Computer-Aided Design and Optimization of dc/dc Power Converters

by

Timothy Carl Neugebauer

B.S., Union College (1997)

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

September, 1999

Signature of Author_

Department of Electrical Engineering and Computer Science
September, 1999

Certified by_

John G. Kassakian
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by_

David J. Perreault
Research Scientist
Secondary Thesis Supervisor

Accepted by__________

Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Computer-Aided Design and optimization of dc/dc Power Converters

by

Timothy Carl Neugebauer

Submitted to the Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Abstract

The imminent introduction of a dual-voltage automotive electrical system has motivated the development of dc/dc converters that are optimized it terms of price, weight, and volume. This thesis investigates the design and optimization of dc/dc converters for dual-voltage automotive electrical system. A prototype dc/dc converter is developed and experimental evaluated, and is used to identify a number of tradeoffs and critical design issues. Based on this information, a CAD optimization tool is developed which allows the design space to be rapidly explored and highly optimized converter design to be developed. The CAD optimization tool is also used to study the effects of variations in system-level specifications on the characteristics of optimized converters.

Thesis Supervisor:	John G. Kassakian Professor of Electrical engineering
Secondary Thesis Supervisor:	David J. Perreault Research Scientist

Acknowledgments

This is the section in which I thank lots of people for all the help they have provided me, intellectually or otherwise, up to and including the time in which this thesis is complete.

All the characters and events portrayed on this page are fictional, and any resemblance to real people or incidence is purely coincidental.

I am most grateful to David Perreault and Harvey Callahan for all the technical help they have provided. I doubt there exists a question in EE that they could not solve and fully explain. I would also like to thank Ms. Dave for allowing her husband to help me for the past two years.

I would also like to thank the following people:

To Mom, just because you're Mom.

To J.G.K., T.K., T.J., and G.D. for providing me with the opportunity that I had always dreamed about.

To V.M. and K.J. for all the work you've done.

To Flo, I could never ask for a better friend.

To V.C. and L.N.P. for giving everyone in the lab something to laugh about.

To the makers of Microsoft Office and other related software, for providing me with the unique pleasure of using your products, which was a once in a lifetime experience.

To the makers of the Unix program LaTex, whose product I will be using for any other major documents that I will have to write.

To B.M., for your thesis was an enormous aid to my research.

To the various members of research unit number 1, for all the late night meeting that occurred.

To my brothers Brian and John, my father, mother, grandmother, and the rest of my family, for providing me with a place I could always go home to.

To the people of L.E.E.S. who also providing me with an office that was just as good as any home.

To the programmers of Blizzard, the makers of StarCraft and Diablo.

To Hoss for bringing a little part of Texas into the lab.

To Scott Adams, for showing that engineers need a sense of humor.

To Fred, for he is like the twin brother I never had. He is also one of the top 3 xkobo players at MIT.

To J.G.G., T.B., and T.N.G. for your help in TimCAD's database.

To all my friends at Union college, A.C., M.L., and E.H. to name a few.

To everyone I forgot to list.

And finally to Mingjuan Zhu, who inspired me to always work as hard as I can.

Table of Contents

Chapter 1

Introduction

A revolution in the automotive electrical system is imminent. The electrical needs of the automobile are ever increasing whereas the power available from conventional alternators is reaching a saturation point. The members of the *M.I.T./Industry Consortium on Advanced Automotive Electrical/Electronic Components and Systems* have agreed that a dual-voltage system is the next evolutionary step for automotive electrical systems. High power loads would be moved up to a high-voltage bus, while incandescent lighting, electronics, and other loads that benefit from a lower voltage can remain on the (currently-used) low-voltage bus. With the advent of the dual-voltage system, new high-voltage, high-power loads that were impractical to implement in the 14 V system will become available. Components and system benefits that will accrue due to the dual-voltage system are described in [1-3].

1.1 A Dual-voltage Architecture

The dual-voltage system will require electrical energy to be supplied to two voltage busses. One widely considered method for doing this is illustrated in Fig. 1.1. A high voltage alternator supplies the high-voltage bus, while the low-voltage bus is

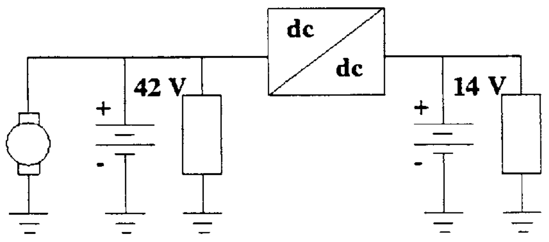


Figure 1.1 The dc/dc converter-based architecture for the dual-voltage system

powered from the high-voltage bus via a dc/dc converter. The dc/dc converter can provide a stable, well-controlled voltage on the lower bus. The amount of current flowing from one bus to another can be easily controlled and limited with the converter. A bi-directional converter can be employed in order for the low voltage bus to temporarily power the high voltage bus so that the high-voltage battery can be recharged from the low-voltage battery. These capabilities make the dc/dc converter-based system the most desirable of available options from a performance standpoint.

The major drawback to the dc/dc converter-based architecture is the cost of the converter. The converter is constructed from relatively high-cost components, such as high-frequency power switches, capacitors, and magnetic elements. Large input and output power filters are also needed to attenuate the conducted EMI generated by the switching action of the power converter. Thus, the viability of the architecture depends upon the ability to develop power converters at a low manufactured cost.

This thesis will address two main challenges in the design of the dc/dc converter. The first challenge is to develop an automated method for finding an appropriate design that will minimize the price, weight and volume of the dc/dc converter based on a set of specifications. The second challenge is to develop a method for finding the sensitivity of the price, weight and volume of an optimized converter to system-level parameters such as the power rating, ambient temperature, or EMI limit.

Estimates[1] of the power requirements for luxury cars employing dual-voltage electrical systems indicate an average 14 V loading of 600 – 800 W, with peak loads exceeding this by a factor of two or more. This suggests the need for a dc/dc converter

[1] See Appendix B

with an output power rating of 800 – 1000 W if a 14 V battery is used, and much higher if a 14 V battery is not employed. As more loads are redesigned and moved to the 42 V bus (leaving lighting and electrical loads on the 14 V bus), one might expect the converter power requirements to be reduced to the 400 – 500 W range.

For the converter to be used in the automotive environment it must meet several other requirements. If the dc/dc converter is to be placed under the hood it must be designed to operate in a high ambient temperature. Furthermore, the converter will have to meet extremely tight electromagnetic interference (EMI) requirements to prevent it from interfering with other equipment on the vehicle. The converter will also need to be configured to control the charging of the 14 V battery, and accept input from an external energy management scheme that will direct the power flow to ensure that neither battery becomes discharged.

A wide range of design possibilities exists for a dc/dc converter in the automotive application. The designer can choose among a wide range of subsystem topologies, design parameters, and component types to design a converter. Because of this large design space, and the tight interrelation among many of the design choices, identifying the converter designs that best meet the design objective is extremely difficult in practice. It is also difficult to ascertain a priori the effects of system level parameter variations on the resulting price, weight, and volume of the "best" converter design.

1.2 Thesis Objectives

The dc/dc converter is not a new device, but introducing it as a component in the automotive electrical system is a new application with unique requirements. Furthermore, there are a number of aspects of the design, utilization, and control of the converter that are unique to this new application. The first objective of this thesis is to develop and evaluate a prototype dc/dc converter for this application.

The second objective of this thesis is to investigate the optimization of power converters for this application. The design of a converter involves numerous tradeoffs that make optimization extremely challenging. To meet this challenge, a computer-aided design (CAD) optimization tool has been developed. The CAD program allows the design space to be rapidly explored for optimal designs and design strategies.

The last objective of this thesis is to use this dc/dc converter optimization program to examine trends within the design space. These trends will show the sensitivities of the converter price, weight and volume to changes in system specifications. The examination of these trends will inform the auto industry as to the quantitative effects of several major system level choices.

1.3 Thesis Organization

This thesis contains seven chapters including this introductory chapter. Chapter 2 covers the design and evaluation of a prototype converter, describing the major tradeoffs and choices. The optimization routine used to analyze the system is described in Chapter 3. Chapter 3 also contains information about the structure of the CAD program and directions for its use. Results from the program are shown in Chapter 4, including the sensitivity of the converter characteristics to power rating, ambient temperature, and EMI specifications. Chapters 5 and 6 contain the models for the devices, passive elements and filters that are used in the program. Lastly, Chapter 7 concludes the thesis.

Chapter 2

Design of the Prototype Converter

In the course of this thesis a prototype dc/dc converter was designed and built. The purpose of developing the prototype converter was to provide an understanding of the converter design tradeoffs, and to allow operational experience to be gained with power converters in this application. For purposes of this thesis, the prototype converter is useful for validating component models in the optimization code. The converter has been designed using many models (or early version of the models) used in the computer-aided design optimization program, but without the benefits of automation of the process. The prototype converter described here has also been used in a number of other automotive electrical system projects both at M.I.T. and at the Ford Scientific Research Laboratories in Dearborn, Michigan.

2.1 Requirements

2.1.1 Power Levels

The proposed dual-voltage electrical system for the next generation of automobiles will contain a 42 V and a 14 V bus. The current automotive electrical system, as shown in Fig. 2.1a, consists of an alternator, a 12 V battery (regulated to 14 V for charging), and a single 14 V bus. Many specific forms of the new dual-voltage system are possible, but the general form of the new voltage system is illustrated in Fig. 2.1b. Of the candidate architectures that are being considered, the baseline form, utilizes a new, high-voltage

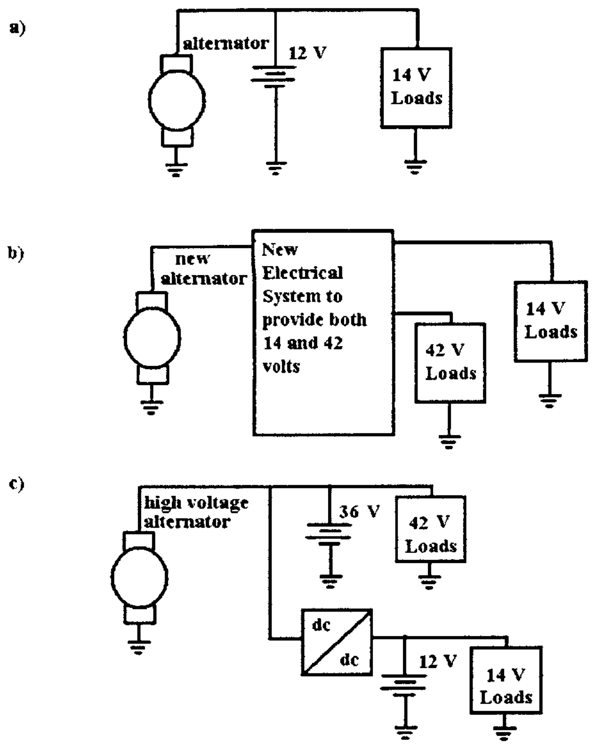


Figure 2.1: (a) The conventional automotive electrical system, (b) the proposed dual voltage automotive electrical system, and (c) an example of a dual voltage system implementation.

alternator and a dc/dc converter, as illustrated in Fig. 2.1c.

The Forum Bordnetz, a European automotive discussion group, has proposed a set of specifications for the new electrical system [4]. These specifications, which are summarized in Appendix A, set the maximum and minimum voltages for both busses under various conditions. The static voltage of the high voltage bus can vary from 33 to 52 V, and the static voltage of the lower voltage bus can vary from 12 to 16 V when the engine is running. These limits have been adopted for the design of the prototype converter.

In order to determine the power level of the electrical system a comprehensive list of electrical loads in the automobile (developed in [5] and shown in appendix B) was examined. It was assumed that when the dual-voltage system is introduced some of the major loads will be moved to the higher bus, and that many other loads will initially remain on the lower bus and be transferred to the higher bus at a later time. With this assumption the load list was examined to determine the worst case and nominal power requirements for the low-voltage bus. Based on the load list, the nominal power for the low-voltage bus was predicted to be approximately 700 W, whereas the peak power needed by the system would be approximately 1500 W. In order for the low-voltage system to operate with these power demands, either the dc/dc converter would have to be sized to deliver the peak power required, or a 12 V battery is needed in the system. To supply 700 W to the 14 V bus the converter would need to deliver between 43.8 and 58.3 A, depending on the 14 V bus battery voltage. The maximum current available from the prototype is 68 A, enough to supply the average power required by the low-voltage bus plus an additional amount to keep the battery charged. The converter can also be used without a battery in a system where the peak load remains below 68 A.

2.1.2 EMI Limits

In order to control the amount of Electromagnetic Interference (EMI) on the conventional 14 V bus the Society of Automotive Engineers (SAE) has adopted the specification SAE J1113/41. This specification limits the amount of conducted EMI any given device can generate on a standard test stand which includes a Line Impedance Stabilization Network (LISN). The conducted EMI is specified as the amount of voltage ripple that appears across the 50 ohm LISN impedance as a function of frequency. Figure 2.2 shows the relationship between the allowed ripple voltage and frequency as specified in the Class 1 narrowband limits. This is the EMI specification that is applied to the converter design. No standards presently exist for the EMI injected into the higher-

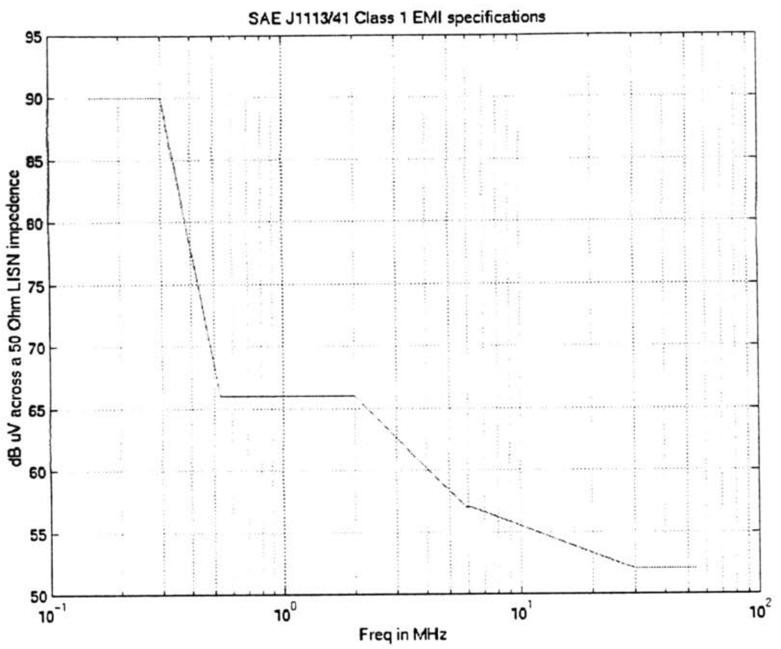


Figure 2.2: The SAE J1113/41 specification for Class 1 narrowband signals. This specification is applied to both the input and output of the dc/dc converter.

voltage bus, but the same limits were applied to this bus for the design of the converter.

2.1.3 Ambient Temperature

The dc/dc converters for this application are likely to be located under the hood of the automobile in close proximity to the alternator and the batteries. In recognition of this, the prototype converter is designed to operate in ambient temperatures of up to 105 °C. This high temperature required the components for the design to be carefully selected, and in many cases oversizing or derating of components was necessary. Also, only natural-convection air cooling is used in the design for simplicity and robustness. It should be pointed out that the high ambient temperature strongly affects the design.

2.2 Tradeoffs

There are many interrelated tradeoffs in the design of the dc/dc converter that complicate the optimization of the design. These tradeoffs include the decision to use one large converter cell or a paralleled cell interleaved design and, if an interleaved system is used, the number of cells to use. More interleaved cells will reduce filter size and power losses in the switches, but will increase the system complexity. Another tradeoff involves the sizing of the converter inductance and ripple ratio versus the size of input and output filters to limit ripple. These tradeoffs were considered in the design of the prototype converter, and will be described here.

2.2.1 Modular or Single Structure

Two main options exist for the basic structure of the converter. Either one large dc/dc converter can be built (using paralleled components where necessary) or the converter can be divided into several paralleled converter cells as shown in Fig. 2.3. The advantages of one large converter include the simplicity of the structure and control, but does not necessarily include fewer parts. Due to the high current, low-voltage characteristics of the converter, and the high ambient temperature specification, most power stage elements in the converter would have to be constructed of paralleled components in order to build a single large converter structure.

If the converter is cellularized, or separated into different cells, then the cells can be *interleaved* in order to increase the fundamental frequency of the ripple and reduce the magnitude of the ripple at the input and output ports of the converter [6-9]. To interleave the cells of the converter, the cell switching times are equally phase displaced over a

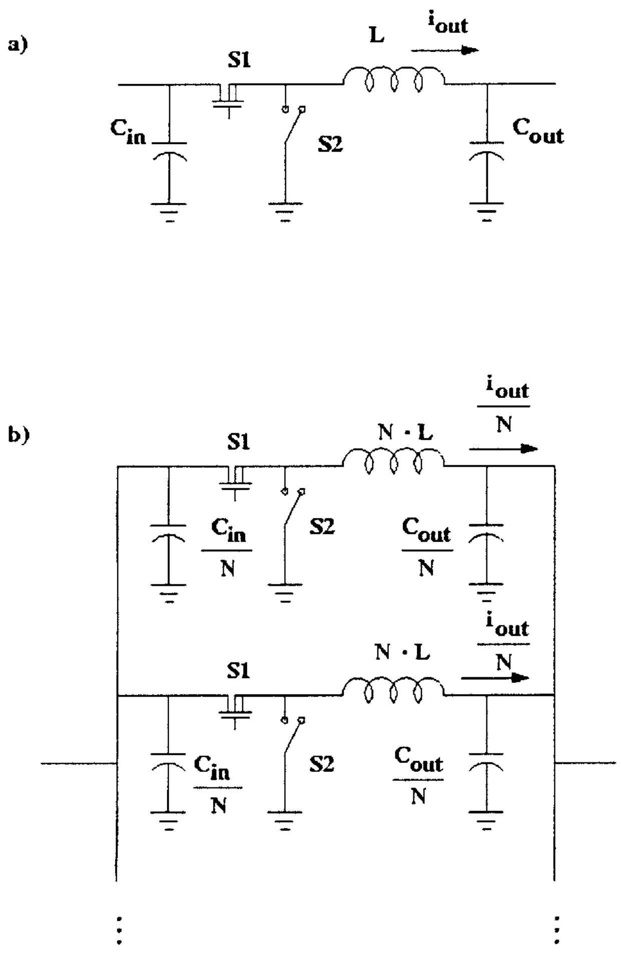


Figure 2.3 Comparison of the power stages of a) a dc/dc converter with a single structure; and b) an equivalent converter made up of N cells.

cycle, as illustrated in Fig. 2.4. Interleaving *N* converter cells results in a net functional ripple frequency at the ports of the converter which is *N* timed the individual cell frequency. Furthermore, it can be shown that the peak-to-peak ripple current amplitude at the ports will be at least a factor of *N* times smaller then would be obtained with a single large converter or with synchronous switching [6-9].

Due to the need for paralleled components even in a single converter design, and considering the additional benefits provided by an interleaved design, a cellular design for the prototype was chosen. Cellular designs using between three and six cells were

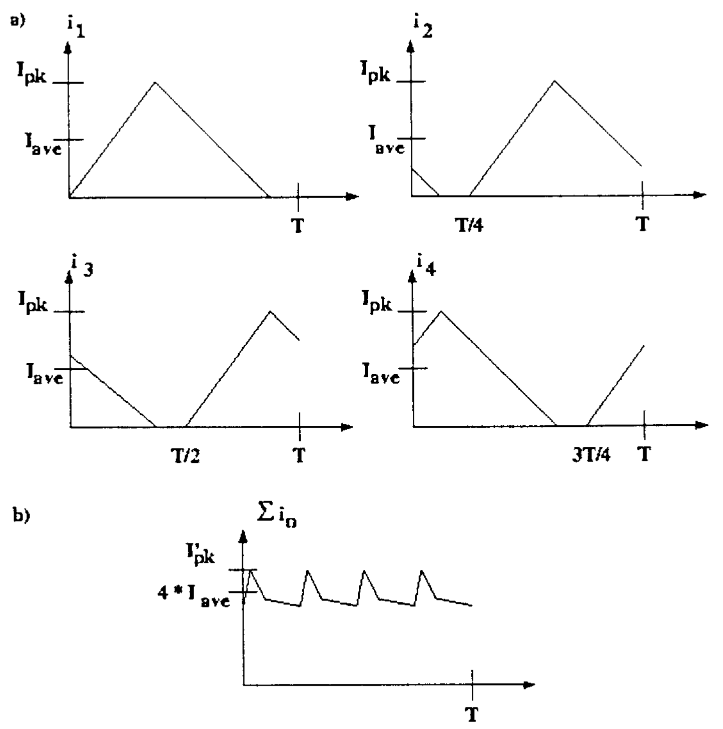


Figure 2.4. Output current waveforms in a 4 cell converter with interleaved switching. a) The output of each of the 4 stages and b) the resulting interleaved output current.

considered. Using two cells would have still required using paralleled components in the cells, and due to practical limitations the benefits of interleaving cease to accrue when more then about 6 cells are used.

2.2.2 Effects of Number of Cells on Output Ripple

Because the converter consists of several cells, each cell can be operated in discontinuous conduction mode and the output of the converter will still be continuous, as illustrated in Fig. 2.4. After considering different options, the prototype converter was designed such that the converter always operates in discontinuous conduction mode (DCM) regardless of the voltage on either bus, and operates at the edge of discontinuous conduction at one edge of its operating voltage range. This decision was made to keep the cell inductor size small and to reduce the complexity of the controller. The converter was designed to operate at a cell switching frequency of 125 kHz. This value represents a typical value for the operation of a dc/dc converter and was an initial assumption for the design of the converter.

The cell inductance needed to ensure that the cell remains in DCM can be determined as follows: at the edge of DCM the average current through the inductor, $I_{ave} = \frac{I_{max}}{N}$ (where I_{max} is the rated current for the converter), is half of the peak current in the cell I_{pk}. The peak current in a cell operating in DCM is

$$I_{pk} = \frac{V_{in} - V_{out}}{L} \cdot D \cdot T \tag{2.1}$$

Assuming that the converter cell is operating at the boundary between discontinuous and continuous (CCM) conduction modes, then the duty cycle is

$$D = \frac{V_{out}}{V_{in}} \tag{2. 2}$$

and I_{pk} is twice I_{ave} ($I_{pk} = 2 \cdot I_{ave}$). Thus, the maximum inductance to ensure DCM is

$$L_{max} = \frac{(V_{in} - V_{out}) \cdot V_{out} \cdot T \cdot N}{2 \cdot V_{in} \cdot I_{ave}} \tag{2. 3}$$

The minimum value of

$$f(V_{in}, V_{out}) = (V_{in} - V_{out}) \cdot \frac{V_{out}}{V_{in}} \tag{2. 4}$$

can be found by solving for when either $\frac{\partial f}{\partial V_{in}}$ or $\frac{\partial f}{\partial V_{out}}$ equals zero. Since there is no solution to the partial derivatives within the voltage ranges, the minimum value will be at one of the edges of the voltage range. In our case the minimum is $f_{min} = 7.63$. The maximum cell inductance to ensure that the converter always operates in DCM is

$.45\mu H \cdot N$, assuming that the cell switching frequency of 125 kHz, the maximum output current of 68 A, and N cells are used.

Based on the inductance, bus voltages, switching frequency, number of cells, and output current, the amount of current ripple that the converter contributes to the input or output busses can be calculated. The waveform of the current in each cell can be computed, then by summing the currents in each cell, the aggregate peak to peak ripple can be found. For any given set of input parameters such as output current or inductance, the ripple can be plotted for all combination of bus voltages. Four examples of these plots are shown in Fig. 2.5 assuming that the cell switching frequency is 125 kHz, that the converts are rated for 68 A, and the cell inductance is $.45\mu H \cdot N$. The plots in this figure help to visualize the effects of the operating conditions on the current ripple for different

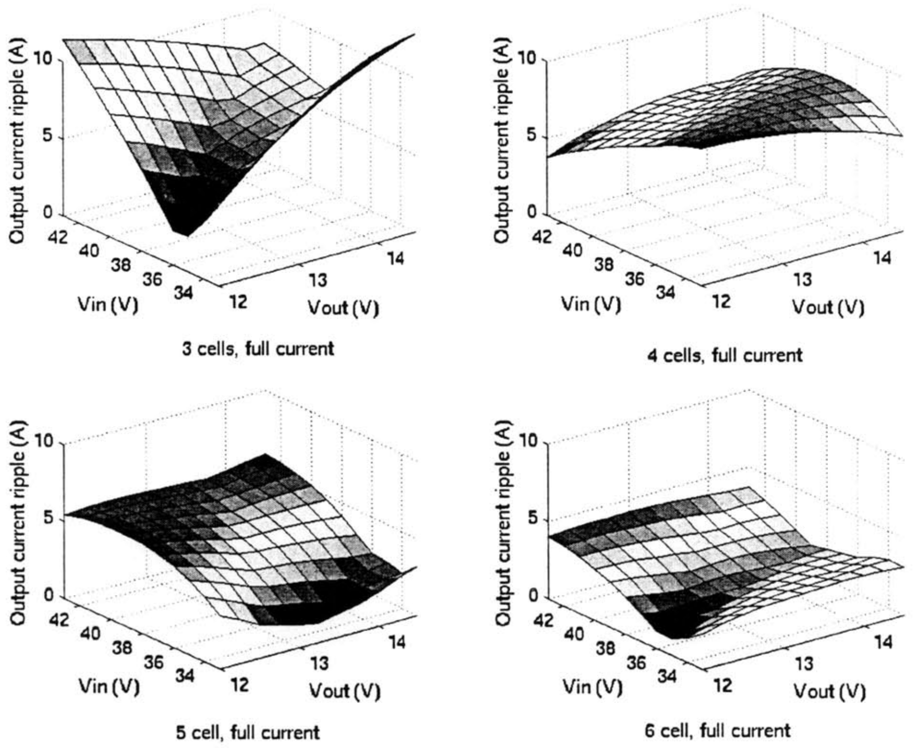


Figure 2. 5 An example of plots relating the output current peak to peak ripple that results from operating the converter at various combinations of bus voltages.

numbers of cells. To further ease visualization of this relationship the data can be shown on a two dimensional plot as in Fig. 2.6. In this figure the maximum ripple was found for every value of V_{out} across all possible values of input voltage and output current. With each increase in the number of cells, the fundamental ripple frequency increases and output peak to peak current ripple decreases. The resulting tradeoff is that increasing the number of cells makes filtering easier, but also increases the part count and the complexity of the converter.

2.2.3 Effects of the Number of Cells and Switch Type on the Temperature Rise and Required Heatsink

Another major effect that the number of cells has on the system is the size of the required heatsink. As the number of cells, *N*, increases, the average current per cell decreases by a factor of *N*. This reduction in current causes a reduction in total

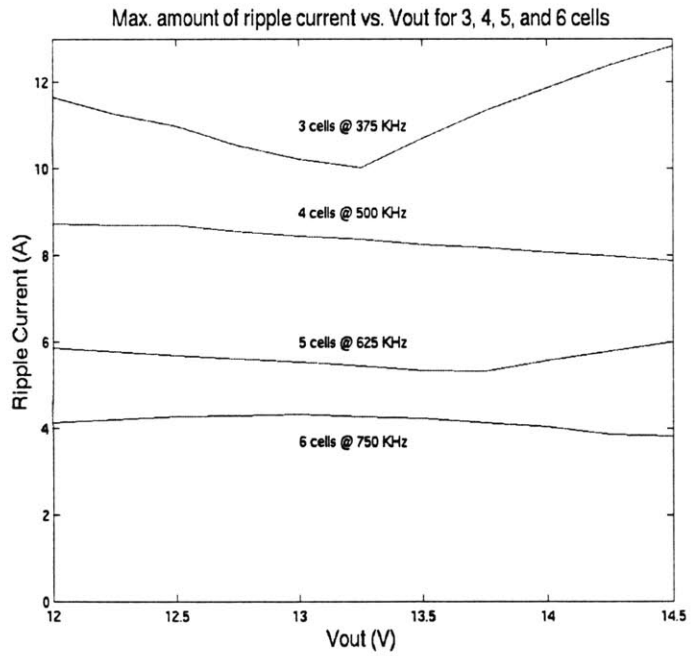


Figure 2.6 Comparison of the worst case output peak to peak current ripple as a function of cell number and output voltage across all allowed output currents. Listed also is the fundamental ripple of the output current. In all cases the switching frequency is 125 kHz.

conduction losses in the switches if the same switches are used independent of the number of cells. (Chapter 5 describes how to calculate the device losses and how to choose the heatsink.) Switches with lower power losses require smaller heatsinks. Thus, the tradeoff is that using more cells results in smaller net heat sinking but a higher part count.

Switch selection affects the power dissipation. The three combinations of topologies considered and shown in Fig. 2.7 result in different power losses under different operating conditions. Also, using two MOSFETs may result in lower power losses but will require extra control circuitry including another gate driver. Table 2.1 displays the predicted power loss and temperature rises of the converter using two MOSFETs and a MOSFET and a Schottky diode. Both systems assume identical heatsinks.

The prototype converter has four cells and uses the *Temic* SUP75N08-10 MOSFET and the *International Rectifier* 40CPQ080 Schottky diode as switches S1 and

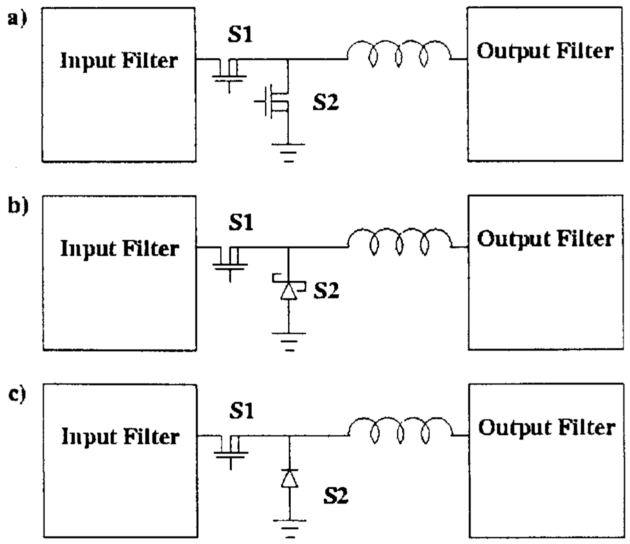


Figure 2.7 The three possible topologies for the power stage: a) MOSFET/MOSFET, b) MOSFET/ Schottky Diode, and c) MOSFET/PiN Diode.

Topology	S1 –MOSFET SUP75N08-10 S2 –Schottky diode 40CPQ080 KM 150-1 Heatsink			S1 –MOSFET SUP75N08-10 S2 –MOSFET SUP75N08-10 KM 150-1 Heatsink		
Cells	4	5	6	4	5	6
S1 Losses	4.45 W	3.24 W	2.53 W	4.45 W	3.24 W	2.53 W
S2 Losses	7.28 W	5.72 W	4.61 W	4.50 W	2.92 W	2.06 W
Total Loss	11.73 W	8.96 W	7.14 W	8.95 W	6.16 W	4.59 W
S1 Junction to Case Temp. Rise	4.9 °C	3.56 °C	2.78 °C	4.9 °C	3.56 °C	2.78 °C
S2 Junction to Case Temp. Rise	8.52 °C	6.69 °C	5.39 °C	4.94 °C	3.21 °C	2.26 °C
Heatsink Temp. Rise	40 °C	31 °C	27 °C	31 °C	24 °C	17 °C
Max Junction Temp. Rise (Either Device)	48.52 °C	37.69 °C	32.39 °C	35.94 °C	27.56 °C	19.78 °C

Table 2.1 A comparison of the predicted temperature rises for two device configurations and various numbers of cells.

S2. The MOSFET/Schottky diode combination was chosen to reduce the complexity in the control. Four cells are used in the design of the converter as a compromise of the various factors. The *Thermalloy* KR-150 heatsink, a model larger then the KM-150, was used in order to ensure that if the devices are in an ambient temperature of 105 °C then the junction temperature of the devices will not exceed 140 °C under any condition.

2.2.4 Cells Inductance and Filter Size

For a given number of cells the cell inductance used in the power converter determines the amount of ripple that appears on the input and output busses. Figure 2.8 demonstrates this relationship for a converter with six cells.

The filters for the converter must attenuate the converter ripple. The converter input and output ripple (ideally) has a fundamental frequency that is the switching frequency times the number of cells. The filters must attenuate the fundamental and the harmonics of these currents such that they meet the EMI specification. Thus, an increase in the size of the cell inductance will decrease the magnitude of the harmonics from the

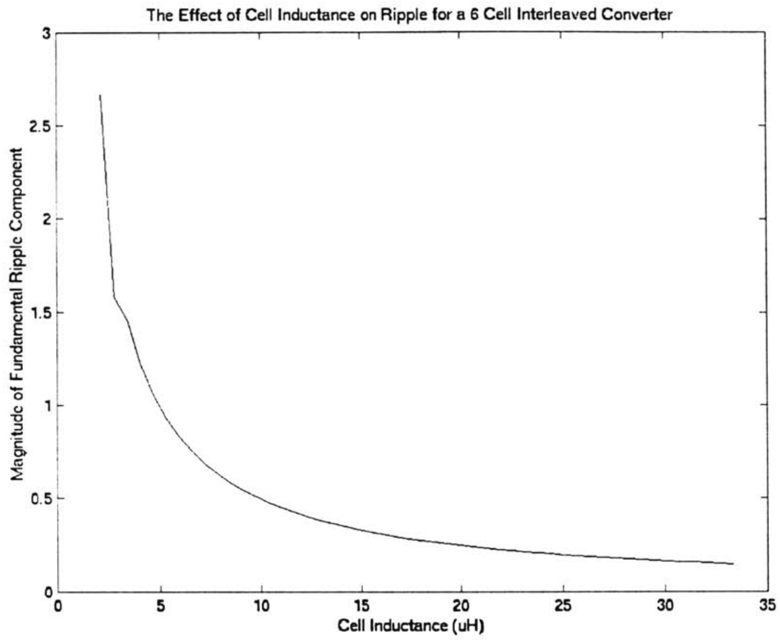


Figure 2.8 The relationship between the magnitude of the ripple and the cell inductance for a six-cell converter. For every data point taken the design range of input and output voltages was examined to determine the maximum fundamental ripple component.

converter and therefore smaller filters can be used since less attenuation is needed. In the design of the prototype converter this tradeoff was not examined; the power stage and the filters were designed separately. EMI filter topologies and designs are discussed further in Chapter 6.

2.3 The Prototype Converter

2.3.1 Major Components

The prototype was built in June of 1998 with the parts and parameters listed in Table 2.2 and shown in Fig. 2.9 and Fig.2.10. The switching frequency and inductance were chosen first. The inductance was chosen to guarantee that the converter cells would run in the discontinuous conduction mode over the entire operating range, ensuring a small inductor core size. Four cells were selected to keep the part count low while also

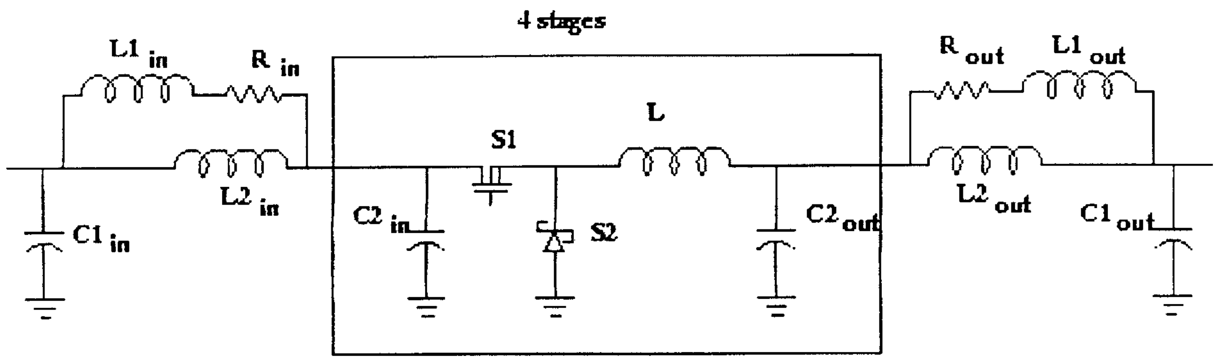


Figure 2. 9 The layout of the prototype converter.

Power Stage	
Voltage ranges	See Appendix A
Maximum Current	68 A
Switching Frequency	125 kHz
Number of cells	4
Switch, S1	Temic SUP75N08-10
Switch, S2	IR 40CPQ080
Heatsink	Thermalloy KR-150
Inductor Core	Philips RM10PA160 core with 3 turns
Inductance (per cell)	1.44 μH
Input Filter	
C2	80 μF Cornell Dubilier 4 X 935C2W20K
C1	20 μF Cornell Dubilier 1 X 935C2W20K
R	.27 Ω
L1	200 nH
L2	20.5 μH Micrometals T157-40 core with 20 turns
Output Filter	
C2	80 μF Cornell Dubilier 4 X 935C1W20K
C1	20 μF Cornell Dubilier 1 X 935C1W20K
R	.22 Ω
L1	100 nH
L2	11.4 μH Micrometals T250-40 core with 4 windings of 10 turns each

Table 2.2 Specifications and major components for the prototype converter.

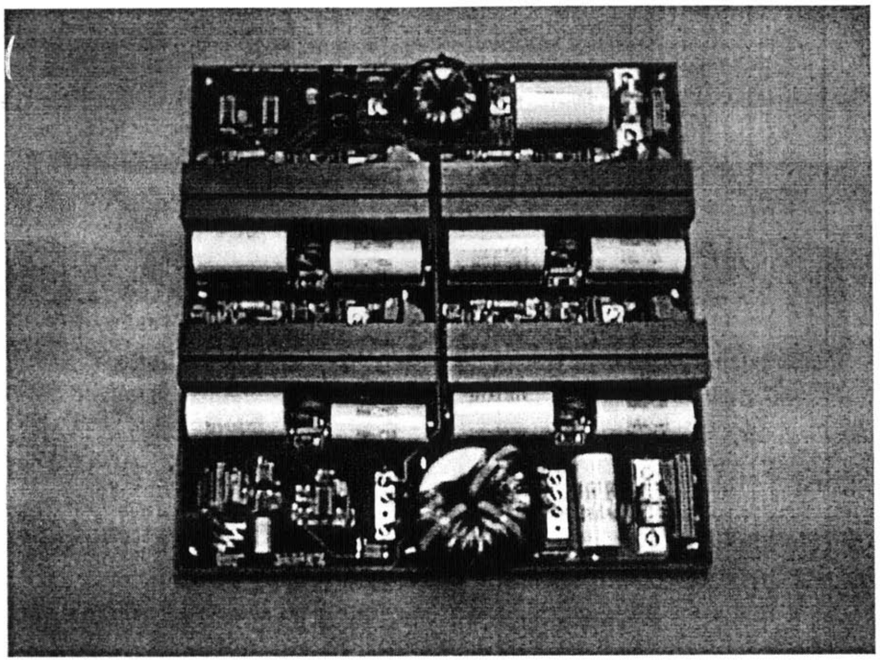


Figure 2.10 Photograph of the dc/dc converter.

benefiting from the ripple reduction of interleaving and only requiring a single PC-board mountable heatsink for each cell. The FET/Schottky diode combination was chosen to reduce complexity in the control. Finally, a heatsink was chosen so as to ensure the maximum junction temperature set for the semiconductor devices would never be exceeded.

Once the key components were determined, the input and output EMI filters were designed. The topology and the design algorithm for the EMI filter were adapted from [10]. To design the filter, the input and output ripple across all operating conditions was examined and the worst-case harmonic at each frequency was computed. The filters were designed so as to ensure that these ripple components were sufficiently attenuated to conform to the SAE J1113/41 Class 1 EMI specification. Additional information about EMI filter topologies and design can be found in Chapter 6.

2.3.2 Control

The output voltage of a modern alternator is controlled with a feedback loop. As the voltage on the bus falls below a temperature-adjusted reference value the alternator will supply more current in order to recharge the battery and raise the bus voltage. Above a certain value, however, the alternator current reaches a limit and ceases to rise. The converter is controlled such that it will operate in a manner similar to an alternator but with much higher accuracy and bandwidth. Figure 2.11 shows a graph of the output voltage/output current characteristics of the converter. A maximum current, I_{max}, of up to 68 A can be set from a digital input. The converter provides a current that is proportional to the difference between the output voltage and a temperature-adjusted reference up to the current limit. The proportionality constant is 170 A/V. The reference voltage is set to 14.2 V at 25 °C, with an ambient temperature adjustment of -7 mV/°C to match lead-acid battery characteristics. Note that the output current limit for the converter is digitally controllable between 0 and 68 A. An 8-bit digital input is provided which allows this

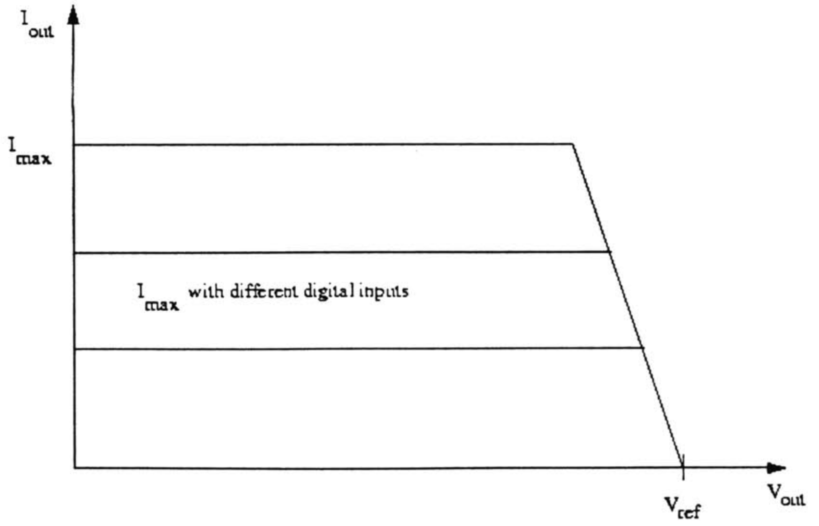


Figure 2.11 The effects of output voltage on the output current of the converter. The level of I_{max} is controllable.

current limit to be set. This input is highly useful for managing the flow between the high and low-voltage batteries.

The converter is controlled with two feedback loops. An outer loop generates a cell current command based on the error between the output voltage and the reference voltage. This current command is fed to the current-control loops of each cell.

The individual cells employ average current-mode control to ensure that their average output currents track the command current from the outer loop. (Average current was selected over peak current control or duty ratio control because it was found to result in the lowest sensitivity of the ripple cancellation to mismatches in cell inductances.) The average cell currents are measured across 5 mΩ sense resistors in series with the cell inductors. The error between the commanded and measured currents is used to determine the duty ratio of the cell.

The inner control loops operate at a relatively high bandwidth. The outer loop operates at a much lower bandwidth resulting in an overall bandwidth on the order of the outer loop crossover frequency of 1 to 10 kHz.

The controller is also required to be stable and well behaved both with and without a battery on the output. More details of the control will be described in [11]. For full schematics of the prototype see Appendix C.

2.4 Test Results

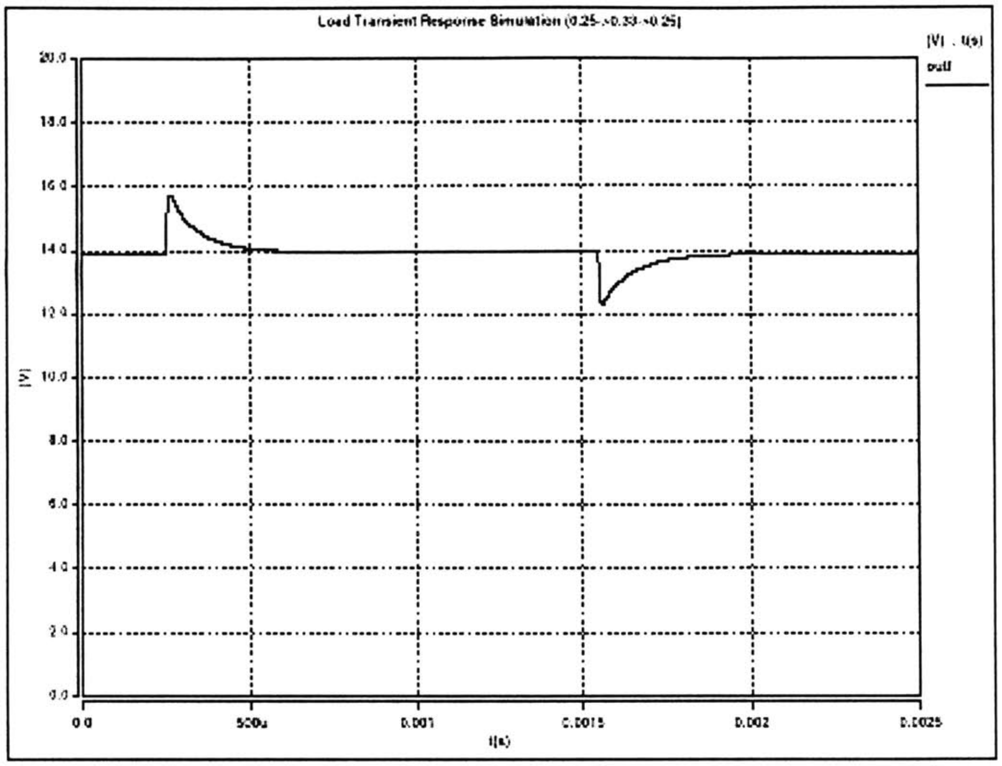
The converter was tested to ensure proper operation and stable control for operation with and without an external battery. A load step test was performed to compare the transient response of the 14 V bus in the dual-voltage system to the nominal system in use currently. The conducted EMI of the converter was measured to evaluate the performance of the filters. The temperature rises of various components were measured in order to estimate the accuracy of the temperature rise prediction of the various models, used to design the converter.

2.4.1 Transient Tests

One of the transient tests preformed on the converter is a step change in load. These transitions occur when a major load is abruptly disconnected from the bus. The dc/dc converter can respond to this step change in current thus allowing a quick regulation of the voltage. Careful control of such transients is especially important in a system where a 14 V battery is not used. Figure 2.12 shows a load-step transient on the low-voltage bus of a load resistance increase from .25 Ω to .33 Ω then back to .25 Ω, changing from 78 % to 59 % to 78 % of full load, without a low-voltage battery. Figure 2.12a is the simulated result from a Saber model, and Fig. 2.12b is the measured transient. These results show that the performance of the converter is as expected and is quite acceptable from a transient point of view. The voltages never exceeded the specifications listed in Appendix A and the transients decay an order of magnitude faster then load step transients in the conventional system under similar conditions.

2.4.2 EMI Measurements

The conducted EMI on the input and output busses were measured and compared to the SAE J1113/41 specification. The results of the tests are shown in Fig. 2.13. In both cases the conducted EMI exceed the specified limits for frequencies less then 3 MHz. This result was due to the fact that the models used to design the input and output filters did not sufficiently consider capacitor parasitics. To rectify this problem a more detailed model of the filter that considers more of the component parasitics (which is fully a)



b)

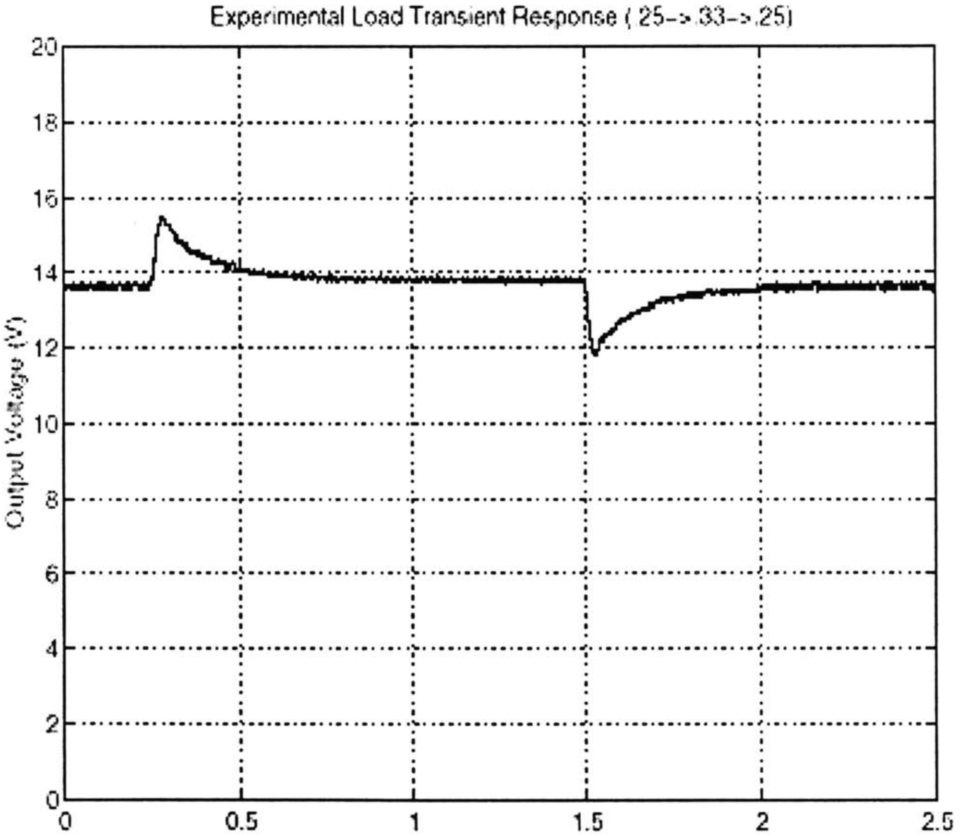


Figure 2.12 Transient response of the output voltage (in Volts) to a step in resistance on the low voltage bus without an output battery a) the simulated response b) the measured response. Along the time axis each division is .5 milliseconds

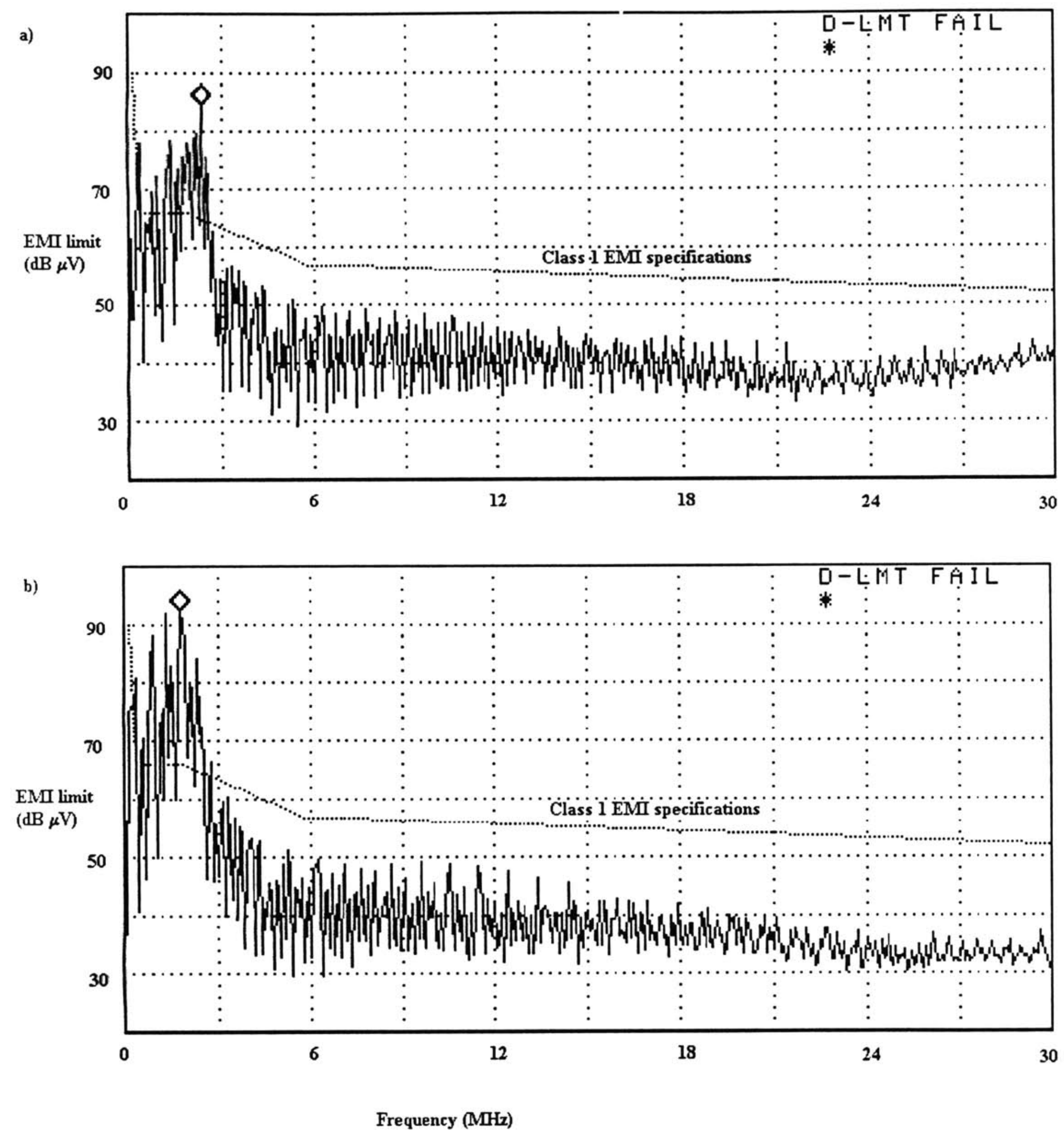


Figure 2.13 The measured conducted EMI on the prototype a) input bus b) output bus at the rated power level.

described in Chapter 6) was developed for use in the CAD optimization program. Nevertheless, the performance of the converter was found to be acceptable for test purposes, i.e. the EMI interference did not adversely affect any other systems.

2.4.3 Thermal Measurements

A major design criteria for most of the components in the converter is a limited maximum temperature. In order for the heatsink and other components to be properly chosen the power losses of the major components and resulting ΔT's must be accurately calculated. Through thermal testing of the prototype converter, the actual temperature rises of key components within the converter were compared to the predicted values in order to test the validity of the models used.

For the thermal tests the converter was operated at an output power of 560 W. The temperature rise of the converter cell inductor centerpost and the heatsink were measured to be 13.3 °C and 11.9 °C, respectively. The outer case of the diode has a temperature that is 3.9 °C higher than the heatsink. The calculated temperature rise of the converter cell inductor is 13.7 °C. The thermal model of the heatsink and both devices, shown in Fig. 2.14, predicts the temperature rise of the heatsink to be 12.3 °C. The measured temperature rises for the converter inductor and the heatsink were close to the estimated temperature rises, implying that the calculated power losses of the elements in the power stage were accurate.

2.5 Conclusions

Several objectives were met by the development of the prototype converter. The design and construction has helped to elucidate a number of tradeoffs and hidden issues within the design. The tradeoffs identified include the effects of the number of cells on system performance and the relationship between the cell inductance and filter size. The converter has been useful for validating and refining the filter and device models used in

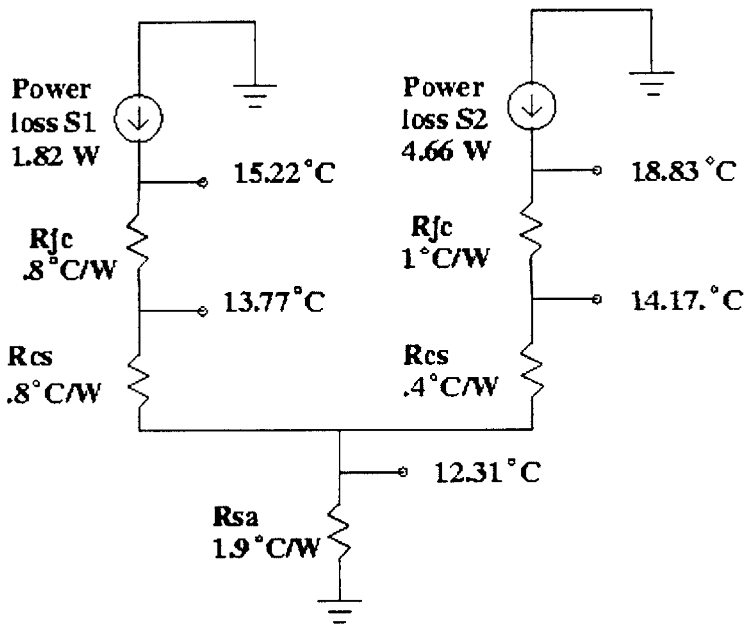


Figure 2.14 The thermal model of the heatsink and devices used in the prototype converter running at the reduced rating of 560 W. R_{jc} are the thermal resistances of the devices listed in Table 2.2. R_{cs} is the thermal resistance of the sil pads used. R_{sa} is the thermal resistance of the heatsink.

the CAD optimization package. The converter has also proven useful in a variety of dual-voltage automotive research projects that include load flow studies and transient analysis.

Chapter 3

TimCAD Architecture

A wide range of design possibilities exists for a dc/dc converter in the dual-voltage application. The converter can be constructed from a single power stage or any number of paralleled power stages. Within the power stage design, a wide range of switching frequencies and ripple ratios are feasible, and the power stage can be constructed from a tremendous variety of component types. Many topological variations are possible as well. Because of this large design space and the tight interrelation of many of the design choices, identifying the converter designs that best meet the design objectives is extremely difficult in practice.

TimCAD is a computer-aided design and optimization tool that allows the design space to be rapidly explored, and the most optimal design approach to be identified. It also allows the effects of system parameter variations on the price, weight, and volume of an optimized converter to be easily determined. This chapter describes the structure of the TimCAD package and how it is used.

3.1 Optimization

There has been some previous work on computer-aided optimization of power electronics. In [12], an optimization routine is coupled with a circuit simulator to select parameters for a power converter filter. The objective function to be minimized is a predetermined function of the filter inductance, capacitance, and output voltage ripple.

The circuit simulator ACSL is used to predict the ripple performance of a design, while a deterministic outer loop searches for the objective function minimum.

In [13] the joint optimization of the converter structure and control behavior is formulated as a constrained optimization problem and solved using a numerical optimization procedure. The structural objective of the optimization is formulated in terms of converter losses or efficiency, which are based on simplified closed-form calculations [14], while the control cost is based on a formulation of the control effort and performance. The authors apply this approach to the design of a buck converter, where the plant design parameters are the converter inductance and capacitance, with other plant parameters fixed. A major disadvantage to this approach is that it requires a very detailed mathematical formulation of the optimization problem.

In [15], computer-aided optimization of a three-phase inverter is considered. The objective function of the optimization is formulated in terms of efficiency or temperature rises. Analytical expressions for converter losses and temperature rises are developed and used for the necessary calculations. The optimization is done via a stepped search across two design parameters (gate resistance and gate drive voltage) with all other design choices fixed.

While having a number of distinct aspects, the existing investigations have a number of important characteristics in common. First, in each case, only a very small number of parameters are searched over, meaning that only a small fraction of the entire design space is explored. It is not clear to what extent the optimization techniques and problem formulations extend to a more extensive optimization. Second, all of the techniques consider numerical parameters only, without bridging the gap to actual components. For example, [12, 13] search for optimized inductance values, but there is no tie to actual component design or consideration of actual component non-idealities. This makes it difficult to optimize for quantities such as price, weight, and volume, and does not address many of the issues found in an actual design. Finally, all of these investigations focus primarily on the computations and formulation aspects of optimization. However for a CAD optimization tool to be useful across a large design space and to address component implementation, user interface and data management issues become very important. It is the authors' belief that these issues need to be addressed in a fully functional optimization tool.

The TimCAD program explores the design space to identify the converter design or set of designs that best minimize a cost function that is the weighted sum of converter price, weight, and volume. The design variables considered by the program include the number of interleaved cells, the cell switching frequency and ripple ratio, input and output filter topologies and designs, and a wide variety of component types. The program designs the converter with specific parts, each with its own price, weight, volume, and operating characteristics. Detailed models are used to very accurately predict the system behavior with these specific parts. The output of the program thus consists of converter designs that are composed of available components and have well-defined performance characteristics.

The optimization program uses a Monte Carlo design approach. The program chooses random starting points in a user-defined design space and designs converters that meet user-defined specifications. After designing a large number of converters across the possible design space, the results can be analyzed to determine which converter properties and designs are the most desirable. The program can be reconfigured to examine local minima in the design space to further refine the search.

3.2 Program Structure

TimCAD is divided into five sections as shown in the block diagram in Fig. 3.1. These five sections are the user interface, the control loop, the database, the design algorithm, and the device models.

3.2.1 User Interface

The user interface of TimCAD is dialog based, i.e., the interface is made up exclusively of dialog boxes. The toolbar and the menu of the main window can call up any of these dialog boxes. There are three main types of dialog boxes in TimCAD: parts, control, and results. For every major component of the converter there exists a parts

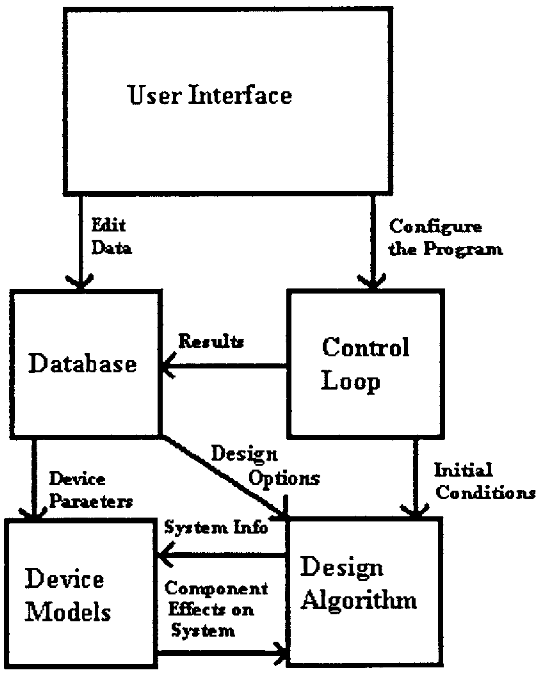


Figure 3.1 Block diagram of TimCAD showing the five major elements of the program.

dialog box. The parts dialog box provides the ability to examine or edit the properties of those parts within the database.

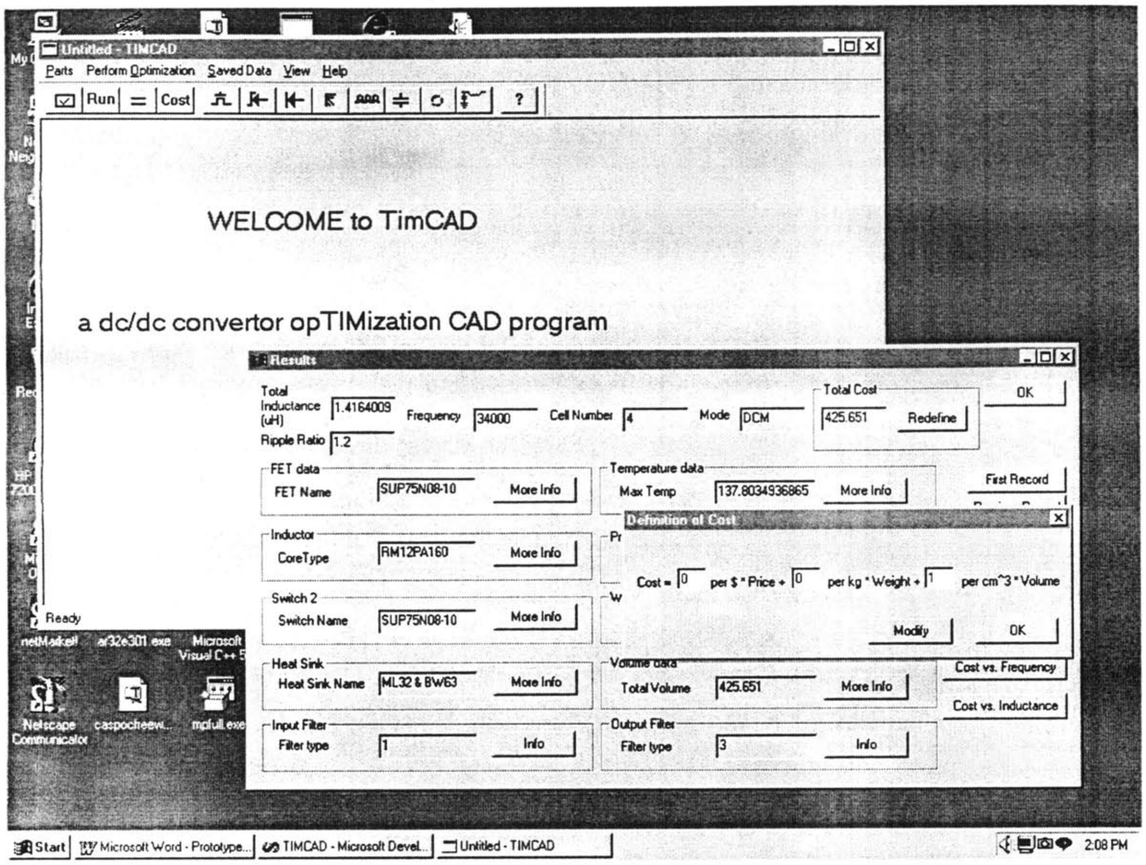
The control dialog boxes, *Configuration* and *EMI Limits*, allow the user to set options that will control the design algorithm or set design guidelines for several of the components. These options include the range of allowable current ripple, the number of cells, the maximum power loss of a component, the Q of the filters, and many other options.

The results dialog boxes, *Results* and *Cost*, are able to manipulate the output of the program. The *Results* box, as seen in the screenshot of Fig 3.2, displays all the necessary information about the design of a dc/dc converter. The records accessed from this dialog box are ranked based on a cost function. The cost function is a specified weighted average of the price, weight, and volume of the converter.

3.2.2 Control Loop

The control loop is the main element in the program which regulates how the design space is searched. Between any two successive designs of the converter many different variable can change. The control loop controls the program by determining which of these designs to use on each successive design of the converter. The control loop uses a Monte Carlo design algorithm to randomly select from a number of design parameters and components that are provided to the design algorithm. The design space considered and the number of iterations used are key factors in the control loop, and can both be changed through the use of the *Configuration* Dialog Box.

To search the design space rapidly, TimCAD actually uses a multi-layer control loop. An outer control loop generates design with randomly selected parameters



WELCOME to TimCAD

a dc/dc convertor opTIMization CAD program

Total Inductance (uH) 1.4164009 Frequency 34000 Cell Number 4 Mode DCM

Total Cost 425.651

Ripple Ratio 1.2

FET data: FET Name SUP75N08-10

Temperature data: Max Temp 137.8034936865

Inductor: CoreType RM12PA160

Switch 2: Switch Name SUP75N08-10

Heat Sink: Heat Sink Name ML32 & BW63

Volume data: Total Volume 425.651

Input Filter: Filter type 1

Output Filter: Filter type 3

Cost = 0 per $ * Price + 0 per kg * Weight + 1 per cm^3 * Volume

Figure 3.2 Screenshot of TimCAD

controlling the power stage design. An inner control loop designs multiple sets of input and output filters (again using the Monte Carlo methods) for each power stage generated by the outer control loop. The filters with the lowest weighted sum of price, weight, and volume are then chosen for the design of the converter. The number of outer loop iterations (which are the most computationally expensive) can then be reduced since the fast inner loop will design the optimized filters on every iteration of the outer loop.

3.2.3 Design Algorithm

The design algorithm is the section of the program that designs the converter based on the parameters provided by the control loop. The control loop of the program randomly selects initial design parameters including the number of cells, switching frequency, cell ripple ratio, filter topology and the selection of several components. The design algorithm uses these values as initial conditions and determines essential values of the design, chooses components to ensure proper operation of the converter, and makes sure a valid design is achieved. The relationship between the control loop and the design algorithm is shown in Fig. 3.3. To generate information necessary for the design of other parts of the converter, the design algorithm uses models of the converter components. For example, the models for the switches are used to calculate the power loss in the devices, which is necessary for the design of the heatsink. If a component can not be found to adequately meet one of the specifications, then the iteration is declared invalid and the

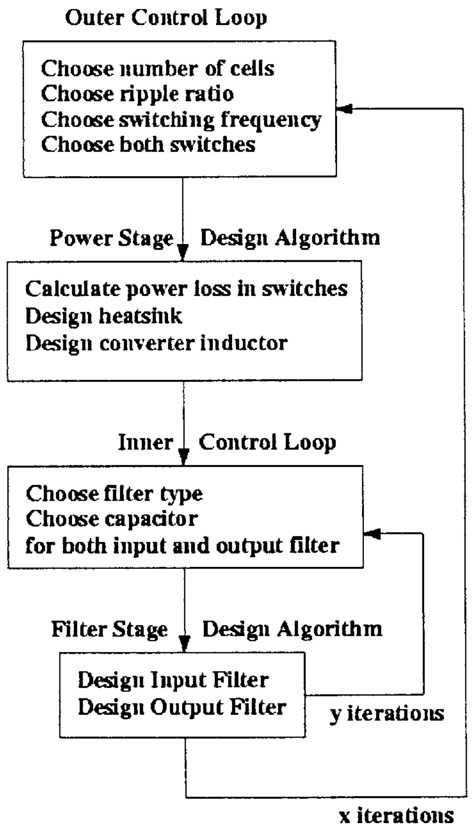


Figure 3.3 Relationships between the control loop and the design algorithm. The outer loop calls the power stage design algorithm. After the power stage is designed the inner control loop runs and the filter stage is designed. The inner control loop iterates y times and the outer control loop iterate x times.

control loop chooses another point in the design space to analyze. Once a design is successfully completed, the program will calculate the total price, weight and volume of the valid design and save this record in a database.

3.2.4 Device Models

An object of code exists for every component used in TimCAD. Within each of these objects is a model for that component describing the necessary aspects of its design and use. When supplied with data from the design algorithm, the code for the component will calculate and relay to the design algorithm all the relevant information about the chosen component.

3.2.5 The Database

The program uses a *Microsoft Access* database to store all essential data. The database is comprised of 23 tables separated into two different files. The *Parts List* file contains a table for every component used in TimCAD with all the properties needed for that component. Three other tables in the *Parts List* file, *EMI Limits*, *Configuration1*, and *Configuration2*, store information on user-defined variables for the converter. In the table *EMI Limits*, each record contains the EMI limit and frequency that define each corner of the EMI specification for up to 30 megahertz. All the variables used to configure TimCAD (listed in Table 3.1) are stored in the *Configuration1* and *Configuration2* tables. The configuration data is separated into two tables because the number of fields in any database that interacts with another program is limited to forty. Along with other configuration data, the *Configuration2* table also stores the constants for the cost function and the information determining which filter topologies to consider.

The *Microsoft Access* file, *Results*, contains all of the output from the program.

Field	Description
Vin min	Minimum voltage on the input bus
Vin max	Maximum voltage on the input bus
Vout min	Minimum voltage on the output bus
Vout max	Maximum voltage on the output bus
Switching frequency min	Minimum switching frequency for the power devices
Switching frequency max	Maximum switching frequency for the power devices
Switching frequency step	The increment of possible switching frequencies
Ripple Ratio min	Minimum ripple ratio for one cell at nominal voltages
Ripple Ratio max	Maximum ripple ratio for one cell at nominal voltages
Ripple Ratio step	The increment of possible ripple ratio
Linear distribution	Determines whether the possible values of ripple ratios is linearly or exponentially distributed
Number of cells min	Minimum number of cells to consider
Number of cells max	Maximum number of cells to consider
Filter 1 Specs Q min	For Filter type 1, the minimum Q to consider
Filter 1 Specs Q max	For Filter type 1, the maximum Q to consider
Ambient Temperature	The ambient temperature that the converter must be designed for
Max Temperature	The maximum temperature of any device in the converter
Average Current	The maximum average current that the converter must deliver to the low voltage bus
FET gate drive current	The average current that the gate drivers use to switch the MOSFETs
Capacitor max	The maximum number of capacitors it place in parallel at the ports of each cell
PC board Price	The price of the PC board in $/cm^2
Rcs	Thermal resistance of material between the case of the device and the heatsink
Outer loop Iterations	The number of runs TimCAD will perform
Inner loop Iterations	The number of filter designs that will be examined for every outer loop iteration
Continuous Mode	Allows designs for continuous mode operations
Discontinuous Mode	Allows designs for discontinuous mode operations
Switch 2 – FET	Allows a MOSFET to be used as the second switch
Switch 2 – Diode	Allows a Diode to be used as the second switch
Switch 2 – Schottky Diode	Allows a Schottky Diode to be used as the second switch
Output filter simple windings	The output EMI Inductor has a simple or complex winding
Output filter max power loss	The maximum power loss acceptable in the output EMI inductor
Input filter simple windings	The input EMI Inductor has a simple or complex winding
Input filter max power loss	The maximum power loss acceptable in the input EMI inductor
Master control price	Price for the control that controls the entire converter
Cell control price	Price for the control that controls an individual cell
Master control weight	Weight for the control that controls the entire converter
Cell control weight	Weight for the control that controls an individual cell
Master control volume	Volume for the control that controls the entire converter
Cell control volume	Volume for the control that controls an individual cell
Max allowable price	Maximum price for valid designs
Max allowable weight	Maximum weight for valid designs
Max allowable volume	Maximum volume for valid designs
Use Max Values	Toggles if maximum values limits are used

Table 3.1 Attributes in the *Configuration* Dialog Box

Again the results from the design must be split into four tables to limit the number of fields per table to 40. The *Results* table contains the price, weight, and volume information about the converters. The *ResultsB* table contains information about the power stage whereas *ResultsC* and *ResultsD* store information about the filter stages. For every results table there are also two tables created to store saved results. TimCAD can copy the current results of any optimization process to either of these saved positions.

3.3 Program Objects

TimCAD is an object-oriented program. An object-oriented program can be viewed as a collection of separate subprograms, or objects, that are linked to other objects to create the full program. There are three main templates for the objects used in TimCAD: CRecordset, CDialog, and CDocument.

3.3.1 CRecordset Objects

The CRecordset class is the object that directly interfaces with the database. Each table used in the database has a corresponding CRecordset associated with it. Code within these classes deals with the components associated with the table they represent. For example, the CRecordset associated with MOSFETs, named *CFETSet,* contains the code that models the MOSFET. When designing the converter, the code will send the *CFETSet* object several parameters and this object will calculate the power loss of the device. The code responsible for modeling all the components are fully contained in these CRecordset classes. Along with code specific to the component, these objects contain the code that link the program to the individual database tables. This code that interacts with *Microsoft Access* allows the program to search and sort data within the database.

3.3.2 CDialog Objects

CDialog classes are the interface elements between the user and the program. For every dialog box there is a CDialog class that controls its operation. This class reads and writes information from the CRecordset class associated with the CDialog class. This class controls all the user interactions such as displaying and editing the database records.

3.3.3 The CDocument Object

The most prominent class in the program is the CDocument class, even though only one CDocument object, named *CTIMCADDoc,* exists in the program. This object contains the design algorithm and the outer control loop. When the optimization algorithm is started this object receives the command and proceeds to start the outer loop. The program will then carry out the commands in the design algorithm and will interface with the device models in the CRecordset objects as needed.

3.4 Program Operation

3.4.1 Installing TimCAD

To install TimCAD on any Windows 95 or 98 computer, one moves the files *TIMCAD.exe, Parts List.mdb,* and *Results.mdb* onto the computer. The next step is to use the **ODBC** program to make the database accessible to various other programs. From the **Start Menu** choose **Settings**, then **Control Panel**. Then, start the **ODBC** program. On the *User DSN* tab choose **Add** and then select the **Microsoft Access Driver** from the *Create New Data Source* box. On the *ODBC Microsoft Access 97 Setup* box choose **Select** and find the *Parts List.mdb* file and choose **OK**. Then repeat this process for the *Results.mdb* file. The program should then be ready to run.

3.4.2 Starting Up

Use of TimCAD involves defining a design space of several variables and designing dc/dc converters randomly in that design space. The resulting converters are evaluated in terms of the price, weight and volume in an attempt to find the minimum-cost design.

Before beginning an optimization run in TimCAD the setup of the program should be examined. This includes the parts database and the configuration options. The parts database, accessible through the toolbar and menu, contains all the components within the database and all the information needed to model each component. (A complete parts database is important. With a well stocked database, the selected components will be closer to optimum, and fewer designs will be rejected for lack of a suitable part.) A component will be ignored by the optimization routine if the *Using* checkbox is unchecked. Most components are either chosen randomly or by matching a component with a desired input value (i.e. choosing the appropriate capacitance for a filter design) however the choice of converter inductor cores and heatsinks are made differently. Each of these components is given a rank. The design algorithm will attempt to use the component with the lowest rank; if the design fails with this component, the next component is used. The ranking of these components can be automatically set based on increasing values of price, weight or volume, or on a customized criterion.

In order to edit properties of the optimization process use th7e *Configuration*, *Filters*, and *EMI Limits* Dialog Boxes. The *Configuration* Dialog Box sets most of the user defined constants in the program, such as number of iterations and maximum current rating. From here the optimization program can be set to optimize over a range of frequencies, number of cells, and ripple ratios. The range to be used is set with the *Configuration* Dialog Box. A full list of attributes of the *Configuration* Dialog Box is shown in Table 3.1. The *Filters* Dialog Box allows the user to examine and chose which filter topologies to consider in the converter design. The *EMI Limits* Dialog Box sets the EMI limits for both the input and output bus by listing the EMI limit at every frequency where the EMI limit changes.

3.4.3 Running TimCAD and Data Manipulation

To start the optimization routine chose *Run* from the toolbar or from *Perform Optimization* in the menu. The program will give you an option to erase the current data or to append the new designs to the database. Choosing either of these will bring up a progress meter that indicates how much time is need for the program to complete the process.

When the optimization routine has completed, the data from TimCAD can be stored in two different saved positions (using *Microsoft Access,* the data can be moved more quickly). The *Saved Data* option on the menu bar contains the options for saving or restoring the data. The data should be saved before any of the data manipulation routines are used, since these routines will delete records.

The *Results* Dialog Box displays all the valid converter designs ranked by a weighted sum of their price, weight, and cost. (The formulation of this weighted sum can be changed using the *CCost* Dialog Box, which is accessible from either the *Results* Dialog Box or the menu.) If the information in the *CCost* Dialog Box is changed at any time, all the records within the database will be updated. This will not necessarily result in the optimal design for the new cost function since the inner control loop will have chosen the filters based on the preexisting cost function.

The Results Dialog Box will list all the components used in the design and design information about the components such as power loss and temperature rise. The buttons *Cost vs. Frequency* and *Cost vs. Ripple Ratio* will delete all the records except for those with the lowest cost for each frequency or ripple ratio.

Once the database has been cleared of all the extra data, the results can be viewed using *Microsoft Access*. The results of the optimization routine are stored in the tables, *Results*, *ResultsB*, *ResultsC*, and *ResultsD*. The query *Results Query* selects from those four tables several of the interesting fields. The forms *CostvsFreq* and *CostvsRippleRatio* show plots of the fields in *Results Query* across the range of either frequencies or ripple ratios.

3.5 Conclusions

TimCAD is a powerful new tool for the design of dc/dc converters. Not only can it be used to find the optimal design for a given set of system parameters, it can also be used to examine the effects of varying these parameters on the characteristics of the optimal design. TimCAD can thus determine the sensitivity of converter cost to variations in system parameters such as ambient temperature or EMI limits.

To learn more about the operations of TimCAD, examine appendices D and E. Appendix D contains the functions within TimCAD that deal with the design of the converter. Flow charts of major functions used in TimCAD are in Appendix E.

Chapter 4

Optimization Results

This chapter shows some results of using TimCAD to search the design space for optimal converter designs. The design variables that are searched across include the number of cells, switching frequency, ripple ratio, filter topology and component choices. One reason for doing this is to identify designs and design approaches that are more optimal then what is easily achieved by hand. Furthermore, by comparing optimized designs having different design specifications, the sensitivity of the converter size, weight, and price to these specifications can be determined.

4.1 Limitations of the Results

Before presenting the results, it is important to state the two major deficiencies that limit the utility of the existing simulation runs: the low number of components in the database and the lack of high-volume pricing data. A sparse database of parts leads to two problems: first, designs will fail because no suitable components could be found. Second, converter designs may have oversized components because the database does not contain smaller components that are suitable for the design. The use of more components in the database results in more possible converter designs and therefore more opportunities to generate a better converter design. Table 4.1 shows the number of components used for the optimization runs described in this chapter. This database is considered dense enough for a preliminary exploration of the design space, but the

Table	Number of Entries
Capacitors	34
Converter Inductor Cores	23
PiN Diodes	2
EMI Inductor Cores	26
Heatsinks	43
MOSFETs	5
Schottky Diodes	1

Table 4.1 Number of components of each type in the database.

addition of more components would be valuable.

The pricing data used in the database are the prices provided by distributors and manufacturers for small quantities of components. The converter prices formulated by TimCAD are a summation of the component prices. Since the prices quoted by manufacturers can vary depending on the purchaser and the quantity ordered, the resulting price information will be slightly skewed. Its is not possible to use a constant factor to translate between the calculated price and the price for another buyer since the disparity in pricing is not the same for different components. Thus, to use TimCAD in obtaining a price-optimized converter with pricing applicable to industry, the industrial price of every component must be entered on the database and the optimization redone. To bypass this issue, the preliminary results in this chapter do not contain pricing data or price optimization.

4.2 The Baseline System

In most of the cases that follow, a system under test will be compared to the baseline system described below. The settings in the *Configuration* and *Filter* dialogs boxes used in this system are listed in Table 4.2 and the EMI limits used are in Table 4.3. The voltage limits for all studies are described in Appendix A. The optimization runs use all the parts possible, assuming the converter is bi-directional and free-convection cooling is being used.

Category	Baseline Value	Category	Baseline Value
Vin min	33 V	Inner loop Iterations	30
Vin max	52 V	Continuous Mode	TRUE
Vout min	12 V	Discontinuous Mode	TRUE
Vout max	16 V	Switch 2 – FET	TRUE
Switching frequency min	10,000 Hz	Switch 2 – Diode	FALSE
Switching frequency max	500,000 Hz	Switch 2 – Schottky Diode	FALSE
Switching frequency step	2,000 Hz	Output filter simple windings	FALSE
Ripple Ratio min	.10	Output filter max power loss	5 W
Ripple Ratio max	1.50	Input filter simple windings	FALSE
Ripple Ratio step	.01	Input filter max power loss	5 W
Linear distribution	TRUE	Master control price	5$
Number of cells min	3	Cell control price	5$
Number of cells max	6	Master control weight	4 g
Filter 1 Specs Q min	.5	Cell control weight	2 g
Filter 1 Specs Q max	2	Master control volume	2 cm^3
Ambient Temperature	100 °C	Cell control volume	4 cm^3
Max Junction Temperature	140 °C	Max allowable price	9999999 $
Average Output Current	68 A	Max allowable weight	9999999 kg
FET gate drive current	1.5 A	Max allowable volume	2000 cm^3
Capacitor max	2	Use Max Values	TRUE
PC board Price	.18 $/cm^2	Filter 1	TRUE
Rcs	.5 °C/W	Filter 2	TRUE
Outer loop Iterations	20,000	Filter 3	TRUE

Table 4.2 Baseline values for the *Configuration* and *Filter* Dialog Boxes.

	Frequency (Hz)	Limit (dB μV)
1	150000	90
2	300000	90
3	530000	66
4	2000000	66
5	5900000	57
6	6200000	57
7	30000000	52

Table 4.3 Baseline values for the *EMI Limits* Dialog Box. Each line represents a corner on the EMI limits diagram.

4.3 A Design Comparison

To get a sense of what is achievable with the program, consider a comparison between the prototype converter shown if Fig 2.10 and a volume-optimized design generated by TimCAD that meets the same specifications. The prototype converter, which was not optimized for volume, has a displacement volume of 1740 cm^3. Accounting for the extra volume of components such as heatsinks that were deliberately oversized, a displacement volume of 1330 cm^3 reflects the design of the prototype. For comparison, consider that a volume optimized design generated by TimCAD has a displacement volume of only 210 cm^3. Part of this tremendous volume reduction is due to the program selecting much more volumetrically-efficient capacitors then used in the prototype, and part of it is due to the program identifying a better design point for the converter. Even allowing for the use of the volumetrically efficient capacitors, the prototype design would have a volume of 1000 cm^3, much larger than that achieved with the optimized design. This illustrates what kind of results are achievable using a CAD optimization program over conventional hand design.

4.4 Identifying Design Trends

In addition to optimizing designs, TimCAD can be used to study trends in the design space. For example, one may wish to know what range of cell switching frequencies yield the best designs. To do this, one can do an optimization run, and save the best designs at each switching frequency. The results of plotting the volume of the lowest-volume converter at each cell switching frequency vs. switching frequency for the baseline specifications are shown in Fig. 4.1. At very low frequencies the cell inductors

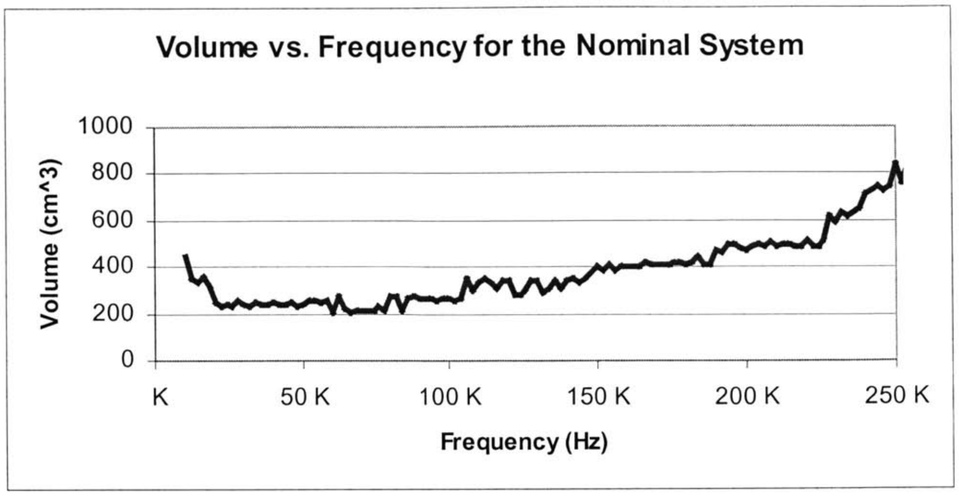


Figure 4.1 The effect of frequency on the volume of the converter for the default system.

and filters become large, increasing converter size. At higher frequencies the power losses of the devices increase, and therefore the volume of the heatsink increases, again making the converter large. In between, there is a broad minimum centered at 75 kHz.

4.5 The Effects of Power Rating

One investigation performed with TimCAD is an examination of how the size of a volume-optimized converter varies with power rating. To vary the power rating of the converter for successive optimization runs, the output current (a variable in the *Configuration* Dialog Box) is changed.

Figure 4.2 shows how the volume of a volume-optimized converter varies with specified power level. (Except for the output current, the specifications of the baseline converter were used.) Within the range of 250 W – 1000 W, doubling the power rating of the converter results in approximately a factor of 1.8 increase in converter volume. This indicates that the power density of an optimized converter increases with power over this range. From this data, the required displacement volume (of converter parts) for a

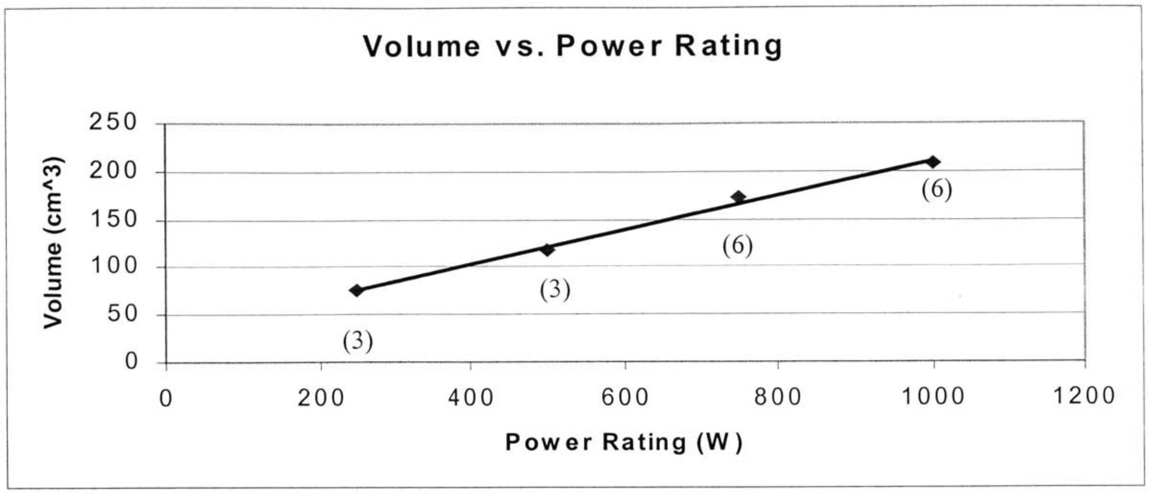


Figure 4.2 The volume of the smallest converter at various power rating. Values in parenthesis represent the number of cells in the volume-optimized design.

volume-optimized converter in this power range is estimated as

$$\text{Volume (cm}^3) = 0.18 \cdot \text{Power (W)} + 29.88 \qquad (4.\ 1)$$

The main trend for the volume-optimized converters is that heatsink volume is always minimized. The heatsink volume makes up a large portion of the total volume of the converter, thus operating conditions that result in low loss and small heatsinks are always chosen. The volume-optimized converters generally have a switching frequency in the vicinity of 75 kHz and a ripple ratio of approximately .45, regardless of power level. From the data collected in Fig. 4.2, the number of cells decreases as the power level decreases. Assuming the best device is always used, using fewer cells in the converter will raise power dissipation in each cell (because each cell must carry more current) whereas converters with a lower power rating will have less power losses. (In the volume-optimized design the device with the largest die is always used, in a price-optimized design this will be different). The net result is that the amount of power dissipation, and therefore heatsink volume needed, decreases slightly as the power level decreases. At lower power ratings the magnitude of the ripple current will generally decrease, but using fewer cells will lessen the benefits of the interleaving process. Thus the filters tend to (but not always) get smaller as the power level decreases. Note that the volume of a cost-optimized converter may have very different characteristics from the volume of a volume-optimized converter. When designing the volume-optimized converter important characteristics such as the volume of the heatsinks take precedence over choosing small, but expensive elements such as capacitors. This results in designs that may have very small heatsinks but use many small and expensive elements. A price-optimized converter will minimize the use of these expensive elements at the cost of using a larger heatsink.

Figure 4.3 shows the relationship between the weight of the volume-optimized converter and power rating. As expected the weights increase with power level, with an approximate relationship of

$$\text{Weight (g)} = 0.20 \cdot \text{Power (W)} + 83.72 \tag{4. 2}$$

These results should not be confused with those for weight optimization. For the 1000 W

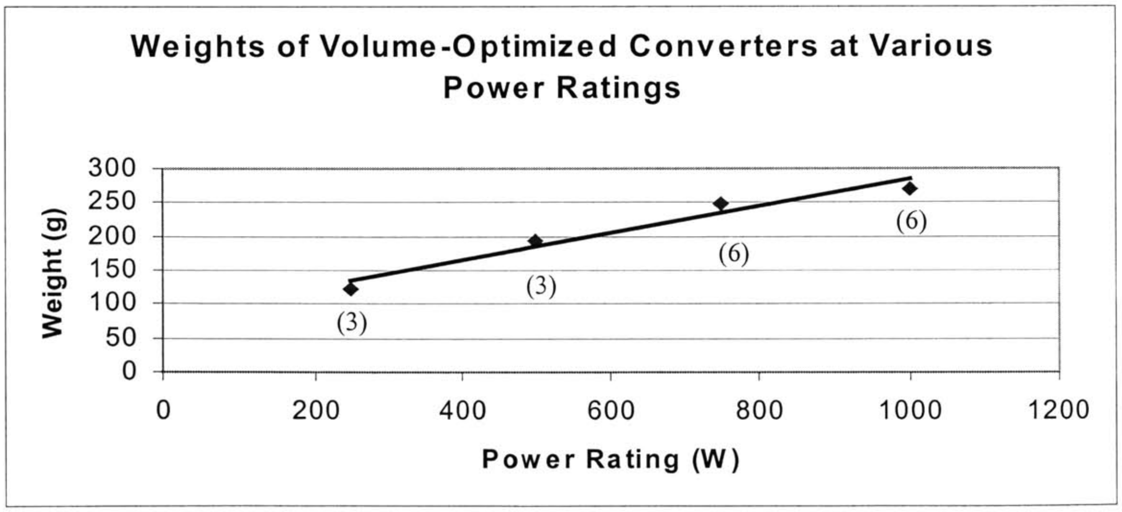


Figure 4.3 The relationship between the weights of the volume-optimized converters and the power rating. Values in parenthesis represent the number of cells in the volume-optimized design.

converter the EMI filter inductor makes up 2 % of the volume and 8 % of the weight of the converter's components. In weight-optimized converters the size of the magnetic elements rather than the heatsinks will be minimized.

4.6 The Effects of Ambient Temperature

The ambient temperature the converter must operate in affects many of the components. The sizing of some components, such as semiconductors, inductors, and heatsinks, is often determined by maximum temperature constraints. These elements can be designed more freely at low ambient temperatures. Other components must be selected directly based on ambient temperature. For example, film capacitors suffer from a severe voltage derating at high temperatures, necessitating the selection of larger, more expensive capacitors at high temperatures.

Figure 4.4 shows how the volume of a volume-optimized converter varies with both the specified rating level and the ambient temperature. Aside from these parameters,

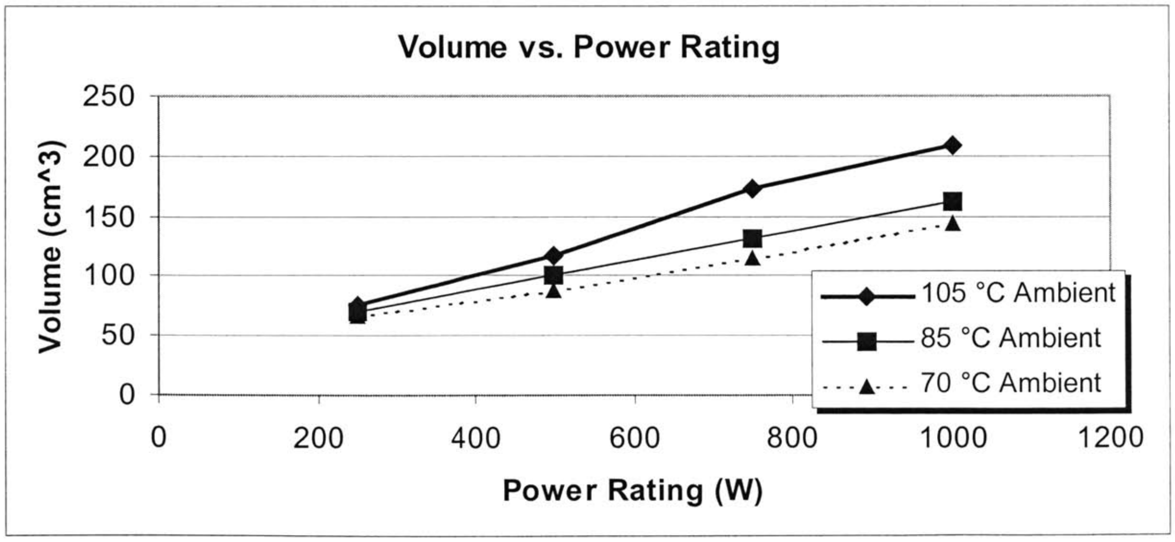


Figure 4.4 The volume of the smallest converter at various power rating for three different ambient temperatures.

the design parameters of the baseline system were used. One important parameter is the maximum allowed junction temperature, set to 140 °C in this case. (Most modern devices have a nominal rating of either 150 °C or 175 °C, but there is also a tradeoff between maximum temperature and lifetime, so 140 °C is a reasonable, conservative design limit for 175 °C devices.) For the 1000 W case, the volume increase by 10 % from 70 °C to 85 °C, and by 22 % as the temperature goes from 85 °C to 105 °C.

4.7 The Effects of EMI Limits

The SAE J1113/41 EMI specifications list five different classes of narrowband Electromagnetic Interference (EMI) specifications. The EMI limits specified in Class 1 are listed in Table 4.4. Each successive EMI specification is at most 10 dB lower than the previous set of limits. Thus, Class 3 and Class 5 EMI specifications, listed in Table 4.4, are as much as 20 dB and 40 dB lower than the Class 1 specification. Figure 4.5 shows this relationship.

Applying different EMI specifications will affect the size of the required EMI filters. As more stringent EMI limitations are applied to the voltage busses, the size of filter capacitances and inductances will grow, resulting in an increase in converter price, weight, and volume. Table 4.5 shows the volume of the input and output filters for six

	Frequency	Class 1 Limits	Class 3 Limits	Class 5 Limits
1	150000	90	70	50
2	300000	90	70	50
3	530000	66	50	34
4	2000000	66	50	34
5	5900000	57	45	33
6	6200000	57	45	33
7	30000000	52	40	28

Table 4.4 Class 3 and Class 5 SAE J1113/41 narrowband specifications.

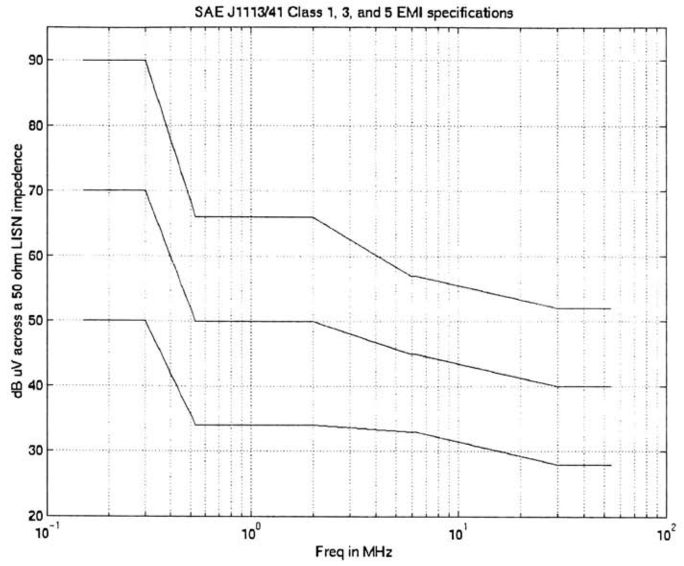


Figure 4.5 Classes 1, 3 and 5 SAE J1113/41 narrowband specifications.

converters designed for the baseline specifications with various numbers of cells, switching frequencies, and ripple ratios. Filters for each system were optimized for three different EMI specifications. In general, as the EMI limits decrease from Class 1 to Class 5, the filter volumes increase. There are several cases in which the filter volume does not change. This indicates that the filter for the lower class has been oversized since it will meet higher-order EMI limits. Filters are oversized when the program cannot construct a more optimal (smaller) EMI filter due to lack of parts in the database.

The optimization results from the baseline system were compared to the results of

Operating Conditions			Input Filter Volume Class 1	Output Filter Volume Class 1	Input Filter Volume Class 3	Output Filter Volume Class 3	Input Filter Volume Class 5	Output Filter Volume Class 5
N	Switching Frequency, f_{sw}	Ripple ratio						
4	68000	.4	9.111	6.5	14.691	12.626	23.045	19.577
5	132000	.72	10.033	8.125	10.0333	10.033	16.45	16.45
4	186000	.82	9.111	9.111	9.876	9.111	19.684	11.683
5	226000	1.36	10.033	10.033	14.643	10.033	19.636	13.101
6	278000	.83	10.955	10.955	14.606	10.955	20.138	18.331
6	440000	.19	12.762	10.955	17.058	10.955	23.324	15.251

Table 4.5 The effects of EMI limitation on filter size.

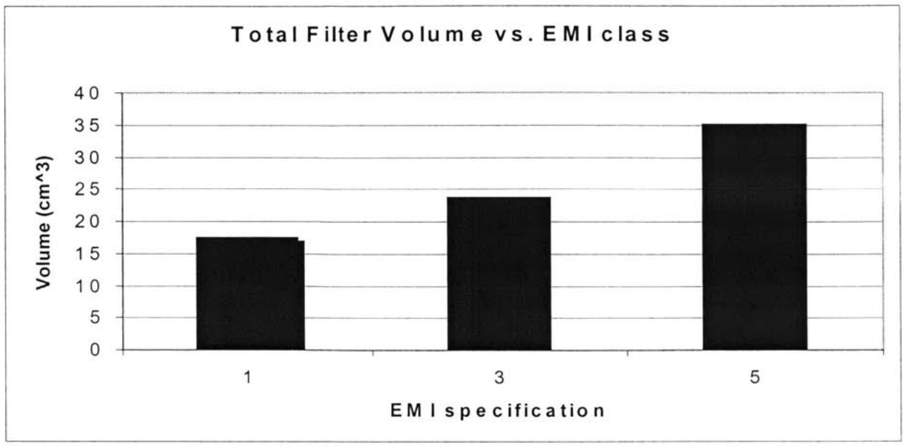


Figure 4.6 This graph shows the affects of EMI specification on the volume of the input and output filters.

systems with Class 3 and Class 5 EMI specifications to find the effects of EMI specifications on the size of the filters needed. As can be seen in Fig. 4.6, the required filter volume approximately doubles when specifications are changed from Class 1 to Class 5.

These filters have all been optimized with respect to volume. The capacitors used are very volumetrically efficient, thus all the filter designs are weighted toward using capacitance rather then inductance. For a price-optimized design the opposite will be true since inductor cores tend to be less expensive than capacitors.

4.8 Conclusions

This chapter illustrates the use of TimCAD to search the design space for optimal designs. It has been shown that TimCAD yields designs that are better optimized then the prototype converter described in Chapter 2, and that TimCAD can be used to examine trends in the design space.

System variables such as power level, ambient temperature, and EMI limits affect the design of the converter in many ways. With TimCAD, the effects of changes of these system variables on the converter design can be quantified. The savings obtained by changing the ambient temperature of the converter, for example, can be compared to the cost of moving the converter out from under the hood. This thereby help the designers of the automotive electrical system make informed decisions on major changes in the design.

Chapter 5

Device Models and Heatsinks

Consider the buck converter in Fig 5.1. Switch S1 must be a controlled switch in order to regulate the flow of energy from the high bus to the low bus. Switch S2 must also be controlled if the converter is required to supply energy in both directions. However, if the converter is unidirectional, a Schottky diode or a PiN diode could also be used to implement switch S2. Regardless of which devices are used the converter will need a heatsink to protect the devices from the power they dissipate. This chapter will analyze the device power losses and temperature rises for each of three possible cases: two MOSFETs as the switching elements, a MOSFET and a Schottky diode, and a MOSFET and a PiN diode.[1] This chapter will also examine the heatsinks necessary for the design of the converter.

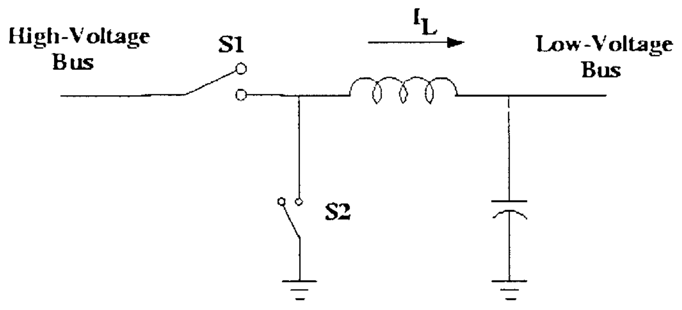


Figure 5.1 Basic cell of a direct converter. S1 and S2 are the switching elements in the converter.

[1] Some of the information for this chapter is derived from [6], [16], and [17].

5.1 Two MOSFETs as Switches

If the two switches are implemented with MOSFETs (see Fig. 5.2) the direction of energy flow between the two busses can be controlled. The use of a bi-directional converter allows the battery on the lower bus to charge the battery on the higher bus. This feature makes a much broader range of energy management routines possible. In case of a discharged high-voltage battery, the converter would be able to use the low-voltage battery to recharge it. The extra switch will, however, require another gate driver and additional complexity in the control of the converter.

There are two main sources of loss in the switching elements: conduction losses and switching losses. Conduction losses that arise in a low-voltage MOSFET are primarily due to the bulk resistance of the device. This loss mechanism is modeled as an on-state resistance, $R_{ds\text{-}on}$, as shown in Fig. 5.3. When the MOSFET is conducting,

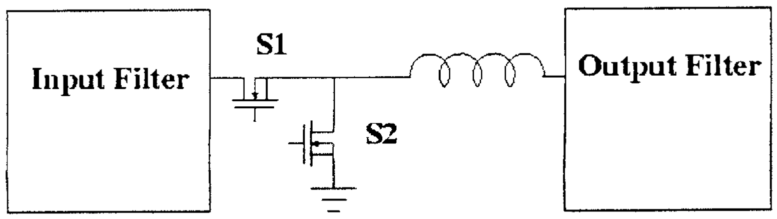


Figure 5.2 The power stage if both switches are implemented with switches.

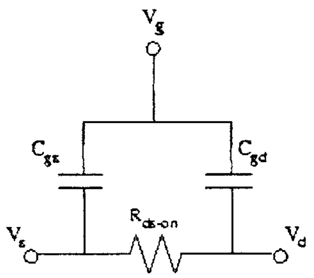


Figure 5.3 Model for the MOSFET in the triode operating region.

current passing through this resistance will cause a voltage drop across the device and incur a power loss of

$$P_{cond} = I_{d\_rms}^{2} \cdot R_{ds-on} \tag{5.1}$$

$R_{ds\text{-}on}$ is dependent on the ambient temperature of the device. For example, at a junction temperature of 100 °C the resistance might be one and a half times its nominal value at 25 °C. The temperature dependence of $R_{ds\text{-}on}$ as given in the MOSFET datasheets appears to be parabolic, therefore the value of $R_{ds\text{-}on}$ used by the program is modeled as a parabolic function of junction temperature.

The other main source of loss is due to the switching of the MOSFET. Whenever the switch changes state, there is a finite amount of time during which the switch will have a significant voltage across it while conducting current. During this interval switching losses will exist. Figure 5.4 shows the switching waveforms of a MOSFET

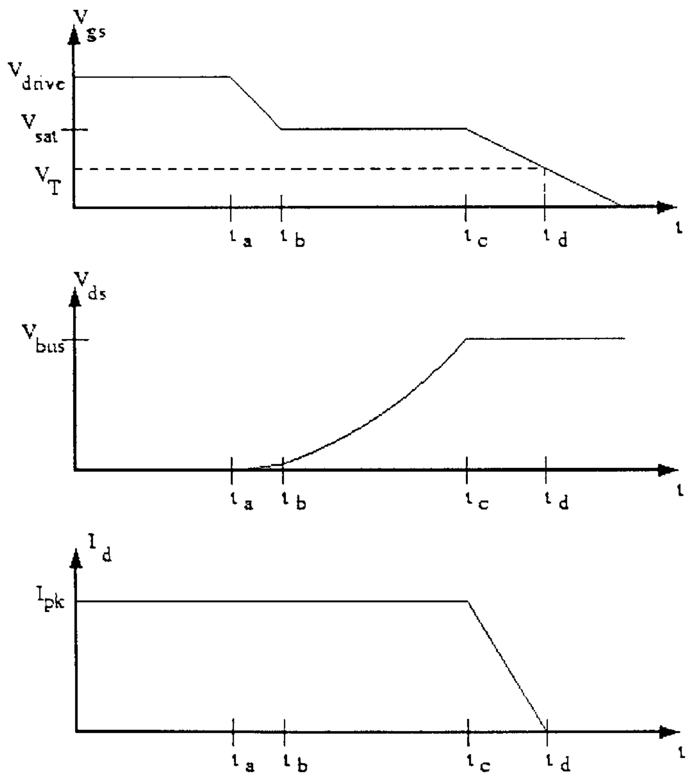


Figure 5.4 Turn-off transition of a MOSFET.

turning off. The gate driver of the MOSFET keeps the gate voltage high while the device is on. When the MOSFET is turned off, the gate driver draws current from the gate of the MOSFET, thereby discharging C_{gs} (the capacitance from the gate to the source of a MOSFET). The gate to source voltage, V_{gs}, is lowered until its value reaches the saturation voltage of the device, calculated as

$$V_{sat} = \frac{I_{pk}}{g_{fs}} + V_T \tag{5.2}$$

This occurs during the time interval t_{ab}, since the current in this time interval is very small the power loss is ignored.

At the gate voltage of V_{sat} at t_b the parasitic capacitance, C_{gd}, between the gate and the drain will begin to charge. The value of this capacitance is not constant in the device. We can model this nonlinear capacitance as

$$C_{gd} = \frac{K_{gd}}{\sqrt{V_{gd}}} \tag{5.3}$$

The value of K_{gd} can be calculated from base device parameters as C_{rss} times the square root of the voltage at which it was measured, where C_{rss} and the voltage are available from the MOSFET's data sheet. Assuming that the gate driver supplies a constant current, the time to charge this capacitance is given by $t_c - t_b = t_{bc}$;

$$t_{bc} = \frac{2 \cdot K_{gd}\sqrt{V_{bus}}}{I_{drive}} \tag{5.4}$$

The voltage across the device then will rise according to

$$v_{ds}(t) = \left[\frac{i_{drive} \cdot (t - t_b)}{2 \cdot K_{gd}}\right]^2 \tag{5.5}$$

Since the current is constant, the energy dissipated is

$$E_{bc} = \frac{2 \cdot I_{pk} \cdot K_{gd}}{3 \cdot I_{drive}} \cdot (V_{bus})^{3/2} \tag{5.6}$$

In the next time interval, $t_d - t_c = t_{cd}$, V_{gs} drops to V_T. During this transition the current through the device will fall to zero. The duration of this time interval is dictated by the time needed for the current to overcome the inductance in the leads and discharge C_{gs}, as well as any ringing that results from these parasitic elements. The full derivation of this time interval can be found in [16].

$$t_{cd} = T_2 + \frac{C_{gs} \cdot (V_{sat} \cdot \cos(\frac{T_2}{\sqrt{L_s \cdot C_{gs}}}) - V_T)}{I_{drive}} \tag{5.7}$$

where L_s is the lead inductance and T_2 is defined as

$$T_2 = \sqrt{L_s \cdot C_{gs}} \cdot \sin^{-1}(I_{drive} \cdot \sqrt{\frac{L_s}{C_{gs}}} \cdot \frac{1}{V_{sat}}) \tag{5.8}$$

and C_{gs} is defined as

$$C_{gs} = \frac{Q_g}{V_g} \tag{5.9}$$

where Q_g is the charge at the gate when there is a voltage V_g on the gate. These values can be determined from the MOSFET's data sheet.

Based on these calculations, the energy lost during this interval is

$$E_{cd} = t_{cd} \cdot \frac{V_{bus} \cdot I_{pk}}{2} \tag{5.10}$$

The total turn off loss for a MOSFET is then

$$P_{turn-off} = (E_{bc} + E_{cd}) \cdot f_{sw} \qquad (5.11)$$

The analysis of the turn-on losses follows a similar argument, but contains several other terms. The energy loss associated with the charging and discharging of the parasitic capacitances in the circuit is defined as

$$E_{on} = \frac{2}{3} \cdot V_{bus}^{3/2} \sum K_{ext} \qquad (5.12)$$

where the values of K_{ext} represent the nonlinear capacitances in the form shown in equation 5.3. These capacitances are a consequence of capacitors external to the device such as the capacitance across the other switch. The graph of the turn-on switch voltage and current shown in Fig. 5.5 is the reverse of Fig. 5.4. The first interval in which losses occur happens when the gate voltage is greater then V_T and less then V_{sat}. In this interval

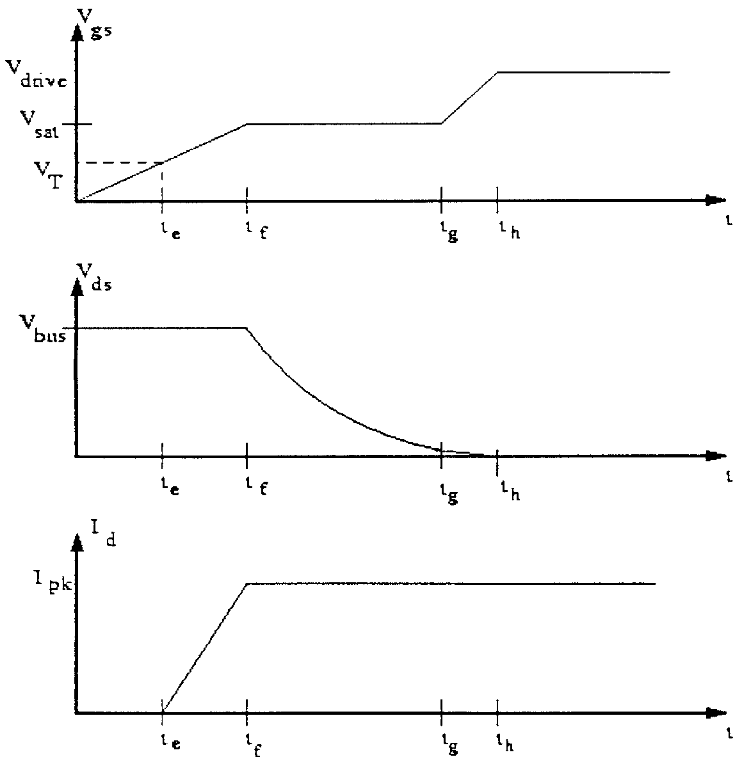


Figure 5.5 Turn-on transition of a MOSFET.

the current will rise to its nominal value while the voltage across V_{ds} will remain constant at the bus value, leading to an energy loss of

$$E_{ef} = \frac{t_{ef} \cdot V_{bus} \cdot I_{pk}}{2} \tag{5.13}$$

over the interval

$$t_{ef} = \frac{(V_{sat} - V_T) \cdot C_{gs}}{I_{drive}} \tag{5.14}$$

The capacitance between the gate and the source can be estimated by dividing the charge on the gate by the gate voltage at the time of the measurement.

When the current reaches its peak value, the capacitance between the source and the drain will start to discharge. This results in an energy loss of

$$E_{fg} = \frac{2 \cdot I_{pk} \cdot K_{gd} \cdot V_{bus}^{3/2}}{3 \cdot I_{drive}} \tag{5.15}$$

Therefore the turn-on power loss of a MOSFET is

$$P_{turn-on} = (E_{on} + E_{ef} + E_{fg}) \cdot f_{sw} \tag{5.16}$$

Since two MOSFETs are used, current can be moving in either direction. If current is flowing out the drain of the MOSFET, then the body diode of the MOSFET will be used to find the device switching losses. This parasitic element is modeled as a PiN diode so the switching losses will only contain the reverse recovery losses in the diode.

The reverse recovery of the PiN diode is caused when the device turns off. When the device turns off the current in the device decreases from the operating current down to the reverse recovery current, and then the current slowly decays to zero during the time t_{rr}. Meanwhile the voltage returns to its nominal value. The total power loss during reverse recovery is a function of the charge on the device given as

$$Q_d = \frac{t_{rr}^2}{4} \cdot \frac{di}{dt} \cdot \frac{(1 + I_{pk} - I_{test})}{I_{test}} \tag{5.17}$$

where the parameters t_{rr}, $\frac{di}{dt}$, and I_{test} are available from the MOSFET's data sheet and the power losses are

$$P_{rr} = Q_d \cdot V_{bus} \cdot f_{sw} \tag{5.18}$$

When the MOSFET is acting as a diode, the capacitance across the device, C_{ds}, will be a cause of power loss in the other MOSFET. This external capacitance will add

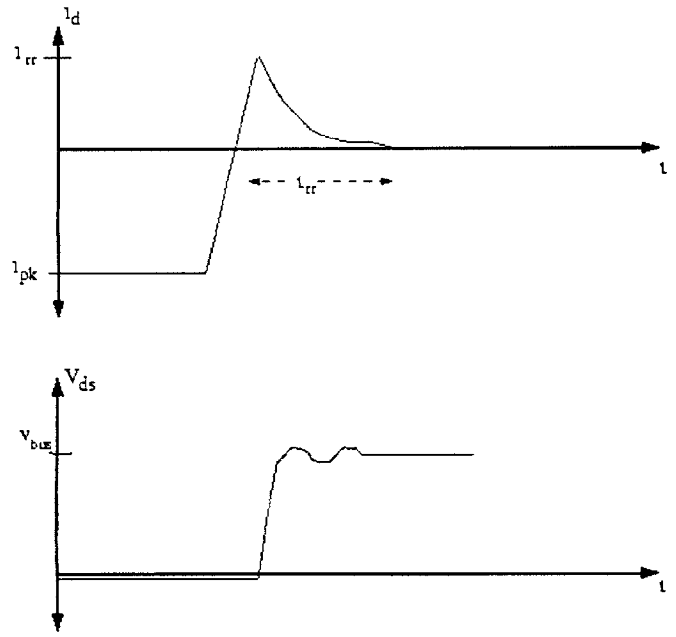


Figure 5.6 Source of reverse recovery loss in diodes.

another term to the summation in 5.12.

Table 5.1 sums up the switching losses for the case of using two MOSFETs. The inductor current, I_L, will normally be greater then zero. However, the inductor current will vary for several different modes of operation. If the converter is operating in discontinuous conduction mode then I_L will be zero for the turn-on transition of switch S1 and the turn-off transition of switch S2, thereby reducing the switching losses for those transitions. When two MOSFETs are used as the switches, the converter can be run such that there will be a reverse energy flow. I_L can be negative for the entire cycle thereby sending energy to the higher bus. Another possible mode is to allow I_L to be positive when switch S1 turns off and negative when switch S2 turns off.

Table 5.2 lists all the parameters used by TimCAD to model the power loss in the device.

5.2 MOSFET and a Diode as Switches

The other option of the converter architecture is to implement the second switch as a diode as in Fig. 5.7. This option restricts the inductor current from going negative, so there can be no reverse energy flow. The diode implementation is useful because the diode doesn't need extra control signals, making the control circuitry simpler.

	$I_L > 0$	$I_L < 0$
Switch S1 turn on loss	$P_{turn-on} = (E_{on} + E_{ef} + E_{fg}) \cdot f_{sw}$	-----
Switch S1 turn off loss	$P_{turn-off} = (E_{bc} + E_{cd}) \cdot f_{sw}$	$P_{rr} = Q_d \cdot V_{bus} \cdot f_{sw}$
Switch S2 turn on loss	-----	$P_{turn-on} = (E_{on} + E_{ef} + E_{fg}) \cdot f_{sw}$
Switch S2 turn off loss	$P_{rr} = Q_d \cdot V_{bus} \cdot f_{sw}$	$P_{turn-off} = (E_{bc} + E_{cd}) \cdot f_{sw}$

Table 5.1 Switching losses for the case of two MOSFETs

Field	Description
MOSFET	The Name of the MOSFET
Package	The casing of the device
Price	The price of the MOSFET
Vdss	The maximum voltage the MOSFET can withstand
Id	The maximum current the MOSFET is rated for
Vt	The threshold voltage
Crss	Reverse transfer capacitance
Coss	Output capacitance
Vds for Crss and Coss	The drain source voltage used when measuring Crss and Coss
Ls	Lead inductance
Gfs	Transconductance
Max Temperature	The maximum temperature of the device
Qg	The Gate charge
Vg for Qg	The gate voltage at which Qg was measured
Tjc	The thermal resistance of the device
Ra	The first coefficient in the quadratic formulation of Rds-on as a function of temperature
Rb	The second coefficient in the quadratic formulation of Rds-on as a function of temperature
Rc	The third coefficient in the quadratic formulation of Rds-on as a function of temperature
trr	The reverse recovery time
If, test	The current used when trr is measured
dI/dt	The current rise used when trr is measured
Using	Determined if the record is used during the optimization routine

Table 5.2 MOSFET parameters used in TimCAD

The losses of the MOSFET are the same as they were in the previous case. Since I_L is non-negative, equations 5.1, 5.11 and 5.16 will describe the power loss of the MOSFET. Note that the capacitance across the diode will add the following term to 5.12

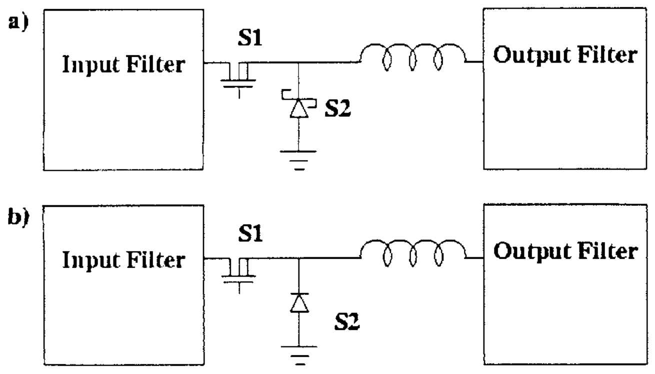


Figure 5.7 The power stage if the second switch is implemented with a) a Schottky diode or b) a PiN diode.

$$K_{diode} = C_T \cdot \sqrt{V_{test}} \tag{5.19}$$

where C_T, the junction capacitance, and V_{test} are values determined from the diode's data sheet.

In order to determine the conduction losses for the diode, the current through the diode and the voltage across the diode must be known. The diode current is the portion of the inductor current that has a negative slope. This current through the diode is known and depends on the inductance, bus voltages and the switching frequency of the converter. The relationship between the diode current and voltage is given in the diode's data sheets. The voltage across the diode is modeled as either a linear or a parabolic function of the diode current, based on the amount of accuracy desired. The conduction power loss is then equal to

$$P_{diode-cond} = \frac{1}{T} \cdot \int_{D \cdot T}^{T} I_{diode}(t) \cdot V(I_{diode}(t)) \cdot dt \tag{5.20}$$

where D is the duty cycle of the converter, T is the period, I_{diode} is the current through the diode as a function of time, and V is the voltage across the diode as a function of I_{diode}.

In this application the recovery losses of a Schottky diode are small and can be safely ignored. But the leakage current of the device may be significant and causes a reverse current power loss of

$$P_{reverse\_I} = I_r \cdot V_{bus} \cdot \frac{t_{off}}{T} \tag{5.21}$$

where I_r is the reverse current associated with the Schottky diode for a given voltage and temperature, and $\frac{t_{off}}{T}$ is the percentage of time that the Schottky diode is off.

The PiN diode doesn't have a significant power loss associated with leakage current, but it does contain a switching loss similar to the loss associated with the parasitic body diode of the MOSFET. The switching loss of the PiN diode is defined as

$$P_{PiN\_sw} = Q_d \cdot V_{bus} \cdot f_{sw} \qquad (5.\ 22)$$

Tables 5.3 and 5.4 list the parameters that TimCAD stores on Schottky and PiN diodes.

5.3 Heat Sinking

In addition to power losses, it is also important to calculate and control device temperature rises. The power lost in the device is converted to heat. The device, standing alone, cannot tolerate the heat it generates when its power loss is only a few watts. To operate the device at its full capacity, one must employ a heatsink. In the program only free-convection cooling is assumed. The addition of fans to the system will increase the cost and lower the reliability needlessly since free-convection cooling is adequate.

Field	Description
Name	The name of the Schottky diode
Package	The casing of the device
Price	The price of the Schottky diode
Vrrm	The reverse voltage the diode can tolerate
If	The current rating of the diode
Rjc	The junction to case thermal resistance
Ct	Junction Capacitance
Vr for Ct	Reverence voltage used when Ct was measured
Ir at max Vin	Reverse leakage current at the voltage in which the device is being used
Max temperature	The maximum temperature the device can tolerate
Detailed modeling	Determines if the I vs. V curve is models as a parabola or piecewise linear
B2	The first coefficient of the I vs. V term for a high order model
B1	The second coefficient of the I vs. V term for a high order model
B0	The third coefficient of the I vs. V term for a high order model
A1	The first coefficient of the I vs. V term for a low order model
A0	The first coefficient of the I vs. V term for a low order model
Using	Determined if the record is used during the optimization routine

Table 5.3 Schottky diode parameters used in TimCAD

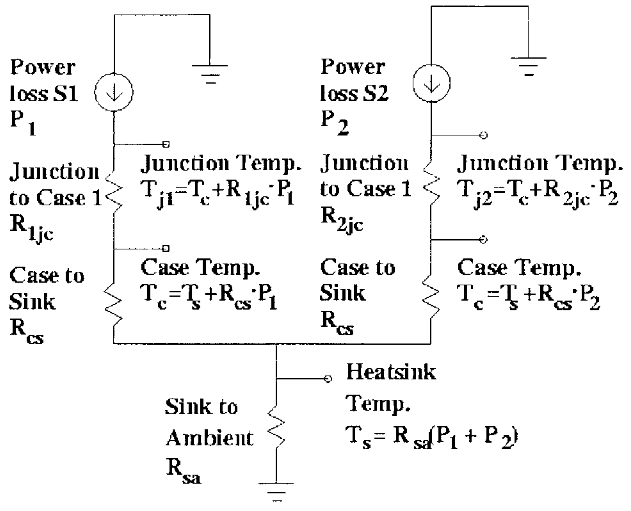
Field	Description
Name	The name of the PiN diode
Package	The casing of the device
Price	The price of the PiN diode
Vrrm	The reverse voltage the diode can tolerate
If	The current rating of the diode
Rjc	The junction to case thermal resistance
Ct	Junction Capacitance
Vr for Ct	Reverence voltage used when Ct was measured
Max temperature	The maximum temperature the device can tolerate
Detailed modeling	Determines if the I vs. V curve is models as a parabola or piecewise linear
B2	The first coefficient of the I vs. V term for a high order model
B1	The second coefficient of the I vs. V term for a high order model
B0	The third coefficient of the I vs. V term for a high order model
A1	The first coefficient of the I vs. V term for a low order model
A0	The first coefficient of the I vs. V term for a low order model
Trr	The reverse recovery time
If, test	The current used when trr is measured
Di/dt	The current rise used when trr is measured
Using	Determined if the record is used during the optimization routine

Table 5.4 PiN diode parameters used in TimCAD

The key feature to model in a heatsink is the thermal resistance. With the thermal resistances of the heatsink, the device, and the material between the objects the junction temperature of the device can be predicted as shown in Fig 5.8. The thermal resistance of the heatsink can be modeled in two ways. Using a low order model, the relationship between the temperature rise and the power dissipation can be assumed to be linear with a zero x-intercept. For some heatsinks the temperature rise is more accurately predicted with a higher order model i.e., as a parabolic function of power loss

$$T_{rise} = R_2 \cdot P^2{}_{diss} + R_1 \cdot P_{diss} + R_0 \qquad (5.\ 23)$$

where R_2, R_1, and R_0 are constants. The level of modeling can be determined for each heatsink.



Power loss S1 P_1
Junction to Case 1 R_{1jc}
Case to Sink R_{cs}
Junction Temp. $T_{j1}=T_c+R_{1jc}\cdot P_1$
Case Temp. $T_c=T_s+R_{cs}\cdot P_1$
Power loss S2 P_2
Junction to Case 1 R_{2jc}
Case to Sink R_{cs}
Junction Temp. $T_{j2}=T_c+R_{2jc}\cdot P_2$
Case Temp. $T_c=T_s+R_{cs}\cdot P_2$
Sink to Ambient R_{sa}
Heatsink Temp. $T_s= R_{sa}(P_1 + P_2)$

Figure 5.8 The thermal model of the devices and the heatsink. In the model, power dissipations are represented as current sources, thermal resistances as resistors, and temperature rises as voltages.

Heatsinks come in a variety of shapes and sizes, with a wide range of thermal resistances. Some heatsinks are designed to hold specific package types. Others are pieces of extrusion whose length can be determined. Regardless of the shape of the

Fields	Description
Name	The name of the heatsink
Rank	The order in which the heatsinks are designed, not used for extruded heatsinks
Volume	The volume of the heatsink
Weight	The weight of the heatsink
Price	The price of the heatsink
R2	The first coefficient of the thermal resistance term for a high order model
R1	The second coefficient of the thermal resistance term for a high order model
R0	The third coefficient of the thermal resistance term for a high order model
Area	The area the heatsink needs on the PC board
Count	The number of devices that can be put on the heatsink, zero indicated an extruded heatsink
Thermal Resistance	The thermal resistance used for a low order model
T0-220	Checked if heatsink can attach to this package type
T0-247	Checked if heatsink can attach to this package type
High Thermal Resistance Modeling	Determines if the thermal resistance is calculated from R2, R1, and R0 or is given
Using	Determined if the record is used during the optimization routine

Table 5.5 Heatsink parameters used in TimCAD

heatsink, it must be chosen such that the temperature of the device does not exceed the 7 maximum temperature allowed. Table 5.5 lists the heatsink properties used by TimCAD.

5.4 Conclusions

This chapter details the methods and models used to compute device losses and temperature rises in converter cells. Loss models are provided for cases where two MOSFETs are used as switches, when a MOSFET and a PiN diode is used, and when a MOSFET and Schottky diode is used. Both conduction losses and switching losses are computed using parameters found in the data sheets of the devices. Also described are methods and models for computing the device temperature rises, and for describing the thermal behavior of heatsink elements. TimCAD utilizes these models in the power stage design.

Chapter 6

Passive Elements and Filters

Inductors, capacitors, and filters play important roles in the design of a power converter. The inductor within the power stage of the converter is a significant energy storage element in the dc/dc converter while capacitors are the main buffers for absorbing the ripple current generated by the switching action of the power stage. Proper sizing and design of these elements require accurate models and careful attention to a variety of design considerations.

Stringent electromagnetic interference (EMI) specifications are applied at the terminals of the power converter. In order to meet these specifications, EMI filters are needed at the input and output of the converter to attenuate the high-frequency ripple generated by the converter. To design these filters, the filter models, and hence the models for their constituent inductors and capacitors, must be accurate for frequencies into the megahertz range.

This chapter describes the models for passive elements and filters used by TimCAD to design dc/dc converters. The passive elements described include the power stage inductor, the EMI filter inductor, and the larger capacitors in the filters. Three different filter topologies used in TimCAD are also described. This chapter contains the methods and formulae used by TimCAD for the design of these elements and filters.

6.1 The Converter Inductor

6.1.1 Overview

The converter inductor, shown in Fig. 6.1, is a major energy storage element in the converter. When the controlled switch, S1, is on, current flows from the 42 V bus though the inductor to the 14 V bus, transferring energy from the 42 to the 14 V bus. At the same time the inductor current increases at a rate of $\frac{V_{in} - V_{out}}{L}$ A/sec, increasing its stored energy. During the second operational state, when S1 is off and the second switch, S2, is on, the inductor is connected between ground and the 14 V bus. The current will then decrease at a rate of $\frac{V_{out}}{L}$ A/sec, transferring some of the energy in the inductor to the output.

As seen in Fig. 6.2, the current through the inductor will contain both dc and ripple components. The magnitude of the ripple current will be a function of inductance, frequency, and the voltage at both busses. The magnitude of the dc current depends on the required output current and the number of cells used in the converter. Significant ac and dc currents are present, thus both core losses and winding losses must be considered in the inductor design. Because a high frequency current will be present, we primarily

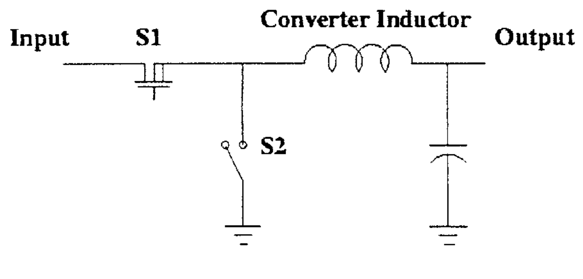


Figure 6.1 The layout of the power stage of the dc/dc converter/

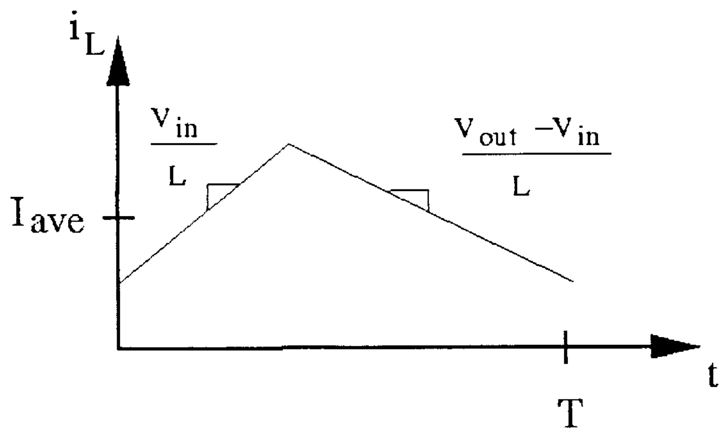


Figure 6.2: The current through the cell inductor for one period of the switching cycle.

consider ferrite cores in this application.

For every converter design TimCAD will design one of these inductors. To design the inductor the program stores the data shown in Table 6.1 about each core. TimCAD then uses the algorithm shown in Fig. E.3 in Appendix E, repeated here as Fig. 6.3, to design the inductor. Some important steps of this algorithm are described in the following subsections.

Field	Description
AL	The constant the correlates inductance to turns squared
Area	The area on the PC board that the inductor will require
Core Area	The cross sectional area of the core
Core Volume	The volume of the core, a magnetic property
Length Per Turn 1 Layer	The mean length of a turn assuming only one layer of windings are used
Length Per Turn Full	The mean length of a turn assuming the window area is full
Name	The name of the core
Price	The price of the inductor core
Rank	The order in which the program tries to use the inductors
Thermal Resistance	A constant that determines the temperature rise based on the power loss of a core.
Using	Determines if the program will ignore a record
Volume	The volume of the inductor core
Weight	The weight of the inductor core
Winding Area	The area in which all windings must fit
Winding Width	The width of the window area

Table 6.1 Fields in the Inductor core database.

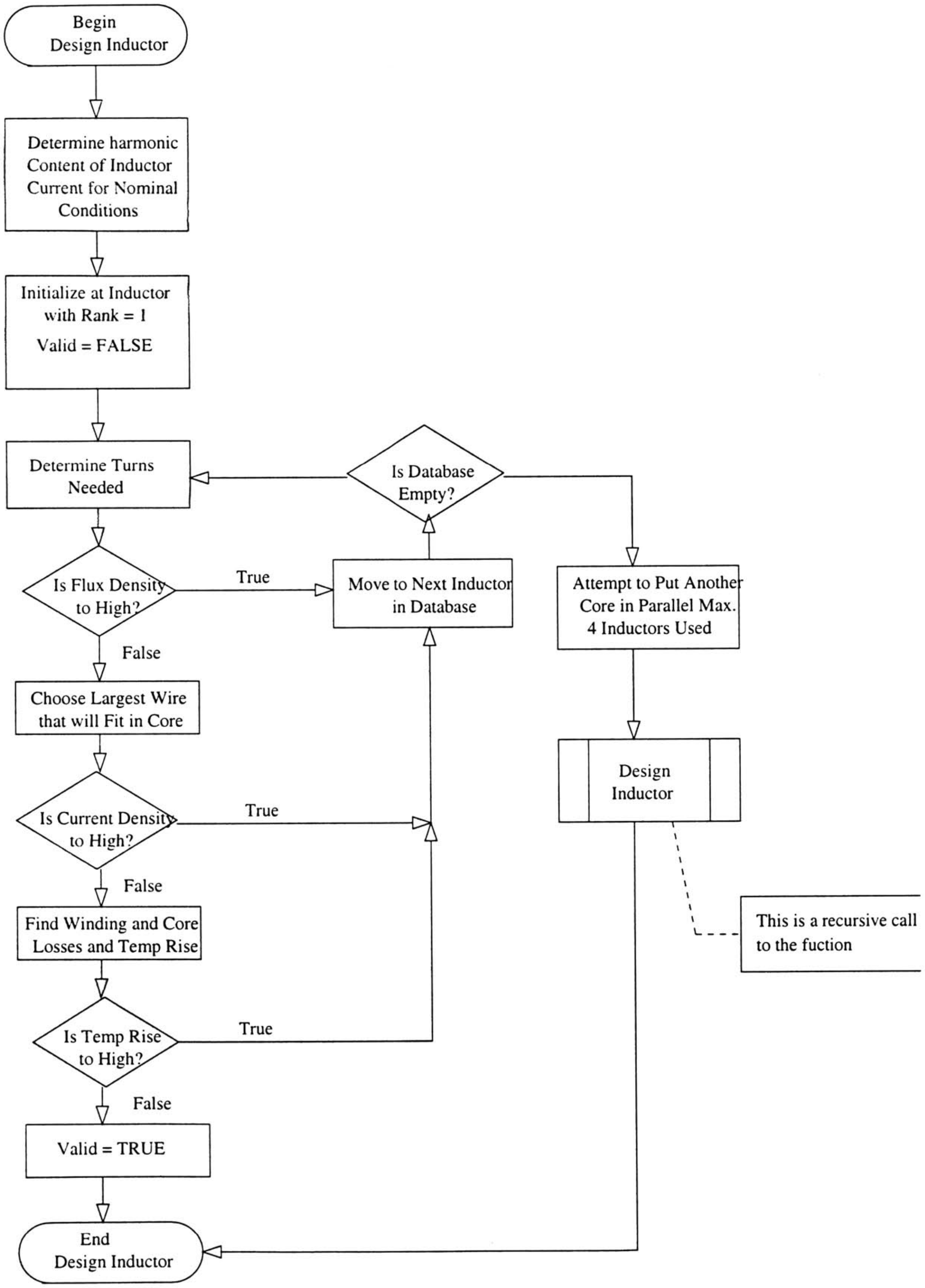

Begin Design Inductor
Determine harmonic Content of Inductor Current for Nominal Conditions
Initialize at Inductor with Rank = 1
Valid = FALSE
Determine Turns Needed
Is Database Empty?
Is Flux Density to High?
True
Move to Next Inductor in Database
Attempt to Put Another Core in Parallel Max. 4 Inductors Used
False
Choose Largest Wire that will Fit in Core
Design Inductor
Is Current Density to High?
True
False
This is a recursive call to the fuction
Find Winding and Core Losses and Temp Rise
Is Temp Rise to High?
True
False
Valid = TRUE
End Design Inductor

Figure 6.3 Flowchart for the converter inductor.

6.1.2 Initial Design Pass

To determine the number of turns, N, needed to achieve a given inductance, equation 6.1 is used:

$$N = \sqrt{\frac{L_{desired}}{A_L}} \qquad (6.1)$$

where A_L is a constant for the core specified by the manufacturer. With gapped cores the value of A_L can be controlled by the length of the gap, so that when custom-gapped cores are used a wide range of inductances is possible using any integer, N, number of turns.

For equation 6.1 to be valid the core can not be saturated, therefore the peak magnetic flux density (B_{pk}) in the core must be kept below the saturation level of the core material. If the core saturates then the inductance will be dependent on the operating conditions. This is a situation that is generally unacceptable since the inductance should be a constant value. The core will not saturate if the saturation point of the core, a material constant called B_{max}, is greater then B_{pk}, which is defined as

$$B_{pk}\ (T) = \frac{0.1 \cdot A_L \cdot N \cdot I_{pk}}{A_{core}} \qquad (6.2)$$

where A_L is the core constant in mH/1000 turns^2, N is the number of turns, I_{pk} is the peak current in A, and A_{core} is effective core cross section in cm^2. The allowed peak flux density, B_{max}, is 3000 Gauss for the Phillips ferrite cores used in TimCAD. If TimCAD finds that the number of turns needed will saturate the core at rated current, the core will be rejected as a candidate.

Once the number of turns is found and it is verified that the core will not saturate, the wire size will be determined. The turns of wire must fit within the window area of the core. (The window area is the cross-sectional area within the core through which all the turns must pass.) TimCAD finds the smallest gauge (largest diameter) wire which will fit within the window area. TimCAD verifies that this wire size will result in a maximum current density, J_{max}, of less then 3000 A/cm^2 in the wire. If not, the core is rejected as a candidate.

6.1.3 Temperature Rise Limit

Because of the significant dc and ripple components in the current waveform, there will be significant winding and core losses. The winding loss for the inductor is computed using the models developed in [15]. The winding loss is a summation of the power losses caused by every frequency component of the inductor current:

$$P_{wire} = \sum_{n} I_{n,rms}^2 \cdot R_{n,wire} \tag{6.3}$$

where $I_{n,rms}$ is the rms current at the nth harmonic frequency, and $R_{n,wire}$ is the effective resistance of the wire at that frequency. The effective resistance of the wire is frequency dependent; at higher frequencies the current will be restricted to the surface of the wire. This outer region, or skin depth, is given as

$$\delta_n = \sqrt{\frac{2 \cdot \rho_{Cu}}{2 \cdot \pi \cdot n \cdot f_{sw} \cdot \mu_{Cu}}} \tag{6.4}$$

where n, ρ_{Cu} and μ_{Cu} are the harmonic number, the resistivity and permeability of copper. The thickness of the wire in skin depths taking into account proximity effects is

$$X_n = \left(\frac{\pi}{4}\right)^{3/4} \cdot \frac{d^{3/2}}{L^{1/2}} \cdot \frac{1}{\delta_n} \tag{6.5}$$

where d is the wire diameter and L is the spacing between the wires. The effective resistance of the wire at the nth harmonic is then

$$R_n = \left(M_n + \frac{m^2 - 1}{3} \cdot D_n \right) \cdot R_{dc} \tag{6.6}$$

where m is the number of layers of windings and

$$M_n = \mathrm{Re}\left\{ \frac{(X_n + j \cdot X_n) \cdot \cosh(X_n + j \cdot X_n)}{\sinh(X_n + j \cdot X_n)} \right\} \tag{6.7}$$

$$D_n = \mathrm{Re}\left\{ \frac{2 \cdot (X_n + j \cdot X_n) \cdot \sinh(\tfrac{1}{2}(X_n + j \cdot X_n))}{\cosh(\tfrac{1}{2}(X_n + j \cdot X_n))} \right\} \tag{6.8}$$

The resistance of the wire at dc is simply the length of the wire used to create all the turns times the resistance per length of the wire. The length of the wire used is the number of turns needed times the mean length per turn (MLT) for a given core.

To calculate core loss, the ac flux swing in the core is approximated as a sinusoid and the core loss model provided by Ferroxcube/Phillips [15] is utilized:

$$P_{core}\ (W) \approx 9.16 \cdot 10^{-13} \cdot (f_{sw}\ (Hz) \cdot .001)^{1.231} \cdot (0.5 \cdot B_{pk}\ (gauss))^{2.793} \cdot V_{core}\ (cm^3) \tag{6.9}$$

where the constants are material dependent and V_{core} is the volume of the core.

Based on the sum of winding and core losses, the centerpost temperature rise of the core can be estimated by multiplying the power dissipation times the thermal resistance of the core

$$(P_{wire} + P_{core}) \cdot R_L = T_{rise} \tag{6.10}$$

If the temperature rises above the maximum allowable temperature, the inductor design is rejected, otherwise the program accepts the design of the inductor.

6.2 EMI Filter Inductor Design

6.2.1 Overview

The inductors used in the EMI filters have different design requirements then those used for the main converter inductor. The inductors in the EMI filters carry large dc currents (the aggregate current of the cells) but only very small levels of ripple current. As a result, core losses and ac winding losses can be neglected, but the dc winding losses and core energy storage capability are important. Because of these design considerations toroidal powdered iron cores are used for these elements, and a different design algorithm is utilized.

To design the EMI filter inductor one must use a core that can store the appropriate amount of energy while maintaining a reasonable temperature rise. One must also consider that the core may be partially saturated at full current when selecting the numerical inductance value and number of turns. To model these inductor cores TimCAD uses the data fields listed in Table 6.2 and the algorithm represented in Fig. E.9 repeated here as Fig. 6.4.

6.2.2 Initial Design Pass

The first step in choosing a suitable core is to identify which cores can store enough energy with an acceptable temperature rise and saturation level. Thus, the EMI inductor cores are initially ranked on energy storage. The toroid cores used in TimCAD have an upper limit for energy that depends on the complexity of the windings and the allowed temperature rise. The energy stored by the inductor,

$$W_m = \frac{1}{2} \cdot L \cdot I^2 \tag{6.11}$$

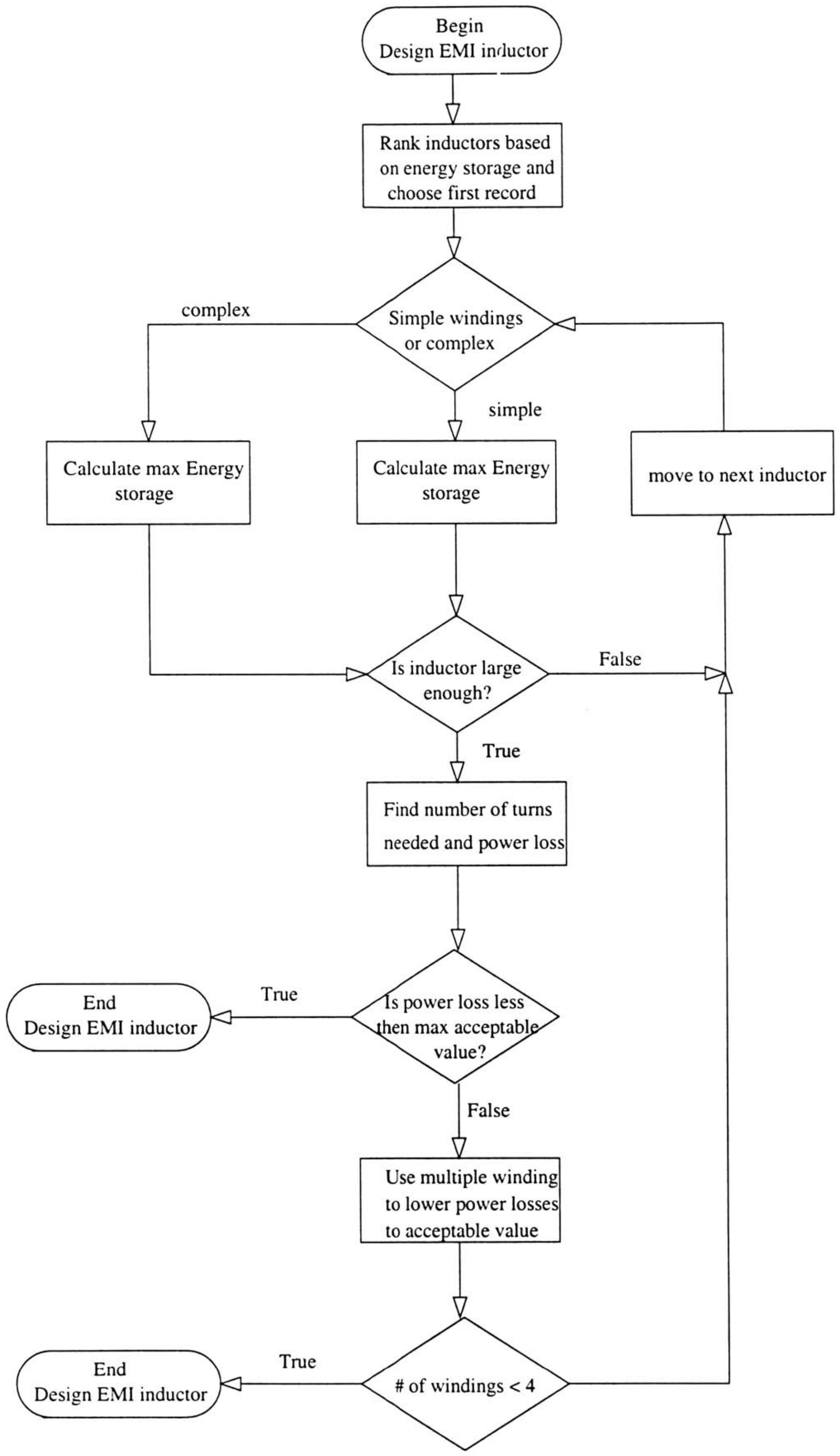

Begin
Design EMI inductor
Rank inductors based on energy storage and choose first record
Simple windings or complex
complex
simple
Calculate max Energy storage
Calculate max Energy storage
move to next inductor
Is inductor large enough?
False
True
Find number of turns needed and power loss
Is power loss less then max acceptable value?
True
End
Design EMI inductor
False
Use multiple winding to lower power losses to acceptable value
# of windings < 4
True
End
Design EMI inductor

Figure 6.4 Flowchart for the design of the EMI filter inductor.

must be less then this maximum limit for the core to be chosen. If the inductor is incapable of storing the energy needed within an acceptable temperature rise then the design is rejected and the next core with more energy storage capability is chosen. Once an acceptable core is found the number of turns can be calculated. Toroidal, ungapped cores are used for the EMI filters, thus the amount of saturation of the core will influence the effective inductance. The number of turns required can be calculated as in [16]:

$$N = \sqrt{\frac{L_{desired}}{A_L \cdot \%\mu}} \tag{6.12}$$

Field	Description
Al	The constant the correlates inductance to turns squared
Area	The area on the PC board that the inductor will require
Core Area	The cross sectional area of the core
Core Volume	The magnetic volume of the core
Full 10	Energy storage allowed for a 10 °C temperature rise for an inductor with a full winding
Full 25	Energy storage allowed for a 25 °C temperature rise for an inductor with a full winding
Full 40	Energy storage allowed for a 40 °C temperature rise for an inductor with a full winding
MLT	The mean length per turn of the wire
Name	The name of the core
Price	The price of the inductor
Simple 10	Energy storage allowed for a 10 °C temperature rise for an inductor with a simple winding
Simple 25	Energy storage allowed for a 25 °C temperature rise for an inductor with a simple winding
Simple 40	Energy storage allowed for a 40 °C temperature rise for an inductor with a simple winding
Surface Area	The surface area of the core
U1	The first coefficient in the parabolic representation of the relationship between energy and percent saturation
U2	The second coefficient in the parabolic representation of the relationship between energy and percent saturation
U3	The third coefficient in the parabolic representation of the relationship between energy and percent saturation
Using	Determines if the program will ignore a record
Volume	The volume of the inductor
Weight	The weight of the inductor
Window Area	The area in which all windings must fit

Table 6.2 Fields in the *EMI Inductor* database.

where $\%\mu$ is a measure of the permeability of the core, a value dependant on the energy storage required from the inductor. The number of turns needed is then used to determine the temperature rise of the core.

6.2.3 Temperature Rise Limit

Since the ac currents in the EMI filter inductor will be small, the power losses are mainly due to the copper losses. The power losses of the core can then be written simply as

$$P_{loss} = I^2 \cdot R_{wire} \qquad (6.13)$$

A user definable value is the maximum power loss of the EMI inductor. The routine will attempt to design the inductor with less than this power loss. If an inductor needs more then four parallel windings to achieve this then the inductor core is rejected.

The window area of a core must be considered when choosing the number of windings necessary for an inductor. The total wire area needed in the core is the number of turns the area required per turn. If the core does not have enough winding area to hold all the turns needed then the core is rejected.

The temperature rise of the EMI filter inductor is a function of the power lost. This function for temperature rise of the Micrometals toroid cores in free-standing air is given by [16] as

$$\Delta T_{core} = \left(\frac{P_{loss}}{A_{surface}} \right)^{.833} \qquad (6.14)$$

where $A_{surface}$ is the surface area of the core. If the temperature rise of the core is within limits, the core can store all the energy needed, and the window area is not exceeded then the inductor has been design successfully.

6.3 Capacitors

Since the filters are built to suppress harmonics into the megahertz range, the model for capacitors must include high-frequency characteristics. The high-frequency model used for the capacitor is shown in Fig. 6.5. It contains an equivalent series resistance (ESR) and an equivalent series inductance (ESL). These parasitic elements affect the impedance of the capacitor starting near the resonant frequency of the capacitor, $\omega = \frac{1}{\sqrt{L_{ESR} \cdot C}}$. At resonance, the ESR dominates the capacitor impedance. Above the resonant frequency, the ESL will dominate the impedance of the capacitor. A typical capacitor selected for this application would be a film capacitor with a resonant frequency on the order of 100 kHz – 1MHz and an ESR on the order of 10 mΩ. Table 6.3 lists the data fields TimCAD uses to save information about the capacitors it uses.

Field	Description
Area	The area on the PC board that the inductor will require
Capacitance	The rating of the capacitor
Bus	Checked if capacitor is used next to the voltage bus
Damping	Checked if capacitor is used in damping branches
ESL	Parasitic inductance in series with the capacitor
ESR	Parasitic resistance in series with the capacitor
Max Ripple Current	The maximum ripple current that the capacitor can tolerate
Max Temp	The maximum temperature that the capacitor can tolerate
Name	The name of the capacitor
Nominal Voltage	The maximum voltage a capacitor can tolerate at nominal temperatures
Price	The price of the capacitor
Switching	Checked if the capacitor is used for low ESL applications
Using	Determines if the program will ignore a record
Voltage/Temp Breakpoint	The temperature at which the voltage needs to be derated
Volts at Max Temp	The maximum voltage a capacitor can tolerate at the max temperature
Volume	The volume of the capacitor
Weight	The weight of the capacitor

Table 6.3 Fields in the *Capacitor* database

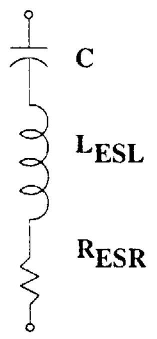


Figure 6.5 The high-frequency equivalent circuit for a capacitor, including parasitic resistance and inductance.

6.4 Filters

6.4.1 Introduction

Two filters are needed for the dc/dc converter. The input and the output of the converter are connected to busses that will have EMI specifications limiting the amount of current ripple allowed on the bus. The unfiltered input current of the converter is the summation of the currents in switch S1 of each cell. As seen in Fig 6.6 the input current in each cell drops to zero at the end of the first stage of operation, so this current, along

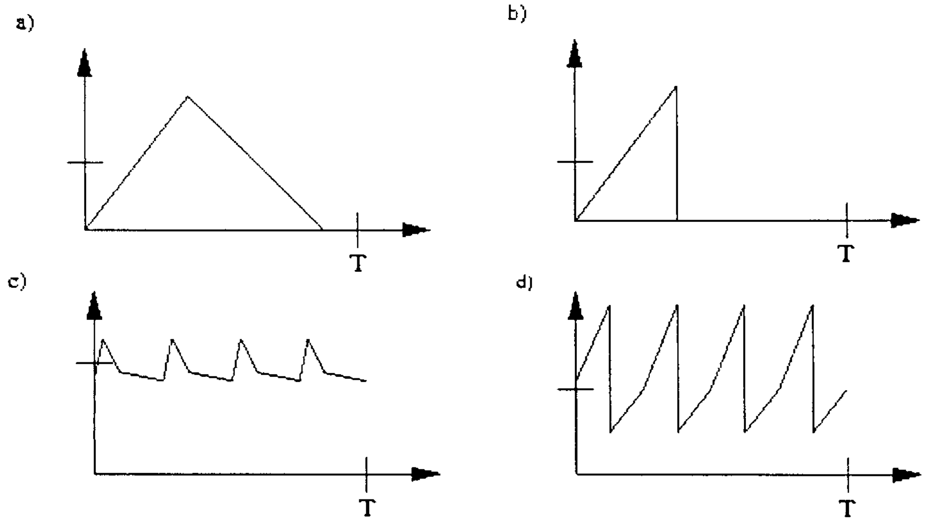


Figure 6.6 The output (a) and the input (b) current of a cell and the total output (c) and input (d) current of a four cell power stage.

with the input current to the converter will contain discontinuities and therefore the high-frequency harmonics in the current will be large. The output current of the converter is the summation of the cell inductor currents, and hence is continuous.

The filters need to meet standards that limit the magnitude of the current ripple out to the megahertz range. To meet the ripple specifications TimCAD uses three different filters.

6.4.2 Filter Specifications and Measurements

The Society of Automotive Engineers' (SAE) EMI specifications for the 14 V bus are stated in SAE J1113/41. These specifications limit the broadband and narrowband signals that can appear in the test setup described in the specifications. Figure 6.7 shows the Class 1 narrowband specifications.

The Line Impedance Stabilization Network (LISN) is a device used in EMI test procedures. This two port device, shown in Fig. 6.8, acts to separate the dc component of

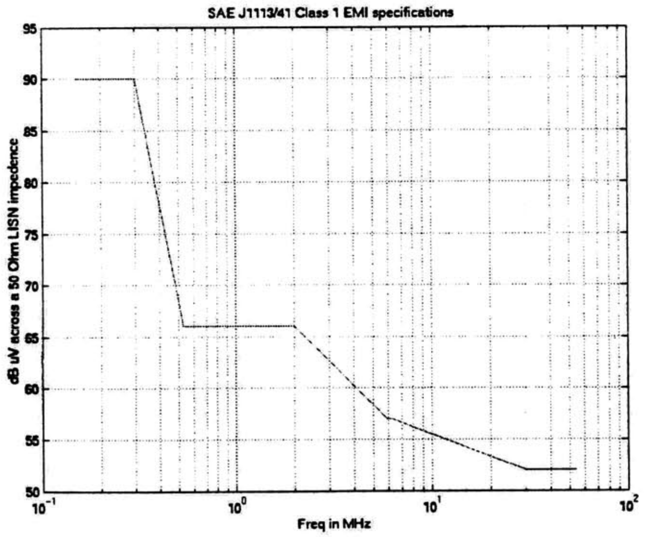


Figure 6.7 The plot of SAE J1113/41 Class 1 narrowband EMI specification (repeated from Fig. 2.2)

the ripple current from its high-frequency components, and to provide a known ripple-frequency impedance to the system under test. The capacitance and inductance in the LISN effectively separate the high-frequency ac ripple, which the specifications address, from the dc signal. The inductor appears as a high impedance for all ac components and the capacitor appears as an open circuit for the dc component. Therefore, only the ripple components of the current will pass through the LISN resistance, and it is the allowed voltage ripple, V_o, across this resistance which is defined in the specifications.

Due to interleaving, the fundamental ripple in the current will occur at

$$f_1 = f_{sw} \cdot N_{cells} \,. \tag{6.15}$$

Therefore the largest and most important components to attenuate will have frequencies in the hundreds of kilohertz and megahertz range. Because we are dealing with these high frequencies, suitable filter capacitors are those that can handle high ripple currents and

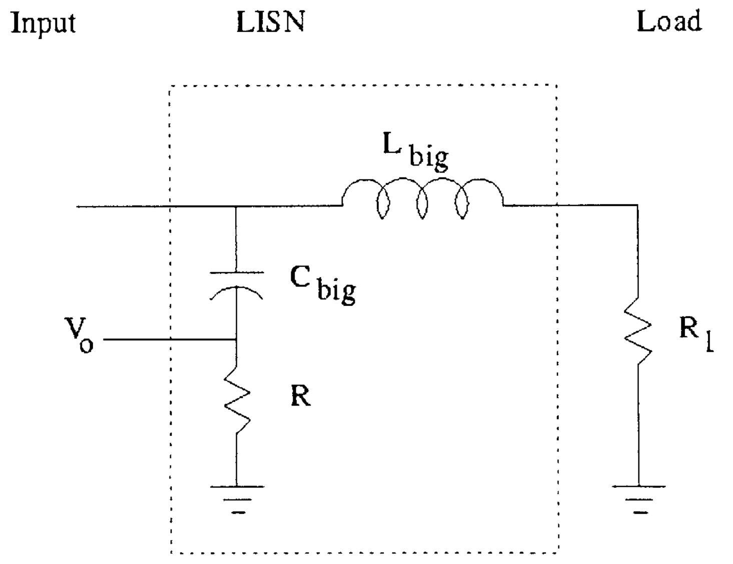


Figure 6.8 The setup for testing the EMI of the output of the converter consists of the LISN and the load. The dc component will travel through the LISN to the load whereas the current ripple will be measured across the LISN resistance.

therefore the largest and most important components to attenuate will have frequencies in the hundreds of kilohertz and megahertz range. Because we are dealing with these high frequencies, suitable filter capacitors are those that can handle high ripple currents and have low ESR and ESL. At these high frequencies the effect of the capacitor's ESL must be included in the design of the filter. Figure 6.9 shows the magnitude response of a filter with and without the effects of the ESL. The lower curve shows the gain of the filter without the parasitic elements in the capacitors. The gain of the filter with ESL considered deviates from this curve at about one megahertz. In typical designs the fundamental frequency of the ripple will in the hundreds of kilohertz range and either curve may be used. For higher harmonics though, the ripple frequency will occur in the megahertz range and the detailed curve must be used.

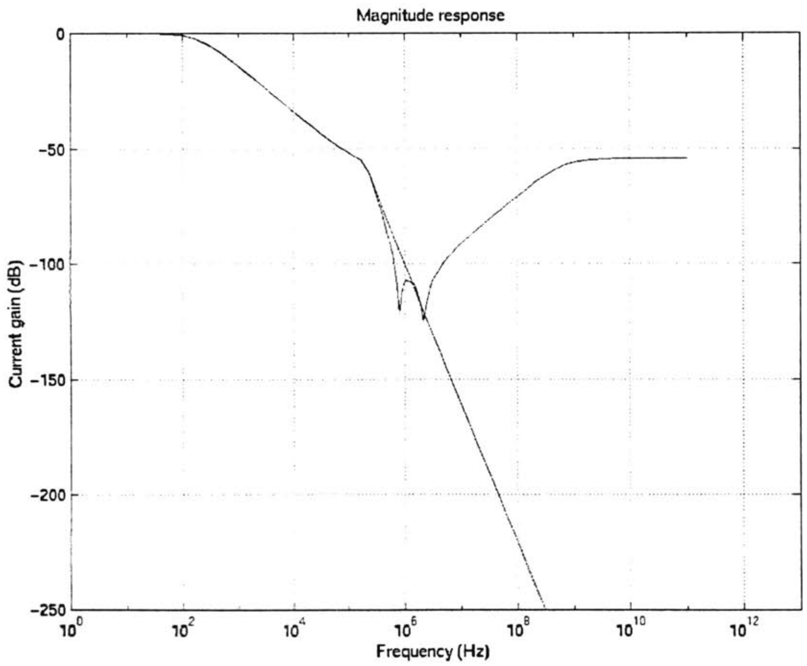


Figure 6.9 The gain of the filter in figure 6.11. The lower curve uses the ideal capacitor model. The upper curve includes the effect of ESL and ESR.

6.4.3 Filter Design Overview

The filter design must allow the EMI specifications to be met under all normal operating conditions. During normal operation, however the input and output voltages can vary. The voltages of the converter ports will affect the shape of the current waveforms seen in Fig. 6.6, and in turn their spectral content.

To ensure that the specifications will be met for all variations of input and output voltage, the worst-case ripple component across all operating points is computed at each frequency. Each filter design algorithm then uses this set of worst case harmonics as the signal to be attenuated, resulting in a filter that will always meet the EMI specifications regardless of the voltage on either of the buses.

To analyze the performance of the filters the system is tested as described in the EMI specifications. Assume that two LISNs surround the converter. The output of the first LISN is connected to a 42 V power supple and the output of the second filter is connected to an appropriate load as shown in Fig. 6.10. Do to the nature of the LISN, dc current will pass through the device, but all the ac currents produced by the converter pass through the 50 Ω LISN resistance. Since the filters are designed to attenuate ac ripple for the EMI test using the LISNs, the output of all the filters is assumed to be the LISN resistance.

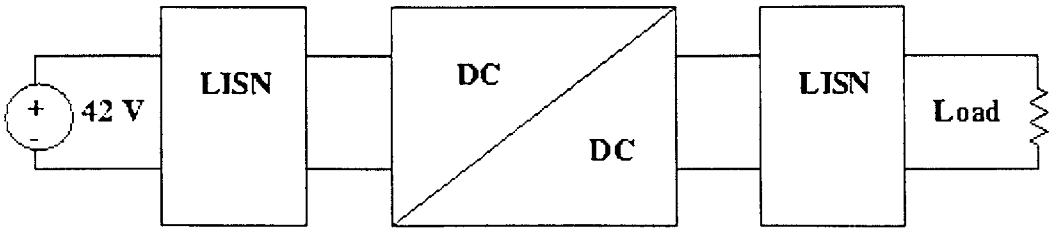


Figure 6.10 The setup for EMI testing. Note that the inputs of the LISNs are toward the dc/dc converter, and for ac frequencies in the ranges given by the specifications the input impedance of the LISN will be 50 Ω.

After calculating the worst-case harmonics of the converter's power stage, TimCAD designs the input and output filters. According to which filter types were selected in the *Filters* Dialog Box, TimCAD will randomly choose a filter type from these and design it according to the filter's design algorithm. The inner control loop of the program will cause the filter to be designed multiple times, and the optimal filter (as determined by the cost function) will be chosen. The flow chart for this operation is shown in Fig. E.5 in Appendix E.

6.4.4 Filter Type 1

To design a filter of the first type (shown in Fig. 6.11) three parameters are needed: the location of the dominant poles, p, the Q of the filter, and one component value. In the design algorithm used for TimCAD, C_2 is chosen randomly based on the components in the database. The variable p is initially assigned a value and the filter is designed. For every frequency at which a harmonic exists the program examines the attenuation provided by the filter, the worst case harmonic of the signal to be filtered, and

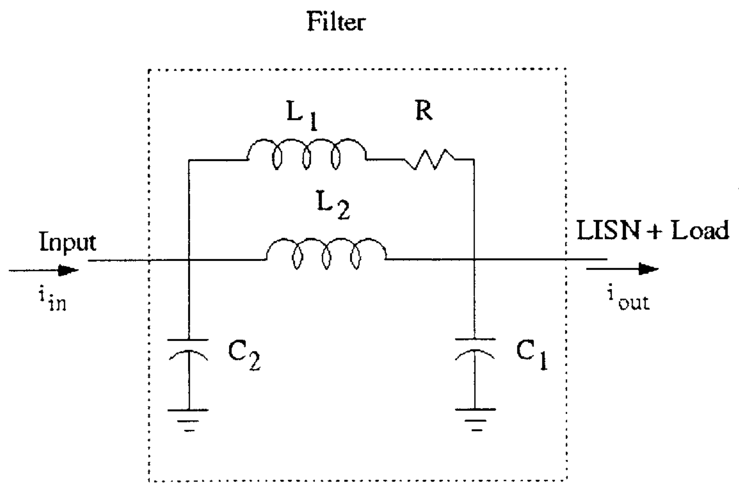


Figure 6.11 Layout of filter type 1.

the EMI limit. The value of p is then adjusted if more or less attenuation is needed from the filter. The values of the other components can be determined using the design algorithm described in [8]. Namely:

$$C_1 = \frac{C_2}{4} \tag{6.16}$$

$$R = \frac{5 \cdot p^2}{4 \cdot C_2} \tag{6.17}$$

$$L_2 = \frac{5 \cdot p^2}{C_2} \tag{6.18}$$

$$L_1 = \frac{4 \cdot p^2}{5 \cdot Q^4 \cdot C_2} \tag{6.19}$$

The inductor, L_2, will carry the dc component of the current, and therefore its energy storage will be significant. The more energy L_2 stores the larger the inductor must be. The inductor L_1 is in a damping branch, and therefore it will carry a relatively low magnitude current ripple and no dc current. L_1 can be implemented with a small inductor or possibly as the parasitic of a wirewound resistor.

At frequencies much greater then the resonant frequency of the capacitors, C_1 and C_2 are effectively inductors L_{C1} and L_{C2} (i.e., the ESL of the capacitors from Table 6.3). Then the transfer function for the filter at high frequencies, found by considering the inductors in the circuit only, is

$$\left.\frac{I_{out}}{I_{in}}\right|_{high\_frequency} = \frac{L_{C2} \cdot (L_1 + L_2)}{L_1 \cdot L_2} \tag{6.20}$$

note that L_{C1} is in parallel to the load. For the purposes of the EMI performance it can be assumed that for all ripple components the load is the 50 Ω LISN resistance. Therefore at high frequencies L_{C1} is in parallel with the load inductance, which is assumed to be zero, thus L_{C1} can be ignored.

6.4.5 Filter Type 2

The second filter type is a two-stage LC filter. This filter has two inductors in the current path; thus both inductors cores must be rated for significant energy storage. This filter naturally has a high Q, i.e., at some frequency near the cutoff of the filter there is little to no attenuation at the noisy port. The Q of the filter must be reduced in order to ensure the filter's proper operation. With a high Q system a frequency in the input could easily contaminate the voltage busses, or the system could resonate at that frequency. In order to reduce the Q two damping legs are added to the circuit resulting in the circuit shown in Fig. 6.12.

The algorithm used to design this filter begins with the selection of a capacitor, C_2. The other capacitors in the circuit are then

$$C_1 = \frac{C_2}{4} \tag{6.21}$$

The capacitors on the damping legs need to be greater then the capacitor in series, thus

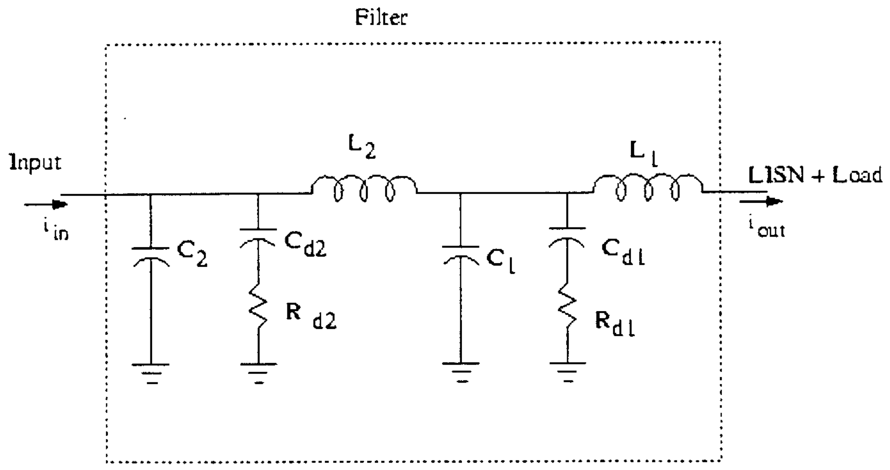


Figure 6.12 Layout of filter type 2

$$C_{d2} = 10 \cdot C_2 \tag{6.22}$$

$$C_{d1} = 10 \cdot C_1 \tag{6.23}$$

The resistors, R_{d1} and R_{d2}, are chosen to limit the Q of the filter to a level what will not interfere with the operation of the filter.

Changes in the value of L_1 only slightly vary the attenuation of the filter at low frequencies, but affect the gain as much as changes in L_2 at frequencies greater then the resonant frequencies of the capacitors. The inductor L_2 affects the gain for all frequencies under consideration. Table 6.1 illustrates the effects of changes in the inductances on the attenuation provided by the filter at several frequency. Thus, to design an adequate filter after the capacitors are chosen the inductance of L_2 will be varied to change the gain of the filter for harmonics less then a few MHz, whereas both L_1 and L_2 will be varied to change the harmonics at higher frequencies.

For high frequencies the capacitors can be replaced with their parasitic inductors and the resistors can be ignored. Since the damping capacitors have a ESL much higher then the capacitors chosen for C_1 and C_2, the inductance in parallel can be approximated to L_{C1} and L_{C2} (the ESL of the capacitors). The high-frequency transfer function is then

	100 kHz	300 kHz	500kHz	750 kHz	1 MHz	5MHz
L_1, L_2	**-106.4**	**-140.0**	**-167.7**	**-170.0**	**-160.0**	**-150.8**
$L_1, 2 \cdot L_2$	**-112.5**	**-146.0**	**-173.8**	**-176.1**	**-166.0**	**-156.9**
$L_1, 3 \cdot L_2$	**-116.0**	**-149.5**	**-177.3**	**-179.6**	**-169.5**	**-160.4**
$L_1, 4 \cdot L_2$	**-118.5**	**-152.0**	**-179.8**	**-182.1**	**-172.0**	**-162.9**
$L_1, 5 \cdot L_2$	**-120.5**	**-154.0**	**-181.7**	**-184.0**	**-173.9**	**-164.8**
$2 \cdot L_1, L_2$	**-106.6**	**-141.4**	**-170.4**	**-173.9**	**-164.5**	**-156.8**
$3 \cdot L_1, L_2$	**-106.9**	**-143.0**	**-172.9**	**-176.8**	**-167.7**	**-160.3**
$4 \cdot L_1, L_2$	**-107.3**	**-144.6**	**-174.9**	**-179.1**	**-170.0**	**-162.8**
$5 \cdot L_1, L_2$	**-107.8**	**-146.0**	**-176.7**	**-180.9**	**-171.9**	**-164.7**

Table 6.1 For filter type 2 the effects of changes in L_2 and L_1 on the current gain in dB are compared. Changes in L_2 are more influential at lower frequencies.

$$\left.\frac{I_{out}}{I_{in}}\right|_{high\_frequency} = \frac{L_{C1} \cdot L_{C2}}{L_1 \cdot L_2} \tag{6.24}$$

6.4.6 Filter Type 3

The third type of filter is the simple LC filter shown in Fig. 6.13. This filter is included so that converters with very low ripple ratios can employ very simple filters, without resorting to awkward higher order filter designs. The design algorithm is to choose the capacitor, C_2, randomly, then choose the inductance, L_2, such that the EMI specifications are met. If the inductance of this filter is not needed the algorithm will design the filter with just the capacitor.

At high frequencies the transfer function of this filter is

$$\left.\frac{I_{out}}{I_{in}}\right|_{high\_frequency} = \frac{L_{C2}}{L_2} \tag{6.25}$$

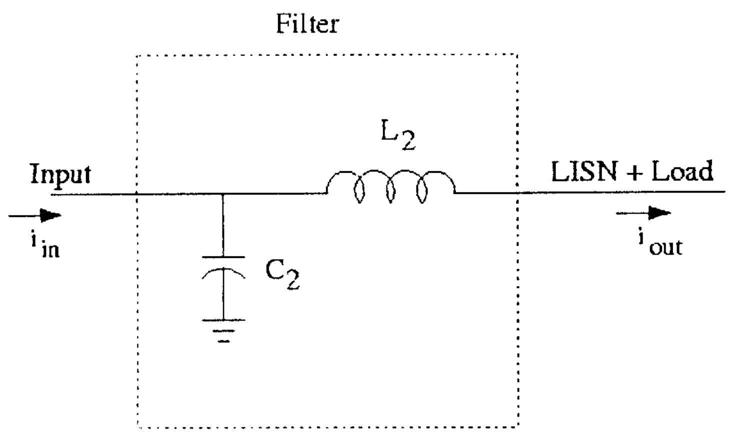


Figure 6.13 Layout of filter type 3

6.4.7 Summary of the Filter Design

Even though TimCAD randomly choose a filter topology on every iteration of the inner loop, each of the three filters is typically found to be optimal for different occasions. In cases when only a little attenuation is needed, such as when the ripple ratio is very low or when the EMI limitations are eased filter type 3 can be used. This filter has fewer components and will usually have less price, weight, and volume then the other topologies. When attenuation is needed for high frequencies, filter type 2 can be used, since the other filters cannot provide the necessary attenuation. Figure 6.14 shows a comparison of the gain of all three filters using the same C_1, C_2, and L_2 components; the L_1 inductor is assumed to be identical to L_2 in the second filter type.

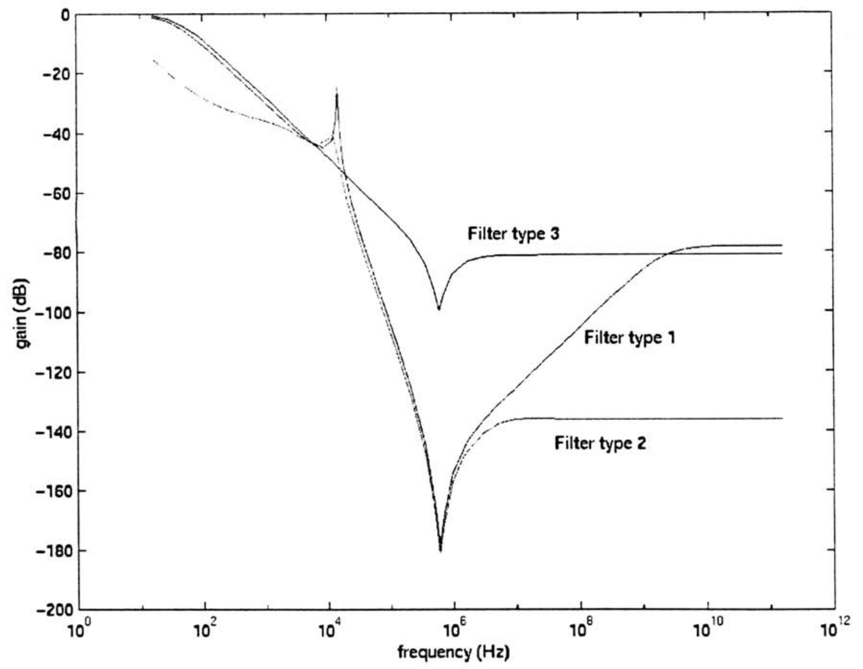


Figure 6.14 Comparison of the magnitude response of the three filter topologies. All inductors are 10 μH, C2 is 80 μF, and C1 is 20 μF.

6.5 Conclusions

The switching action of a power converter causes discontinuities in the voltage and current waveforms that must not interfere with the rest of the system. To remove the undesired components of these signals, filters are needed at the input and output of a power converter. The inductors and capacitors needed must be rated to carry the total current in the converter and withstand the converter's voltage. The biggest, heaviest, and most expensive elements in a power converter are often the inductors and capacitors because of the EMI requirements. Therefore, the design of these elements must be done carefully in order to ensure the optimal design of the converter.

Chapter 7

Conclusions

7.1 Thesis Conclusions

Three major objectives have been addressed in this thesis. The first objective is the development of a prototype dc/dc converter for dual-voltage automotive applications. The second objective is the development of a CAD optimization tool. The final objective is to use the optimization tool to study the effects of system-level specifications on the characteristics of optimized converters.

The development of the dc/dc converter has been useful in a number of ways. First, the process of designing and testing it has helped us identify a number of important tradeoffs and issues in the design process. Second, the converter has been useful for validating and refining the optimization program models described in Chapter 5 and Chapter 6. Finally, the converter has proved useful in a variety of dual voltage automotive research projects.

TimCAD, the dc/dc converter optimization program, has been written and tested. This program is capable of searching the design space of the dc/dc converter and identifying designs that minimize a weighted sum of price, weight, and volume. General design trends can also be analyzed. For example, the sensitivity of converter price, weight, and volume to variations in system parameters such as ambient temperature or EMI limits can be determined by performing many optimization runs with different design requirements.

Initial results found using TimCAD show the effects of various system-level parameters on the total volume of the converter. In every case, the design space for the optimization covered a wide range of frequencies, ripple ratios, number of cells, and component selections. The effects of variations in power level, ambient temperature, and EMI specifications on the volume of the optimized converter have also been examined.

The final conclusion made by this thesis is that, although the addition of the dc/dc converter represents a significant increase in cost to the automobile, this cost can be minimized using CAD optimization techniques. Through an extensive search of the design space, designs for the converter with the lowest possible price, weight, and volume can be found. Furthermore, changes in the design of the electrical system can be better understood if the sensitivities of the converter's cost to system level parameters are known. For example, when listing the savings and costs of moving a load from the 14 V bus to the 42 V bus, the savings from reducing the converter power rating can be determined from the sensitivity of the converter's cost to power level. Thus, the optimized designs and sensitivity information that can be generated using CAD optimization have great potential value.

7.2 Recommendations for Future Work

There are several directions future work in this area should take. To generate more highly optimized designs and to provide more detailed information about the sensitivity of the converter's cost, the database of components must be enlarged. Introducing more parts to TimCAD will further refine the optimization process by providing more possible designs to choose from. The pricing information from a major company for all the components in the parts database would be useful in performing price-optimized designs that truly reflect the cost of a converter to the automotive industry. Although the absolute cost will be impossible to obtain and maintain, results of price-optimized designs can be found that will show the approximate trends in the designs.

Improvements in the optimization program, TimCAD, could also be made. The models could be refined and new attributes could be incorporated. For example, more filter topologies could be added. The addition of other dc/dc converter topologies could be added with only limited changes to the program. Adding a new converter topology would require a new design algorithm and new device models for any component not already represented. Other changes that would be needed include additions to the control loop, user interface, and possibly the database, and the preexisting device models.

Finally, methods to expedite the optimization process are needed. The most time-consuming process in the program involves the interface between TimCAD and *Microsoft Access*. To accelerate the optimization process, future versions of TimCAD would benefit from custom-built data files.

References

[1] J. G. Kassakian, H-C. Wolf, J. M. Miller, C. J. Hurton, "The Future of Automotive Electrical Systems," Proc. *IEEE Workshop on Power Electronics in Transportation*, Hyatt-Regency Hotel, Dearborn, MI, Oct. 24-25, 1996, pp. 3-12.

[2] J. M. Miller, D. Goel, D. Kaminski, H.-P. Schöner, T. M. Jahns, "Making the Case for a Next Generation Automotive Electrical System," *International Congress on Transportation Electronics Convergence '98*, pp. 41-51

[3] J. G. Kassakian, H-C. Wolf, J. M. Miller, C. J. Hurton, "Automotive Electrical Systems circa 2005," *IEEE Spectrum*, August 1996, pp. 22-27.

[4] B. A. Miwa, D. M. Otten, and M. F. Schlecht, "High Efficiency Power Factor Correction Using Interleaving Techniques," *IEEE Applied Power Electronics Conference*, Boston, MA, 1992.

[5] B. A. Miwa, "Interleaving Converter Techniques for High Density Power Supplies," Doctoral Thesis, Dept. of EECS, Massachusetts Institute of Technology, June 1992.

[6] C. Chang and M. Knights, "Interleaving Technique in Distributed Power Conversion Systems," *IEEE Trans. Circuits ans System -I*, Vol. 42, No. 5, May 1995, pp. 245-251.

[7] C. Chang, "Current Ripple Bounds in Interleaved DC-DC Power Converter," *Proceedings of the 1995 International Conference on Power Electronics and Drive Systems, Singapore, 1995.*

[8] T. K. Phelps and W. S. Tate, "Optimizing Passive Input Filter Design," *Proceedings of the 6th National Solid-State Power Conversion Conference (Powercon 6)*, May 1979, pp. G1-1- G1-10.

[9] J. J. Jeyappragash, T. V. Sivakumar, and V. V. Sastry, "Object Oriented Modeling, Simulation and Optimization of Power Electronic Circuits," *IEEE Power Electronics Specialists Conference, 1996*, pp. 581-585.

[10] C. Gezgin, B. S. Heck, and R. M. Bass, "Simultaneous Design of Power Stage and Controller for Switching Power Supplies," IEEE Transactions on Power Electronics, vol.12, no.3, May 1997, pp. 558-566.

[11] A. Reatti, "Steady-state analysis including parasitic components and switching losses of buck and boost DC-DC PWM converters under any operating condition," *International Journal of Electronics*, 1994, Vol. 77, No. 5, pp. 679-701.

[12] F. Blaabjerg and J. K. Pedersen, "Optimized design of a complete three-phase PWM-VS inverter," *IEEE Transactions on Power Electronics*, vol.12, no.3, May 1997, pp. 567-577.

[13] D. Grant and J. Gower, Power MOSFETs: Theory and application, John Wiley and Sons, NY, NY, 1989.

[14] D. J. Perreault, "Design and Evaluation of Cellular Power Converter Architectures Doctoral Thesis, Dept. of EECS, *Massachusetts Institute of Technology*, June 1997.

[15] Ferrite Material and Components Catalog, 8th Ed., Philips Components, Discrete Product Division, Riviera Beach, FL.

[16] Issue I, Power Conversion & Line Filter Applications, Micrometals Inc., Anaheim, CA, February, 1998.

Appendix A Voltage Limits

Symbol	Limit	Meaning
$V_{42, OV\text{-}dyn}$	55 V	Maximum dynamic overvoltage on 42 V bus during fault conditions
$V_{42, OV\text{-}stat}$	52 V	Maximum static overvoltage on 42 V bus
$V_{42, E\text{-}max}$	43 V	Maximum operating voltage of 42 V bus while engine is running
$V_{42, E\text{-}nom}$	41.4V	Nominal operating voltage of 42 V bus while engine is running
$V_{42, E\text{-}min}$	33 V	Minimum operating voltage of 42 V bus while engine is running
$V_{42, OP\text{-}min}$	33 V	Minimum operating voltage of 42 V bus. Also, lower limit operating voltage for all non-critical loads (i.e. loads not required for starting and safety)
$V_{42, FS}$	25 V	Failsafe minimum voltage: lower limit on operating voltage for all loads critical to starting and safety on the 42 V bus
$V_{14, OV\text{-}dyn}$	20 V	Maximum dynamic overvoltage on 14 V bus during fault conditions
$V_{14, OV\text{-}stat}$	16 V	Maximum static overvoltage on 14 V bus
$V_{14, E\text{-}max}$	14.3 V	Maximum operating voltage of 14 V bus while engine is running
$V_{14, E\text{-}nom}$	13.8 V	Nominal operating voltage of 14 V bus while engine is running
$V_{14, E\text{-}min}$	12 V	Minimum operating voltage of 14 V bus while engine is running
$V_{14, OP\text{-}min}$	11 V	Minimum operating voltage of 14 V bus. Also, lower limit operating voltage for all non-critical loads
$V_{14, FS}$	9 V	Failsafe minimum voltage: lower limit on operating voltage for all loads critical to starting and safety on the 14 V bus

"Draft Specification of a Dual Voltage Vehicle Electrical Power System 42V/14V," *Forum Bordnetz working document*, March 4, 1997.

Appendix B Load List

A detailed and thorough examination was made of the electrical loads that are expected to be present on a high-end luxury automobile in the near future. These loads are separated into six categories: motor, solenoid, lighting, heating, electronic, and other. The lists of loads, power ratings, and usage percentages were taken from the MAESTrO database and were compiled by Khurram Afridi.

The summer and winter worst case percentages are the percentage of time a device is on in the winter or summer in a space of several minutes. The year average is the percentage of time the device is on throughout the year. For example, when the windshield wipers are on they are powered continually, thus the worst case percentage is 100 %, but since the wipers are not used all year the year average is less then 100 %.

The loads were separated into 14 and 42 V lists. The dc/dc converter being designed is a model that could be used during the transition to the dual voltage system, hence it is assumed that only a few high power loads would be on the high voltage bus. The average power and the winter and summer worst case power of both busses can then be found. The average power is the nominal power of the load times the year average. The worst case power assumes that all the loads are on, thus it's the maximum power times the worst case percentages.

The summation of the power needed for every category of loads is at the bottom of each table. The system totals are

Voltage Bus	Average Power	Summer Peak Power	Winter peak Power
14 V Bus	612.6	1038.7	1228.7
42 V Bus	1420.4	3526.2	3416.2

Load	ID	Max Power (W)	Nominal Power (W)	Summer Worst Case (%)	Winter Worst Case (%)	Year Average (%)	14 Volt	42 Volt	Average Power 14 Volts	Average Power 42 Volts	Summer Peak Power 14 Volts	Summer Peak Power 42 Volts	Winter Peak Power 14 Volts	Winter Peak Power 42 Volts
Starter	m1	2500.0	2500.0	0.1	0.1	0.1	0	1	0	2.5	0	2.5	0	2.5
Fuel_Pump	m2	100.0	100.0	100.0	100.0	100.0	0	1	0	100	0	100	0	100
Idle_Speed_Bypass_Valve_Actuator	m3	6.0	6.0	50.0	50.0	30.0	1	0	1.8	0	3	0	3	0
Throttle_Valve_Actuator	m4	20.0	20.0	7.0	7.0	7.0	1	0	1.4	0	1.4	0	1.4	0
ABS/Traction_Hydraulic_Pump	m5	600.0	600.0	1.0	1.0	1.0	1	0	6	0	6	0	6	0
Windshield_Wiper_Front	m6	150.0	90.0	100.0	100.0	10.0	0	1	0	9	0	150	0	150
Wash_Pump_Front	m7	5.0	5.0	1.0	1.0	1.0	1	0	0.05	0	0.05	0	0.05	0
Windshield_Wiper_Rear	m8	150.0	90.0	100.0	100.0	5.0	0	1	0	4.5	0	150	0	150
Wash_Pump_Rear	m9	5.0	5.0	1.0	1.0	1.0	1	0	0.05	0	0.05	0	0.05	0
Head_Lamp_Washer_Pump	m10	100.0	100.0	1.0	1.0	1.0	1	0	1	0	1	0	1	0
Pass_Comp_Blower	m11	500.0	220.0	100.0	100.0	50.0	0	1	0	110	0	500	0	500
Power_Window_Front_Left	m12	350.0	350.0	5.0	5.0	5.0	0	1	0	17.5	0	17.5	0	17.5
Power_Window_Front_Right	m13	350.0	350.0	1.0	1.0	1.0	0	1	0	3.5	0	3.5	0	3.5
Power_Window_Back_Left	m14	350.0	350.0	1.0	1.0	1.0	0	1	0	3.5	0	3.5	0	3.5
Power_Window_Back_Right	m15	350.0	350.0	1.0	1.0	1.0	0	1	0	3.5	0	3.5	0	3.5
Sun_Roof	m16	200.0	200.0	1.0	1.0	1.0	0	1	0	2	0	2	0	2
Power_Door_Lock_Front_Left	m17	100.0	100.0	0.2	0.2	0.2	1	0	0.2	0	0.2	0	0.2	0
Power_Door_Lock_Front_Right	m18	100.0	100.0	0.2	0.2	0.2	1	0	0.2	0	0.2	0	0.2	0
Power_Door_Lock_Back_Left	m19	100.0	100.0	0.2	0.2	0.2	1	0	0.2	0	0.2	0	0.2	0
Power_Door_Lock_Back_Right	m20	100.0	100.0	0.2	0.2	0.2	1	0	0.2	0	0.2	0	0.2	0
Power_Trunk_Pull-down/opener	m21	200.0	22.0	0.1	0.1	0.1	1	0	0.022	0	0.2	0	0.2	0
Headrest_Adjustment_Left	m22	60.0	60.0	0.1	0.1	0.1	1	0	0.06	0	0.06	0	0.06	0
Seat_Longitudinal_Adjustment_Left	m23	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Seat_Tilt_Adjustment_Left	m24	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0

Seat_Front_Edge_Height_Adjustment_Left	m25	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Seat_Rear_Edge_Height_Adjustment_Left	m26	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Lumbar_Pump_Left	m27	70.0	70.0	0.1	0.1	0.1	1	0	0.07	0	0.07	0	0.07	0
Headrest_Adjustment_Right	m28	60.0	60.0	0.1	0.1	0.1	1	0	0.06	0	0.06	0	0.06	0
Seat_Longitudinal_Adjustment_Right	m29	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Seat_Tilt_Adjustment_Right	m30	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Seat_Front_Edge_Height_Adjustment_Right	m31	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Seat_Rear_Edge_Height_Adjustment_Right	m32	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Lumbar_Pump_Right	m33	70.0	70.0	0.1	0.1	0.1	1	0	0.07	0	0.07	0	0.07	0
Power_Mirror_Horizontal_Left	m34	5.0	5.0	0.1	0.1	0.1	1	0	0.005	0	0.005	0	0.005	0
Power_Mirror_Horizontal_Right	m35	5.0	5.0	0.1	0.1	0.1	1	0	0.005	0	0.005	0	0.005	0
Power_Mirror_Vertical_Left	m36	5.0	5.0	0.1	0.1	0.1	1	0	0.005	0	0.005	0	0.005	0
Power_Mirror_Vertical_Right	m37	5.0	5.0	0.1	0.1	0.1	1	0	0.005	0	0.005	0	0.005	0
Antenna_Lift	m38	45.0	45.0	0.2	0.2	0.2	1	0	0.09	0	0.09	0	0.09	0
Power_Steering_Pump	m39	1000.0	1000.0	50.0	50.0	17.0	0	1	0	170	0	500	0	500
Engine_Cooling_Fans	m40	800.0	500.0	100.0	5.0	15.0	0	1	0	75	0	800	0	40
Water_Pump	m41	300.0	300.0	100.0	5.0	15.0	0	1	0	45	0	300	0	15
Emissions_Air_Pump	m42	300.0	300.0	8.0	8.0	3.0	0	1	0	9	0	24	0	24
Active_Suspension_Front_Left	m43	750.0	750.0	2.0	2.0	2.0	0	1	0	15	0	15	0	15
Active_Suspension_Front_Right	m44	750.0	750.0	2.0	2.0	2.0	0	1	0	15	0	15	0	15
Active_Suspension_Back_Left	m45	750.0	750.0	2.0	2.0	2.0	0	1	0	15	0	15	0	15
Active_Suspension_Back_Right	m46	750.0	750.0	2.0	2.0	2.0	0	1	0	15	0	15	0	15
Brake-by-Wire_Front_Left	m47	500.0	62.5	1.0	1.0	1.0	0	1	0	0.625	0	5	0	5
Brake-by-Wire_Front_Right	m48	500.0	62.5	1.0	1.0	1.0	0	1	0	0.625	0	5	0	5
Brake-by-Wire_Back_Left	m49	500.0	62.5	1.0	1.0	1.0	0	1	0	0.625	0	5	0	5

Brake-by-Wire_Back_Right	m50	500.0	62.5	1.0	1.0	1.0	0	1	0	0.625	0	5	0	5
A/C_Compressor_Pump	m51	4000.0	3000.0	100.0	5.0	25.0	0	0	0	0	0	0	0	0
Automatic_Tire_Pump	m52	100.0	100.0	0.1	0.1	0.1	1	0	0.1	0	0.1	0	0.1	0
Standard_Trans_Force_Moter	m53	200.0	200.0	2.0	2.0	2.0	1	0	4	0	4	0	4	0
Heating_System_Water_Pump	m54	50.0	50.0	0.0	80.0	10.0	1	0	5	0	0	0	40	0
									Total	Total	Total	Total	Total	Total
									22.192	617.5	18.57	2636.5	58.57	1591.5
									System	Totals				
									612.5625	1420.35	1038.708	3526.15	1228.708	3416.15

Load	ID	Max Power (W)	Nominal Power (W)	Summer Worst Case (%)	Winter Worst Case (%)	Year Average (%)	14 Volt	42 Volt	Average Power 14 Volts	Average Power 42 Volts	Summer Peak Power 14 Volts	Summer Peak Power 42 Volts	Winter Peak Power 14 Volts	Winter Peak Power 42 Volts
Starter_Motor_Solenoid	s1	50.0	50.0	0.1	0.1	0.1	0	1	0	0.05	0	0.05	0	0.05
Fuel_Injectors (1 thru 8)	s2	80.0	80.0	22.0	22.0	22.0	0	1	0	17.6	0	17.6	0	17.6
EGR_Valve	s10	10.0	10.0	40.0	40.0	40.0	1	0	4	0	4	0	4	0
Canister_Purge_Solenoid	s11	10.0	10.0	80.0	80.0	80.0	1	0	8	0	8	0	8	0
Canister_Vent_Solenoid	s12	10.0	10.0	6.0	6.0	6.0	1	0	0.6	0	0.6	0	0.6	0
Brake-to-shift_Solenoid	s13	10.0	10.0	2.0	2.0	0.5	1	0	0.05	0	0.2	0	0.2	0
Trans_Shift_Solenoid_1	s14	10.0	10.0	100.0	100.0	100.0	0	1	0	10	0	10	0	10
Trans_Shift_Solenoid_2	s15	10.0	10.0	100.0	100.0	100.0	0	0	0	0	0	0	0	0
Torq_Convtr_Clutch_Enable	s16	10.0	10.0	50.0	50.0	50.0	0	1	0	5	0	5	0	5
ABS/TC_Solenoids	s17	68.0	68.0	0.1	0.1	0.1	1	0	0.068	0	0.068	0	0.068	0
Horn_1	s18	80.0	80.0	0.1	0.1	0.1	1	0	0.08	0	0.08	0	0.08	0
Horn_2	s19	80.0	80.0	0.1	0.1	0.1	0	0	0	0	0	0	0	0
Power_Trunk_Release	s20	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Power_Fuel_Door_Release	s21	200.0	200.0	0.1	0.1	0.1	1	0	0.2	0	0.2	0	0.2	0
Electromag. Valve Engine (8 cyl)	s22	2000.0	2000.0	32.0	32.0	32.0	0	1	0	640	0	640	0	640
									Total	Total	Total	Total	Total	Total
									13.198	672.65	13.348	672.65	13.348	672.65

Load	ID	Max Power (W)	Nominal Power (W)	Summer Worst Case (%)	Winter Worst Case (%)	Year Average (%)	14 Volt	42 Volt	Average Power 14 Volts	Average Power 42 Volts	Summer Peak Power 14 Volts	Summer Peak Power 42 Volts	Winter Peak Power 14 Volts	Winter Peak Power 42 Volts
Low_Beam_Headlamp_Left	I1	55.0	55.0	100.0	100.0	20.0	1	0	11	0	55	0	55	0
Low_Beam_Headlamp_Right	I2	55.0	55.0	100.0	100.0	20.0	1	0	11	0	55	0	55	0
High_Beam_Headlamp_Left	I3	65.0	65.0	1.0	1.0	1.0	1	0	0.65	0	0.65	0	0.65	0
High_Beam_Headlamp_Right	I4	65.0	65.0	1.0	1.0	1.0	1	0	0.65	0	0.65	0	0.65	0
Parking_Lamp_Left	I5	5.0	5.0	100.0	100.0	5.0	1	0	0.25	0	5	0	5	0
Parking_Lamp_Right	I6	5.0	5.0	100.0	100.0	5.0	1	0	0.25	0	5	0	5	0
Blinking_Lamp_Front_Left	I7	21.0	21.0	50.0	50.0	5.0	1	0	1.05	0	10.5	0	10.5	0
Blinking_Lamp_Front_Right	I8	21.0	21.0	50.0	50.0	5.0	1	0	1.05	0	10.5	0	10.5	0
Fog_Lamp_Front_Left	I9	55.0	55.0	100.0	100.0	5.0	1	0	2.75	0	55	0	55	0
Fog_Lamp_Front_Right	I10	55.0	55.0	100.0	100.0	5.0	1	0	2.75	0	55	0	55	0
Tail_Lamp_Left	I11	5.0	5.0	100.0	100.0	20.0	1	0	1	0	5	0	5	0
Tail_Lamp_Right	I12	5.0	5.0	100.0	100.0	20.0	1	0	1	0	5	0	5	0
Reversing_Lamp_Left	I13	21.0	21.0	1.0	1.0	1.0	1	0	0.21	0	0.21	0	0.21	0
Reversing_Lamp_Right	I14	21.0	21.0	1.0	1.0	1.0	1	0	0.21	0	0.21	0	0.21	0
Brake_Lamp_Left	I15	21.0	21.0	100.0	100.0	15.0	1	0	3.15	0	21	0	21	0
Brake_Lamp_Right	I16	21.0	21.0	100.0	100.0	15.0	1	0	3.15	0	21	0	21	0
Blinking_Lamp_Rear_Left	I17	21.0	21.0	50.0	50.0	5.0	1	0	1.05	0	10.5	0	10.5	0
Blinking_Lamp_Rear_Right	I18	21.0	21.0	50.0	50.0	5.0	1	0	1.05	0	10.5	0	10.5	0
Fog_Lamp_Rear_Left	I19	21.0	21.0	100.0	100.0	5.0	1	0	1.05	0	21	0	21	0
Fog_Lamp_Rear_Right	I20	21.0	21.0	100.0	100.0	5.0	1	0	1.05	0	21	0	21	0
License_Plate_Lamp_Left	I21	5.0	5.0	100.0	100.0	20.0	1	0	1	0	5	0	5	0
License_Plate_Lamp_Right	I22	5.0	5.0	100.0	100.0	20.0	1	0	1	0	5	0	5	0
Cabin_Ceiling_Lamp_1	I23	10.0	10.0	1.0	1.0	0.3	1	0	0.025	0	0.1	0	0.1	0
Cabin_Ceiling_Lamp_2	I24	5.0	5.0	1.0	1.0	0.3	0	0	0	0	0	0	0	0
Cabin_Ceiling_Lamp_3	I25	10.0	10.0	1.0	1.0	0.3	0	0	0	0	0	0	0	0
Cabin_Ceiling_Lamp_4	I26	10.0	10.0	1.0	1.0	0.3	0	0	0	0	0	0	0	0
Map_Lamp	I27	5.0	5.0	1.0	1.0	0.3	1	0	0.0125	0	0.05	0	0.05	0
Glove_Compartment_Lamp	I28	10.0	10.0	1.0	1.0	0.3	1	0	0.025	0	0.1	0	0.1	0
Engine_Compartment_Lamp	I29	10.0	10.0	0.1	0.1	0.1	1	0	0.01	0	0.01	0	0.01	0
Trunk_Compartment_Lamp	I30	10.0	10.0	0.2	0.2	0.2	1	0	0.02	0	0.02	0	0.02	0

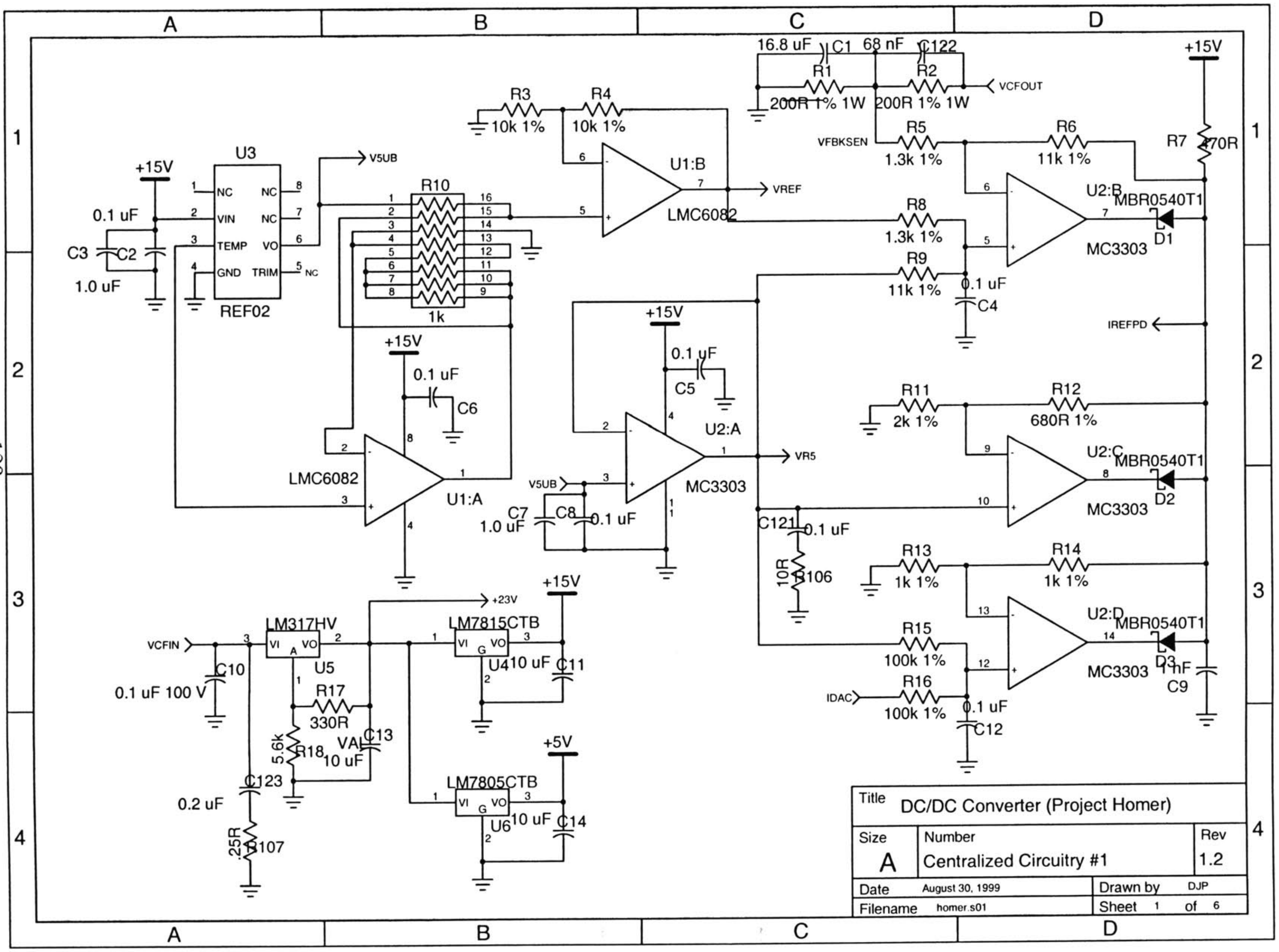
Driver_Door_Exit_Lamp	I31	5.0	5.0	0.2	0.2	0.2	1	0	0.01	0	0.01	0	0.01	0
									Total	Total	Total	Total	Total	Total
									46.4225	0	378.01	0	378.01	0

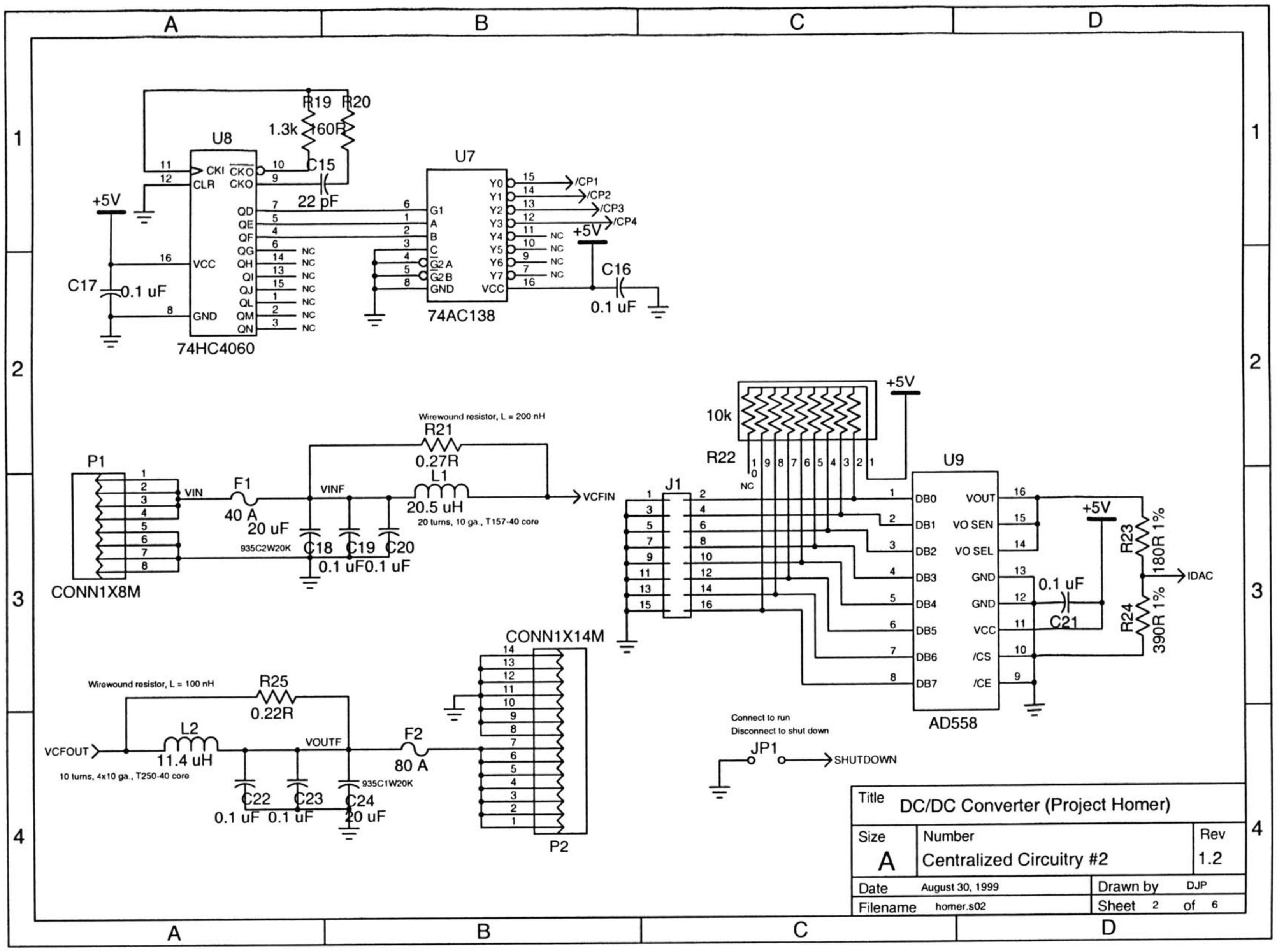
Load	ID	Max Power (W)	Nominal Power (W)	Summer Worst Case (%)	Winter Worst Case (%)	Year Average (%)	14 Volt	42 Volt	Average Power 14 Volts	Average Power 42 Volts	Summer Peak Power 14 Volts	Summer Peak Power 42 Volts	Winter Peak Power 14 Volts	Winter Peak Power 42 Volts
Heated_Rear_Window	h1	400.0	400.0	0.0	100.0	2.0	0	1	0	8	0	0	0	400
Cigar_Lighter	h2	120.0	120.0	0.3	0.3	0.3	1	0	0.36	0	0.36	0	0.36	0
Power_Mirror_Heater_Left	h3	40.0	40.0	0.0	100.0	2.0	1	0	0.8	0	0	0	40	0
Power_Mirror_Heater_Right	h4	40.0	40.0	0.0	100.0	2.0	1	0	0.8	0	0	0	40	0
Seat_Heater_Left	h5	130.0	90.0	0.0	100.0	4.0	0	1	0	3.6	0	0	0	130
Seat_Heater_Right	h6	130.0	90.0	0.0	100.0	2.0	0	1	0	1.8	0	0	0	130
Washer_Jet_Heater	h7	10.0	10.0	0.0	100.0	2.0	1	0	0.2	0	0	0	10	0
Washer_Tube_Heater	h8	60.0	60.0	0.0	100.0	2.0	1	0	1.2	0	0	0	60	0
Heated_Oxygen_Sensor	h9	40.0	40.0	100.0	100.0	100.0	1	0	40	0	40	0	40	0
Heated_Windshield	h10	1000.0	500.0	0.0	20.0	0.1	0	1	0	0.5	0	0	0	200
Heated_Catalytic_Converter	h11	3000.0	3000.0	2.5	2.5	2.0	0	1	0	60	0	75	0	75
Seat_Heater_Rear	h12	250.0	180.0	0.0	30.0	1.0	0	1	0	1.8	0	0	0	75
Parking_Heating	h13	250.0	70.0	0.0	0.0	1.0	1	0	0.7	0	0	0	0	0
									Total	Total	Total	Total	Total	Total
									44.06	75.7	40.36	75	190.36	1010

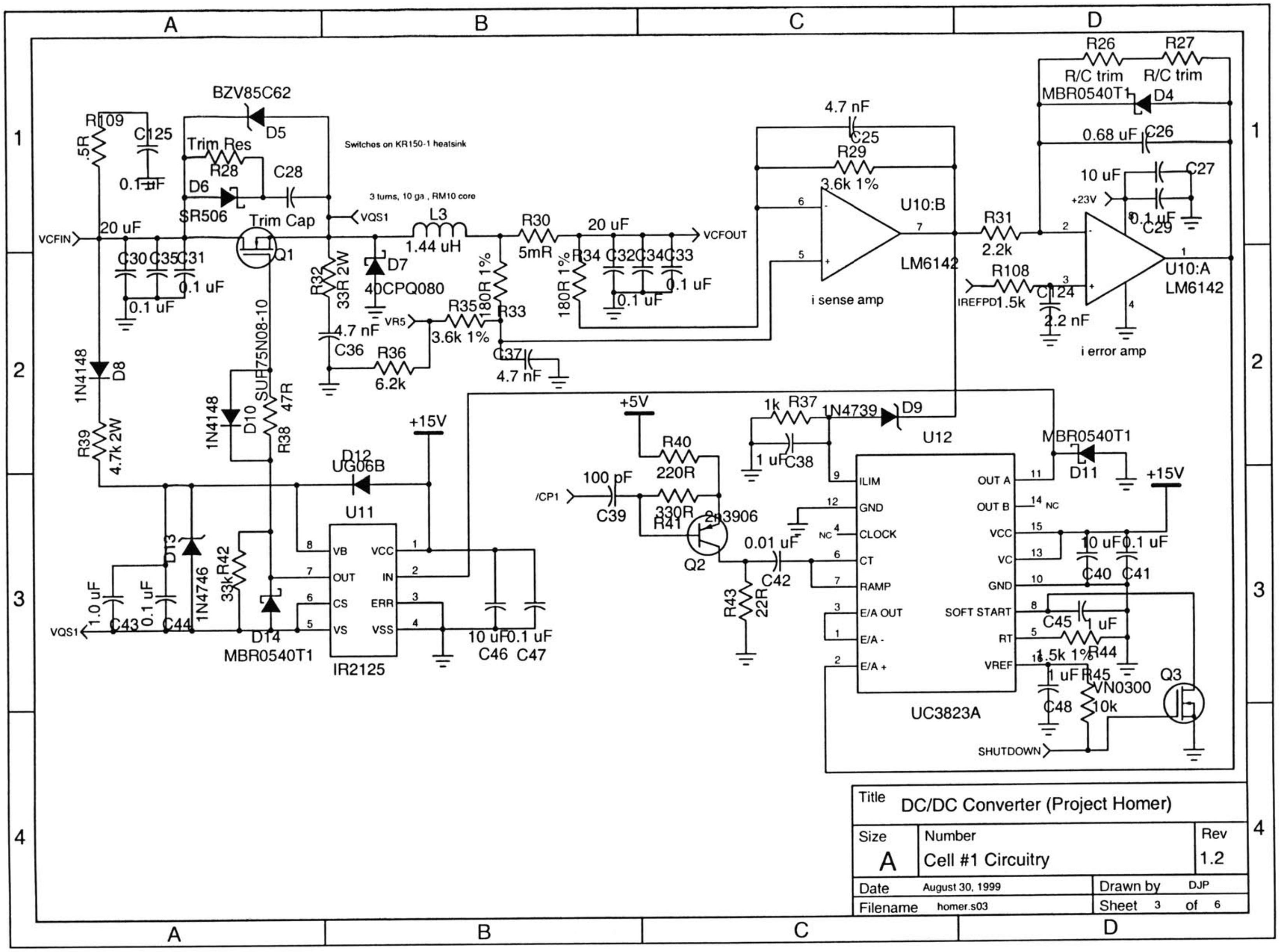
Load	ID	Max Power (W)	Nominal Power (W)	Summer Worst Case (%)	Winter Worst Case (%)	Year Average (%)	14 Volt	42 Volt	Average Power 14 Volts	Average Power 42 Volts	Summer Peak Power 14 Volts	Summer Peak Power 42 Volts	Winter Peak Power 14 Volts	Winter Peak Power 42 Volts
Engine_Control_Unit	e1	17.0	17.0	100.0	100.0	100.0	1	0	17	0	17	0	17	0
Transmission_Control	e2	40.0	40.0	100.0	100.0	100.0	1	0	40	0	40	0	40	0
ABS/Traction_Control	e3	7.0	7.0	100.0	100.0	100.0	1	0	7	0	7	0	7	0
Cruise_Control	e4	30.0	30.0	7.0	7.0	7.0	1	0	2.1	0	2.1	0	2.1	0
Steering_Control	e5	14.0	14.0	100.0	100.0	100.0	1	0	14	0	14	0	14	0
Airbag/Seatbelt_Tensioner	e5	20.0	20.0	1.0	1.0	1.0	1	0	0.2	0	0.2	0	0.2	0
Automatic_Climate_Control	e7	7.0	7.0	100.0	100.0	100.0	1	0	7	0	7	0	7	0
IR_Central_Locking	e8	5.0	5.0	1.0	1.0	1.0	1	0	0.05	0	0.05	0	0.05	0
Navigation_Aid	e9	70.0	70.0	100.0	100.0	100.0	1	0	70	0	70	0	70	0
Lamp_Monitor_&_Control	e10	10.0	10.0	100.0	100.0	100.0	1	0	10	0	10	0	10	0
Anti-theft_Warning_System	e11	10.0	10.0	100.0	100.0	100.0	1	0	10	0	10	0	10	0
Active_Suspension_Control	e12	10.0	10.0	100.0	100.0	100.0	0	0	0	0	0	0	0	0
Telephone	e13	6.0	6.0	100.0	100.0	2.0	1	0	0.12	0	6	0	6	0
Tire_Pressure_Monitor	e14	10.0	10.0	100.0	100.0	100.0	1	0	10	0	10	0	10	0
									total	total	total	total	total	total
									187.47	0	193.35	0	193.35	0

Load	ID	Max Power (W)	Nominal Power (W)	Summer Worst Case (%)	Winter Worst Case (%)	Year Average (%)	14 Volt	42 Volt	Average Power 14 Volts	Average Power 42 Volts	Summer Peak Power 14 Volts	Summer Peak Power 42 Volts	Winter Peak Power 14 Volts	Winter Peak Power 42 Volts
Instrument_Cluster	o1	30	30	100	100	100	1	0	30	0	30	0	30	0
Radio/Tape/CD	o2	70.0	15.0	100.0	100.0	50.0	1	0	7.5	0	70	0	70	0
Amplifier	o3	100.0	25.0	100.0	100.0	50.0	0	1	0	12.5	0	100	0	100
Spark_Ignition	o4	40.0	40.0	100.0	100.0	100.0	0	1	0	40	0	40	0	40
Power_Outlet_(Fax_etc.)	o5	100.0	100.0	2.0	2.0	2.0	0	1	0	2	0	2	0	2
All_Weather_Night_Vision	o6	100.0	100.0	0.0	0.0	50.0	1	0	50	0	0	0	0	0
Active_Engine_Mount	o7	70.0	70.0	0.1	0.1	0.1	1	0	0.07	0	0.07	0	0.07	0
Voice_Control	o8	70.0	70.0	1.0	1.0	10.0	0	0	0	0	0	0	0	0
Side_Vision_Lane_Change	o9	15.0	15.0	100.0	100.0	10.0	1	0	1.5	0	15	0	15	0
Backup_Parking_Assist	o10	15.0	15.0	0.0	0.0	1.0	1	0	0.15	0	0	0	0	0
Motor_Management_4_Cyl	o11	260.0	200.0	100.0	100.0	100.0	0	0	0	0	0	0	0	0
Motor_Management_6_Cyl	o12	280.0	210.0	100.0	100.0	100.0	1	0	210	0	280	0	280	0
Motor_Management_8_Cyl	o13	300.0	220.0	100.0	100.0	100.0	0	0	0	0	0	0	0	0
Motor_Management_12_Cyl	o14	400.0	280.0	100.0	100.0	100.0	0	0	0	0	0	0	0	0
									Total	Total	Total	Total	Total	Total
									299.22	54.5	395.07	142	395.07	142

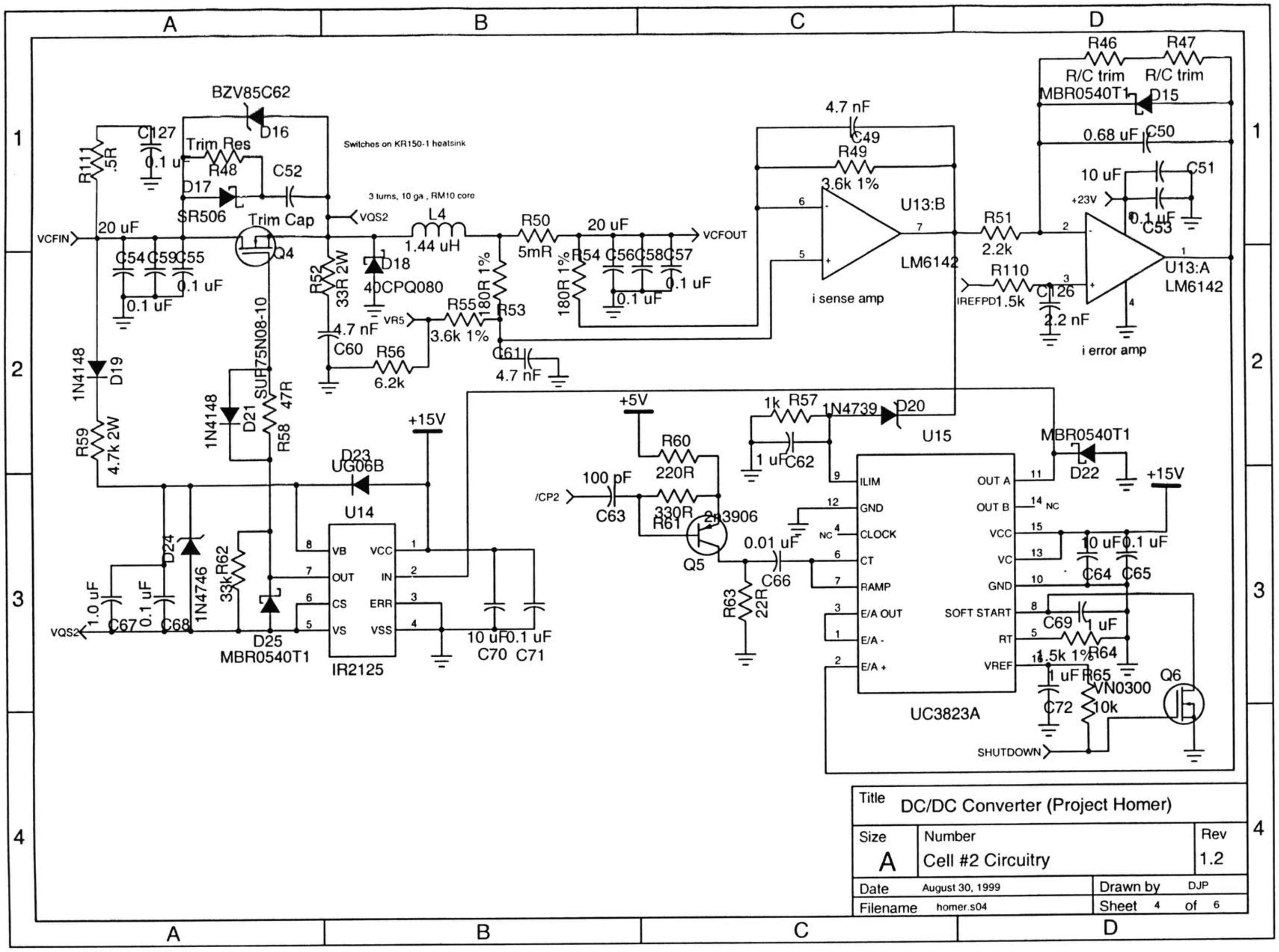
Appendix C Schematics



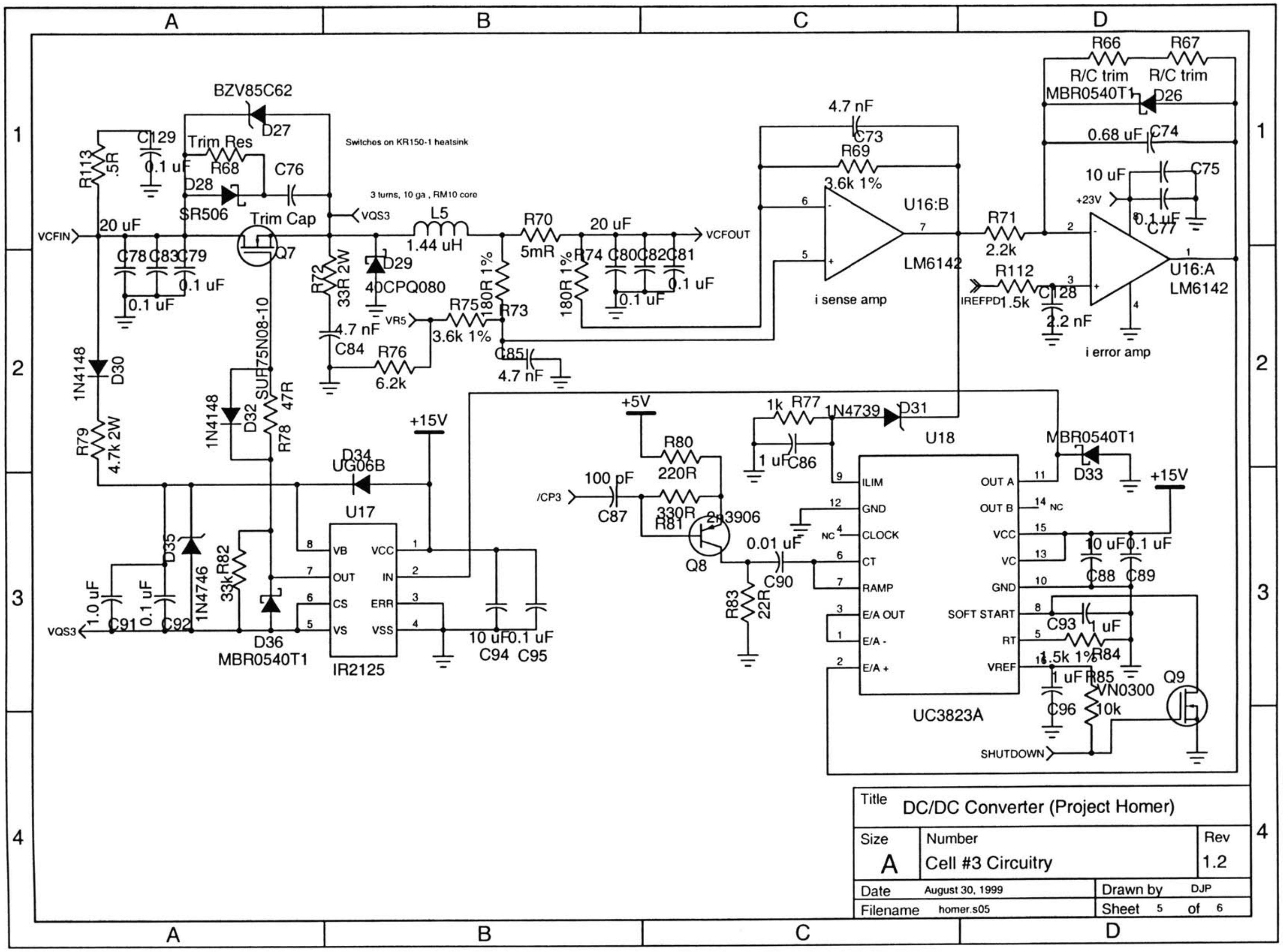

U8
74HC4060
R19
1.3k
R20
160R
C15
22 pF
+5V
C17
0.1 uF
U7
74AC138
/CP1
/CP2
/CP3
/CP4
C16
0.1 uF
Wirewound resistor, L = 200 nH
R21
0.27R
L1
20.5 uH
20 turns, 10 ga., T157-40 core
P1
CONN1X8M
VIN
F1
40 A
VINF
20 uF
935C2W20K
C18
C19
C20
0.1 uF
0.1 uF
VCFIN
10k
R22
J1
U9
AD558
R23
180R 1%
R24
390R 1%
C21
0.1 uF
IDAC
Wirewound resistor, L = 100 nH
R25
0.22R
VCFOUT
L2
11.4 uH
10 turns, 4x10 ga., T250-40 core
VOUTF
F2
80 A
935C1W20K
C22
C23
C24
0.1 uF
0.1 uF
20 uF
CONN1X14M
P2
Connect to run
Disconnect to shut down
JP1
SHUTDOWN
Title
DC/DC Converter (Project Homer)
Size
A
Number
Centralized Circuitry #2
Rev
1.2
Date
August 30, 1999
Drawn by
DJP
Filename
homer.s02
Sheet 2 of 6



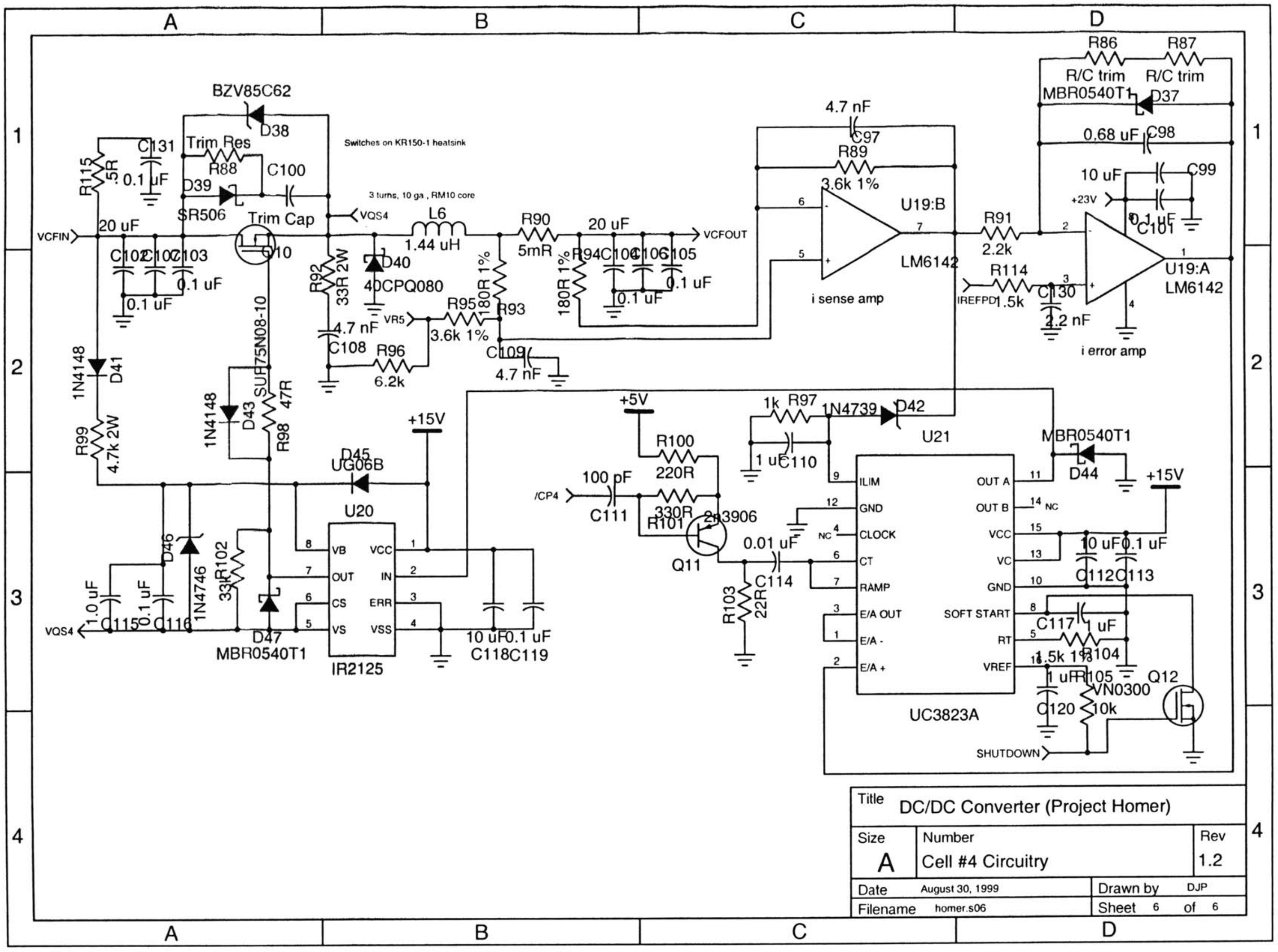
Title	DC/DC Converter (Project Homer)	
Size A	Number Cell #1 Circuitry	Rev 1.2
Date August 30, 1999		Drawn by DJP
Filename homer.s03		Sheet 3 of 6



Title	DC/DC Converter (Project Homer)	
Size A	Number Cell #2 Circuitry	Rev 1.2
Date August 30, 1999		Drawn by DJP
Filename homer.s04		Sheet 4 of 6



Title	DC/DC Converter (Project Homer)	
Size A	Number Cell #3 Circuitry	Rev 1.2
Date August 30, 1999		Drawn by DJP
Filename homer.s05		Sheet 5 of 6



Title	DC/DC Converter (Project Homer)	
Size A	Number Cell #4 Circuitry	Rev 1.2
Date August 30, 1999		Drawn by DJP
Filename homer.s06		Sheet 6 of 6

Appendix D TimCAD Functions

Following is an abridged listing of the functions created for TimCAD and the page where the function begins. Included are all functions that are used in the design of the converter.

DesignFilter3 designs the third type of filter ensuring that the EMI limits are met.

PowLoss member of class *CSchottkySet and CDiodeSet* 193
PowLoss determines the maximum power loss in the Schottky Diode or Diode if used as the second switch in the converter.

PowLoss member of class *CFETSet* 195
PowLoss determines the maximum power loss in the FET if used as either switch in the converter.

The C++ code is as follows

```
/*////////////////////////////////////////////////////////////////////////
CTIMCADDoc::OnRun()

OnRun is the function that starts the optimization process. Through this
function all the other major function in the program are called.

*////////////////////////////////////////////////////////////////////////
void CTIMCADDoc::OnRun()
{
        int             CellNum, //The number of cells
                        turns, //Number of turns in the converter inductor
                        Gauge, //The gauge of wire used in the converter inductor
                        Cores, //The number of converter inductors in parallel
                        S2sel = 0, //Whether switch two is 1=MOSFET 2=Schottky 3=Diode
                        Mode, //1=Continuous Mode 2=Discontinuous Mode 3=Combo determined by
                                //voltages
                        Cap2InIndex, Cap2OutIndex, Cap1InIndex, L2InIndex, L2OutIndex,
                        CapD1InIndex, CapD2InIndex, L1InIndex, Cap1OutIndex, CapD1OutIndex,
                        CapD2OutIndex, L1OutIndex, //The part index for various capacitors and
                                //inductors
                        SinkNum, SinkNum1, //The part index for heatsinks
                        NumOfCapsIn1, NumOfCapsIn2, NumOfCapsOut2, NumOfCapsDIn1,
                        NumOfCapsDIn2, NumOfCapsOut1, NumOfCapsDOut1, NumOfCapsDOut2,
                                //The number of paralleled capacitors
                        In2Gauge, Out2Gauge, In1Gauge, Out1Gauge, //The wire gauge for the EMI
                                //inductors
                        In2Turns, Out2Turns, In1Turns, Out1Turns, //The number of turns on the
                                //EMI inductors
                        In2Windings, Out2Windings, In1Windings, Out1Windings, //The number of
                                //sets of turns on the EMI inductors
                        Length, //The Length of extrusion needed
                        FilterTypeIn, FilterTypeOut, //The type of filter used
                        ans, start, Good, HSneeded, Lneeded, LCellneeded, BusCapneeded,
                        Priceneeded, Weightneeded, Volumeneeded; //Arguments for dialog boxes

        double          Freq, //The Converter switching frequency
                        Inductor, //The total inductance of the power stage
                        CurrentPeak, //The peak current through the converter
                        Irms, //The rms current through the converter inductor
                        S1rms, //the rms current through switch 1
```

```
                    S2rms, //the rms current through switch 2
                    FETPcon, //The conduction power losses in the MOSFET
                    FETPsw, //The switching power losses in the MOSFET
                    FETPrice, //The Price of the MOSFET
                    FETPtot, //The total power loss in the FET
                    Bpksave, //The peak magnetic field in the converter inductor
                    Jwiresave, //The current density in the converter inductor
                    Pcoresave, //The core power loss in the converter inductor
                    Pwindsave, //The winding power loss in the converter inductor
                    deltaTsave, //The max. temperature rise in the converter inductor
                    S2Pcond = 0, //The conduction power losses in switch 2
                    S2Psw = 0, //The switching power losses in switch 2
                    S2Price = 0, //The price of switch 2
                    DeltaT, DeltaT1, //The temperature rise of the heatsinks
                    FETTemp, //The temperature rise of the MOSFET
                    S2Temp, //The temperature rise of switch 2
                    Iinitial, //The amount of current through the MOSFET when it turns on
                    Qin, Qout, //The Q of the filter, for filter type 1
                    L2In, L2Out, L3In, L3Out, L1In, L1Out, //The EMI inductance
                    InPowLoss, Out2PowLoss, In1PowLoss, Out1PowLoss, //The power loss
                              //of the EMI inductors
                    Rin, Rout, //Resistor values in filter type 1
                    Cap1, Cap1Price, Cap1Weight, Cap1Volume, L2Price, L2Weight,
                    L2Volume, //The net values used in the input filter
                    PCboardPrice, //The price of the PCboard
                    Cap2InArea, Cap2OutArea, Cap1Area, L2Area, SinkArea, L1InArea,
                    L1OutArea, //The area on the PCboard that a component uses
                    SinkWeight, SinkVolume, SinkPrice, //Properties of the heatsink used
                    Rd1In, Rd2In, Rd1Out, Rd2Out, //Damping resistors used in filter type 2
                    L1InPrice, L1InWeight, L1InVolume, L1OutPrice, L1OutWeight,
                    L1OutVolume, //Properties of the EMI inductors used
                    InFilterPrice, OutFilterPrice, InFilterWeight, OutFilterWeight,
                    InFilterVolume, OutFilterVolume, //Properties of the filters used
                    RippleRatio, //The ripple ratio for one cell under nominal voltages
                    TotalPrice, TotalWeight, TotalVolume; //The total price, weight and,
                              //volume of the converter

        double      waveform[1200], //Vectors of the current in the inductor
                    waveformS2[1200]; //Vectors of the current in switch 2

        CString     FETName, //The name of the MOSFET
                    S2Name = _T(""), //The name of switch 2
                    Cap1Name, //The name of a Capacitor used
                    L2Name, L1InName, L1OutName, //The name of inductors used
                    SinkName; //The name of the heatsink used

        bool        Valid; //True if current design is Valid

        long        IndNum; //The index of the converter inductor

srand( (unsigned)time( NULL ) );

if (!m_ConfigurationSet.IsOpen())
        m_ConfigurationSet.Open();
if (!m_Configuration2Set.IsOpen())
        m_Configuration2Set.Open();
```

```
if (!m_InductSet.IsOpen())
        m_InductSet.Open();
if (!m_ResultSet.IsOpen())
        m_ResultSet.Open();
if (!m_ResultBSet.IsOpen())
        m_ResultBSet.Open();
if (!m_ResultCSet.IsOpen())
        m_ResultCSet.Open();
if (!m_ResultDSet.IsOpen())
        m_ResultDSet.Open();
if (!m_WireSet.IsOpen())
        m_WireSet.Open();
if (!m_SchottkySet.IsOpen())
        m_SchottkySet.Open();
if (!m_HeatSinkSet.IsOpen())
        m_HeatSinkSet.Open();
if (!m_CapacitorSet.IsOpen())
        m_CapacitorSet.Open();
if (!m_DiodeSet.IsOpen())
        m_DiodeSet.Open();
if (!m_EMILimitsSet.IsOpen())
        m_EMILimitsSet.Open();
if (!m_EMIInductorSet.IsOpen())
        m_EMIInductorSet.Open();

m_EMILimitsSet.m_strSort = "Frequency ASC";
        m_EMILimitsSet.Requery();

ans = MessageBox(NULL, "Do you wish to erase previous records?","TimCAD", MB_YESNOCANCEL
);
if (ans == IDYES)
{
        m_ResultBSet.m_strFilter = "";
        m_ResultBSet.Requery();
        m_ResultCSet.m_strFilter = "";
        m_ResultCSet.Requery();
        m_ResultDSet.m_strFilter = "";
        m_ResultDSet.Requery();
        m_ResultSet.DeleteAll();
        m_ResultBSet.DeleteAll();
        m_ResultCSet.DeleteAll();
        m_ResultDSet.DeleteAll();
        start = 0;
}
else if (ans == IDCANCEL)
        return;
else if (ans == IDNO)
{
        if (!(m_ResultSet.IsEOF() && m_ResultSet.IsBOF()))
        {
                m_ResultSet.m_strSort = "IDNum";
                m_ResultSet.Requery();
                m_ResultSet.MoveLast();
                start = m_ResultSet.m_IDnum + 1;
                m_ResultSet.m_strSort = "";
                m_ResultSet.Requery();
```

```
		}
		else start = 0;
	}
	Good = HSneeded = Lneeded = LCellneeded = BusCapneeded = Priceneeded = 0;
	Weightneeded = Volumeneeded = 0;

	int iteration;
	CCountDownDlg m_CountDown(m_ConfigurationSet.m_Iterations);
	m_CountDown.Create(IDD_COUNTDOWN);
	m_CountDown.Begin();
	for (iteration = 1+start; iteration <= m_ConfigurationSet.m_Iterations+start; iteration++)
	{
		m_CountDown.dec();
		Valid = TRUE;

		GetRandomValues(CellNum, Freq, Inductor, Mode, RippleRatio);

		//This assumes that the worst case power losses will be at the max Vin and Vout
		Valid = GetInductorCurrentInfo(CellNum, Freq, Inductor,m_ConfigurationSet.m_Voutmax,
			m_ConfigurationSet.m_Vinmax, CurrentPeak, Irms, S1rms, S2rms, waveform,
			waveformS2, Iinitial, Mode);

		if (Valid)
		Valid = PowerStage(CurrentPeak, Freq, S1rms, S2rms, Inductor, Irms, waveform, waveformS2,
			FETName, FETPcon, FETPrice, FETPsw, FETPtot, turns, Gauge, Bpksave, Jwiresave,
			Pcoresave, Pwindsave, deltaTsave, S2Pcond, S2Psw, S2Price, S2Name, S2sel, DeltaT,
			FETTemp, S2Temp, Iinitial, Mode, CellNum, IndNum, SinkNum, Cores, SinkNum1,
			DeltaT1, Length, HSneeded, LCellneeded);

		if (Valid)
		    Valid = FilterStage(CellNum, Freq, Inductor, Mode, Qin, Qout, L2In, L2Out, L3In, L3Out,
			L1In, L1Out, InPowLoss, Out2PowLoss, In1PowLoss, Out1PowLoss, Rin, Rout, Rd1In,
			Rd2In, Rd1Out, Rd2Out, Cap2InIndex, Cap2OutIndex, Cap1InIndex, Cap1OutIndex,
			CapD1InIndex, CapD2InIndex, CapD1OutIndex, CapD2OutIndex, L2InIndex,
			L2OutIndex, L1InIndex, L1OutIndex, In2Gauge, Out2Gauge, In1Gauge, Out1Gauge,
			In2Turns, Out2Turns, In1Turns, Out1Turns, In1Windings, In2Windings, Out1Windings,
			Out2Windings, Lneeded, BusCapneeded, NumOfCapsIn1, NumOfCapsIn2,
			NumOfCapsDIn1, NumOfCapsDIn2, NumOfCapsOut2, NumOfCapsOut1,
			NumOfCapsDOut1, NumOfCapsDOut2, FilterTypeIn, FilterTypeOut, InFilterPrice,
			OutFilterPrice, InFilterWeight, OutFilterWeight, InFilterVolume, OutFilterVolume);
		if (Valid)
		{
			m_CapacitorSet.m_Param = Cap2InIndex;
			m_CapacitorSet.m_strFilter = "rank = ?";
			m_CapacitorSet.Requery();
			Cap2InArea = m_CapacitorSet.m_Area*NumOfCapsIn2;

			m_CapacitorSet.m_Param = Cap2OutIndex;
			m_CapacitorSet.m_strFilter = "rank = ?";
			m_CapacitorSet.Requery();
			Cap2OutArea = m_CapacitorSet.m_Area*NumOfCapsIn2;

			m_CapacitorSet.m_Param = Cap1InIndex;
			m_CapacitorSet.m_strFilter = "rank = ?";
			m_CapacitorSet.Requery();
			Cap1Name = m_CapacitorSet.m_CapName;
```

```
Cap1 = m_CapacitorSet.m_Capacitance*NumOfCapsIn1;
Cap1Price = m_CapacitorSet.m_Price*NumOfCapsIn1;
Cap1Weight = m_CapacitorSet.m_Weight*NumOfCapsIn1;
Cap1Volume = m_CapacitorSet.m_Volume*NumOfCapsIn1;
Cap1Area = m_CapacitorSet.m_Area*NumOfCapsIn1;
m_CapacitorSet.m_Param = Cap1OutIndex;
m_CapacitorSet.m_strFilter = "rank = ?";
m_CapacitorSet.Requery();

m_EMIInductorSet.m_Param = L2InIndex;
m_EMIInductorSet.m_strFilter = "rank = ?";
m_EMIInductorSet.Requery();
L2Name = m_EMIInductorSet.m_CoreName;
L2Price = m_EMIInductorSet.m_Price;
L2Weight = m_EMIInductorSet.m_Weight;
L2Volume = m_EMIInductorSet.m_Volume;
L2Area = m_EMIInductorSet.m_Area;
m_EMIInductorSet.m_Param = L1InIndex;
m_EMIInductorSet.m_strFilter = "rank = ?";
m_EMIInductorSet.Requery();
L1InName = m_EMIInductorSet.m_CoreName;
L1InPrice = m_EMIInductorSet.m_Price;
L1InWeight = m_EMIInductorSet.m_Weight;
L1InVolume = m_EMIInductorSet.m_Volume;
L1InArea = m_EMIInductorSet.m_Area;
m_EMIInductorSet.m_Param = L2InIndex;
m_EMIInductorSet.m_strFilter = "rank = ?";
m_EMIInductorSet.Requery();
L1OutName = m_EMIInductorSet.m_CoreName;
L1OutPrice = m_EMIInductorSet.m_Price;
L1OutWeight = m_EMIInductorSet.m_Weight;
L1OutVolume = m_EMIInductorSet.m_Volume;
L1OutArea = m_EMIInductorSet.m_Area;
m_EMIInductorSet.m_Param = L2OutIndex;
m_EMIInductorSet.m_strFilter = "rank = ?";
m_EMIInductorSet.Requery();

m_InductSet.m_Param = IndNum;
m_InductSet.m_strFilter = "InductorID = ?";
m_InductSet.Requery();

m_HeatSinkSet.m_Param = SinkNum;
m_HeatSinkSet.m_strFilter = "rank = ?";
m_HeatSinkSet.Requery();
SinkArea = m_HeatSinkSet.m_Area;
SinkWeight = m_HeatSinkSet.m_Weight;
SinkVolume = m_HeatSinkSet.m_Volume;
SinkPrice = m_HeatSinkSet.m_Price;
SinkName = m_HeatSinkSet.m_Name;
if ((m_HeatSinkSet.m_Count == 0) && (Length == 0))
        Valid = FALSE;
if (SinkNum1 != -1)
{
        m_HeatSinkSet.m_Param = SinkNum1;
        m_HeatSinkSet.m_strFilter = "rank = ?";
        m_HeatSinkSet.Requery();
```

```
            SinkArea = SinkArea + m_HeatSinkSet.m_Area;
            SinkWeight = SinkWeight + m_HeatSinkSet.m_Weight;
            SinkVolume = SinkVolume + m_HeatSinkSet.m_Volume;
            SinkPrice = SinkPrice + m_HeatSinkSet.m_Price;
            SinkName += " & ";
            SinkName += m_HeatSinkSet.m_Name;
}
if (Length != 0)
{
            SinkArea = SinkArea*Length/150;
            SinkWeight = SinkWeight*Length/150;
            SinkVolume = SinkVolume*Length/150;
            SinkPrice = SinkPrice*Length/150;
}

PCboardPrice = m_ConfigurationSet.m_PCboard* (Cap1Area+m_CapacitorSet.m_Area*
            NumOfCapsOut1+L2Area+m_EMIInductorSet.m_Area+m_ConfigurationSet.m
            _MasterArea+ CellNum* (Cap2InArea+ Cap2OutArea+
            m_CapacitorSet.m_Area* NumOfCapsOut2+ m_InductSet.m_Area* Cores+
            SinkArea));

if (m_Configuration2Set.m_UseMaxs)
{
      TotalPrice = PCboardPrice+InFilterPrice+OutFilterPrice+
          m_ConfigurationSet.m_MasterPrice+CellNum*m_ConfigurationSet.m_CellPrice
          +CellNum*(FETPrice+m_InductSet.m_Price*Cores+S2Price+SinkPrice);

      TotalWeight = InFilterWeight + OutFilterWeight +
          m_ConfigurationSet.m_MasterWeight + CellNum *
          m_ConfigurationSet.m_CellWeight + CellNum* (m_InductSet.m_Weight*
          Cores+SinkWeight);

      TotalVolume = InFilterVolume+ OutFilterVolume+
          m_ConfigurationSet.m_MasterVolume+ CellNum*
          m_ConfigurationSet.m_CellVolume+ CellNum* (m_InductSet.m_Volume*
          Cores+SinkVolume);

      if (TotalPrice > m_Configuration2Set.m_MaxPrice)
      {
            Priceneeded++;
            Valid = FALSE;
      }
      if (TotalWeight > m_Configuration2Set.m_MaxWeight)
      {
            Weightneeded++;
            Valid = FALSE;
      }
      if (TotalVolume > m_Configuration2Set.m_MaxVolume)
      {
            Volumeneeded++;
            Valid = FALSE;
      }
}
if (Valid)
{
    Good++;
```

```
                    m_ResultSet.AddRecord(FETPrice, (long) iteration, m_InductSet.m_Weight*Cores,
                         m_InductSet.m_Price*Cores, S2Price, SinkWeight, SinkVolume, SinkPrice,
                         Cap1Price, m_CapacitorSet.m_Price*NumOfCapsOut1, Cap1Weight,
                         m_CapacitorSet.m_Weight*NumOfCapsOut1, Cap1Volume,
                         m_CapacitorSet.m_Volume* NumOfCapsOut1, L2Price,
                         m_EMIInductorSet.m_Price,  L2Weight, m_EMIInductorSet.m_Weight,
                         L2Volume, m_EMIInductorSet.m_Volume,
                         m_ConfigurationSet.m_MasterPrice + CellNum*
                         m_ConfigurationSet.m_CellPrice, m_ConfigurationSet.m_MasterWeight+
                         CellNum*m_ConfigurationSet.m_CellWeight,
                         m_ConfigurationSet.m_MasterVolume+ CellNum*
                         m_ConfigurationSet.m_CellVolume, m_ConfigurationSet.m_MasterArea+
                         CellNum*m_ConfigurationSet.m_CellArea, m_InductSet.m_Volume*Cores,
                         m_InductSet.m_Area*Cores, PCboardPrice, CellNum,
                         m_Configuration2Set.m_Price, m_Configuration2Set.m_Weight,
                         m_Configuration2Set.m_Volume, Length, InFilterPrice, OutFilterPrice,
                         InFilterWeight, OutFilterWeight, InFilterVolume, OutFilterVolume);

                    m_ResultBSet.AddRecord(CellNum, FETName, FETPcon, FETPsw, FETPtot, Freq,
                         (long) iteration, Inductor, m_InductSet.m_CoreName, turns, Gauge, Bpksave,
                         Jwiresave, Pcoresave, Pwindsave, deltaTsave, S2Pcond, S2Psw, S2Name, S2sel,
                         SinkName, DeltaT, FETTemp, S2Temp, m_ConfigurationSet.m_Temperature,
                         Mode, Cores, RippleRatio);

                    m_ResultCSet.AddRecord((long) iteration, Qin, Qout, L2In, L2Out, L3In, L3Out,
                         InPowLoss, Out2PowLoss, Rin, Rout, In2Gauge, Out2Gauge, In2Turns,
                         Out2Turns, In2Windings, Out2Windings, Cap1Name,
                         m_CapacitorSet.m_CapName, Cap1, m_CapacitorSet.m_Capacitance, L2Name,
                         m_EMIInductorSet.m_CoreName, NumOfCapsIn1, NumOfCapsIn2,
                         NumOfCapsOut1, NumOfCapsOut2, NumOfCapsDIn1, NumOfCapsDIn2,
                         In1Gauge, In1Turns,  In1Windings, FilterTypeIn, FilterTypeOut,Cap2InIndex,
                         Cap2OutIndex);

                    m_ResultDSet.AddRecord((long) iteration, NumOfCapsDOut1, NumOfCapsDOut2,
                         Out1Gauge, Out1Turns, Out1Windings, Rd1In, Rd2In, Rd1Out, Rd2Out, L1In,
                         In1PowLoss, L1Out, Out1PowLoss, L1OutName, L1InName, L1InPrice,
                         L1OutPrice, L1InWeight, L1OutWeight, L1InVolume, L1OutVolume,
                         CapD1InIndex, CapD2InIndex, CapD1OutIndex, CapD2OutIndex);
               }
               m_CapacitorSet.m_strFilter = "";
               m_CapacitorSet.Requery();
               m_EMIInductorSet.m_strFilter = "";
               m_EMIInductorSet.Requery();
               m_InductSet.m_strFilter = "";
               m_InductSet.Requery();
               m_HeatSinkSet.m_strFilter = "";
               m_HeatSinkSet.Requery();

          }
}
m_CountDown.DestroyWindow();
char testbuffer[200];
sprintf( testbuffer,"Good %i,\nEMI Inductor %i,\nHeatSink %i\nCell Inductort %i\nBus Caps %i\nPrice
          Max %i\nWeight Max %i\nVolume Max %i", Good, Lneeded, HSneeded, LCellneeded,
          BusCapneeded, Priceneeded, Weightneeded, Volumeneeded);
MessageBox(NULL,testbuffer,"Needed", MB_OK );
```

```
}

/*///////////////////////////////////////////////////////////////////////
CResultSet::DeleteAll()

DeleteAll deletes all the records in the Results database.
*///////////////////////////////////////////////////////////////////////
void CResultSet::DeleteAll()
{
        Requery();
        if (!IsBOF())
                MoveFirst();
        while (!IsEOF())
        {
                Delete();
                MoveNext();
        }
        return;
}

/*///////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetRandomValues(int& CellNum, double& Freq, double& Inductor,
        int& Mode, double& RippleRatio)

GetRandomValues Determines the number of cells, switching frequency, and
inductance to use for a particular inductance.

Parameters:
        CellNum : The number of cells
        Freq : The Converter switching frequency
        Inductor : The total inductance of the power stage
        Mode : 1=Continuous Mode 2=Discontinuous Mode 3=Combo determined by voltages
        RippleRatio : The ripple ratio for one cell under nominal voltages
*///////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetRandomValues(int& CellNum, double& Freq, double& Inductor,
        int& Mode, double& RippleRatio)
{

        int             Nrange, //The range of number of cells
                        PossibleLs, //The range of ripple ratios
                        PossibleFreqs; //The range of frequencies
        double          i, j, //counting indexes
                        Lmax, Lmin, //The Min. and max. inductances
                        Ripmax, Ripmin, //The Min. and max. ripple ratios
                        Vin, Vout, //Input and output voltages
                        minDCM = 99999, maxCM = 0; //Values that find the min and max values of
                                //Vout/Vin *(Vin*Vout)
srand( (unsigned)time( NULL ) );

Nrange = m_ConfigurationSet.m_Nmax - m_ConfigurationSet.m_Nmin + 1;
CellNum = (rand()%Nrange) + m_ConfigurationSet.m_Nmin;

if (m_ConfigurationSet.m_freqstep != 0)
{
        PossibleFreqs = (int)((m_ConfigurationSet.m_freqmax- m_ConfigurationSet.m_freqmin) /
                m_ConfigurationSet.m_freqstep + 1);
        Freq = (rand()%PossibleFreqs)*m_ConfigurationSet.m_freqstep +
                m_ConfigurationSet.m_freqmin;
}
else Freq = m_ConfigurationSet.m_freqmin;

//finding the smallest values of L that guarentees CM operation
//and the largest value that guarentees DCM operation

for (i = m_ConfigurationSet.m_Vinmin; i <= m_ConfigurationSet.m_Vinmax; i = i +
        (m_ConfigurationSet.m_Vinmax - m_ConfigurationSet.m_Vinmin)/10)
{
        for (j = m_ConfigurationSet.m_Voutmin; j <= m_ConfigurationSet.m_Voutmax; j = j +
                (m_ConfigurationSet.m_Vinmax - m_ConfigurationSet.m_Vinmin)/10)
        {
                if ((i-j)*j/i < minDCM)
                        minDCM = (i-j)*j/i;
                if ((i-j)*j/i > maxCM)
                        maxCM = (i-j)*j/i;
        }
}

Lmax = minDCM/(2*m_ConfigurationSet.m_Iout*Freq);
Lmin = maxCM/(2*m_ConfigurationSet.m_Iout*Freq);

Vin = 42;
Vout = 14.3;
if (m_ConfigurationSet.m_CM)
        Ripmax = m_ConfigurationSet.m_Lmax;
else Ripmax = sqrt(minDCM*Vout*(Vin-Vout)/Vin)/(Lmax*Freq);

if (m_ConfigurationSet.m_DCM)
        Ripmin = m_ConfigurationSet.m_Lmin;
else Ripmin = Vout*(Vin-Vout)/(Vin*Lmin*Freq);

if (m_ConfigurationSet.m_Lstep == 0)
        RippleRatio = m_ConfigurationSet.m_Lmin;
else
{
        if (m_Configuration2Set.m_LinearL)
        {
                PossibleLs = (int)((Ripmax - Ripmin)/m_ConfigurationSet.m_Lstep + 1);
                RippleRatio = (rand()%PossibleLs)*m_ConfigurationSet.m_Lstep + Ripmin;
        }
        else
        {
                PossibleLs = (int)((log(Ripmax+1) - log(Ripmin+ 1)) /
                      log(m_ConfigurationSet.m_Lstep+1));
                RippleRatio = exp((rand()%PossibleLs)*log(m_ConfigurationSet.m_Lstep+1) +
                      log(Ripmin+1))-1;
        }
}

if (RippleRatio <= 1)
        Inductor = Vout*(Vin-Vout)/(2*Vin*RippleRatio*Freq*m_ConfigurationSet.m_Iout/CellNum);
else Inductor = Vout*(Vin-Vout) / (2*m_ConfigurationSet.m_Iout / CellNum* Vin* Freq* RippleRatio*
        RippleRatio);
```

```
//this puts the answer back to total inductance i.e. the cell inductors in parallel
Inductor = Inductor/CellNum;
if (Inductor < minDCM/(2*m_ConfigurationSet.m_Iout*Freq))
        Mode = DCM;
else if (Inductor > maxCM/(2*m_ConfigurationSet.m_Iout*Freq))
        Mode = CM;
else Mode = OTHER;

return;
}

/*//////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::GetInductorCurrentInfo(int CellNum, double Freq,
        double Inductor, double Vout, double Vin, double& CurrentPeak,
        double& Irms, double& S1rms, double& S2rms, double waveform[ ],
        double waveformS2[ ], double& Iinitial, int Mode)

GetInductorCurrentInfo determines the shape of the current at the input and
output of each cell of the converter.

Parameters:
        CellNum : The number of cells
        Freq : The Converter switching frequency
        Inductor : The total inductance of the power stage
        Vout : The output Voltage
        Vin : The input voltage
        CurrentPeak : The maximum current in the converter
        Irms : The rms current in the inductor
        S1rms : The rms current in switch 1
        S2rms : The rms current in switch 2
        waveform[ ] : A 1200 point vector of inductor current
        waveformS2[ ] : A 1200 point vector of the current in switch 2
        Iinitial : The current that is flowing in the inductor when the FET turns on
        Mode : 1=Continuous Mode 2=Discontinuous Mode 3=Combo determined by voltages

Output
True is function obtained valid results
*///////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::GetInductorCurrentInfo(int CellNum, double Freq,
        double Inductor, double Vout, double Vin, double& CurrentPeak,
        double& Irms, double& S1rms, double& S2rms, double waveform[],
        double waveformS2[], double& Iinitial, int Mode)
{
                double  iave, //The average current in each cell
                        L, //The Inductor in each cell
                        dutyc, //the duty cycle of the converter
                        dt, //The time step
                        total; //used as a temperary placeholder
                int     i, //Counting index
                        on; //The number of time steps the MOSFET is on

for(i=0;i<1200;i++)
{
        waveform[i] = 0;
        waveformS2[i] = 0;
```

```
}
iave = m_ConfigurationSet.m_Iout/CellNum;
L = Inductor*CellNum;
dt = 1/(Freq*1200);
if (Mode == OTHER)
{
        dutyc = Vout/Vin;
        Iinitial = iave - dutyc*(Vin-Vout)/(L*2*Freq);
        if (Iinitial < 0)
                Mode = DCM;
        else Mode = CM;
}
if (Mode == DCM)
{
        total = (iave*2*L*Vout*Freq)/(Vin*(Vin-Vout));
        dutyc = sqrt(total);
        Iinitial = 0;
}
else
{
        dutyc = Vout/Vin;
        Iinitial = iave - dutyc*(Vin-Vout)/(L*2*Freq);
}
on = (int) floor(dutyc*1200);
waveform[0] = Iinitial+(Vin-Vout)*dt/L;
S1rms = waveform[0]*waveform[0];
for(i=1; i<(on); i++)
{
        waveform[i] = waveform[i-1] + (Vin-Vout)*dt/L;
        S1rms = waveform[i]*waveform[i] + S1rms;
}
i = on;
S1rms = sqrt(S1rms/1200);
CurrentPeak = dutyc*(Vin-Vout)/(L*Freq) + Iinitial;
if (i>1200)
        return FALSE;
if (i == 1200)
        i = 0;
waveform[i] = CurrentPeak - Vout*dt/L*(on+1-dutyc*1200);
waveformS2[i] = waveform[i];
S2rms =waveform[i]*waveform[i];
i++;
do
{
        waveform[i] = waveform[i-1] - Vout*dt/L;
        waveformS2[i] = waveform[i];
        S2rms = S2rms + waveform[i]*waveform[i];
        i++;
}
while ((waveform[i-1] > Iinitial) && (i<1200));

S2rms = sqrt(S2rms/1200);

if (i != 1200)
{
        waveform[i-1] = 0;
        waveformS2[i-1] = 0;
}
Irms = 0;
for (i=0;i<1200;i++)
        Irms = waveform[i]*waveform[i] + Irms;

Irms = sqrt(Irms/1200);
return TRUE;
}

/*/////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::PowerStage(double Ipk,double Freq, double IrmsS1, double IrmsS2,
        double L, double IrmsL, double waveform[ ], double waveformS2[ ], CString& FETName,
        double& FETPcon, double& FETPrice, double& FETPsw, double& FETPtot,
        int& turns, int& Gauge, double& Bpksave, double& Jwiresave, double& Pcoresave,
        double& Pwindsave, double& deltaTsave, double& S2Pcond, double& S2Psw,
        double& S2Price, CString& S2Name, int& Picked, double& DeltaT, double& FETTemp,
        double& S2Temp, double Iinitial, int& Mode, int CellNum, long& IndNum,
        int& SinkNum, int& Cores, int& SinkNum1, double& DeltaT1, int& Length,
        int& HSneeded, int& LCellneeded)

PowerStage Designs the power stage of the converter. This consists of both switches,
the converter inductor, and the heatsink.

Parameters:
        Ipk : The maximum current in the converter
        Freq : The Converter switching frequency
        IrmsS1 : The rms current in switch 1
        IrmsS2 : The rms current in switch 2
        L : The total inductance of the power stage
        IrmsL : The rms current in the inductor
        waveform[ ] : A 1200 point vector of inductor current
        waveformS2[ ] : A 1200 point vector of the current in switch 2
        FETName : The Name of the MOSFET used
        FETPcon : The conduction power losses in the MOSFET
        FETPrice : The Price of the MOSFET
        FETPsw : The switching power losses in the MOSFET
        FETPtot : The total power loss in the FET
        turns : Number of turns in the converter inductor
        Gauge : The gauge of wire used in the converter inductor
        Bpksave : The peak magnetic field in the converter inductor
        Jwiresave : The current density in the converter inductor
        Pcoresave : The core power loss in the converter inductor
        Pwindsave : The winding power loss in the converter inductor
        deltaTsave : The max. temperature rise in the converter inductor
        S2Pcond : The conduction power losses in switch 2
        S2Psw : The switching power losses in switch 2
        S2Price : The price of switch 2
        S2Name : The name of switch 2
        Picked : whether switch two is 1=MOSFET 2=Schottky 3=Diode
        DeltaT : The temperature rise of the first heatsink
        FETTemp : The temperature rise of the MOSFET
        S2Temp : The temperature rise of switch 2
        Iinitial : The current that is flowing in the inductor when the FET turns on
        Mode : 1=Continuous Mode 2=Discontinuous Mode 3=Combo determined by voltages
        CellNum         : The number of cells
        IndNum : The index of the converter inductor
        SinkNum : The index of the first heatsink
        Cores : The number of converter inductors in parallel
        SinkNum1 : The index if any of the second heatsink
        DeltaT1 : The temperature rise if any of the second heatsink
        Length : The Length of extrusion needed
        HSneeded : Number of fail designs due to heatsink
        LCellneeded : Number of fail designs due to converter inductor


Output
True is function obtained valid results
*//////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::PowerStage(double Ipk,double Freq, double IrmsS1, double IrmsS2,
        double L, double IrmsL, double waveform[], double waveformS2[], CString& FETName,
        double& FETPcon, double& FETPrice, double& FETPsw, double& FETPtot,
        int& turns, int& Gauge, double& Bpksave, double& Jwiresave, double& Pcoresave,
        double& Pwindsave, double& deltaTsave, double& S2Pcond, double& S2Psw,
        double& S2Price, CString& S2Name, int& Picked, double& DeltaT, double& FETTemp,
        double& S2Temp, double Iinitial, int& Mode, int CellNum, long& IndNum,
        int& SinkNum, int& Cores, int& SinkNum1, double& DeltaT1, int& Length,
        int& HSneeded, int& LCellneeded)
{
        int     FETnum, //The index of the MOSFET in use
                AvailFET ,//The number of available MOSFETs
                AvailSchottky, //The number of available Schottky diodes
                AvailDiode, //The number of available PiN diodes
                total, //Variable used for counting
                Chosen, //Variable used for determine switch 2
                S2Schottkynum, //The index for the Schottky diode in use
                S2Diodenum, //The index for the PiN diode in use
                Packages; //Packages stores what kind of packages are used.
                        //1st bit is 1 if S1 is T0-220          2nd bit is 1 if S2 is T0-220
                        //3st bit is 1 if S1 is T0-247          4nd bit is 1 if S2 is T0-247
        bool    ValidInductor, //True if the inductor was designed successively
                Valid = TRUE, //True if the power stage was designed successively
                SinkValid; //True if the heatsink was designed successively
        double  DeltaTAllowed, //Max. Temperature - Ambient Temperature
                FETThermalResistance, //The MOSFET's junction to case thermal resistance
                S2ThermalResistance, //Switch 2's junction to case thermal resistance
                FETPcap, S2Pcap; //The power loss associated with the charging and
                        //discharging of the equavalent capacitance across the MOSFET or switch 2


Picked = 0;
S2Name = _T("");

//SECTION 1 ------>Pick and get values of Switch 1
//Pick one of the available FETs at random from the DB
FETnum = m_fETSet.PickFET(FETPrice, FETName, AvailFET,
        m_ConfigurationSet.m_MaxTemperature, m_ConfigurationSet.m_Temperature,
        m_ConfigurationSet.m_Iout/CellNum, Packages);

//Finds the power losses due to the FET
GetMaxPowLossF(FETnum, Freq, 1, CellNum, L, Mode, FETPcon, FETPcap, FETPsw,
                        FETThermalResistance);
```

```
if (LCellneeded > 200)
        LCellneeded = LCellneeded;

Cores = 1;
//SECTION 2 ------>designs the inductor
ValidInductor = m_InductSet.GetInductor(&m_WireSet, L*CellNum, Ipk, IrmsL, Freq,
        m_ConfigurationSet.m_Temperature,m_ConfigurationSet.m_MaxTemperature, waveform, turns,
        Gauge, Bpksave, Jwiresave, Pcoresave, Pwindsave, deltaTsave, IndNum, Cores);

//SECTION 3 ------>Pick and get values of Switch 2
total = 0;

if (m_ConfigurationSet.m_FETasS2)
        total = AvailFET;
else AvailFET = 0;

if (m_ConfigurationSet.m_SchottyDasS2)
{
        AvailSchottky = m_SchottkySet.GetAvail(m_ConfigurationSet.m_MaxTemperature,
                m_ConfigurationSet.m_Iout/CellNum);
        total = total+AvailSchottky;
}
else AvailSchottky = 0;

if (m_ConfigurationSet.m_DiodeasS2)
{
        AvailDiode = m_DiodeSet.GetAvail(m_ConfigurationSet.m_MaxTemperature,
                m_ConfigurationSet.m_Iout/CellNum);
        total = total+AvailDiode;
}
else AvailDiode = 0;

Chosen = (rand()%total)+1;


if (Chosen <= AvailFET)
{
        GetMaxPowLossF(FETnum, Freq, 2, CellNum, L, Mode, S2Pcond, S2Pcap, S2Psw,
                S2ThermalResistance);
        FETPsw = FETPsw + S2Pcap;
        S2Psw = S2Psw+FETPcap;
        Picked = FET;
        S2Name = FETName;
        if (Packages == 1)
                Packages = Packages+4;
        if (Packages == 2)
                Packages = Packages+8;
}
else if (Chosen <= AvailFET+AvailSchottky)
{
        S2Schottkynum = m_SchottkySet.PickSchottky(S2Price,S2Name,
              AvailSchottky,m_ConfigurationSet.m_MaxTemperature,
              m_ConfigurationSet.m_Iout/CellNum, Packages);
```

```
            GetMaxPowLossS(CellNum, Freq, L, S2Pcond, S2Pcap, S2ThermalResistance,
                  Mode,S2Schottkynum);
//Since the switch on losses are the same the powloss functions use there own
//Set data to calculate the final Psw losses
            FETPsw = FETPsw + S2Pcap;
            Picked = SCHOTTKY;
            S2Psw = 0;
}
else
{
            S2Diodenum = m_DiodeSet.PickDiode(S2Price,S2Name, AvailDiode,
                  m_ConfigurationSet.m_MaxTemperature, m_ConfigurationSet.m_Iout/CellNum, Packages);
            GetMaxPowLossD(CellNum, Freq, L, S2Pcond, S2Psw, S2Pcap, S2ThermalResistance,
                  Mode,S2Diodenum);
//Since the switch on losses are the same the powloss functions use there own
//Set data to calculate the final Psw losses
            FETPsw = FETPsw + S2Pcap;
            Picked = DIODE;
}

FETPtot = FETPcon+FETPsw;
FETTemp = FETPtot*(FETThermalResistance+m_Configuration2Set.m_Rcs);
S2Temp = (S2Pcond+S2Psw)*(S2ThermalResistance+m_Configuration2Set.m_Rcs);
DeltaTAllowed = m_ConfigurationSet.m_MaxTemperature - m_ConfigurationSet.m_Temperature;

SinkValid = m_HeatSinkSet.PickHeatSink(DeltaTAllowed, FETTemp, S2Temp, DeltaT, DeltaT1,
            SinkNum, SinkNum1, FETPcon+FETPsw, S2Pcond+S2Psw, Length,
            m_Configuration2Set.m_Rcs, Packages );

if (SinkValid == FALSE)
            HSneeded++;
if (ValidInductor == FALSE)
            LCellneeded++;

Valid = (Valid && ValidInductor) && SinkValid;
return Valid;
}

/*/////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::FilterStage(int CellNum, double Freq, double Inductor, int Mode,
      double& Qin, double& Qout, double& L2In, double& L2Out, double& L3In,
      double& L3Out, double& L1In,double& L1Out, double& In2PowLoss,
      double& Out2PowLoss,double& In1PowLoss, double& Out1PowLoss,
      double& Rin, double& Rout, double& Rd1In, double& Rd2In, double& Rd1Out,
      double& Rd2Out, int& Cap2InIndex, int& Cap2OutIndex, int& Cap1InIndex,
      int& Cap1OutIndex, int& CapD1InIndex, int& CapD2InIndex, int& CapD1OutIndex,
      int& CapD2OutIndex, int& L2InIndex, int& L2OutIndex, int& L1InIndex,
      int& L1OutIndex, int& In2Gauge, int& Out2Gauge, int& In1Gauge, int& Out1Gauge,
      int& In2Turns, int& Out2Turns, int& In1Turns, int& Out1Turns, int& In1Windings,
      int& In2Windings, int& Out1Windings, int& Out2Windings, int& Lneeded,
      int& BusCapneeded, int& NumOfCapsIn1, int& NumOfCapsIn2, int& NumOfCapsDIn1,
      int& NumOfCapsDIn2, int& NumOfCapsOut2, int& NumOfCapsOut1, int& NumOfCapsDOut1,
      int& NumOfCapsDOut2, int& FilterTypeIn, int& FilterTypeOut,
      double& InFilterPrice, double& OutFilterPrice, double& InFilterWeight,
      double& OutFilterWeight, double& InFilterVolume, double& OutFilterVolume)
```

FilterStage Designs the input and output filters of the converter.

Parameters:

CellNum : The number of cells
Freq : The Converter switching frequency
Inductor : The total inductance of the power stage
Mode : 1=Continuous Mode 2=Discontinuous Mode 3=Combo determined by voltages
Qin, Qout : The Q of the filter, for filter type 1
L2In, L2Out, L3In, L3Out ,L1In, L1Out,: The EMI inductances
In2PowLoss, Out2PowLoss, In1PowLoss, Out1PowLoss,: The power loss of the EMI inductors
Rin, Rout, Rd1In, Rd2In, Rd1Out, Rd2Out, : Resistor values
Cap2InIndex, Cap2OutIndex, Cap1InIndex, Cap1OutIndex, CapD1InIndex, CapD2InIndex, CapD1OutIndex, CapD2OutIndex : Indexes for capaciters
L2InIndex, L2OutIndex, L1InIndex, L1OutIndex : Indexes for Inductors
In2Gauge, Out2Gauge, In1Gauge, Out1Gauge : Wire gauges for the EMI inductors
In2Turns, Out2Turns, In1Turns, Out1Turns : The number of turns on the EMI inductors
In1Windings, In2Windings, Out1Windings, Out2Windings : The number of sets of turns on the EMI inductors
Lneeded : The number of times the Design EMI inductor function fails
BusCapneeded : The number of times proper capacitors could not be found
NumOfCapsIn1, NumOfCapsIn2, NumOfCapsDIn1, NumOfCapsDIn2, NumOfCapsOut2, NumOfCapsOut1, NumOfCapsDOut1, NumOfCapsDOut2, : Number of capacitors used in parallel
FilterTypeIn, FilterTypeOut : The type of filter used
InFilterPrice, OutFilterPrice, : The price of the filters used
InFilterWeight, OutFilterWeight : The weight of the filters used
InFilterVolume, OutFilterVolume : The volume of the filters used

Output
True is function obtained valid results

```
*////////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::FilterStage(int CellNum, double Freq, double Inductor, int Mode,
     double& Qin, double& Qout, double& L2In, double& L2Out, double& L3In,
     double& L3Out, double& L1In,double& L1Out, double& In2PowLoss,
     double& Out2PowLoss,double& In1PowLoss, double& Out1PowLoss,
     double& Rin, double& Rout, double& Rd1In, double& Rd2In, double& Rd1Out,
     double& Rd2Out, int& Cap2InIndex, int& Cap2OutIndex, int& Cap1InIndex,
     int& Cap1OutIndex, int& CapD1InIndex, int& CapD2InIndex, int& CapD1OutIndex,
     int& CapD2OutIndex, int& L2InIndex, int& L2OutIndex, int& L1InIndex,
     int& L1OutIndex, int& In2Gauge, int& Out2Gauge, int& In1Gauge, int& Out1Gauge,
     int& In2Turns, int& Out2Turns, int& In1Turns, int& Out1Turns, int& In1Windings,
     int& In2Windings, int& Out1Windings, int& Out2Windings, int& Lneeded,
     int& BusCapneeded, int& NumOfCapsIn1, int& NumOfCapsIn2, int& NumOfCapsDIn1,
     int& NumOfCapsDIn2, int& NumOfCapsOut2, int& NumOfCapsOut1, int& NumOfCapsDOut1,
     int& NumOfCapsDOut2, int& FilterTypeIn, int& FilterTypeOut,
     double& InFilterPrice, double& OutFilterPrice, double& InFilterWeight,
     double& OutFilterWeight, double& InFilterVolume, double& OutFilterVolume)
{
        double          WCIn[250], //A vector containing the magnitudes of the first 250
                                //harmonics for the converter input current
                        WCOut[250], //A vector containing the magnitudes of the first 250
                                //harmonics for the converter output current
                        Cap2In, Cap1In, Cap2Out, CapsDIn1, CapsDIn2, CapsDOut1, CapsDOut2,
                                Cap1Out, //Capacitor values
                        Rc1, Rc2, Rinloop,Rd1Inloop,Rd2Inloop, Routloop, Rd1Outloop,
                                Rd2Outloop, //Resistor values
                        Lc1, Lc2, Ld1, Ld2, L2Inloop, L3Inloop, L1Inloop, L2Outloop,
                                L3Outloop, L1Outloop, //Inductor values
                        Price, Weight, Volume, //Running totals for the filters
                        Qinloop, Qoutloop, //The Q of the filter
                        In2PowLossloop, In1PowLossloop, Out2PowLossloop, Out1PowLossloop,
                                //The Power lost in the EMI inductor
                        InFilterPriceloop, InFilterWeightloop, InFilterVolumeloop,
                                //attrubuted for the input filter
                        MinFilterCost, //basis of ranking filter designs
                        OutFilterPriceloop, OutFilterWeightloop, OutFilterVolumeloop,
                                //attrubuted for the input filter
                        FilterCost; //The cost of a filter design
        bool            Valid = TRUE, //True if both filters were designed
                        Validloop; //True is a given filter if design correctly for one loop
                                //of the inner control loop
        int             i, //Counting index
                        Cap2InIndexloop, Cap1InIndexloop, CapD1InIndexloop, CapD2InIndexloop,
                                Cap2OutIndexloop, Cap1OutIndexloop, CapD1OutIndexloop,
                                CapD2OutIndexloop, //Capacitor indexes
                        In2Gaugeloop, In1Gaugeloop, Out2Gaugeloop, Out1Gaugeloop, //Wire
                                //Gauge for the EMI inductors
                        In2Turnsloop, In1Turnsloop, Out2Turnsloop, Out1Turnsloop,//Number of
                                //turns for the EMI inductors
                        In2Windingsloop, In1Windingsloop, Out2Windingsloop,Out1Windingsloop,
                                //Number of sets of windings for the EMI inductors
                        L2InIndexloop, L1InIndexloop, L2OutIndexloop, L1OutIndexloop, //EMI
                                //Inductor indexes
                        NumOfCapsIn1loop, NumOfCapsIn2loop, NumOfCapsDIn1loop,
                                NumOfCapsDIn2loop, NumOfCapsOut1loop,
                                NumOfCapsOut2loop,NumOfCapsDOut1loop,
                                NumOfCapsDOut2loop,//Number of capacitors in parallel
                        FilterTypeInloop, FilterTypeOutloop;//The type of filter topology



GetWorstCaseHarmonics(CellNum, Inductor, Freq, WCIn, WCOut, Mode);
//input filer
srand( (unsigned)time( NULL ) );

MinFilterCost = 99999999;
Valid = FALSE;
for ( i = 0; i < m_Configuration2Set.m_ICLiterations; i++)
{
    InFilterPriceloop = InFilterWeightloop = InFilterVolumeloop = 0;

//picks the number of capaciters to use - from 1 to CapMax randomly
    NumOfCapsIn2loop = rand()%m_ConfigurationSet.m_CapMax+1;

//pics the capacitor randomly
    Cap2InIndexloop = m_CapacitorSet.PickCap(m_ConfigurationSet.m_Temperature,
      m_ConfigurationSet.m_Vinmax+10, Cap2In, Rc2, Lc2, Price, Weight, Volume, SWITCHING);

    InFilterPriceloop = Price*NumOfCapsIn2loop*CellNum;
    InFilterWeightloop = Weight*NumOfCapsIn2loop*CellNum;
    InFilterVolumeloop = Volume*NumOfCapsIn2loop*CellNum;
```

```
FilterTypeInloop = m_Configuration2Set.GetFilterType();

if (FilterTypeInloop == 1)
{
   //looks for the bus capacitor of value Cap2In*NumOfCapsIn1/4
   Cap1InIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
        m_ConfigurationSet.m_Vinmax+10, CellNum*Cap2In*NumOfCapsIn2loop/4, Rc1, Lc1,
        NumOfCapsIn1loop, m_ConfigurationSet.m_CapMax*2, Cap1Out, Price, Weight, Volume,
        BUS);

   InFilterPriceloop += Price*NumOfCapsIn1loop;
   InFilterWeightloop += Weight*NumOfCapsIn1loop;
   InFilterVolumeloop += Volume*NumOfCapsIn1loop;

   Validloop = DesignFilter1(Freq, CellNum, WCIn, Qinloop, L2Inloop, L3Inloop,
        Cap2In*CellNum*NumOfCapsIn2loop, Rinloop, Rc1/NumOfCapsIn1loop,
        Rc2/(CellNum*NumOfCapsIn2loop), Lc1/NumOfCapsIn1loop,
        Lc2/(CellNum*NumOfCapsIn2loop));
   if (Validloop)
   {
     L2InIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L2Inloop,
        m_ConfigurationSet.m_Iout*m_ConfigurationSet.m_Voutmax/m_ConfigurationSet.m_Vinmi
        n, m_ConfigurationSet.m_MaxTemperature - m_ConfigurationSet.m_Temperature ,
        m_ConfigurationSet.m_InFilterPmax, m_ConfigurationSet.m_InFilterSimple, In2Gaugeloop,
        In2Turnsloop, In2Windingsloop, In2PowLossloop, Price, Weight, Volume);

     InFilterPriceloop += Price;
     InFilterWeightloop += Weight;
     InFilterVolumeloop += Volume;

   }
   CapD1InIndexloop = CapD2InIndexloop = NumOfCapsDIn1loop = NumOfCapsDIn2loop = -1;
   L1InIndexloop = In1Gaugeloop = -1;
   In1Turnsloop = In1Windingsloop = -1;
   Rd1Inloop = Rd2Inloop = L1Inloop = In1PowLossloop = -1;
    if (L2InIndexloop == -1)
    {
     Validloop = FALSE;
     Lneeded++;
    }
    if (Cap1InIndexloop == -1)
     Validloop = FALSE;
    if (Cap2InIndexloop == -1)
    {
      Validloop = FALSE;
      BusCapneeded++;
    }
} //if (FilterTypeInloop == 1)
if (FilterTypeInloop == 2)
{
   //looks for another capacitor of value Cap2In*NumOfCapsIn1/4
   Cap1InIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
        m_ConfigurationSet.m_Vinmax+10, CellNum*Cap2In*NumOfCapsIn2loop/4, Rc1, Lc1,
        NumOfCapsIn1loop, m_ConfigurationSet.m_CapMax*2, Cap1In, Price, Weight, Volume,
        SWITCHING);
```

```
InFilterPriceloop += Price*NumOfCapsIn1loop;
InFilterWeightloop += Weight*NumOfCapsIn1loop;
InFilterVolumeloop += Volume*NumOfCapsIn1loop;

CapD1InIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
    m_ConfigurationSet.m_Vinmax+10, Cap1In*NumOfCapsIn1loop*10, Rd1Inloop, Ld1,
    NumOfCapsDIn1loop, m_ConfigurationSet.m_CapMax*4,CapsDIn1, Price, Weight,
    Volume, DAMPING);

InFilterPriceloop += Price*NumOfCapsDIn1loop;
InFilterWeightloop += Weight*NumOfCapsDIn1loop;
InFilterVolumeloop += Volume*NumOfCapsDIn1loop;

CapD2InIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
    m_ConfigurationSet.m_Vinmax+10, CellNum*Cap2In*NumOfCapsIn2loop*10, Rd2Inloop,
    Ld2, NumOfCapsDIn2loop, m_ConfigurationSet.m_CapMax*4,CapsDIn2, Price, Weight,
    Volume, DAMPING);

InFilterPriceloop += Price*NumOfCapsDIn2loop;
InFilterWeightloop += Weight*NumOfCapsDIn2loop;
InFilterVolumeloop += Volume*NumOfCapsDIn2loop;

Rd1Inloop = Rd1Inloop/NumOfCapsDIn1loop;
Rd2Inloop = Rd2Inloop/NumOfCapsDIn2loop;
if ((Cap1InIndexloop != -1) && (CapD1InIndexloop != -1) && (CapD2InIndexloop != -1))
{
    Validloop = DesignFilter2(Cap1In*NumOfCapsIn1loop, Rc1/NumOfCapsIn1loop,
        Lc1/NumOfCapsIn1loop, Cap2In*CellNum*NumOfCapsIn2loop,
        Rc2/(CellNum*NumOfCapsIn2loop), Lc2/(CellNum*NumOfCapsIn2loop),
        CapsDIn1*NumOfCapsDIn1loop, Rd1Inloop, Ld1/NumOfCapsDIn1loop,
        CapsDIn2*NumOfCapsDIn2loop, Rd2Inloop, Ld2/NumOfCapsDIn2loop, L1Inloop,
        L2Inloop, Freq, CellNum, WCIn);
}
else Validloop = FALSE;

if (Validloop)
{
L1InIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L1Inloop,
    m_ConfigurationSet.m_Iout*m_ConfigurationSet.m_Voutmax/m_ConfigurationSet.m_Vinmi
    n, m_ConfigurationSet.m_MaxTemperature - m_ConfigurationSet.m_Temperature ,
    m_ConfigurationSet.m_InFilterPmax, m_ConfigurationSet.m_InFilterSimple, In1Gaugeloop,
    In1Turnsloop, In1Windingsloop, In1PowLossloop, Price, Weight, Volume);

InFilterPriceloop += Price;
InFilterWeightloop += Weight;
InFilterVolumeloop += Volume;

L2InIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L2Inloop,
    m_ConfigurationSet.m_Iout*m_ConfigurationSet.m_Voutmax/m_ConfigurationSet.m_Vinmi
    n, m_ConfigurationSet.m_MaxTemperature - m_ConfigurationSet.m_Temperature ,
    m_ConfigurationSet.m_InFilterPmax, m_ConfigurationSet.m_InFilterSimple, In2Gaugeloop,
    In2Turnsloop, In2Windingsloop, In2PowLossloop, Price, Weight, Volume);

InFilterPriceloop += Price;
InFilterWeightloop += Weight;
```

```
    InFilterVolumeloop += Volume;

    }
    Qinloop = Rinloop = L3Out = -1;
    if ((L1InIndexloop == -1) || (L2InIndexloop == -1))
    Validloop = FALSE;
}//if (FilterTypeInloop == 2)
if (FilterTypeInloop == 3)
{
    Validloop = DesignFilter3(Cap2In*CellNum*NumOfCapsIn2loop,
         Rc2/(CellNum*NumOfCapsIn2loop), Lc2/(CellNum*NumOfCapsIn2loop), L2Inloop, Freq,
         CellNum, WCIn);

    if (Validloop)
    {
         L2InIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L2Inloop,
              m_ConfigurationSet.m_Iout*m_ConfigurationSet.m_Voutmax/m_ConfigurationSet.m_V
              inmin, m_ConfigurationSet.m_MaxTemperature - m_ConfigurationSet.m_Temperature ,
              m_ConfigurationSet.m_InFilterPmax, m_ConfigurationSet.m_InFilterSimple,
              In2Gaugeloop, In2Turnsloop, In2Windingsloop, In2PowLossloop, Price, Weight,
              Volume);

         InFilterPriceloop += Price;
         InFilterWeightloop += Weight;
         InFilterVolumeloop += Volume;
    }
    Qinloop = L3Inloop = Rinloop = Rd1Inloop = Rd2Inloop = L1Inloop = In1PowLossloop = -1;
    Cap1InIndexloop = NumOfCapsIn1loop = CapD1InIndexloop = CapD2InIndexloop = -1;
    NumOfCapsDIn1loop = NumOfCapsDIn2loop = -1;
    L1InIndexloop = In1Gaugeloop = In1Turnsloop = In1Windingsloop = Cap1OutIndex = -1;
    if (L2InIndexloop == -1)
              Validloop = FALSE;
}//if (FilterTypeInloop == 3)

FilterCost = InFilterPriceloop* m_Configuration2Set.m_Price+InFilterWeightloop*
         m_Configuration2Set.m_Weight + InFilterVolumeloop* m_Configuration2Set.m_Volume;
if ((FilterCost < MinFilterCost) && Validloop)
{
    MinFilterCost = FilterCost;
    Qin = Qinloop;
    L2In = L2Inloop;
    L3In = L3Inloop;
    In2PowLoss = In2PowLossloop;
    Rin = Rinloop;
    Cap2InIndex = Cap2InIndexloop;
    Cap1InIndex = Cap1InIndexloop;
    In2Gauge = In2Gaugeloop;
    In2Turns = In2Turnsloop;
    In2Windings = In2Windingsloop;
    L2InIndex = L2InIndexloop;
    NumOfCapsIn1 = NumOfCapsIn1loop;
    NumOfCapsIn2 = NumOfCapsIn2loop;
    CapD1InIndex = CapD1InIndexloop;
    CapD2InIndex = CapD2InIndexloop;
    NumOfCapsDIn1 = NumOfCapsDIn1loop;
    NumOfCapsDIn2 = NumOfCapsDIn2loop;
```

```
            L1InIndex = L1InIndexloop;
            In1Gauge = In1Gaugeloop;
            In1Turns = In1Turnsloop;
            In1Windings = In1Windingsloop;
            Rd1In = Rd1Inloop;
            Rd2In = Rd2Inloop;
            L1In = L1Inloop;
            In1PowLoss = In1PowLossloop;
            FilterTypeIn = FilterTypeInloop;
            InFilterPrice = InFilterPriceloop;
            InFilterWeight = InFilterWeightloop;
            InFilterVolume = InFilterVolumeloop;
            Valid = TRUE;
        }

  }
if (!Valid)
      return FALSE;

MinFilterCost = 99999;
for ( i = 0; i < m_Configuration2Set.m_ICLiterations; i++)
{
//output filter
OutFilterPriceloop = OutFilterWeightloop = OutFilterVolumeloop = 0;

NumOfCapsOut2loop = rand()%m_ConfigurationSet.m_CapMax+1;

Cap2OutIndexloop = m_CapacitorSet.PickCap(m_ConfigurationSet.m_Temperature,
        m_ConfigurationSet.m_Voutmax+5, Cap2Out, Rc2, Lc2, Price, Weight, Volume, SWITCHING);

OutFilterPriceloop = Price*NumOfCapsOut2loop*CellNum;
OutFilterWeightloop = Weight*NumOfCapsOut2loop*CellNum;
OutFilterVolumeloop = Volume*NumOfCapsOut2loop*CellNum;

FilterTypeOutloop = m_Configuration2Set.GetFilterType();

if (FilterTypeOutloop == 1)
{
      Cap1OutIndex = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
              m_ConfigurationSet.m_Voutmax+5, CellNum*Cap2Out*NumOfCapsOut2loop/4, Rc1, Lc1,
              NumOfCapsOut1loop, m_ConfigurationSet.m_CapMax*2, Cap1Out, Price, Weight, Volume,
              BUS);

      OutFilterPriceloop += Price*NumOfCapsOut1loop;
      OutFilterWeightloop += Weight*NumOfCapsOut1loop;
      OutFilterVolumeloop += Volume*NumOfCapsOut1loop;

      Validloop = DesignFilter1(Freq, CellNum, WCOut, Qoutloop, L2Outloop,
              L3Outloop,Cap2Out*NumOfCapsOut2loop*CellNum, Routloop, Rc1/NumOfCapsOut1loop,
              Rc2/(CellNum*NumOfCapsOut2loop), Lc1/NumOfCapsOut1loop,
              Lc2/(CellNum*NumOfCapsOut2loop));

      if (Validloop)
      {
         L2OutIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L2Outloop,
              m_ConfigurationSet.m_Iout, m_ConfigurationSet.m_MaxTemperature -
```

```
                m_ConfigurationSet.m_Temperature , m_ConfigurationSet.m_OutFilterPmax,
                m_ConfigurationSet.m_OutFilterSimple, Out2Gaugeloop, Out2Turnsloop,
                Out2Windingsloop, Out2PowLossloop, Price, Weight, Volume);

            OutFilterPriceloop += Price;
            OutFilterWeightloop += Weight;
            OutFilterVolumeloop += Volume;

        }
        Cap1OutIndexloop = CapD1OutIndexloop = NumOfCapsDOut1loop = -1;
        CapD2OutIndexloop = NumOfCapsDOut2loop =  L1OutIndexloop = Out1Gaugeloop = -1;
        Out1Turnsloop = -1;
        Out1Windingsloop = -1;
        Rd1Outloop = Rd2Outloop = L1Outloop = Out1PowLossloop = -1;
        if (L2OutIndexloop == -1)
        {
            Validloop = FALSE;
            Lneeded++;
        }
        if (Cap1OutIndex == -1)
            Validloop = FALSE;
        if (Cap2OutIndexloop == -1)
        {
           Validloop = FALSE;
           BusCapneeded++;
        }
}

if (FilterTypeOutloop == 2)
{
//looks for another capacitor of value Cap2In*NumOfCapsIn1/4
        Cap1OutIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
                m_ConfigurationSet.m_Voutmax+5, CellNum*Cap2Out*NumOfCapsOut2loop/4, Rc1, Lc1,
                NumOfCapsOut1loop, m_ConfigurationSet.m_CapMax*2, Cap1Out, Price, Weight, Volume,
                SWITCHING);

        OutFilterPriceloop += Price*NumOfCapsOut1loop;
        OutFilterWeightloop += Weight*NumOfCapsOut1loop;
        OutFilterVolumeloop += Volume*NumOfCapsOut1loop;

        CapD1OutIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
                m_ConfigurationSet.m_Voutmax+5, Cap1Out*NumOfCapsOut1loop*10, Rd1Outloop, Ld1,
                NumOfCapsDOut1loop, m_ConfigurationSet.m_CapMax*4,CapsDOut1, Price, Weight,
                Volume,DAMPING);

        OutFilterPriceloop += Price*NumOfCapsDOut1loop;
        OutFilterWeightloop += Weight*NumOfCapsDOut1loop;
        OutFilterVolumeloop += Volume*NumOfCapsDOut1loop;

        CapD2OutIndexloop = m_CapacitorSet.PickSetCap(m_ConfigurationSet.m_Temperature,
                m_ConfigurationSet.m_Voutmax+5, CellNum*Cap2Out*NumOfCapsOut2loop*10,
                Rd2Outloop, Ld2, NumOfCapsDOut2loop, m_ConfigurationSet.m_CapMax*4,CapsDOut2,
                Price, Weight, Volume,DAMPING);

        OutFilterPriceloop += Price*NumOfCapsDOut2loop;
        OutFilterWeightloop += Weight*NumOfCapsDOut2loop;
```

```
OutFilterVolumeloop += Volume*NumOfCapsDOut2loop;

Rd1Outloop = Rd1Outloop/NumOfCapsDOut1loop;
Rd2Outloop = Rd2Outloop/NumOfCapsDOut2loop;
if ((Cap1OutIndexloop != -1) && (CapD1OutIndexloop != -1) && (CapD2OutIndexloop != -1))
{
   Validloop = DesignFilter2(Cap1Out*NumOfCapsOut1loop, Rc1/NumOfCapsOut1loop,
        Lc1/NumOfCapsOut1loop, Cap2Out*CellNum*NumOfCapsOut2loop,
        Rc2/(CellNum*NumOfCapsOut2loop), Lc2/(CellNum*NumOfCapsOut2loop),
        CapsDOut1*NumOfCapsDOut1loop, Rd1Outloop, Ld1/NumOfCapsDOut1loop,
        CapsDOut2*NumOfCapsDOut2loop, Rd2Outloop, Ld2/NumOfCapsDOut2loop, L1Outloop,
        L2Outloop, Freq, CellNum, WCOut);
}
else Validloop = FALSE;

if (Validloop)
{
   L1OutIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L1Outloop,
        m_ConfigurationSet.m_Iout, m_ConfigurationSet.m_MaxTemperature -
        m_ConfigurationSet.m_Temperature , m_ConfigurationSet.m_OutFilterPmax,
        m_ConfigurationSet.m_OutFilterSimple, Out1Gaugeloop, Out1Turnsloop,
        Out1Windingsloop, Out1PowLossloop, Price, Weight, Volume);

   OutFilterPriceloop += Price;
   OutFilterWeightloop += Weight;
   OutFilterVolumeloop += Volume;

   L2OutIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L2Outloop,
        m_ConfigurationSet.m_Iout, m_ConfigurationSet.m_MaxTemperature -
        m_ConfigurationSet.m_Temperature , m_ConfigurationSet.m_OutFilterPmax,
        m_ConfigurationSet.m_OutFilterSimple, Out2Gaugeloop, Out2Turnsloop,
        Out2Windingsloop, Out2PowLossloop, Price, Weight, Volume);

   OutFilterPriceloop += Price;
   OutFilterWeightloop += Weight;
   OutFilterVolumeloop += Volume;

}
Qoutloop = Routloop = L3Outloop = -1;
if ((L1OutIndexloop == -1) || (L2OutIndexloop == -1))
   Validloop = FALSE;
}
if (FilterTypeOutloop == 3)
{
   Validloop = DesignFilter3(Cap2Out*CellNum*NumOfCapsOut2loop,
        Rc2/(CellNum*NumOfCapsOut2loop), Lc2/(CellNum*NumOfCapsOut2loop), L2Outloop,
        Freq, CellNum, WCOut);

   if (Validloop)
   {
        L2OutIndexloop = m_EMIInductorSet.DesignEMIInductor(&m_WireSet, L2Outloop,
             m_ConfigurationSet.m_Iout, m_ConfigurationSet.m_MaxTemperature -
             m_ConfigurationSet.m_Temperature , m_ConfigurationSet.m_OutFilterPmax,
             m_ConfigurationSet.m_OutFilterSimple, Out2Gaugeloop, Out2Turnsloop,
             Out2Windingsloop, Out2PowLossloop, Price, Weight, Volume);
```

```
            OutFilterPriceloop += Price;
            OutFilterWeightloop += Weight;
            OutFilterVolumeloop += Volume;
        }
        Qoutloop = L3Outloop = Routloop = Rd1Outloop = Rd2Outloop = L1Outloop = -1;
        Out1PowLossloop = -1;
        Cap1OutIndex = NumOfCapsOut1loop = CapD1OutIndexloop = NumOfCapsDOut1loop = -1;
        CapD2OutIndexloop = NumOfCapsDOut2loop = -1;
        L1OutIndexloop=Cap1OutIndexloop =Out1Gaugeloop=Out1Turnsloop=Out1Windingsloop = -1;
        if (L2OutIndexloop == -1)
                Validloop = FALSE;
    }

    FilterCost = OutFilterPriceloop* m_Configuration2Set.m_Price+ OutFilterWeightloop*
        m_Configuration2Set.m_Weight+OutFilterVolumeloop*m_Configuration2Set.m_Volume;
    if ((FilterCost < MinFilterCost) && Validloop)
    {
        MinFilterCost = FilterCost;
        Qout = Qoutloop;
        L2Out = L2Outloop;
        L3Out = L3Outloop;
        Out2PowLoss = Out2PowLossloop;
        Rout = Routloop;
        Cap2OutIndex = Cap2OutIndexloop;
        Cap1OutIndex = Cap1OutIndexloop;
        Out2Gauge = Out2Gaugeloop;
        Out2Turns = Out2Turnsloop;
        Out2Windings = Out2Windingsloop;
        L2OutIndex = L2OutIndexloop;
        NumOfCapsOut1 = NumOfCapsOut1loop;
        NumOfCapsOut2 = NumOfCapsOut2loop;
        CapD1OutIndex = CapD1OutIndexloop;
        CapD2OutIndex = CapD2OutIndexloop;
        NumOfCapsDOut1 = NumOfCapsDOut1loop;
        NumOfCapsDOut2 = NumOfCapsDOut2loop;
        L1OutIndex = L1OutIndexloop;
        Out1Gauge = Out1Gaugeloop;
        Out1Turns = Out1Turnsloop;
        Out1Windings = Out1Windings;
        Rd1Out = Rd1Outloop;
        Rd2Out = Rd2Outloop;
        L1Out = L1Outloop;
        Out1PowLoss = Out1PowLossloop;
        FilterTypeOut = FilterTypeOutloop;
        OutFilterPrice = OutFilterPriceloop;
        OutFilterWeight = OutFilterWeightloop;
        OutFilterVolume = OutFilterVolumeloop;
        Valid = TRUE;
    }
 }
 return Valid;
}
```

```
/*////////////////////////////////////////////////////////////////////
void CResultSet::AddRecord(double FETPrice, long IDnum, double Weight, double Price,
        double S2Price, double SinkWeight, double SinkVolume, double SinkPrice, double Cap1InPrice,
        double Cap1OutPrice, double Cap1InWeight, double Cap1OutWeight, double Cap1InVolume,
        double Cap1OutVolume, double L2InPrice, double L2OutPrice, double L2InWeight,
        double L2OutWeight, double L2InVolume, double L2OutVolume, double ControlPrice,
        double ControlWeight, double ControlVolume, double ControlArea, double IndVolume,
        double IndArea, double PCboardPrice, int CellNum, double CostPrice, double CostWeight,
        double CostVolume, int Length, double InFilterPrice, double OutFilterPrice,
        double InFilterWeight, double OutFilterWeight, double InFilterVolume, double OutFilterVolume)

AddRecord stores all necessary information into the database.


Parameters:
        FETPrice: The price of the MOSFET
        IDnum : The iteration number
        Weight, Price : The price and weight of the converter inductor per cell
        S2Price: The price of switch 2
        SinkWeight, SinkVolume, SinkPrice : The price, weight and volume of the heatsink
        Cap1InPrice, Cap1OutPrice, Cap1InWeight, Cap1OutWeight, Cap1InVolume,
                Cap1OutVolume : The price, weight, and volume of the filter capacitors
        L2InPrice, L2OutPrice, L2InWeight, L2OutWeight, L2InVolume, L2OutVolume :
                The price, weight, and volume of the EMI filter inductors
        ControlPrice, ControlWeight, ControlVolume, ControlArea : Properties of the
                control circuitry for the converter
        IndVolume, IndArea : The volume and board space of the converter inductor per cell
        PCboardPrice : The cost of the PC board in $/cm^2
        CellNum : The number of cells
        CostPrice, CostWeight, CostVolume : The coefficients used to calculate the converter cost
        Length : The length of extruded heatsink needed
        InFilterPrice, OutFilterPrice, InFilterWeight, OutFilterWeight, InFilterVolume,
                OutFilterVolume : Properties of the filters used

*////////////////////////////////////////////////////////////////////
void CResultSet::AddRecord(double FETPrice, long IDnum, double Weight, double Price,
        double S2Price, double SinkWeight, double SinkVolume, double SinkPrice, double Cap1InPrice,
        double Cap1OutPrice, double Cap1InWeight, double Cap1OutWeight, double Cap1InVolume,
        double Cap1OutVolume, double L2InPrice, double L2OutPrice, double L2InWeight,
        double L2OutWeight, double L2InVolume, double L2OutVolume, double ControlPrice,
        double ControlWeight, double ControlVolume, double ControlArea, double IndVolume,
        double IndArea, double PCboardPrice, int CellNum, double CostPrice, double CostWeight,
        double CostVolume, int Length, double InFilterPrice, double OutFilterPrice,
        double InFilterWeight, double OutFilterWeight, double InFilterVolume, double OutFilterVolume)
{
                double  TotalPrice, //The price of the converter
                        TotalWeight, //The weight of the converter
                        TotalVolume; //The volume of the converter

        AddNew();
        m_IDnum = IDnum;
        m_FETPrice = FETPrice;
        m_CoreWeight = Weight;
        m_CorePrice = Price;
        m_S2Price = S2Price;
        m_SinkWeight = SinkWeight;
```

```
	m_SinkVolume = SinkVolume;
	m_SinkPrice = SinkPrice;
	m_Cap1InPrice = Cap1InPrice;
	m_Cap1OutPrice = Cap1OutPrice;
	m_Cap1InWeight = Cap1InWeight;
	m_Cap1OutWeight = Cap1OutWeight;
	m_Cap1InVolume = Cap1InVolume;
	m_Cap1OutVolume = Cap1OutVolume;
	m_L2InPrice = L2InPrice;
	m_L2OutPrice = L2OutPrice;
	m_L2InWeight = L2InWeight;
	m_L2OutWeight = L2OutWeight;
	m_L2InVolume = L2InVolume;
	m_L2OutVolume = L2OutVolume;
	m_ControlPrice = ControlPrice;
	m_ControlWeight = ControlWeight;
	m_ControlVolume = ControlVolume;
	m_ControlArea = ControlArea;
	m_PCboardPrice = PCboardPrice;
	m_CellNum = CellNum;
	m_IndVolume = IndVolume;
	m_IndArea = IndArea;
	m_Length = Length;
	m_InFilterPrice = InFilterPrice;
	m_OutFilterPrice = OutFilterPrice;
	m_InFilterWeight = InFilterWeight;
	m_OutFilterWeight = OutFilterWeight;
	m_InFilterVolume = InFilterVolume;
	m_OutFilterVolume = OutFilterVolume;

	TotalPrice = m_PCboardPrice+InFilterPrice+OutFilterPrice+m_ControlPrice+
		CellNum*(m_FETPrice+m_CorePrice+m_S2Price+m_SinkPrice);

	TotalWeight = InFilterWeight+OutFilterWeight+m_ControlWeight+
		CellNum*(m_CoreWeight+m_SinkWeight);

	TotalVolume = InFilterVolume+OutFilterVolume+m_ControlVolume+
		CellNum*(m_IndVolume+m_SinkVolume);

	m_TotalCost = (TotalPrice*CostPrice+TotalWeight*CostWeight+
		TotalVolume*CostVolume);

	if ( CanUpdate() )
		Update();
}

/*/////////////////////////////////////////////////////////////////////
int CFETSet::PickFET(double& Price, CString& Name, int& NumUsing,
		double MaxTemp, double Temp, double AveCurrent, int& Package)
```

PickFET chooses a MOSFET randomly from the available records in the database.

Parameters:

	Price : The price of the MOSFET chosen
	Name : The name of the MOSFET chosen
	NumUsing : The number of available MOSFETs

```
          MaxTemp : The Max. Temperature allowed of the system
          Temp : The ambient temperature
          AveCurrent : The average current through the MOSFET
          Package : The case of the FET //1st bit is 1 if MOSFET is T0-220
                    //3rd bit is 1 if MOSFET is T0-247

Output
The index of the chosen MOSFET, -1 is no valid MOSFET was found
*//////////////////////////////////////////////////////////////////////////
int CFETSet::PickFET(double& Price, CString& Name, int& NumUsing,
                    double MaxTemp, double Temp, double AveCurrent, int& Package)
{
                    int       chosen, //Variable that counts MOSFETs
                              FETnum = 0; //The index of the MOSFET

NumUsing = 0;
srand( (unsigned)time( NULL ) );
//First count the number of available records
MoveFirst();
while (!IsEOF())
{
          if (m_Using && ((MaxTemp + 20) < m_MaxTemperature) &&
                    (AveCurrent*(m_Ra*Temp*Temp+m_Rb*Temp+m_Rc) < 10))
//20 is a threshold of the temp we are going to allow to be used
//10 is a crude method of calculating power dissipation to insure an
//inadequate device is not used
          NumUsing++;
          MoveNext();
}
//if no FET available then cause error
if (NumUsing == 0)
          return -1;
//randomly choose a FET
chosen = rand()%NumUsing;
MoveFirst();
FETnum = 0;
while (!IsEOF() && (chosen > -1))
{
          if (m_Using && ((MaxTemp + 20) < m_MaxTemperature)&&
                              (AveCurrent*(m_Ra*Temp*Temp+m_Rb*Temp+m_Rc) < 10))
                    chosen--;
          FETnum++;
          MoveNext();
          }
if (!IsBOF())
          MovePrev();
Price = atof(m_UnitPrice);
Name = m_FET;
Package = 0;
if (m_Package == "T0-220")
          Package = 1;
if (m_Package == "T0-247")
          Package = Package+4;
MoveFirst();
return FETnum;
}
```

```
/*////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetMaxPowLossF(int FETnum, double Freq, int Switch, int CellNum,
                douole L, int Mode, double& PowMax, double& FETPcapSave, double& FETPswSave,
                double& S2ThermalResistanceSave)

GetMaxPowLossF sweeps the range of possible input and output voltages in order to determine when a
MOSFET has the maximum power loss.


Parameters:
        FETNum : the index of the FET that is being used.
        Freq : the switching frequency of the converter
        Switch : determines if this is switch one or two.
        CellNum : Number of cells in the converter
        L : the value of the converter inductance
        Mode : CM or DCM
        Powmax : the maximum power loss
        FET2PcapSave : for the worst case power loss, the power lost by S2 due to
                the capacitance across the FET
        FETPswSave : for the worst case switching power loss, the power lost by the FET
                due to the switching losses.
        S2ThermalResistanceSave : the thermal resistance for the FET with worst case
                power loss

*////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetMaxPowLossF(int FETnum, double Freq, int Switch, int CellNum,
                double L, int Mode, double& PowMax, double& FETPcapSave, double& FETPswSave,
                double& S2ThermalResistanceSave)
{
                int     x, y; //Counting Indexes
                double  CurrentPeak, //The peak current in the converter
                        Irms, //The rms current in the Inductor
                        S1rms, //The rms current in the switch 1
                        S2rms, //The rms current in the switch 2
                        waveform[1200], //Vectors of the current in the inductor
                        waveformS2[1200], //Vectors of the current in switch 2
                        Iinitial, //The amount of current through the MOSFET when it turns on
                        Vin, //The input voltage
                        Vout, //The output voltage
                        FETPcond, //The conduction power losses in the MOSFET
                        FETPsw, //The switching power losses in the MOSFET
                        FETPcap, //The power associated with the capacitance across the MOSFET
                        S2ThermalResistance, //The thermal resistance of the MOSFET chosen
                        Loss, //The power loss of the MOSFET
                        MaxLoss; //The max Power loss of the MOSFET
                bool    Valid; //True if no errors occured

MaxLoss = 0.0;
for (x = 0; x < 11; x++)
{
        Vin = m_ConfigurationSet.m_Vinmin + (m_ConfigurationSet.m_Vinmax -
            m_ConfigurationSet.m_Vinmin)*x/10;
        for (y=0;y<11;y++)
        {
        Vout = m_ConfigurationSet.m_Voutmin + (m_ConfigurationSet.m_Voutmax -
            m_ConfigurationSet.m_Voutmin)*y/10;
        Valid = GetInductorCurrentInfo(CellNum, Freq, L, Vout, Vin, Irms, S1rms, S2rms, waveform,
            waveformS2, Iinitial, Mode);
        if (Valid == FALSE)
                continue;
        if (Switch == 1)
                Loss = m_fETSet.PowLoss(FETnum, Vin, CurrentPeak, m_ConfigurationSet.m_Idrive,
                  Freq, S1rms, FETPcond, FETPsw, FETPcap, S2ThermalResistance, Iinitial,
                  Switch, m_ConfigurationSet.m_Temperature);
        else
                Loss = m_fETSet.PowLoss(FETnum, Vin, Iinitial, m_ConfigurationSet.m_Idrive,
                  Freq, S2rms, FETPcond, FETPsw, FETPcap, S2ThermalResistance, CurrentPeak,
                  Switch, m_ConfigurationSet.m_Temperature);
         if (Loss > MaxLoss)
        {
                MaxLoss = Loss;
                PowMax = FETPcond;
                FETPcapSave = FETPcap;
                FETPswSave = FETPsw;
                S2ThermalResistanceSave = S2ThermalResistance;
         }
        }
  }
}
```

/*///
bool CInductSet::GetInductor(CWireSet* m_pWireData, double L, double Ipk,
                    double Irms, double freq, double Temp, double TempMax,
                    double waveform[1200], int& turns, int& Gauge, double& Bpksave,
                    double& Jwiresave, double& Pcoresave, double& Pwindsave,
                    double& deltaTsave, long& IndNum, int& Cores)

GetInductor chooses an appropriate core for each converter cells and insures a proper design of the inductor.

Parameters:

m_pWireData	The Wire database
L	The value of the converter inductance
Ipk	The peak current in the converter inductor
Irms	The rms current in the converter inductor
freq	The switching frequency of the converter
Temp	The ambient temperature
TempMax	The Max. allowable temperature of the converter inductor
waveform[1200]	A vector of the inductor current
turns	The number of turns on the core
Gauge	The gauge of wire to use to wind the core
Bpksave	The max. flux density in the core
Jwiresave	The current density of the wire in the inductor
Pcoresave	The power loss due to the core
Pwindsave	The power loss due to the windings
deltaTsave	The temperature rise of the core
IndNum	An Index to the core used
Cores	The number of cores used in parallel

Output
True if an Inductor was successively designed

```
*///////////////////////////////////////////////////////////////////////
bool CInductSet::GetInductor(CWireSet* m_pWireData, double L, double Ipk, double Irms, double freq,
        double Temp, double TempMax, double waveform[1200], int& turns, int& Gauge, double&
        Bpksave, double& Jwiresave, double& Pcoresave, double& Pwindsave, double& deltaTsave,
        long& IndNum, int& Cores)
{
                bool    Valid = FALSE; //True if the design is valid
                int     N, //The Number of turns
                        m, //Layers of winding used
                        i, //Counting number
                        harmonic; //Counting number
                double  Bpk, //Peak flux density
                        wirearea, //The area taken up the turns of wire needed
                        Jwire, //Current density in the core
                        Pcore, //The core power loss
                        RwireDC, //The wire's total resistance at DC current
                        Resistance, //The wire's resistance per cm
                        heff, //The effective height of the wire
                        MA, MB, MC, MD, M1, DA, DB, DC, DD, D1, //Terms used to find the effects
                                //of skindepth on the wire
                        dc = 0, t[1200]; //Terms used to decompose the inductor current
                double  Xk[240], //The skin depth of the wire
                        Rmult[240], //The resistance as a function of frequency
                        real = 0, imag = 0, //Terms used to do fourier transforms
                        current[240], //The coefficents of the fourier transforms
                        Pwind = 0, //The windings power loss
                        deltaT =0.0, //The temperature rise of the core
                        Leff, //The inductance do to the cores in parallel
                        Ipkeff, //The peak current do to the cores in parallel
                        Irmseff; //The rms current do to the cores in parallel

Leff = L*Cores;
Ipkeff = Ipk/Cores;
Irmseff = Irms/Cores;

MoveFirst();
for (i = 0;i<1200;i++)
{
        dc = waveform[i]/(Cores*1200)+dc;
        t[i] = i/(freq*1200);
}
for (harmonic = 1;harmonic<240;harmonic++)
{
        for (i = 0;i<1200;i++)
        {
                real = waveform[i]/Cores*cos(harmonic*freq*2*PI*t[i])/1200+real;
                imag = -1*waveform[i]/Cores*sin(harmonic*freq*2*PI*t[i])/1200+imag;
        }
        current[harmonic] = 2*sqrt(real*real + imag*imag)/sqrt(2);//rms values
        real = 0;
        imag = 0;
}
current[0] = dc;

m_strSort = "InductorID";
if(CanRestart( ) )
```

```
          Requery( );
MoveFirst();
while(!IsEOF())
{
 if (m_Using)
 {
          Valid = TRUE;
//the number of turns on the inductor
double Nd;
          Nd = (1000*sqrt(1000*Leff/m_AL));
          N = (int)floor(Nd);
          if ((Nd-N) > .5)
                              N++;
//the B field at peak current
          Bpk = 0.1*m_AL*N*Ipkeff/m_Core_Area;
          if (Bpk > 3000)
                    Valid = FALSE;

//chooses the lowest gauge that will fit on the core
          if (Valid)
          {
                    m_pWireData->MoveFirst();
                    while (Valid)
                    {
                              if (m_pWireData->IsEOF())
                              {
                                        Valid = FALSE;
                                        break;
                              }
                              wirearea = N/m_pWireData->m_turns_per_square_inch;
                              if (wirearea < m_Winding_Area)
                                        break;
                              if (!m_pWireData->IsEOF())
                                        m_pWireData->MoveNext();
                    }
          }

//checks the current density
          if (Valid)
          {
                    Jwire = Irmseff/(PI*pow(m_pWireData->m_diameter/2,2));
                    if (Jwire > 3000)
                      Valid = FALSE;
          }

          if (Valid)
          {
                    Pcore = .000000000000916*pow(.001*freq,1.231)*pow(Bpk/2,2.793)*m_Core_Volume;
                    Resistance = m_pWireData->m_Resistance_per_inch*(1+.004*(Temp-20));
double md;
                    md = (N/(0.9*m_Winding_Width/m_pWireData->m_insulated_wire_diameter));
                    m = (int)floor(md);
                    if ((md-m) > .5)
                              m++;
                    if ( m != 1)
                              RwireDC = Resistance*m_LengthPerTurn_Full*N;
```

```
                else RwireDC = Resistance*m_LengthPerTurn_1_layer*N;
                heff = pow(PI/4,.75)*pow(m_pWireData->m_diameter,1.5)/sqrt(m_pWireData->
                        m_insulated_wire_diameter);

                Rmult[0] = 1;
                Pwind = current[0]*current[0]*RwireDC*Rmult[0];
                for (i=1;i<240;i++)
                {
                        Xk[i] = heff/(2.6/(sqrt(i/(freq*2*PI))));
                        MA = cos(Xk[i])*(exp(Xk[i])+exp(-1*Xk[i]));
                        MB = cos(Xk[i])*(exp(Xk[i])-exp(-1*Xk[i]));
                        MC = sin(Xk[i])*(exp(Xk[i])-exp(-1*Xk[i]));
                        MD = sin(Xk[i])*(exp(Xk[i])+exp(-1*Xk[i]));
                        M1 = Xk[i]*(MA*MB+MC*MD+MA*MD-MB*MC)/(MB*MB+MD*MD);
                        DA = cos(Xk[i]/2)*(exp(Xk[i]/2)-exp(-1*Xk[i]/2));
                        DB = cos(Xk[i]/2)*(exp(Xk[i]/2)+exp(-1*Xk[i]/2));
                        DC = sin(Xk[i]/2)*(exp(Xk[i]/2)+exp(-1*Xk[i]/2));
                        DD = sin(Xk[i]/2)*(exp(Xk[i]/2)-exp(-1*Xk[i]/2));
                        D1 = 2*Xk[i]*(DA*DB+DC*DD+DA*DD-DB*DC)/(DB*DB+DD*DD);
                        Rmult[i] = M1 + (m*m-1)*D1/3;
                        Pwind = current[i]*current[i]*RwireDC*Rmult[i]+Pwind;
                }
        }

        if (Valid)
        {
                deltaT = m_Thermal_Resistance *(Pwind+Pcore);
                if (deltaT > (TempMax - Temp))
                        Valid = FALSE;
        }
        if (Valid)
                break;
        }
        MoveNext();
}
if (Valid)
{
        turns = N;
        Gauge = m_pWireData->m_gauge;
        Bpksave = Bpk;
        Jwiresave = Jwire;
        Pcoresave = Pcore;
        Pwindsave = Pwind;
        deltaTsave = deltaT;
        IndNum = (int) m_InductorID;
}

if (Valid == FALSE && Cores < 4)
{
        Cores++;
        Valid = GetInductor(m_pWireData, L, Ipk, Irms, freq, Temp, TempMax, waveform, turns, Gauge,
             Bpksave, Jwiresave, Pcoresave, Pwindsave, deltaTsave, IndNum, Cores);
}
return Valid;

}
```

```
/*//////////////////////////////////////////////////////////////////////
int CSchottkySet::PickSchottky(double& Price, CString& Name, int NumUsing,
                    double MaxTemp, double AveCurrent, int& Package)

PickSchottky chooses a Schottky Diode randomly from the available records in the database.

Parameters:
        Price : The price of the Schottky diode
        Name : The name of the Schottky diode
        NumUsing: The number of available Schottky diodes
        MaxTemp : The max. temperature allowed
        AveCurrent : The average current in the diode
        Package : The case the diode is in 2nd bit is 1 if S2 is T0-220
                4th bit is 1 if S2 is T0-247

Output
The index of the Schottky diode chosen

*//////////////////////////////////////////////////////////////////////
int CSchottkySet::PickSchottky(double& Price, CString& Name, int NumUsing,
                    double MaxTemp, double AveCurrent, int& Package)
{
                int     chosen, num = 0; //Variables used for counting

srand( (unsigned)time( NULL ) );
//First count the number of available records
MoveFirst();
//if no FET available then cause error
if (NumUsing == 0)
        return -1;
//randomly choose a FET
chosen = rand()%NumUsing;
num = 0;
while (!IsEOF() && (chosen > -1))
{
        if (m_Using && ((MaxTemp + 20) < m_Max_Temperature) &&
                        (AveCurrent < m_If))
                chosen--;
        num++;
        MoveNext();
}
if (!IsBOF())
        MovePrev();
Price = (m_Price);
Name = m_Name;
if (m_Package == _T("T0-220"))
        Package = Package+2;
if (m_Package == _T("T0-247"))
        Package = Package+8;
MoveFirst();
return num;
}
```

```
/*/////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetMaxPowLossS(int CellNum, double Freq, double Inductor, double& PowMax,
    double& S2PcapSave, double& S2ThermalResistanceSave, int Mode, int SchottkyNum)

GetMaxPowLossS sweeps the range of possible input and output voltages in order to determine when a
Schottky Diode has the maximum power loss.

Parameters:
        CellNum : Number of cells in the converter
        Freq : The switching frequency of the converter
        Inductor : The value of the converter inductance
        Powmax : The maximum power loss
        S2PcapSave : For the worst case power loss, the power lost by the FET
                due to the capacitance across the schottky diode
        S2ThermalResistanceSave : The thermal resistance for the diode with worst case
                power loss
        Mode : CM or DCM
        SchottkyNum : The index of the Schottky diode used

*/////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetMaxPowLossS(int CellNum, double Freq, double Inductor, double& PowMax,
    double& S2PcapSave, double& S2ThermalResistanceSave, int Mode, int SchottkyNum)
{
                int     x, y;    //Counting indexes
                double  Vin, //Input Voltage
                        Vout, //Output Voltage
                        CurrentPeak, //The peak current in the Schottky Diode
                        S1rms, //The rms current in switch 1
                        S2rms, //The rms current in switch 2
                        waveform[1200], //Vectors of the current in the inductor
                        waveformS2[1200], //Vectors of the current in the the Schottky diode
                        Iinitial, //The current when the Schottky diode turns off
                        S2Pcond, //Conduction power loss in the schottky diode
                        S2Pcap, //for the worst case power loss, the power lost by the FET
                                //due to the capacitance across the schottky diode
                        S2ThermalResistance, //The thermal resistance for the diode with worst case
                                //power loss
                        Irms; //The rms current in the inductor
                bool    Valid; //True if no error occured

 m_SchottkySet.MoveFirst();
 for (x = 1;x < SchottkyNum; x++)
        if (!m_SchottkySet.IsEOF())
                m_SchottkySet.MoveNext();
 PowMax = 0.0;
 for (x = 0; x < 11; x++)
 {
        Vin = m_ConfigurationSet.m_Vinmin + (m_ConfigurationSet.m_Vinmax -
            m_ConfigurationSet.m_Vinmin) * x/10;
        for (y=0;y<11;y++)
        {
            Vout = m_ConfigurationSet.m_Voutmin + (m_ConfigurationSet.m_Voutmax -
                m_ConfigurationSet.m_Voutmin)*y/10;
            Valid = GetInductorCurrentInfo(CellNum, Freq, Inductor,Vout, Vin, CurrentPeak, Irms,
                 S1rms, S2rms, waveform, waveformS2, Iinitial, Mode);
            if (Valid == FALSE)
```

```
                continue;
            m_SchottkySet.PowLoss(waveformS2, Vin, Freq, S2Pcond, S2Pcap, S2ThermalResistance);
            if (S2Pcond > PowMax)
            {
                PowMax = S2Pcond;
                S2PcapSave = S2Pcap;
                S2ThermalResistanceSave = S2ThermalResistance;
            }
        }
  }
}

/*/////////////////////////////////////////////////////////////////////
int CDiodeSet::PickDiode(double& Price, CString& Name, int NumUsing,
                double MaxTemp, double AveCurrent, int& Package)

PickDiode chooses a Diode randomly from the available records in the database.

Parameters:
        Price : The price of the PiN diode
        Name : The name of the PiN diode
        NumUsing: The number of available PiN diodes
        MaxTemp : The max. temperature allowed
        AveCurrent : The average current in the diode
        Package : The case the diode is in 2nd bit is 1 if S2 is T0-220
                4th bit is 1 if S2 is T0-247

Output
The index of the PiN diode chosen

*/////////////////////////////////////////////////////////////////////
int CDiodeSet::PickDiode(double& Price, CString& Name, int NumUsing,
                double MaxTemp, double AveCurrent, int& Package)
{
                int        chosen, num = 0; //Counting variables

srand( (unsigned)time( NULL ) );
MoveFirst();
//if no FET available then cause error
if (NumUsing == 0)
        return -1;
//randomly choose a FET
chosen = rand()%NumUsing;
while (!IsEOF() && (chosen > -1))
{
        if (m_Using && ((MaxTemp + 20) < m_Max_Temperature) &&
                        (AveCurrent < m_If))
//20 is the headroom for the device to be used
                chosen--;
        num++;
        MoveNext();
}
MovePrev();
Price = m_Price;
Name = m_Name;
if (m_Package == _T("T0-220"))
```

```
        Package = Package+2;
if (m_Package == _T("T0-247"))
        Package = Package+8;
MoveFirst();
return num;
}

/*//////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetMaxPowLossD(int CellNum, double Freq, double L,
                double& PowMax, double& S2PswSave, double& S2PcapSave,
                double& S2ThermalResistanceSave, int Mode, int DiodeNum)

GetMaxPowLossD sweeps the range of possible input and output voltages in order to determine when a
Diode has the maximum power loss.

Parameters:
        CellNum : Number of cells in the converter
        Freq : The switching frequency of the converter
        L : The value of the converter inductance
        Powmax : The maximum power loss
        S2PswSave : For the worst case power loss, the switching power loss
        S2PcapSave : For the worst case power loss, the power lost by the FET
                due to the capacitance across the PiN diode
        S2ThermalResistanceSave : The thermal resistance for the diode with worst case
                power loss
        Mode : CM or DCM
        DiodeNum : The index of the PiN diode used

*//////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetMaxPowLossD(int CellNum, double Freq, double L,
                double& PowMax, double& S2PswSave, double& S2PcapSave,
                double& S2ThermalResistanceSave, int Mode, int DiodeNum)
{
                int     x, y;   //Counting indexes
                double  Vin, //Input Voltage
                        Vout, //Output Voltage
                        CurrentPeak, //The peak current in the PiN Diode
                        S1rms, //The rms current in switch 1
                        S2rms, //The rms current in switch 2
                        waveform[1200], //Vectors of the current in the inductor
                        waveformS2[1200], //Vectors of the current in the the PiN diode
                        Iinitial, //The current when the PiN diode turns off
                        S2Pcond, //Conduction power loss in the PiN diode
                        S2Psw, //Switching power loss in the PiN diode
                        S2Pcap, //for the worst case power loss, the power lost by the FET
                                //due to the capacitance across the PiN diode
                        S2ThermalResistance, //The thermal resistance for the diode with worst case
                                //power loss
                        Irms; //The rms current in the inductor
                bool    Valid; //True if no error occured

m_DiodeSet.MoveFirst();
for (x = 1;x < DiodeNum; x++)
        if (!m_DiodeSet.IsEOF())
                m_DiodeSet.MoveNext();
PowMax = 0.0;
```

```
for (x = 0; x < 11; x++)
{
        Vin = m_ConfigurationSet.m_Vinmin + (m_ConfigurationSet.m_Vinmax -
            m_ConfigurationSet.m_Vinmin)*x/10;
        for (y=0;y<11;y++)
        {
                Vout = m_ConfigurationSet.m_Voutmin + (m_ConfigurationSet.m_Voutmax -
                    m_ConfigurationSet.m_Voutmin)*y/10;
                Valid = GetInductorCurrentInfo(CellNum, Freq, L, Vout, Vin, CurrentPeak, Irms, S1rms,
                    S2rms, waveform, waveformS2, Iinitial, Mode);
                if (Valid == FALSE)
                    continue;
                m_DiodeSet.PowLoss(Iinitial, waveformS2, Vin, Freq, S2Pcond, S2Psw, S2Pcap,
                    S2ThermalResistance);
                if (S2Pcond > PowMax)
                {
                        PowMax = S2Pcond;
                        S2PcapSave = S2Pcap;
                        S2PswSave = S2Psw;
                        S2ThermalResistanceSave = S2ThermalResistance;
                }
        }
}
}

/*/////////////////////////////////////////////////////////////////////////
bool CHeatSinkSet::PickHeatSink(double DeltaTAllowed, double FETTemp, double S2Temp,
                double& TempHeatSink, double& TempHeatSink1, int& SinkNum, int& SinkNum1,
                double FETPtot, double S2Ptot, int& Length, double Rcs, int Packages)

PickHeatSink chooses an appropriate heatsink based upon the maximum power losses the devices may
endure.

Parameters:
        DeltaTAllowed : The difference between the max temperature and the ambient
        FETTemp : The temperature rise on the MOSFET
        S2Temp : The temperature rise on switch 2
        TempHeatSink : The temperature rise on the first heatsink
        TempHeatSink1 : The temperature rise on the second heatsink
        SinkNum : The index of the first heatsink
        SinkNum1 : The index of the second heatsink
        FETPtot : The max power dissapation from the FET
        S2Ptot : The max power dissapation from switch 2
        Length : The length of the extrusion
        Rcs : The thermal resistance from case to sink
        Packages : The casing of the devices
                1st bit is 1 if S1 is T0-220          2nd bit is 1 if S2 is T0-220
                3rd bit is 1 if S1 is T0-247          4th bit is 1 if S2 is T0-247

Output
True is a heatsink is designed successively
*/////////////////////////////////////////////////////////////////////////
bool CHeatSinkSet::PickHeatSink(double DeltaTAllowed, double FETTemp, double S2Temp,
                double& TempHeatSink, double& TempHeatSink1, int& SinkNum, int& SinkNum1,
                double FETPtot, double S2Ptot, int& Length, double Rcs, int Packages)
{
```

```
                bool    Valid;   //True if heatsink is valid
                double  DeltaT, //The temperature rise
                        Ptot, //The power dissapated from both devices
                        Rl; //the ratio of thermal resistances
                int     doubles, //The number of heatsink that accept 2 devices
                        singles, //The number of heatsink that accept 1 device
                        num, //Variable used for counting
                        total, //Total number of heatsinks
                        extru, //The number of extruded heatsinks
                        record; //The index of the database table

Ptot = FETPtot + S2Ptot;
doubles = 0;
singles = 0;
Valid = FALSE;
Length = 0;
total = 0;
extru = 0;
SinkNum1 = -1;
MoveFirst();
while(!IsEOF())
{
        if (m_Using)
        {
        if (m_Count == 2)
                doubles++;
        else if (m_Count == 1)
                singles++;
        else if (m_Count == 0)
                extru++;
        }
        MoveNext();
}
total = doubles+singles+extru;
srand( (unsigned)time( NULL ) );
num = rand()%total;
m_strSort = "rank";
if(CanRestart( ) )
        Requery( );
MoveFirst();

//Use a heatsink designed for two chips
if (num < doubles)
{
        SinkNum1 = -1;
        TempHeatSink1 = -1;
        while(!IsEOF())
        {
          if (m_Using && m_Count == 2)
          {
                Valid = TRUE;
                if (m_HighModel)
                        TempHeatSink = m_R2*Ptot*Ptot+m_R1*Ptot+m_R0;
                else
                        TempHeatSink = Ptot*m_Thermal_Resistance;
                if (FETTemp > S2Temp)
                {
                        DeltaT = TempHeatSink+FETTemp + Rcs*FETPtot;
                        if (DeltaT > DeltaTAllowed)
                                Valid = FALSE;
                }
                else
                {
                        DeltaT = TempHeatSink+S2Temp + Rcs*S2Ptot;
                        if (DeltaT > DeltaTAllowed)
                                Valid = FALSE;
                }
                if (Valid)
                {
                        SinkNum = m_rank;
                        break;
                }
           }
           MoveNext();
        }
}
else if (num < singles+doubles)
{
        while(!IsEOF())
        {
         //First the FET
                if ((Packages&1 == 1 && m_T0_220) || (Packages&2 == 2 && m_T0_247))
                        if (m_Using && m_Count == 1)
                        {
                           Valid = TRUE;
                           if (m_HighModel)
                                TempHeatSink = m_R2*FETPtot*FETPtot+m_R1*FETPtot+m_R0;
                           else
                                TempHeatSink = FETPtot*m_Thermal_Resistance;
                           DeltaT = TempHeatSink+FETTemp + Rcs*FETPtot;
                           if (DeltaT > DeltaTAllowed)
                                Valid = FALSE;
                           else break;
                        }
                MoveNext();
        }
        if (Valid)
                SinkNum = m_rank;
        MoveFirst();
        if (Valid)
        {
                while(!IsEOF())
                {
                 //Next S2
                        if ((Packages&4 == 4 && m_T0_220) || (Packages%8 == 8 && m_T0_247))
                           if (m_Using && m_Count == 1)
                           {
                                Valid = TRUE;
                                if (m_HighModel)
                                   TempHeatSink1 = m_R2*S2Ptot*S2Ptot+m_R1*S2Ptot+m_R0;
                                else
                                   TempHeatSink1 = S2Ptot*m_Thermal_Resistance;
                                DeltaT = TempHeatSink1+S2Temp + Rcs*S2Ptot;
                                if (DeltaT > DeltaTAllowed)
                                    Valid = FALSE;
                                else break;
                            }
                            MoveNext();
                }
            if (Valid)
            SinkNum1 = m_rank;
            }
}
if (!Valid)
//if nothing available goto extrusion design
{
        SinkNum1 = -1;
        m_strFilter = "Count = 0";
        m_strSort = "ThermalResistance DESC";
        Requery();
        if (IsEOF() && IsBOF())
        {
                m_strFilter = "";
                Requery();
        }
        else
        {
                record = GetRandomNumber(0, extru);
                SetAbsolutePosition(record);
                while(1)
                {
                    if (m_Using && m_Count == 0)
                    {
                        Valid = TRUE;
                        if (FETTemp > S2Temp)
                                Rl = DeltaTAllowed/(Ptot)-FETTemp/FETPtot-Rcs;
                        else
                                Rl = DeltaTAllowed/(Ptot)-S2Temp/S2Ptot-Rcs;
                        Rl = Rl/m_Thermal_Resistance;//the ratio of thermal resistances
                        Length = (int)ceil(62.1934*Rl*Rl-308.7066*Rl+408.9255);
                                //the constants are from the thermoally book for determining the rating
                                //for various lengths of heatsinks
                        if (Length < 40)
                        {
                                if (record == 1)
                                {
                                    Valid = FALSE;
                                    m_strFilter = "";
                                    Requery();
                                    break;
                                }
                                extru = record-1;
                                record = GetRandomNumber(0, extru);
                                SetAbsolutePosition(record);
                                continue;
                        }
                        if (Length > 150)
                                Valid = FALSE;
                        else
                        {
                                TempHeatSink = Rl*m_Thermal_Resistance*Ptot;
                                SinkNum = m_rank;
                                m_strFilter = "";
                                Requery();
                                break;
                        }
                }
                if (record == extru)
                {
                        m_strFilter = "";
                        m_strSort = "rank";
                        Requery();
                        break;
                }
                record = GetRandomNumber(record, extru);
                SetAbsolutePosition(record);
                }//while(1)
        }
        if (!Valid)
                TempHeatSink = 0;
}//extrusion
else TempHeatSink = max(TempHeatSink, TempHeatSink1);
return Valid;
}

/*//////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetWorstCaseHarmonics(int CellNum, double Inductor,
        double Freq, double WCIn[ ], double WCOut[ ], int Mode)
```

GetWorstCaseHarmonics determines the maximum amplitude of each harmonic in the input and output currents.

Parameters:
- CellNum : The number of cells in the converter
- Inductor : The total inductance of the converter
- Freq : The switching frequency
- WCIn[ ] : Vector containing the harmonic amplitudes for the current through the MOSFET
- WCOut[ ] : Vector containing the harmonic amplitudes for the current through the inductor
- Mode : 1=Continuous Mode 2=Discontinuous Mode 3=Combo determined by voltages

```
*//////////////////////////////////////////////////////////////////////
void CTIMCADDoc::GetWorstCaseHarmonics(int CellNum, double Inductor,
        double Freq, double WCIn[ ], double WCOut[ ], int Mode)
{
                int     i, j, x, y, //Counting indexes
                        delaypts, //The number of points an interleaved cell is delayed by
                        harmonic; //Counts the harmonics
                double  Vin, //Input voltage
                        Vout, //Output voltage
                        realIn, realOut, //The real part of the harmonic currents
                        imagIn, imagOut; //The imaginary part of the harmonic currents
                double  wfOut[1200], wfS2[1200], //The current waveforms for one cell
                        totalOut[1200],   totalIn[1200], //The current waveforms totaled
                        HarmonicsIn[250], HarmonicsOut[250], //The harmonic content
                t[1200]; //time vector
        double  junk1, junk2, junk3, junk4, //ignored variables
                Iinitial, //The initial current when the MOSFET turns on
                Vinmin, Vinmax, //The range of input voltages
                Voutmin, Voutmax; //The range of output voltages
        bool    Valid; //True if no erros have occured


//voltages range considered are voltages that will occur then the car is running because EMI specs are
//intended for normal operation not extreme cases
Vinmin = 33;
Vinmax = 43;
Voutmin = 12;
Voutmax = 14.3;
for (i = 0;i<250;i++)
{
        WCIn[i] = 0;
        WCOut[i] = 0;
}
for (x = 0; x < 11; x++)
{
        Vin = Vinmin + (Vinmax - Vinmin)*x/10;
        for (y=0;y<11;y++)
        {
         Vout = Voutmin + (Voutmax - Voutmin)*y/10;
         for (i = 0; i<1200;i++)
         {
                totalIn[i] = 0;
                totalOut[i] = 0;
         }
         for (i = 1; i <= CellNum; i++)
         {
                Valid = GetInductorCurrentInfo(CellNum, Freq, Inductor, Vout, Vin, junk1, junk2,
                    junk3, junk4, wfOut, wfS2, Iinitial, Mode);
                if (Valid == FALSE)
                        continue;
                delaypts = (int) 1200*(i-1)/CellNum;

                for (j = 0; j<1200/CellNum; j++)
                {
                 if (delaypts == 0)
                 {
                  totalOut[j] = totalOut[j] + 50*wfOut[j];
                  totalIn[j] = totalIn[j] + 50*(wfOut[j] - wfS2[j]);
                 }
                 else
                 {
                  totalOut[j] = totalOut[j] + 50*wfOut[j+1200 - delaypts];
                  totalIn[j] = totalIn[j] + 50*(wfOut[j+1200-delaypts]-wfS2[j+1200-delaypts]);
                 }
                }
         }
         HarmonicsIn[0] = 0;
         HarmonicsOut[0] = 0;

         for (i = 0;i<1200/CellNum;i++)
        {
                HarmonicsOut[0] = totalOut[i]*CellNum/1200+HarmonicsOut[0];
                HarmonicsIn[0] = totalIn[i]*CellNum/1200+HarmonicsIn[0];
                t[i] = i/(Freq*1200);
        }

        for (harmonic = 1;(harmonic*CellNum*Freq<30000000) && (harmonic <= 250);harmonic++)
        {
                realOut = 0;
                imagOut = 0;
                realIn = 0;
                imagIn = 0;

                for (i = 0;i<1200/CellNum;i++)
                {
                 realOut = totalOut[i]*cos(harmonic*CellNum*Freq*2*PI*t[i])*CellNum/1200+realOut;
                 imagOut = -1* totalOut[i]* sin(harmonic*CellNum*Freq*2*PI*t[i]) *CellNum/1200+
                        imagOut;
                 realIn = totalIn[i]*cos(harmonic*CellNum*Freq*2*PI*t[i])*CellNum/1200+realIn;
                 imagIn = -1*totalIn[i]*sin(harmonic*CellNum*Freq*2*PI*t[i])* CellNum/1200+
                        imagIn;
                }
                HarmonicsOut[harmonic] = sqrt(realOut*realOut + imagOut*imagOut);
                HarmonicsIn[harmonic] = sqrt(realIn*realIn + imagIn*imagIn);
        }
        for (i = 0;i<250;i++)
        {
                if (HarmonicsOut[i] > WCOut[i])
                        WCOut[i] = HarmonicsOut[i];
                if (HarmonicsIn[i] > WCIn[i])
                        WCIn[i] = HarmonicsIn[i];
        }
        }
  }

}
```

```
/*///////////////////////////////////////////////////////////////////////
int CCapacitorSet::PickCap(double Temp, double Voltage, double& Capacitance, double& ESR, double&
    ESL, double& Price, double& Weight, double& Volume, int Type)
```

PickCap chooses a capacitor randomly from the available records in the database.

Parameters:

```
        Temp : The ambient temperature
        Voltage : The bus voltage across the capacitor
        Capacitance : The capacitance of the capacitor chosen
        ESR : The equavalent resistance of the capacitor chosen
        ESL : The equavalent inductance of the capacitor chosen
        Price : The price of the capacitor chosen
        Weight : The weight of the capacitor chosen
        Volume : The volume of the capacitor chosen
        Type : The type of capacitor wanted
                SWITCHING = 2, BUS = 3, DAMPING = 5
```

```
Output
The index of the capacitor chosen, -1 if there is an error
*////////////////////////////////////////////////////////////////////////
int CCapacitorSet::PickCap(double Temp, double Voltage, double& Capacitance,
                    double& ESR, double& ESL, double& Price, double& Weight, double& Volume, int
Type)
{
          int       chosen, //Counting variable
                    Capnum = 0, //The index of the capacitor
                    NumUsing, //Number of available capacitors
                    PossibleTypes; //The type of capacitors

NumUsing = 0;
srand( (unsigned)time( NULL ) );
//First count the number of available records
MoveFirst();
while (!IsEOF())
{
          PossibleTypes = 1;
          if (m_Switching == TRUE)
                    PossibleTypes *= 2;
          if (m_Bus == TRUE)
                    PossibleTypes *= 3;
          if (m_Damping == TRUE)
                    PossibleTypes *= 5;

          if (m_Using && (PossibleTypes%Type == 0))
          {
                    if ((Temp <= m_Volt_Temp_Breakpoint) && (Voltage < m_Nominal_Voltage))
                              NumUsing++;
                    if ((Temp <= m_Max_Temp) && (Voltage < m_Nominal_Voltage+
                                        (Temp-m_Volt_Temp_Breakpoint)*(m_Nominal_Voltage-
                                        m_DCvoltAtTemp)/(m_Volt_Temp_Breakpoint-m_Max_Temp)))
                              NumUsing++;
          }
          MoveNext();
}
//if no available then cause error
if (NumUsing == 0)
          return -1;
//randomly choose a Cap
chosen = rand()%NumUsing;
MoveFirst();
Capnum = 0;
while (!IsEOF() && (chosen > -1))
{
          PossibleTypes = 1;
          if (m_Switching == TRUE)
                    PossibleTypes *= 2;
          if (m_Bus == TRUE)
                    PossibleTypes *= 3;
          if (m_Damping == TRUE)
                    PossibleTypes *= 5;

          if (m_Using && (PossibleTypes%Type == 0))
```

```
        {
                if ((Temp <= m_Volt_Temp_Breakpoint) && (Voltage < m_Nominal_Voltage))
                        chosen--;
                if ((Temp <= m_Max_Temp) && (Voltage < m_Nominal_Voltage+
                                (Temp-m_Volt_Temp_Breakpoint)*(m_Nominal_Voltage-
                                m_DCvoltAtTemp)/(m_Volt_Temp_Breakpoint-m_Max_Temp)))
                        chosen--;
        }
        Capnum++;
        MoveNext();
}
if (!(IsBOF()))
 MovePrev();
else return -1;
Capacitance = m_Capacitance;
ESR = m_ESR;
ESL = m_ESL;
Price = m_Price;
Weight = m_Weight;
Volume = m_Volume;
Capnum = m_rank;
MoveFirst();
return Capnum;
}


/*//////////////////////////////////////////////////////////////////////////
int CConfiguration2Set::GetFilterType()

GetFilterType chooses a filter randomly from the available records in the database.

Output
The filter type number
*//////////////////////////////////////////////////////////////////////////
int CConfiguration2Set::GetFilterType()
{
        int                     count, chosen;

        count = 0;
        if (m_Filter1)
                count++;
        if (m_Filter2)
                count++;
        if (m_Filter3)
                count++;


        chosen = rand()%count;

        if (m_Filter1)
                chosen--;
        if (chosen < 0)
                return 1;
        if (m_Filter2)
                chosen--;
        if (chosen < 0)
```

```
			return 2;
		if (m_Filter3)
			chosen--;
		if (chosen < 0)
			return 3;
		return 0;
}

/*////////////////////////////////////////////////////////////////////////
int CCapacitorSet::PickSetCap(double Temp, double Voltage, double CapDesired, double& ESR, double&
		ESL, int& Caps, int CapMax, double& Capacitance, double& Price, double& Weight, double&
		Volume, int Type)

PickSetCap chooses a capacitor from the available records in the database that bests matches a desired
capacitance.

Parameters:
		Temp : The ambient temperature
		Voltage : The bus voltage across the capacitor
		CapDesired : The capacitance desired in the chosen capacitor
		ESR : The equavalent resistance of the capacitor chosen
		ESL : The equavalent inductance of the capacitor chosen
		Caps : The number of capacitors in parallel
		CapMax : The maximum capacitors in parallel allowed
		Capacitance : The capacitance of the capacitor that was picked
		Price : The price of the capacitor chosen
		Weight : The weight of the capacitor chosen
		Volume : The volume of the capacitor chosen
		Type : The type of capacitor wanted
			SWITCHING = 2, BUS = 3, DAMPING = 5

Output
The index of the capacitor chosen, -1 if there is an error
*////////////////////////////////////////////////////////////////////////
int CCapacitorSet::PickSetCap(double Temp, double Voltage, double CapDesired, double& ESR,
				double& ESL, int& Caps, int CapMax, double& Capacitance, double& Price,
				double& Weight, double& Volume, int Type)
{
			int	Capnum = 0, //Counting variable
				NumUsing, //Number of availabe capacitors
				Choose = 0, chosen, //Counting variables
				PossibleTypes; //The types a capacitor could be
			bool	Valid = FALSE; //True is the capacitor is valid
			double	best = 999999; //Used to find best capacitor

//First count the number of available records
MoveFirst();
NumUsing = 0;
while (!IsEOF())
{
		PossibleTypes = 1;
		if (m_Switching == TRUE)
			PossibleTypes *= 2;
		if (m_Bus == TRUE)
			PossibleTypes *= 3;
		if (m_Damping == TRUE)
```

```
				PossibleTypes *= 5;

			if (m_Using && (PossibleTypes%Type == 0))
			{
				if ((Temp <= m_Volt_Temp_Breakpoint) && (Voltage < m_Nominal_Voltage))
					Valid = TRUE;
				if ((Temp <= m_Max_Temp) && (Voltage < m_Nominal_Voltage+
						(Temp-m_Volt_Temp_Breakpoint)*(m_Nominal_Voltage-
						m_DCvoltAtTemp)/(m_Volt_Temp_Breakpoint-m_Max_Temp)))
					Valid = TRUE;
			}
			if (Valid)
			{
				Caps = (int)(CapDesired/m_Capacitance);
				if ((CapDesired/m_Capacitance - Caps) > .5)
					Caps++;
				if (Caps == 0) Caps++;
				if ((fabs(CapDesired-Caps*m_Capacitance) < best) && (Caps <= CapMax))
				{
					best = fabs(CapDesired-Caps*m_Capacitance);
					NumUsing = m_rank;
					Choose = 1;
				}
				else if ((fabs(CapDesired-Caps*m_Capacitance) == best) && (Caps <= CapMax))
					Choose++;
			}

			Valid = FALSE;
			MoveNext();
		}
		//if no available then cause error
		if (NumUsing == 0)
			return -1;
		MoveFirst();
		if (Choose == 1)
		{
			m_Param = NumUsing;
			m_strFilter = "rank = ?";
			Requery();
			ESR = m_ESR;
			ESL = m_ESL;
			Capacitance = m_Capacitance;
			Price = m_Price;
			Weight = m_Weight;
			Volume = Volume;
			Caps = (int)(CapDesired/m_Capacitance);
			if ((CapDesired/m_Capacitance - Caps) > .5)
					Caps++;
			m_strFilter = "";
			Requery();
			return NumUsing;
		}
		srand( (unsigned)time( NULL ) );
		chosen = rand()%Choose;
		Capnum = 0;
		while (!IsEOF() && (chosen > -1))
```

```
{
        PossibleTypes = 1;
        if (m_Switching == TRUE)
                PossibleTypes *= 2;
        if (m_Bus == TRUE)
                PossibleTypes *= 3;
        if (m_Damping == TRUE)
                PossibleTypes *= 5;

        Caps = (int)(CapDesired/m_Capacitance);
        if ((CapDesired/m_Capacitance - Caps) > .5)
                Caps++;
        if (Caps == 0)
                Caps = 1;
        if ((m_Using && (PossibleTypes%Type == 0) && (fabs(CapDesired-Caps*m_Capacitance) ==
            best))&& (Caps <= CapMax))
        {
                if ((Temp <= m_Volt_Temp_Breakpoint) && (Voltage < m_Nominal_Voltage))
                        chosen--;
                if ((Temp <= m_Max_Temp) && (Voltage < m_Nominal_Voltage+
                                (Temp-m_Volt_Temp_Breakpoint)*(m_Nominal_Voltage-
                                m_DCvoltAtTemp)/(m_Volt_Temp_Breakpoint-m_Max_Temp)))
                        chosen--;
        }
        Capnum++;
        MoveNext();
}
if (!(IsBOF()))
  MovePrev();
else return -1;
ESR = m_ESR;
ESL = m_ESL;
Price = m_Price;
Weight = m_Weight;
Volume = Volume;
Capacitance = m_Capacitance;
Caps = (int)(CapDesired/m_Capacitance);
if ((CapDesired/m_Capacitance - Caps) > .5)
        Caps++;
if (Caps == 0)
        Caps = 1;
return Capnum = m_rank;

}

/*/////////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::DesignFilter1(double Freq, double CellNum, double WCHarmonics[ ], double& Q,
    double& L2, double& L3, double C2, double& R, double Rc1, double Rc2, double Lc1, double Lc2)
```

DesignFilter1 designs the first type of filter ensuring that the EMI limits are met.

Parameters:
 Freq : The switching frequency of the converter
 CellNum : Number of cells imployed by the converter
 WCHarmonics[] : A vector of the values each representing theworst case harmonics of a signal, values are
 dc, freq*CellNum, 2*freq*CellNum, etc...

```
 Q : The Q of the filter designed, choosen randomly
 L2 : The value of the inductor that will handel the dc current
 L3 : The inductor that is in series with the resister
 C2 : The capacitor closer to the converter
 R : The rester in the filter
 Rc1 : The ESR for the bus capacitor
 Rc2 : The ESR for the converter capacitor
 Lc1 : The ESL for the bus capacitor
 Lc2 : The ESL for the converter capacitor

Output
The index of the capacitor chosen, -1 if there is an error
*/////////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::DesignFilter1(double Freq, double CellNum, double WCHarmonics[ ], double& Q,
     double& L2, double& L3, double C2, double& R, double Rc1, double Rc2, double Lc1, double Lc2)
{
                        double  C1, //The second capacitor in the circuit
                                L1, //Placeholder variable
                                gain, //The gain of the filter at a given frequency
                                dB, //The worst case harmonics in dB
                                EMI = 66, //The EMI limit at a given frequency
                                num0, num1, num2, num3, num4, num5, den0, den1, den2, den3, den4, den5,
                                        //Terms in the transfer function
                                Rl;     //The load resistance
                        bool    DesignOK = FALSE, //True is design is valid
                                OneWay = TRUE; //Ensures that the while loop is not infinite
                        double  numreal, numimag, nummag, denreal, denimag, denmag, //Terms used in the
                                        //transfer function
                                OldLimit = 100000, //The previous EMI limit
                                OldFreq = 0, //The frequency where the old EMI limit started
                                pole; //The location of the pole of the filter
                        int     i, //Countinf index
                                PossibleQs, //The possible numbers of Q
                                FirstHarmonicLimit; //The harmonic that first falls within the EMI limits
                        CRecordsetStatus        rStatus; //Gets status of the EMI limit recordset

pole = .00001;
if (m_ConfigurationSet.m_Qmin == m_ConfigurationSet.m_Qmax)
        Q = m_ConfigurationSet.m_Qmin;
else
{
        srand( (unsigned)time( NULL ) );
        PossibleQs = (int) ((m_ConfigurationSet.m_Qmax - m_ConfigurationSet.m_Qmin)*100);
        Q = (rand()%PossibleQs)*.01 + m_ConfigurationSet.m_Qmin;
}

while (!DesignOK)
{
        L1 = (4*pole*pole)/(5*C2);
        L2 = (25*L1)/4;
        L3 = L1*pow(Q,4);
        R = (5*pole)/(4*C2);
        C1 = C2/4;
        Rl = 50;
        DesignOK = TRUE;
        m_EMILimitsSet.MoveFirst();
```

```
OldFreq = m_EMILimitsSet.m_Frequency;
OldLimit = m_EMILimitsSet.m_Limit;
FirstHarmonicLimit = (int) ceil(m_EMILimitsSet.m_Frequency/(Freq*CellNum));
num0 = R;
num1 = R*Rc1*C1+R*Rc2*C2+L2+L3;
num2 = R*Lc2*C2+(Rc2*C2+Rc1*C1)*(L2+L3)+R*Rc1*Rc2*C1*C2+Lc1*C1*R;
num3 = Lc2*C2*(L3+L2)+Lc2*R*C2*Rc1*C1+Rc1*Rc2*C1*C2*(L2+L3)+
      R*Lc1*C1*Rc2*C2+ Lc1*C1*(L2+L3);
num4 = Rc1*Lc2*(L3+L2)*C1*C2+R*Lc1*Lc2*C1*C2+Rc2*Lc1*C1*C2*(L2+L3);
num5 = Lc2*(L2+L3)*Lc1*C1*C2;
den0 = R;
den1 = R*Rl*(C1+C2)+R*Rc1*C1+R*Rc2*C2+L2+L3;
den2 = R*C1*Lc1+R*Rl*Rc2*C1*C2+R*Rc1*Rc2*C1*C2+R*Lc2*C2+R*L2*C2+
     Rl*(L2+L3)*(C1+C2)+Rc1*(L1+L2)*C1+Rc2*(L2+L3)*C2+R*Rl*Rc1*C1*C2;
den3 = R*Rc2*Lc1*C1*C2+R*Rl*(Lc2+Lc1)*C1*C2+ R*Rc1*Lc2*C1*C2+ Lc1*(L2+L3)*C1+
     Rl*(Rc2+Rc1)*(L2+L3)*C1*C2+ Rc1*Rc2*(L2+L3)*C1*C2+ Lc2*(L2+L3)*C2+
     R*Rl*C1*L2*C2+ L2*L3*C2+R*L2*C2*Rc1*C1;
den4 = R*Lc1*Lc2*C1*C2+Rc2*Lc1*(L2+L3)*C1*C2+Rl*(Lc2+Lc1)*(L2+L3)*C1*C2+
     Rc1*Lc2*(L2+L3)*C1*C2+(Rl+Rc1)*L2*L3*C2*C1+R*L2*C2*Lc1*C1;
den5 = Lc1*Lc2*C1*C2*(L2+L3)+L2*L3*C2*Lc1*C1;
for (i=1;i<50;i++)
{
          numreal = num0-pow(2*PI*Freq*i*CellNum,2)*num2+ pow(2*PI*Freq*i*CellNum,4)*
                    num4;
          numimag = 2*PI*Freq*i*CellNum*num1-pow(2*PI*Freq*i*CellNum,3)*num3+
                    pow(2*PI*Freq*i*CellNum,5) * num5;
          nummag = sqrt(numreal*numreal+numimag*numimag);
          denreal = den0-pow(2*PI*Freq*i*CellNum,2)*den2+ pow(2*PI*Freq*i*CellNum,4)*
                   den4;
          denimag = 2*PI*Freq*i*CellNum*den1-pow(2*PI*Freq*i*CellNum,3)* den3+
                    pow(2*PI*Freq*i*CellNum,5)* den5;
          denmag = sqrt(denreal*denreal+denimag*denimag);
          gain = nummag/denmag;
          dB = 20*log10(gain*WCHarmonics[i]*1000000);
          while ((Freq*i*CellNum > m_EMILimitsSet.m_Frequency) &&
                    (!m_EMILimitsSet.IsEOF()))
          {
                    OldFreq = m_EMILimitsSet.m_Frequency;
                    OldLimit = m_EMILimitsSet.m_Limit;
                    if (!m_EMILimitsSet.IsEOF())
                    {
                              m_EMILimitsSet.MoveNext();
                              m_EMILimitsSet.GetStatus(rStatus);
                              if (rStatus.m_lCurrentRecord == 0)
                                        EMI = 100000; //no limit so use a high number
                              else
                              EMI = (OldLimit - (OldLimit-m_EMILimitsSet.m_Limit)*
                                        (Freq*i*CellNum-OldFreq)/(m_EMILimitsSet.m_Frequency-
                                        OldFreq));
                     }
          };
          if (Freq*i*CellNum < m_EMILimitsSet.m_Frequency)
          {
                    m_EMILimitsSet.GetStatus(rStatus);
                    if (rStatus.m_lCurrentRecord == 0)
                              EMI = 100000; //no limit so use a high number
```

```
                        else
                                EMI = (OldLimit - (OldLimit-m_EMILimitsSet.m_Limit)*
                                 (Freq*i*CellNum-OldFreq)/(m_EMILimitsSet.m_Frequency-
                                 OldFreq));
                }
                else EMI = OldLimit;
                if (dB > EMI+1)
                {
                        if (pole > .0001)
                                return FALSE;
                        DesignOK = FALSE;
                        pole+= .0000001;
                        OneWay = FALSE;
                        break;
                }
                if (i == FirstHarmonicLimit && (OneWay) && (EMI - dB) > 2)
                {
                        DesignOK = FALSE;
                        pole-= .00000003;
                        if (pole < 0)
                                return FALSE;
                        break;
                }
        }
}
return TRUE;
}

/*/////////////////////////////////////////////////////////////////////////
int CEMIInductorSet::DesignEMIInductor(CWireSet* m_pWireData,double Inductance, double Current,
    double TempRange, double PLossMax, BOOL SimpleWindings, int& Gauge, int& Turns, int&
    Windings, double& PowLoss, double& Price, double& Weight, double& Volume)

DesignEMIInductor designs the inductor for the filters.

Parameters:
        m_pWireData : The Wire database
        Inductance : The desired inductance
        Current : The dc current through the inductor
        TempRange : The difference between the max. temperature and ambient
        PLossMax The maximum allows power loss
        SimpleWindings : Determines if simple or complex windings are used
        Gauge : The gauge of wire used
        Turns : The number of turns used
        Windings : The number of sets of turns used
        PowLoss : The power loss of the inductor
        Price : The price of the inductor
        Weight : The weight of the inductor
        Volume : The volume if the inductor

Output
The index of the inductor chosen, -1 if there is an error
*/////////////////////////////////////////////////////////////////////////
int CEMIInductorSet::DesignEMIInductor(CWireSet* m_pWireData,double Inductance, double Current,
    double TempRange, double PLossMax, BOOL SimpleWindings, int& Gauge, int& Turns, int&
    Windings, double& PowLoss, double& Price, double& Weight, double& Volume)
```

```
{
              double   Storage, //The maximum energy an inductor can store
                       Energy, //The energy the inductor needs to store
                       Window, //The usable window area of the inductor
                       Length, // The length of wire need for the turns
                       Resistance, //The resistance of the wire in the turns
                       U; //The percent permeability of the core
              bool     Valid; //True if design is valid

//if no inductor is needed then return zeros for all parameters. Index 16 corresponds to no inductor needed
if (Inductance == 0)
{
        Gauge = 0;
        Turns = 0;
        Windings = 0;
        PowLoss = 0;
        Price = 0;
        Weight = 0;
        Volume = 0;
        return 16;
}
Valid = FALSE;
m_strSort = "Full10 ASC";
Requery();
MoveFirst();
while (!(IsEOF()) && Valid == FALSE)
{
 if (SimpleWindings)
 {
        if (TempRange < 10)
          Storage = m_Simple10;
        else if (TempRange < 25)
          Storage = (TempRange-10)*(m_Simple25-m_Simple10)/15+m_Simple10;
        else if (TempRange < 40)
          Storage = (TempRange-25)*(m_Simple40-m_Simple25)/15+m_Simple25;
        else Storage = m_Simple40;
 }
 else
 {
        if (TempRange < 10)
          Storage = m_Full10;
        else if (TempRange < 25)
          Storage = (TempRange-10)*(m_Full25-m_Full10)/15+m_Full10;
        else if (TempRange < 40)
          Storage = (TempRange-25)*(m_Full40-m_Full25)/15+m_Full25;
        else Storage = m_Full40;
 }
 Energy = Inductance*1000000*Current*Current/2;
 if (Energy > Storage)
 {
          if (!(IsEOF()))
            MoveNext();
          else return -1;
          continue;
 }
 U = m_U1*Energy*Energy+m_U2*Energy+m_U3;
Turns = int (sqrt(Inductance*1000000000/(m_AL*.01*U)));
if (Turns == 0)
        Turns = 1;
if (SimpleWindings)
        Window = m_Window_Area*.22;
else Window = m_Window_Area*.55;
m_pWireData->MoveFirst();
Length = m_MLT*Turns;
while (!(m_pWireData->IsEOF()) && (Valid == FALSE))
{
        if (m_pWireData->m_area*Turns > Window)
        {
                if (!(m_pWireData->IsEOF()))
                  m_pWireData->MoveNext();
                else return -1;
        }
        else
        {
         Resistance = Length*m_pWireData->m_Resistance_per_cm;
         Windings = 1;
         PowLoss = Current*Current*Resistance;
         if (PowLoss > PLossMax)
         {
                Windings = int (PowLoss/PLossMax) + 1;
                PowLoss = Current*Current*Resistance/Windings;
         }
         if (Windings <= 4)
         {
                if ((m_pWireData->m_area*Turns*Windings < Window) &&
                    (pow((PowLoss/m_Surface_Area), .8333)<TempRange))
                        Valid = TRUE;
                else
                {
                        if (!(m_pWireData->IsEOF()))
                          m_pWireData->MoveNext();
                        else return -1;
                }
         }
         else
         {
                 if (!(IsEOF()))
                   MoveNext();
                 else return -1;
                 break;
         }
        }
}
}
Gauge = m_pWireData->m_gauge;
Price = m_Price;
Weight = m_Weight;
Volume = m_Volume;
if (Valid)
        return m_rank;
else return (-1);
}
```

```
/*//////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::DesignFilter2(double C1, double Rc1, double Lc1, double C2, double Rc2, double
    Lc2, double Cd1, double& Rdd1, double Ld1, double Cd2, double& Rdd2, double Ld2, double& L1,
    double& L2, double Freq, int CellNum, double WCHarmonics[])

DesignFilter2 designs the second type of filter ensuring that the EMI limits are met.

Parameters:
        C1 : The value of the capacitor further from the converter
        Rc1 : The ESR for the C1 capacitor
        Lc1 : The ESL for the C1 capacitor
        C2 : The capacitor closer to the converter
        Rc2 : The ESR for the C2 capacitor
        Lc2 : The ESL for the C2 capacitor
        Cd1 : The damping capacitor closer to C1
        Rdd1 : The damping resistor closer to C1
        Ld1 : The ESL for the Cd1 capacitor
        Cd2 : The damping capacitor closer to C2
        Rdd2 : The damping resistor closer to C2
        Ld2 :  : The ESL for the Cd2 capacitor
        L1 : The inductance further from the converter
        L2 :  The inductance closer to the converter
        Freq : The switching frequency of the converter
        CellNum : Number of cells imployed by the converter
        WCHarmonics[] : A vector of the values each representing the
                worst case harmonics of a signal, values are dc, freq*CellNum,
                2*freq*CellNum, etc...

Output
True if no errors occured
*//////////////////////////////////////////////////////////////////////
#pragma optimize("", off)
bool CTIMCADDoc::DesignFilter2(double C1, double Rc1, double Lc1, double C2, double Rc2, double
    Lc2, double Cd1, double& Rdd1, double Ld1, double Cd2, double& Rdd2, double Ld2, double& L1,
    double& L2, double Freq, int CellNum, double WCHarmonics[])
{
                double  gain, //The gain of the filter at a given frequency
                        dB, //The magnitude of the worst case harmonic in dB
                        EMI = 66, //The EMI limit at a given frequency
                        num0, num1, num2, num3, num4, num5, num6, num7, num8, den0, den1, den2,
                                den3, den4, den5, den6, den7, den8, //Terms in the transfer function
                        Rl, Rd1, Rd2; //Resistive values in the filter
                bool    DesignOK = FALSE, //True if the design is valid
                        OneWay = TRUE; //Insures that the while loop is not infinite
                double  numreal, numimag, nummag, denreal, denimag, denmag, //Terms used in the
                                //transfer function
                        OldLimit = 100000, //Previous EMI limit
                        OldFreq = 0, //Frequency where old EMI limit started
                        freqs, //The frequency of intrest
                        oldgain, //The gain of the filter at a previous frequncy
                        freqmingain; //The frequency where the min. gain occured
                int     i, //Counting index
                        once1, once2, //Insure that the while loop in not infinite
                        FirstHarmonicLimit; //The first harmonic that the EMI limits apply to
                CrecordsetStatus        rStatus; //The status of the EMI database
```

```
once1 = once2 = SET;
freqmingain = 800000;
L2 = .00001;
L1 = .00001;

if (2*Rdd1 < .1)
  Rd1 = .1;
else Rd1 = 2*Rdd1;

if (2*Rdd2 < .1)
  Rd2 = .1;
else Rd2 = 2*Rdd2;

Rl = 50;
while (!DesignOK)
{
        if (L2 < .00000001)
                return FALSE;
        if (L2 > .001)
                return FALSE;

        DesignOK = TRUE;
        m_EMILimitsSet.MoveFirst();
        FirstHarmonicLimit = (int) ceil(m_EMILimitsSet.m_Frequency/(Freq*CellNum));
        num0 = 1;
        num1 = Rc1*C1+Rd1*Cd1+Rc2*C2+Rd2*Cd2;
         num2 Lc1*C1+ Ld1*Cd1+ Lc2*C2+ Ld2*Cd2+ Rc1*C1*Rd1*Cd1+ Rc2*C2*Rd2*Cd2+
             Rd2*Cd2*Rc1*C1+ Rc2*C2*Rc1*C1+ Rc2*C2*Rd1*Cd1+ Rd2*Cd2*Rd1*Cd1;
        num3 = Rc1*C1*Ld1*Cd1+ Lc1*C1*Rd1*Cd1+ Rc2*C2*Ld2*Cd2+ Lc2*C2*Rd2*Cd2+
             Rd2*Cd2*Lc1*C1+ Ld2*Cd2*Rc1*C1+ Rc2*C2*Lc1*C1+ Rc2*C2*Rd2*Cd2*Rc1*C1+
             Lc2*C2*Rc1*C1+ Lc2*C2*Rd1*Cd1+ Rc2*C2*Ld1*Cd1+ Ld2*Cd2*Rd1*Cd1+
             Rd2*Cd2*Ld1*Cd1+ Rc2*C2*Rd2*Cd2*Rd1*Cd1+ Rd2*Cd2*Rc1*C1*Rd1*Cd1+
             Rc2*C2*Rc1*C1*Rd1*Cd1;
        num4 = Lc1*C1*Ld1*Cd1+ Lc2*C2*Ld2*Cd2+ Ld2*Cd2*Lc1*C1+
             Rc2*C2*Rd2*Cd2*Lc1*C1+ Rc2*C2*Ld2*Cd2*Rc1*C1+ Lc2*C2*Lc1*C1+
             Lc2*C2*Rd2*Cd2*Rc1*C1+ Lc2*C2*Ld1*Cd1+ Ld2*Cd2*Ld1*Cd1+
             Rc2*C2*Rd2*Cd2*Ld1*Cd1+ Rc2*C2*Ld2*Cd2*Rd1*Cd1+
             Lc2*C2*Rd2*Cd2*Rd1*Cd1+ Rd2*Cd2*Rc1*C1*Ld1*Cd1+
             Rd2*Cd2*Lc1*C1*Rd1*Cd1+ Ld2*Cd2*Rc1*C1*Rd1*Cd1+ Rc2*C2*Rc1*C1*Ld1*Cd1+
             Rc2*C2*Lc1*C1*Rd1*Cd1+ Rc2*C2*Rd2*Cd2*Rc1*C1*Rd1*Cd1+
             Lc2*C2*Rc1*C1*Rd1*Cd1;
        num5 = Rc2*C2*Ld2*Cd2*Lc1*C1+ Lc2*C2*Rd2*Cd2*Lc1*C1+ Lc2*C2*Ld2*Cd2*Rc1*C1+
             Rc2*C2*Ld2*Cd2*Ld1*Cd1+ Lc2*C2*Rd2*Cd2*Ld1*Cd1+ Lc2*C2*Ld2*Cd2*Rd1*Cd1+
             Rd2*Cd2*Lc1*C1*Ld1*Cd1+ Ld2*Cd2*Rc1*C1*Ld1*Cd1+ Ld2*Cd2*Lc1*C1*Rd1*Cd1+
             Rc2*C2*Lc1*C1*Ld1*Cd1+ Rc2*C2*Rd2*Cd2*Rc1*C1*Ld1*Cd1+
             Rc2*C2*Rd2*Cd2*Lc1*C1*Rd1*Cd1+ Rc2*C2*Ld2*Cd2*Rc1*C1*Rd1*Cd1+
             Lc2*C2*Rc1*C1*Ld1*Cd1+ Lc2*C2*Lc1*C1*Rd1*Cd1+
             Lc2*C2*Rd2*Cd2*Rc1*C1*Rd1*Cd1;
        num6 = Lc2*C2*Ld2*Cd2*Lc1*C1+ Lc2*C2*Ld2*Cd2*Ld1*Cd1+
             Ld2*Cd2*Lc1*C1*Ld1*Cd1+ Rc2*C2*Rd2*Cd2*Lc1*C1*Ld1*Cd1+
             Rc2*C2*Ld2*Cd2*Rc1*C1*Ld1*Cd1+ Rc2*C2*Ld2*Cd2*Lc1*C1*Rd1*Cd1+
             Lc2*C2*Lc1*C1*Ld1*Cd1+ Lc2*C2*Rd2*Cd2*Rc1*C1*Ld1*Cd1+
             Lc2*C2*Rd2*Cd2*Lc1*C1*Rd1*Cd1+ Lc2*C2*Ld2*Cd2*Rc1*C1*Rd1*Cd1;
        num7 = Rc2*C2*Ld2*Cd2*Lc1*C1*Ld1*Cd1+ Lc2*C2*Rd2*Cd2*Lc1*C1*Ld1*Cd1+
             Lc2*C2*Ld2*Cd2*Rc1*C1*Ld1*Cd1+ Lc2*C2*Ld2*Cd2*Lc1*C1*Rd1*Cd1;
        num8 = Lc2*C2*Ld2*Cd2*Lc1*C1*Ld1*Cd1;
```

```
den0 = 1;
den1 = Rl*C2+Rl*Cd1+Rl*Cd2+Rl*C1+Rc1*C1+Rd1*Cd1+Rc2*C2+Rd2*Cd2;
den2 = L1*C2+ L1*Cd2+ L1*C1+ L2*Cd2+ L2*C2+ L1*Cd1+ Lc1*C1+ Ld1*Cd1+ Lc2*C2+
    Ld2*Cd2+ Rc2*C2*Rd2*Cd2+ Rc1*C1*Rd1*Cd1+ Rl*Rc1*C1*Cd1+ Rl*Rd1*C1*Cd1+
    Rd2*Cd2*Rd1*Cd1+ Rd2*Cd2*Rc1*C1+ Rd2*Cd2*Rl*Cd1+ Rd2*Cd2*Rl*C1+
    Rc2*C2*Rd1*Cd1+ Rc2*C2*Rc1*C1+ Rc2*C2*Rl*Cd1+ Rc2*C2*Rl*C1+
    Rl*Rc2*C2*Cd2+ Rl*Rd2*C2*Cd2+ Cd1*Rd1*Rl*Cd2+ Cd1*Rd1*Rl*C2+
    C1*Rc1*Rl*Cd2+ C1*Rc1*Rl*C2;
den3 = Rc2*C2*Ld2*Cd2+ Lc2*C2*Rd2*Cd2+ Rc1*C1*Ld1*Cd1+ Lc1*C1*Rd1*Cd1+
    L1*Rc1*C1*Cd1+ L1*Rd1*C1*Cd1+ Rl*Lc1*C1*Cd1+ Rl*Ld1*C1*Cd1+
    L2*Rd2*C2*Cd2+ L2*Rc2*C2*Cd2+ L2*C2*C1*Rl+ L2*C2*Cd1*Rl+ L2*Cd1*Rl*Cd2+
    L2*C1*Rl*Cd2+ L2*Rd1*Cd1*Cd2+ L2*Rd1*Cd1*C2+ L2*Rc1*C1*Cd2+
    L2*Rc1*C1*C2+ Lc2*C2*Rd1*Cd1+ Lc2*C2*Rc1*C1+ Lc2*C2*Rl*Cd1+
    Lc2*C2*Rl*C1+ Ld2*Cd2*Rd1*Cd1+ Ld2*Cd2*Rc1*C1+ Ld2*Cd2*Rl*Cd1+
    Ld2*Cd2*Rl*C1+ Rc2*C2*Rd2*Cd2*Rd1*Cd1+ Rc2*C2*Rd2*Cd2*Rc1*C1+
    Rc2*C2*Rd2*Cd2*Rl*Cd1+ Rc2*C2*Rd2*Cd2*Rl*C1+ Rd2*Cd2*Ld1*Cd1+
    Rd2*Cd2*Rc1*C1*Rd1*Cd1+ Rd2*Cd2*Lc1*C1+ Rd2*Cd2*L1*Cd1+ Rd2*Cd2*L1*C1+
    Rd2*Cd2*Rl*Rc1*C1*Cd1+ Rd2*Cd2*Rl*Rd1*C1*Cd1+ Rc2*C2*Ld1*Cd1+
    Rc2*C2*Rc1*C1*Rd1*Cd1+ Rc2*C2*Lc1*C1+ Rc2*C2*Rl*Rc1*C1*Cd1+
    Rc2*C2*L1*Cd1+ Rc2*C2*L1*C1+ Rc2*C2*Rl*Rd1*C1*Cd1+ L1*Rc2*C2*Cd2+
    L1*Rd2*C2*Cd2+ Rl*Lc2*C2*Cd2+ Rl*Ld2*C2*Cd2+ Cd1*Rd1*L1*Cd2+
    Cd1*Rd1*L1*C2+ Cd1*Rd1*Rl*Rc2*C2*Cd2+ Cd1*Rd1*Rl*Rd2*C2*Cd2+
    Cd1*Ld1*Rl*Cd2+ Cd1*Ld1*Rl*C2+ C1*Rc1*L1*Cd2+ C1*Rc1*L1*C2+
    C1*Rc1*Rl*Rc2*C2*Cd2+ C1*Rc1*Rl*Rd2*C2*Cd2+ C1*Cd1*Rc1*Rd1*Rl*Cd2+
    C1*Cd1*Rc1*Rd1*Rl*C2+ C1*Lc1*Rl*Cd2+ C1*Lc1*Rl*C2;
den4 = Lc2*C2*Ld2*Cd2+ Lc1*C1*Ld1*Cd1+ L1*Lc1*C1*Cd1+ L1*Ld1*C1*Cd1+
    L2*Ld2*C2*Cd2+ L2*Lc2*C2*Cd2+ L2*C2*Cd1*L1+ L2*C2*C1*L1+
    L2*C2*Rd1*C1*Cd1*Rl+ L2*C2*Rc1*C1*Cd1*Rl+ L2*C2*Rd2*Cd2*Cd1*Rl+
    L2*C2*Rd2*Cd2*C1*Rl+ L2*Cd1*Rl*Rc2*C2*Cd2+ L2*Cd1*L1*Cd2+
    L2*Rc1*C1*Cd1*Rl*Cd2+ L2*C1*L1*Cd2+ L2*C1*Rl*Rc2*C2*Cd2+
    L2*C1*Rd1*Cd1*Rl*Cd2+ L2*Rd1*Cd1*Rc2*C2*Cd2+ L2*Rd1*Cd1*Rd2*C2*Cd2+
    L2*Rc1*C1*Rc2*C2*Cd2+ L2*Rc1*C1*Rd2*C2*Cd2+ L2*Ld1*Cd1*Cd2+
    L2*Ld1*Cd1*C2+ L2*Rc1*C1*Rd1*Cd1*Cd2+ L2*Rc1*C1*Rd1*Cd1*C2+
    L2*Lc1*C1*Cd2+ L2*Lc1*C1*C2+ Lc2*C2*Ld1*Cd1+ Lc2*C2*Rc1*C1*Rd1*Cd1+
    Lc2*C2*Lc1*C1+ Lc2*C2*L1*Cd1+ Lc2*C2*L1*C1+ Lc2*C2*Rl*Rc1*C1*Cd1+
    Lc2*C2*Rl*Rd1*C1*Cd1+ Ld2*Cd2*Ld1*Cd1+ Ld2*Cd2*Rc1*C1*Rd1*Cd1+
    Ld2*Cd2*Lc1*C1+ Ld2*Cd2*L1*Cd1+ Ld2*Cd2*L1*C1+ Ld2*Cd2*Rl*Rc1*C1*Cd1+
    Ld2*Cd2*Rl*Rd1*C1*Cd1+ Rc2*C2*Rd2*Cd2*Ld1*Cd1+
    Rc2*C2*Rd2*Cd2*Rc1*C1*Rd1*Cd1+ Rc2*C2*Rd2*Cd2*Lc1*C1+
    Rc2*C2*Rd2*Cd2*L1*Cd1+ Rc2*C2*Rd2*Cd2*L1*C1+
    Rc2*C2*Rd2*Cd2*Rl*Rc1*C1*Cd1+ Rc2*C2*Rd2*Cd2*Rl*Rd1*C1*Cd1+
    Rd2*Cd2*Rc1*C1*Ld1*Cd1+ Rd2*Cd2*Lc1*C1*Rd1*Cd1+ Rd2*Cd2*L1*Rc1*C1*Cd1+
    Rd2*Cd2*L1*Rd1*C1*Cd1+ Rd2*Cd2*Rl*Lc1*C1*Cd1+ Rd2*Cd2*Rl*Ld1*C1*Cd1+
    Rc2*C2*Rc1*C1*Ld1*Cd1+ Rc2*C2*Lc1*C1*Rd1*Cd1+ Rc2*C2*L1*Rc1*C1*Cd1+
    Rc2*C2*L1*Rd1*C1*Cd1+ Rc2*C2*Rl*Ld1*C1*Cd1+ Rc2*C2*Ld2*Cd2*Rd1*Cd1+
    Rc2*C2*Ld2*Cd2*Rc1*C1+ Rc2*C2*Ld2*Cd2*Rl*Cd1+ Rc2*C2*Ld2*Cd2*Rl*C1+
    Lc2*C2*Rd2*Cd2*Rd1*Cd1+ Lc2*C2*Rd2*Cd2*Rc1*C1+ Lc2*C2*Rd2*Cd2*Rl*Cd1+
    Lc2*C2*Rd2*Cd2*Rl*C1+ L1*Lc2*C2*Cd2+ L1*Ld2*C2*Cd2+
    Cd1*Rd1*L1*Rc2*C2*Cd2+ Cd1*Rd1*L1*Rd2*C2*Cd2+ Cd1*Rd1*Rl*Lc2*C2*Cd2+
    Cd1*Rd1*Rl*Ld2*C2*Cd2+ Cd1*Ld1*L1*Cd2+ Cd1*Ld1*L1*C2+
    Cd1*Ld1*Rl*Rc2*C2*Cd2+ Cd1*Ld1*Rl*Rd2*C2*Cd2+ C1*Rc1*L1*Rc2*C2*Cd2+
    C1*Rc1*L1*Rd2*C2*Cd2+ C1*Rc1*Rl*Lc2*C2*Cd2+ C1*Rc1*Rl*Ld2*C2*Cd2+
    C1*Cd1*Rc1*Rd1*L1*Cd2+ C1*Cd1*Rc1*Rd1*L1*C2+
    C1*Cd1*Rc1*Rd1*Rl*Rc2*C2*Cd2+ C1*Cd1*Rc1*Rd1*Rl*Rd2*C2*Cd2+
    C1*Cd1*Rc1*Ld1*Rl*Cd2+ C1*Cd1*Rc1*Ld1*Rl*C2+ C1*Lc1*L1*Cd2+
```

C1*Lc1*L1*C2+ C1*Lc1*Rl*Rc2*C2*Cd2+ C1*Lc1*Rl*Rd2*C2*Cd2+
C1*Cd1*Lc1*Rd1*Rl*Cd2+ C1*Cd1*Lc1*Rd1*Rl*C2+Rc2*C2*Rl*Lc1*C1*Cd1;
den5 = L2*C2*Rc1*C1*Cd1*L1+ L2*C2*Rd1*C1*Cd1*L1+ L2*C2*Rd2*Cd2*Cd1*L1+
L2*C2*Rd2*Cd2*C1*L1+ L2*C2*Lc1*C1*Cd1*Rl+ L2*C2*Ld1*C1*Cd1*Rl+
L2*C2*Rd2*Cd2*Rc1*C1*Cd1*Rl+ L2*C2*Rd2*Cd2*Rd1*C1*Cd1*Rl+
L2*C2*Ld2*Cd2*Cd1*Rl+ L2*C2*Ld2*Cd2*C1*Rl+ L2*Cd1*L1*Rc2*C2*Cd2+
L2*Cd1*Rl*Lc2*C2*Cd2+ L2*Rc1*C1*Cd1*Rl*Rc2*C2*Cd2+
L2*Rc1*C1*Cd1*L1*Cd2+ L2*Lc1*C1*Cd1*Rl*Cd2+ L2*C1*L1*Rc2*C2*Cd2+
L2*C1*Rd1*Cd1*Rl*Rc2*C2*Cd2+ L2*C1*Rl*Lc2*C2*Cd2+ L2*C1*Rd1*Cd1*L1*Cd2+
L2*C1*Ld1*Cd1*Rl*Cd2+ L2*Rd1*Cd1*Lc2*C2*Cd2+ L2*Rd1*Cd1*Ld2*C2*Cd2+
L2*Lc1*C1*Rc2*C2*Cd2+ L2*Lc1*C1*Rd2*C2*Cd2+ L2*Ld1*Cd1*Rc2*C2*Cd2+
L2*Ld1*Cd1*Rd2*C2*Cd2+ L2*Rc1*C1*Lc2*C2*Cd2+ L2*Rc1*C1*Ld2*C2*Cd2+
L2*Rc1*C1*Rd1*Cd1*Rc2*C2*Cd2+ L2*Rc1*C1*Rd1*Cd1*Rd2*C2*Cd2+
L2*Rc1*C1*Ld1*Cd1*Cd2+ L2*Rc1*C1*Ld1*Cd1*C2+ L2*Lc1*C1*Rd1*Cd1*Cd2+
L2*Lc1*C1*Rd1*Cd1*C2+ Lc2*C2*Rc1*C1*Ld1*Cd1+ Lc2*C2*Lc1*C1*Rd1*Cd1+
Lc2*C2*L1*Rc1*C1*Cd1+ Lc2*C2*L1*Rd1*C1*Cd1+ Lc2*C2*Rl*Lc1*C1*Cd1+
Lc2*C2*Rl*Ld1*C1*Cd1+ Ld2*Cd2*Rc1*C1*Ld1*Cd1+ Ld2*Cd2*Lc1*C1*Rd1*Cd1+
Ld2*Cd2*L1*Rc1*C1*Cd1+ Ld2*Cd2*L1*Rd1*C1*Cd1+ Ld2*Cd2*Rl*Lc1*C1*Cd1+
Ld2*Cd2*Rl*Ld1*C1*Cd1+ Rc2*C2*Rd2*Cd2*Rc1*C1*Ld1*Cd1+
Rc2*C2*Rd2*Cd2*Lc1*C1*Rd1*Cd1+ Rc2*C2*Rd2*Cd2*L1*Rc1*C1*Cd1+
Rc2*C2*Rd2*Cd2*L1*Rd1*C1*Cd1+ Rc2*C2*Rd2*Cd2*Rl*Lc1*C1*Cd1+
Rc2*C2*Rd2*Cd2*Rl*Ld1*C1*Cd1+ Rd2*Cd2*Lc1*C1*Ld1*Cd1+
Rd2*Cd2*L1*Lc1*C1*Cd1+ Rd2*Cd2*L1*Ld1*C1*Cd1+ Rc2*C2*Lc1*C1*Ld1*Cd1+
Rc2*C2*L1*Lc1*C1*Cd1+ Rc2*C2*L1*Ld1*C1*Cd1+ Rc2*C2*Ld2*Cd2*Ld1*Cd1+
Rc2*C2*Ld2*Cd2*Rc1*C1*Rd1*Cd1+ Rc2*C2*Ld2*Cd2*Lc1*C1+
Rc2*C2*Ld2*Cd2*L1*Cd1+ Rc2*C2*Ld2*Cd2*L1*C1+
Rc2*C2*Ld2*Cd2*Rl*Rc1*C1*Cd1+ Rc2*C2*Ld2*Cd2*Rl*Rd1*C1*Cd1+
Lc2*C2*Rd2*Cd2*Ld1*Cd1+ Lc2*C2*Rd2*Cd2*Rc1*C1*Rd1*Cd1+
Lc2*C2*Rd2*Cd2*Lc1*C1+ Lc2*C2*Rd2*Cd2*L1*Cd1+ Lc2*C2*Rd2*Cd2*L1*C1+
Lc2*C2*Rd2*Cd2*Rl*Rc1*C1*Cd1+ Lc2*C2*Rd2*Cd2*Rl*Rd1*C1*Cd1+
Lc2*C2*Ld2*Cd2*Rd1*Cd1+ Lc2*C2*Ld2*Cd2*Rc1*C1+ Lc2*C2*Ld2*Cd2*Rl*Cd1+
Lc2*C2*Ld2*Cd2*Rl*C1+ Cd1*Rd1*L1*Lc2*C2*Cd2+ Cd1*Rd1*L1*Ld2*C2*Cd2+
Cd1*Ld1*L1*Rc2*C2*Cd2+ Cd1*Ld1*L1*Rd2*C2*Cd2+
C1*Cd1*Rc1*Rd1*L1*Rc2*C2*Cd2+ C1*Cd1*Rc1*Rd1*L1*Rd2*C2*Cd2+
C1*Cd1*Rc1*Rd1*Rl*Lc2*C2*Cd2+ Cd1*Ld1*Rl*Ld2*C2*Cd2+
C1*Cd1*Rc1*Rd1*Rl*Ld2*C2*Cd2+ C1*Cd1*Rc1*Ld1*L1*Cd2+
C1*Cd1*Rc1*Ld1*L1*C2+ C1*Cd1*Rc1*Ld1*Rl*Rc2*C2*Cd2+
C1*Cd1*Rc1*Ld1*Rl*Rd2*C2*Cd2+ C1*Lc1*L1*Rc2*C2*Cd2+
C1*Lc1*L1*Rd2*C2*Cd2+ C1*Lc1*Rl*Lc2*C2*Cd2+ C1*Lc1*Rl*Ld2*C2*Cd2+
C1*Cd1*Lc1*Rd1*L1*Cd2+ C1*Cd1*Lc1*Rd1*L1*C2+
C1*Cd1*Lc1*Rd1*Rl*Rc2*C2*Cd2+ C1*Cd1*Lc1*Rd1*Rl*Rd2*C2*Cd2+
C1*Cd1*Lc1*Ld1*Rl*Cd2+ C1*Cd1*Lc1*Ld1*Rl*C2+ Cd1*Ld1*Rl*Lc2*C2*Cd2+
C1*Rc1*L1*Lc2*C2*Cd2+ C1*Rc1*L1*Ld2*C2*Cd2;
den6 = L2*C2*Lc1*C1*Cd1*L1+ L2*C2*Ld1*C1*Cd1*L1+
L2*C2*Rd2*Cd2*Rc1*C1*Cd1*L1+ L2*C2*Rd2*Cd2*Rd1*C1*Cd1*L1+
L2*C2*Ld2*Cd2*Cd1*L1+ L2*C2*Ld2*Cd2*C1*L1+
L2*C2*Ld2*Cd2*Rc1*C1*Cd1*Rl+ L2*C2*Rd2*Cd2*Lc1*C1*Cd1*Rl+
L2*C2*Rd2*Cd2*Ld1*C1*Cd1*Rl+ L2*C2*Ld2*Cd2*Rd1*C1*Cd1*Rl+
L2*Cd1*L1*Lc2*C2*Cd2+ L2*Rc1*C1*Cd1*Rl*Lc2*C2*Cd2+
L2*Rc1*C1*Cd1*L1*Rc2*C2*Cd2+ L2*Lc1*C1*Cd1*L1*Cd2+
L2*Lc1*C1*Cd1*Rl*Rc2*C2*Cd2+ L2*C1*L1*Lc2*C2*Cd2+
L2*C1*Rd1*Cd1*Rl*Lc2*C2*Cd2+ L2*C1*Rd1*Cd1*L1*Rc2*C2*Cd2+
L2*C1*Ld1*Cd1*L1*Cd2+ L2*C1*Ld1*Cd1*Rl*Rc2*C2*Cd2+
L2*Lc1*C1*Lc2*C2*Cd2+ L2*Lc1*C1*Ld2*C2*Cd2+
L2*Lc1*C1*Rd1*Cd1*Rc2*C2*Cd2+ L2*Lc1*C1*Rd1*Cd1*Rd2*C2*Cd2+
L2*Ld1*Cd1*Lc2*C2*Cd2+ L2*Ld1*Cd1*Ld2*C2*Cd2+

```
      L2*Rc1*C1*Rd1*Cd1*Lc2*C2*Cd2+ L2*Rc1*C1*Rd1*Cd1*Ld2*C2*Cd2+
      L2*Rc1*C1*Ld1*Cd1*Rc2*C2*Cd2+ L2*Rc1*C1*Ld1*Cd1*Rd2*C2*Cd2+
      L2*Lc1*C1*Ld1*Cd1*C2+ L2*Lc1*C1*Ld1*Cd1*Cd2+ Lc2*C2*Lc1*C1*Ld1*Cd1+
      Lc2*C2*L1*Lc1*C1*Cd1+ Lc2*C2*L1*Ld1*C1*Cd1+ Ld2*Cd2*Lc1*C1*Ld1*Cd1+
      Ld2*Cd2*L1*Lc1*C1*Cd1+ Ld2*Cd2*L1*Ld1*C1*Cd1+
      Rc2*C2*Rd2*Cd2*Lc1*C1*Ld1*Cd1+ Rc2*C2*Rd2*Cd2*L1*Lc1*C1*Cd1+
      Rc2*C2*Rd2*Cd2*L1*Ld1*C1*Cd1+ Rc2*C2*Ld2*Cd2*Rc1*C1*Ld1*Cd1+
      Rc2*C2*Ld2*Cd2*Lc1*C1*Rd1*Cd1+ Rc2*C2*Ld2*Cd2*L1*Rc1*C1*Cd1+
      Rc2*C2*Ld2*Cd2*L1*Rd1*C1*Cd1+ Rc2*C2*Ld2*Cd2*Rl*Lc1*C1*Cd1+
      Rc2*C2*Ld2*Cd2*Rl*Ld1*C1*Cd1+ Lc2*C2*Rd2*Cd2*Rc1*C1*Ld1*Cd1+
      Lc2*C2*Rd2*Cd2*Lc1*C1*Rd1*Cd1+ Lc2*C2*Rd2*Cd2*L1*Rc1*C1*Cd1+
      Lc2*C2*Rd2*Cd2*L1*Rd1*C1*Cd1+ Lc2*C2*Rd2*Cd2*Rl*Lc1*C1*Cd1+
      Lc2*C2*Rd2*Cd2*Rl*Ld1*C1*Cd1+ Lc2*C2*Ld2*Cd2*Ld1*Cd1+
      Lc2*C2*Ld2*Cd2*Rc1*C1*Rd1*Cd1+ Lc2*C2*Ld2*Cd2*Lc1*C1+
      Lc2*C2*Ld2*Cd2*L1*Cd1+ Lc2*C2*Ld2*Cd2*L1*C1+
      Lc2*C2*Ld2*Cd2*Rl*Rc1*C1*Cd1+ Lc2*C2*Ld2*Cd2*Rl*Rd1*C1*Cd1+
      Cd1*Ld1*L1*Lc2*C2*Cd2+ Cd1*Ld1*L1*Ld2*C2*Cd2+
      C1*Cd1*Rc1*Rd1*L1*Lc2*C2*Cd2+ C1*Cd1*Rc1*Rd1*L1*Ld2*C2*Cd2+
      C1*Cd1*Rc1*Ld1*L1*Rc2*C2*Cd2+ C1*Cd1*Rc1*Ld1*L1*Rd2*C2*Cd2+
      C1*Cd1*Rc1*Ld1*Rl*Lc2*C2*Cd2+ C1*Cd1*Rc1*Ld1*Rl*Ld2*C2*Cd2+
      C1*Lc1*L1*Lc2*C2*Cd2+ C1*Lc1*L1*Ld2*C2*Cd2+
      C1*Cd1*Lc1*Rd1*L1*Rc2*C2*Cd2+ C1*Cd1*Lc1*Rd1*L1*Rd2*C2*Cd2+
      C1*Cd1*Lc1*Rd1*Rl*Lc2*C2*Cd2+ C1*Cd1*Lc1*Rd1*Rl*Ld2*C2*Cd2+
      C1*Cd1*Lc1*Ld1*L1*Cd2+ C1*Cd1*Lc1*Ld1*L1*C2+
      C1*Cd1*Lc1*Ld1*Rl*Rc2*C2*Cd2+ C1*Cd1*Lc1*Ld1*Rl*Rd2*C2*Cd2;
den7 = L2*C2*Rd2*Cd2*Lc1*C1*Cd1*L1+ L2*C2*Rd2*Cd2*Ld1*C1*Cd1*L1+
      L2*C2*Ld2*Cd2*Rc1*C1*Cd1*L1+ L2*C2*Ld2*Cd2*Rd1*C1*Cd1*L1+
      L2*C2*Ld2*Cd2*Lc1*C1*Cd1*Rl+ L2*C2*Ld2*Cd2*Ld1*C1*Cd1*Rl+
      L2*Rc1*C1*Cd1*L1*Lc2*C2*Cd2+ L2*Lc1*C1*Cd1*L1*Rc2*C2*Cd2+
      L2*Lc1*C1*Cd1*Rl*Lc2*C2*Cd2+ L2*C1*Rd1*Cd1*L1*Lc2*C2*Cd2+
      L2*C1*Ld1*Cd1*L1*Rc2*C2*Cd2+ L2*C1*Ld1*Cd1*Rl*Lc2*C2*Cd2+
      L2*Lc1*C1*Rd1*Cd1*Lc2*C2*Cd2+ L2*Rc1*C1*Ld1*Cd1*Lc2*C2*Cd2+
      L2*Rc1*C1*Ld1*Cd1*Ld2*C2*Cd2+ L2*Lc1*C1*Rd1*Cd1*Ld2*C2*Cd2+
      L2*Lc1*C1*Ld1*Cd1*Rc2*C2*Cd2+ L2*Lc1*C1*Ld1*Cd1*Rd2*C2*Cd2+
      Rc2*C2*Ld2*Cd2*Lc1*C1*Ld1*Cd1+ Rc2*C2*Ld2*Cd2*L1*Lc1*C1*Cd1+
      Rc2*C2*Ld2*Cd2*L1*Ld1*C1*Cd1+ Lc2*C2*Rd2*Cd2*Lc1*C1*Ld1*Cd1+
      Lc2*C2*Rd2*Cd2*L1*Lc1*C1*Cd1+ Lc2*C2*Rd2*Cd2*L1*Ld1*C1*Cd1+
      Lc2*C2*Ld2*Cd2*Rc1*C1*Ld1*Cd1+ Lc2*C2*Ld2*Cd2*Lc1*C1*Rd1*Cd1+
      Lc2*C2*Ld2*Cd2*L1*Rc1*C1*Cd1+ Lc2*C2*Ld2*Cd2*L1*Rd1*C1*Cd1+
      Lc2*C2*Ld2*Cd2*Rl*Lc1*C1*Cd1+ Lc2*C2*Ld2*Cd2*Rl*Ld1*C1*Cd1+
      C1*Cd1*Rc1*Ld1*L1*Lc2*C2*Cd2+ C1*Cd1*Lc1*Rd1*L1*Lc2*C2*Cd2+
      C1*Cd1*Lc1*Rd1*L1*Ld2*C2*Cd2+ C1*Cd1*Lc1*Ld1*L1*Rc2*C2*Cd2+
      C1*Cd1*Lc1*Ld1*L1*Rd2*C2*Cd2+ C1*Cd1*Lc1*Ld1*Rl*Lc2*C2*Cd2+
      C1*Cd1*Lc1*Ld1*Rl*Ld2*C2*Cd2+ C1*Cd1*Rc1*Ld1*L1*Ld2*C2*Cd2;
den8 = L2*C2*Ld2*Cd2*Ld1*C1*Cd1*L1+ L2*C2*Ld2*Cd2*Lc1*C1*Cd1*L1+
      L2*Lc1*C1*Cd1*L1*Lc2*C2*Cd2+ L2*C1*Ld1*Cd1*L1*Lc2*C2*Cd2+
      L2*Lc1*C1*Ld1*Cd1*Lc2*C2*Cd2+ L2*Lc1*C1*Ld1*Cd1*Ld2*C2*Cd2+
      Lc2*C2*Ld2*Cd2*Lc1*C1*Ld1*Cd1+ Lc2*C2*Ld2*Cd2*L1*Lc1*C1*Cd1+
      Lc2*C2*Ld2*Cd2*L1*Ld1*C1*Cd1+ C1*Cd1*Lc1*Ld1*L1*Lc2*C2*Cd2+
      C1*Cd1*Lc1*Ld1*L1*Ld2*C2*Cd2;
//long equation huh?
for (i=1;i<50;i++)
{
        freqs = Freq*i*CellNum;
        numreal = num0-pow(2*PI*freqs,2)*num2+pow(2*PI*freqs,4)*num4-
            pow(2*PI*freqs,6)* num6+pow(2*PI*freqs,8)*num8;
```

```
numimag = 2*PI*freqs*num1-pow(2*PI*freqs,3)*num3+pow(2*PI*freqs,5)*num5-
    pow(2*PI*freqs,7)*num7;
nummag = sqrt(numreal*numreal+numimag*numimag);

denreal = den0-pow(2*PI*freqs,2)*den2+pow(2*PI*freqs,4)*den4-pow(2*PI*freqs,6)*
    den6+pow(2*PI*freqs,8)*den8;
denimag = 2*PI*freqs*den1-pow(2*PI*freqs,3)*den3+pow(2*PI*freqs,5)*den5-
     pow(2*PI*freqs,7)*den7;
denmag = sqrt(denreal*denreal+denimag*denimag);
oldgain = gain;
gain = nummag/denmag;
if ((gain > oldgain) && (freqs > 800000))
        freqmingain = freqs;
dB = 20*log10(gain*WCHarmonics[i]*1000000);
while ((freqs > m_EMILimitsSet.m_Frequency) && (!m_EMILimitsSet.IsEOF()))
{
 OldFreq = m_EMILimitsSet.m_Frequency;
 OldLimit = m_EMILimitsSet.m_Limit;
 if (!m_EMILimitsSet.IsEOF())
 {
        m_EMILimitsSet.MoveNext();
        m_EMILimitsSet.GetStatus(rStatus);
        if (rStatus.m_lCurrentRecord == 0)
          EMI = 100000; //no limit so use a high number
        else
          EMI = (OldLimit - (OldLimit-m_EMILimitsSet.m_Limit)*
            (freqs-OldFreq)/(m_EMILimitsSet.m_Frequency-OldFreq));
 }
};
if (freqs < m_EMILimitsSet.m_Frequency)
{
        m_EMILimitsSet.GetStatus(rStatus);
        if (rStatus.m_lCurrentRecord == 0)
                EMI = 100000; //no limit so use a high number
        else
                EMI = (OldLimit - (OldLimit-m_EMILimitsSet.m_Limit)*
                  (freqs-OldFreq)/(m_EMILimitsSet.m_Frequency-OldFreq));
}
else EMI = OldLimit;
if (dB > EMI+1)
{
        if (once1 == SET)
                once1 = USED;
        if (once2 == SET)
                once2 = USED;
        if ((gain > oldgain) && (freqs < freqmingain))
        {
                if (Rd1 > Rd2)
                        Rd1 -= Rd1*.1;
                else Rd2 -= Rd2*.1;
                DesignOK = FALSE;
                break;
        }
        if (freqs > 1000000)
        {
                if (L1 > L2)
```

```
                                                L1 +=L1*.05;
                                        else L2 += L2*.5;
                                        DesignOK = FALSE;
                                        break;
                                }
                                if (freqs < 1000000)
                                {
                                        L2 += L2*.05;
                                        DesignOK = FALSE;
                                        break;
                                }
                        }
                        if ((i == FirstHarmonicLimit) && (once1 != USED) && ((dB +2) < EMI))
                        {
                                once1 = SET;
                                L2 -= L2*.03;
                                DesignOK = FALSE;
                                break;
                        }
                        if ((freqs == freqmingain) && (once2 != USED) && ((dB + 10)<EMI))
                        {
                                if (L1 > .0000005)
                                {
                                 once2 = SET;
                                 L1 -= L1*.03;
                                 DesignOK = FALSE;
                                 break;
                                }
                        }
                }
        }
Rdd1 = Rd1;
Rdd2 = Rd2;
return TRUE;
}
#pragma optimize("", on)
```

/*///

bool CTIMCADDoc::DesignFilter3(double C2, double Rc2, double Lc2, double& L2, double Freq,
int CellNum, double WCHarmonics[ ])

DesignFilter3 designs the third type of filter ensuring that the EMI limits are met.

Parameters:

C2 : The capacitor closer to the converter
Rc2 : The ESR for the C2 capacitor
Lc2 : The ESL for the C2 capacitor
L2 : The inductance of the filter
Freq : The switching frequency of the converter
CellNum : Number of cells imployed by the converter
WCHarmonics[ ] : A vector of the values each representing the
worst case harmonics of a signal, values are dc, freq*CellNum,
2*freq*CellNum, etc...

Output

```
True if no errors occured
*///////////////////////////////////////////////////////////////////////
bool CTIMCADDoc::DesignFilter3(double C2, double Rc2, double Lc2, double& L2, double Freq,
                int CellNum, double WCHarmonics[])
{

                double  gain, //The gain of the filter at a given frequency
                        dB, //The magnitude of the worst case harmonic in dB
                        EMI = 66, //The EMI limit at a given frequency
                        num0, num1, num2, den0, den1, den2, //Terms in the transfer function
                        Rl; //Resistive values in the filter
                bool    DesignOK = FALSE, //True if the design is valid
                        OneWay = TRUE, //Insures that the while loop is not infinite
                        NoInductor; //True is filter is designed without the inductor
                double  numreal, numimag, nummag, denreal, denimag, denmag, //Terms used in the
                                //transfer function
                        OldLimit = 100000, //Previous EMI limit
                        OldFreq = 0, //Frequency where old EMI limit started
                        freqs, //The frequency of intrest
                        LOWL2, HIGHL2; //Constants for the min and max L2 value
                int     i, //Counting index
                        once1, once2, //Insure that the while loop in not infinite
                        FirstHarmonicLimit; //The first harmonic that the EMI limits apply to
                CRecordsetStatus        rStatus; //The status of the EMI database

LOWL2 = 5e-8;
HIGHL2 = .0001;
NoInductor = FALSE;
once1 = once2 = SET;
L2 = .00001;
Rl = 50;
OneWay = TRUE;
while (!DesignOK)
{

        if (L2 > HIGHL2)
                return FALSE;
        DesignOK = TRUE;
        m_EMILimitsSet.MoveFirst();
        FirstHarmonicLimit = (int) ceil(m_EMILimitsSet.m_Frequency/(Freq*CellNum));
        num0 = 1;
        num1 = Rc2*C2;
        num2 = Lc2*C2;

        den0 = 1;
        den1 = (Rl+Rc2)*C2;
        den2 = (L2+Lc2)*C2;

        for (i=1;i<50;i++)
        {
                freqs = Freq*i*CellNum;

                numreal = num0-pow(2*PI*freqs,2)*num2;
                numimag = 2*PI*freqs*num1;
                nummag = sqrt(numreal*numreal+numimag*numimag);
```

```
denreal = den0-pow(2*PI*freqs,2)*den2;
denimag = 2*PI*freqs*den1;
denmag = sqrt(denreal*denreal+denimag*denimag);
gain = nummag/denmag;

dB = 20*log10(gain*WCHarmonics[i]*1000000);
while ((freqs > m_EMILimitsSet.m_Frequency) && (!m_EMILimitsSet.IsEOF()))
{
 OldFreq = m_EMILimitsSet.m_Frequency;
 OldLimit = m_EMILimitsSet.m_Limit;
 if (!m_EMILimitsSet.IsEOF())
 {
        m_EMILimitsSet.MoveNext();
        m_EMILimitsSet.GetStatus(rStatus);
        if (rStatus.m_lCurrentRecord == 0)
          EMI = 100000; //no limit so use a high number
        else
          EMI = (OldLimit - (OldLimit-m_EMILimitsSet.m_Limit)*
            (freqs-OldFreq)/(m_EMILimitsSet.m_Frequency-OldFreq));
 }
};
if (freqs < m_EMILimitsSet.m_Frequency)
{
        m_EMILimitsSet.GetStatus(rStatus);
        if (rStatus.m_lCurrentRecord == 0)
                EMI = 100000; //no limit so use a high number
        else
                EMI = (OldLimit - (OldLimit-m_EMILimitsSet.m_Limit)*
                  (freqs-OldFreq)/(m_EMILimitsSet.m_Frequency-OldFreq));
}
else EMI = OldLimit;

if (dB > EMI+1 && (NoInductor))
{
        L2 = LOWL2;
        DesignOK = FALSE;
        OneWay = FALSE;
        NoInductor = FALSE;
        break;
}
if (dB > EMI+1)
{
        L2 += L2*.05;
        DesignOK = FALSE;
        OneWay = FALSE;
        break;
}
if ((i == FirstHarmonicLimit) && (OneWay) && ((dB +2) < EMI) && (L2 > LOWL2))
{
        L2 -= L2*.03;
        DesignOK = FALSE;
        break;
}
if ((i == FirstHarmonicLimit) && (OneWay) && ((dB +2) < EMI) && (L2 < LOWL2)
        && (!NoInductor))
{
```

```
                        L2 = 0;
                        DesignOK = FALSE;
                        NoInductor = TRUE;
                        break;
                }
        }
}
return TRUE;

}

/*//////////////////////////////////////////////////////////////////////
void CSchottkySet::PowLoss(double waveformS2[ ], double VinMax, double freq, double& Pcond,
    double& Pcap, double& ThermalResistance)

PowLoss determines the maximum power loss in the Schottky Diode if used as the second switch in the
converter.

Parameters:
        waveformS2[ ] : The current through switch 2
        VinMax : The maximum voltage at the input
        freq : The switching frequency of the converter
        Pcond : The conduction losses
        Pcap : The energy associated with charging the capacitance across the diode
        ThermalResistance : The junction to case thermal resistance

*//////////////////////////////////////////////////////////////////////
void CSchottkySet::PowLoss(double waveformS2[], double VinMax, double freq, double& Pcond,
    double& Pcap, double& ThermalResistance)
{
                int     i, count; //Counting variables
                double  Loss, //Copnduction losses
                        Eon, //Turn on energy losses
                        t_reverse_on, //time the device is off
                        P_reverse, //Reverse current power loss
                        Kdiode; //A constant that is in F/(V)^1/2

Pcond = 0;
count = 0;
if (m_Detail)
{
        for(i = 0; i < 1200; i++)
        {
                if (waveformS2[i] == 0)
                        Loss = 0;
                else Loss = waveformS2[i]* (m_B2*waveformS2[i]* waveformS2[i]+
                                m_B1*waveformS2[i]+ m_B0);
                        Pcond = Pcond+Loss/1200;
        }
}
else
{
        for(i = 0; i < 1200; i++)
        {
                if (waveformS2[i] < 0)
                        Loss = 0;
```

```
                    else Loss = waveformS2[i]*(m_A1*waveformS2[i]+m_A0);
                    Pcond = Pcond+Loss/1200;
          }
}
for (i = 0; i < 1200; i++)
{
          if (waveformS2[i] == 0)
                    count++;
}

t_reverse_on = (double) count/1200;
P_reverse = m_Ir_Vmax*VinMax*t_reverse_on;

Pcond = Pcond + P_reverse;

Kdiode = m_Ct*sqrt(m_Vr_Ct);
Eon = 2*Kdiode*pow(VinMax,1.5)/3;
Pcap = Eon*freq;

ThermalResistance = m_Rjc;

return;
}

/*/////////////////////////////////////////////////////////////////////////////
void CDiodeSet::PowLoss(double Iinitial, double waveformS2[ ], double VinMax, double freq, double&
     Pcond, double& Psw, double& Pcap, double& ThermalResistance)

PowLoss determines the maximum power loss in the PiN Diode if used as the second switch in the
converter.

Parameters:
          Iinitial : The current in the diode when it turns off
          waveformS2[ ] : The current through switch 2
          VinMax : The maximum voltage at the input
          freq : The switching frequency of the converter
          Pcond : The conduction losses
          Psw : The switching losses
          Pcap : The energy associated with charging the capacitance across the diode
          ThermalResistance : The junction to case thermal resistance

*//////////////////////////////////////////////////////////////////////////////
void CDiodeSet::PowLoss(double Iinitial, double waveformS2[], double VinMax, double freq, double&
     Pcond, double& Psw, double& Pcap, double& ThermalResistance)
{
                    int       i, count = 0; //Counting variables
                    double    Loss, //Conduction losses
                              Eon, //Turn on losses
                              Kdiode, //Constant in the units of F/(V)^1/2
                              Qd, Qdo; //Charge within the diode

Pcond = 0;
if (m_Detail)
{
          for(i = 0; i < 1200; i++)
          {
```

```
                    if (waveformS2[i] == 0)
                              Loss = 0;
                    else Loss = waveformS2[i]* (m_B2*waveformS2[i]* waveformS2[i]+
                                        m_B1*waveformS2[i]+ m_B0);
                              Pcond = Pcond+Loss/1200;
          }
}
else
{
          for(i = 0; i < 1200; i++)
          {
                    if (waveformS2[i] < 0)
                              Loss = 0;
                    else Loss = waveformS2[i]*(m_A1*waveformS2[i]+m_A0);
                    Pcond = Pcond+Loss/1200;
          }
}
for (i = 0; i < 1200; i++)
{
          if (waveformS2[i] == 0)
                    count++;
}
Qdo = m_trr*m_trr*m_didt/2;
Qd = Qdo*(1+(Iinitial-m_Iftest)/(2*m_Iftest));
Psw = Qd*VinMax*freq;

Kdiode = m_Ct*sqrt(m_Vr_Ct);
Eon = 2*Kdiode*pow(VinMax,1.5)/3;
Pcap = Eon*freq;

ThermalResistance = m_Rjc;
return;
}


/*////////////////////////////////////////////////////////////////////////
double CFETSet::PowLoss(int FET,double Vin, double Ifinal, double Idrive, double freq, double Irms,
     double& Pcon, double& Psw, double& Pcap, double& ThermalResistance, double Iinitial, int Switch,
     double Temp)
```

PowLoss determines the maximum power loss in the FET if used as either switch in the converter.

Parameters:

FET : The index of the MOSFET in use
Vin : The maximum voltage at the input
Ifinal : The current when the MOSFET turns off
Idrive : The current used by the gate driver
freq : The switching frequency of the converter
Irms : The rms current through the MOSFET
Pcon : The conduction losses
Psw : The switching losses
Pcap : The energy associated with charging the capacitance across the MOSFET
ThermalResistance : The junction to case thermal resistance
Iinitial : The current in the MOSFET when it turns on
Switch : whether the MOSFET is used as switch 1 or switch 2
Temp : The ambient temperature

```
Output
The total power lost
*///////////////////////////////////////////////////////////////////////
double CFETSet::PowLoss(int FET,double Vin, double Ifinal, double Idrive,
          double freq, double Irms, double& Pcon, double& Psw, double& Pcap,
          double& ThermalResistance, double Iinitial, int Switch, double Temp)
{
                    double  Loss = 0, //The total power loss
                            Kgd, //The gate to drain capacitance per root voltage
                            Kds, //The drain to source capacitance per root voltage
                            Ecb, //The energy loss for the c-b transition
                            Cgs, //The gate to source capacitance
                            sqr, //A constant
                            Qdo, Qd, //The charge in the device when acting like a diode
                            VsatOff, //Turn off saturation voltage
                            VsatOn, //Turn on saturation voltage
                            T1, T2, //Time intervals
                            Tdc, //Time interval for the d-c transition
                            Edc, //Energy loss during the d-c transition
                            Etot, //Total energy loss
                            P2, P3, //power loss during the 1 and 2 time intervals
                            Prr; //Reverse recovery power loss

//Gets the data for the current FET
SetAbsolutePosition(FET);

//The capacitive turn-on losses and other constants
Kgd = m_Crss*sqrt(m_Vds_crss);
Kds = (m_Coss-m_Crss)*sqrt(m_Vds_crss);

Cgs = m_Qg/m_Vg_Qg;
sqr = sqrt(m_Ls*Cgs);
VsatOff = fabs(Ifinal)/m_gfs+m_Vt;
VsatOn  = fabs(Iinitial)/m_gfs+m_Vt;

if (((Iinitial >= 0) && (Switch == 1)) || ((Iinitial < 0) && (Switch == 2)))
{
//turn on switching losses
          T2 = (VsatOn-m_Vt)*Cgs/Idrive;
          P2 = T2*Vin*fabs(Iinitial)*freq/2;
          P3 = freq*fabs(Iinitial)*Kgd*2*pow(Vin,3/2)/(Idrive*3);
//the engery to charge the capacitances across the gate turn on loss
//          Eo = 2*(Kds+Kgd)*Vin*sqrt(Vin)/3; removed since it is accounted already

}
else
          P2 = P3 = 0;

if (((Ifinal >= 0) && (Switch == 1)) || ((Ifinal < 0) && (Switch == 2)))
{
//The first section of the turn off losses ie charging the Miller Capacitance
          Ecb = 2*fabs(Ifinal)*Kgd*Vin*sqrt(Vin)/(3*Idrive);
//turn off switching losses
          T1 = sqr*atan(sqrt(VsatOff*VsatOff-m_Vt*m_Vt)/m_Vt);
          if (VsatOff*sqrt(Cgs/m_Ls) < Idrive)
```

```
				Tdc = T1;
			else
			{
				T2 = sqr*asin(Idrive*sqrt(m_Ls/Cgs)/VsatOff);
				if (T2 > T1)
					Tdc = T1;
				else Tdc = T2 + Cgs*(VsatOff*cos(T2/sqr)-m_Vt)/Idrive;
			}
			Edc = Tdc*Vin*fabs(Ifinal)/2;
		}
		else
			Ecb = Edc = 0;

		if (((Ifinal < 0) && (Switch == 1)) || ((Ifinal >= 0) && (Switch == 2)))
		{
			Pcap = freq*2*Kds*Vin*sqrt(Vin)/3;
			Qdo = m_trr*m_trr*m_didt/2;
			Qd = Qdo*(1+(fabs(Ifinal)-m_Iftest)/(2*m_Iftest));
			Prr = Qd*Vin*freq;
		}
		else Pcap = Prr = 0;

		Etot = Ecb + Edc;
		Psw = Etot*freq + P2 + P3 + Prr;
		//conduction losses
		Pcon = Irms*Irms*(m_Ra*Temp*Temp+m_Rb*Temp+m_Rc);
		Loss = Psw+Pcon;
		ThermalResistance = m_ThermalResistance;

		return Loss;
		}
```

Appendix E TimCAD Flowcharts

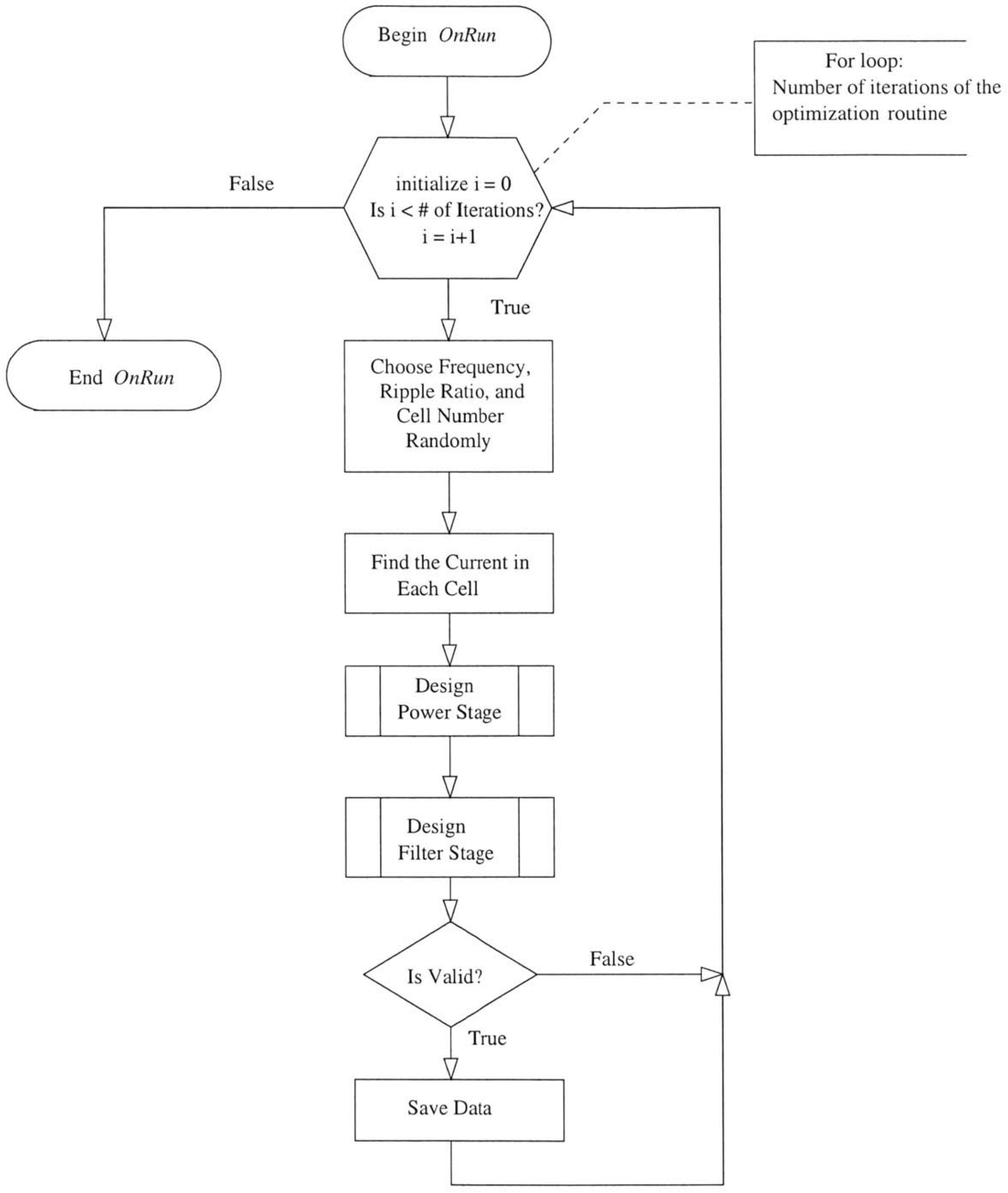


Figure E.1: Flowchart for the optimization routine

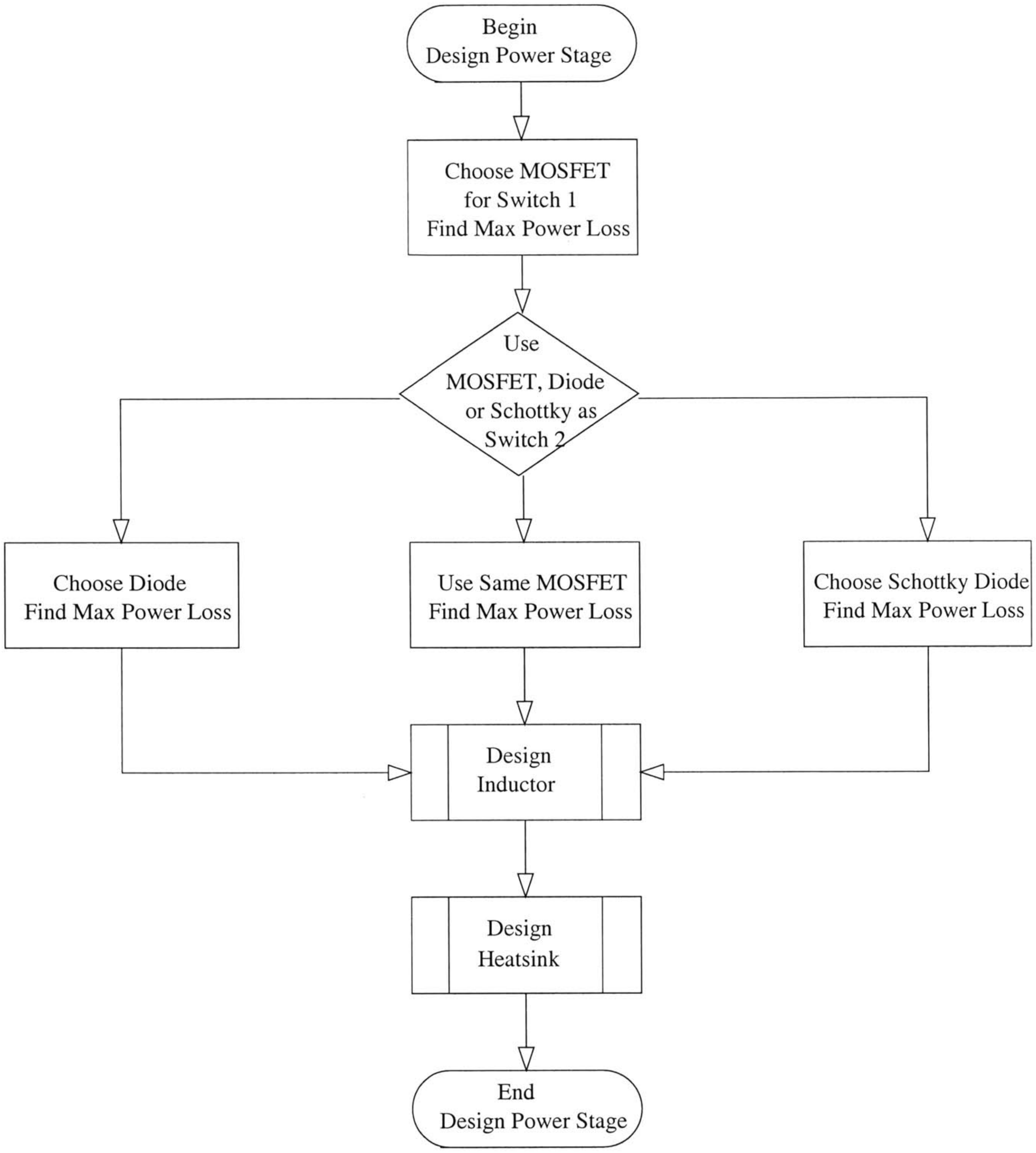


Figure E.2 Flowchart for the power stage design of the converter.

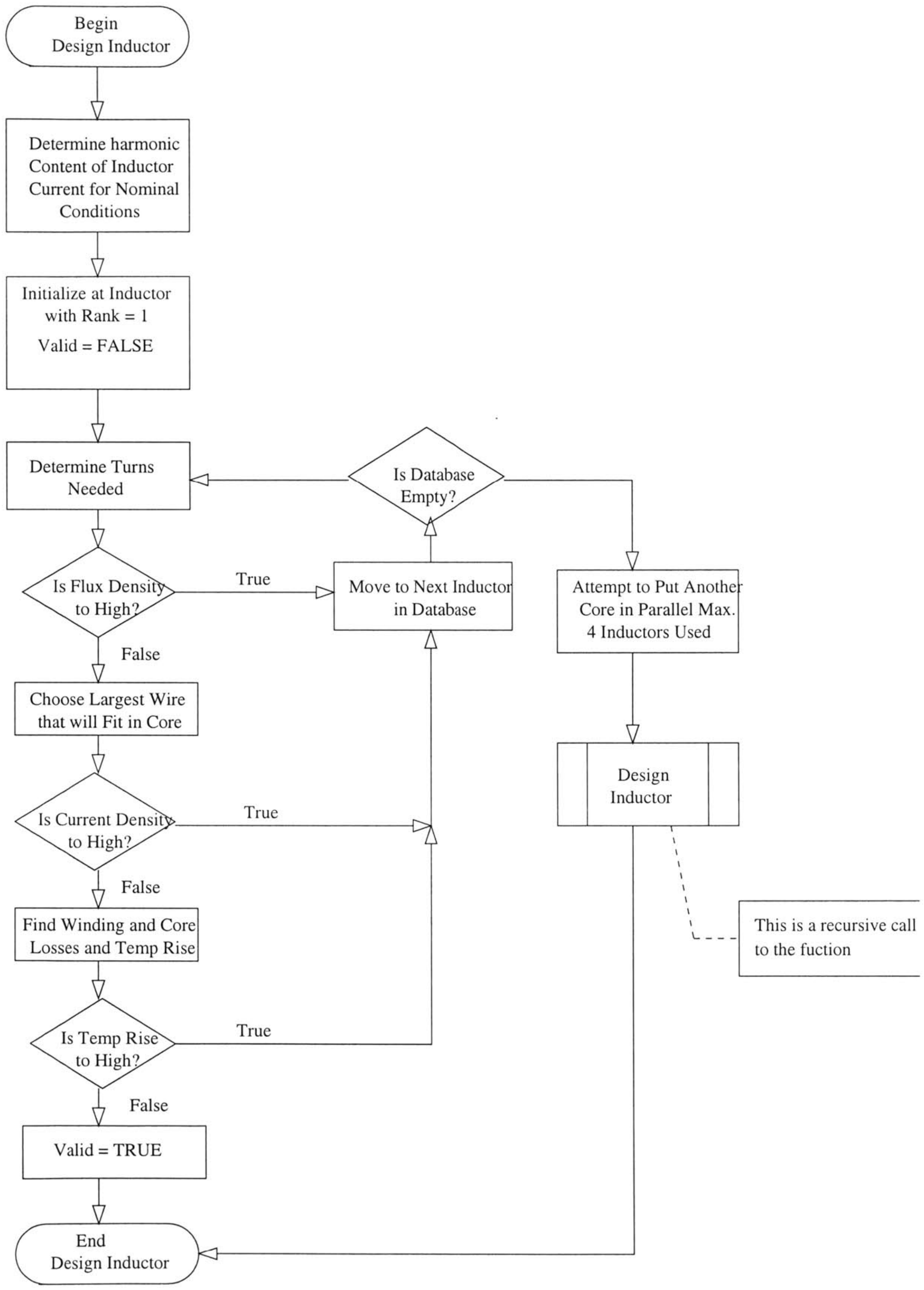

Begin Design Inductor
Determine harmonic Content of Inductor Current for Nominal Conditions
Initialize at Inductor with Rank = 1
Valid = FALSE
Determine Turns Needed
Is Database Empty?
Is Flux Density to High?
True
Move to Next Inductor in Database
Attempt to Put Another Core in Parallel Max. 4 Inductors Used
False
Choose Largest Wire that will Fit in Core
Design Inductor
Is Current Density to High?
True
False
This is a recursive call to the fuction
Find Winding and Core Losses and Temp Rise
Is Temp Rise to High?
True
False
Valid = TRUE
End Design Inductor

Figure E.3 Flowchart for the converter inductor.

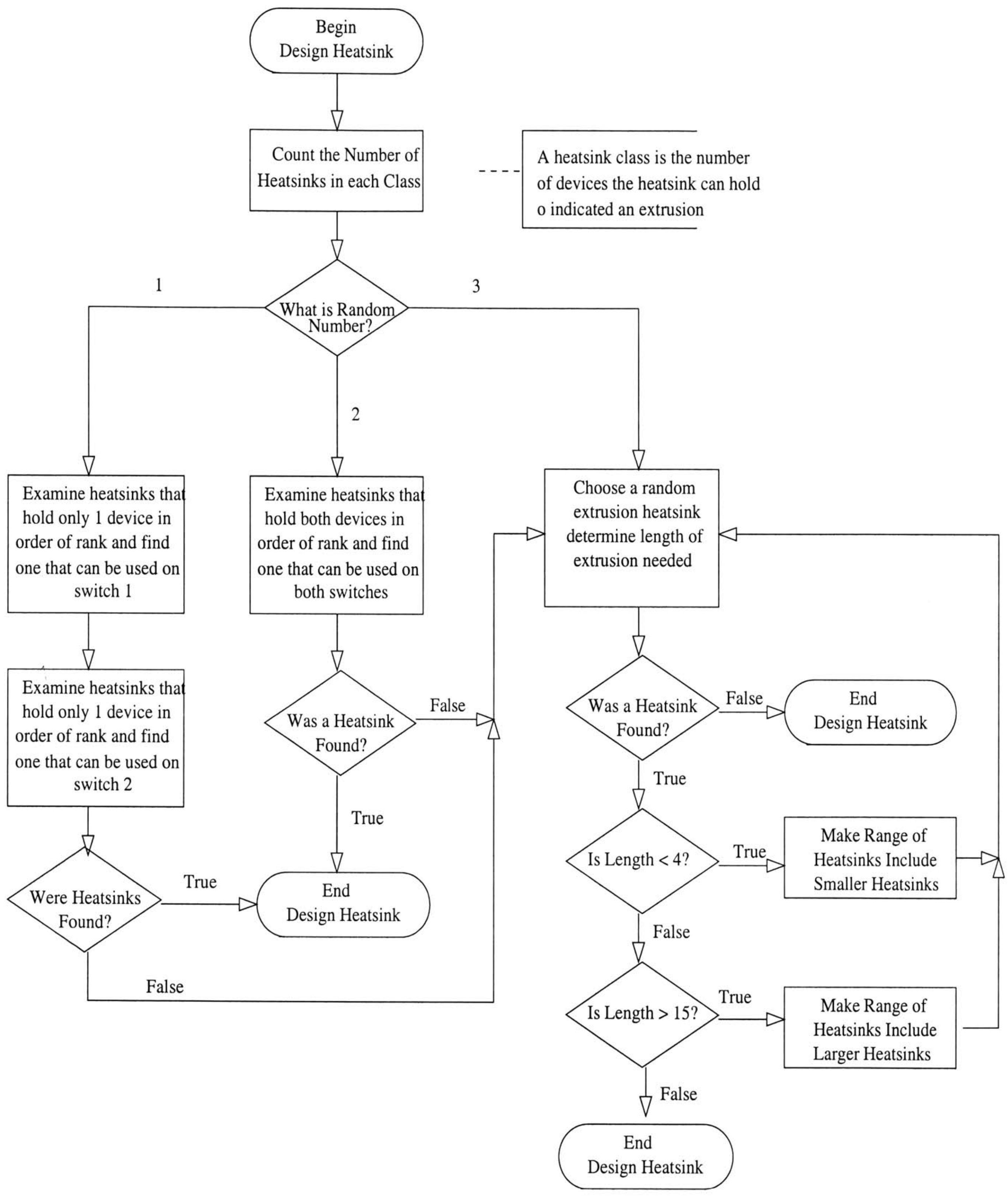
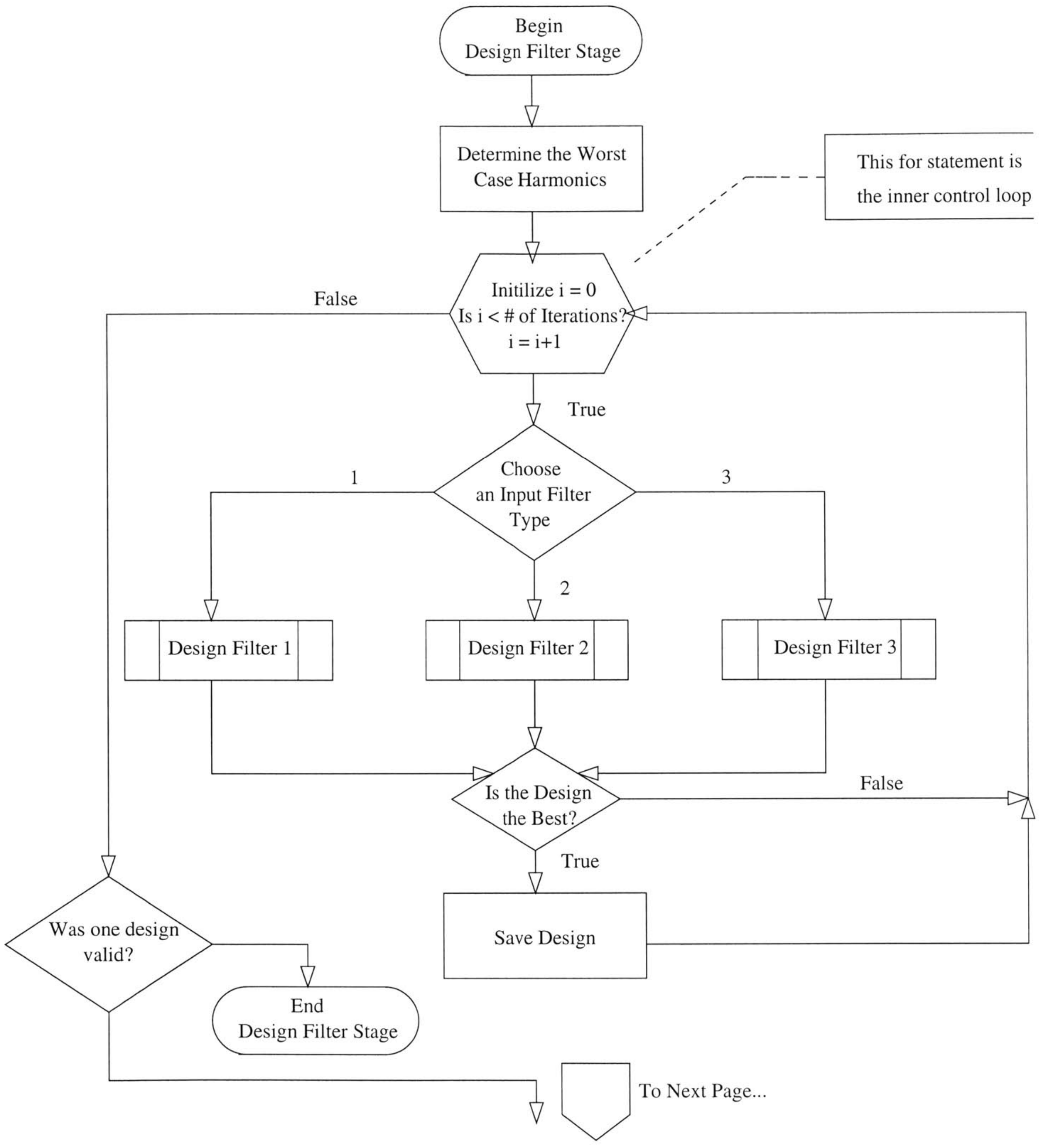

Begin
Design Heatsink
Count the Number of Heatsinks in each Class
A heatsink class is the number of devices the heatsink can hold o indicated an extrusion
What is Random Number?
1
2
3
Examine heatsinks that hold only 1 device in order of rank and find one that can be used on switch 1
Examine heatsinks that hold only 1 device in order of rank and find one that can be used on switch 2
Were Heatsinks Found?
True
False
Examine heatsinks that hold both devices in order of rank and find one that can be used on both switches
Was a Heatsink Found?
False
True
End
Design Heatsink
Choose a random extrusion heatsink determine length of extrusion needed
Was a Heatsink Found?
False
End
Design Heatsink
True
Is Length < 4?
True
Make Range of Heatsinks Include Smaller Heatsinks
False
Is Length > 15?
True
Make Range of Heatsinks Include Larger Heatsinks
False
End
Design Heatsink

Figure E.4 Flowchart for the heatsink design


Begin
Design Filter Stage
Determine the Worst Case Harmonics
This for statement is the inner control loop
Initilize i = 0
Is i < # of Iterations?
i = i+1
False
True
Choose an Input Filter Type
1
2
3
Design Filter 1
Design Filter 2
Design Filter 3
Is the Design the Best?
False
True
Save Design
Was one design valid?
End
Design Filter Stage
To Next Page...

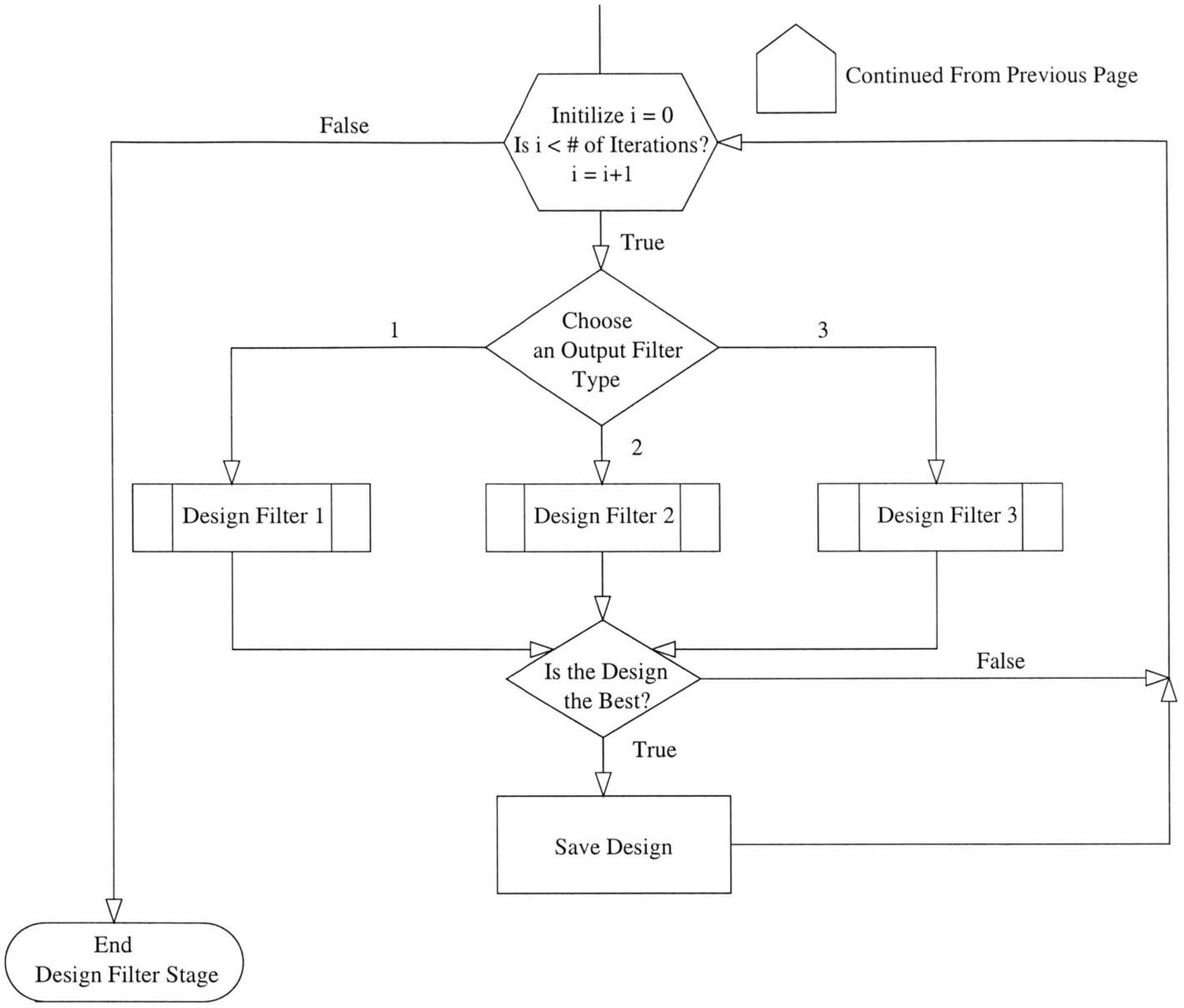


Figure E.5 Flowchart for the design of the input and output filters.

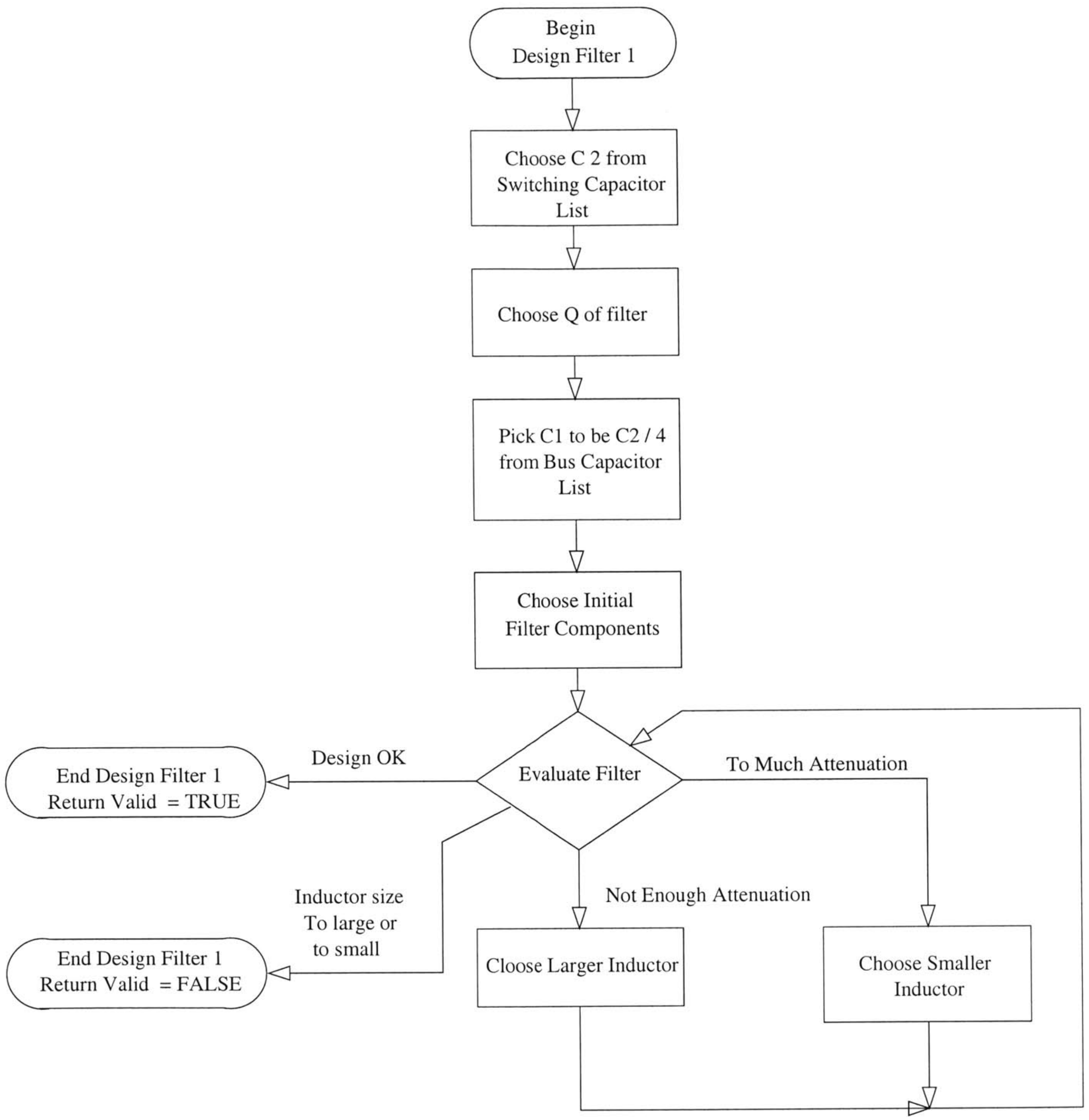


Figure E.6 Flowchart for the design of filter type 1

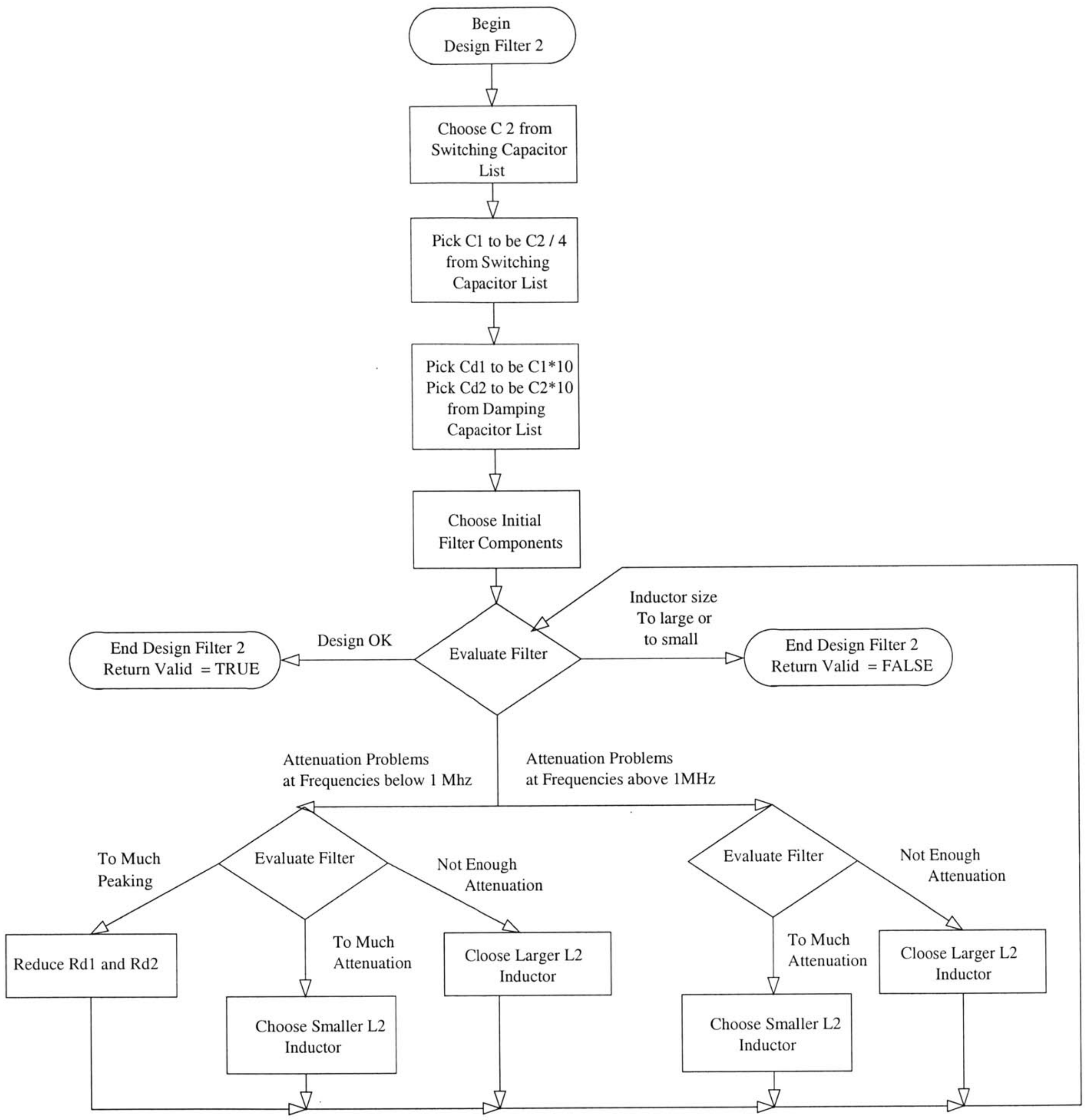

Begin
Design Filter 2
Choose C 2 from Switching Capacitor List
Pick C1 to be C2 / 4 from Switching Capacitor List
Pick Cd1 to be C1*10
Pick Cd2 to be C2*10
from Damping Capacitor List
Choose Initial Filter Components
Evaluate Filter
Design OK
End Design Filter 2
Return Valid = TRUE
Inductor size To large or to small
End Design Filter 2
Return Valid = FALSE
Attenuation Problems at Frequencies below 1 Mhz
Attenuation Problems at Frequencies above 1MHz
Evaluate Filter
To Much Peaking
Not Enough Attenuation
To Much Attenuation
Reduce Rd1 and Rd2
Choose Smaller L2 Inductor
Choose Larger L2 Inductor
Evaluate Filter
Not Enough Attenuation
To Much Attenuation
Choose Smaller L2 Inductor
Choose Larger L2 Inductor

Figure E.7 Flowchart for the design of filter type 3

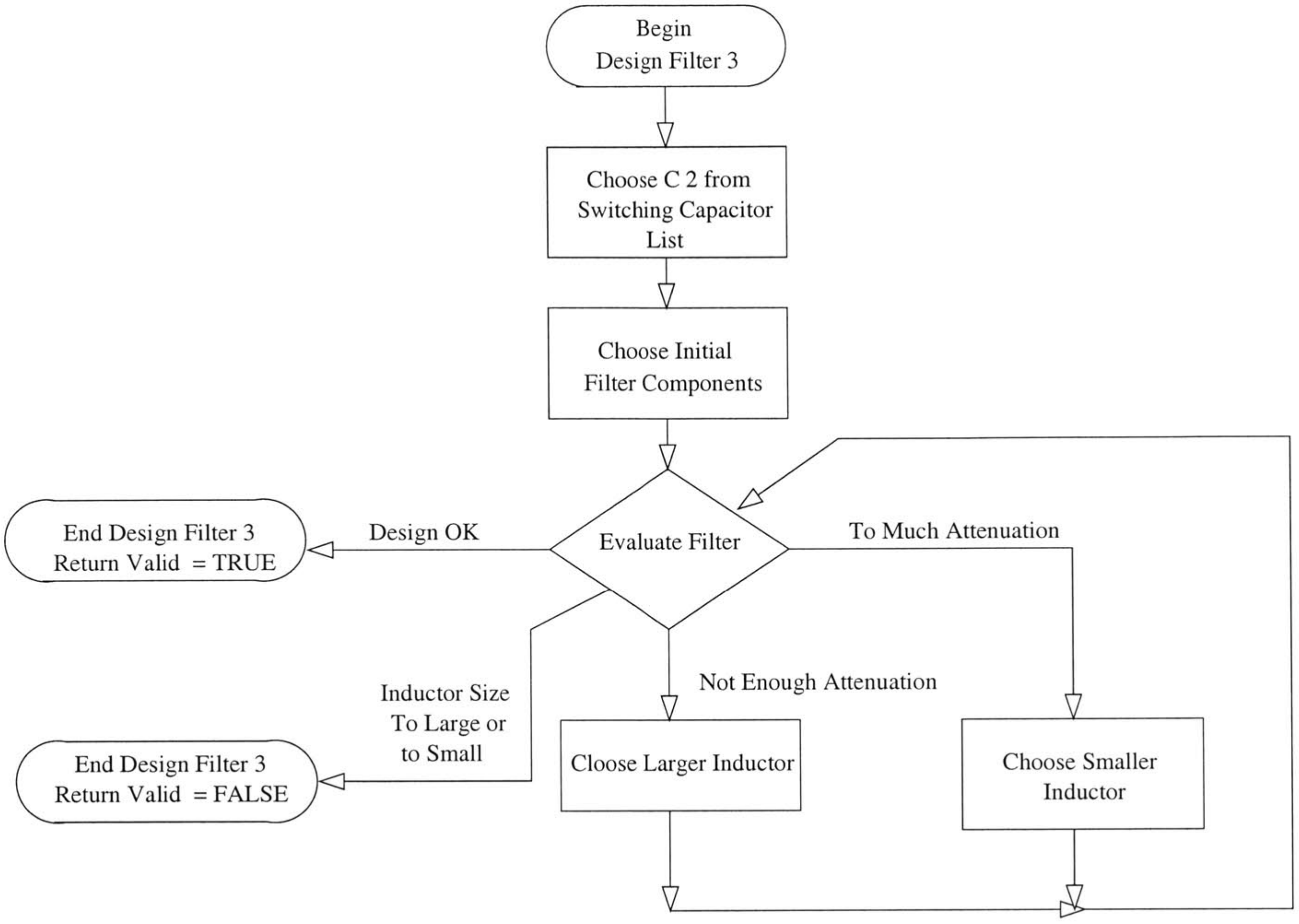


Figure E.8 Flowchart for the design of filter type 3

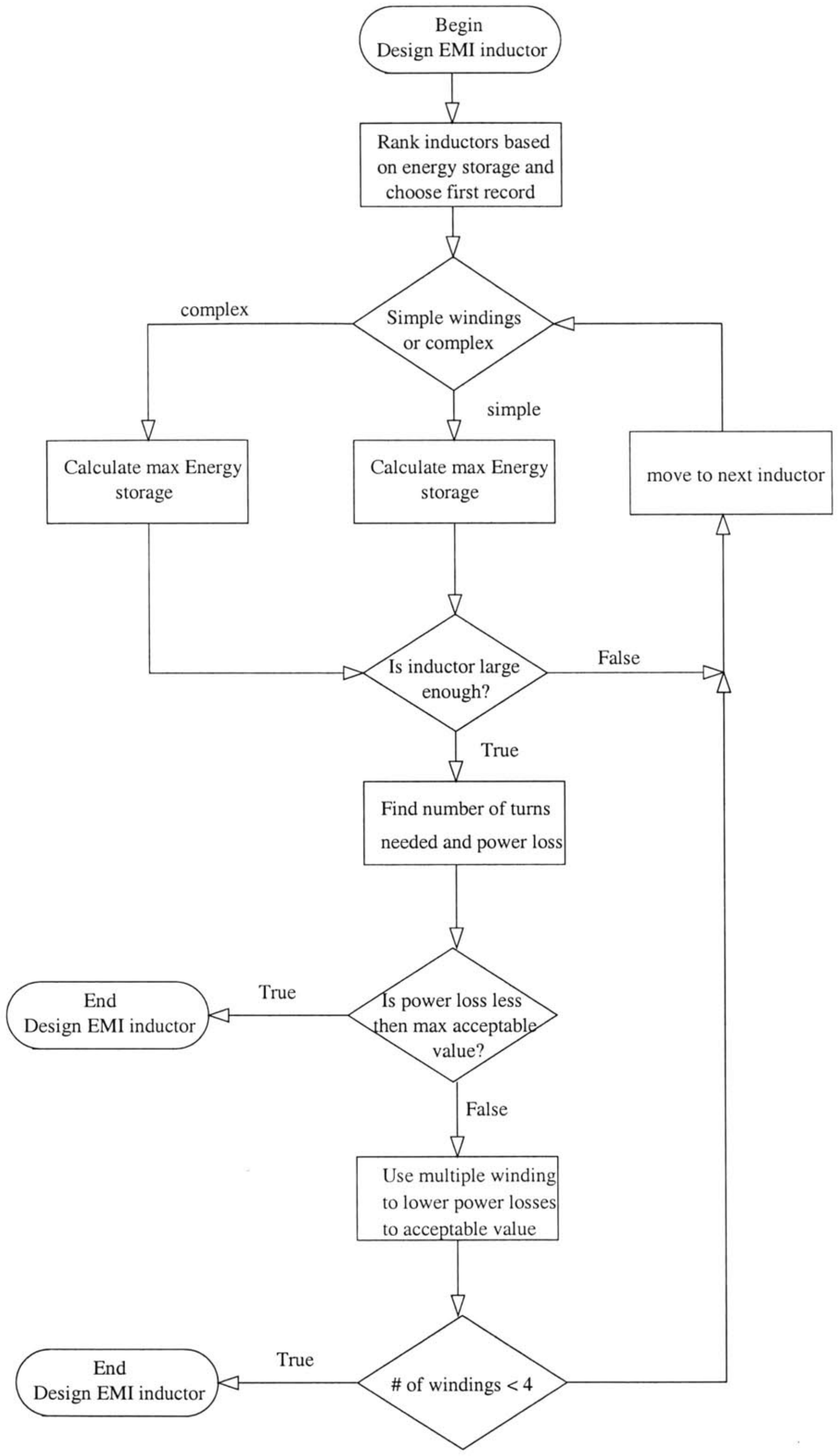

Begin
Design EMI inductor
Rank inductors based on energy storage and choose first record
Simple windings or complex
complex
simple
Calculate max Energy storage
Calculate max Energy storage
move to next inductor
Is inductor large enough?
False
True
Find number of turns needed and power loss
Is power loss less then max acceptable value?
True
End
Design EMI inductor
False
Use multiple winding to lower power losses to acceptable value
# of windings < 4
True
End
Design EMI inductor

Figure E.9 Flowchart for the design of the EMI filter inductor.

3400 - 93