_IQ16 = _IQ16(0.3);
volatile _iq16 I_grid_angle_offset_IQ16 = 0;
volatile _iq16 dummy1_IQ16;
volatile _iq16 dummy2_IQ16;
volatile _iq16 dummy3_IQ16; //dummy variables for computations
volatile _iq16 V_grid_pk_IQ16[2]; //Instantaneous peak grid voltage container and one history term
//volatile _iq16 angle_pu_IQ16;
volatile char V_grid_polarity; //Grid voltage polarity
volatile struct current_controller_structure cc; //Create an instance of a current controller
volatile struct error_structure ERROR_STATUS; //Create an instance of an error structure
volatile struct ADC_structure ADC; //Create an instance of an ADC structure
volatile struct PLL_structure PLL; //Create an instance of a PLL structure
volatile struct zcd_structure zcd; //Create an instance of a zcd structure
volatile struct power_control_structure power_loop; //Create an instance of a power control structure
volatile int debug1[200];
volatile int debug2[200];
volatile int dummy_counter;
int k = 0; //Counter
int j = 0; // counter
int m = 0; //counter
Uint16 start_timer = 0;
char CONVERTER_STARTED = 0;
char SOURCE_OF_UNFOLDING_SIGNAL = HARDWARE_ZCD;

volatile int MEP_ScaleFactor[7] = {0,0,0,0,0,0}; // Global array used by the SFO library. Only HRPWM1A will be used to compute scale factor.
So it's HRPWM must be disabled
volatile struct EPWM_REGS *ePWM[7] =
                          {  &EPwm1Regs, &EPwm1Regs, &EPwm2Regs,     &EPwm3Regs,
                             &EPwm4Regs,     &EPwm5Regs,       &EPwm6Regs};
volatile int SFO_status = 0;

void init_variables(void)
{
        V_pv_IQ16 = 0;
        I_pv_IQ16 = 0;
        V_grid_IQ16 = 0;
        V_inv_IQ16 = 0;
        I_grid_IQ16 = 0;
        I_grid_reference_IQ16 = 0;
        I_grid_reference_peak_IQ16 = _IQ16(0);
        I_grid_angle_offset_IQ16 = 0;
        //angle_pu_IQ16 = 0;
        dummy1_IQ16 = 0;
```

```
dummy2_IQ16 = 0;
dummy3_IQ16 = 0;
V_grid_pk_IQ16[0] = 0; V_grid_pk_IQ16[1] = 0; //Instantaneous peak grid voltage container and one history term
V_grid_polarity = POSITIVE_POLARITY;
dummy_counter = 0;

//Initialize ADC
/*Ipv = ADC_buff0 (or ADC_A0)
 * Vpv = ADC_buff1 (or ADC_B0)
 * Iac = ADC_buff2 (or ADC_A1)
 * Vg = ADC_buff3 (or ADC_B1)
 * TEMP1 = ADC_buff4 (or ADC_A2)
 * Vinv = ADC_buff5 (or ADC_B2)
 * POT1 = ADC_buff6 (or ADC_A3)
 * TEMP2 = ADC_buff7 (or ADC_B3)
 * POT2 = ADC_buff8 (or ADC_B4)
 *
*/
ADC.mean0 = 0; ADC.mean1 = 0; ADC.mean2 = 0; ADC.mean3 = 0; ADC.mean4 = 0;
ADC.mean5 = 0; ADC.mean6 = 0; ADC.mean7 = 0; ADC.mean8 = 0; ADC.mean9 = 0;
ADC.buffer0[0] = 0; ADC.buffer0[1] = 0; ADC.buffer1[0] = 0; ADC.buffer1[1] = 0;
ADC.buffer2[0] = 0; ADC.buffer2[1] = 0; ADC.buffer3[0] = 0; ADC.buffer3[1] = 0;
ADC.buffer4[0] = 0; ADC.buffer4[1] = 0; ADC.buffer5[0] = 0; ADC.buffer5[1] = 0;
ADC.buffer6[0] = 0; ADC.buffer6[1] = 0; ADC.buffer7[0] = 0; ADC.buffer7[1] = 0;
ADC.buffer8[0] = 0; ADC.buffer8[1] = 0; ADC.buffer9[0] = 0; ADC.buffer9[1] = 0;

//Initialize current_controller
cc.error_IQ16[0] = 0; cc.error_IQ16[1] = 0;
cc.duty_KI_IQ16[0] = 0; cc.duty_KI_IQ16[1] = 0;
cc.duty_KP_IQ16 = 0;
cc.duty_lag_IQ16[0] = 0; cc.duty_lag_IQ16[1] = 0;
cc.duty_PI_IQ16 = 0;
cc.duty_decoupled_IQ16 = 0;
cc.duty_total_IQ16[0] = 0; cc.duty_total_IQ16[1] = 0;
cc.KP_IQ16 = 2772;
cc.KI_z_IQ16 = 94; //132;
cc.lag_a_IQ16 = 21845;
cc.lag_b_IQ16 = 21845;
cc.I_grid_folded_IQ16 = 0;
cc.I_grid_reference_folded_IQ16 = 0;
cc.duty_DSP = 0;
cc.duty_frac = 0; //Fractional duty cycle.
cc.V_GRID_POLARITY = POSITIVE_POLARITY;
cc.DEAD_TIME = 0; //The current controller's own dead time shadow variable
cc.CURRENT_CONTROLLER_ACTIVATED = 0;

//Initialize error status
ERROR_STATUS.I_GRID_OVR_CURRENT = 0; ERROR_STATUS.V_INV_OVR_VOLTAGE = 0;
ERROR_STATUS.V_GRID_OVR_VOLTAGE = 0; ERROR_STATUS.V_GRID_OVR_VOLTAGE_SHDW = 0;
ERROR_STATUS.V_GRID_UNDR_VOLTAGE = 0; ERROR_STATUS.V_GRID_UNDR_VOLTAGE_SHDW = 0;
ERROR_STATUS.V_GRID_WRONG_POLARITY = 0; ERROR_STATUS.V_PV_OVR_VOLTAGE = 0;
ERROR_STATUS.V_PV_OVR_VOLTAGE_SHDW = 0; ERROR_STATUS.V_PV_UNDR_VOLTAGE = 0;
ERROR_STATUS.V_PV_UNDR_VOLTAGE_SHDW = 0;

ERROR_STATUS.I_PV_OVR_CURRENT = 0;
ERROR_STATUS.TEMP1_HIGH = 0;
ERROR_STATUS.TEMP2_HIGH = 0;
ERROR_STATUS.v_grid_ovr_voltage_timer = 0; ERROR_STATUS.v_grid_undr_voltage_timer = 0;
ERROR_STATUS.v_pv_ovr_voltage_timer = 0; ERROR_STATUS.v_pv_undr_voltage_timer = 0;

ERROR_STATUS.FATAL_ERROR = 0;
ERROR_STATUS.STOP = 0;

//Initialize debugging variables
for (k=0; k < 200; k = k+1)
{
		debug1[k] = 0;
		debug2[k] = 0;
}
```

```
	//Initialize PLL variables
	PLL.Kp = _IQ20(300); //PLL proportional
	PLL.Ki = _IQ20(1); //PLL integral term (K*Ts/2) where K = 50000 (that's a huge integrator!)
	PLL.Ki2 = 21; //Integrator coefficient to compute the grid angle from speed (this is otherwise known as a VCO) ( = 1/(2*25000) * 
2^20)
	PLL.cos_theta  = _IQ20(1); //cos_theta
	PLL.sin_theta = 0; //sin_theta
	PLL.w[0] = W_NOMINAL_IQ20; PLL.w[1] = W_NOMINAL_IQ20; PLL.w[2] = W_NOMINAL_IQ20;//Angular velocity
	PLL.w_filtered[0] = W_NOMINAL_IQ20; PLL.w_filtered[1] = W_NOMINAL_IQ20; PLL.w_filtered[2] = W_NOMINAL_IQ20; 
//Filtered angular velocity
	PLL.w_non_offset_P = 0; //Proportional result of PLL PI controller
	PLL.w_non_offset_I[0] = 0; PLL.w_non_offset_I[1] = 0; //Integral result of PLL PI controller
	PLL.w_offset = W_NOMINAL_IQ20; //Offset angular speed for faster response
	PLL.theta[0] = 0; PLL.theta[1] = 0; //Angle in radians from PLL
	PLL.SOGI_Valpha[0] = 0; PLL.SOGI_Valpha[1] = 0; PLL.SOGI_Valpha[2] = 0; //Alpha-axis voltage
	PLL.SOGI_Vbeta[0] = 0; PLL.SOGI_Vbeta[1] = 0; PLL.SOGI_Vbeta[2] = 0; //Beta-axis voltage
	PLL.SOGI_k = 838861; //SOGI k coefficient in IQ format (chosen as 0.8*2^20)
	PLL.SOGI_wnTs = 0;
	PLL.SOGI_x = 0;
	PLL.SOGI_y = 0;
	PLL.SOGI_denominator = 0;
	PLL.SOGI_quotient = 0;
	PLL.SOGI_b0 = 0;
	PLL.SOGI_a1 = 0;
	PLL.SOGI_a2 = 0;
	PLL.SOGI_ky = 0;
	PLL.SOGI_qb0 = 0;
	PLL.V_grid_pu[0] = _IQ20div(V_grid_IQ16<<4,(long)V_GRID_BASE_IQ20); //Since V_grid_IQ16 is an IQ16, it needs to be 
multiplied by 16 (left shift of 4) to convert to IQ20
	PLL.V_grid_pu[1] = PLL.V_grid_pu[0];
	PLL.V_grid_pu[2] = PLL.V_grid_pu[0];
	PLL.V_grid_base = (long)V_GRID_BASE_IQ20;	//Base grid voltage
	PLL.error[0] = 0; PLL.error[1] = 0; PLL.error[2] = 0; //PLL error
	PLL.notch_a = 1026689; //Notch filter coefficient a(0.979127*2^20)
	PLL.notch_b = -2052445; //Notch filter coefficient b(-1.957364*2^20)
	PLL.notch_f = 1004802; //Notch filter coefficient f(0.958254*2^20)
	PLL.notch_out[0] = 0; PLL.notch_out[1] = 0; PLL.notch_out[2] = 0; //Output of notch filter with two history terms.
	PLL.Vd = 0;
	PLL.Vq = 0;
	PLL.V_grid_pk_pu[0] = 0; PLL.V_grid_pk_pu[1] = 0;  //Normalized (per-unitized) unfiltered peak grid voltage
	PLL.V_grid_pk_filtered_pu[0] = 0; PLL.V_grid_pk_filtered_pu[1] = 0; //Normalized filtered peak grid voltage
	PLL.A = 2;// (1.576333e-6 * 2^20)Variables that will be used as coefficients for filtering the angular speed and the peak normalized 
grid voltage. Can also be constants in IQ20 format
	PLL.B = -2093426; // (-1.99644623 * 2^20)
	PLL.C = 1044856; // (0.99645253 * 2^20)
	PLL.D = 1316; //(1.255e-3 *2^20)
	PLL.E = 1045944;//(0.99748988 * 2^20)
	PLL.PERIOD_HAS_OCCURED = 0;
	PLL.HAS_STARTED = 0;
	PLL.DEAD_TIME = 0; //Dead time as given  by PLL module

	//Initialize zcd variables
	zcd.half_period[0] = 1042; //if ADC frequency (125kHz) is used, then 1/120 seconds correspond to 1042 counts
	zcd.half_period[1] = 1042;
	zcd.full_period = 2083; //Similarly for 1/60 seconds
	zcd.angle_pu_IQ16 = 0; //Grid angle measurement by using the hardware zcd method.
	zcd.PERIOD_TOO_HIGH = 0;
	zcd.PERIOD_TOO_LOW = 0;
	zcd.half_period_counter = 0;
	zcd.full_period_counter = 0;
	zcd.DEAD_TIME = 0; //Dead time off

	power_loop.P_ref_IQ16 = 0;	//Reference active power in IQ16 format in IQ16 format.
	power_loop.Q_ref_IQ16 = 0;	//Reference reactive power in IQ16 format in IQ16 format.
	power_loop.error_P_ref[0] = 0; power_loop.error_P_ref[1] = 0;//Active power tracking error
	power_loop.error_Q_ref[0] = 0; power_loop.error_Q_ref[1] = 0;//Reactive power tracking error
	power_loop.P_dc_IQ16 = 0;	//Active power computed by P-Q transform method in IQ16 format.
	power_loop.Q_dc_IQ16 = 0;	//Reactive power computed by P-Q transform method in IQ16 format.
	power_loop.P_output_avg_IQ16 = 0; //Counter to be used in computing the average output power by integration method
	power_loop.P_input_avg_IQ16 = 0; //Counter to be used in computing the average input power by integration method
```

```
            power_loop.Kp = _IQ16(1);       //Proportional gain to be used for power control loop, in IQ16 format
            power_loop.Kiz = _IQ16(.008);   //(Integrator gain)*(Ts/2) integrator gain for power control loop in IQ16 format
            power_loop.inst_input_power_integrator = 0; //Sums the instantaneous input power at regular sample intervals.
            power_loop.inst_output_power_integrator = 0; //Sums the instantaneous output power at regular sample intervals.
            power_loop.timer = 0; //Control loop timer (approximates the sampling period).
            power_loop.avg_power_counter = 0; //Counter to be used in computing the average power by integration method

            return;
}
void init_epwm1(void)
{
  EALLOW;
                    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1;   // Disable pull-up on GPIO0 (EPWM1A)
                    GpioCtrlRegs.GPAPUD.bit.GPIO1 = 1;   // Disable pull-up on GPIO1 (EPWM1B)

                    /* Configure ePWM-1 pins using GPIO regs*/
                    // This specifies which of the possible GPIO pins will be ePWM1 functional pins.
                    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;   // Configure GPIO0 as EPWM1A
                    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;   // Configure GPIO1 as EPWM1B
  EDIS;

  // Setup TBCLK
  EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
  EPwm1Regs.TBPRD = 600;//EPWM1_TIMER_TBPRD;       // Set timer period
  EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Disable phase loading
  EPwm1Regs.TBPHS.half.TBPHS = 0x0000;       // Phase is 0
  EPwm1Regs.TBCTR = 0x0000;                  // Clear counter
  EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;   // Clock ratio to SYSCLKOUT
  EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;

  // Setup shadow register load on ZERO
  EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
  EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
  EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
  EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

  // Set Compare values
  EPwm1Regs.CMPA.half.CMPA = 0;//EPWM1_MIN_CMPA;    // Set compare A value
  EPwm1Regs.CMPB = 0;//EPWM1_MIN_CMPB;              // Set Compare B value

  // Set actions
  EPwm1Regs.AQCTLA.bit.ZRO = AQ_SET;          // Set PWM1A on Zero
  EPwm1Regs.AQCTLA.bit.CAU = AQ_CLEAR;        // Clear PWM1A on event A, up count

  EPwm1Regs.AQCTLB.bit.ZRO = AQ_CLEAR;//AQ_SET;          // Set PWM1B on Zero
  EPwm1Regs.AQCTLB.bit.CBU = AQ_SET;//AQ_CLEAR;        // Clear PWM1B on event B, up count

  // Interrupt where we will change the Compare Values
  EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
  EPwm1Regs.ETSEL.bit.INTEN = 0;              // Disable interrupt INT
  EPwm1Regs.ETPS.bit.INTPRD = 0x00;         // Disable the interrupt event counter. No interrupt will be generated and ETFRC[INT] is
ignored.

  // Active Low PWMs - Setup Deadband
  EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
  EPwm1Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;//DB_ACTV_LO;
  EPwm1Regs.DBRED = 20;//5;//EPWM1_MIN_DB;
  EPwm1Regs.DBFED = 40;//5;//50;//EPWM1_MIN_DB; //Falling edge deadtime

  EALLOW;

  //Trip-zone configuration for fault conditions
  EPwm1Regs.TZSEL.bit.OSHT1 = 0x01; //Enable TZ1 as a one-shot trip source for this ePWM module
  EPwm1Regs.TZCTL.bit.TZA = 0x02; //Force EPWM1A to a low state
  EPwm1Regs.TZCTL.bit.TZB = 0x02; //Force EPWM1B to a low state

  EDIS;

  while (SFO_MepDis_V5(1) != 1); //Run HRPWM calibration routine while PWM1 is disabled and use the result to seed the
MEP_ScaleFactor[0].
```

```
  MEP_ScaleFactor[0] = MEP_ScaleFactor[1]; //Seeding initial MEP_ScaleFactor (Necessary step for SFO_MepEn_V5 to work well)
}


void init_epwm2(void)
{
        EALLOW;
                        GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1;    // Disable pull-up on GPIO2 (EPWM2A)
                        GpioCtrlRegs.GPAPUD.bit.GPIO3 = 1;    // Disable pull-up on GPIO3 (EPWM3B)

                        /* Configure ePWM-2 pins using GPIO regs*/
                        // This specifies which of the possible GPIO pins will be ePWM2 functional pins.

                        GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // Configure GPIO2 as EPWM2A
                        GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;   // Configure GPIO3 as EPWM2B
  EDIS;
  // Setup TBCLK
  EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
  EPwm2Regs.TBPRD = SWITCHING_PERIOD;//EPWM2_TIMER_TBPRD;       // Set timer period
  EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Disable phase loading
  EPwm2Regs.TBPHS.half.TBPHS = 0x0000;       // Phase is 0
  EPwm2Regs.TBCTR = 0x0000;                  // Clear counter
  EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; //TB_DIV2;   // Clock ratio to SYSCLKOUT
  EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1; //TB_DIV2;

  // Setup shadow register load on ZERO
  EPwm2Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
  EPwm2Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
  EPwm2Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
  EPwm2Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

  // Set Compare values
  EPwm2Regs.CMPA.half.CMPA = SWITCHING_PERIOD;                  // Set compare A value. We want PWMB to be zero on start, so
make PWMA maximum on start.
  EPwm2Regs.CMPB = 0;//EPWM2_MAX_CMPB;              // Set Compare B value
  EPwm2Regs.CMPA.half.CMPAHR = (1 << 8); //Set initial value for CMPAHR

  // Set actions
  EPwm2Regs.AQCTLA.bit.ZRO = AQ_SET;//AQ_CLEAR;           // Clear PWM2A on Period
  EPwm2Regs.AQCTLA.bit.CAU = AQ_CLEAR;//AQ_SET;            // Set PWM2A on event A, up count

  EPwm2Regs.AQCTLB.bit.ZRO = AQ_CLEAR;           // Clear PWM2B on Period
  EPwm2Regs.AQCTLB.bit.CBU = AQ_SET;            // Set PWM2B on event B, up count

  //Enable SOCA
  EPwm2Regs.ETSEL.bit.SOCAEN = 0x01; //Enable EPWM2 SOCA Pulse. This will trigger the ADC conversion sequence
  EPwm2Regs.ETSEL.bit.SOCASEL = 0x01; //Enable event time-base counter equal to period (TBCTR = TBPRD)
  EPwm2Regs.ETPS.bit.SOCAPRD = 0x02;       // Generate pulse on 2nd event. This means that if PWM switching frequency is 250kHz, then
ADC frequency is 125kHz.

  // Configure Interrupt
  EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero event
  EPwm2Regs.ETSEL.bit.INTEN = 0;//1;                //Disable INT. Interrupt will be enable in main.c
  EPwm2Regs.ETPS.bit.INTPRD = 0x00; //Disable interrupt event counter //ET_3RD;            // Generate INT on 3rd event

  // Active Low PWMs - Setup Deadband
  EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
  EPwm2Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;//DB_ACTV_LO;
  EPwm2Regs.DBRED = 10;//5;//EPWM1_MIN_DB;
  EPwm2Regs.DBFED = 20;//5;//50;//EPWM1_MIN_DB;

  //Trip-zone configuration for fault conditions and High-resoultion configuration
  EALLOW;
  EPwm2Regs.TZSEL.bit.OSHT1 = 0x01; //Enable TZ1 as a one-shot trip source for this ePWM module
  EPwm2Regs.TZCTL.bit.TZA = 0x02; //Force EPWM1A to a low state
  EPwm2Regs.TZCTL.bit.TZB = 0x02; //Force EPWM1B to a low state
  EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition

  EPwm2Regs.HRCNFG.bit.HRLOAD = 0; //High resolution counter loaded at "counter equal zero"
  EPwm2Regs.HRCNFG.bit.CTLMODE = 0; //CMPAHR controls edge position
```

```
  EPwm2Regs.HRCNFG.bit.EDGMODE = 0x02; //MEP control is done on falling edge. Any non-zero value for this configuration effectively
activates the HRPWM
  EDIS;
}

void init_epwm5(void)
{
        EALLOW;
                              GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1;    // Disable pull-up on GPIO8 (EPWM5A)
                              GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1;    // Disable pull-up on GPIO9 (EPWM5B)

                              /* Configure ePWM-5 pins using GPIO regs*/
                              // This specifies which of the possible GPIO pins will be ePWM5 functional pins.

                              GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1;   // Configure GPIO8 as EPWM5A
                              GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1;   // Configure GPIO9 as EPWM5B
  EDIS;
  // Setup TBCLK
  EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
  EPwm5Regs.TBPRD = SWITCHING_PERIOD;       // Set timer period
  EPwm5Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Disable phase loading
  EPwm5Regs.TBPHS.half.TBPHS = 0x0000;       // Phase is 0
  EPwm5Regs.TBCTR = 0x0000;                  // Clear counter
  EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1; //TB_DIV2;   // Clock ratio to SYSCLKOUT
  EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1; //TB_DIV2;

  // Setup shadow register load on ZERO
  EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
  EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
  EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
  EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

  // Set Compare values
  EPwm5Regs.CMPA.half.CMPA = SWITCHING_PERIOD;                  // Set compare A value. We want PWMB to be zero on start, so
make PWMA maximum on start.
  EPwm5Regs.CMPB = 0;//EPWM2_MAX_CMPB;              // Set Compare B value
  EPwm5Regs.CMPA.half.CMPAHR = (1 << 8); //Set initial value for CMPAHR

  // Set actions
  EPwm5Regs.AQCTLA.bit.ZRO = AQ_CLEAR;           // Clear PWM5A on Period
  EPwm5Regs.AQCTLA.bit.CAU = AQ_SET;             // Set PWM5A on event A, up count

  EPwm5Regs.AQCTLB.bit.ZRO = AQ_SET;           // Clear PWM5B on Period
  EPwm5Regs.AQCTLB.bit.CBU = AQ_CLEAR;             // Set PWM5B on event B, up count

  //Enable SOCA
  EPwm5Regs.ETSEL.bit.SOCAEN = 0x00; //Disable EPWM5 SOCA Pulse.
  EPwm5Regs.ETSEL.bit.SOCASEL = 0x01; //Enable event time-base counter equal to period (TBCTR = TBPRD)
  EPwm5Regs.ETPS.bit.SOCAPRD = 0x02;       // Generate pulse on 2nd event. This means that if PWM switching frequency is 250kHz, then
ADC frequency is 125kHz.

  // Configure Interrupt
  EPwm5Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;       // Select INT on Zero event
  EPwm5Regs.ETSEL.bit.INTEN = 0;                  //Disable INT.
  EPwm5Regs.ETPS.bit.INTPRD = 0x00; //Disable interrupt event counter //ET_3RD;           // Generate INT on 3rd event

  // Active Low PWMs - Setup Deadband
  EPwm5Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
  EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC;//DB_ACTV_LO;
  EPwm5Regs.DBRED = 10;//5;//EPWM1_MIN_DB;
  EPwm5Regs.DBFED = 20;//5;//50;//EPWM1_MIN_DB;

  //Trip-zone configuration for fault conditions and High-resoultion configuration
  EALLOW;
  EPwm5Regs.TZSEL.bit.OSHT1 = 0x01; //Enable TZ1 as a one-shot trip source for this ePWM module
  EPwm5Regs.TZCTL.bit.TZA = 0x02; //Force EPWM5A to a low state
  EPwm5Regs.TZCTL.bit.TZB = 0x02; //Force EPWM5B to a low state
  EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition

  EPwm5Regs.HRCNFG.bit.HRLOAD = 0; //High resolution counter loaded at "counter equal zero"
```

```
  EPwm5Regs.HRCNFG.bit.CTLMODE = 0; //CMPAHR controls edge position
  EPwm5Regs.HRCNFG.bit.EDGMODE = 0x01; //MEP control is done on rising edge
  EDIS;
}


void init_epwm4(void) //This EPWM4 is only used as to generate an ISR for low-priority control and as a clock source for some housekeeping
functions
{
  EALLOW;
                                   GpioCtrlRegs.GPAPUD.bit.GPIO6 = 1;    // Disable pull-up on GPIO6 (EPWM4A)
                                   GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;   // Configure GPIO6 as EPWM4A
  EDIS;

  // Setup TBCLK
  EPwm4Regs.TBCTL.bit.CTRMODE = TB_COUNT_UP; // Count up
  EPwm4Regs.TBPRD = 1500;//EPWM3_TIMER_TBPRD;       // Set timer period. Corresponds to about 25kHz
  EPwm4Regs.TBCTL.bit.PHSEN = TB_DISABLE;    // Disable phase loading
  EPwm4Regs.TBPHS.half.TBPHS = 0x0000;       // Phase is 0
  EPwm4Regs.TBCTR = 0x0000;                  // Clear counter
  EPwm4Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;   // Clock ratio to SYSCLKOUT
  EPwm4Regs.TBCTL.bit.CLKDIV = 0x02; //Divide TBCLK by 4

  // Setup shadow register load on ZERO
  EPwm4Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
  EPwm4Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
  EPwm4Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
  EPwm4Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

 // Set Compare values
  EPwm4Regs.CMPA.half.CMPA = 300;//10000;//EPWM3_MIN_CMPA; // Set compare A value
  EPwm4Regs.CMPB = 300;//10000;//EPWM3_MAX_CMPB;          // Set Compare B value

  // Set Actions
  EPwm4Regs.AQCTLA.bit.ZRO = AQ_SET;//AQ_CLEAR;           // Clear PWM2A on Period
  EPwm4Regs.AQCTLA.bit.CAU = AQ_CLEAR;//AQ_SET;             // Set PWM2A on event A, up count

  EPwm4Regs.AQCTLB.bit.ZRO = AQ_SET;           // Clear PWM2B on Period
  EPwm4Regs.AQCTLB.bit.CBU = AQ_CLEAR;             // Set PWM2B on event B, up count

  // Interrupt where we will change the Compare Values
  EPwm4Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
  EPwm4Regs.ETSEL.bit.INTEN = 0;              // Disble INT. INT will be enabled in main when conditions are right
  EPwm4Regs.ETPS.bit.INTPRD = 0x01; //interrupt on 1st event
}

//---------------------------------------------------------------------------
// InitAdc:
//---------------------------------------------------------------------------
// This function initializes ADC to a known state.
//
void init_adc(void)
{
   extern void DSP28x_usDelay(Uint32 Count);


   // *IMPORTANT*
              // The ADC_cal function, which  copies the ADC calibration values from TI reserved
              // OTP into the ADCREFSEL and ADCOFFTRIM registers, occurs automatically in the
              // Boot ROM. If the boot ROM code is bypassed during the debug process, the
              // following function MUST be called for the ADC to function according
              // to specification. The clocks to the ADC MUST be enabled before calling this
              // function.
              // See the device data manual and/or the ADC Reference
              // Manual for more information.

                 EALLOW;
                        SysCtrlRegs.PCLKCR0.bit.ADCENCLK = 1;
                        ADC_cal();
                        EDIS;
```

```
    // To powerup the ADC the ADCENCLK bit should be set first to enable
    // clocks, followed by powering up the bandgap, reference circuitry, and ADC core.
    // Before the first conversion is performed a 5ms delay must be observed
            // after power up to give all analog circuits time to power up and settle

    // Please note that for the delay function below to operate correctly the
            // CPU_RATE define statement in the DSP2833x_Examples.h file must
            // contain the correct CPU clock period in nanoseconds.

    AdcRegs.ADCTRL3.all = 0x00E0;  // Power up bandgap/reference/ADC circuits
    DELAY_US(ADC_usDELAY);         // Delay before converting ADC channels

    AdcRegs.ADCTRL1.bit.ACQ_PS = ADC_SHCLK; //S/H window
    AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 0x01; //SEQ1 Interrupt Enable (Works for cascaded sequencer as well)
    AdcRegs.ADCTRL2.bit.RST_SEQ1 = 0x01; //Reset Sequencer to state Conv00
    AdcRegs.ADCTRL2.bit.EPWM_SOCA_SEQ1 = 0x01; //Allow the cascaded sequencer to be triggered by EMPWx SOCA
    AdcRegs.ADCTRL3.bit.ADCCLKPS = ADC_CKPS; //ADC clock divider

    AdcRegs.ADCTRL3.bit.SMODE_SEL = 0x1; // Setup simultaneous sampling mode
    AdcRegs.ADCTRL1.bit.SEQ_CASC = 0x1; // Setup cascaded sequencer mode

    AdcRegs.ADCMAXCONV.all = 0x0004; // 5 double conv's (10 total)
    AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; // Setup conv from ADCINA0 & ADCINB0
    AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; // Setup conv from ADCINA1 & ADCINB1
    AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2; // Setup conv from ADCINA2 & ADCINB2
    AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3; // Setup conv from ADCINA3 & ADCINB3
    AdcRegs.ADCCHSELSEQ2.bit.CONV04 = 0x4; // Setup conv from ADCINA4 & ADCINB4

    AdcRegs.ADCTRL1.bit.CONT_RUN = 0; //Disable continuous run. You must now manually re-start sampling and reset sequencer

}

void init_gpio(void)
{
            //GPIO8 is used for debugging while GPIO7 is used for external interrupt (zero-crossing)
            EALLOW;
            GpioCtrlRegs.GPAPUD.bit.GPIO10 = 0;   // Enable pullup on GPIO10
            GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 0;  // GPIO10 = GPIO10
            GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;   // GPIO10 = output
            GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;    // Clear output latch of debug pin

            GpioCtrlRegs.GPAPUD.bit.GPIO11 = 0;   // Enable pullup on GPIO11
            GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 0;  // GPIO11 = GPIO11
            GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;   // GPIO11 = output
            GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;    // Clear output latch of debug pin

            GpioCtrlRegs.GPAPUD.bit.GPIO12 = 0;   // Enable pullup on GPIO12
            GpioCtrlRegs.GPAMUX1.bit.GPIO12 = 0;  // GPIO12 = GPIO12
            GpioCtrlRegs.GPADIR.bit.GPIO12 = 1;   // GPIO12 = output
            GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;    // Clear output latch of debug pin

            GpioCtrlRegs.GPAPUD.bit.GPIO13 = 1;   // Disable pullup on GPIO13
            GpioCtrlRegs.GPAMUX1.bit.GPIO13 = 0;  // GPIO13 = GPIO13
            GpioCtrlRegs.GPADIR.bit.GPIO13 = 1;   // GPIO13 = output
            GpioDataRegs.GPACLEAR.bit.GPIO13 = 1;    // Clear output latch of debug pin

            GpioCtrlRegs.GPAPUD.bit.GPIO14 = 1;   // Disable pullup on GPIO14
            GpioCtrlRegs.GPAMUX1.bit.GPIO14 = 0;  // GPIO14 = GPIO14
            GpioCtrlRegs.GPADIR.bit.GPIO14 = 1;   // GPIO14 = output
            GpioDataRegs.GPACLEAR.bit.GPIO14 = 1;    // Clear output latch of debug pin

            GpioCtrlRegs.GPAPUD.bit.GPIO15 = 1;   // Disable pullup on GPIO15
            GpioCtrlRegs.GPAMUX1.bit.GPIO15 = 0;  // GPIO15 = GPIO15
            GpioCtrlRegs.GPADIR.bit.GPIO15 = 1;   // GPIO15 = output
            GpioDataRegs.GPACLEAR.bit.GPIO15 = 1;    // Clear output latch of debug pin

            GpioCtrlRegs.GPBPUD.bit.GPIO61 = 1;   // Disable pullup on GPIO61 (GRN)
            GpioCtrlRegs.GPBMUX2.bit.GPIO61 = 0;  // GPIO61 = GPIO61
```

```
            GpioCtrlRegs.GPBDIR.bit.GPIO61 = 1;   // GPIO61 = output
            GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;   // Clear output latch of debug pin

            GpioCtrlRegs.GPBPUD.bit.GPIO59 = 1;   // Disable pullup on GPIO59 (RED)
            GpioCtrlRegs.GPBMUX2.bit.GPIO59 = 0;  // GPIO59 = GPIO59
            GpioCtrlRegs.GPBDIR.bit.GPIO59 = 1;   // GPIO59 = output
            GpioDataRegs.GPBCLEAR.bit.GPIO59 = 1;   // Clear output latch of debug pin


            //GPIO7 will be used as the zero crossing input
            GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 0;        // GPIO7 = GPIO7
            GpioCtrlRegs.GPADIR.bit.GPIO7 = 0;         // input
            GpioCtrlRegs.GPAQSEL1.bit.GPIO7 = 0x02; //Qualification using 6 samples                                    0;      // Xint1 Synch
to SYSCLKOUT only
            GpioCtrlRegs.GPACTRL.bit.QUALPRD0 = 75;          //Sampling frequency is 1MHz
            GpioIntRegs.GPIOXINT1SEL.bit.GPIOSEL = 7;   // Xint1 is GPIO7
            XIntruptRegs.XINT1CR.bit.POLARITY = 3;      // Falling edge and rising edge interrupt

            //GPIO4 is used for Q1/Q2 while GPIO5 is used for Q3/Q4
            GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Clear output latch
            GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;   // Enable pullup on GPIO4
            GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 0;  // GPIO4 = GPIO4
            GpioCtrlRegs.GPADIR.bit.GPIO4 = 1;   // GPIO4 = output

            GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Clear output latch
            GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;   // Enable pullup on GPIO5
            GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 0;  // GPIO5 = GPIO5
            GpioCtrlRegs.GPADIR.bit.GPIO5 = 1;   // GPIO5 = output

            EDIS;
}

void init_tmr0(void)
{
            //Step 4. Initialize the Device Peripheral. This function can be
            //        found in DSP2833x_CpuTimers.c
            InitCpuTimers();   // For this example, only initialize the Cpu Timers
            // Configure CPU-Timer 0, 1, and 2 to interrupt every second:
            // 150MHz CPU Freq, 1 second Period (in uSeconds)

            ConfigCpuTimer(&CpuTimer0, 150, 14.286); //70kHz timer
            //ConfigCpuTimer(&CpuTimer0, 150, 19.995); //50kHz timer
            //ConfigCpuTimer(&CpuTimer0, 150, 33.3333);
}
```

- ***Interrupt sub-routines***

```
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File
#include "IQmathLib.h"
#include "defines.h"
#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"   // DSP2833x Examples Include File
#include "DSP2833x_SWPrioritizedIsrLevels.h"
#include "SFO_V5.h"               // SFO V5 library headerfile - required to use SFO library functions

//All variables used in this program.
typedef struct ADC_structure{
            Uint16 mean0;
            Uint16 mean1;
            Uint16 mean2;
            Uint16 mean3;
            Uint16 mean4;
            Uint16 mean5;
            Uint16 mean6;
            Uint16 mean7;
```

```
        Uint16 mean8;
        Uint16 mean9;
        Uint16 buffer0[2], buffer1[2], buffer2[2], buffer3[2], buffer4[2],
        buffer5[2], buffer6[2], buffer7[2], buffer8[2], buffer9[2]; //Buffers for ADC, will be used to compute mean
        Uint16 timer;
} ADC_structure;

typedef struct current_controller_structure{
        _iq16 error_IQ16[2]; //Controller error and one history term
        _iq16 duty_KI_IQ16[2]; //Duty cycle from integral contribution and one history term
        _iq16 duty_KP_IQ16; //Duty cycle from proportional contribution
        _iq16 duty_lag_IQ16[2]; //Duty cycle from the lag term and one history term
        _iq16 duty_PI_IQ16; //Duty cycle from the PI controller (sum of Kp and Ki duty cycle terms)
        _iq16 duty_decoupled_IQ16; //Decoupled duty cycle
        _iq16 duty_total_IQ16[2]; //Total duty cycle from the sum of de-coupled and controller terms with one history term
        _iq16 lag_a_IQ16; //Lag coefficients
        _iq16 lag_b_IQ16;
        _iq16 KP_IQ16;        //Controller proportional gain
        _iq16 KI_z_IQ16; //Controller integral gain times Tsamp/2 * 2^16
        _iq16 I_grid_reference_folded_IQ16; //Controller reference current in "folded" form
        _iq16 I_grid_folded_IQ16; //Measured grid current in "folded" form
        int duty_DSP; //Duty cycle in DSP units to be fed to PWM Counters.
        long duty_frac; //Fractional duty cycle.
        char V_GRID_POLARITY; //This is used to determine the sign of the "folded" grid reference and measured currents
        char DEAD_TIME;   //The current controller's own dead time shadow variable
        char CURRENT_CONTROLLER_ACTIVATED;
} current_controller_structure;

typedef struct error_structure{
        char I_GRID_OVR_CURRENT;
        char V_INV_OVR_VOLTAGE;

        char V_GRID_OVR_VOLTAGE;
        char V_GRID_OVR_VOLTAGE_SHDW;
        char V_GRID_UNDR_VOLTAGE;
        char V_GRID_UNDR_VOLTAGE_SHDW;

        char V_GRID_WRONG_POLARITY;

        char V_PV_OVR_VOLTAGE;
        char V_PV_OVR_VOLTAGE_SHDW;
        char V_PV_UNDR_VOLTAGE;
        char V_PV_UNDR_VOLTAGE_SHDW;

        char I_PV_OVR_CURRENT;
        char TEMP1_HIGH;
        char TEMP2_HIGH;
        int v_grid_ovr_voltage_timer, v_grid_undr_voltage_timer;
        int v_pv_ovr_voltage_timer, v_pv_undr_voltage_timer;

        char FATAL_ERROR;
        char STOP;
} error_structure;

typedef struct PLL_structure{
        _iq20 Kp; //PLL proportional
        _iq20 Ki; //PLL integral term (Ki*Ts/2)
        _iq20 Ki2; //(VCO) Integrator coefficient for computing the grid angle (Ts/2) in iq20 format
        _iq20 cos_theta; //cos_theta
        _iq20 sin_theta; //sin_theta
        _iq20 w[3]; //This is the angular velocity from the PLL alongside two history term
        _iq20 w_filtered[3]; //This is the filtered angular velocity along with two history terms
        _iq20 w_non_offset_P; //This is the proportional component of the unfiltered output of the PLL's PI controller.
        _iq20 w_non_offset_I[2]; //This is the integral component of the unfiltered output of the PLL's PI controller along with one history
term
        _iq20 w_offset; //This is the offset w that will be added (feedforward) to improve response of the PLL and reduce controller effort
        _iq20 theta[2]; //Angle in radians at output of VCO, with one history term
        _iq20 SOGI_Valpha[3]; //This is the alpha voltage from the second order generalized integrator
        _iq20 SOGI_Vbeta[3]; //This is the beta voltage from the second order generalized integrator
        _iq20 SOGI_k; //SOGI k coefficient
```

```
        _iq20 SOGI_wnTs;
        _iq20 SOGI_x;
        _iq20 SOGI_y;
        _iq20 SOGI_denominator;
        _iq20 SOGI_quotient;
        _iq20 SOGI_b0;
        _iq20 SOGI_a1;
        _iq20 SOGI_a2;
        _iq20 SOGI_ky;
        _iq20 SOGI_qb0;
        _iq20 V_grid_pu[3]; //Normalized (per-unitized) instantaneous grid voltage with two history terms;
        _iq20 V_grid_base;    //Base grid voltage
        _iq20 error[3];//Present error term and two history terms
        _iq20 notch_a; //Notch filter coefficient a(0.979127)
        _iq20 notch_b; //Notch filter coefficient b(-1.957364)
        _iq20 notch_f; //Notch filter coefficient f(0.958254)
        _iq20 notch_out[3]; //Output of notch filter with two history terms.
        _iq20 Vd; //Direct-axis voltage
        _iq20 Vq; //Quadrature-axis voltage
        _iq20 V_grid_pk_pu[2]; //Normalized (per-unitized) unfiltered peak grid voltage
        _iq20 V_grid_pk_filtered_pu[2]; //Normalized filtered peak grid voltage
        _iq20 A,B,C,D,E; //Variables that will be used as coefficients for filtering the angular speed and the peak normalized grid voltage.
Can also be constants in IQ20 format
        char PERIOD_HAS_OCCURED; //Takes note of when one grid period has occurred
        char HAS_STARTED;
        char V_GRID_POLARITY; //Grid polarity as given by PLL module
        char DEAD_TIME; //Dead time as given  by PLL module
/*
        char LOCKED; //This variable is set when it is determined that the PLL is 'locked' (when frequency varies within a narrowband)
        char FAILED; //This variable is set when it is determined that the PLL has failed to lock after several attempts.
        Uint16 lock_timer; //Timer used to keep track of PLL Locked status
*/
}PLL_structure;

typedef struct zcd_structure{
        Uint16 half_period[2]; //Grid half period
        Uint16 full_period; //Grid full period
        Uint16 half_period_counter; //Counter for determining the half_period
        Uint16 full_period_counter; //Counter for determining grid angle from 0 to 360 degrees
        Uint16 angle_pu_IQ16; //Grid angle measurement by using the hardware zcd method.
        char V_GRID_POLARITY; //Grid polarity as given by zcd circuit.
        char PERIOD_TOO_LOW; //High frequency limit as given by zcd module
        char PERIOD_TOO_HIGH; //Low frequency limit as given by zcd module
        char DEAD_TIME; //Dead time event indicator
} zcd_structure;

typedef struct power_control_structure{
        _iq16 P_ref_IQ16;     //Reference active power in IQ16 format in IQ16 format.
        _iq16 Q_ref_IQ16;     //Reference reactive power in IQ16 format in IQ16 format.
        _iq16 error_P_ref[2]; //Active power tracking error
        _iq16 error_Q_ref[2]; //Reactive power tracking error
        _iq16 P_dc_IQ16;      //Active power computed by P-Q transform method in IQ16 format.
        _iq16 Q_dc_IQ16;      //Reactive power computed by P-Q transform method in IQ16 format.
        _iq16 P_input_avg_IQ16;          //Average input power. Computed by integration. IQ16 format
        _iq16 P_output_avg_IQ16;         //Average output power. Computed by integration. IQ16 format
        _iq16 I_grid_reference_peak_IQ16; //Peak reference grid current from power loop. In IQ16 format.
        _iq16 Kp; //Proportional gain to be used for power control loop, in IQ16 format
        _iq16 Kiz; //(Integrator gain)*(Ts/2) integrator gain for power control loop in IQ16 format
        _iq16 inst_input_power_integrator; //Sums the instantaneous input power at regular sample intervals.
        _iq16 inst_output_power_integrator; //Sums the instantaneous output power at regular sample intervals.
        Uint16 timer; //Control loop timer (approximates the sampling period).
        Uint16 avg_power_counter; //Counter to be used in computing the average power by integration method
} power_control_structure;

//All variables used in this program.
extern _iq16 I_pv_IQ16; //PV current
extern _iq16 V_pv_IQ16; //PV voltage
extern _iq16 V_grid_IQ16; //Grid voltage
extern _iq16 V_inv_IQ16; //Inverter pseudo-dc link voltage
extern _iq16 I_grid_IQ16; //Grid-injected current
```

```
extern _iq16 I_grid_reference_IQ16; //amperes
extern _iq16 I_grid_reference_peak_IQ16; //Peak reference current
extern _iq16 I_grid_angle_offset_IQ16; //Offset grid reference angle
//extern _iq16 angle_pu_IQ16;
extern _iq16 dummy1_IQ16;
extern _iq16 dummy2_IQ16;
extern _iq16 dummy3_IQ16; //dummy variables for computations
extern _iq16 V_grid_pk_IQ16[2]; //Instantaneous peak grid voltage container and one history term
extern char V_grid_polarity; //Grid voltage polarity
extern struct current_controller_structure cc; //Create an instance of a current controller
extern struct error_structure ERROR_STATUS; //Create an instance of an error structure
extern struct ADC_structure ADC; //Create an instance of an ADC structure
extern struct PLL_structure PLL; //Create an instance of a PLL structure
extern struct zcd_structure zcd; //Create an instance of a zcd structure
extern struct power_control_structure power_loop; //Create an instance of a power control structure
extern int debug1[200];
extern int debug2[200];
extern int dummy_counter;
extern int j;
extern int m;
extern Uint16 start_timer;
extern char CONVERTER_STARTED;
extern char SOURCE_OF_UNFOLDING_SIGNAL;

extern int MEP_ScaleFactor[7]; // Global array used by the SFO library. Only HRPWM1A will be used to compute scale factor. So it's HRPWM
must be disabled
extern volatile struct EPWM_REGS *ePWM[7]; //
extern int SFO_status;

_iq16 dummy_reference_IQ16 = 0; //For debugging the positive offset current in the output

interrupt void xint1_isr(void)
{
        // Set interrupt priority:
        volatile Uint16 TempPIEIER = PieCtrlRegs.PIEIER1.all;
        IER |= M_INT1;
        IER     &= MINT1;                          // Set "global" priority
        PieCtrlRegs.PIEIER1.all &= MG14;   // Set "group"  priority
        PieCtrlRegs.PIEACK.all = 0xFFFF;   // Enable PIE interrupts
        EINT;

        //Insert ISR code here
        if (zcd.half_period_counter > 208) //This is a 'small' filter to ignore false zero-crossings
        {
                //GpioDataRegs.GPASET.bit.GPIO11 = 1;   // Turn on pin for debugging purposes

                if (GpioDataRegs.GPADAT.bit.GPIO7)
                {
                        zcd.V_GRID_POLARITY = 1; //'1' for positive grid voltage; else '0'
                        zcd.full_period_counter = 0; //Reset the zcd full period counter on every positive half cycle (grid angle starts
counting from positive grid voltage)
                }
                else
                {
                        zcd.V_GRID_POLARITY = 0; //'1' for positive grid voltage; else '0'
                }

                if (SOURCE_OF_UNFOLDING_SIGNAL == HARDWARE_ZCD) //Only control switches if the source of the unfolding
signal is from the hardware zcd.
                {
                        GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // First turn off all switches in unfolder at zero-crossing to prevent
shoot-through
                        GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;
                        EALLOW;
                        EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                        EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                        EDIS;

                        cc.V_GRID_POLARITY = zcd.V_GRID_POLARITY; //Update current controller zero crossing info
```

```
                        if ((zcd.V_GRID_POLARITY) && (CONVERTER_STARTED)) //If positive grid voltage and if converter has started
                        {
                                GpioDataRegs.GPASET.bit.GPIO4 = 1;   // Turn on U1/U2 for positive grid voltage
                        }
                        else if (CONVERTER_STARTED)//If grid voltage is negative and converter has started.
                        {
                                GpioDataRegs.GPASET.bit.GPIO5 = 1;   // Turn on U3/U4 for negative grid voltage
                        }
                        //zcd.DEAD_TIME = 0; //Dead time period is de-activated. Must be activated by dead-time checker in fast 125kHz ADC routine
                }

                zcd.half_period[0] = zcd.half_period_counter; //Update half-period info.
                zcd.half_period_counter = 0; //Reset half_period_counter
                zcd.full_period = zcd.half_period[0] + zcd.half_period[1]; //Update zcd.full_period
                zcd.half_period[1] = zcd.half_period[0]; //update history term of half_period

                //Reset peak instantaneous grid voltage detector
                V_grid_pk_IQ16[0] = V_grid_pk_IQ16[1];
                V_grid_pk_IQ16[1] = 0;

                //Check for zcd.full_period error
                if ((zcd.full_period > ZCD_HIGH_PERIOD)&&(CONVERTER_STARTED)) //Corresponds to 45Hz //1111
                {
                        zcd.PERIOD_TOO_HIGH = 1;
                        ERROR_STATUS.FATAL_ERROR = 1; //This fatal error is only acknowledged when converter has started
                }

                if ((zcd.full_period < ZCD_LOW_PERIOD)&&(CONVERTER_STARTED)) //Corresponds to 65Hz //1923
                {
                        zcd.PERIOD_TOO_LOW = 1;
                        ERROR_STATUS.FATAL_ERROR = 1; //This fatal error is only acknowledged when converter has started
                }

                //Update reference active and reactive power inputs
                power_loop.P_ref_IQ16 = ((110*(long)ADC.mean6) << 4); //110watts*POTENTIOMETER/4096
                power_loop.Q_ref_IQ16 = ((110*(long)ADC.mean9) << 4); //110VAR*POTENTIOMETER/4096

                I_grid_reference_peak_IQ16 = ((2*(long)ADC.mean6) << 4);
                I_grid_angle_offset_IQ16 = ((_IQ10(0.25)*(long)ADC.mean9) >> 6);
                //Soft-turn off
                if (ERROR_STATUS.STOP) //(GpioDataRegs.GPADAT.bit.GPIO10 == 1)) //Stop converter on push of button
                {
                        GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   //Turn off all switches
                        GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;
                        CONVERTER_STARTED = 0; //Turn off converter
                        EALLOW;
                        EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                        EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                        EDIS;
                        cc.CURRENT_CONTROLLER_ACTIVATED = 0;
                        GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;   // Clear output latch of debug pin
                }
        }

        // Acknowledge this interrupt to get more from group 1
        //PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
        // Restore registers saved:

        DINT;
        PieCtrlRegs.PIEIER1.all = TempPIEIER;
}

interrupt void cpu_timer0_isr(void)
{
        // Set interrupt priority:
        volatile Uint16 TempPIEIER = PieCtrlRegs.PIEIER1.all;
        IER |= M_INT1;
        IER     &= MINT1;                               // Set "global" priority
```

```
PieCtrlRegs.PIEIER1.all &= MG17;   // Set "group"  priority
PieCtrlRegs.PIEACK.all = 0xFFFF;   // Enable PIE interrupts
EINT;

GpioDataRegs.GPASET.bit.GPIO10 = 1;   // Set debugging pin high


//Convert measured quantities to SI units by making use of calibration data
I_pv_IQ16 = (long)k_I_pv*ADC.mean0 + c_I_pv; //Compute PV current
V_pv_IQ16 = (long)k_V_pv*ADC.mean1 + c_V_pv; //Compute PV voltage
I_grid_IQ16 = (long)k_I_grid*ADC.mean2 + c_I_grid; //Compute grid-injected current
V_grid_IQ16 = (long)k_V_grid*ADC.mean3 + c_V_grid; //Compute grid voltage
V_inv_IQ16 = (long)k_V_inv*ADC.mean5 + c_V_inv; //Compute inverter output voltage
dummy3_IQ16 = _IQ16abs(V_grid_IQ16);

//Increment zcd.half_period_counter and full period counter;
zcd.half_period_counter = zcd.half_period_counter + 1;
zcd.full_period_counter = zcd.full_period_counter + 1;

//Implement a dead time period to prevent shoot through at unfolder
if ((CONVERTER_STARTED) && (SOURCE_OF_UNFOLDING_SIGNAL == HARDWARE_ZCD))
{
        if (zcd.half_period_counter >= (zcd.half_period[0] - 4)) //Gives a dead band of 32us before the end of the half period (if
half_period is correct, this will give a dead time just before the zero-crossing)
        {
/*
                GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off unfolder U1/U2
                GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off unfolder U3/U4


                zcd.DEAD_TIME = 1; //Dead time period is activated. Must be de-activated by zero-crossing event
                EALLOW;
                EPwm2Regs.TZFRC.bit.OST = 1; ////Force one-shot trip condition in order to turn off PWM during this period.
Must be cleared by current controller routine
                EDIS;*/
        }
        else //If dead time is not activated, then make sure unfolder is properly turned on
        {
/*              zcd.DEAD_TIME = 0; //Dead time period is de-activated.
                if (zcd.V_GRID_POLARITY)
                {
                        GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off unfolder U3/U4
                        GpioDataRegs.GPASET.bit.GPIO4 = 1;   // Turn on U1/U2
                }
                else
                {
                        GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off U1/U2
                        GpioDataRegs.GPASET.bit.GPIO5 = 1;   // Turn on U3/U4
                }*/
        }
        cc.V_GRID_POLARITY = zcd.V_GRID_POLARITY; //Update current controller polarity and dead time shadow info
        cc.DEAD_TIME = zcd.DEAD_TIME;
}

//Implement software-emulated zcd by checking phase angle (will cause trouble if PLL is not locked).
if ((CONVERTER_STARTED) && (SOURCE_OF_UNFOLDING_SIGNAL == SOFTWARE_ZCD))
{
        //turn-off U3/U4
        if (PLL.theta[0] >= U3_U4_DEAD_TIME_ANGLE_360_MINUS_IQ20)
        {
                GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off U3/U4
                EALLOW;
                EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
                EDIS;
                PLL.DEAD_TIME = 1;
        }

        //turn-on U1/U2
```

```
			if ((PLL.theta[0] > U1_U2_DEAD_TIME_ANGLE_ZERO_MINUS_IQ20) || ((PLL.theta[0] >=
U1_U2_DEAD_TIME_ANGLE_ZERO_PLUS_IQ20) && (PLL.theta[0] < U1_U2_DEAD_TIME_ANGLE_180_MINUS_IQ20)))
			{
				GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off U3/U4
				GpioDataRegs.GPASET.bit.GPIO4 = 1;   // Turn on U1/U2
				PLL.DEAD_TIME = 0;
			}

			//turn-off U1/U2
			if ((PLL.theta[0] >= U1_U2_DEAD_TIME_ANGLE_180_MINUS_IQ20)&&(PLL.theta[0] <=
U1_U2_DEAD_TIME_ANGLE_ZERO_MINUS_IQ20))
			{
				GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off U1/U2
				EALLOW;
				EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
				EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
				EDIS;
				PLL.DEAD_TIME = 1;
			}

			//turn-on U3/U4
			if ((PLL.theta[0] >= U3_U4_DEAD_TIME_ANGLE_180_MINUS_IQ20) && (PLL.theta[0] <
U3_U4_DEAD_TIME_ANGLE_360_MINUS_IQ20))
			{
				GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off U1/U2
				GpioDataRegs.GPASET.bit.GPIO5 = 1;   // Turn on U3/U4
				PLL.DEAD_TIME = 0;
			}
			cc.V_GRID_POLARITY = PLL.V_GRID_POLARITY;
			cc.DEAD_TIME = PLL.DEAD_TIME;
		}

		zcd.angle_pu_IQ16 = _IQ16div(zcd.full_period_counter,zcd.full_period) - I_grid_angle_offset_IQ16;//This calculates the angle in
per_units as given by zcd
		if (zcd.angle_pu_IQ16 > 65535) //If per unit angle is greater than 1, then reset to zero.
		{
			zcd.angle_pu_IQ16 = 0;
		}


		//Implement Controller
		I_grid_reference_IQ16 = _IQ16mpy(I_grid_reference_peak_IQ16, _IQ16sinPU(zcd.angle_pu_IQ16)); //This computes instantaneous
reference current using half-cycle periodicity. Comment out if using PLL.
		//I_grid_reference_IQ16 = _IQ16mpy(I_grid_reference_peak_IQ16,(PLL.sin_theta>>4)); //Compute instantaneous reference current


		if (cc.V_GRID_POLARITY)
		{
			cc.I_grid_folded_IQ16 = I_grid_IQ16;
			cc.I_grid_reference_folded_IQ16 = I_grid_reference_IQ16;
		}
		else //Flip error signal when in negative half cycle if using the full period reference current method (e.g. PLL)
		{
			dummy_reference_IQ16 = _IQ16mpy(I_grid_reference_peak_IQ16,_IQ16(1.12)); //Offset at negative half cycle
			I_grid_reference_IQ16 = _IQ16mpy(dummy_reference_IQ16,(PLL.sin_theta>>4)); //Compute instantaneous reference
current

			cc.I_grid_folded_IQ16 = -I_grid_IQ16;
			cc.I_grid_reference_folded_IQ16 = -I_grid_reference_IQ16;
		}

		//cc.error_IQ16[0] = I_grid_reference_IQ16 - I_grid_IQ16;
		cc.error_IQ16[0] = cc.I_grid_reference_folded_IQ16 - cc.I_grid_folded_IQ16;			//Compute the error in the grid
current
		if (_IQ16abs(I_grid_reference_IQ16) <= _IQ16(0.9))
		{
			cc.duty_KI_IQ16[0] = _IQ16mpy(cc.KI_z_IQ16,(cc.error_IQ16[0] + cc.error_IQ16[1])) + cc.duty_KI_IQ16[1];
		//Compute the result from the integrator term
			cc.duty_KP_IQ16 = _IQ16mpy(cc.KP_IQ16,(cc.error_IQ16[0]));   //Compute the result from the proportional term
			cc.duty_PI_IQ16 = cc.duty_KI_IQ16[0] + cc.duty_KP_IQ16;      //Total duty cycle for PI controller
```

```
}
else
{
        cc.duty_KI_IQ16[0] = _IQ16mpy((cc.KI_z_IQ16),(cc.error_IQ16[0] + cc.error_IQ16[1])) + cc.duty_KI_IQ16[1];
//Compute the result from the integrator term
        cc.duty_KP_IQ16 = _IQ16mpy((cc.KP_IQ16),(cc.error_IQ16[0]));//Compute the result from the proportional term. Divide
proportional term by 2
        cc.duty_PI_IQ16 = cc.duty_KI_IQ16[0] + cc.duty_KP_IQ16;        //Total duty cycle for PI controller
}

//Decoupled duty cycle implementation
//dummy3_IQ16 = _IQ16abs(V_grid_IQ16);
if (V_pv_IQ16 <= 65536) //This avoids division by zero
{
        V_pv_IQ16 = 65536;
        ERROR_STATUS.FATAL_ERROR = 1;
}
cc.duty_decoupled_IQ16 =  _IQ16div(dummy3_IQ16,(dummy3_IQ16 + (long)TRANSFORMER_TURNS_RATIO*V_pv_IQ16));

//Sum of controller duty cycle and decoupled duty cycle
cc.duty_total_IQ16[0] = cc.duty_PI_IQ16 + cc.duty_decoupled_IQ16;          //This is the sum of controller duty cycle and decoupled
duty cycle

//Implement lag term
//cc.duty_lag_IQ16[0] = -_IQ16mpy(cc.lag_b_IQ16,cc.duty_lag_IQ16[1]) + _IQ16mpy(cc.lag_a_IQ16,(cc.duty_total_IQ16[0] +
cc.duty_total_IQ16[1]));
//cc.duty_DSP = _IQ16int(cc.duty_lag_IQ16[0]*(long)SWITCHING_PERIOD); //Multiply by switching period (divide by frequency)
to get the duty cycle (in DSP units*2^16) then convert to integer format
//cc.duty_DSP = _IQ16int(cc.duty_total_IQ16[0]*(long)SWITCHING_PERIOD);
cc.duty_DSP = ((long)cc.duty_total_IQ16[0]*(long)SWITCHING_PERIOD) >> 16; //This gets the integral (whole) part of the duty
cycle in integer format
cc.duty_frac = ((long)cc.duty_total_IQ16[0]*(long)SWITCHING_PERIOD) - ((long)cc.duty_DSP << 16); //This gets the fractional
part of the duty cycle in IQ16 format
cc.duty_frac = (((long)cc.duty_frac*MEP_ScaleFactor[1]) >> 8) + 0x0180; //This gets the fractional duty cycle

//Controller saturation and anti-wind up
if (cc.duty_DSP > DUTY_CYCLE_MAX)
{
        cc.duty_DSP = DUTY_CYCLE_MAX;
}
else if (cc.duty_DSP < DUTY_CYCLE_MIN)
{
        cc.duty_DSP = DUTY_CYCLE_MIN;
}
else
{
        cc.error_IQ16[1] = cc.error_IQ16[0];
        cc.duty_KI_IQ16[1] = cc.duty_KI_IQ16[0];
        cc.duty_total_IQ16[1] = cc.duty_total_IQ16[0];
        cc.duty_lag_IQ16[1] = cc.duty_lag_IQ16[0];
}
if (cc.duty_frac < 0)
{
        cc.duty_frac = 0;
}

if ((cc.CURRENT_CONTROLLER_ACTIVATED))//&&(cc.DEAD_TIME == 0))
{
        EPwm1Regs.CMPA.half.CMPA = cc.duty_DSP;        // Update compare A value (PWM Q0 and PWM Qx)
        EPwm2Regs.CMPA.all = ((long)cc.duty_DSP)<<16 | cc.duty_frac; //Update compare A value (PWM Q_main)
        EPwm5Regs.CMPA.all = EPwm2Regs.CMPA.all; //Update compare A value (PWM Q5)

        EALLOW;
        EPwm2Regs.TZCLR.bit.OST = 1; //Clear one-shot trip
        EPwm5Regs.TZCLR.bit.OST = 1; //Clear one-shot trip
        EDIS;
}

//Compute the peak instantaneous grid voltage
if (dummy3_IQ16 > V_grid_pk_IQ16[1])
```

```
		{
			V_grid_pk_IQ16[1] = dummy3_IQ16;
		}

		GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;   // Set debugging pin high

		// Restore registers saved:
		DINT;
		PieCtrlRegs.PIEIER1.all = TempPIEIER;

		//PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

interrupt void epwm4_isr(void)
{
		// Set interrupt priority:
		volatile Uint16 TempPIEIER = PieCtrlRegs.PIEIER3.all;
		IER |= M_INT3;
		IER	&= MINT3;					// Set "global" priority
		PieCtrlRegs.PIEIER3.all &= MG34;   // Set "group"  priority
		PieCtrlRegs.PIEACK.all = 0xFFFF;   // Enable PIE interrupts
		EINT;

		GpioDataRegs.GPASET.bit.GPIO11 = 1;   // Set debugging pin high

		//This "if" section is only for logging some data for debugging purposes
		if (cc.CURRENT_CONTROLLER_ACTIVATED)
		{
			if (m<100)
			{
				if (j > 10)
				{
					debug1[dummy_counter] = (PLL.sin_theta>>10); //(PLL.w[0]>>14);//ADC.mean3;
					debug2[dummy_counter] = (V_grid_IQ16 >> 10);//(V_inv_IQ16>>14);
//(PLL.theta[0]>>15);//(PLL.w_filtered[0]>>14); //(PLL.sin_theta>>14); //ADC.mean9;
					dummy_counter = dummy_counter + 1;
					if (dummy_counter > 199)
					{
						dummy_counter = 0;
						m = m+1;
					}
					j = 0;
				}
				j = j + 1;
			}
			else
			{
				m = 150;
			}
		}
		//end "if"

		start_timer = start_timer + 1;

		//This part of the code should run with a frequency of 25kHz; for example, it can run in the 50kHz sub-routine with a divide-by-2-
counter
		if (PLL.HAS_STARTED)
		{
			PLL.w_offset = PLL.w_filtered[0]; //Attribute the correct offset to reduce controller effort when PLL is running
			//PLL.V_grid_base = _IQ20mpy(PLL.V_grid_pk_filtered_pu[0],PLL.V_grid_base); //Update the base grid voltage based
on measurement from
			//PLL.V_grid_pu[0] = _IQ20div((V_grid_IQ16<<4),PLL.V_grid_base); //Update per-unit voltage based on peak grid
voltage computed by dq method
		}
/*		else
		{
			PLL.V_grid_pu[0] = _IQ20div((V_grid_IQ16<<4),(V_grid_pk_IQ16[0]<<4)); //Update per-unit voltage based on peak
grid voltage detected by ADC method
		}*/
```

```
	PLL.V_grid_pu[0] = _IQ20div((V_grid_IQ16<<4),(V_grid_pk_IQ16[0]<<4)); //Update per-unit voltage based on peak grid voltage detected by ADC method


	//PLL.V_grid_base = V_GRID_BASE_IQ20; //(V_grid_pk_IQ16[0] << 4);	//Update base voltage with peak grid voltage detected by 50kHz sub routine.
				//Right shifting by 4 bits because ADC.V_grid_pk is in IQ16 format and we need IQ20

	//Normalize input voltage measurement
	//PLL.V_grid_pu[0] = _IQ20div((V_grid_IQ16<<4),(long)V_GRID_BASE_IQ20); //Convert normalized value to an IQ20 format
	//PLL.V_grid_pu[0] = _IQ20div((V_grid_IQ16<<4),(V_grid_pk_IQ16[0]<<4)); //Convert normalized value to an IQ20 format

	//Implement phase detection and notch filter
	PLL.error[0] = _IQ20mpy(PLL.V_grid_pu[0],PLL.cos_theta); //Compute present error
	PLL.notch_out[0] = _IQ20mpy(PLL.notch_a, (PLL.error[0] + PLL.error[2])) + _IQ20mpy(PLL.notch_b, (PLL.error[1] - PLL.notch_out[1])) - _IQ20mpy(PLL.notch_f,PLL.notch_out[2]);

	//Implement PI controller
	PLL.w_non_offset_P = _IQ20mpy(PLL.Kp,PLL.notch_out[0]);
	PLL.w_non_offset_I[0] = PLL.w_non_offset_I[1] + _IQ20mpy(PLL.Ki, (PLL.notch_out[1] + PLL.notch_out[0]));
	if (PLL.w_non_offset_I[0] < 0) //Cannot have negative speeds
	{
		PLL.w_non_offset_I[0] = 0;
	}
	PLL.w[0] = PLL.w_non_offset_P + PLL.w_non_offset_I[0] + PLL.w_offset; //This is the actual dynamic unfiltered angular velocity from the PLL

	//Update history terms.
	PLL.error[2] = PLL.error[1];
	PLL.error[1] = PLL.error[0];
	PLL.notch_out[2] = PLL.notch_out[1];
	PLL.notch_out[1] = PLL.notch_out[0];
	PLL.w_non_offset_I[1] = PLL.w_non_offset_I[0];

	//Implement VCO (the integration to calculate the grid angle)
	PLL.theta[0] = PLL.theta[1] + _IQ20mpy(PLL.Ki2,(PLL.w[0] + PLL.w[1])); //This is the grid angle
	//Reset the integrator when angle reaches 2*pi or when angle is less than 0
	if (PLL.theta[0] >= TWO_PI_IQ20)
	{
		PLL.theta[0] = 0;
		PLL.PERIOD_HAS_OCCURED = 1;
		PLL.V_GRID_POLARITY = POSITIVE_POLARITY;
	}
	if (PLL.theta[0] >= PI_IQ20)
	{
		PLL.V_GRID_POLARITY = NEGATIVE_POLARITY;
	}
	if (PLL.theta[0] < 0)
	{
		PLL.theta[0] = 0;
	}
	//Update VCO history terms
	PLL.theta[1] = PLL.theta[0];
	//PLL.w[1] = PLL.w[0];

	//Compute sin_theta and cos_theta
	PLL.cos_theta = _IQ20cos(PLL.theta[0]);
	PLL.sin_theta = _IQ20sin(PLL.theta[0]);

	//Compute PLL.w_filtered
	//PLL.w_filtered[0] = _IQ20mpy(PLL.A,(PLL.w[0] + 2*PLL.w[1] + PLL.w[2])) - _IQ20mpy(PLL.B,PLL.w_filtered[1]) - _IQ20mpy(PLL.C,PLL.w_filtered[2]); //Second order filter with 10Hz cut-off
	PLL.w_filtered[0] = _IQ20mpy(PLL.w_filtered[1],PLL.E) + _IQ20mpy(PLL.D,(PLL.w[0] + PLL.w[1])); //First order filter with 10Hz cut-off

	//Update history terms
	PLL.w[2] = PLL.w[1];
	PLL.w[1] = PLL.w[0];
	PLL.w_filtered[2] = PLL.w_filtered[1];
	PLL.w_filtered[1] = PLL.w_filtered[0];
```

```
        //Compute PLL.SOGI_Valpha and PLL.SOGI_Vbeta through the SOGI
        PLL.SOGI_wnTs = _IQ20mpy(42,PLL.w_offset); //Ts*wn in _iq20
        PLL.SOGI_x = 2*_IQ20mpy(PLL.SOGI_wnTs, PLL.SOGI_k);
        PLL.SOGI_y = _IQ20mpy(PLL.SOGI_wnTs, PLL.SOGI_wnTs);
        PLL.SOGI_denominator = PLL.SOGI_x + PLL.SOGI_y + _IQ20(4); //Calculate the denominator (x + y + 4) only once
        PLL.SOGI_quotient = _IQ20div(_IQ20(1),PLL.SOGI_denominator); //Compute the quotient 1/(x+y+4) only once because
multiplications are more efficient than divisions.
        PLL.SOGI_b0 = _IQ20mpy(PLL.SOGI_x,PLL.SOGI_quotient);
        PLL.SOGI_a1 = 2*_IQ20mpy((_IQ20(4) - PLL.SOGI_y),PLL.SOGI_quotient);
        PLL.SOGI_a2 = _IQ20mpy((PLL.SOGI_x - PLL.SOGI_y - _IQ20(4)),PLL.SOGI_quotient);
        PLL.SOGI_ky = _IQ20mpy(PLL.SOGI_k,PLL.SOGI_y);
        PLL.SOGI_qb0 = _IQ20mpy(PLL.SOGI_ky,PLL.SOGI_quotient);
        PLL.SOGI_Valpha[0] = _IQ20mpy(PLL.SOGI_a1,PLL.SOGI_Valpha[1]) + _IQ20mpy(PLL.SOGI_a2,PLL.SOGI_Valpha[2]) +
_IQ20mpy(PLL.SOGI_b0,(PLL.V_grid_pu[0] - PLL.V_grid_pu[2]));
        PLL.SOGI_Vbeta[0] = _IQ20mpy(PLL.SOGI_a1, PLL.SOGI_Vbeta[1]) + _IQ20mpy(PLL.SOGI_a2,PLL.SOGI_Vbeta[2]) +
_IQ20mpy(PLL.SOGI_qb0,(PLL.V_grid_pu[0] + 2*PLL.V_grid_pu[1] + PLL.V_grid_pu[2]));

        //Update history terms
        PLL.SOGI_Valpha[2] = PLL.SOGI_Valpha[1];
        PLL.SOGI_Valpha[1] = PLL.SOGI_Valpha[0];
        PLL.SOGI_Vbeta[2] = PLL.SOGI_Vbeta[1];
        PLL.SOGI_Vbeta[1] = PLL.SOGI_Vbeta[0];
        PLL.V_grid_pu[2] = PLL.V_grid_pu[1];
        PLL.V_grid_pu[1] = PLL.V_grid_pu[0];


        //Compute PLL.Vd and PLL.Vq through a Park Transform
        PLL.Vd = _IQ20mpy(PLL.cos_theta,PLL.SOGI_Valpha[0]) + _IQ20mpy(PLL.sin_theta,PLL.SOGI_Vbeta[0]);
        PLL.Vq = _IQ20mpy(-PLL.sin_theta,PLL.SOGI_Valpha[0]) + _IQ20mpy(PLL.cos_theta,PLL.SOGI_Vbeta[0]);

        //Compute the Peak grid voltage and filter it
        PLL.V_grid_pk_pu[0] = _IQ20mag(PLL.Vd,PLL.Vq); //Unfiltered peak pu voltage
        //Filter it
        PLL.V_grid_pk_filtered_pu[0] = _IQ20mpy(PLL.V_grid_pk_filtered_pu[1],PLL.E) + _IQ20mpy(PLL.D,(PLL.V_grid_pk_pu[0] +
PLL.V_grid_pk_pu[1])); //Filtered peak pu voltage
        //Update history terms
        PLL.V_grid_pk_pu[1] = PLL.V_grid_pk_pu[0];
        PLL.V_grid_pk_filtered_pu[1] = PLL.V_grid_pk_filtered_pu[0];

        //End of PLL functions.


        //Implement the power control loop algorithm.        //Compute the average power (active power) by direct integration and
averaging

        power_loop.timer = power_loop.timer + 1;
        power_loop.inst_output_power_integrator = power_loop.inst_output_power_integrator + _IQ16mpy(V_grid_IQ16,I_grid_IQ16)>>8;
        power_loop.inst_input_power_integrator = power_loop.inst_input_power_integrator + _IQ16mpy(V_pv_IQ16,I_pv_IQ16)>>8;
        if (PLL.PERIOD_HAS_OCCURED)
        {
                power_loop.P_output_avg_IQ16 = _IQ16div(power_loop.inst_output_power_integrator,_IQ16(power_loop.timer));
//Average the output power integration
                power_loop.P_input_avg_IQ16 = _IQ16div(power_loop.inst_input_power_integrator,_IQ16(power_loop.timer));
//Average the input power integration

                PLL.PERIOD_HAS_OCCURED = 0; //Reset PLL period indicator.
                power_loop.inst_output_power_integrator = 0; //Reset power integrator and average power counter
                power_loop.inst_input_power_integrator = 0; //Reset power integrator and average power counter
                power_loop.timer = 0;
        }
        //End of average power computation

        GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;   // Clear debugging pin

  // Clear INT flag for this timer
  EPwm4Regs.ETCLR.bit.INT = 1;

        // Restore registers saved:
        DINT;
```

```
		PieCtrlRegs.PIEIER3.all = TempPIEIER;

	// Acknowledge this interrupt to receive more interrupts from group 3
	//PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}


interrupt void  adc_isr(void)
{
		// Set interrupt priority:
		volatile Uint16 TempPIEIER = PieCtrlRegs.PIEIER1.all;
		IER |= M_INT1;
		IER		&= MINT1;			// Set "global" priority
		//PieCtrlRegs.PIEIER1.all &= MG16;   // Set "group"  priority
		PieCtrlRegs.PIEIER1.all &= MG11;   // Set "group"  priority
		PieCtrlRegs.PIEACK.all = 0xFFFF;   // Enable PIE interrupts
		EINT;

		GpioDataRegs.GPASET.bit.GPIO13 = 1;   // Set debugging pin high

		ADC.buffer0[1] = (AdcRegs.ADCRESULT0>>4); //Ipv
		ADC.buffer1[1] = (AdcRegs.ADCRESULT1>>4); //Vpv
		ADC.buffer2[1] = (AdcRegs.ADCRESULT2>>4); //Igrid
		ADC.buffer3[1] = (AdcRegs.ADCRESULT3>>4); //Vgrid
		ADC.buffer4[1] = (AdcRegs.ADCRESULT4>>4); //TEMP1
		ADC.buffer5[1] = (AdcRegs.ADCRESULT5>>4); //Vinv
		ADC.buffer6[1] = (AdcRegs.ADCRESULT6>>4); //POT1
		ADC.buffer7[1] = (AdcRegs.ADCRESULT7>>4); //TEMP2
		ADC.buffer8[1] = (AdcRegs.ADCRESULT8>>4); //undefined
		ADC.buffer9[1] = (AdcRegs.ADCRESULT9>>4); //POT2

		//Check for fatal errors
		if ((ADC.buffer0[1] <= FATAL_ADC_I_PV_MAX_POSITIVE) || (ADC.buffer0[1] >= FATAL_ADC_I_PV_MAX_NEGATIVE))
		{
			GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off unfolder U1/U2
			GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off unfolder U3/U4
			EALLOW;
			EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EDIS;
			ERROR_STATUS.I_PV_OVR_CURRENT = 1;
			ERROR_STATUS.FATAL_ERROR = 1;
			CONVERTER_STARTED = 0;
			cc.CURRENT_CONTROLLER_ACTIVATED = 0;
		}

		if ((ADC.buffer1[1] <= FATAL_ADC_V_PV_MAX))
		{
			GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off unfolder U1/U2
			GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off unfolder U3/U4
			EALLOW;
			EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EDIS;
			ERROR_STATUS.V_PV_OVR_VOLTAGE = 1;
			ERROR_STATUS.FATAL_ERROR = 1;
			CONVERTER_STARTED = 0;
			cc.CURRENT_CONTROLLER_ACTIVATED = 0;
		}

		if ((ADC.buffer2[1] >= FATAL_ADC_I_GRID_MAX_POSITIVE) || (ADC.buffer2[1] <=
FATAL_ADC_I_GRID_MAX_NEGATIVE))
		{
			GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;   // Turn off unfolder U1/U2
			GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;   // Turn off unfolder U3/U4
			EALLOW;
			EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EDIS;
			ERROR_STATUS.I_GRID_OVR_CURRENT = 1;
```

```
			ERROR_STATUS.FATAL_ERROR = 1;
			CONVERTER_STARTED = 0;
			cc.CURRENT_CONTROLLER_ACTIVATED = 0;
		}

		if ((ADC.buffer3[1] >= FATAL_ADC_V_GRID_MAX_POSITIVE) || (ADC.buffer3[1] <=
FATAL_ADC_V_GRID_MAX_NEGATIVE))
		{
			GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;	// Turn off unfolder U1/U2
			GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;	// Turn off unfolder U3/U4
			EALLOW;
			EPwm2Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EPwm5Regs.TZFRC.bit.OST = 1; //Force one-shot trip condition
			EDIS;
			ERROR_STATUS.V_GRID_OVR_VOLTAGE = 1;
			ERROR_STATUS.FATAL_ERROR = 1;
			CONVERTER_STARTED = 0;
			cc.CURRENT_CONTROLLER_ACTIVATED = 0;
		}

		//Compute the mean of selected ADC inputs
		ADC.mean0 = ADC.buffer0[1];
		ADC.mean1 = ADC.buffer1[1];
		ADC.mean2 = ADC.buffer2[1];
		ADC.mean3 = ADC.buffer3[1];
		ADC.mean4 = ADC.buffer4[1];
		ADC.mean5 = ADC.buffer5[1];
		ADC.mean6 = ADC.buffer6[1];
		ADC.mean7 = ADC.buffer7[1];
		ADC.mean8 = ADC.buffer8[1];
		ADC.mean9 = ADC.buffer9[1];

		GpioDataRegs.GPACLEAR.bit.GPIO13 = 1;	// Set debugging pin low
		// Reinitialize for next ADC sequence
		AdcRegs.ADCTRL2.bit.RST_SEQ1 = 1;		// Reset SEQ1
		AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;		// Clear INT SEQ1 bit

		// Restore registers saved:
		DINT;
		PieCtrlRegs.PIEIER1.all = TempPIEIER;

		//PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;	// Acknowledge interrupt to PIE
}
```

Bibliography

[1] G. Masson, S. Orlandi, and M. Rekinger, "GLOBAL MARKET OUTLOOK For Photovoltaics 2014-2018," European Photovoltaic Industry Association2014.

[2] P. Mints. (2014, May 15th 2015). *Despite Everything, 2014 Is Another Growth Year for Solar PV*. Available: http://www.renewableenergyworld.com/rea/news/article/2014/11/despite-everything-2014-is-another-growth-year-for-solar-pv

[3] M. Osborne. (2015, May 15th 2015). *Global solar demand in 2015 to hit 57GW on strong 30% growth rate – IHS*. Available: http://www.pv-tech.org/news/global_solar_demand_in_2015_to_hit_57gw_on_strong_30_growth_rate_ihs

[4] "Technology Roadmap Solar Photovoltaic Energy," International Energy Agency2014.

[5] U. E. I. Administration. (May 17th, 2015). *International Energy Statistics*. Available: http://www.eia.gov/cfapps/ipdbproject/iedindex3.cfm?tid=2&pid=2&aid=7&cid=regions&syid=2011&eyid=2012&unit=MK

[6] Q. Li and P. Wolfs, "A Review of the Single Phase Photovoltaic Module Integrated Converter Topologies With Three Different DC Link Configurations," *IEEE Transactions on Power Electronics,* vol. 23, pp. 1320 - 1333, May 2008.

[7] S. B. Kjaer, J. K. Pedersen, and F. Blaabjerg, "A Review of Single-Phase Grid-Connected Inverters for Photovoltaic Modules," *IEEE Transactions on Industry Applications,* vol. 41, pp. 1292 - 1306, Oct. 2005.

[8] W. Xiao, N. Ozog, and W. G. Dunford, "Topology Study of Photovoltaic Interface for Maximum Power Point Tracking," *IEEE Transactions on Industrial Electronics,* vol. 54, pp. 1696 - 1704, 2007.

[9] M. Kamil, "Grid-Connected Solar Microinverter Reference Design Using a dsPIC® Digital Signal Controller," *Microchip Application Note,* vol. AN1338, pp. 1 - 56, 2011.

[10] G. Petrone and C. A. Ramos-Paja, "Modeling of photovoltaic fields in mismatched conditions for energy yield evaluations," *Electric Power Systems Research,* pp. 1003 – 1013, Jan. 2011.

[11] "IEEE Standard for Interconnecting Distributed Resources with Electric Power Systems," *IEEE Std 1547-2003,* pp. 1-28, 2003.

[12] R. Carnieletto, Branda, x, D. I. o, F. A. Farret, Simo*, et al.*, "Smart Grid Initiative," *Industry Applications Magazine, IEEE,* vol. 17, pp. 27-35, 2011.

[13] J. C. Vasquez, R. A. Mastromauro, J. M. Guerrero, and M. Liserre, "Voltage Support Provided by a Droop-Controlled Multifunctional Inverter," *IEEE Transactions on Industrial Electronics,* vol. 56, pp. 4510 - 4519, Nov 2009 2009.

[14] V. Khadkikar, R. K. Varma, R. Seethapathy, A. Chandra, and H. Zeineldin, "Impact of distributed generation penetration on grid current harmonics considering non-linear loads,"

in *Power Electronics for Distributed Generation Systems (PEDG), 2012 3rd IEEE International Symposium on*, 2012, pp. 608-614.

[15] U. WANG. (2011) Small is better. *PV Magazine*. Available: http://www.pv-magazine.com/archive/articles/beitrag/small-is-better-_100003032/329/#axzz3U0aZ0ffD

[16] E. F. Fongang, W. Xiao, and V. Khadkikar, "Dynamic Modeling and Control of Interleaved Flyback Module Integrated Converter for PV Power Applications," *IEEE Transactions on Industrial Electronics,* vol. 61, pp. 1377 - 1388, March 2013.

[17] N. Kasa, T. Iida, and L. Chen, "Flyback Inverter Controlled by Sensorless Current MPPT for Photovoltaic Power System," *IEEE Transactions on Industrial Electronics,* vol. 52, pp. 1145 - 1152, August 2005.

[18] A. C. Kyritsis, E. C. Tatakis, and N. P. Papanikolaou, "Optimum Design of the Current-Source Flyback Inverter for Decentralized Grid-Connected Photovoltaic Systems," *IEEE Transactions on Energy Conversion,* vol. 23, pp. 281 - 293, March 2008.

[19] Y. Li and R. Oruganti, "A Low Cost Flyback CCM Inverter for AC Module Application," *IEEE Transactions on Industrial Electronics,* vol. 27, pp. 1295 - 1303, March 2012.

[20] A. C. Nanakos, E. C. Tatakis, and N. P. Papanikolaou, "A Weighted-Efficiency-Oriented Design Methodology of Flyback Inverter for AC Photovoltaic Modules," *IEEE Transactions on Power Electronics,* vol. 27, pp. 3221 - 3233, July 2012.

[21] F. F. Edwin, W. Xiao, and V. Khadkikar, "Topology Review of Single Phase Grid-Connected Module Integrated Converters for PV Applications," *Proc. Annual Conference of IEEE Industrial Electronics Society,* pp. 821 - 827, 2012.

[22] J. B. Wang, J. H. Wu, D. Kao, and T. Jung-Li, "Injection Current Phase Lag Effect of the Flyback Inverter," *IEEE Conference on Industrial Electronics and Applications,* pp. 1803 - 1808, 2011.

[23] J. G. Kassakian, M. F. Schlecht, and G. C. Verghese, *Principles of Power Electronics*: PEARSON.

[24] R. W. Erickson and D. Maksimovic, *Fundamentals of Power Electronics*, Second Ed. ed.: Kluver Academic Publishers, 2004.

[25] J. Klein, "AN-6005: Synchronous Buck MOSFET Loss Calculations," *Fairchild Semiconductor Application Notes,* pp. 1 - 7, 2006.

[26] K. Venkatachalam, C. R. Sullivan, T. Abdallah, and H. Tacca, "Accurate prediction of ferrite core loss with nonsinusoidal waveforms using only Steinmetz parameters," *8th IEEE Workshop on Computers in Power Electronics,* pp. 36 - 41, 2002.

[27] C. P. Steinmetz, "On the law of hysteresis," *Proceedings of the IEEE,* vol. 72, pp. 197-221, 1984.